MINIMIZATION PROCEDURE IN THE DESIGN OF DIGITAL SYSTEMS

A Thesis

Presented to

the Faculty of the Department of Electrical Engineering

The University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

Robert E. Taranto

1966

ACKNOWLEDGMENT

Grateful acknowledgment is made to Professor T.N. Whitaker and Mr. A.H. McMorris for their counsel and guidance. The author wishes to thank the many who were helpful and considerate during the preparation of this thesis.

MINIMIZATION PROCEDURE IN THE DESIGN OF DIGITAL SYSTEMS

An Abstract of a Thesis
Presented to
the Faculty of the Department of Electrical Engineering
The University of Houston

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

---

by
Robert E. Taranto
1966

ABSTRACT

Representing that research resolves itself through development to beneficial application, this thesis proposes an advancement in the technique of economical logic design.

While much work has been performed in perfecting implementation of hardware, existing methods fail to provide absolute minimality in all applications. The objective is to provide a procedure for the logic designer to develop solutions through rapid, error free multi-comparison methods which provide minimal expression and the proof and record thereof.

The presentation considers existing hardware, present minimization methods and their problems; develops examples describing the procedure and finally describes a computer program for economical reliable design.

Time consuming calculations are eliminated through the development of graphical solutions by computerprograms, rather than existing numerical methods of minimizing the Boolean expression. The approach reported here utilizes a combination of present methods, with new interpretations, to obtain minimized optimum circuitry - while simultaneously displaying the basic characteristics of the Boolean expression.

TABLE OF CONTENTS

LIST OF FIGURE

CHAPTER I

GENERAL OUTLINE

Introduction

The rapid advance of integrated circuit technology has increased the applications for digital systems, requiring the logic designer to solve a variety of problems of function implementations. Obtaining minimal solutions requires the use of tedious, time consuming calculations.

While numerical methods and computer programs do reduce the logic designer's effort, a solution must be obtained before the designer will be in the position to optimize his function implementation. A procedure is required to permit logic circuit optimization rapidly, reliably, economically and simply. This procedure must allow the designer to compare the effects of parameter variance, without resorting to tedious calculations. The procedure described here utilizes a computer program to achieve the implementation of Boolean functions.

Review and Definitions

Similarities between logic propositions and switching circuits permit use of algebraic logic for analysis, synthesis and minimization. The algebra of logic uses the postulates and theorems of appendix I.

The following definitions apply throughout this thesis:

Literal: A variable or its complement, (A or $\overline{A}$).

Minterm or Product term: Literals in a group multiplied together, (A.B.C).

Maxterm or Sum term: Literals summed together, (A+B+C).

Canonical term or Standard term: A term containing exactly one occurrence of the literals, f (A,B,C) = A$\overline{B}$C + ... is an example of a canonical product term.

Sum of Products form: A function form, where the product terms are summed, (AB + $\overline{A}$C + BC).

Product of Sums form: A function form, where the sum terms are multiplied, (A+B) (A+$\overline{C}$) (B+C).

Canonical form or Standard form: A function form, where all the terms are canonical and appear only once. The canonical form may be expressed as a product of sums or as a sum of products, each being unique.

Truth Table or Table of Combinations: A tabular representation of a function, giving the function value for each of the possible combinations of the variables. There are 2^n row combinations for a function of n variables.

Boolean Expression: A function form that describes some logical properties or a network construction by Boolean algebra.

Logic Diagram: The logic circuit configuration implemented from the logic expression.

Analysis: The derivation of the truth tables from block diagrams or equations.

Synthesis: Deriving equations from truth tables.

Minimization: Manipulation of Boolean functions culminating with implementation in a block diagram with some parameters minimized.

Hardware

Hardware representation of present state of the art appears in the form of monolithic integrated circuit performs a multitude of logic functions in the form of NAND, NOR, and EXOR.

The advantages obtained by integrated circuits are:

High speed

High fan-out capability

High noise immunity

Low power dissipation

Moderate cost

Miniaturization

The appendix contains standard logic symbols and diagrams referring to integrated circuits.

Minimization Methods

The process of finding the form of a given Boolean function, which may be implemented directly into hardware at lower cost and increased reliability, is known as "minimization" in the logic design. Cost and reliability are directly related to minimality of quantity of standard logic blocks, switching levels and block interconnections.

Arbitrary application of Boolean algebraic laws to the manipulations of algebraic representations of logic functions may result in prohibitive labor to achieve simplification. A definite need exists for a systematic simplification procedure. Of the large number of published methods the majority are oriented to relay circuit minimization and are categorized in three groups:

1. An algebraic method for minimization found by Quine.[1] Any function of any number of variables can be minimized to a two level minimum expression by first expanding the function to its canonical form, comparing each term against the others and eliminating redundant terms by using the basic theorem:

$$AB + A\bar{B} = A$$

A selection must then be made of the fewest number of irredundant terms (prime implicants) that yield the function. The two main disadvantages of this method are that the expression is only the two level minimum sum of products or product of sums, possibly far from being the minimum one. The method is time consuming and prone to errors.

2. Mc Cluskey reduced the disadvantage involving time by using binary numbers.[2] The resulting numerical method uses the same algorithms as Quine's, where each term of the canonical expression is represented as a binary number as in:

$$T = \bar{A}\bar{B}\bar{C}\bar{D} + ABCD$$

$$T = \sum(0,15)$$

Digital computer programs have applied this method to solve Boolean functions. The minimized function obtained is a minimum two level expression of sum

of products or product of sums. However, it is not readily adaptable into logic blocks.

3. The Graphical method originated by Veitch [3] has been extended by others, possibly the best being the Karnaugh maps and partition matrix. This geometrical representation gives an insight into the functions and permits acquisition of a real minimal circuit for the hardware to be used, as will be seen in the following chapters. The graphical method is limited inasmuch, as functions above six variables must be disjunctively treated as two subfunctions, each with six or less variables. Such occurrences are infrequent. In most cases digital systems are built as functional modules in which the number of variables is less than six.

Procedure Outline

Briefly, the optimizing procedure is:

1. Engineering and functional specifications are combined to determine the logic specification for a system.
2. This specification is then reduced to functionally related logic subsystems for example decoders, arithmetic units, input-output, control, etc.

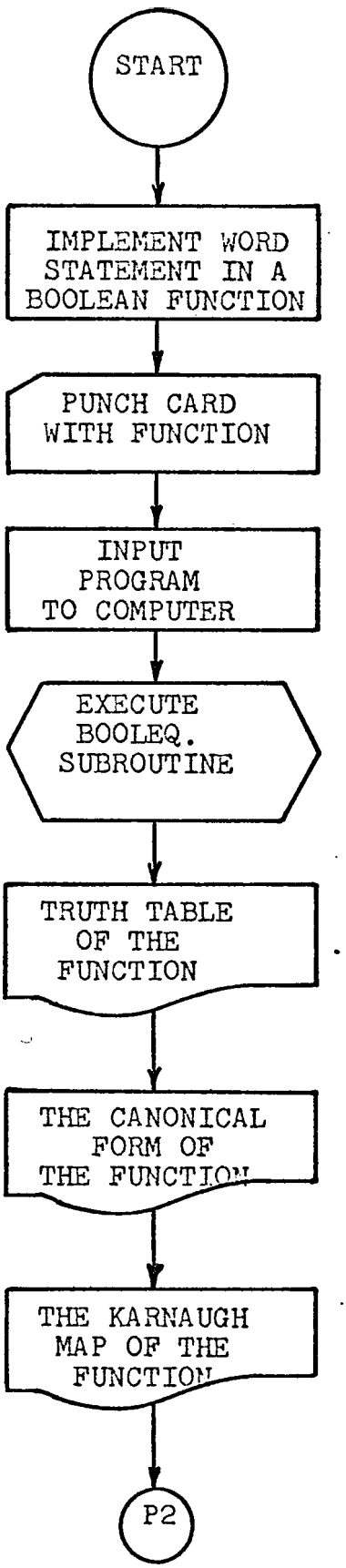
3. The logic subsystems are described by Boolean expressions.
4. Each Boolean expression in punched card form is read by the computer and the "BOOLEQ" subroutine is executed.
5. The resulting output includes the information as shown in figure 1.1.

   a) The truth table of the function.
   b) The canonical form of the function.
   c) The Karnaugh map of the function.
   d) The partition matrix of the function.
   e) The minimum number of terms.

The procedure now provides three alternatives, depending upon the number and grouping of terms. The first alternative is selected when examination of the Karnaugh map reveals an approximately equal number of minterms and maxterms without large groupings. A close examination of the partition matrix must be made to obtain a disjunctive decomposition function as explained in Chapter IV. The resultant expression, if found, is then ready for hardware implementation.

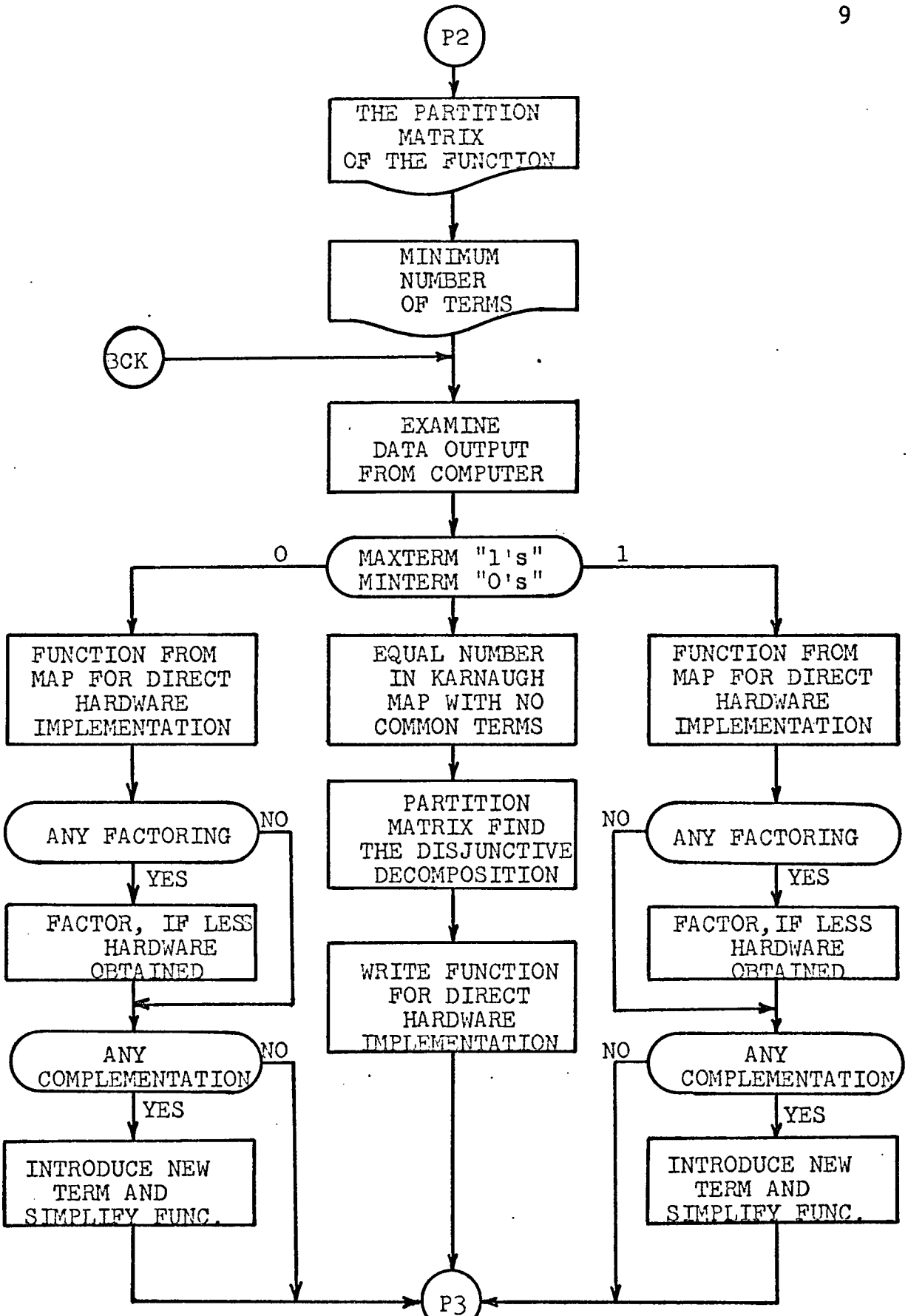
The last two alternative procedures are similar. One utilizes the minterms and the other utilizes the maxterms. In both cases a close examination of the Karnaugh maps will reveal a direct form for hardware implementation. Factoring may be done, if indicated, and if less hardware is required.

The procedure also provides for inhibiting, introducing new terms and simplification of functions.



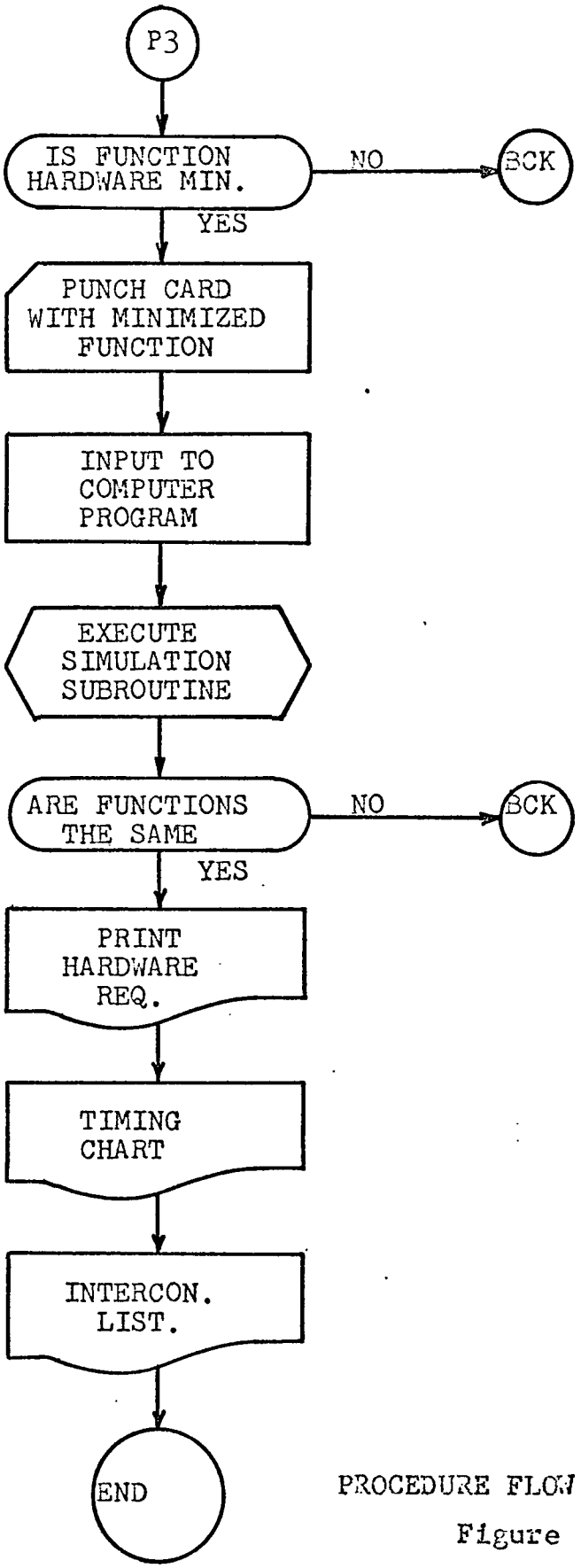
PROCEDURE FLOW DIAGRAM

Figure 1.1-a



PROCEDURE FLOW DIAGRAM

Figure 1.1-b



PROCEDURE FLOW DIAGRAM

Figure 1.1-c

Should satisfactory minimization not be achieved, the designer re-examines the print-out.

When the minimized function is obtained, it is processed by a computer simulation program. If the output does not display the equivalence of the original and minimized functions, an error has occurred and the designer must re-examine his work. When the two functions are equivalent, the hardware requirements and interconnection list may be produced.

Chapter II is wholly devoted to Karnaugh maps and the process of factoring, inhibiting and function transformation, while Chapter III explains implementation of NAND-NOR logic.

CHAPTER II

THE KARNAUGH MAP IN MINIMIZATION

A preferred graphical method for minimization is the Karnaugh map.[4] It achieves simplification of switching functions by applying elementary geometrical concepts related to algebraic properties. With previous experience of map handling, a visual inspection can minimize a function to two or more levels minimal expression of any form. Factoring and inhibiting permits direct implementation of the function into integrated circuit logic blocks of the form NAND/NAND, NOR/NOR, NOR/OR etc.

Description

Geometrically, a switching function is expressed graphically as a map. Each n variables are allocated 2^n cells, one for every possible input combination of the variables. (See figure 2.1). Functions are represented by entering a 1 in the cells which are associated with each term in the algebraic function. Cells with 1 are minterms to the canonical expression of the function. The map completely defines the function.

Terms are so ordered that any two adjacent cells will differ in only one variable. The cells along the left-hand edge are adjacent to the corresponding cells along the right-hand edge. Similarly, the top edge is adjacent to the

bottom edge. This arrangement is refered to as reflected ordering to differ from the straight binary, where more than one variable is permitted to change. The adjacency of opposite ends of row and column can be better observed by considering the map as being inscribed on a Torus. Close examination of the map reveals that adjacent cells, either horizontally or vertically, both having 1, can be grouped according to the distributive and complementary algebraic rules:

$$T = AB + A\overline{B}$$

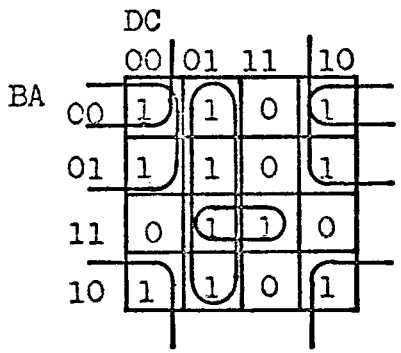
$$T = A(B+\overline{B}) = A$$

Larger numbers of cells can be grouped if the number of adjacent 1's is some power of 2. The terms are defined by constant variables throughout the group.

General Rules

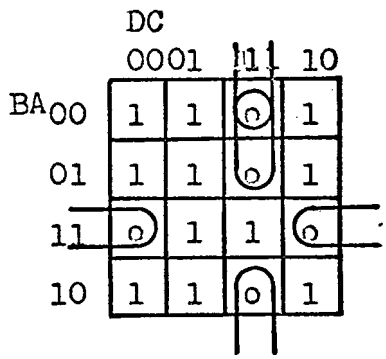
A good approach for optimum solution is embodied in the following rules:

1. Computer subroutine plots Karnaugh map.
2. Account for all 1 cells that can be grouped only one way.
3. Cells that do not combine are prime implicants.
4. Combine group of two cells. When all groups of two are exhausted, combine groups of four and so forth.



