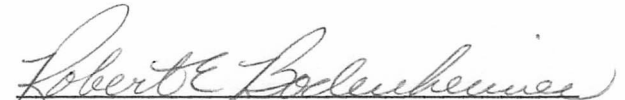


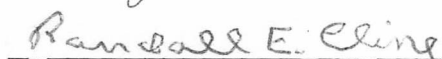
November 25, 1970

To the Graduate Council:

I am submitting herewith a thesis written by Alfredo Caldwell entitled "Computer Implemented Synthesis of Multiple Level Combinational Networks." I recommend that it be accepted for nine quarter hours of credit in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:


Vice Chancellor for
Graduate Studies and Research

COMPUTER IMPLEMENTED SYNTHESIS OF MULTIPLE
LEVEL COMBINATIONAL NETWORKS

A Thesis
Presented to
the Graduate Council of
The University of Tennessee

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Alfredo Caldwell
December 1970

ACKNOWLEDGMENTS

The author takes this opportunity to express his sincere gratitude to Dr. Robert E. Bodenheimer for his suggestion of the topic and for his advice and encouragement during the preparation of this thesis. Also, the author wishes to express his appreciation to Dr. Harvey P. Carter and Dr. Randall E. Cline for their services on the Master's degree committee. In addition, the author wishes to thank David M. Marshall of The University of Tennessee Computing Center for his suggestions in writing the computer programs, and Mrs. Evelyn Brann of The University of Tennessee Public Relations Office who typed the final copy of the thesis. Finally, the author is deeply indebted to his parents, Mr. and Mrs. Ivan E. L. Caldwell, and to his wife, Judith Anne, without whose encouragement and moral support this study would not have been possible or worthwhile.

ABSTRACT

The purpose of this investigation has been to implement, in the form of computer programs, two algorithms which are used in the synthesis of multiple level combinational networks. The algorithms implemented were devised by Professor Paul E. Wood, Jr., of M.I.T., and by Professor Eugene L. Lawler, of The University of Michigan. In the course of the investigation a more efficient way of implementing the two algorithms was discovered. The combined version of the algorithms takes advantage of the best features of the original algorithms.

In the synthesis of multiple level combinational networks minimal complexity is highly desirable. The whole point of minimizing a network is that this leads to lower manufacturing cost, greater ease of construction, and greater reliability for the electronic circuit being designed.

The computer programs which were developed to implement the algorithms can minimize Boolean expressions with as many as seven variables. Theoretically, there is no limit to the number of variables per expression that can be minimized by these algorithms. But when an expression has more than seven variables, the processing time increases dramatically, and for this reason this number was set as a limit.

The final solutions obtained from the computer programs are minimal two and three level multiple-input single-output Boolean expressions which realize a given set of conditions.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
II. THEORETICAL BACKGROUND FOR WOOD'S ALGORITHM	5
III. WOOD'S ALGORITHM AND PROGRAM DESCRIPTION	21
IV. LAWLER'S MINIMIZATION ALGORITHM	32
V. CONCLUSION	40
BIBLIOGRAPHY	50
APPENDIXES	52
A. BOOLEAN THEOREMS	53
B. EXAMPLE USING WOOD'S ALGORITHM	54
C. EXAMPLE USING LAWLER'S ALGORITHM	58
D. COMPUTER LISTING OF WOOD'S ALGORITHM PROGRAM	63
E. COMPUTER LISTING OF LAWLER'S ALGORITHM PROGRAM	85
F. COMPUTER LISTING OF THE COMBINED ALGORITHM PROGRAM	94
VITA	105

LIST OF FIGURES

FIGURE		PAGE
1.	Schematic Diagrams of Electronic Gate-Elements . . .	11
2.	Examples of Various Network Operator Structures	13
3.	Flowcharts of the Minimization Algorithms	41
4.	Karnaugh Map for $f = \sum(0,3,5,6,9,10,12,15)$	42
5.	Karnaugh Map for $f = \sum(0,1,2,3,4,5,6,7)$	42
6.	Number of Minterms, Number of Variables, Time Graph for Wood's Algorithm	43
7.	Number of Minterms, Number of Variables, Time Graph for Lawler's Algorithm	45
8.	Number of Minterms, Number of Variables, Time Graph for the Combined Algorithm	46

CHAPTER I

INTRODUCTION

Switching networks are used in the construction of computers and many other electronic devices. In the design of these networks minimal complexity is highly desirable. The ideal minimum complexity network which has a specified behavior has the least possible number of one or more of the following: switch elements, gate inputs, gate elements, gate fan-in, gate fan-out, and levels of switching.

The minimization of a given network can be achieved by trial and error, or by systematically reducing the Boolean expression which represents the network. The whole point of minimizing a network is that this leads to lower cost, greater ease of construction, and greater reliability for the machine being designed.

The objective of this thesis is twofold: first, to present the theoretical background leading up to three algorithms used to minimize Boolean expressions; second, to describe the computer programs which implement these algorithms.

In 1854, George Boole, an English mathematician, published a work on systematic logic (1). Since then this system of logic has become known as Boolean algebra. The first person to use Boolean algebra to analyze switching

networks was C. E. Shannon in 1938 (2). Thereafter Boolean algebra has been used extensively in the analysis and synthesis of switching networks, because the structure of these networks can often be described in terms of Boolean expressions.

As will be shown later, given a certain desired behavior for a network, there are an infinite number of equivalent network structures (Boolean expressions) which will produce this desired behavior. There are trial and error methods for finding minimal expressions, but if the number of terms and/or the number of variables in the expression are large, then these methods are time consuming and unsystematic. This is where the value of systematic algorithms lies. If they can be programmed, then the computer can do all the tedious work in a matter of seconds.

In Chapter II the theoretical background of a minimization algorithm devised by Paul E. Wood, Jr. of the Lincoln Laboratory of M.I.T. (3) is presented. A number of theorems associated with this algorithm will be stated without proof, although the necessary references will be given. Chapter III consists of an outline of Wood's algorithm and a description of the program which implements the algorithm. In Chapter IV, some additional theoretical background for the second minimization algorithm, which was devised by Eugene L. Lawler of the University of Michigan (4), is

presented. This also includes some additional theorems (without proof) and their corresponding references. The last part of Chapter IV is a description of the program which implements Lawler's algorithm. Chapter V includes: (a) a flow chart of an algorithm which combines the best features of Wood's and Lawler's algorithms; (b) a comparison of the three algorithms; (c) some studies on the relationship between the number of variables of a Boolean expression, the number of minterms of a Boolean expression, and the amount of time required by the three computer programs to minimize various functions; (d) a summary of the thesis and some conclusions. Finally, in the Appendices the following are presented: Appendix A - a list of Boolean theorems used in the minimization of Boolean expressions; Appendix B - an example of a minimization using Wood's algorithm; Appendix C - an example of a minimization using Lawler's algorithm; Appendixes D, E, F, - listings of the programs which implement Wood's algorithm, Lawler's algorithm, and the combined algorithm, with some examples of minimized functions.

Two previous theses have dealt with the general area of minimization of Boolean expressions. The first (5) presents a number of different methods for finding the prime implicants of a Boolean expression. The second (6) deals with the minimization of multiple-output-two-level Boolean

functions. This thesis is concerned with algorithms for determining minimal two-and-three-level-single-output Boolean functions.

CHAPTER II

THEORETICAL BACKGROUND FOR WOOD'S ALGORITHM

Most of the following material is presented in greater detail in the book Switching Theory by Paul E. Wood, Jr.

(3). The following summary is presented here so as to provide the reader with a frame of reference for the pursuing work.

A switching network is a group of logical operators or their electronic counterparts (AND, OR, NOT gates, for example) interconnected in such a way as to perform some useful function. Switching networks can be broken up into two classes: combinational networks and sequential networks. A combinational network is one in which the present output is uniquely determined by the present input. In a sequential network the present output is a function of the previous input(s). In this thesis only combinational networks will be considered.

A switching network can be considered from two points of view, that is, in terms of its structure and/or in terms of its behavior. The structure of a switching network is the interconnection pattern of its components. The behavior of a switching network is the cause-effect relationship between the input and the output. From this point of view, the structure of a network is considered as a "black box"

which when given a certain input, produces a corresponding output. This thesis will be concerned with the minimization of the structure of switching networks and not with their behavior.

The structure of the networks that will be considered will be specified in terms of Boolean functions and expressions. The combinational networks to be studied will be two-valued, that is, their components can take on only one of two values (0-1, true-false, on-off, etc.) at any given time. It is for this reason that Boolean algebra is useful in switching theory. It serves as a mathematical language for the description of two-valued combinational networks.

The networks considered in this thesis will consist of input terminals, switches or gates, interconnecting lines, and output terminals. For each of the input and output terminals there will be assigned a two-valued variable. The input variables will be denoted by the symbols ...,x,y,z, and the output variable by ...,f,g,h.

The input-output relationship of a network can be represented by a table of combinations (see Table I) or by a Boolean function.

A Boolean function is an equation which states that the output(s) of a switching network depends on the inputs and on the internal structure of the network. This type of function has the following general form: $f(\dots,x,y,z) =$

TABLE I

TABLE OF COMBINATIONS FOR $F(X,Y,Z) = X'Y'Z' + X'Y'Z + X'YZ' + XYZ$

Decimal	Input xyz	Output $f(x,y,z)$	Minterm	Maxterm
0	000	1	$x'y'z'$	$x+y+z$
1	001	1	$x'y'z$	$x+y+z'$
2	010	1	$x'yz'$	$x+y'+z$
3	011	0	$x'yz$	$x+y'+z'$
4	100	0	$xy'z'$	$x'+y+z$
5	101	0	$xy'z$	$x'+y+z'$
6	110	0	xyz'	$x'+y'+z$
7	111	1	xyz	$x'+y'+z'$

"expression," where "expression" is a combination of any of the following:

- (a) a constant (0 or 1)
- (b) a variable or its complement ($\dots, x, y, z$ or $\dots, x', y', z'$)
- (c) if A and B are expressions, then $A + B$ (A or B), AB (A and B), A' (not A), and B' (not B) are expressions, where A and B are generic variables that represent constants, variables, or other expressions.

In Appendix A there is a list of postulates and

theorems on Boolean algebra which are used to describe and to simplify Boolean expressions. By applying these theorems any expression can be manipulated into an infinite number of equivalent but distinct expressions. The following is an example of some of these transformations with references to the Boolean theorems (see Appendix A) used to obtain the equivalent expressions.

$$\begin{aligned}
 f(x,y,z) &= x'y'z' + x'y'z + x'yz' + xyz \\
 &= x'y'(z' + z) + x'yz' + xyz \quad (\text{Distributive Law}) \\
 &= x'y' + x'yz' + xyz \quad (\text{Idempotent Law}) \\
 &= x'(y' + yz') + xyz \quad (\text{Distributive Law}) \\
 &= x'(y' + y)(y' + z') + xyz \quad (\text{Distributive Law}) \\
 &= x'(y' + z') + xyz \quad (\text{Idempotent Law})
 \end{aligned}$$

The fundamental theorems of the Boolean algebra of expressions are the following:

Theorem 1. (The canonic-sum-form of an expression)

(3, p. 21).

$$f(x_1, \dots, x_n) = \sum (x_1 * x_2 \dots x_n f(a_1, a_2, \dots, a_n))$$

where $x_i^* = x_i$ if $a_i = 1$

or $x_i^* = x_i'$ if $a_i = 0$

and $\sum$ denotes the Boolean sum over all

$(a_1, a_2, \dots, a_n)$ with $a_i = 1$ or 0 , $i = 1, 2, \dots, n$

Theorem 2. (The canonic-product-form of an expression)

(3, p. 21).

$$f(x_1, \dots, x_n) = \prod (x_1^* + x_2^* + \dots + x_n^* + f(a_1, a_2, \dots, a_n))$$

$$\text{where } x_i^* = x_i \text{ if } a_i = 0$$

$$\text{or } x_i^* = x_i' \text{ if } a_i = 1$$

and $\prod$ denotes the Boolean product over all $(a_1, a_2, \dots, a_n)$ with $a_i = 1$ or 0 , $i = 1, 2, \dots, n$

The product term $x_1^* x_2^* \dots x_n^*$ is called a minterm, and the sum term $x_1 + x_2^* + \dots + x_n^*$ is called a maxterm. Some examples of these are given in Table I, p. 7. Any expression can be manipulated into either of two equivalent forms (a sum of minterms or a product of maxterms) by applying the Boolean theorems as in the example above (p. 8).

The sum-of-minterms expression is called a canonic sum, and the product-of-maxterms expression is called a canonic product. For the function shown in Table I, the canonic sum is:

$$\begin{aligned} f(x, y, z) &= x'y'z + x'yz' + x'yz + xyz \\ &= 000 + 001 + 010 + 111 \\ &= \sum(0, 1, 2, 7) \end{aligned}$$

For the same function the canonic product is:

$$\begin{aligned} f(x, y, z) &= (x+y'+z') (x'+y+z) (x'+y+z') (x'+y'+z) \\ &= (011) (100) (101) (110) \\ &= \prod(3, 4, 5, 6) \end{aligned}$$

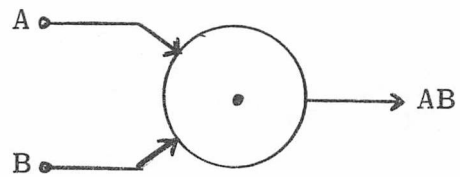
The above examples also illustrate a more compact notation for canonic sums and canonic products which uses the decimal numbers corresponding to the value of

$(a_1, a_2, \dots, a_n)$ considered as a binary number. The decimal number is included in the notation if and only if (iff) its corresponding minterm or maxterm appears in the canonic sum or product.

Combinational networks can be realized and described in terms of switch-elements or in terms of gate-elements. In the description of networks, gate-elements will be used since the relationship between a Boolean expression and its corresponding network diagram is easy to visualize, and especially because the number of levels of an expression is easy to "see" (given its gate-element representation).

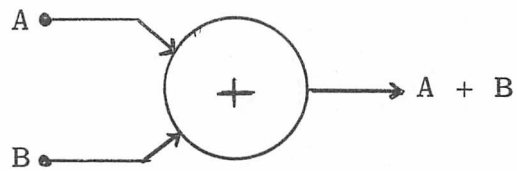
In the minimization of networks, the number of switch elements will be the minimal complexity criterion. The reason for this is that there is a one-to-one correspondence between the number of switches in a logical circuit and the number of literals in the corresponding Boolean expression. The number of gates, or the number of inputs per gate, or the gate-element fan-in could have been used as minimization criteria, but, for the reason mentioned above, the number of switch elements was chosen as the minimization criterion.

Gate-elements perform electronically the logical operations associated with their respective names. The following gate-elements (see Figure 1) will be used in the description of the combinational networks considered in this thesis.



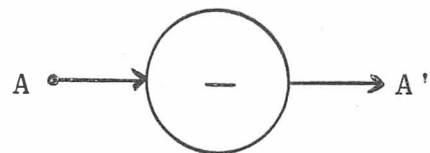
AND GATE

$$AB = 1 \text{ iff } A = 1 \text{ and } B = 1$$



OR GATE

$$A + B = 1 \text{ if } A = 1 \text{ or } B = 1; \text{ or if } A = 1 \text{ and } B = 1$$



NOT GATE

$$\text{if } A = 1, A' = 0; \text{ if } A = 0, A' = 1$$

Figure 1. Schematic diagrams of electronic gate-elements.

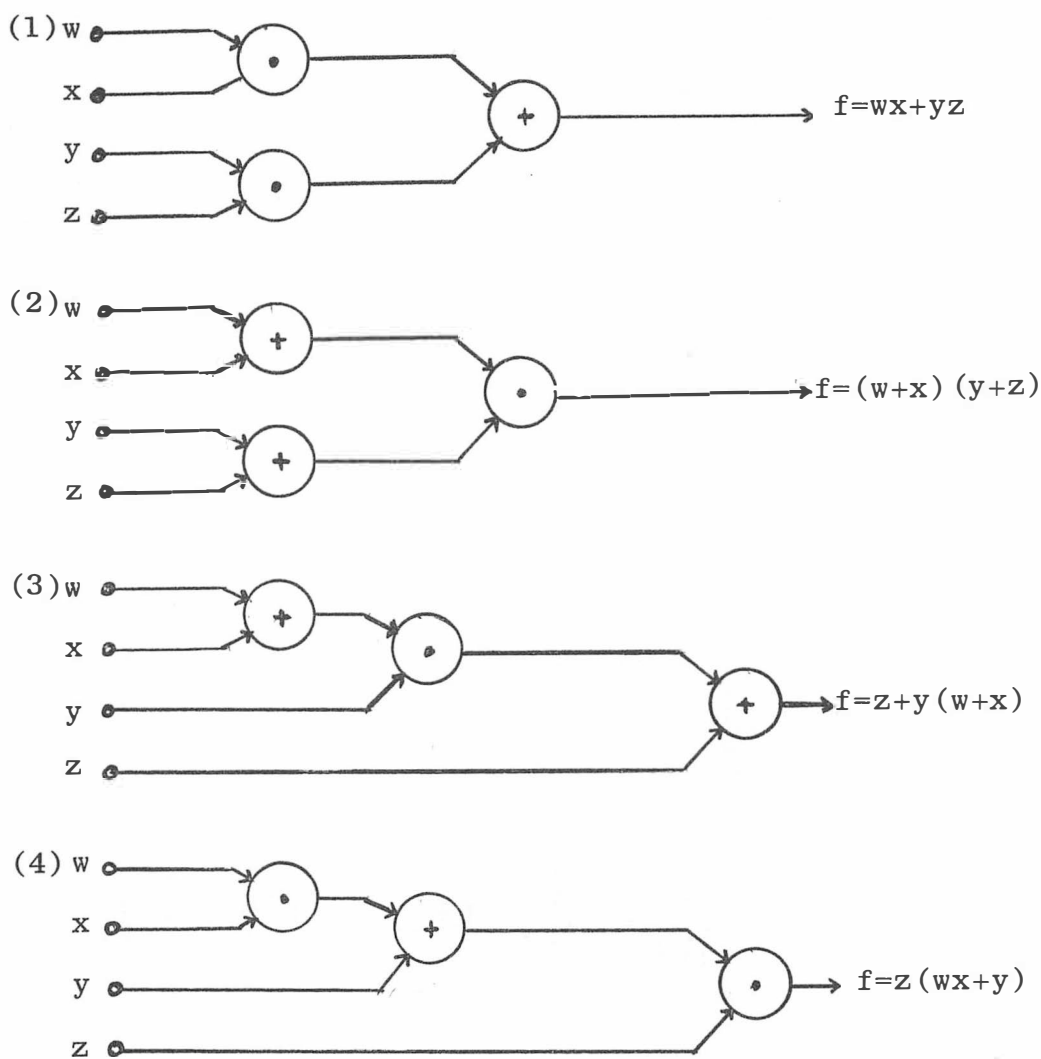
The levels of a network correspond to the number of logical operators (AND, OR, NOT) or their electronic counterparts through which an input signal passes from the time it enters the network to the time the output is produced. The maximum number of levels of a network is called the level of the network. Complemented variables are assumed to be available and thus the NOT gates which would normally complement the input variables are not counted as separate levels.