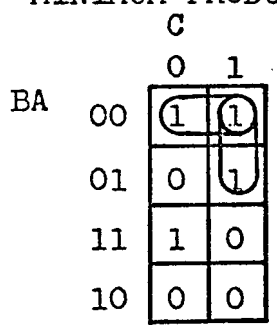
$$T = \bar{A}\bar{C} + ABC + \bar{B}\bar{C} + C\bar{D}$$

MINIMUM SUM OF PRODUCTS



$$T = (AB\bar{C} + \bar{B}DC + \bar{A}DC)'$$
$$T = (\bar{A}+\bar{B}+C)\cdot(\bar{B}+\bar{D}+\bar{C})\cdot(A+\bar{D}+\bar{C})$$

MINIMUM PRODUCT OF SUMS



$$T = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$$
$$T = \bar{A}\bar{B} + \bar{B}C + AB\bar{C}$$
$$T = \bar{B}\ (\bar{A}+C) + AB\bar{C}$$

THREE AND FOUR VARIABLE KARNAUGH MAPS

Figure 2.1

5. Write the function, taking in account each group in the form required.

The function obtained is not necessarily the minimal two stage algebraic form. Fewer terms and/or variables may possibly result from grouping 0's and applying the General Rules.

The three and four variable maps are illustrated in the example in figure 2.1. As seen, the minimum sum of products has more terms than the minimum product of sums, each function being directly derived from the map by grouping the 1's or the 0's.

$$T = \bar{A}\bar{C} + ABC + \bar{B}\bar{C} + C\bar{D} \quad \text{(From the 1's)}$$

$$T = (AB\bar{C} + \bar{B}DC + \bar{A}DC)' \quad \text{(From the 0's)}$$

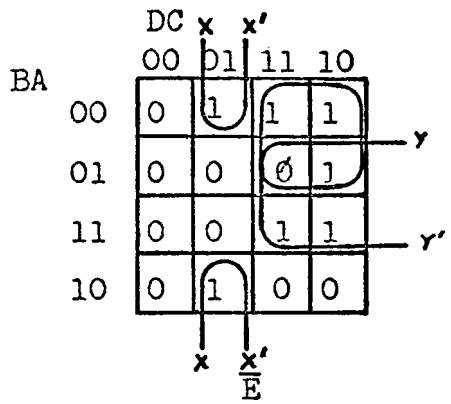
$$T = (\bar{A} + \bar{B} + C).\ (B + \bar{D} + \bar{C}).\ (A + \bar{D} + \bar{C})$$

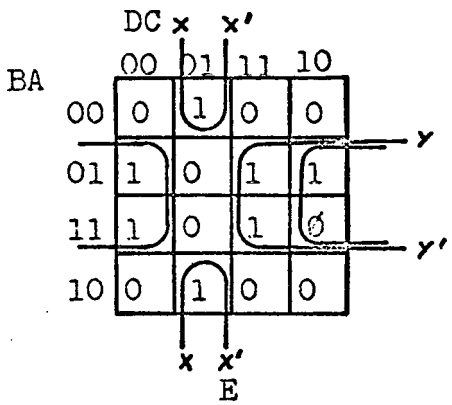
Optional terms may be introduced for functions whose values are not all specified. These terms can be specified as either 1 or 0.

Five and Six Variable Maps

When minimizing expressions of five variables, two four variable maps are plotted. The two maps provide the same information, giving the 32 combinations of the five variables and their complements. Each map covers the same four variables, while the fifth variable is assigned

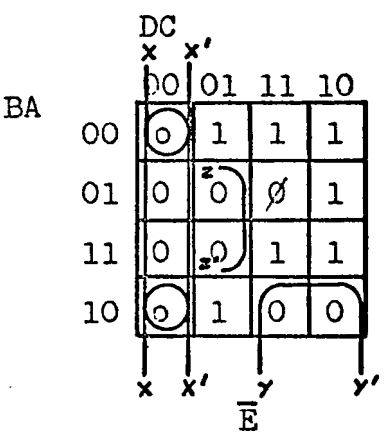
$$T = \bar{A}\bar{B}D\bar{E} + \bar{A}C\bar{D}E + A\bar{C}\bar{D}E + \bar{A}C\bar{D} + ABD\bar{E} + A\bar{C}D\bar{E} + ACDE + A\bar{B}\bar{C}E$$
$$OP = A\bar{B}CD\bar{E} + AB\bar{C}DE$$

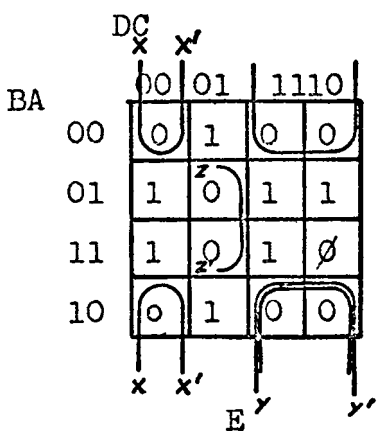




$$T = \bar{A}C\bar{D} + AD + \bar{B}D\bar{E} + A\bar{C}E$$

FIVE VARIABLE MAP IMPLEMENTING FROM 1's





$$T = (A+\bar{C}+D)\cdot(A+\bar{D}+\bar{E})\cdot(A+\bar{B}+\bar{D})\cdot(C+D+E)\cdot(A+C+D)$$

FIVE VARIABLE IMPLEMENTING FROM 0's

Figure 2.2

0 on the first map and 1 on the second map. Adjacency in each map occurs between the same cells on different maps in addition to the original adjacencies. A five variable example is shown in figure 2.2. The illustration demonstrates the advantage of using Karnaugh Map, when some terms are optional.

Six variables are displayed by plotting four four-variable maps. Each represents sixteen combinations of the first four variables, while the fifth and sixth variables take the values 00, 01, 10, 11 for each consecutive map.

Six variable map minimization requires practice to obtain proficiency. The four maps are viewed as if there were a third dimension map displaying the adjacencies between the maps. The example below shows some of the simplification possibilities. The computer program has been restricted to six variables, usefulness above six being very limited. Figure 2.3 illustrates a six variable map and its solution of a function representing the prime binary numbers of the first six digits:

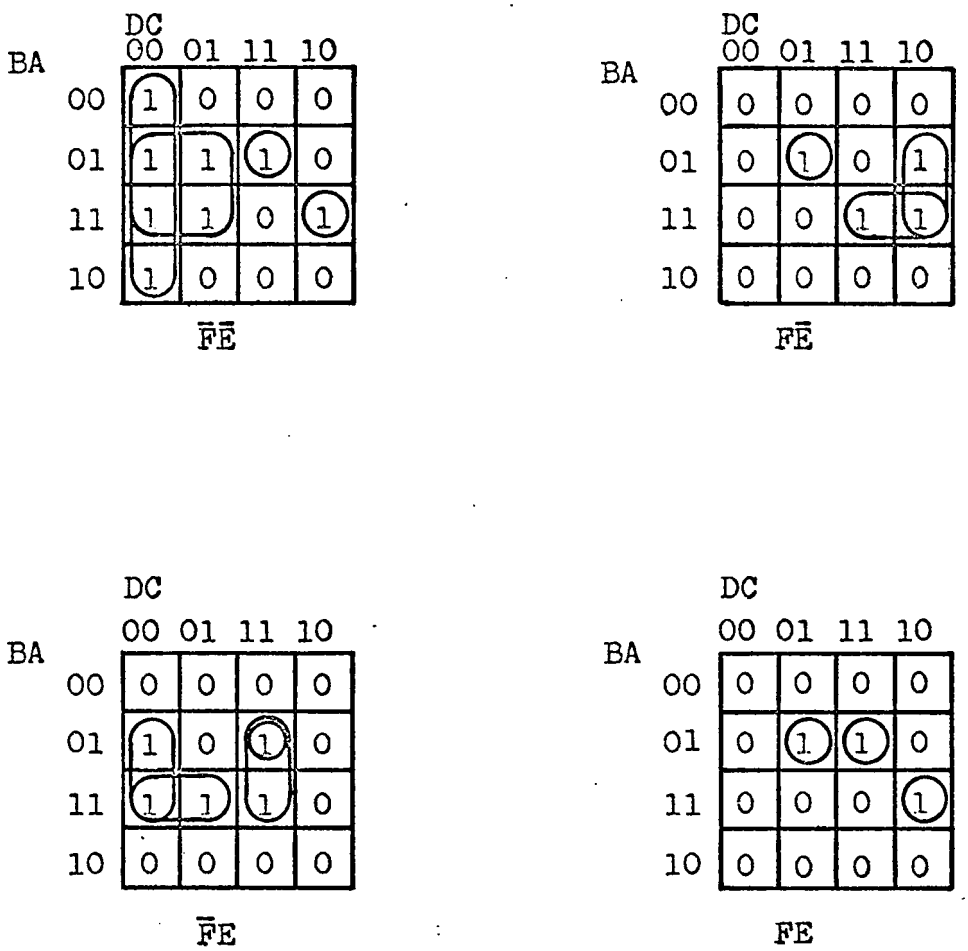
$$T = \Sigma(0,1,2,3,5,7,11,13,17,19,23,29,31,41,43,47)$$

Where the minimum sum of products from the maps is:

$$T = \bar{C}\bar{D}\bar{E}\bar{F}+A\bar{D}\bar{E}\bar{F}+AC\bar{D}\bar{F}+AB\bar{D}\bar{F}+A\bar{B}CD\bar{F}+AB\bar{C}D\bar{E}+A\bar{C}D\bar{E}\bar{F}+ABD\bar{E}\bar{F}$$

Or in factored form:

$$T = AD\left[\bar{B}C(E+\bar{F})+B\bar{C}(E+F)\right]+A(B+\bar{C})(\bar{D}\bar{F}+D\bar{E}\bar{F})+AC(\bar{B}\bar{D}\bar{F}+DE\bar{F})$$
$$D\bar{E}\bar{F}(A+C)$$



MAP OF A SIX VARIABLES FUNCTION

Minimization of a Function representing the Prime Binary Numbers of the first six digits using Karnaugh Maps.

$$T = \sum (0,1,2,3,5,7,11,13,17,19,23,29,31,41,43,47)$$

$$T = \bar{C}\bar{D}\bar{E}\bar{F}+A\bar{D}\bar{E}\bar{F}+A\bar{C}\bar{D}\bar{F}+AB\bar{D}\bar{F}+A\bar{B}CD\bar{F}+AB\bar{C}D\bar{E}+A\bar{C}D\bar{E}F+ABD\bar{E}F$$

$$T = AD\left[\bar{B}C(E+\bar{F})+B\bar{C}(E+F)\right]+A(B+\bar{C})(\bar{D}\bar{F}+D\bar{E}F)+AC(\bar{B}\bar{D}F+DE\bar{F}) + \bar{D}\bar{E}\bar{F}(A+\bar{C})$$

Figure 2.3

Factoring with Karnaugh Maps

The two level minimum product of sums of a Boolean function is far from being best suited for implementation with standard solid state blocks. Factoring of common terms, whenever possible, can result in a function requiring less hardware and interconnections, but increases the number of levels a signal may pass and therefore the switching time. General methods for factoring are inapplicable and each solution must be checked; sometimes redundant terms must be introduced to obtain a better result. The factoring is obvious when a variable is common to more than one term. Examples:

A minimum sum of products can be written

$T = A\bar{B}\bar{C}+A\bar{B}C+AD$ to $T = A.(B\bar{C}+\bar{B}C+D)$

or a minimum product of sums

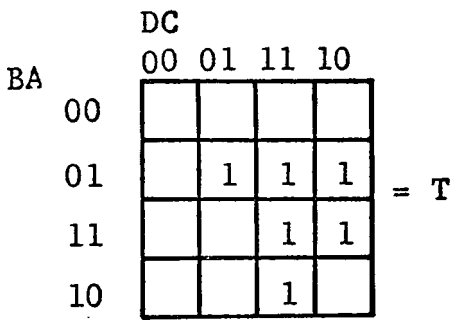
$T = (A+B+CD)(A+\bar{B}+\bar{D})$ to $T = A+(B+CD)(\bar{B}+\bar{D})$

The Karnaugh maps are good aids for factoring. One method selects two functions which, when the outputs either logically AND or OR together produce the original function. The example in figure 2.4 results in the product of a sum of products.

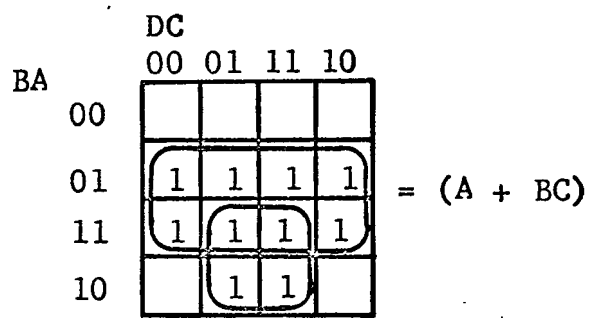
$T = A\bar{B}C + AD + BCD$

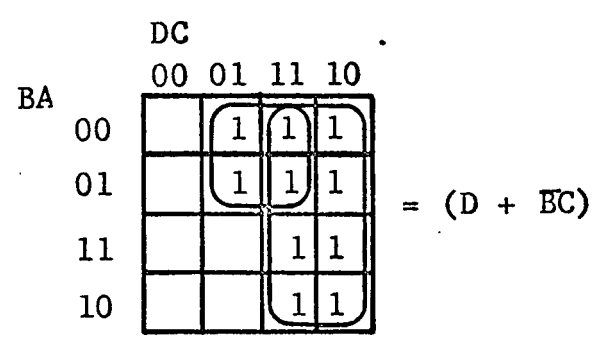
$T = (1^{st}\ \text{Map}).\ (2^{nd}\ \text{Map})$

$T = (A + \bar{B}C)\ (D + BC)$



$$T = A\bar{B}C + AD + BCD$$





$$T = (A + BC)(D + \bar{B}C)$$

FACTORING BY AN "AND" OF TWO MAPS

Figure 2.4

There are no fixed rules regarding the placing of 1's and 0's, however, large groups of 1's that can be expressed with one or two variables will minimize the function.

Inhibiting in Karnaugh Maps

This method utilizes considerable redundancy wherever advantageous and also applies the full logic power of the NAND/NAND or the NOR/NOR functions, unavailable in previous transform methods.

The method is simple. Where a group of 1's is upset by a few 0's, it may be resolved by inhibiting the 0's to achieve the 1's group. Figure 2.5 shows that the function may be implemented, as A and not ABC or, C and not ABC; for example,

$$T = A.\ (ABC)'+C.\ (ABC)'$$

implemented with four NAND gates. Algebraic factoring and simplification of the function, without inhibiting, results in:

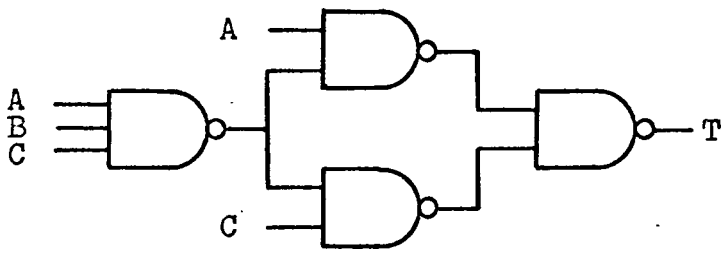
$$T = \overline{A}C + A\overline{B} + A\overline{C}$$

$$T = \overline{A}C + A.\ (\overline{B} + \overline{C})$$

thereby requiring five NAND gates. This method of taking a loop of 1's (including the bothersome 0's) and intersecting with the complement of the looping produces the form "product term times the complement of a product", expressible by a NAND/NAND function.

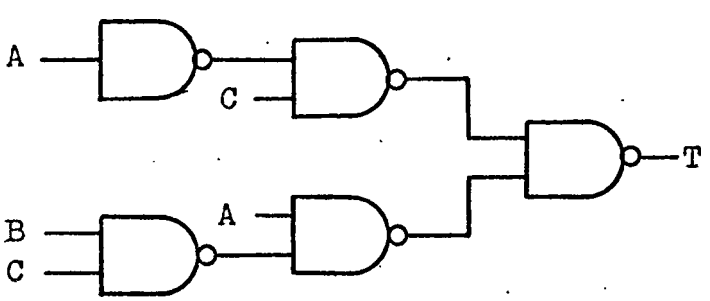
C \ AB	00	01	11	10
0	0	0	1	1
1	1	1	1	1

$$T = A.(\overline{ABC}) + C.(\overline{ABC})$$



C \ AB	00	01	11	10
0	0	0	1	1
1	1	1	0	1

$$T = \overline{A}C + A\overline{B} + A\overline{C} = \overline{A}C + A.(\overline{B} + \overline{C})$$



KARNAUGH MAP INHIBITING

Figure 2.5

Map to Hardware Implementation

Any two level function can be expressed in eight different forms, each tending to directly fit a particular mode of logic. Consider the three-variables Karnaugh map in figure 2.1. From the 1 cells the minimum sum of products can be written:

$$T = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$$

Where the function is double primed and one prime moved in, applying De Morgan's theorem results in:

$$T = (\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C})''$$

$$T = \left[(\bar{A}\bar{B}\bar{C})'.\ (\bar{A}\bar{B}C)'.\ (A\bar{B}C)'.\ (AB\bar{C})'\right]'$$

a NAND/NAND expression of the function. Move in the primes around the terms and the function for OR/NAND implementation is obtained:

$$T = \left[(A+B+C)(A+B+\bar{C})(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)\right]'$$

A NOR/OR function results, if the overall prime is moved in:

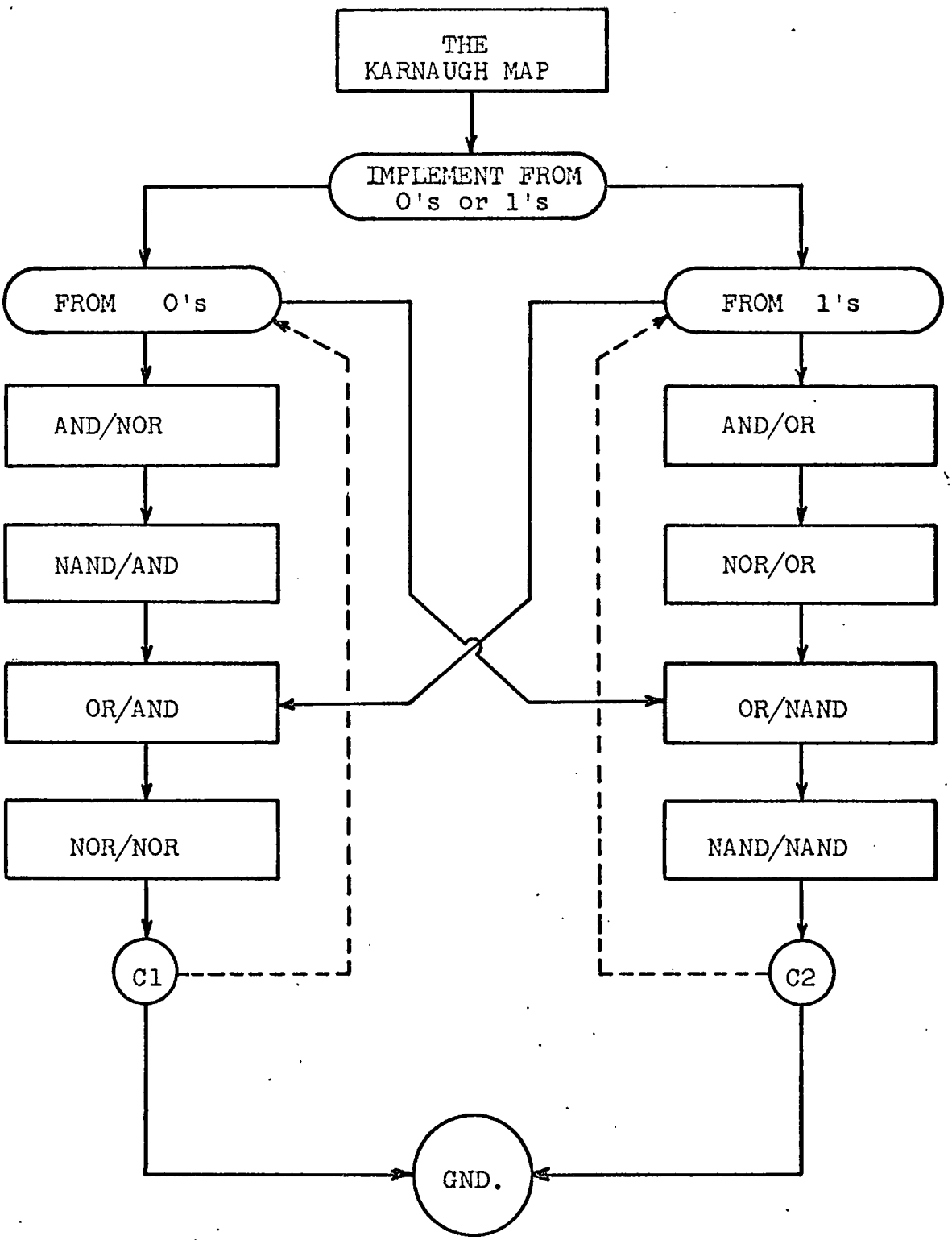
$$T = (A+B+C)' + (A+B+\bar{C})' + (\bar{A}+B+\bar{C})' + (\bar{A}+\bar{B}+C)'$$

In the last transformation if the primes of the terms are moved in, the original function is obtained:

$$T = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$$

In considering the 0's the function may be initially expressed as NOR/OR

$$T = (\bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC)'$$



FUNCTION TRANSFORMATION FLOW DIAGRAM

Figure 2.6

and by moving-in the overall prime, a NAND/AND expression is seen:

$$T = (\bar{A}B\bar{C})'.\ (\bar{A}BC)'.\ (A\bar{B}\bar{C})'.\ (ABC)'$$

Moving-in the prime of the term, the minimum product of sums is now identified:

$$T = (A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+\bar{C})$$

To obtain the original function, the function is double primed and one prime is moved in. It is possible to repeat the same cycle starting with the minimum product of sums, although there will be different starting points. A flow diagram of function transformation starting from Karnaugh Map and moving through each step is illustrated in figure 2.6.

These examples prove that a two level function may be expressed in eight different ways, resulting in the practical value that each one suits its particular mode of logic.

CHAPTER III

NAND-NOR IMPLEMENTATION

One possible choice of single functional block implementation is the NAND/NAND function which sometimes is referred to as the "Stroke Function". As seen before, a NAND is a logical block whose output is a 0 whenever all three inputs are 1.

Another choice for single functional block implementation is the dual of the NAND/NAND expression, the NOR/NOR, sometimes referred to as the "Dagger Function".

Duality provides similarity of implementation techniques, for it is proven that by complementing the variables and the function of either will result in the other function, similar to a change from a positive to a negative logic.

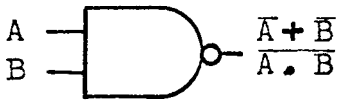
$$A \downarrow B = \overline{\overline{A}.\overline{B}} = \overline{A.B} = \overline{A} + \overline{B}$$

$$A \uparrow B = \overline{\overline{A}+\overline{B}} = \overline{A+B} = \overline{A}.\overline{B}$$

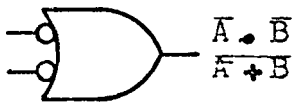
Without an established method even the experienced logic designer will have difficulty recognizing the minimality of the circuits. Because most of the digital integrated circuits are NAND or NOR gates the functions may be directly implemented to hardware. (In this example fan-in and fan-out have been restricted to four and eight).

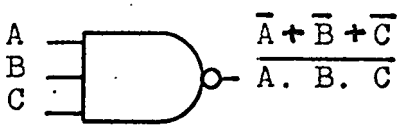
Four modes of expression transformation will be discussed:

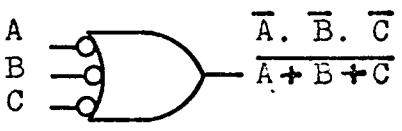
AB	$\overline{A+B}$
00	1
01	1
10	1
11	0



AB	$\overline{A \cdot B}$
00	1
01	0
10	0
11	0

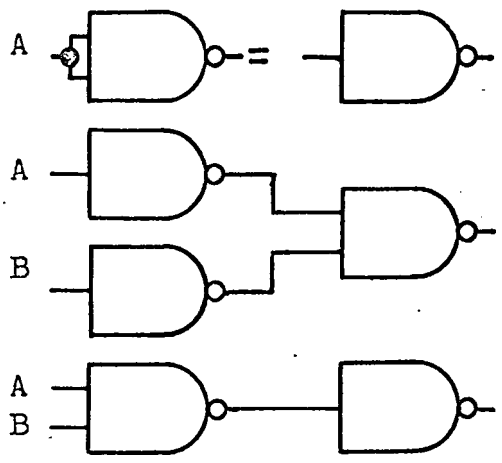






THE NAND FUNCTION

THE NOR FUNCTION

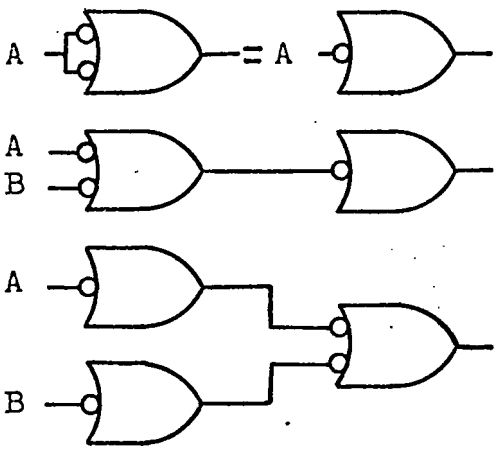


$\overline{A} + \overline{A} = \overline{A}$ (NEGATION)

$\overline{\overline{A}} + \overline{\overline{B}} = A + B$ (OR)

$\overline{\overline{A} + \overline{B}} = A \cdot B$ (AND)

NAND FUNCTION EQUIVALENTS OF AND, OR, AND NEGATION



$\overline{A} \cdot \overline{A} = \overline{A}$ (NEGATION)

$\overline{\overline{A} \cdot \overline{B}} = A + B$ (OR)

$\overline{\overline{A}} \cdot \overline{\overline{B}} = A \cdot B$ (AND)

NOR FUNCTION EQUIVALENTS OF AND, OR, AND NEGATION

Figure 3.1

1. Equivalent AND-OR substitution.
2. Transformation by De Morgan theorem. [5]
3. Earle transformation method. [6]
4. Direct implementation from graphical representation.

Equivalent AND-OR Substitution

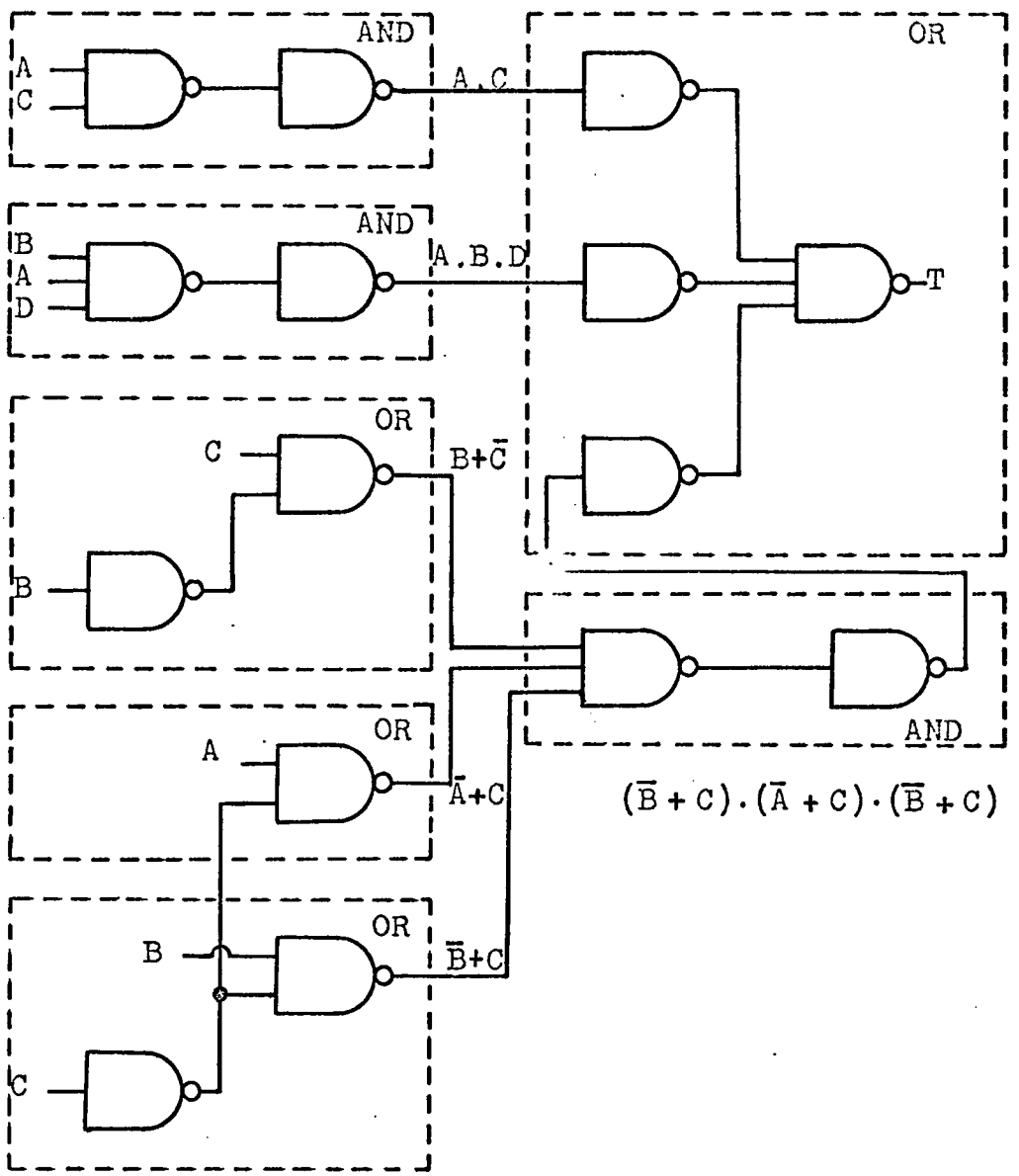
The simplest and possibly least used method is the generation of AND, OR and Negation Function Blocks using NAND or NOR, illustrated in figure 3.1, directly implemented into the AND-OR expression. Figure 3.2 demonstrates a circuit using this method to implement the function T, using NAND blocks.

$$T = AB + ACD + (B+\overline{C}).(\overline{B}+C).(\overline{A}+B)$$

It is evident that simplification may be achieved by eliminating two inverters in series as $\overline{\overline{A}} = A$ resulting in the circuit expressed in figure 3.3; however, a minimal circuit is not produced. Nine NAND gates appear rather than seven gates in the minimal circuit. If complemented inputs are available then

$$T = (A+\overline{B})' + (\overline{A}+\overline{C}+\overline{D})' + (A+B+C)' + (\overline{B}+\overline{C})'$$

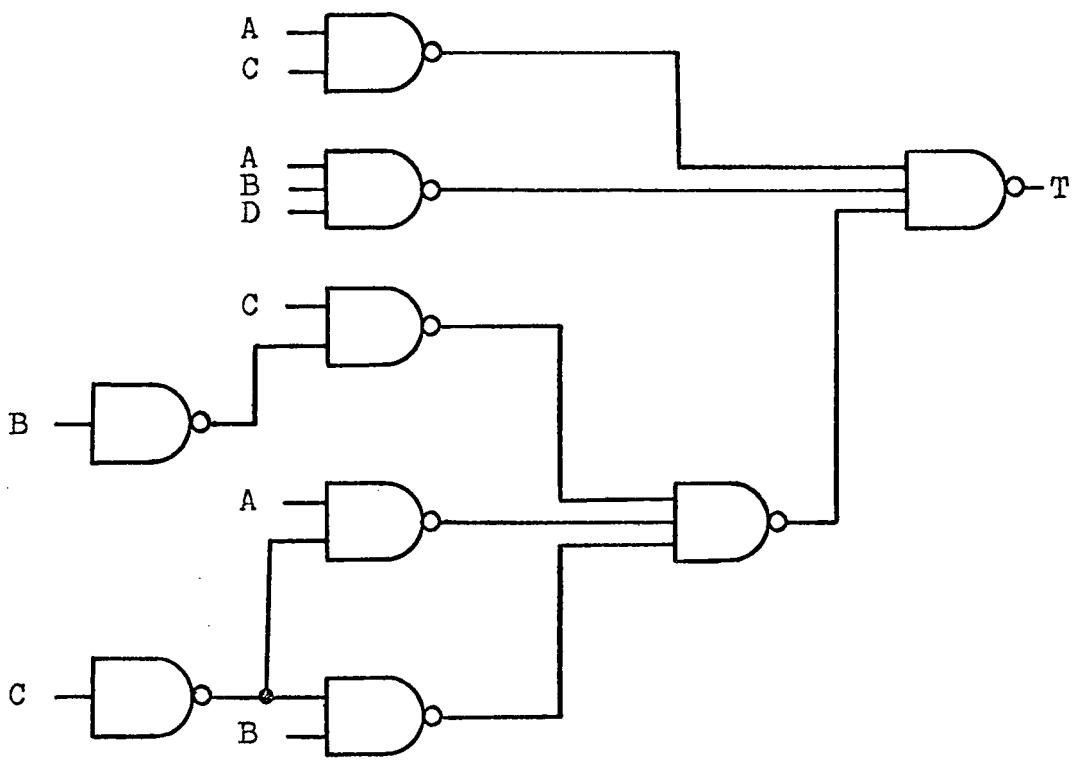
resulting in only five gates. The analyst will recognize that minimal circuits could be achieved by extending the rules. Further discussion will show that this would result in effective conversion to another method and would not then be a true equivalent AND-OR substitution as presented.



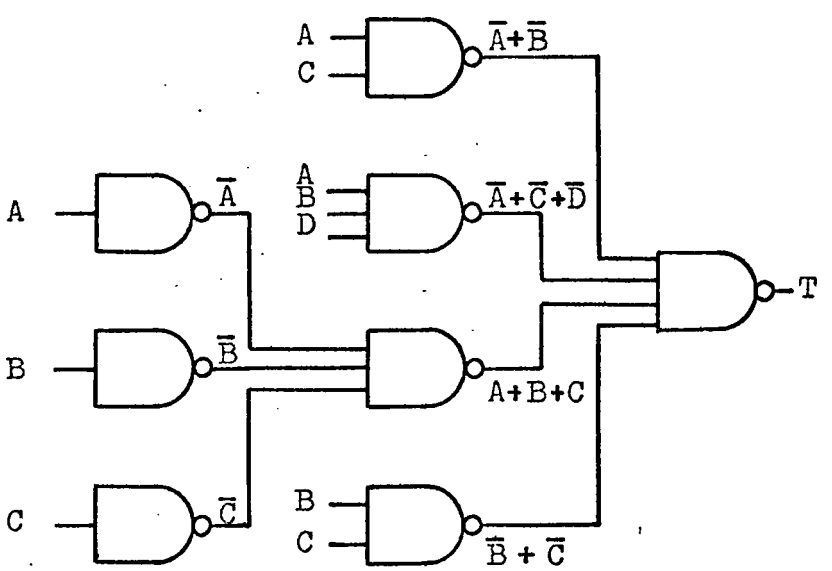
$$T = AC + ABD + (B+\bar{C}) \cdot (\bar{A}+C) \cdot (\bar{B}+C)$$

EQUIVALENT AND OR CIRCUIT

Figure 3.2



REDUCED EQUIVALENT AND OR CIRCUIT



MINIMAL CIRCUIT

Figure 3.3

The Transformation by De Morgan Theorem

This methodical and algorithmic approach to synthesis is a valuable aid to the logic designer in solving practical problems. The fact that a single output exists determines a NAND at the end of the gating and the De Morgan theorem is applied by working backwards from the end of the gating. Typically, to implement with NORS:

$$T = (A + \overline{C}).(B + D)$$

at the input of the NOR, (see figure 3.4)

$$T' = \overline{A}\ C + \overline{B}\ \overline{D}$$

where $\overline{A}\ C$ and $\overline{B}\ \overline{D}$ are the inputs. At the next level of NORS the inputs are:

$$P_1' = (\overline{AC})' = A + \overline{C}$$

$$P_2' = (\overline{B}\overline{D})' = B + D$$

where A and C are inputs of one gate and B and D of the other. If C is not available, an additional NOR gate is required to complement C to $\overline{C}$.