In this thesis only two and three level networks will be considered. A list of these networks, their operator structure, their abbreviations, and some examples are shown in Figure 2.

The number of gate inputs which are connected to a gate output is called the gate-element fan-out. Networks considered in this thesis will have a fan-out of 1. The number of inputs per gate is called the gate-element fan-in. Networks considered in this thesis will have unlimited fan-in.

A gate-network path is any path from gate to gate via interconnections in the arrow direction. If the same gate is encountered twice in a path, then the path is closed and forms a loop. A gate network is a feedforward network iff it has no loops.

Theorem 3. An expression can be realized by a



Network	Operator Structure	Abbrev.
(1) sum of products	AND/OR	sp
(2) product of sums	OR/AND	ps
(3) sum of products of sums	OR/AND/OR	sps
(4) product of sums of products	AND/OR/AND	psp

Figure 2. Examples of various network operator structures.

feed-forward gate network, constructed of AND, OR, and NOT gates with unity fan-out and such that gates of the same type are never connected together, whose structure uniquely specifies the expression (3, p. 61).

The minimization algorithms to be presented in this thesis make extensive use of the concept of prime implicant. An expression A implies the expression B ($A \rightarrow B$) iff $B = 1$ for every valuation of variables for which $A = 1$. If $A \rightarrow B$, then B is said to cover A .

All of the following material will be presented in terms of product terms (sp expressions). A similar development could be presented in terms of sum terms (ps expressions).

The product term A subsumes the product term B iff $A \rightarrow B$. Considering a literal as any variable symbol or its complement, then a product term A subsumes a product term B iff every literal of B is a literal of A .

A product term A is a prime implicant of the function f iff (a) $A \rightarrow f$, and if (b) there does not exist an expression B such that $A \rightarrow B \rightarrow f$, where the number of literals of B is less than the number of literals of A .

Theorem 4. A prime implicant A of f has the following properties:

- (a) A contains no repeated literals.
- (b) A never contains both a variable and its complement.

- (c) If any literal is deleted from A, the remainder is not a prime implicant of f; that is, A never contains a redundant variable (3, p. 75).

One of the more important steps in Wood's algorithm is the generation of the prime implicants of the function being minimized. This is accomplished by a method called iterated consensus of minterms.

The operation of consensus is based on the identity $xy + x'z = xy + x'z + yz$. In practice, the consensus of two product terms xA and $x'B$ is AB , where A and B are product terms which contain neither x nor x'. All duplicate literals are assumed to be deleted from terms xA , $x'B$, and AB . No consensus term is generated if there is no opposing pair of variables, and $AB = 0$ if there are two or more opposing pairs of variables. For example,

First term	Second term	Consensus term
vyz	$v'xyz$	xyz
xyz	x'	yz
xyz'	$y'z$	0
xyz'	vz'	None

The consensus operation is used in the following theorem to generate the prime implicants of a function.

Theorem 5. Any sp expression for function f can be converted to a sum of exactly all prime implicants of f by the application of the following steps to the canonic-sum expression of n variables.

Step 0. Generate the minterms of f from the canonic-sum expression of f .

Step i ($i = 1, 2, \dots, n$). All terms are added to the expression which can be generated by consensus of pairs of terms which were themselves generated at step $i-1$. After all such terms are generated, all terms of step $i-1$ used in a consensus operation are deleted from the expression. When no more consensus terms can be generated, the remaining terms are the prime implicants of f (3, p. 81).

The sum of these prime implicants may or may not be the minimal solution, that is, some of these terms may be redundant. An irredundant sp expression is the desired result. Such an expression is a sum of prime implicants of f such that no prime implicant can be deleted and the expression still represent f .

The irredundant sp expression(s) for a function f can be derived from a prime implicant table based on the following theorem.

Theorem 6. Each minterm of f must subsume at least one term of an sp expression which represents f (3, p. 84).

In other words, the terms of an irredundant sp expression must cover all the minterms of f .

For any function f , the subsuming relationships between the minterms and the prime implicants of f can be displayed in a prime implicant table which is constructed as follows:

- (a) The minterms of f are listed along the horizontal axis and the prime implicants of f along the vertical axis.
- (b) An x is placed at the intersection of a minterm and prime implicant if the minterm subsumes the prime implicant.
- (c) A capital letter is associated with each prime implicant of the table and this letter denotes the statement that the corresponding prime implicant is used as a term in a sum-of-prime-implicants expression.
- (d) Then the covering expression for f is formed as follows:
 - (i) For every x that appears in a column write the corresponding letter is "OR'ed" with the letters corresponding to any other x 's which may appear in the same column.
 - (ii) Form the product of the sums of the columns.
 - (iii) Apply Boolean theorem G (Appendix A) to convert the covering expression to sp form.
 - (iv) Reduce the covering expression as much as possible by applying Boolean theorems H and L so that no term has repeated literals and no term subsumes another.
- (e) Each term of the reduced expression specifies an

irredundant sp expression for f where a prime implicant is included as a term of the irredundant expression iff its corresponding letter is included in the terms of the covering expression.

The above process may yield more than one irredundant expression for a given function. If so, then the one(s) with the least number of literals is the one(s) with minimum complexity.

For example, to find the minimal sp expression for the function $f = \sum(0,1,2,7) = x'y'z' + x'y'z + x'yz' + xyz$ the process would be as follows:

- (a) Find the prime implicants of f by iterated consensus of the minterms (Theorem 5, p. 15).

step	pairs of terms used in the consensus operation	consensus terms	delete terms
1.	$x'y'z' - x'y'z$	$x'y'$	$x'y'z'$ $x'y'z$
	$x'y'z' - z'yz'$	$x'z'$	$z'yz'$
	$x'y'z - xyz$	0	
	$x'y'z - x'yz'$	0	
	$x'y'z - xyz$	0	
	$x'yz' - xyz$	0	
2.	$x'y' - x'z'$	none	
	$x'y' - xyz$	0	
	$x'z' - xyz$	0	

Since no new consensus terms were found in step 2, then at this stage $f =$ the sum of its prime implicants

$$= x'y' + x'z' + xyz$$

- (b) Find the irredundant expression(s) of f by using a prime implicant table to form a covering expression (Theorem 6, p. 16).

	$x'y'z'$	$x'y'z$	$z'yz'$	xyz
A $x'y'$	*	*		
B $x'z'$	*		*	
C xyz				*

Covering expression $= (A+B)ABC = ABC + ABC = ABC$

Therefore $f = A + B + C = x'y' + x'z' + xyz$

The techniques presented thus far will yield the minimal sp (or ps) irredundant expressions of a given function. The ultimate objective of this thesis also includes techniques for finding the minimal sps (or psp) irredundant expressions of a given function.

An sps (psp) expression A is a prime sps (psp) implicant of f iff A implies f and there does not exist an sps (psp) expression B such that A implies B , and the number of literals in B is less than the number of literals in A .

The generation of generalized prime implicants in a recursive procedure; that is, prime ps implicants of f are products of sum prime implicants of f , prime sps implicants of f are sums of prime ps implicants of f , and so forth.

This process can be carried to any number of levels. In the algorithms presented here the number of levels of a network will be restricted to at most three levels. The computer programs included in this thesis are designed to implement the following theorem.

Theorem 7. The structure of every minimum-switch-element 3-level switch network which realizes function f is represented by a sum of prime p_s implicants of f (i.e., an sps expression) or by a product of prime sp implicants of f (i.e., a psp expression) (3, p. 100).

CHAPTER III

WOOD'S ALGORITHM AND PROGRAM DESCRIPTION

Based on the theorems presented in Chapter II, Wood's minimization algorithm is as follows:

- (a) Assume that the function to be minimized, f , has been given as a sum of minterms (product of maxterms).
- (b) Systematically consider all functions g which imply f ($2^m - 1$ functions if f has m minterms (maxterms)) in order of increasing number of minterms (maxterms) of f .
- (c) For each of the g functions generate its prime ps (sp) implicants by using iterated consensus of the minterms (maxterms).
- (d) Construct a prime- ps (sp) implicant table for f .
- (e) A covering expression is then used to generate all sums (products) of prime- ps (sp) implicants which cover the minterms (maxterms) of f .
- (f) The sum (product) of prime- ps (sp) implicants with the least number of literals is the minimum sps (psp) expression. Of course, there may be more than one minimum sps (psp) expression.

This algorithm yields not only the minimum sps (psp) expression(s) of a function, but also its minimum ps (sp)

expression. The latter is the expression generated when the minterms (maxterms) of a g function are the same as those of f ; that is, as the g functions are generated in order of increasing number of minterms (maxterms), the last g function to be generated is the one whose minterms (maxterms) are the same as those of the original function f . This last g function is the irredundant ps (sp) function equivalent to the original function f (see Appendix B).

The following is a description of the program which implements Wood's algorithm (see Appendix D).

(1) The program inputs the following variables: N - the number of minterms in the function to be minimized; $LENGTH$ - the number of literals in a minterm (maxterm); $MINDEC$ - the decimal equivalents of the minterms of the function to be minimized.

(2) After reading in the data the program generates all the minterms (maxterms) in a numerical form described in a paper by H. Mott and C. C. Carroll (7). In this notation, a "2" indicates the presence of a variable in a term, a "1" indicates the presence of its complement, and a "0" indicates the absence of a variable.

Subroutine MAXX generates the minterms (maxterms) as follows:

(a) The number of terms = 2^m , where m = the number of variables in a term.

- (b) Each of the integers from 1 to 2^m is converted to its binary form by repeated division by 2.
- (c) Wherever a "0" occurs in the binary form of a term, a "1" is substituted, and wherever a "1" occurs in the binary form of a term, a "2" is substituted.

For example, when $m = 3$, Table II illustrates the result of applying subroutine MAXX.

TABLE II
GENERATION OF MINTERMS (MAXTERMS) BY SUBROUTINE MAXX

Minterm	Binary Form	Numerical Form
$x'y'z'$	000	111
$x'y'z$	001	112
$x'yz'$	010	121
$x'yz$	011	122
$xy'z'$	100	211
$xy'z$	101	212
xyz'	110	221
xyz	111	222

(3) Subroutine ONE performs step (b) of Wood's algorithm as follows:

- (a) First generate the combinations of subscripts that will be used to select the minterms (maxterms)

of the g functions of f . There are $2^n - 1$ such combinations (n = number of minterms (maxterms) in the original function f). These combinations are generated by the subroutine MDD which uses modular arithmetic to convert the numbers from 1 to $2^n - 1$ into binary form, and finally into subscripts. Table III illustrates the result of applying subroutine MDD for $n = 3$.

TABLE III
GENERATION OF COMBINATIONS OF SUBSCRIPTS BY SUBROUTINE MDD

Integer	Binary Form	Subscripts
1	001	1
2	010	2
3	011	1, 2
4	100	3
5	101	1, 3
6	110	2, 3
7	111	1, 2, 3

(b) The selection of minterms (maxterms) of a g function is accomplished by eliminating those which will not be used in the consensus operation. For example, if the g function includes the minterms

(0,2,4), then the maxterms (1,3,5,6,7) will be used in the consensus operation to find the irredundant ps expression(s) for g. These terms are stored in the array MIX.

(4) The subroutine CONSNS uses the method described in a book by E. J. McCluskey (8) and the numerical notation by Mott and Carroll (7) to generate the prime implicants of the given minterms (maxterms). The steps of the procedure are:

- (a) Arrange the maxterms in groups according to the number of uncomplemented variables in each of the terms.
- (b) Find the difference between two maxterms in adjacent groups. If this difference = 10^n ($n = 0, 1, 2, \dots$), then there exists a consensus term equal to the smaller of the two terms minus 10^n .
- (c) Generate all possible consensus terms between adjacent groups, and store them in the group with the lower number of uncomplemented variables.
- (d) Tag the terms which are used to form a consensus term; tag any duplicate consensus terms and eliminate them from the consensus list; eliminate those terms used in the consensus operation.
- (e) Generate all possible consensus terms between adjacent groups.

(f) Repeat this process using the new consensus terms until no new consensus terms can be formed.

(g) The remaining terms and those which have not entered into consensus anywhere in the process are the prime implicants of g .

For example, if the maxterms of g are $(0,2,4)$, the prime implicants of g are found by CONSNS as follows:

group 1	0	111
group 2	2	121
	4	211

Find differences between pairs of maxterms in adjacent groups.

4 211

0 -111

$$100 = 10^2,$$

$$\text{therefore the consensus term} = 111 - 100 =$$

$$011 = y + z$$

2 121

0 -111

$$10 = 10^1,$$

$$\text{therefore the consensus term} = 111 - 10 =$$

$$101 = x + z$$

All of the original terms were used to form consensus terms. Therefore, they are eliminated from the consensus list. The consensus terms formed $(011, 101)$ have the same number of

uncomplemented variables. Hence, no new consensus terms can be formed from the, and the prime implicants of g are 011, 101.

(5) After CONSNS finds the prime implicants, the sub-routine TABLE constructs a prime implicant table which will be used to determine the irredundant ps (sp) expression for g . The maxterms of the g function being processed are listed on the horizontal axis, and the prime implicants are listed on the vertical axis. If maxterm_i (minterm_i), includes implicant_j , then $\text{JTAB}(I,J) = 1$, otherwise, $\text{JTAB}(I,J) = 0$.

In the program, whether or not a maxterm (minterm) includes a prime implicant is determined in the following way:

(a) If the sum of a maxterm (minterm) and an implicant is made up of "1's" and/or "2's" and/or "4's", then the maxterm (minterm) includes the prime implicant.

(b) If the sum contains any "3's", then the maxterm (minterm) does not include the prime implicant.

For example:

Implicants	Maxterms			
	$x+y+z$	$x+y+z'$	$x+y'+z$	$x+y+z$
$x+y$	*	*		
$x+z$	*		*	
$x+y+z$				*

		Maxterms			
Implicants		111	112	121	222
110		<u>221</u>	<u>222</u>	231	332
101		<u>212</u>	213	<u>222</u>	323
222		333	334	343	<u>444</u>

		Maxterms			
Implicants		111	112	121	222
110		1	1	0	0
101		1	0	1	0
222		0	0	0	1

Next the subroutine selects the essential prime implicants (if there are any). A prime implicant is essential if there is at least one maxterm (minterm) which includes this prime implicant and no other. In terms of the prime implicant table, an essential prime implicant is one whose corresponding row contains a "1" in a column with no other "1's".

The reduced prime implicant table is then formed by eliminating all those rows of the original table which correspond to maxterms (minterms) implied by the essential implicants. In this reduced table, if a row consists only of "0's", then the corresponding implicant is absolutely

eliminable; that is, it will never be part of the irredundant expression(s) for g .

The final segment of the subroutine TABLE selects the combinations of conditionally eliminable implicants that, together with the essential implicants, form an irredundant expression for g . All possible combinations are generated by the subroutine MDD (see p. 24). These combinations are then "OR'ed" together and if the result of this "OR'ing" covers all the maxterms (minterms) of the reduced table (all "1's"), then the conditionally eliminable implicants used in the operation form part of the irredundant expression for g .

(6) After all the g functions of the original function have been generated the subroutine SPORPS selects the minimal ps (sp) solution of function f . This solution corresponds to the last g function generated (the (2^n-1) th one), since this one is the only g function which covers all the minterms (maxterms) of the original function.

Subroutine SPORPS also calculates the maximum gate-fan-in, the number of gate elements, the number of gate inputs, and the number of switches of the minimal ps (sp) solution.

(7) The subroutine SPSPSP uses the subroutine ORR to find the minimal sps (psp) solution(s) of the original function as follows:

- (a) All possible combinations of 2 numbers from 1 to 2^n-1 are generated by the subroutine MDD (see p. 24).

- (b) Each of these pairs of numbers are "OR'ed" by the subroutine ORR.
 - (c) If the result of this "OR'ing" is all "1's", then the two g functions corresponding to the numbers "OR'ed" together form a solution.
 - (d) The solution(s) with the least number of literals is the desired one.
 - (e) The above process is repeated for combinations of 3, 4, ... numbers until the minimal solution found during one pass is greater than the minimal solution found during the previous pass. When this occurs, the process stops, and the minimal solution found during the previous pass is the minimal solution of f.
- (8) The subroutine CONVRT changes each term of the solution from numerical form to the equivalent literal form as follows:

Position of a digit from left to right

[illegible]

Parentheses are placed at each end of a term, and there is a plus sign between successive literals.

(9) The subroutine CONMIN converts a product term from numerical form to its equivalent literal form in a manner similar to CONVRT, except that the prime signs are reversed, there are no parentheses at each end of a term, and there is no plus sign between successive literals.

(10) The last group of instructions selects the maxterms of the original function which will be used in generating the minimal sp and psp expressions. Then the program goes through the whole cycle again. The reason for re-cycling is that the minimal ps and sps solutions are not necessarily absolutely minimal. The sp and psp solutions may be the absolutely minimal solutions.

CHAPTER IV

LAWLER'S MINIMIZATION ALGORITHM

The theoretical background necessary to understand Lawler's algorithm is similar to that for Wood's algorithm, but in addition the following is required.

Lemma 1. If g is a minimal sp (ps) expression of f (the given function), and another expression implies g , and g implies f ; that is, $h \rightarrow g \rightarrow f$, then g is a minimal sp (ps) expression for each and every other expression h^* , where $h \rightarrow h^* \rightarrow g \rightarrow f$ (4, p. 286).

This lemma implies that when generating minimal irredundant expressions of a function, it is not necessary to use all of the subfunctions in the prime implicant table because some of these will be covered by others with fewer literals.

Lemma 2. Let $G = (g_1, g_2, \dots, g_k)$ be the set of all minimal sp (ps) expressions for h (a subfunction of f , the original function). Then G contains all minimal sp (ps) expressions for every other expression h^* , where $h \rightarrow h^* \rightarrow g_i$ for any g_i which is a member of G (4, p. 286).