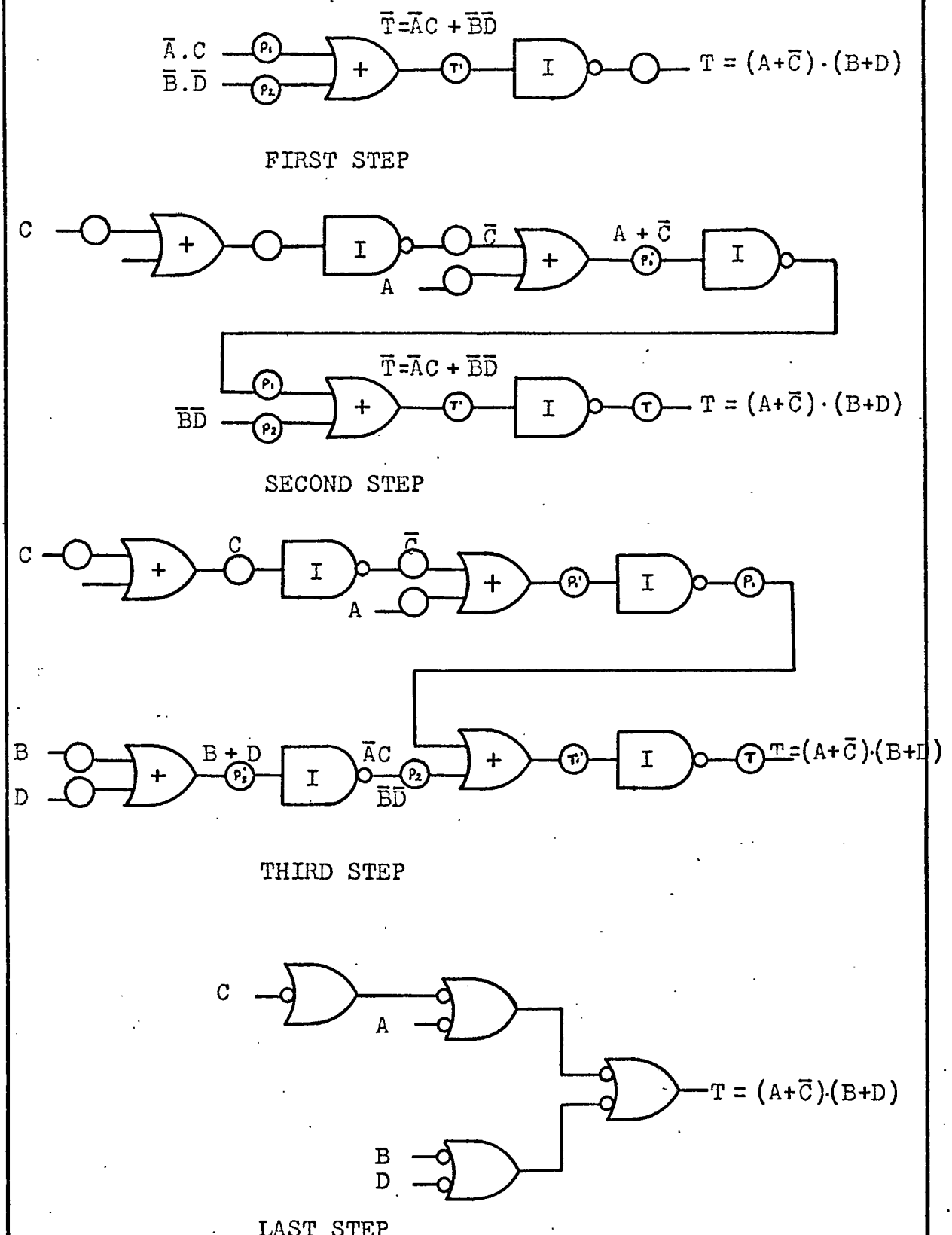
If the function

$$T = AC + \overline{B}C$$

is implemented with NOR gates the result is

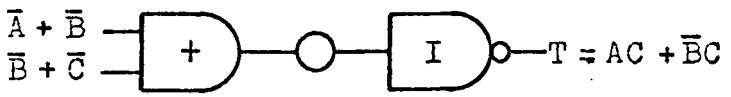
$$T' = (\overline{A}+\overline{C})(B+\overline{C})$$

Figure 3.5 clearly shows the conflict of the existence of an AND function at T' instead of an OR function. A solution would be the conversion of the equation to an OR-AND form

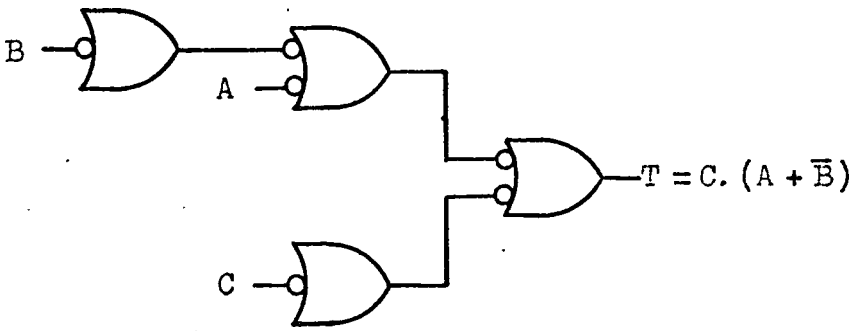


IMPLEMENTING NI OR-INVERTERS WITH DE MORGANS THEOREM

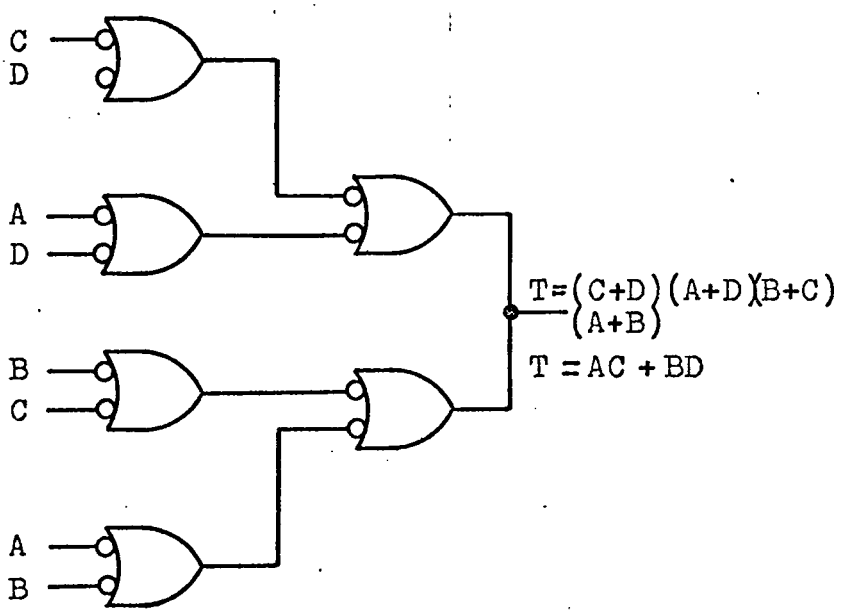
Figure 3.4



CONFLICT ILLUSTRATED



CONFLICT RESOLVED



CONFLICT RESOLVED BY TAKING COMPLEMENT FUNCTION

CONVERSION TO OR-AND FORM

Figure 3.5

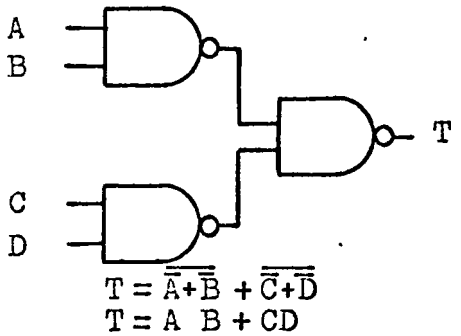
before starting. A potential pitfall exists in using some of the theorems converting this expression to OR-AND form, generally resulting in loss of minimality. The preferred transformation is from the original Karnaugh map which will yield the minimal OR-AND expression. Changing the function to its dual form is not economical in every case. Infrequently, complementing the output when the complement of the input is available will be more desirable.

Earle Transform Method

The preceding methods are slow and the total problem concept is obscured by limiting concentration to a part at a time; whereas the transform method permits rapid analysis and implementation, as in AND-OR logic, while exhibiting the overall problem neatly.

Rules for implementing:

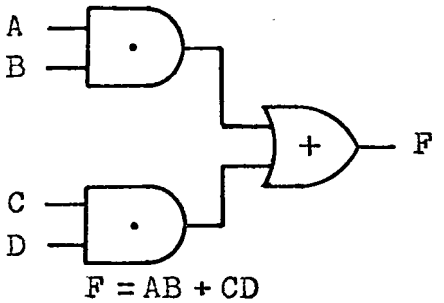
1.
   a) For NAND--Factor the Boolean equation to a form where output is an OR (OR-AND-OR etc.). Odd levels complemented variables and even levels uncomplemented variables are desired objectives.
   b) For NOR--Factor to output AND (AND-OR-AND etc.) achieving the desired levels objective.



$T = \overline{\bar{A}+\bar{B}} + \overline{\bar{C}+\bar{D}}$

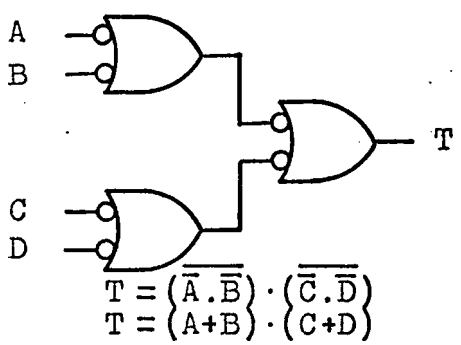
$T = A\ B + CD$

TERMINAL ANALYSIS
NAND/NAND FUNCTION



$F = AB + CD$

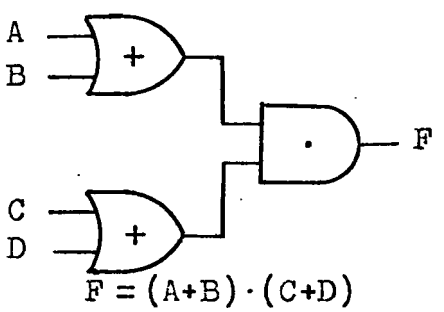
EQUIVALENT OF
NAND/NAND FUNCTION



$T = \{\overline{A}.\overline{B}\} \cdot \{\overline{\bar{C}.\bar{D}}\}$

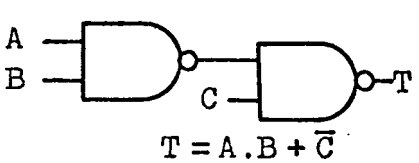
$T = \{A+B\} \cdot \{C+D\}$

TERMINAL ANALYSIS
NOR/NOR FUNCTION



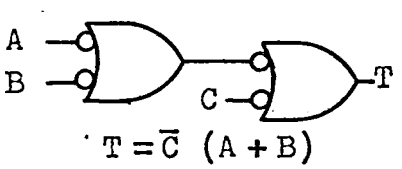
$F = (A+B) \cdot (C+D)$

EQUIVALENT OF
NOR/NOR FUNCTION



$T = A.B + \overline{C}$

ODD LEVEL GATING
NAND/NAND FUNCTION

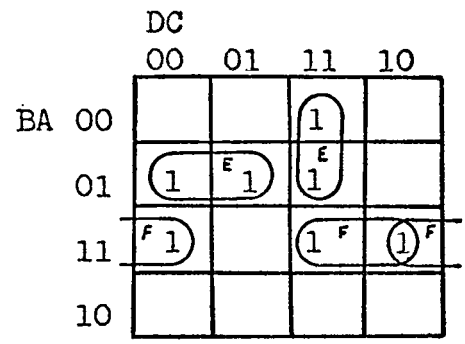


$T = \overline{C}\ (A + B)$

ODD LEVEL GATING
NOR/NOR FUNCTION

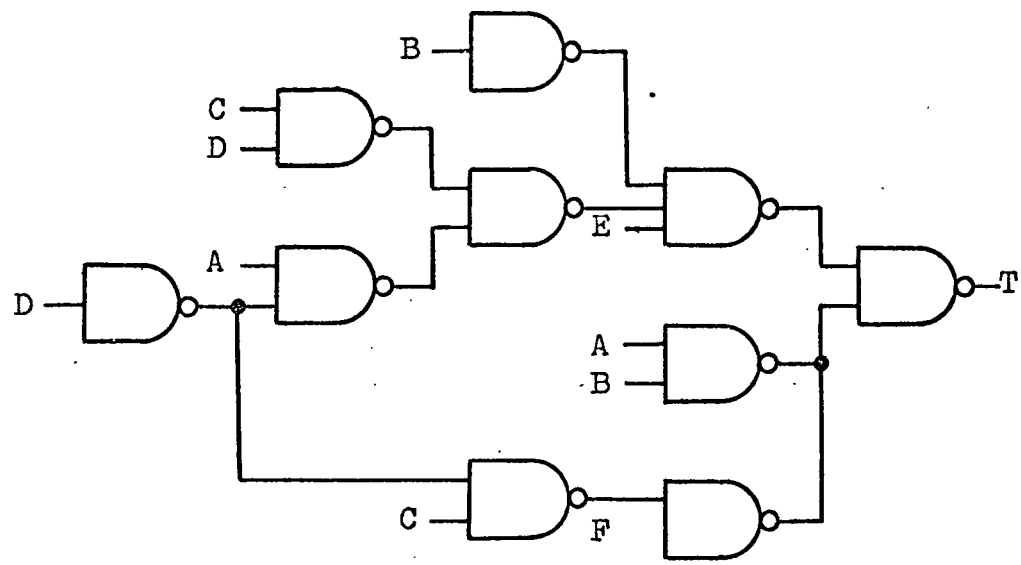
EARLE TRANSFORMS

Figure 3.6



E and F

$$T = A\bar{B}\bar{D}E + AB\bar{C}F + ABDF + \bar{B}CDE$$



$$T = ABF.(\bar{C}+D) + \bar{B}E.(A\bar{D} + CD)$$

FACTORING EARLE TRANSFORMS

Figure 3.7

2. Lay out the gating from the equations in AND or OR format rather than NAND and NOR except that NAND and NOR variables coming in at odd levels of gating are complemented, as in figure 3.6.

Use of factoring is suggested when the fan-in of several blocks is exceeded, when complements are unavailable or when signals are overloaded.

For example: To implement T using NAND's (Figure 3.7)

$$T = \bar{A}\bar{B}CDE+A\bar{B}CE+A\bar{B}\bar{C}\bar{D}E+AB\bar{C}F+ABDF$$

The restriction to three input gates results in thirteen NAND gates total, a requirement of two NAND's per term by a direct two levels implementation from the Karnaugh Map. Factoring achieves:

$$T = \bar{B}E.(CD + \bar{A}D) + ABF.\ (\bar{C} + D)$$

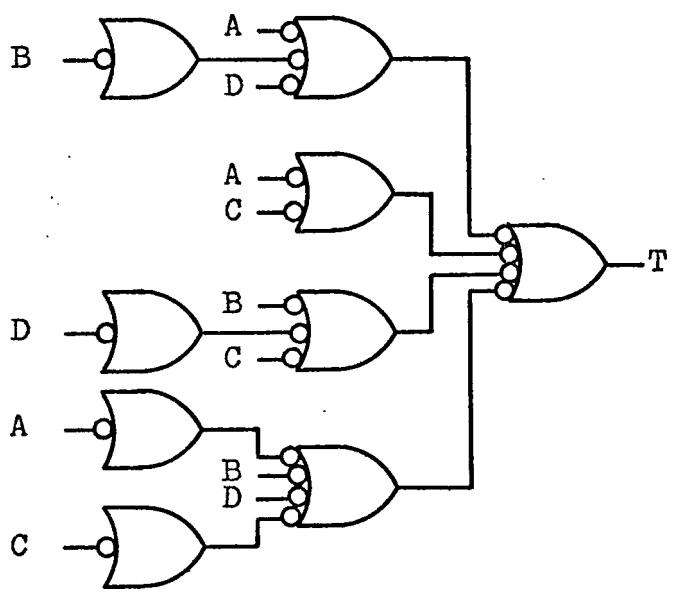
A four level equation in AND-OR-AND-OR form, as required, permitting direct implementation as AND's - OR's with complementing at odd levels. A reduction of three NAND less than the first solution.

Direct Implementation from Maps

Perhaps the most natural approach to NAND/NAND function and the NOR/NOR function is to treat them as functions in their own right rather than force them into the form of AND's and OR's. This renews insight into the fundamental processes of synthesis and proves the minimality

BA \ DC	00	01	11	10
00	1	1	1	1
01	1	ø	0	0
11	1	ø	0	1
10	1	0	1	0

$$T = \left[(\overline{A+\overline{B}+D}) + (\overline{A+C}) + (\overline{B+C+\overline{D}}) + (\overline{A+B+C+D}) \right]'$$



NOR/NOR IMPLEMENTATION FROM MAP

Figure 3.8

of the two previous methods.

1. The NOR/NOR function:

   A Karnaugh map using the NOR/NOR function concludes that:

   a) The NOR function is a 1, when, and only when, the input variables are 0.

   b) The 0's of the Karnaugh map represents the input terms of the function.

   c) To minimize the function the fewest possible terms and literals are selected, as in AND-OR expressions.

   The example in figure 3.8 shows the implementation of a Karnaugh map to a two level NOR/NOR function;

   $$T = \left[(A+\overline{B}+D)'+(A+C)'+(B+C+\overline{D})'+(A+B+C+D)' \right]'$$

   a minimal expression for two levels and checked by successive substitutions.

2. The NAND/NAND function:

   The NAND/NAND function may be implemented in a similar manner:

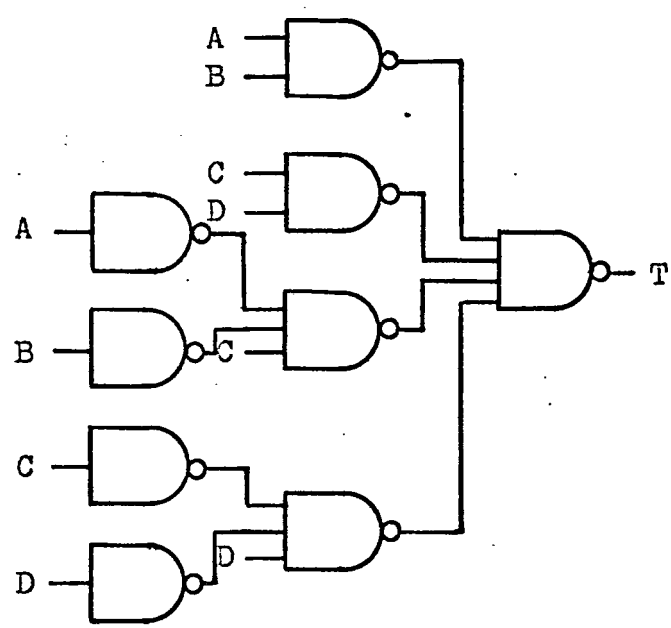
   a) The term is 1 whenever there is a 1 on the map.

   b) Combining the 0's of the map results in a 1 whenever any term is 0.

   c) The complement of the function may be found

BA \ DC	00	01	11	10
00	0	1	1	0
01	1	0	1	0
11	1	1	1	1
10	ø	0	1	ø

$$T = (\overline{AB}.\ \overline{CD}.\ \overline{A\bar{B}C}.\ \overline{A\bar{C}D})'$$



NAND/NAND IMPLEMENTATION FROM MAP

Figure 3.9

by interchanging 1's and 0's on the map. From the map in figure 3.9 and applying the above rules results in:

$$T = \left[(AB)'.\ (CD)'.\ (ABC)'.\ (ACD)' \right]'$$

Previous examples reveal that the NAND/NAND functions and the NOR/NOR functions are related in the same way as the AND and OR functions. Similar interchanges of 1's and 0's, with the functions written from 0's, should give the NOR/NOR function. Since the transformation yields minimal equations with identical map configuration, a two level minimal expression is evident. The previous two examples could be algebraically verified to establish that the equations are identical.

In conclusion, it is proven that from one Karnaugh map, each of the two possible minimal two level equations in the AND, OR, INVERTED function, the NAND/NAND function and the NOR/NOR function may be written. This permits a nearly simultaneous minimization and cost investigation of the circuit logic defacto obtainable by Karnaugh maps.

CHAPTER IV

DECOMPOSITION OF FUNCTIONS

The fundamental concept of switching theory is the possibility of expressing any logic circuit as a Boolean function. The algebraic properties and postulates provided the foundations for the graphical Karnaugh method for minimization employed in the previous chapters.