This lemma implies that it is not necessary to use all the minimal ps (sp) solutions generated from the $2^m - 1$ (m = the number of terms in the original function) subfunctions

of f (the original function) in the final prime implicant table used to form the minimal sps (psp) expression(s) of f . The reason for this is that Lawler's algorithm generates subfunctions of f in groups which are subsets of succeeding groups (see Appendix B for detailed example). After each group of subfunctions is generated, the minimal expression which covers all the subfunctions generated up to that point is found, and the minterms (maxterms) covered by the intermediate solution are tagged. At the end of this process (when all the minterms (maxterms) have been tagged), the intermediate solutions are the ones used to find the minimal sps (psp) solution of the original function (instead of all of the subfunctions).

Any differences between Lawler's algorithm and Wood's algorithm will be pointed out in the following outline of Lawler's algorithm.

(1) Assume that the function to be minimized has been given as a sum of minterms (product of maxterms).

(2) Systematically consider all functions which imply f and are implied by the first minterm; generate the prime-ps (sp) implicants of each function; select the one with the least number of literals; tag the minterms (maxterms) covered by this minimal expression. (At this stage in Wood's algorithm, all of the g functions which imply f would be considered, not just those implied by the first minterm (maxterm).)

(3) Select the next untagged minterm (maxterm); systematically consider all functions which imply f and are implied by all subsets of minterms (maxterms) selected thus far; find an irredundant ps (sp) expression for each of the functions.

(4) After this group of g functions has been generated, find the minimal ps (sp) expression which covers the irredundant ps (sp) expressions generated so far. This intermediate solution will be used in the final prime implicant table. (In Wood's algorithm all of the g functions are generated before a covering expression is found; that is, no intermediate solutions are found.)

(5) After each of the above intermediate solutions is found, tag all the terms covered by the intermediate solution. (In Wood's algorithm no tagging of the terms takes place because all of the g functions are used to find the minimal sps (psp) solution.)

(6) Repeat steps c, d, e, until all the terms have been tagged. (In Wood's algorithm the equivalent steps would be repeated until all the g functions have been generated.)

(7) Construct a prime-ps (sp) implicant table for f using the intermediate solutions and those g functions not covered by the intermediate solutions. (In Wood's algorithm the prime-ps (sp) implicant table is constructed using all of the g functions.)

(8) A final covering expression is then used to

generate all sums (products) of prime-ps (sp) implicants which cover the minterms (maxterms) of f .

(9) The sum (product) of prime-ps (sp) implicants with the least number of literals is the minimum sps (psp) expression for f .

As with Wood's algorithm, Lawler's algorithm yields not only the minimum sps (psp) expression(s) of a function, but also its minimum ps (sp) expression. The latter is the expression generated when the minterms (maxterms) of a g function are the same as those of f .

Since much of the program which implements Lawler's algorithm is the same as the program which implements Wood's algorithm (see Appendix D), a separate chapter is not necessary to describe the program which implements Lawler's algorithm. Similarities and differences will be pointed out in the description which follows.

(1) The input is the same as in the program which implements Wood's algorithm (step 1, p. 22).

(2) The generation of all the minterms (maxterms) is the same as before (step 2, p. 22).

(3) Subroutine FUN performs step (b) of Lawler's algorithm as follows:

(a) Two sets of subscripts are generated using the subroutine MDD (see p. 24). The first set of subscripts (KSUB) is used to generate combinations of untagged minterms (maxterms) which will be used to

produce subfunctions of the original function. The second set of subscripts (ISUB) is used to generate all the possible combinations of minterms (maxterms) which can be used to generate subfunctions of the original function (as in Wood's program, step 3(a), p. 23).

- (b) The two sets of minterms are then compared. If the former is a subset of the latter, then the latter is used to select the minterms (maxterms) which will be used in the consensus operation to find the prime implicants of the subfunction being generated (as in Wood's program, step 3(b), pp. 24, 25).
- (c) Next, subroutine CONSNS (Wood's program, step 4, pp. 25, 26) finds the prime implicants of the selected subfunction.
- (d) Then subroutine TABLE (called by CONSNS) finds the irredundant ps (sp) expression for the selected subfunction (as in Wood's program, step 5, pp. 25, 26, 27).
- (e) Subsequently, the number of literals of the subfunction just generated is checked. If this number is less than or equal to the minimum number of literals of the expression(s) generated to this point which are implied by the minterms (maxterms) contained in the array MUZ, then the terms of the subfunction just generated are stored in the arrays

MINMIN (which is used in the subroutine COVER to find the intermediate covering expressions) and ISOL (which is used in the final steps to find the minimal sps (psp) expression(s)).

- (f) Subroutine COVER finds the expression which covers the minterms of the original function used thus far to generate subfunctions. This operation is performed after 1, 3, 7, 15, ..., 2^n-1 ($n=1,2,3, \dots$) subfunctions have been generated. It uses the ones which were generated in the previous set(s) as well as the subfunctions generated most recently to form a prime implicant table which is then simplified.
- (g) After the intermediate covering expression is found the minterms which are covered by this expression (stored in MINMIN) are tagged so that they will not be used in the generation of the next set of subfunctions by subroutine FUN.
- (h) Sometimes the intermediate covering expression is a simple subfunction implied by the minterms tagged thus far, instead of a combination of subfunctions as found by the subroutine COVER. In order to allow for this possibility, the simple subfunction implied by the minterms (maxterms) tagged so far is generated by the subroutines CONSNS and TABLE (as in Wood's program, steps 4, 5, pp. 25-27). If the

number of literals in this simple subfunction is less than or equal to the number of literals of the expression found by COVER, then the simple subfunction is stored as an intermediate solution.

- (i) Next, the program checks to see if all the minterms (maxterms) have been tagged. If not, then the first non-tagged minterm (maxterm) is added to the list of minterms (maxterms) used to generate subfunctions, and the program goes back through the main loop to generate the next set of subfunctions. If all of the minterms have been tagged, then the program uses the array MINMIN to form the final prime implicant table as follows:

- (i) First, it must be kept in mind that the array MINMIN contains the minterms (maxterms) covered by each of the subfunctions and intermediate covering expressions generated by the program.
 - (ii) Each of these expressions is examined individually, and its minterms (maxterms) are compared to the minterms (maxterms) of the original function. Whenever the minterms match, a "1" is placed in the corresponding column to indicate that the corresponding minterm (maxterm) is covered by the expression being examined. This operation was originally

done in a two-dimensional array, but it was discovered that the same effect could be achieved by using a one dimensional array in which the binary equivalents of decimal numbers could be used to indicate which min-terms were covered by which expression. For example, if the original function is $f = \sum(0,3,6,11)$, and one of the subfunctions (intermediate covering expressions) covers the minterms 0, 6, 11, then the corresponding entry in the prime implicant table would be 1011.

- (j) After the prime implicant table is constructed, the minimal ps (sp) and sps (psp) solutions are found and converted from numerical to literal form (as in Wood's program, steps 6, 7, 8, pp. 29-30).
- (k) The last two steps are the same as in Wood's program (p. 31).

CHAPTER V

CONCLUSION

In the process of writing the computer programs which implement Wood's and Lawler's algorithms, the possibility of combining the best features of the two algorithms was considered, and the result was the combined algorithm illustrated in the flowchart in Figure 3.

The major disadvantage of Wood's algorithm is that the final prime implicant table contains 2^m-1 (m = number of terms of the original function) g functions, which means that a considerable amount of time is spent simplifying the final prime implicant table when m is large. A major disadvantage of Lawler's algorithm is that some of the g functions are generated more than once, so that instead of generating 2^m-1 g functions, the algorithm generates two to three times that many g functions. Another disadvantage of Lawler's algorithm is that when the minterms (maxterms) of a function are grouped "loosely" (see Figure 4), a considerable amount of time is spent trying to form intermediate covering solutions which don't exist.

A major advantage of Wood's algorithm is that regardless of whether the minterms (maxterms) of a function are grouped "tightly" (see Figure 5) or "loosely" (see Figure 4), the processing time is reasonable (see Figure 6). A major

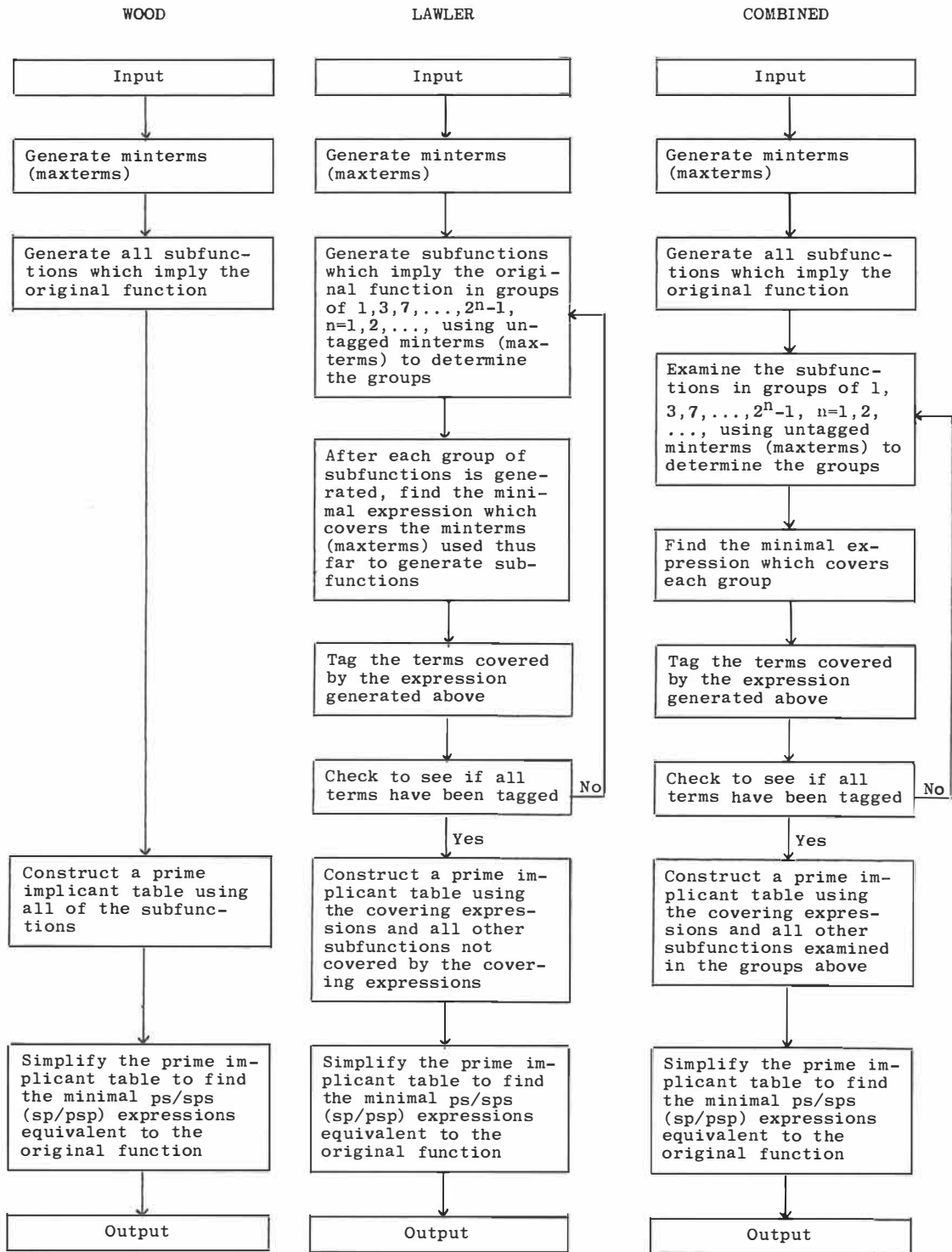


Figure 3. Flowcharts of the minimization algorithms.

yz \ wx				
	00	01	11	10
00	x		x	
01		x		x
11	x		x	
10		x		x

Figure 4. Karnaugh map for $f = \sum(0,3,5,6,9,10,12,15)$.

yz \ wx				
	00	01	11	10
00	x	x		
01	x	x		
11	x	x		
10	x	x		

Figure 5. Karnaugh map for $f = \sum(0,1,2,3,4,5,6,7)$.

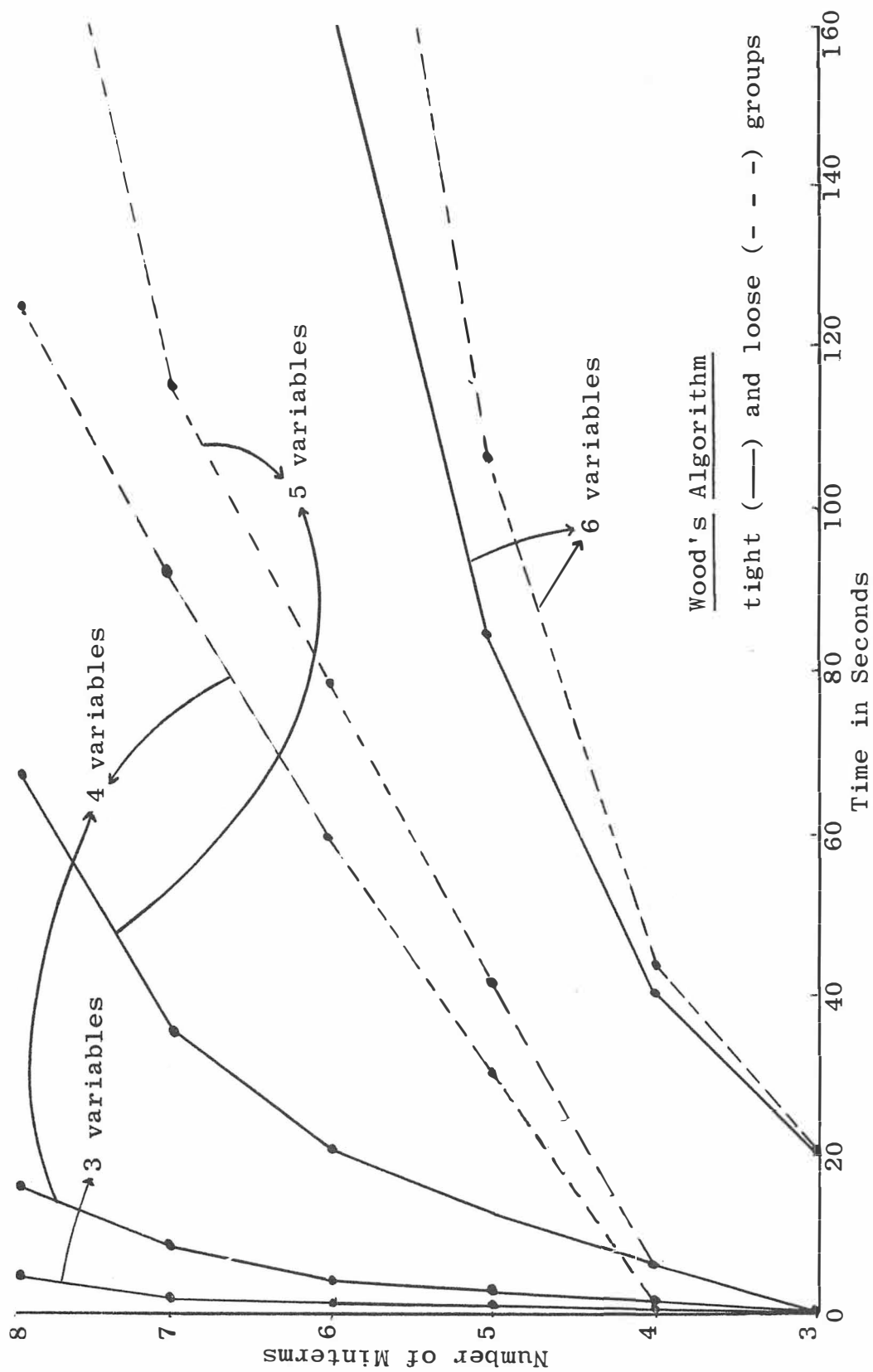


Figure 6. Number of minterms, number of variables, time graph for Wood's algorithm.

advantage of Lawler's algorithm is that it can process functions whose minterms (maxterms) are "tightly" grouped (see Figure 5) in considerably less time than either Wood's algorithm or the combined algorithm (see Figures 6, 7, 8).

A major disadvantage of the combined algorithm is that when the minterms (maxterms) of a function are grouped "loosely" (see Figure 4) the algorithm spends a considerable amount of time trying to form intermediate covering solutions which don't exist. The major advantages of the combined algorithm are: (a) only $2^m - 1$ g functions are generated instead of two to three times that many as in Lawler's algorithm; (b) the final prime implicant table contains considerably fewer entries than the equivalent table in Wood's algorithm, thus reducing the processing time of the final prime implicant table.