The question arises whether these algebraic properties are unique. Examination of the general theory of functions reveals another basic property which should be considered, that of decomposability, that is, the way it can be decomposed into a set of simpler functions. In logic circuits, the prime implicants may be seen as the basic set of functions forming a minimum sum representation. Further, a Boolean function implementing a multiple-level logical circuit may be considered as being decomposed into a set of simpler functions.

After extensive research at Harvard University, based on the above properties of Boolean functions, R.L. Ashenhurst presented his paper "The Decomposition of Switching Functions" in a Bell Laboratory report.[7] The Ashenhurst theory and its application to minimization of logical integrated circuit blocks is the subject of this chapter.

Disjunctive Decomposition

It is interesting to examine the conditions in which a Boolean function may be expressed with one or more subfunctions. Given the Boolean function f with m + p variables: $x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_p$. The function is said to be decomposable where it can be represented in the form: $f = (x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_p) = F \left[x_1, x_2, \ldots, x_m, \phi (y_1, y_2, \ldots, y_p)\right]$.

It appears obvious that any Boolean function of n variables may be written in this form, where ϕ will be a one variable function: $f (x_1, x_2, \ldots, x_p) = F \left[x_1, x_2, \ldots, x_n\text{-}1, \phi (x_n)\right]$. For any sequence of configurations of the variables a function is said to be disjunctive decomposable if at least one subset is different from the preceding trivial case.

A convenient abbreviation of the above notation provides that a, b, c, ... represents the subsets of the variables $x_1, x_2, \ldots, x_n$ with the only stipulation being that the subsets are mutually exclusive. In this condition it may be said that a disjunctive decomposition of f (a, b) in terms of a and ϕ (b) exists and results in:

$$f (a,b) = F \left[a, \phi (b)\right] .$$

The notation of disjunctive decomposition may easily be analytically formulated. In a four variables

Boolean function f (a, b, c, d) where it is desired to determine if a disjunctive decomposition exist of the form: $f(a, b, c, d) = F\left[a, b, \phi(c, d)\right]$.

Any function of four variables may be expressed in the partially factored form, where capital A, B, C, D represents any subsets of the two variables c and d.

$f(a,b,c,d) = a.b.A(c,d) + a.b.B(c,d) + a.b.C(c,d) + a.b.D(c,d)$.

These subsets are uniquely determining f. However, where A,B,C and D can be expressed in terms of some single function $\phi(c,d)$, that is, if they assume only values chosen from the set $\phi(c,d)$, $\phi'(c,d)$, 0 and 1, there then exists disjunctive decomposition, being the necessary as well as sufficient condition. Example:

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

$T = \bar{a}\bar{b}(\bar{c}\bar{d} + cd) + ab(\bar{c}d + c\bar{d})$, where

$\bar{c}\bar{d} + cd = (\bar{c}d + c\bar{d})'$

$T = \bar{a}\bar{b}\,\phi(c,d) + ab\,\phi'(c,d)$, where $\phi = \bar{c}\bar{d} + cd$.

In this case $A(c,d) = \phi$, $B(c,d) = 0$, $C(c,d) = 0$ and $D = \phi'(c,d)$ by the general partial factored form and is disjunctive decomposable having only one function (ϕ) its complement (ϕ') and 0.

The algebraic method followed in this example may always be applied to any function. To determine a decomposition, the partial factored form expression for all the possible partionings of the variables requires examination of the literals to detect disjunctive decomposition.

	c'd' 0 0	c'd 0 1	c d' 1 0	c d 1 1	
a'b' 00	P0 (1)	P1 0	P2 0	P3 (1)	A
a'b 01	P4 0	P5 0	P6 0	P7 0	B
a b' 10	P8 0	P9 0	P10 0	P11 0	C
a b 11	P12 0	P13 (1)	P14 (1)	P15 0	D

$$T = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + AB\bar{C}D + ABC\bar{D}$$

PARTITION MATRIX OF EXAMPLE FUNCTION

Figure 4.1

The Partition Matrix

Fortunately, this tedious algebraic manipulation is replacible by simple graphics. Visual inspection of a set of partition matrices may reveal disjunctive decomposition.

Figure 4.1 being an output of the computer program "BOLEQ.", used in the minimization procedure, shows no need for reflected ordering to detect decomposition as in the Karnaugh map. The columns and rows are arranged in straight binary to facilitate implementation (ordering is really of no consequence). Each product is represented by a cell in which 0 or a 1 represents the output and a P with a subscript, identifies the binary variables in dyadic form. For example, P 10, is the product term a $\bar{b}$ c $\bar{d}$. The P_i are arranged in the map so that each appears in a column headed by the combination of c and d that it contains, and in a row headed by the combination of a and b that it contains. The first row therefore has the combination with the terms $\bar{a}\bar{b}$ A (c,d) in the equation of partially factored form, the second row contains $\bar{a}b$ B (c,d) and so forth. By employing the function of the previous example, the map usage may be illustrated.

The standard sum of products form of the above function is as follows:

$$T = \sum (0,3,11,14)$$

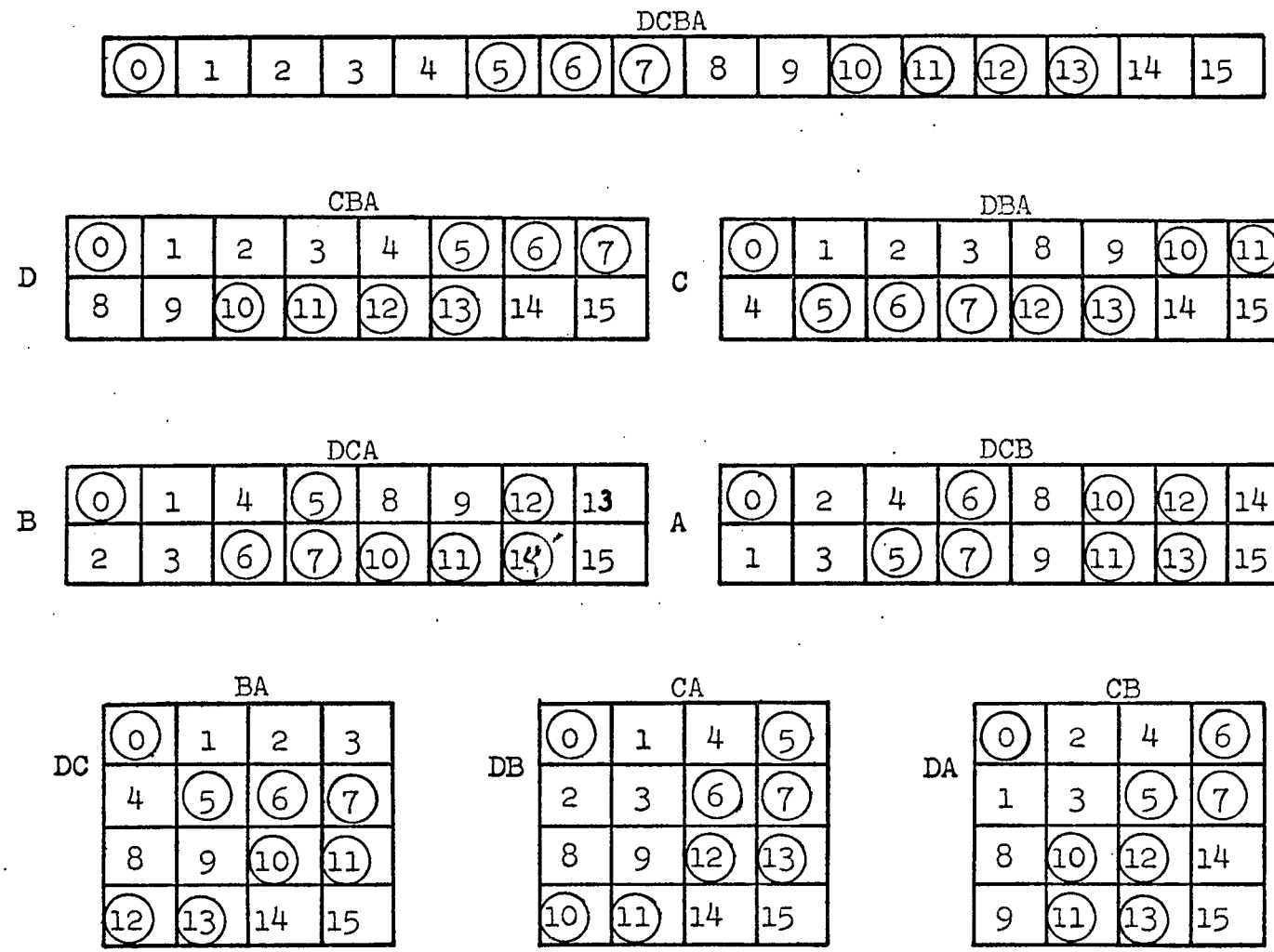


Figure 4.2

DECOMPOSITION CHART FOR THE FUNCTION T = (0,5,6,7,10,11,12,13)

This function has the value 1 for the combinations P_0, P_7, P_9 and P_{14}, and the value 0 for the rest of the sixteen combinations. Accordingly, there is a 1 or 0 in each block. The 1's in the first row correspond to the terms of A (c,d) and those in the second, third and fourth row are B (c,d), C (c,d) and D (c,d) respectively. The fact that the fourth row has the complement of 1's and 0's of the first row, results in D (c,d) = A' (c,d). The fact that the second and third rows do not contain a 1 implies that B (c,d) = C (c,d) = 0; if all the combinations in a row were 1's, it will imply that the function associated with the row is equal to 1.

A given configuration of 1's and 0's in a row is called a Pattern. A row where the 1's and 0's are the complement of the pattern is called the Inverse of P.. Rows of all 1's or all 0's are called Trivial.

Using the above terminology a function is disjunctively decomposable in the term ab and ϕ (c,d):

> " Should a pattern P appear in a row, then a decomposition of this form exist and only if every other row exhibits the pattern P, or its inverse, or one of the trivial patterns".

Where the numbers of 1's and 0's are equal, pattern and inverse selections are totally arbitrary and can be reversed without detriment.

Corresponding maps should be examined to determine

whether decomposition of terms of different partition of variables exists. Examination of seven maps row-wise and column-wise will determine complete function decomposability for a four variable function. (See figure 4.2).

Minimization by Decomposition

The characteristic of a disjunctively decomposed function is the possibility of realizing a minimum logical network. Generally, the circuit realized by a decomposition utilizes fewer logical blocks than circuits derived by the methods of the previous Chapters. For example the function $T = \sum (0,5,6,7,10,11,12,13)$ of four variables takes the unity value for all the combinations of the variables 0,5,6,7,10,11,12 and 13, and the zero value for all other. From the Karnaugh map in figure 4.3 the function may be minimized to a minimum sum of products:

$$T = \bar{A}B\bar{C}\bar{D} + AC\bar{D} + BC\bar{D} + B\bar{C}D + \bar{B}CD$$

$$T = \bar{A}B\bar{C}\bar{D} + \bar{B}CD + B\bar{C}D + C\bar{D}. \quad (A+B)$$

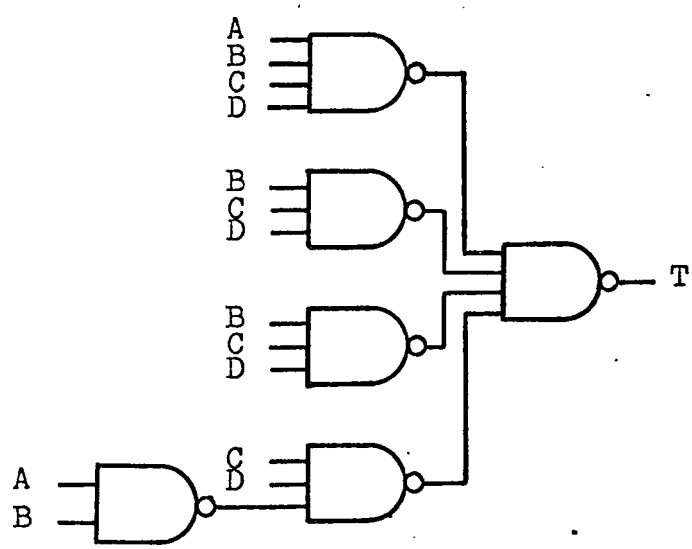
The corresponding logic network for this function requires six "NAND" circuits and twenty-two interconnections. The same function is also disjunctive and decomposable in terms of c and ϕ (a,b,d). This is seen from the decomposition chart in figure 4.2. The form of subfunctions can be read directly from the chart c - dba:

$$T(a,b,c,d) = F\left[c\ \phi\ (a,b,d)\right]$$

BA \ DC	00	01	11	10
00	(1)	0	1	0
01	0	1	1	0
11	0	1	0	1
10	0	1	0	1

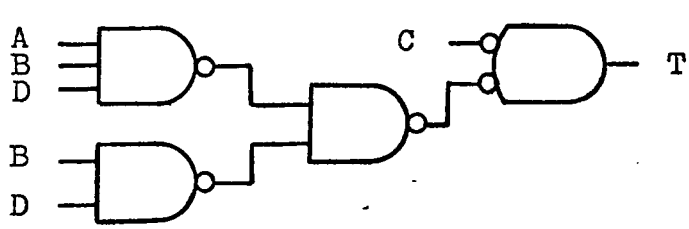
$$T = \bar{A}\bar{B}\bar{C}\bar{D} + B\bar{C}\bar{D} + A\bar{C}\bar{D} + B\bar{C}D + \bar{B}CD$$

MINIMUM SUM FROM KARNAUGH MAP



$$T = C\bar{D}\cdot(A+B) + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{B}CD + B\bar{C}D$$

LOGIC NETWORK DERIVED FROM MINIMUM SUM



$$T = (\bar{A}\bar{B}\bar{D} + BD) \oplus C$$

LOGIC NETWORK REALIZATION BY DECOMPOSITION

Figure 4.3

$$T = \bar{c}.\ \phi\ +\ c.\ \bar{\phi}, \text{ where}$$

$$\phi = \bar{A}\bar{B}\bar{D} + \bar{A}BD + ABD \text{ and simplified to}$$

$$\phi = \bar{A}\bar{B}\bar{D} + BD$$

$$T = \bar{C}.\ (\bar{A}\bar{B}\bar{D}+BD) + C.\ (\bar{A}\bar{B}\bar{D}+BD)'$$

The total implementation of the decomposed function requires only four logical blocks (three NAND's and one EXOR), a reduction in hardware by one third compared to the previous logical network. The number of interconnections (ten) represents a reduction of 55 per cent. To simplify the minimum sum of products to four gates, it would be necessary to recognize that, $\bar{A}\bar{B}\bar{D} + BD$ is the complement of $A\bar{D} + B\bar{D} + \bar{B}D$, which is most unlikely to be found by simple algebraic simplification.

The minimization procedure outlined in Chapter I includes the search for disjunctive decompositions. The time required for inspection of the partition matrix is only a few seconds and short for the minimality obtained. A disadvantage in the use of the disjunctive function is the increased propagation time, but this is the case in factoring a minimum sum of products also.

CHAPTER V

THE COMPUTER MINIMIZATION PROGRAM

The majority of the Boolean function minimization work is performed by a digital computer, thereby relieving the logic designer from tedious operation, but insuring a minimized equivalent function capable of direct implementation into integrated circuit blocks.

It is not necessary that the program user have knowledge of computer programming, as complete information will be presented in this chapter to plan input, output and execution. A section describing the exact methods employed, together with the flow chart and the MAD program [8] is provided for the logic designer familiar with programming and interested in the algorithm.

Purpose

For a given Boolean Function the subroutine "BOOLEQ" will compute and print.

1. The Truth Table of all possible combinations.
2. The Canonical expressions in the dyadic form.
3. A comment defining which Canonical form has less terms and their number.
4. The number of Karnaugh Maps required.
5. The Partition Matrix.

6. The comparison of input unit hardware function and comment equivalent or error definition.

Input

The Boolean functions to be minimized are punched according to the following card format. (See figure 5.1).

1. Columns 1 to 10 are left blank, spaces are not relevant.
2. Column 11 is used to designate continuation of the Boolean function (0,1,2,...,9).

   The function is interpreted by card order sequence and is not affected by the order of the digit; however any one function may not exceed ten cards.
3. Columns 12 to 72 are for the function statement which may start or finish within these limits and spaces may be inserted without relevance.
4. Columns 73 to 80 are not used by the computer and may be well employed as function and project identifier.

Expressing the Function

The MAD language and subroutine require definite rules to express the Boolean functions.

1. The letters for the input variables are A,B,C, D,E,F.
2. The input constants can be only 0 B for false statements and 1 B for true.
3. The Boolean operators available are: .NOT., .OR., .AND., .THEN., .EXOR. and .EQV. corresponding to the symbol: -, +, ., 0, v, and =. See appendix for table of symbols.
4. The letter T followed by an equal sign (=) starts any function, whereever T has the value binary 0 or 1 representing the function.
5. Parentheses have the same symbol and are used to specify the order of the computation. Redundant parentheses are allowed.
6. The sequence of computation is the same as in Boolean algebra, unless otherwise indicated by parentheses. The order being: .NOT., .AND., .OR., .EXOR., .THEN., .EQV. and =.

To illustrate the function:

$$T = (AB\overline{C} + \overline{B}DC + \overline{A}DC)'$$

will be expressed in MAD as follows:

```
T = .NOT. (A. AND. B. NOT. C. OR. .NOT. B.
    AND. D. AND. C. OR. .NOT. A. AND. D.
    AND. C)
```

For further examples see Input functions to computer program on page 71 .

Output

Maximum effort has been made to present the output data in clear and explicit form. The truth table is printed for any function and for any number of variables up to six. The computer program output shows different truth tables for four, five and six variables. Following, the standard sum of products or the standard product of sums are printed in dyadic form:

SUM = 0, 4, 14, 15

PROD = 8, 9, 12, 14

A comment is printed to show which standard function has fewer terms deciding if implementation should be done from 1's or 0's.

The final output contains a map (or maps) of the function. Each new column, square or map is identified to facilitate direct implementation. (See page 74.). An error comment will be printed upon failure of an error check involving the new minimized function and standard summation.

Execution

The subroutine "BOOLEQ" has been written as an

external function and can be called for execution by any "MAD" program with the following statement:

```
EXECUTE BOOLEQ. (V,I1,I2,I3)
```

The names in parentheses are the dummy arguments, where:

V = the number of variables in the Boolean function.

I1 = is a flag with the value of 0, when subroutine should find equivalence of two Boolean functions, otherwise should be 1.

I2 = is a flag, if 0 suppresses output print of the Boolean function, otherwise should be a 1.

I3 = is a flag to the subroutine, if 0 suppresses the print of Karnaugh maps, otherwise should be a 1.

The Boolean function to be minimized is a separate external function FUN. with the following program statements:

```
EXTERNAL FUNCTION (A,B,C,D,E,F,T)
ENTRY TO FUN.
...
...                    } The Boolean functions
...
FUNCTION RETURN
END OF FUNCTION
```

The usage of external functions provides maximum flexibility, allowing for the subroutine to be incorporated as part of a large program or as a Library Subroutine.