Figures 6, 7, and 8 display the relationships between the number of variables of a Boolean function, the number of minterms of a Boolean function, and the amount of time required by the three computer programs to minimize two functions. The two functions used to obtain the time data for Figures 6, 7, and 8 were a "tightly" grouped function ($f = \sum(0,1,2,3,4,5,6,7)$, see Figure 5), and a "loosely" grouped function ($f = \sum(0,3,5,6,9,10,12,15)$, see Figure 4). In all of the graphs the processing time increases as the number of minterms and/or the number of variables increases, except in Lawler's algorithm when the number of minterms is

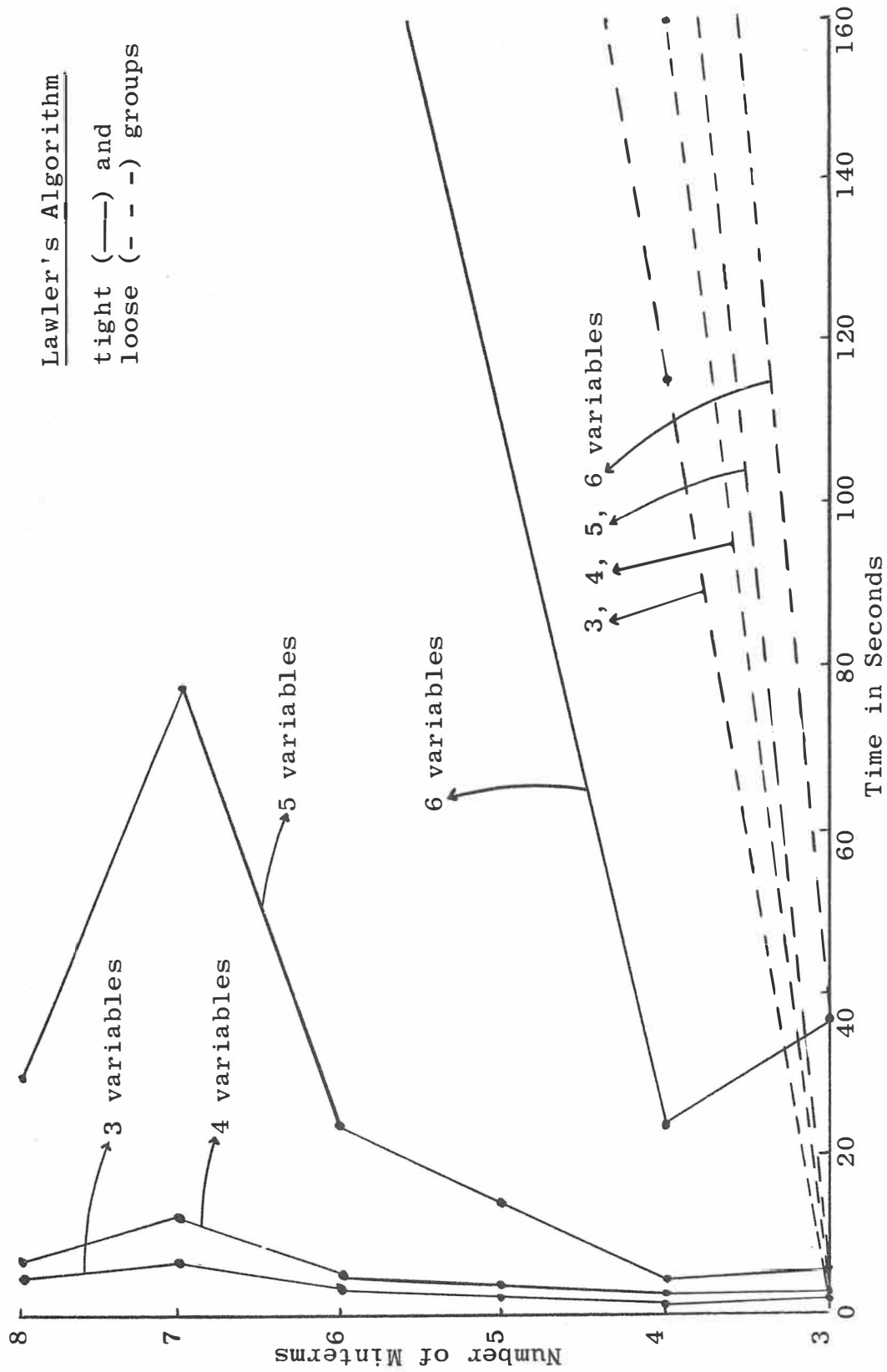


Figure 7. Number of minterms, number of variables, time graph for Lawler's algorithm.

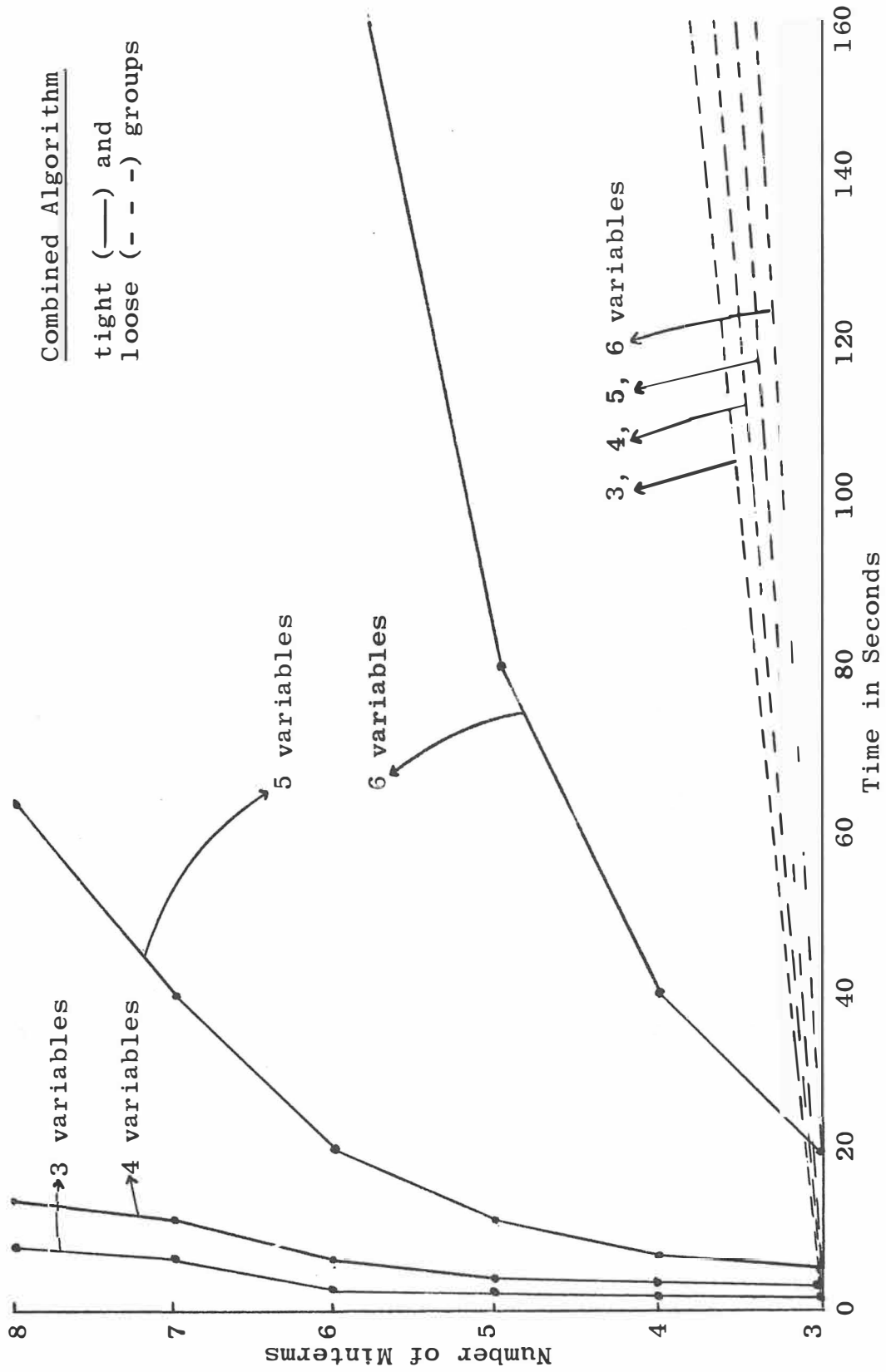


Figure 8. Number of minterms, number of variables, time graph for combined algorithm.

a power of 2 and the minterms are grouped "tightly." The reason for the reduction in processing time is that whenever the minterms of a function are adjacent to each other ("tightly" grouped) in groups of 2, 4, 8, ..., 2^n , this "power of 2"-group of minterms can be represented by a simpler expression. For example, if $f = \sum(0,1,2,3,4,5,6,7) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'zy'z + w'xyz' + w'xyz$ (see Figure 5), then $f = w'$. Since a number of minterms of the original function are covered by a simpler expression, the covered minterms are tagged, and therefore are not used to generate g functions (thus cutting down the processing time considerably).

The least squares method of fitting a curve to a set of data was used to find the following set of equations which can be used to estimate the amount of time required by the combined algorithm program to process a given function (N = number of variables in the given function, x = number of minterms in the given function).

$$(a) \ N = 3, \text{ time} = 0.03x^3 + 0.71x^2 - 3.91x + 6.58$$

$$(b) \ N = 4, \text{ time} = 0.46x^2 - 3.15x + 5.95$$

$$(c) \ N = 5, \text{ time} = 0.52x^3 - 5.48x^2 + 21.88x - 27.56$$

$$(d) \ N = 6, \text{ time} = 14.58x^2 - 82.02x + 133.19$$

In the course of this investigation the following conclusions were drawn:

(1) Even though theoretically the algorithms studied

can minimize any Boolean function, having any number of variables, made up of any number of minterms (maxterms), and with any number of levels, in practice, this generality is considerably limited for the following reasons:

- (a) As the number of variables of a function increases, the total number of minterms (maxterms) increases exponentially (2^m , m = number of variables in the function), thus greatly augmenting the amount of time required to generate the prime implicants of the g functions by the iterated consensus of minterms (maxterms) method.
- (b) As the number of minterms (maxterms) of a function increases, the total number of g functions increases exponentially ($2^m - 1$, m = number of terms of the function), thus greatly augmenting the amount of time required to generate the g functions.
- (c) The generation of multiple level solutions of a Boolean function is a recursive procedure; that is, prime-ps implicants of f are products of sum prime implicants of f , prime-sps implicants of f are sums of prime-ps implicants of f , and so forth.

(2) The computer programs presented in this thesis were limited to handling Boolean functions with at most seven variables. The programs could have handled ten-variable functions (since in Fortran IV the largest possible

number of digits-per-integer is 10), but the limit was set at seven variables because of the great amount of time required to process functions with eight or more variables.

(3) The most significant limitation of the computer programs implemented in this thesis is the great amount of time required to process functions with a large number of minterms (maxterms) and/or variables. The programs were run on an IBM 360-65 computer. On a faster machine the processing time would be reduced considerably. Perhaps another way of reducing the processing time would be by implementing the programs in an assembly language, thus taking advantage of the binary representation of minterms (maxterms) to perform the most time consuming operations; that is, iterated consensus of minterms (maxterms), generation of solutions of subsets of minterms (maxterms), and simplification of the final prime implicant table.

(4) A possibility for future research would be to develop an algorithm to determine whether a set of minterms (maxterms) is either "tightly" grouped, or "loosely" grouped, or neither. Once this is established, then the best method for simplifying the given function could be easily determined; that is, if a set of terms is grouped "tightly," then use Lawler's algorithm; if a set of terms is grouped "loosely," then use Wood's algorithm; if a set of terms is grouped neither way, then use the combined algorithm.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Boole, George. An Investigation of the Laws of Thought. London, 1854.
2. Shannon, C. E. "A Symbolic Analysis of Relay and Switching Circuits," AIEE Transactions, vol. 57, pp. 713-723, 1938.
3. Wood, Paul E., Jr. Switching Theory. McGraw-Hill Book Co., New York, 1968.
4. Lawler, Eugene L. "An Approach to Multiple Level Boolean Minimization," Journal of the ACM, vol. 11, no. 3, pp. 283-295, 1964.
5. Rao, Voddi Venkat Rama. "Minimization of Boolean Functions with Aid of a Digital Computer," Unpublished Master's Thesis, The University of Tennessee, Knoxville, 1967.
6. Beeson, Cecil E., Jr. "Computer Implemented Minimization of Multiple-output Boolean Functions," Unpublished Master's Thesis, The University of Tennessee, Knoxville, 1969.
7. Carroll, C. C. and Mott, H. "Numerical Procedures for Boolean Function Minimization," IEEE Transactions on Electronic Computers, EC-13:470, 1964.
8. McCluskey, E. J. Introduction to the Theory of Switching Circuits. New York, 1965.
9. Barnes, Ben B. "Class Notes EE 5610," The University of Tennessee, Knoxville, 1969.

APPENDIXES

APPENDIX A

BOOLEAN THEOREMS

A. Reflexive Theorem:	$X=X$
B. Symmetric Theorem:	if $X=Y$ then $Y=X$
C. Transitive Theorem:	if $X=Y$ and $Y=Z$ then $X=Z$
D. Substitution Theorems:	if $X=Y$ then $X+Z=Y+Z$ if $X=Y$ then $XZ=YZ$ if $X=Y$ then $X'=Y'$
E. Commutative Theorems:	$X+Y=Y+X$ $XY=YX$
F. Associative Theorems:	$X+(Y+Z)=(X+Y)+Z$ $X(Y+Z)=(XY)+Z$
G. Distributive Theorems:	$X+YZ=(X+Y)(X+Z)$ $X(Y+Z)=XY+XZ$
H. Idempotence Theorems:	$X+X=X$ $XX=X$
I. Universal Bounds Theorems:	$0+X=X$ $1X=X$ $1+X=1$ $0X=0$
J. Complements Theorems:	$X+X'=1$ $XX'=0$
K. Dualization Theorems:	$(X+Y)'=X'Y'$ $(XY)'=X'+Y'$
L. Absorption Theorems:	$X+XY=X$ $X(X+Y)=X$
M. Involution Theorem:	$(X')'=X$

APPENDIX B

EXAMPLE USING WOOD'S ALGORITHM

The following is an example of how Wood's algorithm works. The steps in the example correspond to the steps in the algorithm (pp. 21-22).

(a) $f = \Sigma(0, 2, 4, 7) = x'y'z' + x'yz' + xy'z' + xyz$

(b,c) All the functions which imply f and their prime- π s implicants are:

$$g_1 = \Sigma(0) = x'y'z'$$

$$g_2 = \Sigma(2) = x'yz'$$

$$g_3 = \Sigma(4) = xy'z'$$

$$g_4 = \Sigma(7) = xyz$$

$$g_5 = \Sigma(0, 2) = x'z'$$

$$g_6 = \Sigma(0, 4) = y'z'$$

$$g_7 = \Sigma(0, 7) = (x+z')(y'+z)(x'+y)$$

$$g_8 = \Sigma(2, 4) = z'(x'y+xy')$$

$$g_9 = \Sigma(2, 7) = y(x+z')(x'+z)$$

$$g_{10} = \Sigma(4, 7) = x(y'+z)(y+z')$$

$$g_{11} = \Sigma(0, 2, 4) = z'(x'+y')$$

$$g_{12} = \Sigma(0, 2, 7) = (x+z')(x'+z)(x'+y)$$

$$g_{13} = \Sigma(0, 4, 7) = (y+z')(x+z')(y'+z)$$

$$g_{14} = \Sigma(2, 4, 7) = (x+y)(y+z')(x+z')(x'+y'+z)$$

$$g_{15} = \Sigma(0, 2, 4, 7) = (y+z')(x+z')(x'+y'+z)$$

(d) Construct a prime implicant table for f.

		Minterms			
		0	2	4	7
g_1	$= \Sigma(0)$	1	0	0	0
g_2	$= \Sigma(2)$	0	1	0	0
g_3	$= \Sigma(4)$	0	0	1	0
g_4	$= \Sigma(7)$	0	0	0	1
g_5	$= \Sigma(0, 2)$	1	1	0	0
g_6	$= \Sigma(0, 4)$	1	0	1	0
g_7	$= \Sigma(0, 7)$	1	0	0	1
g_8	$= \Sigma(2, 4)$	0	1	1	0
g_9	$= \Sigma(2, 7)$	0	1	0	1
g_{10}	$= \Sigma(4, 7)$	0	0	1	1
g_{11}	$= \Sigma(0, 2, 4)$	1	1	1	0
g_{12}	$= \Sigma(0, 2, 7)$	1	1	0	1
g_{13}	$= \Sigma(0, 4, 7)$	1	0	1	1
g_{14}	$= \Sigma(2, 4, 7)$	0	1	1	1
g_{15}	$= \Sigma(0, 2, 4, 7)$	1	1	1	1

(e)	<u>Possible solutions</u>	<u>Number of literals</u>
	$g_1 + g_{14}$	10
	$g_1 + g_{15}$	8
	$g_2 + g_{13}$	9
	$g_2 + g_{15}$	10
	$g_3 + g_{12}$	9
	$g_3 + g_{15}$	10
	$g_4 + g_{11}$	6
	$g_4 + g_{15}$	10
	$g_5 + g_{10}$	7
	$g_5 + g_{13}$	8
	$g_5 + g_{14}$	11
	$g_5 + g_{15}$	9
	$g_6 + g_9$	7
	$g_6 + g_{12}$	8
	$g_6 + g_{14}$	11
	$g_6 + g_{15}$	9
	$g_7 + g_8$	11
	$g_7 + g_{11}$	9
	$g_7 + g_{14}$	15
	$g_7 + g_{15}$	13
	$g_8 + g_{12}$	11
	$g_8 + g_{13}$	11
	$g_8 + g_{15}$	12
	$g_9 + g_{11}$	8

<u>Possible solutions</u>	<u>Number of literals</u>
$g_9 + g_{13}$	11
$g_9 + g_{15}$	12
$g_{10} + g_{11}$	8
$g_{10} + g_{12}$	11
$g_{10} + g_{15}$	12
$g_{11} + g_{12}$	9
$g_{11} + g_{13}$	9
$g_{11} + g_{14}$	12
$g_{11} + g_{15}$	10
$g_{12} + g_{13}$	12
$g_{12} + g_{14}$	15
$g_{12} + g_{15}$	13
$g_{13} + g_{14}$	15
$g_{13} + g_{15}$	13
$g_{14} + g_{15}$	16

(f) The minimum literal sps solution is $g_4 + g_{11} = xyz + z'(x'+y')$. The minimum literal ps solution is $g_{15} =$

$$\Sigma(0,2,4,7) = (y+z')(x+z)(x'+y'+z).$$

APPENDIX C

EXAMPLE USING LAWLER'S ALGORITHM

The following is an example of how Lawler's algorithm works. The steps in the example correspond to the steps in the algorithm (pp. 32-35).

(a) $f = \Sigma(0, 2, 4, 7) = x'y'z' + x'yz' + xy'z' + xyz$

(b) The functions implied by $x'y'z'$ are:

$$g_1 = \Sigma(0) = x'y'z'$$

$$g_2 = \Sigma(0, 2) = x'z'$$

$$g_3 = \Sigma(0, 4) = x'y'$$

$$g_4 = \Sigma(0, 2, 4) = z'(x'+y')$$

$$g_5 = \Sigma(0, 7) = (x+z')(y'+z)(x'+y)$$

$$g_6 = \Sigma(0, 4, 7) = (y+z')(x+z')(y'+z)$$

$$g_7 = \Sigma(0, 2, 7) = (x+z')(x'+z)(x'+y)$$

$$g_8 = \Sigma(0, 2, 4, 7) = (y+z')(x+z')(x'+y'+z)$$

g_3 is minimal; tag $x'y'z'$ and $xy'z'$.

(c₁) The next untagged minterm is $x'yz'$; the subsets of the minterms selected thus far ($x'y'z'$ and $x'yz'$) are $G_1 =$

$(x'y'z')$, $G_2 = (x'yz')$, and $G_3 = (x'y'z', x'yz')$; the

functions implied by G_1 have already been generated in

step (b); the functions implied by G_2 are:

$$g_1 = \Sigma(2) = x'yz'$$

$$g_2 = \Sigma(0, 2) = \text{same as above}$$

$$g_3 = \Sigma(2, 4) = z'(x'y+xy')$$

$$g_4 = \Sigma(2, 7) = y(x+z')(x'+z)$$

$$g_5 = \Sigma(0, 2, 4) = \text{same as above}$$

$$g_6 = \Sigma(0, 2, 7) = \text{same as above}$$

$$g_7 = \Sigma(2, 4, 7) = (x+y)(y+z')(x+z')(x'+y'+z)$$

$$g_8 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_2 is minimal:

the functions implied by G_3 are:

$$g_1 = \Sigma(0, 2) = \text{same as above}$$

$$g_2 = \Sigma(0, 2, 4) = \text{same as above}$$

$$g_3 = \Sigma(0, 2, 7) = \text{same as above}$$

$$g_4 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_2 is minimal.