Diagnostic

The subroutine has been checked with many variations of Boolean functions and found to perform as specified. If an error is encountered in the input function due to illegal operation or misspelling, the subroutine will not execute and a comment will be printed identifying the type of error.

The Algorithm

The program uses the same algebraic topological formulation covered in the previous Chapters for the minimization of the switching function. The computation method of this program wi l be described in terms of the flow chart in figure 5.2 and the MAD program shown on page 67. The program contains the following sections:

1. The entry to the subroutine "BOOLEQ" also defines the number of variables (V) and the three flags for suppressing different sections of the program.
2. Normal mode is integer except that Boolean function variables and the Boolean vector (TV)

are in the Boolean mode.

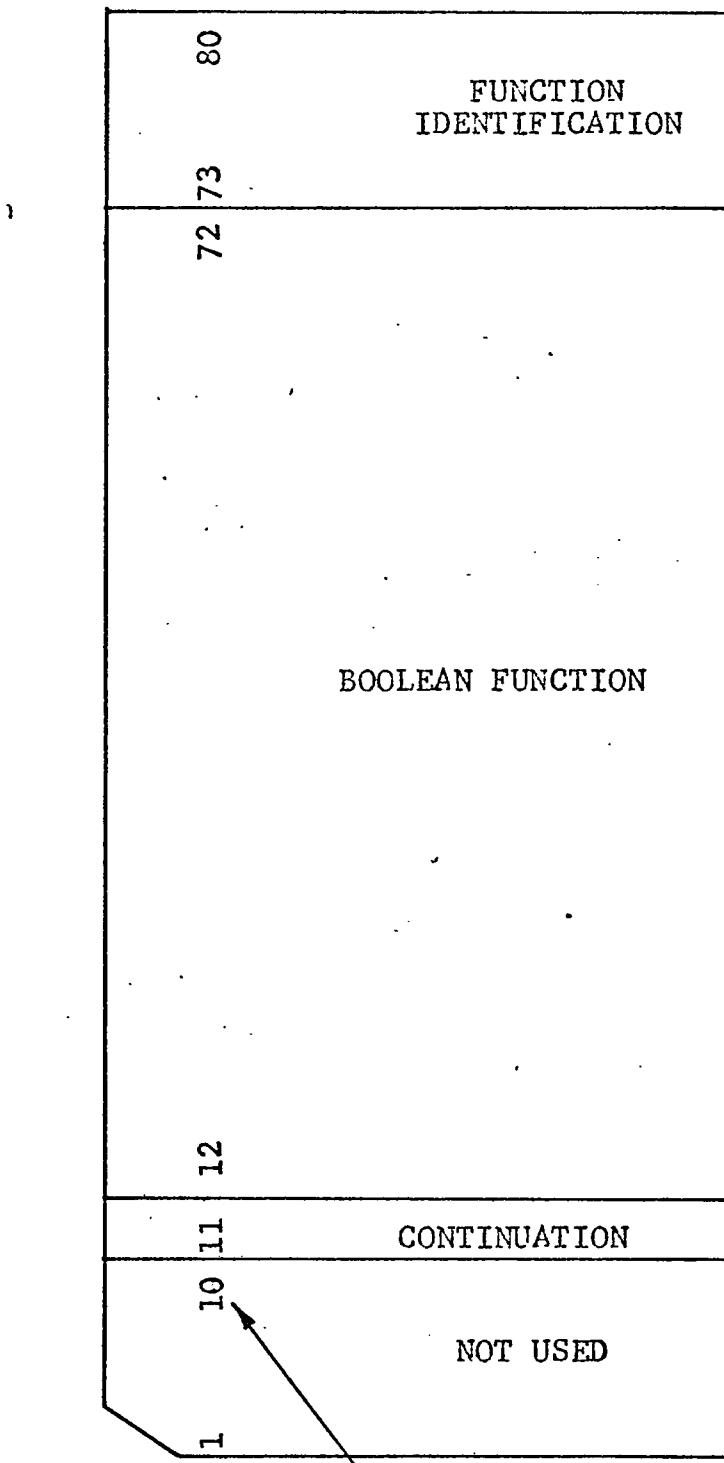
3. There are three vectors: TV, the binary outputs of the input combinations; SUM; and PROD. The latter two vectors being the canonical form of the function.
4. The next group of statements forms and prints the title and header of the Truth Table. A set of conditional statements is incorporated to control the printing of the number of letters.
5. A nested Loop (W) of six iteration statements assign the values of binary 0 and 1 to the six Boolean variables of the function. There are 2^6 possible combinations which the variables can take, giving a maximum of 64 iterations.
6. "FUN" is executed and the value of the Boolean function (T) assumes a new value depending upon the present values of the input variables (A,B,C,D,E,F). The output (T) is transferred to the Boolean vector (TV). The counter I representing the term in dyadic form is incremented by 1.
7. A conditional statement defines the branching, where output (T)= 1 a set of statements introduce a new term in the sum of products. Where the output (T) = 0 the product of the sums is

introduced by a new term. With the addition of the term in the dyadic form the number of term counters M or N are incremented by one.

8. A set of conditional and print statements control the printing of one line from the truth table. Different print format statements are used depending on the number of variables.
9. The number of iterations of the section described in point 6, 7 and 8 are dependent on the number of Boolean variables (V). A set of conditional statements transfers the program out of the loop, when the number of iteration (I) is equal to 2^V.
10. Once out of the loop (W), the canonical form of the function is printed. A conditioned statement defines which expression has fewer terms (M.L.N) and prints a comment of minimality.
11. A loop L1 controls the number of Karnaugh maps to be printed dependant on the number of variables (V).
12. Another loop (L2) nested in L1 iterates and prints one row of a map four times. A set of instructions compute the index for a reflected ordering of terms.
13. The four partition matrices of eight by two are printed. The TV vector is indexed directly.

14. The three partition matrices of four by four are printed, using the same TV vector. Vector indexing is performed directly.
15. The subroutine has now been executed and returns to the Main program.

The program described requires 2,034 memory locations and subroutine execution time is approximately $.2^V$ x .250 m/seconds where V = number of variables.



PUNCH CARD FORMAT

Figure 5.1

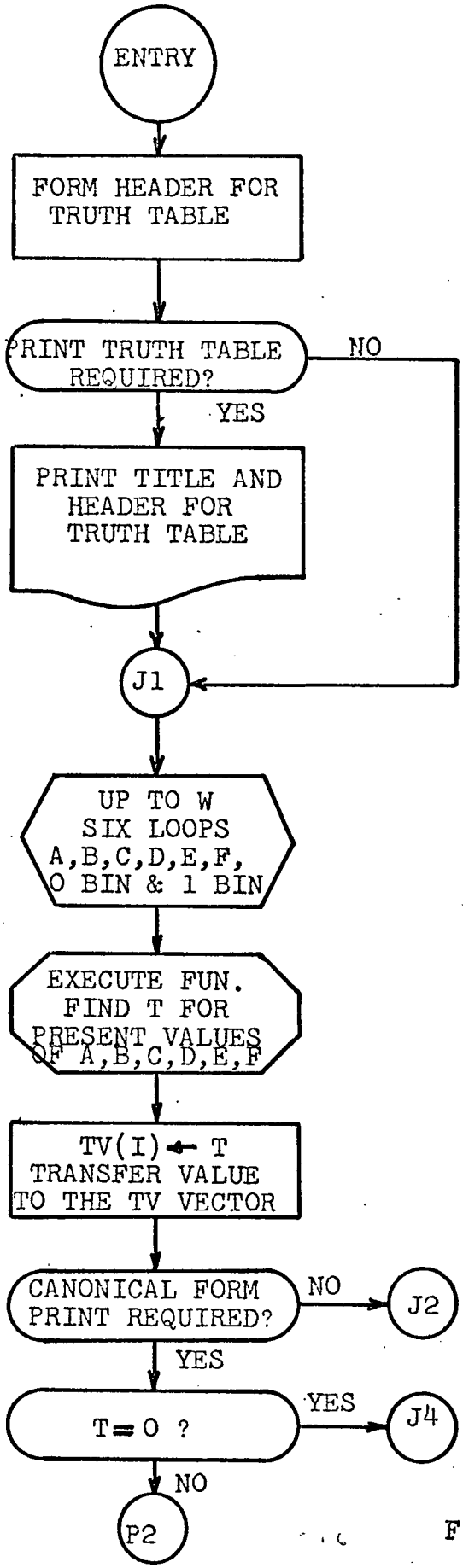

ENTRY
FORM HEADER FOR TRUTH TABLE
PRINT TRUTH TABLE REQUIRED?
NO
YES
PRINT TITLE AND HEADER FOR TRUTH TABLE
J1
UP TO W SIX LOOPS A,B,C,D,E,F, 0 BIN & 1 BIN
EXECUTE FUN. FIND T FOR PRESENT VALUES OF A,B,C,D,E,F
TV(I) ← T TRANSFER VALUE TO THE TV VECTOR
CANONICAL FORM PRINT REQUIRED?
NO
J2
YES
T= 0 ?
YES
J4
NO
P2

Figure 5.2-a

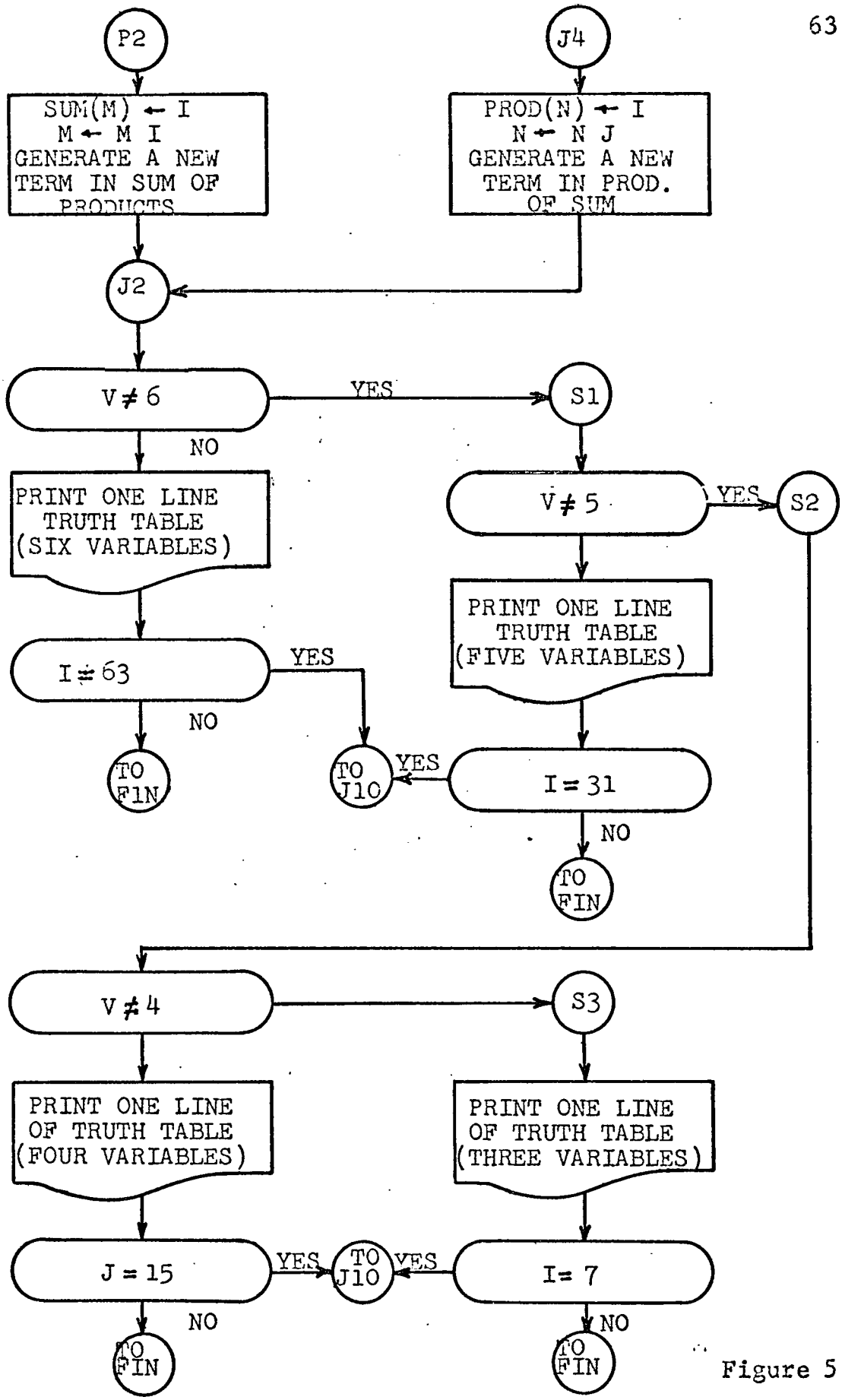

P2
SUM(M) ← I
M ← M I
GENERATE A NEW TERM IN SUM OF PRODUCTS
J4
PROD(N) ← I
N ← N J
GENERATE A NEW TERM IN PROD. OF SUM
J2
V ≠ 6
YES
S1
NO
PRINT ONE LINE TRUTH TABLE (SIX VARIABLES)
V ≠ 5
YES
S2
PRINT ONE LINE TRUTH TABLE (FIVE VARIABLES)
I ≠ 63
YES
NO
TO FIN
TO J10
YES
I = 31
NO
TO FIN
V ≠ 4
S3
PRINT ONE LINE OF TRUTH TABLE (FOUR VARIABLES)
PRINT ONE LINE OF TRUTH TABLE (THREE VARIABLES)
J = 15
YES
TO J10
YES
I = 7
NO
TO FIN
NO
TO FIN

Figure 5.2-b

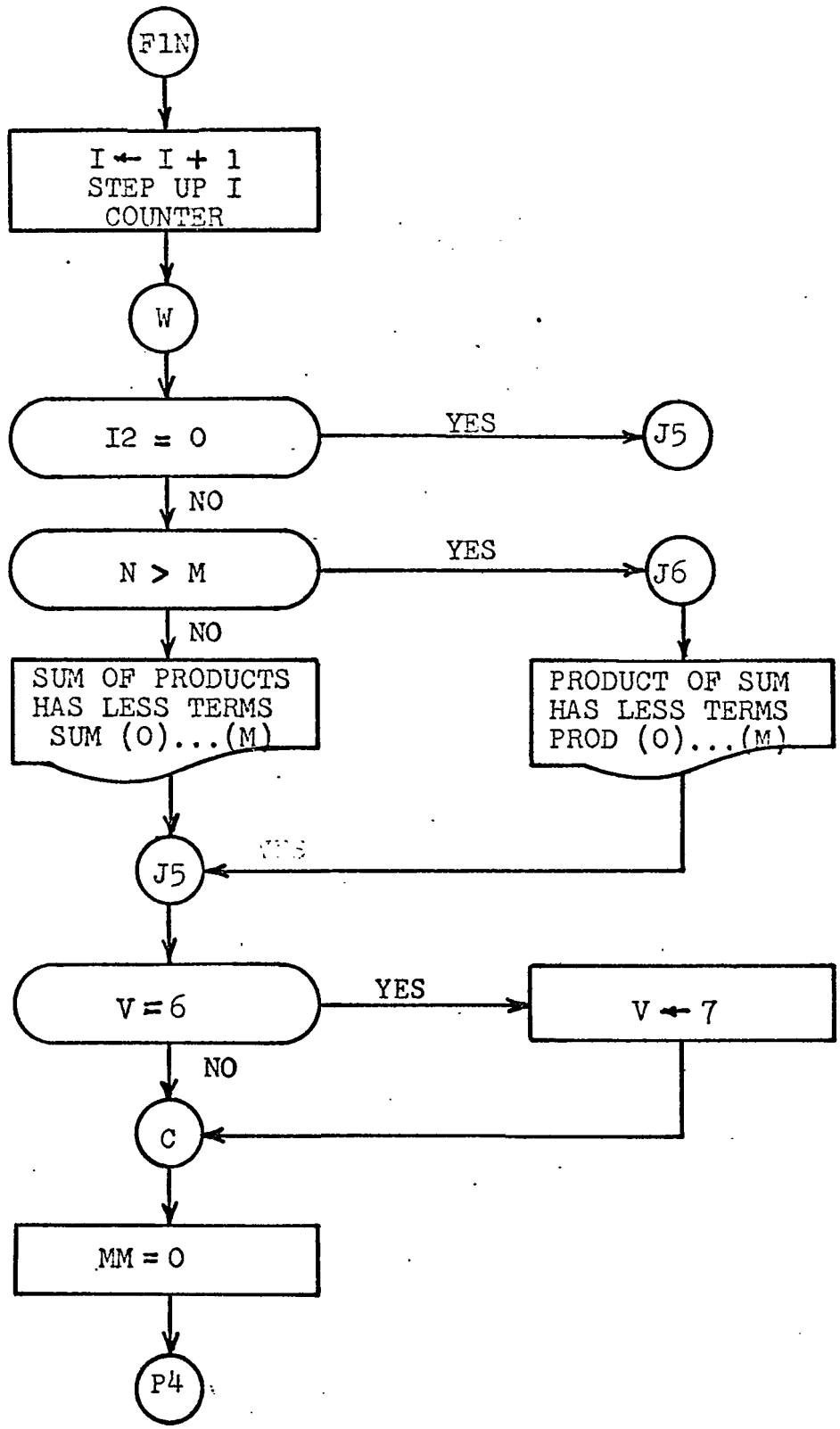

F1N
I ← I + 1
STEP UP I
COUNTER
W
I2 = 0
YES
J5
NO
N > M
YES
J6
NO
SUM OF PRODUCTS
HAS LESS TERMS
SUM (0)...(M)
PRODUCT OF SUM
HAS LESS TERMS
PROD (0)...(M)
J5
V = 6
YES
V ← 7
NO
C
MM = 0
P4