(d₁) The intermediate covering solution for the g functions generated thus far is $i = (0, 2, 4) = z'(x'+y')$.

(e₁) Tag $x'y'z'$ (already tagged) and $x'yz'$ and xyz' (already tagged).

(c₂) The next untagged minterm is xyz ; the subsets of the minterms selected thus far ($x'y'z'$ and $x'yz'$ and xyz) are:

$$G_1 = (x'y'z'), G_2 = (x'yz'), G_3 = (x'y'z', x'yz'),$$

$$G_4 = (xyz), G_5 = (x'y'z', xyz), G_6 = (x'yz', xyz),$$

$G_7 = (x'y'z', x'yz', xyz)$; the functions implied by G_1, G_2, G_3 , have already been generated in steps (b) and (c₁);

the functions implied by G_4 are:

$$g_1 = \Sigma(7) = xyz$$

$$g_2 = \Sigma(0, 7) = \text{same as above}$$

$$g_3 = \Sigma(2, 7) = \text{same as above}$$

$$g_4 = \Sigma(4, 7) = x*y' + z)(y+z')$$

$$g_5 = \Sigma(0, 2, 7) = \text{same as above}$$

$$g_6 = \Sigma(0, 4, 7) = (y+z')(x+y')(y'+z)$$

$$g_7 = \Sigma(2, 4, 7) = \text{same as above}$$

$$g_8 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_1 is minimal;

the functions implied by G_5 are:

$$g_1 = \Sigma(0, 7) = \text{same as above}$$

$$g_2 = \Sigma(0, 2, 7) = \text{same as above}$$

$$g_3 = \Sigma(0, 4, 7) = \text{same as above}$$

$$g_4 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_3 is minimal;

the functions implied by G_6 are:

$$g_1 = \Sigma(2, 7) = \text{same as above}$$

$$g_2 = \Sigma(0, 2, 7) = \text{same as above}$$

$$g_3 = \Sigma(2, 4, 7) = \text{same as above}$$

$$g_4 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_1 is minimal;

the functions implied by G_7 are:

$$g_1 = \Sigma(2, 4, 7) = \text{same as above}$$

$$g_2 = \Sigma(0, 2, 4, 7) = \text{same as above}$$

g_2 is minimal.

- (d₂) The intermediate covering solution for the terms selected thus far ($x'y'z'$ and $x'yz'$ and xyz) is $i = \Sigma(0, 2, 4, 7) = (y+z')(x+z')(x'+y'+z)$.

(e₂) Tag $x'y'z'$, $x'yz'$, $xy'z'$ (all three have already been tagged) and xyz .

(f) All the terms of the original function have been tagged.

(g)		Minterms			
		0	2	4	7
	minimal $G_1 = \Sigma(0,4)$	1	0	1	0
	minimal $G_2 = \Sigma(0,2)$	1	1	0	0
	minimal $G_3 = \Sigma(0,2,4)$	1	1	1	0
	minimal $G_4 = \Sigma(7)$	0	0	0	1
	minimal $G_5 = \Sigma(0,4,7)$	1	0	1	1
	minimal $G_6 = \Sigma(2,7)$	0	1	0	1
	minimal $G_7 = \Sigma(0,2,4,7)$	1	1	1	1

(h)	<u>Possible solutions</u>	<u>Number of laterals</u>
	$G_1 + G_6$	7
	$G_1 + G_7$	9
	$G_2 + G_5$	8
	$G_2 + G_7$	9
	$G_3 + G_4$	6
	$G_3 + G_5$	9
	$G_3 + G_6$	8
	$G_3 + G_7$	10
	$G_4 + G_7$	10
	$G_5 + G_6$	11
	$G_5 + G_7$	13
	$G_6 + G_7$	12

- (i) $G_3 + G_4 = xyz + z'(x'+y')$ is the minimal sps solution because it has the least number of literals. The minimal ps solution is the irredundant ps expression $\Sigma(0, 2, 4, 7) = (y+z')(x+z')(x'+y'+z)$.

APPENDIX D

COMPUTER LISTING OF WOOD'S ALGORITHM PROGRAM

```

C
C   THIS PROGRAM IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
      DIMENSION ISUB(32),MINDEC(128),MAXT(128),MIN(999),
C MIX(999),MINUZD(16),MTAG(16),LINE(16),MAX(128)
      COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
      COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
      COMMON MAXCHK,KSOL(10,32)
C   N IS THE NUMBER OF MINTERMS
C   LENGTH IS THE NUMBER OF LITERALS IN A TERM
1000 READ (5,10) N,LENGTH
      IF(N.EQ.0) STOP
      10 FORMAT (2I2)
C   MINDEC IS THE DECIMAL EQUIVALENT OF A MINTERM
      READ (5,12) (MINDEC(I), I=1,N)
      12 FORMAT (25I3)
      CALL ZEROTK
C   MAXT IS AN ARRAY IN WHICH ARE STORED THE MAXTERMS OR
C   MINTERMS
      CALL MAXX (LENGTH, MAXT)
C   MAXCHK=0 WHILE FINDING PS AND SPS SOLUTIONS
C   MAXCHK>0 WHILE FINDING SP AND PSP SOLUTIONS
      2 MAXCHK=0
      DO 7 I=1,N
C   MIN IS THE NUMERIC EQUIVALENT OF A MINTERM
      7 MIN(I)=MAXT(MINDEC(I)+1)
      PRINT 5
      5 FORMAT (1H0, 'MINTERMS OF THE ORIGINAL FUNCTION')
      PRINT 6, (MIN(I), I=1,N)
      6 FORMAT (1H ,6I10)
      GO TO 11
      9 PRINT 8
      8 FORMAT (1H0, 'MAXTERMS OF THE ORIGINAL FUNCTION')
      PRINT 6, (MIN(I), I=1,N)
C   NN IS THE TOTAL NUMBER OF SUBFUNCTIONS THAT MUST
C   BE GENERATED
      11 NN=2**N-1
C   NUMTER IS THE TOTAL NUMBER OF TERMS GENERATED
      NUMTER=2**LENGTH
      MINLIT=100
      CALL ONE (NN,N,NUMTER,MAXT,MIN,LENGTH)
      CALL SPORPS (NN)
      CALL SPSPSP (NN,N)
      CALL TIMETK (T)
      PRINT 1, T

```

```

1  FORMAT (1H , 'TIME=', F6.2)
   IF(MAXCHK.GT.0) GO TO 1000
C   THE NEXT FEW INSTRUCTIONS SELECT THE MAXTERMS TO BE
C   USED IN FINDING THE MINIMAL SP AND PSP EXPRESSIONS
799 DO 800 I=1,N
   DO 800 J=1,NUMTER
   IF(MIN(I).EQ.MAXT(J)) MAXT(J)=-MAXT(J)
800 CONTINUE
   J=0
   DO 801 I=1,NUMTER
   IF(MAXT(I).LT.0) GO TO 801
   J=J+1
   MIN(J)=MAXT(I)
801 CONTINUE
   N=J
   DO 802 I=1,NUMTER
802 MAXT(I)=IABS(MAXT(I))
   MAXCHK=MAXCHK+1
   CALL ZEROTK
   GO TO 9
END

```

```

SUBROUTINE MAXX (LENGTH, MAXT)

C
C THIS SUBROUTINE GENERATES ALL THE MAXTERMS OR MINTERMS
C IN NUMERICAL FORM
C THE NUMBER OF LITERALS IN EACH OF THE TERMS IS
C SPECIFIED BY "LENGTH"
C

  DIMENSION MAXT(128)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  N=LENGTH
  NN=2**N
  DO 700 I=1,NN
    MAX=0
    IDIV=I-1
    J=1
705  IQUD=IDIV/2
    INEW=IQUD*2
    IDIF=IDIV-INEW
    IF(IDIF.EQ.0) GO TO 710
    MAX=MAX+2*10**(J-1)
    J=J+1
    IDIV=IQUD
    IF(IDIV.EQ.0) GO TO 720
    GO TO 705
710  MAX=MAX+10**(J-1)
    J=J+1
    IF(IDIV.EQ.0) GO TO 720
    IDIV=IQUD
    GO TO 705
720  NM1=N-1
    DO 740 K=1,NM1
      NTEN=10**K
      IF(MAX.LT.NTEN) GO TO 750
      GO TO 740
750  MAX=MAX+NTEN
740  CONTINUE
      MAXT(I)=MAX
700  CONTINUE
      PRINT 729
729  FORMAT (1H0, 'ALL MAXTERMS OR MINTERMS IN NUMERICAL
C FORM')
      PRINT 730,(MAXT(I), I=1,NN)
730  FORMAT (1H0,5I10)
      RETURN
      END

```

```

SUBROUTINE ONE (NN,N,NUMTER,MAXT,MIN,LENGTH)
C
C THIS SUBROUTINE GENERATES ALL THE G FUNCTIONS OF THE
C FUNCTION BEING MINIMIZED
C
  DIMENSION ISUB(32),MAXT(128),MIN(999),MIX(999),
C MAX(128)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  DO 600 I=1,NN
C   III IS THE NUMBER OF THE SUBFUNCTION BEING GENERATED
   III=I
C   THE NEXT FEW INSTRUCTIONS GENERATE THE COMBINATIONS OF
C   TERMS THAT WILL ENTER INTO THE CONSENSUS OPERATION
   CALL MDD(N,ISUB,I,J)
C   JJJ IS THE NUMBER OF TERMS THAT WILL ENTER INTO THE
C   CONSENSUS OPERATION
   JJJ=J-1
607 DO 615 L=1,NUMTER
C   MAX IS AN ARRAY IN WHICH ARE STORED THE MINTERMS OR
C   MAXTERMS TEMPORARILY
615 MAX(L)=MAXT(L)
C   THE NEXT FEW INSTRUCTIONS SELECT THE MINTERMS OR
C   MAXTERMS THAT WILL BE USED IN THE CONSENSUS OPERATION
   DO 620 L=1,JJJ
   DO 620 LL=1,NUMTER
C   TAG TERMS THAT WILL NOT BE USED IN THE CONSENSUS
C   OPERATION
   IF(MAX(LL).EQ.MIN(ISUB(L))) MAX(LL)=-MIN(ISUB(L))
620 CONTINUE
622 J=1
   DO 630 L=1,NUMTER
   IF(MAX(L).LT.0) GO TO 630
C   STORE IN MIX THE TERMS THAT WILL BE USED IN THE
C   CONSENSUS OPERATION
   MIX(J)=MAX(L)
   J=J+1
630 CONTINUE
C   NTERMS IS THE NUMBER OF TERMS TO BE USED IN THE
C   CONSENSUS OPERATION
   NTERMS=J-1
   CALL CONSNS(NTERMS,MIX,LENGTH)
600 CONTINUE
   RETURN
   END

```

```

SUBROUTINE MDD(N,ISUB,I,J)
C
C   THIS SUBROUTINE USES MODULAR ARITHMETIC TO CONVERT
C   WHOLE NUMBERS TO THEIR MIRROR BINARY EQUIVALENTS;
C   THEN, WHENEVER A "1" OCCURS IN THE MIRROR BINARY
C   NUMBER, THE "1" IS CONVERTED TO ITS DECIMAL POSITIONAL
C   VALUE AND STORED IN ISUB; THUS ISUB CONTAINS ALL THE
C   POSSIBLE COMBINATIONS OF NUMBERS FROM 1 TO N
C
  DIMENSION ISUB(32)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  J=1
C   N IS THE NUMBER OF TERMS THAT WILL BE USED TO FORM
C   COMBINATIONS
  DO 604 II=1,N
C   I IS THE NUMBER BEING CONVERTED TO MIRROR BINARY FORM
    ISUB(J)=MOD(I/2**((II-1),2)
    IF(ISUB(J).EQ.1) GO TO 603
    GO TO 604
603  ISUB(J)=II
      J=J+1
604  CONTINUE
      RETURN
      END

```

```

SUBROUTINE CONSNS (N,MIN,LENGTH)
C
C   THIS SUBROUTINE USES NUMERICAL ITERATED CONSENSUS TO
C   FIND THE PRIME IMPLICANTS OF THE FUNCTION WHOSE TERMS
C   ARE STORED IN MIN
C
  DIMENSION INDX(500,9),J(9),L(9)
  DIMENSION MIN(999),MINCON(999),MAXT(128)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  NN=N
  DO 15 I=1,N
15  MAXT(I)=MIN(I)
  LP1=LENGTH+1
  DO 99 I=1,LP1
99  J(I)=0
  DO 100 I=1,N
  NUM2S=0
  MTEMP=MIN(I)
C   COUNT THE NUMBER OF UNCOMPLEMENTED VARIABLES
101 MQUD=MTEMP/10
  MDIF=MTEMP-MQUO*10
  IF(MDIF.EQ.2) NUM2S=NUM2S+1
  IF(MQUO.EQ.0) GO TO 102
  MTEMP=MQUO
  GO TO 101
102 DO 103 I2=1,LP1
  NO2S=I2-1
  IF(NUM2S.NE.NO2S) GO TO 103
  J(I2)=J(I2)+1
C   STORE TERMS ACCORDING TO NUMBER OF UNCOMPLEMENTED
C   VARIABLES
  INDX(J(I2),I2)=MIN(I)
  GO TO 100
103 CONTINUE
100 CONTINUE
  L(LP1)=0
98  NTOT=0
  DO 109 I=1,LENGTH
  K=0
  NCON=0
  I3=J(I)
  IF(I3.EQ.0) GO TO 114
  I4=J(I+1)
  IF(I4.EQ.0) GO TO 114
C   ATTEMPT TO FORM CONSENSUS TERMS BETWEEN ADJACENT GROUPS
  DO 104 I5=1,I3
  DO 104 I6=1,I4
  MIN1=IABS(INDX(I5,I))
  MIN2=IABS(INDX(I6,I+1))

```

```

MDIF=IABS(MIN1-MIN2)
C CHECK TO SEE WHETHER A CONSENSUS TERM HAS BEEN FORMED
  IF(MDIF.EQ.1) GO TO 106
  IF(MDIF.EQ.10) GO TO 106
  IF(MDIF.EQ.100) GO TO 106
  IF(MDIF.EQ.1000) GO TO 106
  IF(MDIF.EQ.10000) GO TO 106
  IF(MDIF.EQ.100000) GO TO 106
  IF(MDIF.EQ.1000000) GO TO 106
  IF(MDIF.EQ.10000000) GO TO 106
  GO TO 104
106 NCON=NCON+1
  NTOT=NTOT+1
C STORE THE CONSENSUS TERM
  MINCON(NTOT)=MINO(MIN1,MIN2)-MDIF
C TAG THE TERMS USED IN THE CONSENSUS OPERATION
  INDX(I5,I)=-MIN1
  INDX(I6,I+1)=-MIN2
104 CONTINUE
C NCNN IS THE NUMBER OF CONSENSUS TERMS (WITHOUT
C DUPLICATES) GENERATED BETWEEN THE PREVIOUS PAIRS OF
C GROUPS (IF ANY)
  NCNN=NTOT-NCON
C CHECK TO SEE WHETHER 2 OR MORE CONSENSUS TERMS HAVE
C BEEN FORMED; IF NOT GO TO THE NEXT PAIR OF ADJACENT
C GROUPS; IF YES, CHECK FOR DUPLICATES
  IF(NCNN.LT.2) GO TO 114
  NTOTM1=NTOT-1
  NPLUS=NTOT-NCON+1
C TAG DUPLICATE CONSENSUS TERMS
  DO 110 K1=NPLUS,NTOTM1
    MIN1=IABS(MINCON(K1))
    K1P1=K1+1
    DO 110 K2=K1P1,NTOT
      MIN2=IABS(MINCON(K2))
      IF(MIN1.EQ.MIN2) MINCON(K2)=-MIN2
110 CONTINUE
C ELIMINATE TAGGED TERMS FROM THE CONSENSUS LIST
  DO 111 K1=NPLUS,NTOT
    IF(MINCON(K1).LT.0) GO TO 117
    NCNN=NCNN+1
    MINCON(NCNN)=MINCON(K1)
    GO TO 111
117 K=K+1
111 CONTINUE
  NTOT=NCNN
114 L(I)=NCON-K
109 CONTINUE
C CHECK TO SEE WHETHER ANY CONSENSUS TERMS HAVE BEEN
C FORMED DURING THIS PASS
  IF(NTOT.EQ.0) GO TO 199

```

```

      KK=0
C      ELIMINATE THE TERMS USED IN THE CONSENSUS OPERATION
      DO 113 I=1,LP1
      K=0
      I2=J(I)
      IF(I2.EQ.0) GO TO 113
      DO 107 I3=1,I2
      IF(INDX(I3,I).LT.0) GO TO 107
      K=K+1
      INDX(K,I)=INDX(I3,I)
107  CONTINUE
      I3=L(I)
      IF(I3.EQ.0) GO TO 118
C      ADD THE NEW CONSENSUS TERMS TO THE CORRESPONDING
C      GROUPS
      DO 108 I4=1,I3
      KK=KK+1
108  INDX(K+I4,I)=MINCON(KK)
118  J(I)=K+L(I)
113  CONTINUE
C      GO BACK THROUGH THE LOOP AND ATTEMPT TO FORM NEW
C      CONSENSUS TERMS
      GO TO 98
199  KK=0
      DO 115 I=1,LP1
      I2=J(I)
      IF(I2.EQ.0) GO TO 115
      DO 116 I3=1,I2
      KK=KK+1
116  MIN(KK)=INDX(I3,I)
115  CONTINUE
      N=KK
      CALL TABLE (N,NN,MIN,MAXT)
      RETURN
      END