Figure 5.2-c

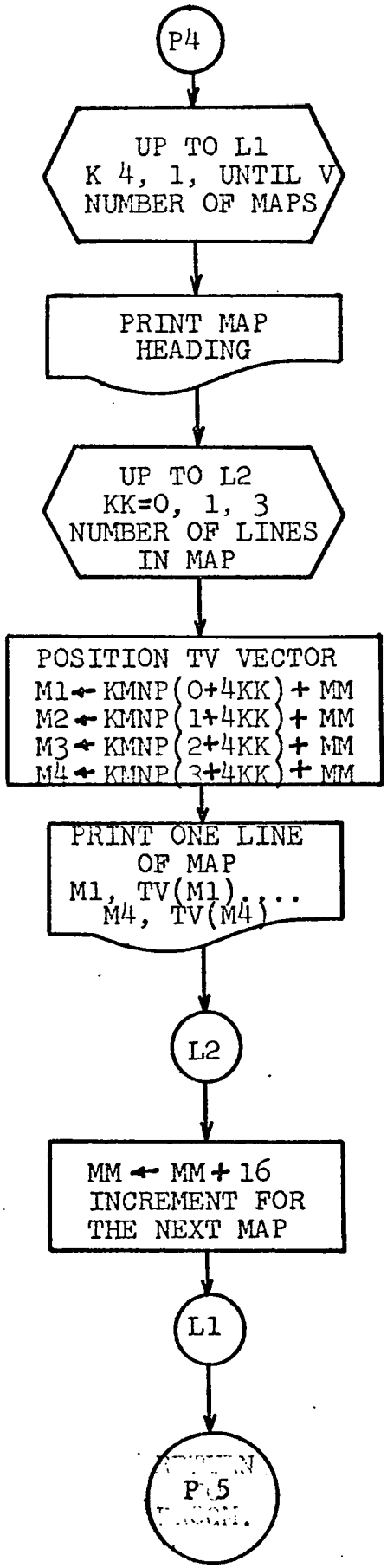

P4
UP TO L1
K 4, 1, UNTIL V
NUMBER OF MAPS
PRINT MAP
HEADING
UP TO L2
KK=0, 1, 3
NUMBER OF LINES
IN MAP
POSITION TV VECTOR
M1 ← KMNP(0+4KK) + MM
M2 ← KMNP(1+4KK) + MM
M3 ← KMNP(2+4KK) + MM
M4 ← KMNP(3+4KK) + MM
PRINT ONE LINE
OF MAP
M1, TV(M1),...
M4, TV(M4)
L2
MM ← MM + 16
INCREMENT FOR
THE NEXT MAP
L1
P5

Figure 5.2-d

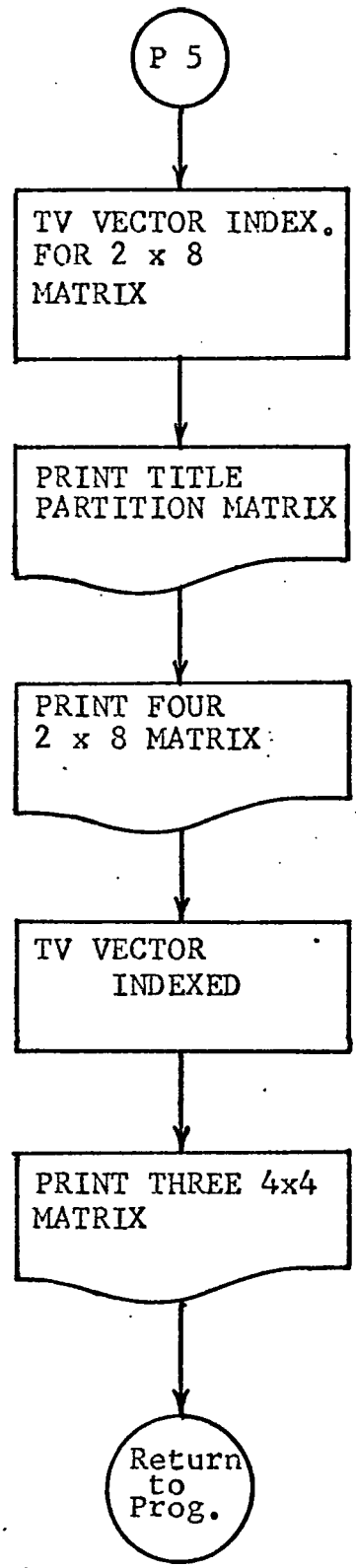

P 5
TV VECTOR INDEX. FOR 2 x 8 MATRIX
PRINT TITLE PARTITION MATRIX
PRINT FOUR 2 x 8 MATRIX
TV VECTOR INDEXED
PRINT THREE 4x4 MATRIX
Return to Prog.

Figure 5.2-e

```
SUBROUTINE FOR MINIMIZING BOOLEAN FUNCTIONS

      EXTERNAL FUNCTION (V,I1,I2,I3)
      NORMAL MODE IS INTEGER
      DIMENSION TV(64),SUM(64),PROD(64)
      BOOLEAN A,B,C,D,E,F,    T,TV,FUN.
      ENTRY TO BOOLEQ.
      I=0
      M=0
      N=0
      BL=$      $
      A1=$    A$
      B1=$    B$
      C1=$    C$
      D1=$    D$
      E1=$    E$
      F1=$    F$
      T1=$    T$
      WHENEVER V.G.5,TRANSFER TO JJ
      F1=BL
      WHENEVER V.G.4,TRANSFER TO JJ
      E1=BL
      WHENEVER V.G.3,TRANSFER TO JJ
      D1=BL
JJ    CONTINUE
      PRINT FORMAT TITLE
      PRINT FORMAT HEADER,F1,E1,D1,C1,B1,A1,  T1
      THROUGH W,FOR VALUES OF F=0B,1B
      THROUGH W,FOR VALUES OF E=0B,1B
      THROUGH W, FOR VALUES OF D=0B,1B
      THROUGH W,FOR VALUES OF C=0B,1B
      THROUGH W,FOR VALUES OF B=0B,1B
      THROUGH W,FOR VALUES OF A=0B,1B
      EXECUTE FUN.(A,B,C,D,E,F,T)
      TV(I)=T
      WHENEVER I1.E.1,TRANSFER TO J1
      EXECUTE MUN.(A,B,C,D,E,F,T)
      WHENEVER TV(I).E.T,TRANSFER TO J3
      PRINT COMMENT $ ERROR IN IMPLEMENTATION $
J3    CONTINUE
J1    CONTINUE
      WHENEVER TV(I).E.0B,TRANSFER TO J4
      SUM(M)=I
      M=M+1
      TRANSFER TO J2
J4    PROD(N)=I
      N=N+1
J2    CONTINUE
      WHENEVER V.NE.6,TRANSFER TO S1
      PRINT FORMAT TAB1,F,E,D,C,B,A,T
      WHENEVER I.E.63,TRANSFER TO J10
      TRANSFER TO FIN
```

```
S1      WHENEVER V.NE.5,TRANSFER TO S2
        PRINT FORMAT TAB2,E,D,C,B,A,T
        WHENEVER I.E.31,TRANSFER TO J10
        TRANSFER TO FIN
S2      WHENEVER V.NE.4,TRANSFER TO S3
        PRINT FORMAT TAB3,D,C,B,A,T
        WHENEVER I.E.15,TRANSFER TO J10
        TRANSFER TO FIN
S3      PRINT FORMAT TAB4,C,B,A,T
        WHENEVER I.E.7,TRANSFER TO J10
FIN     CONTINUE
        I=I+1
W       CONTINUE
J10     CONTINUE
        WHENEVER I2.E.0,TRANSFER TO J5
        WHENEVER N.L.M,TRANSFER TO J6
        PRINT COMMENT $1          SUM OF PRODUCTS HAS LESS TERMS $
        PRINT FORMAT MTERM,M
        PRINT FORMAT SU,SUM(0)...SUM(M-1)
        TRANSFER TO J5
J6      PRINT COMMENT $1          PRODUCT OF SUMS HAS LESS TERMS $
        PRINT FORMAT NTERM,N
        PRINT FORMAT PRO,PROD(0)...PROD(N-1)
J5      PRINT FORMAT TOT,TV(0)...TV(I-1)
        WHENEVER I3.E.0,TRANSFER TO J7
        WHENEVER V.E.6,V=7
        MM=0
        THROUGH L1,FOR K=4,1,K.G.V
        PRINT FORMAT MAP1
        PRINT FORMAT MAP2,DASH(0)...DASH(7)
        THROUGH L2,FOR KK=0,1,KK.G.3
        M1=KMAP(0+KK*4)+MM
        M2=KMAP(1+KK*4)+MM
        M3=KMAP(2+KK*4)+MM
        M4=KMAP(3+KK*4)+MM
        PRINT FORMAT MAP7,M1,M2,M3,M4
        PRINT FORMAT MAP3
        PRINT FORMAT MAP3
        PRINT FORMAT MAP5,MK(KK),TV(M1),TV(M2),TV(M3),TV(M4)
        THROUGH L4,FOR II=1,1,II.G.3
L4      PRINT FORMAT MAP3
L2      PRINT FORMAT MAP4,DASH(0)...DASH(7)
        ID=(V+K)-8
        PRINT FORMAT MAP6,MAPID(ID)
        MM=MM+16
L1      CONTINUE
J7      CONTINUE
        PRINT FORMAT NPG
        PRINT FORMAT LDM,IDM(0),IDM(1)
        PRINT FORMAT EDM,TV(0)...TV(7)
        PRINT FORMAT DDM,TV(8)...TV(15)
        PRINT FORMAT LDM,IDM(2),IDM(3)
        PRINT FORMAT EDM,TV(0)...TV(3),TV(8)...TV(11)
        PRINT FORMAT DDM,TV(4)...TV(7),TV(12)...TV(15)
        PRINT FORMAT LDM,IDM(4),IDM(5)
        PRINT FORMAT EDM,TV(0),TV(1),TV(4),TV(5),TV(8),TV(9),TV(12),TV(13)
        PRINT FORMAT DDM,TV(2),TV(3),TV(6),TV(7),TV(10),TV(11),TV(14),TV(15)
        PRINT FORMAT LDM,IDM(6),IDM(7)
        PRINT FORMAT EDM,TV(0),TV(2),TV(4),TV(6),TV(8),TV(10),TV(12),TV(14)
        PRINT FORMAT DDM,TV(1),TV(3),TV(5),TV(7),TV(9),TV(11),TV(13),TV(15)
```

```
      PRINT FORMAT LDM,IDM(8),IDM(9)
      PRINT FORMAT FDM,TV(0)...TV(3)
      PRINT FORMAT FDM,TV(4)...TV(7)
      PRINT FORMAT FDM,TV(8)...TV(11)
      PRINT FORMAT FDM,TV(12)...TV(15)
      PRINT FORMAT LDM,IDM(10),IDM(11)
      PRINT FORMAT FDM,TV(0),TV(1),TV(4),TV(5)
      PRINT FORMAT FDM,TV(2),TV(3),TV(6),TV(7)
      PRINT FORMAT FDM,TV(8),TV(9),TV(12),TV(13)
      PRINT FORMAT FDM,TV(10),TV(11),TV(14),TV(15)
      PRINT FORMAT LDM,IDM(12),IDM(13)
      PRINT FORMAT FDM,TV(0),TV(2),TV(4),TV(6)
      PRINT FORMAT FDM,TV(1),TV(3),TV(5),TV(7)
      PRINT FORMAT FDM,TV(8),TV(10),TV(12),TV(14)
      PRINT FORMAT FDM,TV(9),TV(11),TV(13),TV(15)
      VECTOR VALUES TITLE=$1H1,S45,13H TRUTH TABLE *$
      VECTOR VALUES HEADER=$1HO,S20,7C6/ *$
      VECTOR VALUES TAB1=$1H ,S20,7I6*$
      VECTOR VALUES TAB2=$1H ,S20,S6,6I6*$
      VECTOR VALUES TAB3=$1H ,S32,5I6*$
      VECTOR VALUES TAB4=$1H ,S38,4I6*$
      VECTOR VALUES MTERM=$1HO,S10,16HNUMB. OF MIDTERM,I5*$
      VECTOR VALUES SU=$5H SUM=(16(I2,1H,))*$
      VECTOR VALUES NTERM=$1HO,S10,16HNUMB. OF MAXTERM,I5*$
      VECTOR VALUES PRO=$6H PROD=(16(I2,1H,))*$
      VECTOR VALUES TOT=$6H TVAL=(16(I2,1H,))*$
      VECTOR VALUES MAP1=$1H1,S20,2HDC,S3,2H00,S10,2H01,S10,2H11,S10,2H10*$
      VECTOR VALUES MAP2=$1HO,S14,2HBA,S4,8C6,1H+*$
      VECTOR VALUES MAP3=$1H ,S20,5(1HI,S11)*$
      VECTOR VALUES MAP5=$1H ,S14,C2,S4,4(1HI,S5,I1,S5),1HI*$
      VECTOR VALUES MAP4=$1H ,S20,8C6,1H+*$
      VECTOR VALUES MAP6=$1HO,S28,13HKARNAUGH MAP ,C4*$
      VECTOR VALUES MAP7=$1H ,S20,4(1HI,I2,S9),1HI*$
      VECTOR VALUES MAPID=$    $,$ -E $,$  E $,$-E-F$,$-E F$,$ E-F$,$ E F$
      VECTOR VALUES KMAP=0,4,12,8,1,5,13,9,3,7,15,11,2,6,14,10
      VECTOR VALUES MK=$00$,$01$,$11$,$10$
      VECTOR VALUES DASH=$+-----$,$------$,$+-----$,
     1 $------$,$+-----$,$------$,$+-----$,$------$
      VECTOR VALUES NPG=$1H1,S20,14HPARTION MATRIX*$
      VECTOR VALUES LDM=$1HO,S16,C4,S4,C40 *$
      VECTOR VALUES EDM=$1H ,S20,8I2*$
      VECTOR VALUES DDM=$1H ,S20,8I2*$
      VECTOR VALUES FDM=$1H ,S20,4I2*$
      VECTOR VALUES IDM=$D$,$CBA$,$C$,$DBA$,$B$,$DCA$,$A$,$DCB$,
     0 $DC$,$BA$,$DB$,$CA$,$DA$,$CB$
      FUNCTION RETURN
      END OF FUNCTION
```

.131246.....131246.....131246.....131246.....131246.....131246.....131246.

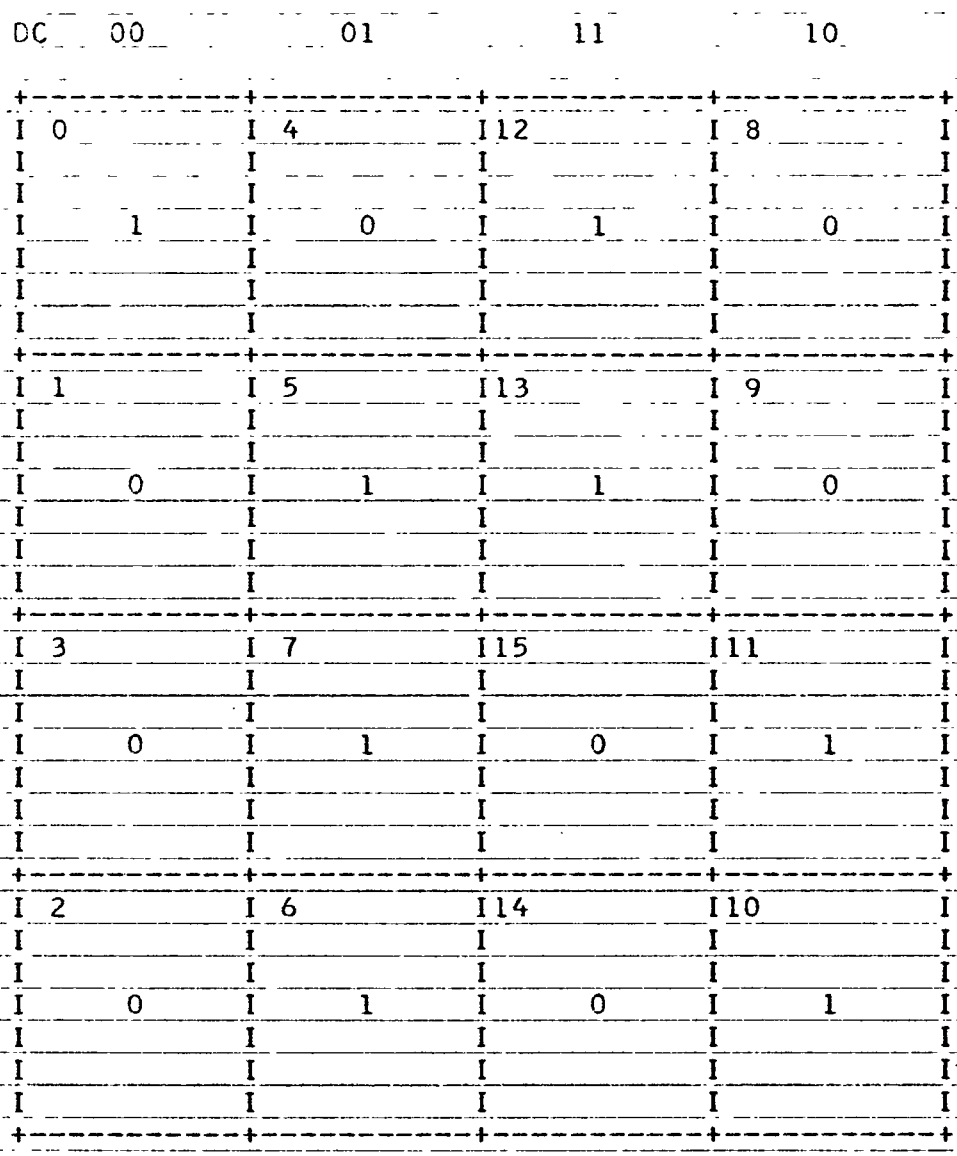
```
$ COMPILE MAD, EXECUTE

AD (01 OCT 1965 VERSION) PROGRAM LISTING ... ... ...

        TESTING BOOLEAN EXTERNAL FUNCTION

          NORMAL MODE IS INTEGER
          EXECUTE BOOLEQ.(4,0,1,1)
          END OF PROGRAM
```

```
$ COMPILE MAD

MAD (01 OCT 1965 VERSION) PROGRAM LISTING ... ... ...


          EXTERNAL FUNCTION (A,B,C,D,E,F,T)
          NORMAL MODE IS BOOLEAN
          ENTRY TO FUN.
          T=.NOT.A.AND..NOT.B.AND..NOT.C.AND..NOT.D.OR.
        0 A.AND..NOT.B.AND.C.AND..NOT.D.OR.
        1 .NOT.A.AND.B.AND.C.AND..NOT.D.OR.
        3 A.AND.B.AND.C.AND..NOT.D.OR.
        4 .NOT.A.AND.B.AND..NOT.C.AND.D.OR.
        5 A.AND.B.AND..NOT.C.AND.D.OR.
        6 .NOT.A.AND..NOT.B.AND.C.AND.D.OR.
        7 A.AND..NOT.B.AND.C.AND.D
          FUNCTION RETURN
          ENTRY TO MUN.
          T=.NOT.A.AND..NOT.B.AND.      D.          OR.
        1   .NOT.A.AND.      C.AND..NOT.D.          OR.
        2        A.AND..NOT.C.AND..NOT.D.          OR.
        3   .NOT.A.AND.      C.AND..NOT.D.          OR.
        4        A.AND.      B.AND.      D.          OR.
        5        A.AND..NOT.C.AND.       D.          OR.
        6        A.AND.      C.AND.      D.          OR.
        7        A.AND..NOT.B.AND..NOT.C
          FUNCTION RETURN
          END OF FUNCTION
```

TRUTH TABLE

D	C	B	A	T	
					ERROR IN IMPLEMENTATION
0	0	0	0	0	
					ERROR IN IMPLEMENTATION
0	0	0	1	1	
0	0	1	0	0	
					ERROR IN IMPLEMENTATION
0	0	1	1	1	
					ERROR IN IMPLEMENTATION
0	1	0	0	1	
					ERROR IN IMPLEMENTATION
0	1	0	1	0	
0	1	1	0	1	
					ERROR IN IMPLEMENTATION
0	1	1	1	0	
					ERROR IN IMPLEMENTATION
1	0	0	0	1	
					ERROR IN IMPLEMENTATION
1	0	0	1	1	
					ERROR IN IMPLEMENTATION
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	0	
					ERROR IN IMPLEMENTATION
1	1	1	1	1	

```
SUM OF PRODUCTS HAS LESS TERMS

NUMB. OF MIDTERM  8
SUM= 0, 5, 6, 7,10,11,12,13,
TVAL= 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
```