```

```

SUBROUTINE TABLE (K,N,IMP,MAXT)
C
C   THIS SUBROUTINE CONSTRUCTS A TABLE IN WHICH THE
C   HORIZONTAL AXIS REPRESENTS THE TERMS OF THE FUNCTION,
C   AND THE VERTICAL AXIS REPRESENTS THE PRIME IMPLICANTS
C   OF THE FUNCTION; JTAB(I,J)=1 IF TERM(I) INCLUDES
C   IMPLICANT(J), OTHERWISE JTAB(I,J)=0
C
      INTEGER*2 JTAB
      DIMENSION JTAB(128,20),IMP(999),MAXT(128)
      DIMENSION ISUB(32)
      COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
      COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
      COMMON MAXCHK,KSOL(10,32)
      DO 305 J=1,K
      DO 300 I=1,N
      ISUM=MAXT(I)+IMP(J)
301  IQUO=ISUM/10
      IDIF=ISUM-IQUO*10
      IF(IDIF.NE.3) GO TO 302
      GO TO 303
302  ISUM=IQUO
      IF(ISUM.EQ.0) GO TO 304
      GO TO 301
303  JTAB(I,J)=0
      GO TO 300
304  JTAB(I,J)=1
300  CONTINUE
305  CONTINUE
C
C   THE SUBROUTINE NEXT SELECTS THE ESSENTIAL IMPLICANTS
C   (IF ANY) AND FORMS A REDUCED TABLE WHICH HAS ON ITS
C   HORIZONTAL AXIS THE MINTERMS OR MAXTERMS WHICH DO NOT
C   COVER THE ESSENTIAL IMPLICANTS, AND ON THE VERTICAL
C   AXIS THE CONDITIONALLY ELIMINABLE IMPLICANTS (IF ANY)
C
      DO 400 J=1,N
      ISUM=0
      DO 410 I=1,K
410  ISUM=ISUM+JTAB(J,I)
      IF(ISUM.EQ.1) GO TO 420
      GO TO 400
420  DO 430 II=1,K
      IF(JTAB(J,II).EQ.1) GO TO 440
      GO TO 430
440  IF(IMP(II).LT.0) GO TO 400
      IMP(II)=-IMP(II)
      GO TO 400
430  CONTINUE
400  CONTINUE
      DO 450 I=1,K

```

```

      IF(IMP(I).LT.0) GO TO 460
      GO TO 450
460  DO 470 J=1,N
      IF(JTAB(J,I).EQ.1) MAXT(J)=-IABS(MAXT(J))
470  CONTINUE
450  CONTINUE
      J=1
      DO 480 I=1,N
      IF(MAXT(I).LT.0) GO TO 480
      DO 479 JJ=1,K
479  JTAB(J,JJ)=JTAB(I,JJ)
      J=J+1
480  CONTINUE
      N=J-1
      JJJJ=0
      DO 495 I=1,K
      IF(IMP(I).LT.0) GO TO 496
      GO TO 495
496  JJJJ=JJJJ+1
      IF(JJJJ.GT.10) GO TO 492
      IF(JJJJ.GT.MINLIT) GO TO 501
      ISOL(JJJJ,III)=IABS(IMP(I))
      NJJJJ(III)=JJJJ
495  CONTINUE
      IF(N.EQ.0) GO TO 492
      JJJ=0
      J=1
      DO 481 I=1,K
      IF(IMP(I).LT.0) GO TO 482
      DO 483 JJ=1,N
483  JTAB(JJ,J)=JTAB(JJ,I)
      IMP(J)=IMP(I)
      J=J+1
      GO TO 481
482  JJJ=JJJ+1
481  CONTINUE
      K=K-JJJ

```

```

C
C      NEXT, THE SUBROUTINE SELECTS THE COMBINATIONS OF
C      CONDITIONALLY ELIMINABLE IMPLICANTS THAT TOGETHER WITH
C      THE ESSENTIAL IMPLICANTS FORM AN IRREDUNDANT
C      EXPRESSION
C

```

```

      NN=2*K-1
      NSUB=1
500  DO 600 I=1,NN
      CALL MDD(N,ISUB,I,J)
      JJJ=J-1
      IF(JJJ.EQ.NSUB) GO TO 601
      GO TO 600
601  DO 640 L=1,N

```

```
      ISUM=0
      DO 630 M=1, JJJ
      ISUM=ISUM+JTAB(L, ISUB(M))
630  CONTINUE
      IF(ISUM.EQ.0) GO TO 600
640  CONTINUE
      DO 670 II=1, JJJ
      JJJJ=JJJJ+1
      IF(JJJJ.GT.10) GO TO 492
      IF(JJJJ.GT.MINLIT) GO TO 501
      ISOL(JJJJ, III)=IMP(ISUB(II))
      NJJJJ(III)=JJJJ
670  CONTINUE
      GO TO 492
600  CONTINUE
      NSUB=NSUB+1
      IF(NSUB.GT.N) GO TO 492
      GO TO 500
492  LITSUM=0
      NNN=NJJJJ(III)
      DO 711 L=1, NNN
      ITEMP=ISOL(L, III)
712  IQUO=ITEMP/10
      IDIF=ITEMP-IQUO*10
      IF(IDIF.EQ.0) GO TO 707
      LITSUM=LITSUM+1
707  IF(IQUO.EQ.0) GO TO 711
      ITEMP=IQUO
      GO TO 712
711  CONTINUE
      NMLTEX(III)=LITSUM
      RETURN
501  NMLTEX(III)=JJJJ
      RETURN
      END
```

```

SUBROUTINE SPORPS (NN)
C
C THIS SUBROUTINE SELECTS AND OUTPUTS THE MINIMAL SP
C (PS) SOLUTION
C
  DIMENSION IMPOUT (22)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  IF(MAXCHK.EQ.0) PRINT 708
  IF(MAXCHK.GT.0) PRINT 803
803 FORMAT (1H0, 'MINIMUM SP SOLUTION')
708 FORMAT (1H0, 'MINIMUM PS SOLUTION')
  NTWM2L=0
  MAXLIT=1
  NSWTCH=0
  IF(MAXLIT.LE.NJJJJ(NN)) MXGTFI=NJJJJ(NN)
  IF(MAXLIT.GE.NJJJJ(NN)) MXGTFI=MAXLIT
  NNN=NJJJJ(NN)
  NGTEL=NNN+1
  DO 700 I=1,NNN
    IMPIN=ISOL(I,NN)
    IF(MAXCHK.EQ.0) CALL CONVRT (IMPIN,IMPOUT,LENGTH)
    IF(MAXCHK.GT.0) CALL CONMIN (IMPIN,IMPOUT,LENGTH)
723 NSWTCH=NSWTCH+LENGTH
    PRINT 752,IMPOUT
752 FORMAT (1H , 25A1)
700 CONTINUE
    NGTIN=NSWTCH+NNN
718 PRINT 709,NSWTCH
709 FORMAT(1H , 'NUMBER OF SWITCHES=',I2)
    PRINT 715,NGTIN
715 FORMAT(1H , 'NUMBER OF GATE INPUTS=', I2)
    PRINT 713,NGTEL
713 FORMAT(1H , 'NUMBER OF GATE ELEMENTS=', I2)
    PRINT 714,MXGTFI
714 FORMAT(1H , 'MAXIMUM GATE FAN-IN=', I2)
  RETURN
  END

```

```

SUBROUTINE SPSPSP (NSOL,N)
C
C THIS SUBROUTINE FINDS THE NUMERICAL SPS (PSP) SOLUTION
C
  DIMENSION IMPOUT(22),ISUB(32),MAXLIT(9),NTWM2L(9)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
C
C THE NEXT FEW INSTRUCTIONS GENERATE THE COMBINATIONS OF
C EXPRESSIONS THAT FORM A SOLUTION AND SELECT THOSE
C THAT HAVE THE LEAST NUMBER OF LITERALS
C
  NN=NSOL
  MINSUM=100
  NM1=NSOL-1
719 DO 700 I=1,NM1
    IP1=I+1
    DO 700 J=IP1,NSOL
      NMLTEX(1)=0
      NTWM2L(1)=0
      MAXLIT(1)=1
      NMLTEX(2)=0
      NTWM2L(2)=0
      MAXLIT(2)=1
      CALL ORR (I,J,K)
      IF(K.NE.NN) GO TO 700
      LITSUM=NMLTEX(I)+NMLTEX(J)
      IF(LITSUM.GT.MINSUM) GO TO 700
      MINSUM=LITSUM
      IF(MAXCHK.EQ.0) PRINT 751
751 FORMAT (1H0, 'MINIMUM SPS SOLUTION')
      IF(MAXCHK.GT.0) PRINT 758
758 FORMAT (1H0, 'MINIMUM PSP SOLUTION')
757 NNN=NJJJJ(I)
      III=1
      DO 753 K=1,NNN
        IMPIN=ISOL(K,I)
        IF(MAXCHK.EQ.0) CALL CONVRT (IMPIN,IMPOUT,LENGTH)
        IF(MAXCHK.GT.0) CALL CONMIN (IMPIN,IMPOUT,LENGTH)
445 IF(LENGTH.GT.1) NTWM2L(III)=NTWM2L(III)+1
        IF(LENGTH.GE.MAXLIT(III)) MAXLIT(III)=LENGTH
        NMLTEX(III)=NMLTEX(III)+LENGTH
        PRINT 752, IMPOUT
752 FORMAT (1H ,22A1)
753 CONTINUE
        IF(MAXCHK.GT.0) GO TO 760
        PRINT 755
755 FORMAT (1H , 'ORED WITH')
        GO TO 761
760 PRINT 762

```

```

762 FORMAT (1H , 'ANDED WITH')
761 NNN=NJJJJ(J)
    III=2
    DO 754 K=1,NNN
      IMPIN=ISOL(K,J)
      IF(MAXCHK.EQ.0) CALL CONVRT (IMPIN,IMPOUT,LENGTH)
      IF(MAXCHK.GT.0) CALL CONMIN (IMPIN,IMPOUT,LENGTH)
545 IF(LENGTH.GT.1) NTWM2L(III)=NTWM2L(III)+1
      IF(LENGTH.GE.MAXLIT(III)) MAXLIT(III)=LENGTH
      NMLTEX(III)=NMLTEX(III)+LENGTH
      PRINT 752, IMPOUT
754 CONTINUE
      NSWTCH=NMLTEX(1)+NMLTEX(2)
      NGTIN=NSWTCH+NTWM2L(1)+NTWM2L(2)+2
      NGTEL=NTWM2L(1)+NTWM2L(2)+3
      IF(MAXLIT(1).GE.MAXLIT(2)) MXGTFI=MAXLIT(1)
      MXGTFI=MAXLIT(2)
      IF(NJJJJ(I).GE.MXGTFI) MXGTFI=NJJJJ(I)
      IF(NJJJJ(J).GE.MXGTFI) MXGTFI=NJJJJ(J)
      PRINT 709,NSWTCH
      PRINT 715,NGTIN
      PRINT 713,NGTEL
      PRINT 714,MXGTFI
709 FORMAT(1H , 'NUMBER OF SWITCHES=',I2)
715 FORMAT(1H , 'NUMBER OF GATE INPUTS=', I2)
713 FORMAT(1H , 'NUMBER OF GATE ELEMENTS=', I2)
714 FORMAT(1H , 'MAXIMUM GATE FAN-IN=', I2)
700 CONTINUE
799 RETURN
    END

```

C SUBROUTINE ORR(I,J,K) FORMS THE BOOLEAN UNION OF I AND
 C J, AND STORES THE RESULT IN K
 C

```

ORR  START  0
      SAVE  (14,12)
      BALR  15,0
      USING *,15
      L     10,0(0,1)  LOAD REG. 10 W. ADD. OF 1ST ARG.
      L     9,0(10)    LD. RG. 9 W. CNT. OF ADD. IN RG.10
      L     10,4(0,1)  LD. REG. 10 W. ADD OF 2ND ARG.
      L     8,0(10)    LD. RG.8 W. CNT. OF ADD. IN RG. 10
      OR     8,9        "OR" THE CONTENTS OF REG. 8 AND 9
      L     10,8(0,1)  LD. ADD. OF 3RD ARG. IN RG. 10
      ST     8,0(10)    STORE CNT.OF RG.8 IN ADD. IN RG.10
      RETURN (14,12)
      END
  
```

```

SUBROUTINE CONVRT (IMPIN, IMPOUT,LENGTH)
C
C THIS SUBROUTINE CONVERTS MAXTERMS FROM THEIR NUMERICAL
C FORM TO THEIR LITERAL FORM
C
      INTEGER*4 PLUS,BLANK,PRIME,RPAREN
      DATA LETV,LETW,LETX,LETY,LETZ,PLUS,BLANK,PRIME,LPAREN,
C RPAREN/1HV,1HW,1HX,1HY,1HZ,1H+,1H-,1H',1H(,1H)/
      DATA LETU,LETT,LETS/1HU,1HT,1HS/
      DIMENSION IMPOUT (22)
      COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
      COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
      COMMON MAXCHK,KSOL(10,32)
      I=1
      LENGTH=0
      IMPOUT(I)=LPAREN
59  IQUO=IMPIN/2000000
      IF(IQUO.EQ.1) GO TO 60
      IQUO=IMPIN/1000000
      IF(IQUO.EQ.1) GO TO 70
      GO TO 89
60  I=I+1
      IMPOUT(I)=LETT
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-2000000
      GO TO 89
70  I=I+1
      IMPOUT(I)=LETT
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-1000000
89  IQUO=IMPIN/200000
      IF(IQUO.EQ.1) GO TO 80
      IQUO=IMPIN/100000
      IF(IQUO.EQ.1) GO TO 90
      GO TO 99
80  I=I+1
      IMPOUT(I)=LETU
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-200000
      GO TO 99
90  I=I+1

```

```
IMPOUT(I)=LETU
LENGTH=LENGTH+1
I=I+1
IMPOUT(I)=PLUS
IMPIN=IMPIN-100000
99  IQUO=IMPIN/20000
    IF(IQUO.EQ.1) GO TO 100
    IQUO=IMPIN/10000
    IF(IQUO.EQ.1) GO TO 110
    GO TO 120
100  I=I+1
     IMPOUT(I)=LETV
     LENGTH=LENGTH+1
     I=I+1
     IMPOUT(I)=PRIME
     I=I+1
     IMPOUT(I)=PLUS
     IMPIN=IMPIN-20000
     GO TO 120
110  I=I+1
     IMPOUT(I)=LETV
     LENGTH=LENGTH+1
     I=I+1
     IMPOUT(I)=PLUS
     IMPIN=IMPIN-10000
120  IQUO=IMPIN/2000
     IF(IQUO.EQ.1) GO TO 130
     IQUO=IMPIN/1000
     IF(IQUO.EQ.1) GO TO 140
     GO TO 150
130  I=I+1
     IMPOUT(I)=LETW
     LENGTH=LENGTH+1
     I=I+1
     IMPOUT(I)=PRIME
     I=I+1
     IMPOUT(I)=PLUS
     IMPIN=IMPIN-2000
     GO TO 150
140  I=I+1
     IMPOUT(I)=LETW
     LENGTH=LENGTH+1
     I=I+1
     IMPOUT(I)=PLUS
     IMPIN=IMPIN-1000
150  IQUO=IMPIN/200
     IF(IQUO.EQ.1) GO TO 160
     IQUO=IMPIN/100
     IF(IQUO.EQ.1) GO TO 170
     GO TO 180
160  I=I+1
```

```
      IMPOUT(I)=LETX
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-200
      GO TO 180
170  I=I+1
      IMPOUT(I)=LETX
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-100
180  IQUO=IMPIN/20
      IF(IQUO.EQ.1) GO TO 190
      IQUO=IMPIN/10
      IF(IQUO.EQ.1) GO TO 200
      GO TO 210
190  I=I+1
      IMPOUT(I)=LETY
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-20
      GO TO 210
200  I=I+1
      IMPOUT(I)=LETY
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PLUS
      IMPIN=IMPIN-10
210  IF(IMPIN.EQ.2) GO TO 220
      IF(IMPIN.EQ.1) GO TO 230
      GO TO 240
220  I=I+1
      IMPOUT(I)=LETZ
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      I=I+1
      IMPOUT(I)=RPAREN
      GO TO 250
230  I=I+1
      IMPOUT(I)=LETZ
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=RPAREN
      GO TO 250
```

```
240 IMPOUT(I)=RPAREN
250 IF(I.GE.22) GO TO 260
    I=I+1
    IMPOUT(I)=BLANK
    GO TO 250
260 RETURN
    END
```

SUBROUTINE CONMIN (IMPIN, IMPOUT, LENGTH)

C
C
C
C

THIS SUBROUTINE CONVERTS MINTERMS FROM THEIR NUMERICAL
FORM TO THEIR LITERAL FORM

```

      INTEGER*4 LETU,LETT
      INTEGER*4 LETV,LETW,LETX,LETY,LETZ,BLANK,PRIME,
C  IMPOUT(22)
      DATA LETV,LETW,LETX,LETY,LETZ,PRIME,BLANK/1HV,1HW,1HX,
C 1HY,1HZ,1H',1H /
      DATA LETU,LETT,LETS/1HU,1HT,1HS/
      COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
      COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
      COMMON MAXCHK,KSOL(10,32)
      I=0
      LENGTH=0
59  IQUO=IMPIN/2000000
      IF(IQUO.EQ.1) GO TO 60
      IQUO=IMPIN/1000000
      IF(IQUO.EQ.1) GO TO 70
      GO TO 89
60  I=I+1
      IMPOUT(I)=LETT
      LENGTH=LENGTH+1
      IMPIN=IMPIN-2000000
      GO TO 89
70  I=I+1
      IMPOUT(I)=LETT
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      IMPIN=IMPIN-1000000
89  IQUO=IMPIN/200000
      IF(IQUO.EQ.1) GO TO 80
      IQUO=IMPIN/100000
      IF(IQUO.EQ.1) GO TO 90
      GO TO 99
80  I=I+1
      IMPOUT(I)=LETU
      LENGTH=LENGTH+1
      IMPIN=IMPIN-200000
      GO TO 99
90  I=I+1
      IMPOUT(I)=LETU
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      IMPIN=IMPIN-100000
99  IQUO=IMPIN/20000
      IF(IQUO.EQ.1) GO TO 100
      IQUO=IMPIN/10000

```

```

        IF(IQUO.EQ.1) GO TO 110
        GO TO 120
100    I=I+1
        IMPOUT(I)=LETV
        LENGTH=LENGTH+1
        IMPIN=IMPIN-20000
        GO TO 120
110    I=I+1
        IMPOUT(I)=LETV
        LENGTH=LENGTH+1
        I=I+1
        IMPOUT(I)=PRIME
        IMPIN=IMPIN-10000
120    IQUO=IMPIN/2000
        IF(IQUO.EQ.1) GO TO 130
        IQUO=IMPIN/1000
        IF(IQUO.EQ.1) GO TO 140
        GO TO 150
130    I=I+1
        IMPOUT(I)=LETW
        LENGTH=LENGTH+1
        IMPIN=IMPIN-2000
        GO TO 150
140    I=I+1
        IMPOUT(I)=LETW
        LENGTH=LENGTH+1
        I=I+1
        IMPOUT(I)=PRIME
        IMPIN=IMPIN-1000
150    IQUO=IMPIN/200
        IF(IQUO.EQ.1) GO TO 160
        IQUO=IMPIN/100
        IF(IQUO.EQ.1) GO TO 170
        GO TO 180
160    I=I+1
        IMPOUT(I)=LETX
        LENGTH=LENGTH+1
        IMPIN=IMPIN-200
        GO TO 180
170    I=I+1
        IMPOUT(I)=LETX
        LENGTH=LENGTH+1
        I=I+1
        IMPOUT(I)=PRIME
        IMPIN=IMPIN-100
180    IQUO=IMPIN/20
        IF(IQUO.EQ.1) GO TO 190
        IQUO=IMPIN/10
        IF(IQUO.EQ.1) GO TO 200
        GO TO 210
190    I=I+1

```

```
      IMPOUT(I)=LETY
      LENGTH=LENGTH+1
      IMPIN=IMPIN-20
      GO TO 210
200  I=I+1
      IMPOUT(I)=LETY
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
      IMPIN=IMPIN-10
210  IF(IMPIN.EQ.2) GO TO 220
      IF(IMPIN.EQ.1) GO TO 230
      GO TO 250
220  I=I+1
      IMPOUT(I)=LETZ
      LENGTH=LENGTH+1
      GO TO 250
230  I=I+1
      IMPOUT(I)=LETZ
      LENGTH=LENGTH+1
      I=I+1
      IMPOUT(I)=PRIME
250  IF(I.GE.22) GO TO 260
      I=I+1
      IMPOUT(I)=BLANK
      GO TO 250
260  RETURN
      END
```

APPENDIX E

COMPUTER LISTING OF LAWLER'S ALGORITHM PROGRAM

```

C
C   THIS PROGRAM IMPLEMENTS LAWLER'S ALGORITHM
C
      DIMENSION ISUB(32),MINDEC(128),MAXT(128),MIN(999),
C MIX(999),MINUZD(16),MTAG(16),LINE(16),MAX(128)
      COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
      COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
      COMMON MAXCHK,KSOL(10,32)
1    READ (5,10) N,LENGTH
10   FORMAT (3I2)
      IF(N.EQ.0) STOP
      CALL ZEROTK
      READ (5,12) (MINDEC(I), I=1,N)
12   FORMAT (25I3)
3    CALL MAXX (LENGTH,MAXT)
2    MAXCHK=0
      DO 7 I=1,N
7     MIN(I)=MAXT(MINDEC(I)+1)
      PRINT 5
5     FORMAT (1H0, 'MINTERMS OF THE ORIGINAL FUNCTION')
      PRINT 6, (MIN(I), I=1,N)
6     FORMAT (1H ,6I10)
      GO TO 11
9     PRINT 8
8     FORMAT (1H0, 'MAXTERMS OF THE ORIGINAL FUNCTION')
      PRINT 6,(MIN(I), I=1,N)
11   NN=2**N-1
      NUMTER=2**LENGTH
      DO 644 J=1,N
      DO 644 I=1,NUMTER
      IF(MIN(J).NE.MAXT(I)) GO TO 644
      MTAG(J)=I
644  CONTINUE
      DO 646 J=1,8
      DO 646 I=1,8
646  MINMIN(I,J)=0
      MINUZD(1)=MIN(1)
      NSOL=0
      NO=1
100  N2=2**NO-1
      N1=N2-NSOL
      DO 101 I=N1,N2
      KKK=I
      NSOL=NSOL+1
      III=NSOL+1

```

```

      CALL ONE (NN,N,NUMTER,MAXT,MIN,KKK,MINUZD,LENGTH)
      NNN=NJJJJ(NSOL)
101  CONTINUE
      CALL COVER (NSOL,NO,LINE,NLINES,LITSUM)
      DO 105 J=1,NLINES
      DO 115 KK=1,N
      DO 114 K=1,N
      IF(MINMIN(KK,LINE(J)).EQ.MIN(K)) MTAG(K)=
      C -IABS(MTAG(K))
114  CONTINUE
115  CONTINUE
105  CONTINUE
      DO 127 I=1,NUMTER
127  MAX(I)=MAXT(I)
      K=0
      DO 121 J=1,N
      IF(MTAG(J).LT.0) GO TO 122
      GO TO 121
122  K=K+1
      MINDEC(K)=MIN(J)
121  CONTINUE
      DO 123 L=1,K
      DO 123 M=1,NUMTER
      IF(MINDEC(L).EQ.MAX(M)) MAX(M)=-MAX(M)
123  CONTINUE
622  MM=0
      DO 128 I=1,NUMTER
      IF(MAX(I).LT.0) GO TO 128
      MM=MM+1
      MIX(MM)=MAX(I)
128  CONTINUE
      CALL CONSNS (MM,MIX,LENGTH)
      IF(NMLTEX(III).GT.LITSUM) GO TO 124
      NNN=NJJJJ(III)
      NJJJJ(NSOL)=NJJJJ(III)
      NMLTEX(NSOL)=NMLTEX(III)
      DO 125 J=1,NNN
125  ISOL(J,NSOL)=ISOL(J,III)
      K=1
      DO 126 J=1,N
      IF(MTAG(J).LT.0) GO TO 131
      GO TO 126
131  MINMIN(K,NSOL)=MIN(J)
      K=K+1
126  CONTINUE
      NLINES=1
      LINE(1)=NSOL
124  DO 109 J=1,N
      IF(MTAG(J).LT.0) GO TO 109
      GO TO 110
109  CONTINUE

```

```

      GO TO 111
110  ND=ND+1
      MINUZD(NO)=MIN(J)
      GO TO 100
111  DO 120 J=1,NSOL
120  ITAB(J)=0
      DO 118 I=1,N2
      DO 118 J=1,N
      DO 118 K=1,N
      IF(MINMIN(J,I).EQ.MIN(K)) ITAB(I)=ITAB(I)+2** (K-1)
118  CONTINUE
      CALL SPORPS (NSOL)
      CALL SPSPSP(NSOL,NN)
      CALL TIMETK (T)
      PRINT 777, T
777  FORMAT (1H0, 'TIME=', F6.2)
      IF(MAXCHK.GT.0) GO TO 1
C    THE NEXT FEW INSTRUCTIONS SELECT THE MAXTERMS TO BE
C    USED IN FINDING THE MINIMAL SP AND PSP EXPRESSIONS
      DO 800 I=1,N
      DO 800 J=1,NUMTER
      IF(MIN(I).EQ.MAXT(J)) MAXT(J)=-MAXT(J)
800  CONTINUE
      J=0
      DO 801 I=1,NUMTER
      IF(MAXT(I).LT.0) GO TO 801
      J=J+1
      MIN(J)=MAXT(I)
801  CONTINUE
      N=J
      DO 802 I=1,NUMTER
802  MAXT(I)=IABS(MAXT(I))
      MAXCHK=MAXCHK+1
      CALL ZERO TK
      GO TO 9
      END

```

```

SUBROUTINE ONE (NN,N,NUMTER,MAXT,MIN,KKK,MUZ,LENGTH)
C
C   THIS SUBROUTINE SELECTS THE EXPRESSION(S) WITH THE
C   LEAST NUMBER OF LITERALS IMPLIED BY THE MINTERMS
C   (MAXTERMS) IN MUZ
C
  DIMENSION KSUB(32),ISUB(32),MAXT(128),MAX(128),
C MIX(999),MIN(999),MUZ(16)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  CALL MDD(N,KSUB,KKK,J)
  NKSUB=J-1
  MINLIT=100
  DO 600 I=1,NN
C
C   THE NEXT FEW INSTRUCTIONS GENERATE THE COMBINATIONS OF
C   TERMS THAT WILL ENTER INTO THE CONSENSUS OPERATION
C
  CALL MDD(N,ISUB,I,J)
  JJJ=J-1
  DO 200 M=1,NKSUB
  DO 206 L=1,JJJ
    IF(MUZ(KSUB(M)).NE.MIN(ISUB(L))) GO TO 206
  GO TO 200
206 CONTINUE
  GO TO 600
200 CONTINUE
607 DO 615 L=1,NUMTER
615 MAX(L)=MAXT(L)
C
C   THE NEXT FEW INSTRUCTIONS SELECT THE TERMS
C   THAT WILL BE USED IN THE CONSENSUS OPERATION
C
  DO 620 L=1,JJJ
  DO 620 LL=1,NUMTER
    IF(MAX(LL).EQ.MIN(ISUB(L))) MAX(LL)=-MIN(ISUB(L))
620 CONTINUE
622 J=1
  DO 630 L=1,NUMTER
    IF(MAX(L).LT.0) GO TO 630
    MIX(J)=MAX(L)
    J=J+1
630 CONTINUE
  NTERMS=J-1
  IM1=III-1
  CALL CONSNS(NTERMS,MIX,LENGTH)
  IF(NMLTEX(III).LE.MINLIT) GO TO 202
  GO TO 600
202 MINLIT=NMLTEX(III)
  NJJJJ(IM1)=NJJJJ(III)

```

```
NMLTEX(IM1)=NMLTEX(III)
DO 204 L=1, JJJ
204 MINMIN(L, IM1)=MIN(ISUB(L))
   NNN=NJJJJ(III)
   DO 205 L=1, NNN
205 ISOL(L, IM1)=ISOL(L, III)
600 CONTINUE
   RETURN
   END
```

```

SUBROUTINE COVER (NROWS,NCOLS,LINE,NLINES,LITSUM)
C
C THIS SUBROUTINE FINDS THE COVERING EXPRESSION FOR THE
C TERMS USED SO FAR TO GENERATE SETS OF SUBFUNCTIONS
C
  DIMENSION ISUB(32),MINSUM(16),LINE(16)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  NSUB=1
  NN=2*NROWS-1
  DO 598 I=1,NROWS
598  MINSUM(I)=100
599  DO 600 I=1,NN
    CALL MDD(NROWS,ISUB,I,J)
    JJJ=J-1
    IF(JJJ.EQ.NSUB) GO TO 601
    GO TO 600
601  IARG=0
    DO 630 M=1,JJJ
      JARG=ISUB(M)
      CALL ORR (IARG,JARG,KARG)
630  IARG=KARG
    IF(IARG.NE.NROWS) GO TO 600
    LITSUM=0
    DO 650 M=1,JJJ
      LINE(M)=ISUB(M)
      LITSUM=LITSUM+NMLTEX(ISUB(M))
    IF(LITSUM.GT.MINSUM(NSUB)) GO TO 600
650  CONTINUE
    NLINES=JJJ
    MINSUM(NSUB)=LITSUM
600  CONTINUE
    NSUB=NSUB+1
    IF(NSUB.GT.NROWS) GO TO 603
    IF(NSUB.EQ.2) GO TO 599
    IF(MINSUM(NSUB-1).GT.MINSUM(NSUB-2)) GO TO 603
    GO TO 599
603  LITSUM=MINSUM(NSUB-1)
    RETURN
    END

```

```

SUBROUTINE SPSPSP (NN,NSOL)
C
C THIS SUBROUTINE FINDS THE NUMERICAL SPS (PSP) SOLUTION
C
  DIMENSION IMPOUT(22),ISUB(32),MAXLIT(9),NTWM2L(9)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
C
C THE NEXT FEW INSTRUCTIONS GENERATE THE COMBINATIONS OF
C EXPRESSIONS THAT FORM A SOLUTION AND SELECT THOSE
C THAT HAVE THE LEAST NUMBER OF LITERALS
C
  MINSUM=100
  NM1=NSOL-1
719 DO 700 I=1,NM1
    IP1=I+1
    DO 700 J=IP1,NSOL
      NMLTEX(1)=0
      NTWM2L(1)=0
      MAXLIT(1)=1
      NMLTEX(2)=0
      NTWM2L(2)=0
      MAXLIT(2)=1
      IARG=ITAB(I)
      JARG=ITAB(J)
      ISUB(1)=I
      ISUB(2)=J
      CALL ORR (IARG,JARG,K)
      IF(K.NE.NN) GO TO 700
      LITSUM=0
      DO 711 M=1,2
        NNN=NJJJJ(ISUB(M))
        DO 711 L=1,NNN
          ITEMP=ISOL(L,ISUB(M))
712  IQUO=ITEMP/10
          IDIF=ITEMP-IQUO*10
          IF(IDIF.EQ.0) GO TO 707
          LITSUM=LITSUM+1
707  IF(IQUO.EQ.0) GO TO 711
          ITEMP=IQUO
          GO TO 712
711  CONTINUE
          IF(LITSUM.GT.MINSUM) GO TO 700
          MINSUM=LITSUM
          IF(MAXCHK.EQ.0) PRINT 751
751  FORMAT (1H0, 'MINIMUM SPS SOLUTION')
          IF(MAXCHK.GT.0) PRINT 758
758  FORMAT (1H0, 'MINIMUM PSP SOLUTION')
757  NNN=NJJJJ(I)
      III=1

```

```

DO 753 K=1,NNN
IMPIN=ISOL(K,I)
IF(MAXCHK.EQ.0) CALL CONVRT(IMPIN,IMPOUT,LENGTH)
IF(MAXCHK.GT.0) CALL CONMIN(IMPIN,IMPOUT,LENGTH)
445 IF(LENGTH.GT.1) NTWM2L(III)=NTWM2L(III)+1
IF(LENGTH.GE.MAXLIT(III)) MAXLIT(III)=LENGTH
NMLTEX(III)=NMLTEX(III)+LENGTH
PRINT 752, IMPOUT
752 FORMAT (1H ,25A1)
753 CONTINUE
IF(MAXCHK.GT.0) GO TO 760
PRINT 755
755 FORMAT (1H , 'ORED WITH')
GO TO 761
760 PRINT 762
762 FORMAT (1H , 'ANDED WITH')
761 NNN=NJJJJ(J)
III=2
DO 754 K=1,NNN
IMPIN=ISOL(K,J)
IF(MAXCHK.EQ.0) CALL CONVRT(IMPIN,IMPOUT,LENGTH)
IF(MAXCHK.GT.0) CALL CONMIN(IMPIN,IMPOUT,LENGTH)
545 IF(LENGTH.GT.1) NTWM2L(III)=NTWM2L(III)+1
IF(LENGTH.GE.MAXLIT(III)) MAXLIT(III)=LENGTH
NMLTEX(III)=NMLTEX(III)+LENGTH
PRINT 752, IMPOUT
754 CONTINUE
NSWTCH=NMLTEX(1)+NMLTEX(2)
NGTIN=NSWTCH+NTWM2L(1)+NTWM2L(2)+2
NGTEL=NTWM2L(1)+NTWM2L(2)+3
IF(MAXLIT(1).GE.MAXLIT(2)) MXGTFI=MAXLIT(1)
MXGTFI=MAXLIT(2)
IF(NJJJJ(I).GE.MXGTFI) MXGTFI=NJJJJ(I)
IF(NJJJJ(J).GE.MXGTFI) MXGTFI=NJJJJ(J)
PRINT 709,NSWTCH
PRINT 715,NGTIN
PRINT 713,NGTEL
PRINT 714,MXGTFI
709 FORMAT(1H , 'NUMBER OF SWITCHES=',I2)
715 FORMAT(1H , 'NUMBER OF GATE INPUTS=', I2)
713 FORMAT(1H , 'NUMBER OF GATE ELEMENTS=', I2)
714 FORMAT(1H , 'MAXIMUM GATE FAN-IN=', I2)
700 CONTINUE
799 RETURN
END

```

```
C      SUBROUTINE MAXX (LENGTH, MAXT)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE MOD(N,ISUB,I,J)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONSNS (N,MIN,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE TABLE (K,N,IMP,MAXT)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE SPORPS (NN)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE ORR(I,J,K)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONVRT (IMPIN, IMPOUT,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONMIN(IMPIN,IMPOUT,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
```

APPENDIX F

COMPUTER LISTING OF THE COMBINED ALGORITHM PROGRAM

```

C
C   THIS PROGRAM IMPLEMENTS THE COMBINED ALGORITHM
C
C   DIMENSION ISUB(32),MINDEC(128),MAXT(128),MIN(999),
C   MIX(999),MINU2D(16),MTAG(16),LINE(16),MAX(128)
C   COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
C   COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
C   COMMON MAXCHK,KSOL(10,32)
C   N IS THE NUMBER OF MINTERMS INPUT
C   LENGTH IS THE NUMBER OF VARIABLES IN A MINTERM
1  READ (5,10) N,LENGTH
10 FORMAT (3I2)
   IF(N.EQ.0) STOP
   CALL ZEROTK
C   MINDEC IS THE DECIMAL EQUIVALENT OF A MINTERM
   READ (5,12) (MINDEC(I), I=1,N)
12 FORMAT (25I3)
C   MAXT IS AN ARRAY WHICH CONTAINS ALL OF THE MINTERMS
   CALL MAXX (LENGTH, MAXT)
C   MAXCHK=0 WHEN THE PROGRAM IS FINDING PS/SPS SOLUTIONS
2  MAXCHK=0
   DO 7 I=1,N
C   MIN IS AN ARRAY WHICH CONTAINS THE MINTERMS OF THE
C   FUNCTION BEING MINIMIZED
7  MIN(I)=MAXT(MINDEC(I)+1)
   PRINT 5
5  FORMAT (1H0, 'MINTERMS OF THE ORIGINAL FUNCTION')
   PRINT 6, (MIN(I), I=1,N)
6  FORMAT (1H ,6I10)
   GO TO 11
9  PRINT 8
8  FORMAT (1H0, 'MAXTERMS OF THE ORIGINAL FUNCTION')
   PRINT 6,(MIN(I), I=1,N)
C   NN IS THE TOTAL NUMBER OF SUBFUNCTIONS
11 NN=2**N-1
C   NUMTER IS THE TOTAL NUMBER OF MINTERMS
   NUMTER=2**LENGTH
   MINLIT=100
   CALL ONE (NN,N,NUMTER,MAXT,MIN,LENGTH)
   CALL SPORPS (NN)
   DO 644 J=1,N
   DO 644 I=1,NUMTER
   IF(MIN(J).NE.MAXT(I)) GO TO 644
C   MTAG IS AN ARRAY WHICH IS USED TO TAG THE MINTERMS
C   WHICH ARE COVERED BY INTERMEDIATE SOLUTIONS

```

```

        MTAG(J)=I
644  CONTINUE
        DO 646 J=1,8
        DO 646 I=1,8
C      MINMIN IS AN ARRAY WHICH CONTAINS THE MINTERMS OF THE
C      MINIMUM LITERAL SUBFUNCTIONS
646  MINMIN(I,J)=0
C      MINUZD IS AN ARRAY WHICH CONTAINS THE MINTERMS WHICH
C      ARE BEING USED TO GENERATE SUBFUNCTIONS
        MINUZD(1)=MIN(1)
C      NSOL KFEPS COUNT OF THE NUMBER OF SUBFUNCTIONS WHICH
C      WILL BE USED IN THE FINAL PRIME IMPLICANT TABLE TO
C      FIND THE MINIMAL SPS (PSP) SOLUTION
        NSOL=0
C      NO IS THE NUMBER OF MINTERMS BEING USED TO GENERATE
C      SUBFUNCTIONS
        NO=1
100  N2=2**NO-1
        N1=N2-NSOL
        DO 101 I=N1,N2
        KKK=I
        NSOL=NSOL+1
        III=NSOL+1
        CALL FUN (NN,N,NUMTER,MAXT,MIN,KKK,MINUZD)
        NNN=NEX(NSOL)
101  CONTINUE
        CALL COVER (NSOL,NO,LINE,NLINES,LITSUM)
C      THIS LOOP TAGS THE MINTERMS WHICH ARE COVERED BY THE
C      INTERMEDIATE SOLUTIONS GENERATED SO FAR
        DO 105 J=1,NLINES
        DO 115 KK=1,N
        DO 114 K=1,N
            IF(MINMIN(KK,LINE(J)).EQ.MIN(K)) MTAG(K)=
C      -IABS(MTAG(K))
114  CONTINUE
115  CONTINUE
105  CONTINUE
        JJJJ=0
        DO 888 K=1,N
            IF(MTAG(K).LT.0) JJJJ=JJJJ+2** (K-1)
888  CONTINUE
            IF(NMLTEX(JJJJ).GT.LITSUM) GO TO 124
            NNN=NJJJJ(JJJJ)
            NEX(NSOL)=NNN
            NLIT(NSOL)=NMLTEX(JJJJ)
            DO 125 J=1,NNN
125  KSOL(J,NSOL)=ISOL(J,JJJJ)
            K=1
            DO 126 J=1,N
            IF(MTAG(J).LT.0) GO TO 131
            GO TO 126

```

```

131 MINMIN(K,NSOL)=MIN(J)
    K=K+1
126 CONTINUE
    NLINES=1
    LINE(1)=NSOL
124 DO 109 J=1,N
    IF(MTAG(J).LT.0) GO TO 109
    GO TO 110
109 CONTINUE
    GO TO 111
110 NO=NO+1
    MINUZD(NO)=MIN(J)
    GO TO 100
111 DO 120 J=1,NSOL
120 ITAB(J)=0
    DO 118 I=1,N2
    DO 118 J=1,N
    DO 118 K=1,N
    IF(MINMIN(J,I).EQ.MIN(K)) ITAB(I)=ITAB(I)+2** (K-1)
118 CONTINUE
    CALL SPSPSP (NSOL,NN)
    CALL TIMETK (T)
    PRINT 777, T
777 FORMAT (1H0, 'TIME=', F6.2)
    IF(MAXCHK.EQ.0) GO TO 1
C    THE NEXT FEW INSTRUCTIONS SELECT THE MAXTERMS TO BE
C    USED IN FINDING THE MINIMAL SP AND PSP SOLUTIONS
799 DO 800 I=1,N
    DO 800 J=1,NUMTER
    IF(MIN(I).EQ.MAXT(J)) MAXT(J)=-MAXT(J)
800 CONTINUE
    J=0
    DO 801 I=1,NUMTER
    IF(MAXT(I).LT.0) GO TO 801
    J=J+1
    MIN(J)=MAXT(I)
801 CONTINUE
    N=J
    DO 802 I=1,NUMTER
802 MAXT(I)=IABS(MAXT(I))
    MAXCHK=MAXCHK+1
    CALL ZERO TK
    GO TO 9
END

```

```

SUBROUTINE FUN (NN,N,NUMTER,MAXT,MIN,KKK,MUZ)
C
C   THIS SUBROUTINE SELECTS THE EXPRESSION(S) WITH THE
C   LEAST NUMBER OF LITERALS IMPLIED BY THE MINTERMS
C   (MAXTERMS) IN MUZ
C
  DIMENSION KSUB(32),ISUB(32),MAXT(128),MAX(128),
C MIX(999),MIN(999),MUZ(16)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
  CALL MDD(N,KSUB,KKK,J)
  NKSUB=J-1
  MINLIT=100
  DO 600 I=1,NN
  CALL MDD(N,ISUB,I,J)
  JJJ=J-1
  DO 200 M=1,NKSUB
  DO 206 L=1,JJJ
  IF(MUZ(KSUB(M)).NE.MIN(ISUB(L))) GO TO 206
  GO TO 200
206 CONTINUE
  GO TO 600
200 CONTINUE
  IM1=III-1
  IF(NMLTEX(I).LE.MINLIT) GO TO 202
  GO TO 600
202 MINLIT=NMLTEX(I)
  NEX(IM1)=NJJJJ(I)
  NLIT(IM1)=NMLTEX(I)
  DO 204 L=1,JJJ
204 MINMIN(L,IM1)=MIN(ISUB(L))
  NNN=NJJJJ(I)
  DO 205 L=1,NNN
205 KSOL(L,IM1)=ISOL(L,I)
600 CONTINUE
  RETURN
  END

```

```

SUBROUTINE SPSPSP (NSOL,NN)
C
C   THIS SUBROUTINE FINDS THE NUMERICAL SPS (PSP) SOLUTION
C
  DIMENSION IMPOUT(22),ISUB(32),MAXLIT(9),NTWM2L(9)
  COMMON III,NMLTEX(512),ISOL(10,512),NJJJJ(512),JJJJ
  COMMON MINLIT,MINMIN(8,8),ITAB(32),NEX(32),NLIT(32)
  COMMON MAXCHK,KSOL(10,32)
C   NSUB IS THE NUMBER OF EXPRESSIONS BEING COMBINED TO
C   FORM A SOLUTION
  NSUB=2
  MINSUM=100
  N2=2** (NSOL-1)-1
  N3=2**NSOL-1
701 NCHK=0
C   THE NEXT FEW INSTRUCTIONS GENERATE THE COMBINATIONS OF
C   EXPRESSIONS THAT FORM A SOLUTION AND SELECT THE ONES
C   THAT HAVE THE LEAST NUMBER OF LITERALS
101 I1M1=N2
  I2M1=N2
  IF(NSUB.EQ.2) GO TO 2
  IF(NSUB.EQ.3) GO TO 3
  IF (NSUB.GT.4) RETURN
C   GENERATE NUMBERS WHOSE BINARY EQUIVALENTS
C   HAVE 2, 3, 4 "1'S"
  4 DO 104 I1=3,N2
    I1M1=I1-1
  3 DO 103 I2=2,I1M1
    I2M1=I2-1
  2 DO 102 I3=1,I2M1
    DO 102 I4=1,I3
      I5=I3-I4
      ICOM=2**I3+2**I5
      IF(ICOM.GT.N3.OR.ICOM.LT.0) GO TO 105
      IF(NSUB.EQ.2) GO TO 106
      ICOM=ICOM+2**I2
      IF(ICOM.GT.N3.OR.ICOM.LT.0) GO TO 105
      IF(NSUB.EQ.3) GO TO 106
      ICOM=ICOM+2**I1
      IF(ICOM.GT.N3.OR.ICOM.LT.0) GO TO 105
106 J=1
C   GENERATE COMBINATIONS OF 2,3,4,... SUBSCRIPTS
  DO 604 II=1,NSOL
    ISUB(J)=MOD(ICOM/2**(II-1),2)
    IF(ISUB(J).EQ.1) GO TO 603
    GO TO 604
603 ISUB(J)=II
    J=J+1
604 CONTINUE
    JJJ=J-1
C   CHECK TO SEE IF THE NUMBER OF SUBSCRIPTS = THE NUMBER

```

```

C      OF EXPRESSIONS BEING COMBINED IN THIS PASS
601  IARG=0
      DO 630 M=1, JJJ
      JARG=ITAB(ISUB(M))
C      "OR" THE MINTERMS COVERED BY THE EXPRESSIONS BEING
C      COMBINED IN THIS PASS
      CALL ORR (IARG, JARG, KARG)
630  IARG=KARG
C      CHECK TO SEE IF THIS COMBINATION OF EXPRESSIONS COVERS
C      ALL MINTERMS
      IF(IARG.NE.NN) GO TO 102
      LITSUM=0
      DO 640 M=1, JJJ
      NNN=NEX(ISUB(M))
C      COUNT THE NUMBER OF LITERALS IN THE COMBINED
C      EXPRESSION
      DO 639 MN=1, NNN
      IDIV=KSOL(MN, ISUB(M))
638  IQUO=IDIV/10
      IDIF=IDIV-IQUO*10
      IF(IDIF.NE.0) LITSUM=LITSUM+1
      IF(IQUO.EQ.0) GO TO 639
      IDIV=IQUO
      GO TO 638
639  CONTINUE
640  CONTINUE
C      CHECK TO SEE IF THE NUMBER OF LITERALS IS MINIMAL
      IF(LITSUM.GT.MINSUM) GO TO 102
      NCHK=1
      MINSUM=LITSUM
      DO 702 J=1, JJJ
      NTWM2L(J)=0
702  MAXLIT(J)=0
      IF(MAXCHK.EQ.0) PRINT 751
751  FORMAT (1H0, 'MINIMUM SPS SOLUTION')
      IF(MAXCHK.GT.0) PRINT 758
758  FORMAT (1H0, 'MINIMUM PSP SOLUTION')
C      OUTPUT MINIMAL SOLUTIONS
757  DO 798 M=1, JJJ
      NNN=NEX(ISUB(M))
      DO 753 K=1, NNN
      IMPIN=KSOL(K, ISUB(M))
C      CONVERT SOLUTION FROM NUMERICAL TO LITERAL FORM
      IF(MAXCHK.EQ.0) CALL CONVRT(IMPIN, IMPOUT, LENGTH)
      IF(MAXCHK.GT.0) CALL CONMIN(IMPIN, IMPOUT, LENGTH)
445  IF(LENGTH.GT.1) NTWM2L(M)=NTWM2L(M)+1
      IF(LENGTH.GE.MAXLIT(M)) MAXLIT(M)=LENGTH
      PRINT 752, IMPOUT
752  FORMAT (1H , 22A1)
753  CONTINUE
      IF(M.GE.JJJ) GO TO 798

```

```

      IF(MAXCHK.GT.0) GO TO 760
      PRINT 755
755  FORMAT (1H , 'ORED WITH')
      GO TO 798
760  PRINT 762
762  FORMAT (1H , 'ANDED WITH')
798  CONTINUE
      MXGTFI=1
      NSWTCH=LITSUM
      NGTIN=0
      DO 797 M=1,JJJ
      IF(MAXLIT(M).GT.MXGTFI) MXGTFI=MAXLIT(M)
      IF(NEX(ISUB(M)).GT.MXGTFI) MXGTFI=NEX(ISUB(M))
797  NGTIN=NGTIN+NTWM2L(M)
      NGTEL=NGTIN+JJJ+1
      NGTIN=NGTIN+NSWTCH+JJJ
      PRINT 709,NSWTCH
      PRINT 715,NGTIN
      PRINT 713,NGTEL
      PRINT 714,MXGTFI
709  FORMAT(1H , 'NUMBER OF SWITCHES=',I2)
715  FORMAT(1H , 'NUMBER OF GATE INPUTS=', I2)
713  FORMAT(1H , 'NUMBER OF GATE ELEMENTS=', I2)
714  FORMAT(1H , 'MAXIMUM GATE FAN-IN=', I2)
102  CONTINUE
      IF(NSUB.GE.3) GO TO 103
      GO TO 105
103  CONTINUE
      IF(NSUB.EQ.4) GO TO 104
      GO TO 105
104  CONTINUE
C    IF A MINIMAL SOLUTION WAS FOUND ON THIS LAST SET OF
C    PASSES USING "NSUB" EXPRESSIONS, TRY TO FIND A "MORE"
C    MINIMAL SOLUTION USING "NSUB + 1" EXPRESSIONS
105  IF(NCHK.EQ.0) GO TO 799
      NSUB=NSUB+1
      GO TO 701
799  RETURN
      END

```

```
C      SUBROUTINE MAXX (LENGTH, MAXT)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE ONE (NN,N,NUMTER,MAXT,MIN,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE MDD (N,ISUB,I,J)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONSNS (N,MIN,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE TABLE (K,N,IMP,MAXT)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE SPORPS (NN)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE COVER (NROWS,NCOLS,LINE,NLINES,LITSUM)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS LAWLER'S ALGORITHM
C
C      SUBROUTINE ORR(I,J,K)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONVRT (IMPIN, IMPOUT,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
C
C      SUBROUTINE CONMIN(IMPIN,IMPOUT,LENGTH)
C      THIS SUBROUTINE IS THE SAME AS IN THE PROGRAM WHICH
C      IMPLEMENTS WOOD'S MINIMIZATION ALGORITHM
```

EXAMPLES OF MINIMIZED FUNCTIONS

EXAMPLE 1.

ALL MAXTERMS OR MINTERMS IN NUMERICAL FORM

111 112 121 122 211 212 221 222

MINTERMS OF THE ORIGINAL FUNCTION

111 121 211 222

MINIMUM PS SOLUTION

 $(X+Z')(Y+Z')(X'+Y'+Z)$

NUMBER OF SWITCHES = 7

NUMBER OF GATE INPUTS = 10

NUMBER OF GATE ELEMENTS = 4

MAXIMUM GATE FAN-IN = 3

MINIMUM SPS SOLUTION

 $XYZ + Z'(X'+Y')$

NUMBER OF SWITCHES = 6

NUMBER OF GATE INPUTS = 9

NUMBER OF GATE ELEMENTS = 4

MAXIMUM GATE FAN-IN = 3

MAXTERMS OF THE ORIGINAL FUNCTION

112 122 212 221

MINIMUM SP SOLUTION

 $X'Z' + Y'Z' + XYZ$

NUMBER OF SWITCHES = 7

NUMBER OF GATE INPUTS = 10

NUMBER OF GATE ELEMENTS = 4

MAXIMUM GATE FAN-IN = 3

MINIMUM PSP SOLUTION

 $\{Z'+XY\}(X'+Y'+Z)$

NUMBER OF SWITCHES = 6

NUMBER OF GATE INPUTS = 9

NUMBER OF GATE ELEMENTS = 4

MAXIMUM GATE FAN-IN = 3

EXAMPLE 2.

ALL MAXTERMS OR MINTERMS IN NUMERICAL FORM

1111	1112	1121	1122	1211	1212	1221	1222
2111	2112	2121	2122	2211	2212	2221	2222

MINTERMS OF THE ORIGINAL FUNCTION

1111 1122 1212 1221 2112

MINIMUM PS SOLUTION

$$(W+X+Y+Z')(X+Y'+Z)(X'+Y+Z)(W'+Z)(W'+Y')(W'+X') \\ (X'+Y'+Z')$$

NUMBER OF SWITCHES = 19

NUMBER OF GATE INPUTS = 26

NUMBER OF GATE ELEMENTS = 8

MAXIMUM GATE FAN-IN = 7

MINIMUM SPS SOLUTION

$$(X+Y)(X+Z)(Y+Z)(W')(X'+Y'+Z') + (W+Z')(W'+Z)(Y'X')$$

NUMBER OF SWITCHES = 16

NUMBER OF GATE INPUTS = 24

NUMBER OF GATE ELEMENTS = 9

MAXIMUM GATE FAN-IN = 5

ANOTHER MINIMAL SPS SOLUTION IS:

$$(X+Y)(W'Z)(X'+Y') + (X+Y')(X'+Y)(W'Z') + (WX'Y'Z)$$

NUMBER OF SWITCHES = 16

NUMBER OF GATE INPUTS = 23

NUMBER OF GATE ELEMENTS = 8

MAXIMUM GATE FAN-IN = 4

EXAMPLE 3.

ALL MAXTERMS OR MINTERMS IN NUMERICAL FORM

11111	11112	11121	11122	11211	11212	11221
11222	12111	12112	12121	12122	12211	12212
12221	12222	21111	21112	21121	21122	21211
21212	21221	21222	22111	22112	22121	22122
22211	22212	22221	22222			

MINTERMS OF THE ORIGINAL FUNCTION

11111	11112	11121	11211	12111	12112	12121
12211	22111	22112	22211			

MINIMUM PS SOLUTION

$$(Y'+Z')(X'+Z')(X'+Y')(U'+Y')(U'+W)$$

NUMBER OF SWITCHES = 10

NUMBER OF GATE INPUTS = 15

NUMBER OF GATE ELEMENTS = 6

MAXIMUM GATE FAN-IN = 5

MINIMUM SPS SOLUTION

$$Y'(X'+Z')(U'+W) + U'X'Z'$$

NUMBER OF SWITCHES = 8

NUMBER OF GATE INPUTS = 12
NUMBER OF GATE ELEMENTS = 5
MAXIMUM GATE FAN-IN = 3

VITA

Alfredo Caldwell was born in Caracas, Venezuela, on August 2, 1941. He attended elementary schools in that city until 1950. For the next five years his family lived in San Juan, Puerto Rico, where he completed his elementary education. From 1955 to 1957 his family again lived in Caracas, Venezuela, where he attended La Salle High School. In 1957 his family moved to Signal Mountain, Tennessee, and he completed his high school education at Notre Dame High School, Chattanooga, Tennessee, in June 1959. The following September he entered St. Bernard College, St. Bernard, Alabama, and in May, 1963, he received a Bachelor of Arts degree in Mathematics. In the fall of 1963, he entered St. Bernard Seminary, St. Bernard, Alabama, where he pursued theological studies for one year. During the following year he attended St. Mary's Seminary, Baltimore, Maryland.

In November 1965, he was accepted by the Peace Corps as a trainee. In May, 1966, he was sent to Nigeria as a high school mathematics teacher where he served until August 1967. From September 1967 to August 1968 he served as a high school mathematics teacher in Tanzania.

In the fall of 1968 he accepted a teaching assistantship at The University of Tennessee and began study toward a Master's degree. Since August 1970, he has been employed

by Oak Ridge Associated Universities. In December 1970, he received the Master of Science degree with a major in Computer Science.

He is married to the former Judith Anne Schaerer, of Chattanooga, Tennessee