KARNAUGH MAP

PARTITION MATRIX

D CBA
1 0 0 0 0 1 1 1
0 0 1 1 1 1 0 0

C DBA
1 0 0 0 0 0 1 1
0 1 1 1 1 1 0 0

B DCA
1 0 0 1 0 0 1 1
0 0 1 1 1 1 0 0

A DCB
1 0 0 1 0 1 1 0
0 0 1 1 0 1 1 0

DC BA
1 0 0 0
0 1 1 1
0 0 1 1
1 1 0 0

DB CA
1 0 0 1
0 0 1 1
0 0 1 1
1 1 0 0

DA CB
1 0 0 1
0 0 1 1
0 1 1 0
0 1 1 0

```
$ COMPILE MAD, EXECUTE

MAD (01 OCT 1965 VERSION) PROGRAM LISTING ... ... ...


            TESTING BOOLEAN EXTERNAL FUNCTION

              NORMAL MODE IS INTEGER
              EXECUTE BOOLEQ.(5,0,1,1)
              END OF PROGRAM



              EXTERNAL FUNCTION (A,B,C,D,E,F,T)
              NORMAL MODE IS BOOLEAN
              ENTRY TO FUN.
              T=.NOT.A.AND..NOT.B.AND.      D.AND..NOT.E.OR.
            1   .NOT.A.AND.      C.AND..NOT.D.AND.      E.OR.
            2        A.AND..NOT.C.AND..NOT.D.AND.      E.OR.
            3   .NOT.A.AND.      C.AND..NOT.D.                OR.
            4        A.AND.      B.AND.      D.AND..NOT.E.OR.
            5        A.AND..NOT.C.AND.      D.AND..NOT.E.OR.
            6        A.AND.      C.AND.      D.AND.      E.OR.
            8 A.AND..NOT.B.AND.C.AND.D.AND..NOT.E.OR.
            9 A.AND.B.AND..NOT.C.AND.D.AND.E.OR.
            7        A.AND..NOT.B.AND..NOT.C.AND.      E
              FUNCTION RETURN
              ENTRY TO MUN.
              T=.NOT.A.AND.C.AND..NOT.D.OR.
            1 A.AND.D.OR.
            2 .NOT.B.AND.D.AND..NOT.E.OR.
            3 A.AND..NOT.C.AND.E
              FUNCTION RETURN
              END OF FUNCTION
```

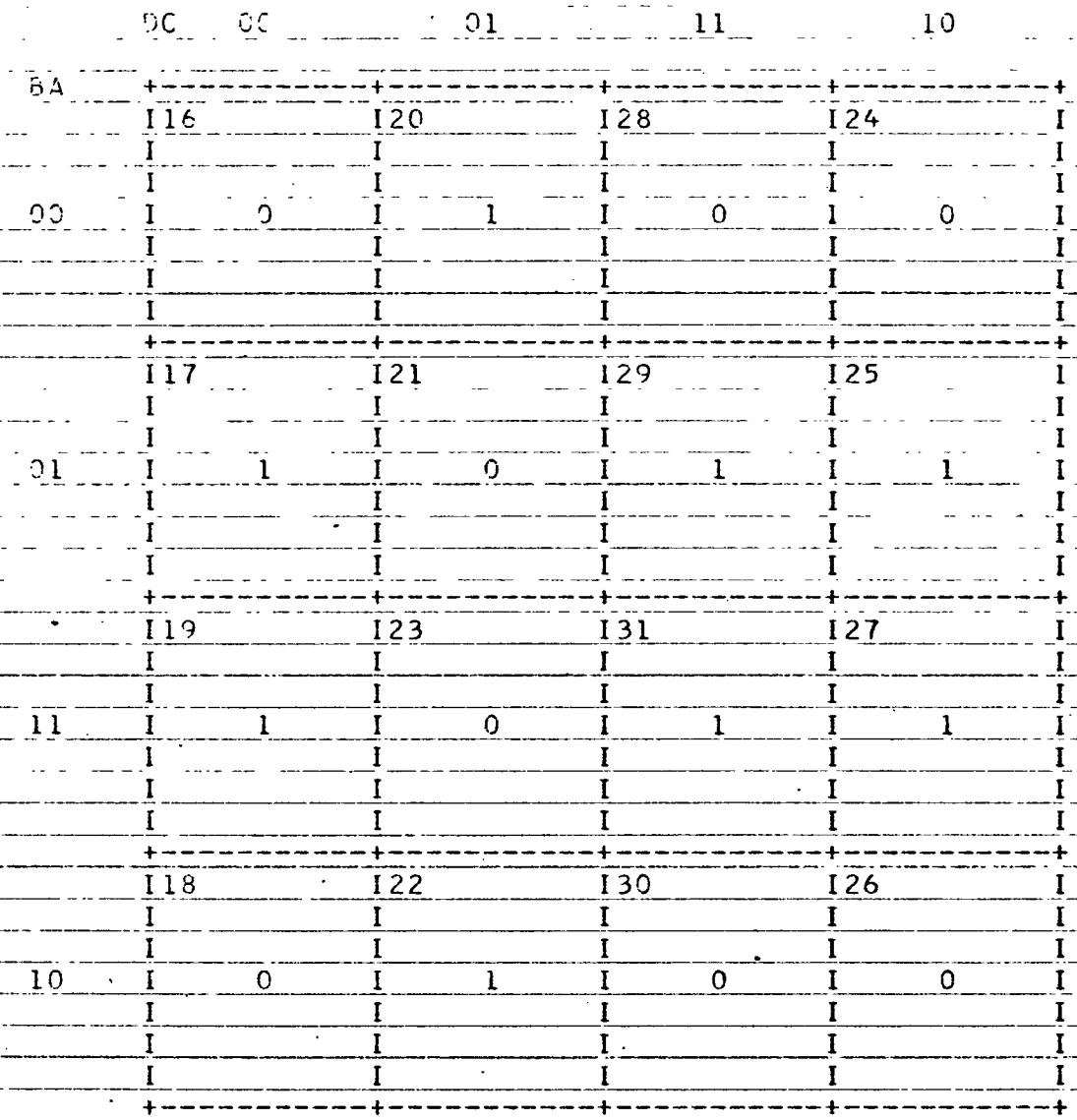
TRUTH TABLE

E	D	C	B	A	T
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

SUM OF PRODUCTS HAS LESS TERMS

NUMB. OF MIDTERM 16

SUM= 4, 6, 8, 9,11,12,13,15,17,19,20,22,25,27,29,31,
TVAL= 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,



KARNAUGH MAP E

EA \ DC CO	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

KARNAUGH MAP -E

CHAPTER VI

SUMMARY AND CONCLUSION

Recapitulation

Many logic designers use a pragmatic approach when simplifying switching circuits for digital applications. That is so because no firmly established routine covers all aspects of minimization. The procedure, described in this paper, may be utilized as a simplification - implementation tool by any engineer acquainted with switching circuitry. Further, the computer program may be adapted as a subroutine within the framework of a bigger logic design and implementation program.

Research Direction

Computers have been successfully applied to a variety of problems by adapting the revelant data to a computer language. Algorithmic iterative minimization methods are developed having computer languages in mind. Nevertheless, a manual-visual task, like that of minimization by graphical methods, may be as efficient as any other if a computer is utilized as a logic designer's tool.

The procedure described in this paper may be further automated, provided enough high speed random access storage is available for manipulating the parameters involved.

Author's Contribution

This research resulted in a procedure for minimization of Boolean functions aided by a computer program. Included in this procedure are original techniques for direct map to multi-level NAND/NOR implementation.

Postulates

$A = 1$ or else $A = 0$

$1.1 = 1$	$0 + 0 = 0$
$1.0 = 0.1 = 0$	$0 + 1 = 1 + 0 = 1$
$0.0 = 0$	$1 + 1 = 1$
$\overline{1} = 0$	$\overline{0} = 1$

Theorems

1a. $0.A = 0$	1b. $1 + A = 1$
2a. $1.A = A$	2b. $0 + A = A$
3a. $A\,A = A$	3b. $A + A = A$
4a. $A\,\overline{A} = 0$	4b. $A + \overline{A} = 1$
5a. $A\,B = BA$	5b. $A + B = B + A$

6a. $ABC = (AB)C = A\,(BC)$

6b. $A + B + C = (A + C) + C = A + (B + C)$

7a. $\overline{AB\ldots C} = \overline{A} + \overline{B} + \ldots + \overline{C}$

7b. $\overline{A + B + \ldots + C} = \overline{A}\,\overline{B}\ldots\overline{C}$

8. $\overline{f}\,(A,B,\ldots,C,.,+) = f\,(\overline{A},\overline{B},\ldots,\overline{C},+,.)$

9a. $A\,B + A\,C = A\,(B + C)$

9b. $(A + B)(A + C) = A + BC$

10a. $A\,B + A\,\overline{B} = A$

10b. $(A + B)(A + \overline{B}) = A$

11a. $A + A\,B = A$

11b. $A\,(A + B) = A$

Theorems

12a. $A + \overline{A}B = A + B$

12b. $A(\overline{A} + B) = AB$

12a. $AC + \overline{A}BC = AC + BC$

12b. $(A + C)(\overline{A} + B + C) = (A + C)(A + B)$

13a. $AB + \overline{A}C + BC = AB = \overline{A}C$

13b. $(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$

14a. $AB + \overline{A}C = (A + C)(\overline{A} + B)$

14b. $(A + B)(\overline{A} + C) = AC + \overline{A}B$

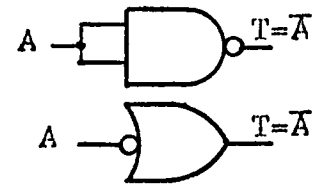
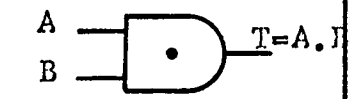
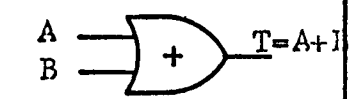
15a. $A \cdot f(A,\overline{A},B,\ldots,C) = A \cdot f(1,0,B,\ldots,C)$

15b. $A + f(A,\overline{A},B,\ldots,C) = A + f(0,1,B,\ldots,C)$

16a. $f(A,\overline{A},B,\ldots,C) = A \cdot f(1,0,B,\ldots,C) + \overline{A} \cdot f(0,1,B,\ldots,C)$

16b. $f(A,\overline{A},B,\ldots,C)$

$= (A + f(0,1,B,\ldots,C))\ (\overline{A} + f(1,0,B,\ldots,C))$

N A M E	LOGIC SYMBOL	MAD LANGUAGE	TRUTH TABLE	LOGIC DIAGRAM
NEGATION COMPLEMENT or PRIME	$\bar{}$ or ' $\bar{A}$ or A' $\sim$ $\sim$ A	.NOT. .NOT.A	$\bar{A}$ | T 0 | 1 1 | 0	
AND or INTERSECTION	. A . B A v B	.AND. A.AND.B	A.B | T 00 | 0 01 | 0 10 | 0 11 | 1	
OR or UNION	+ A + B v A v B	.OR. A.OR.B	A+B | T 00 | 0 01 | 1 10 | 1 11 | 1	
EXCLUSIVE OR or SYMETRIC DIFFERENCE	$\oplus$ A $\oplus$ B A B	.EXOR. A.EXOR.B	A$\oplus$B | T 00 | 0 01 | 1 10 | 1 11 | 0	
NAND or STROKE	$\overline{A.B} = \bar{A} + \bar{B}$ A$\uparrow$B	.NOT.(A.AND.B)	$\overline{A.B}$ | T 00 | 1 01 | 1 10 | 1 11 | 0	
NOR or DAGGER	$\overline{A+B} = \bar{A}.\bar{B}$ A$\downarrow$B	.NOT.(A.OR.B)	A+B | T 00 | 1 01 | 0 10 | 0 11 | 0	

		f e d c b a			f e d c b a
$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$	0	0 0 0 0 0 0	$\bar{A}\bar{B}\bar{C}\bar{D}E\bar{F}$	16	0 1 0 0 0 0
$A\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$	1	0 0 0 0 0 1	$A\bar{B}\bar{C}\bar{D}E\bar{F}$	17	0 1 0 0 0 1
$\bar{A}B\bar{C}\bar{D}\bar{E}\bar{F}$	2	0 0 0 0 1 0	$\bar{A}B\bar{C}\bar{D}E\bar{F}$	18	0 1 0 0 1 0
$AB\bar{C}\bar{D}\bar{E}\bar{F}$	3	0 0 0 0 1 1	$AB\bar{C}\bar{D}E\bar{F}$	19	0 1 0 0 1 1
$\bar{A}\bar{B}C\bar{D}\bar{E}\bar{F}$	4	0 0 0 1 0 0	$\bar{A}\bar{B}C\bar{D}E\bar{F}$	20	0 1 0 1 0 0
$A\bar{B}C\bar{D}\bar{E}\bar{F}$	5	0 0 0 1 0 1	$A\bar{B}C\bar{D}E\bar{F}$	21	0 1 0 1 0 1
$\bar{A}BC\bar{D}\bar{E}\bar{F}$	6	0 0 0 1 1 0	$\bar{A}BC\bar{D}E\bar{F}$	22	0 1 0 1 1 0
$ABC\bar{D}\bar{E}\bar{F}$	7	0 0 0 1 1 1	$ABC\bar{D}E\bar{F}$	23	0 1 0 1 1 1
$\bar{A}\bar{B}\bar{C}D\bar{E}\bar{F}$	8	0 0 1 0 0 0	$\bar{A}\bar{B}\bar{C}DE\bar{F}$	24	0 1 1 0 0 0
$A\bar{B}\bar{C}D\bar{E}\bar{F}$	9	0 0 1 0 0 1	$A\bar{B}\bar{C}DE\bar{F}$	25	0 1 1 0 0 1
$\bar{A}B\bar{C}D\bar{E}\bar{F}$	10	0 0 1 0 1 0	$\bar{A}B\bar{C}DE\bar{F}$	26	0 1 1 0 1 0
$AB\bar{C}D\bar{E}\bar{F}$	11	0 0 1 0 1 1	$AB\bar{C}DE\bar{F}$	27	0 1 1 0 1 1
$\bar{A}\bar{B}CD\bar{E}\bar{F}$	12	0 0 1 1 0 0	$\bar{A}\bar{B}CDE\bar{F}$	28	0 1 1 1 0 0
$A\bar{B}CD\bar{E}\bar{F}$	13	0 0 1 1 0 1	$A\bar{B}CDE\bar{F}$	29	0 1 1 1 0 1
$\bar{A}BCD\bar{E}\bar{F}$	14	0 0 1 1 1 0	$\bar{A}BCDE\bar{F}$	30	0 1 1 1 1 0
$ABCD\bar{E}\bar{F}$	15	0 0 1 1 1 1	$ABCDE\bar{F}$	31	0 1 1 1 1 1

DECIMAL-BINARY CONVERSION TABLE

		f e d c b a			f e d c b a
$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}F$	3 2	1 0 0 0 0 0	$\bar{A}\bar{B}\bar{C}DEF$	4 8	1 1 0 0 0 0
$A\bar{B}\bar{C}\bar{D}\bar{E}F$	3 3	1 0 0 0 0 1	$A\bar{B}\bar{C}DEF$	4 9	1 1 0 0 0 1
$\bar{A}B\bar{C}\bar{D}\bar{E}F$	3 4	1 0 0 0 1 0	$\bar{A}B\bar{C}DEF$	5 0	1 1 0 0 1 0
$AB\bar{C}\bar{D}\bar{E}F$	3 5	1 0 0 0 1 1	$AB\bar{C}DEF$	5 1	1 1 0 0 1 1
$\bar{A}\bar{B}C\bar{D}\bar{E}F$	3 6	1 0 0 1 0 0	$\bar{A}\bar{B}CDEF$	5 2	1 1 0 1 0 0
$A\bar{B}C\bar{D}\bar{E}F$	3 7	1 0 0 1 0 1	$A\bar{B}CDEF$	5 3	1 1 0 1 0 1
$\bar{A}BC\bar{D}\bar{E}F$	3 8	1 0 0 1 1 0	$\bar{A}BCDEF$	5 4	1 1 0 1 1 0
$ABC\bar{D}\bar{E}F$	3 9	1 0 0 1 1 1	$ABCDEF$	5 5	1 1 0 1 1 1
$\bar{A}\bar{B}\bar{C}D\bar{E}F$	4 0	1 0 1 0 0 0	$\bar{A}\bar{B}\bar{C}DEF$	5 6	1 1 1 0 0 0
$A\bar{B}\bar{C}D\bar{E}F$	4 1	1 0 1 0 0 1	$A\bar{B}\bar{C}DEF$	5 7	1 1 1 0 0 1
$\bar{A}B\bar{C}D\bar{E}F$	4 2	1 0 1 0 1 0	$\bar{A}B\bar{C}DEF$	5 8	1 1 1 0 1 0
$AB\bar{C}D\bar{E}F$	4 3	1 0 1 0 1 1	$AB\bar{C}DEF$	5 9	1 1 1 0 1 1
$\bar{A}\bar{B}CD\bar{E}F$	4 4	1 0 1 1 0 0	$\bar{A}\bar{B}CDEF$	6 0	1 1 1 1 0 0
$A\bar{B}CD\bar{E}F$	4 5	1 0 1 1 0 1	$A\bar{B}CDEF$	6 1	1 1 1 1 0 1
$\bar{A}BCD\bar{E}F$	4 6	1 0 1 1 1 0	$\bar{A}BCDEF$	6 2	1 1 1 1 1 0
$ABCD\bar{E}F$	4 7	1 0 1 1 1 1	$ABCDEF$	6 3	1 1 1 1 1 1

DECIMAL-BINARY CONVERSION TABLE

CITED REFERENCES

1
W.V.O. Quine, "A Way to Simplify Truth Functions," American Mathematical Monthly, Vol. 62, 1955, pp. 627-631.

2
E.J. McCluskey, Jr., "Minimization of Boolean Functions," Bell System Technical Journal, Vol. 35, Nov. 1956, pp. 1417-1444.

3
E.W. Veitch, "A Chart Method for Simplifying Truth Functions," Proceedings of Association for Computing Machinery, Pittsburg, Pennsylvania, Meeting May 2 and 3, 1952, pp. 127-133.

4
M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits," Transactions AIEE, Part 1, Communications and Electronics, Vol. 72, Nov. 1953, pp. 593-599.

5
J.M. Copri, "Symbolic Logic," New York, Dover Publications, Inc.

6
Earle, "Synthesizing Minimal Stroke and Dagger Function," IRE Convention 1960.

7
R.L. Ashenhurst, "The Decomposition of Switching Functions" Bell Laboratories, Report 1953 & 1956.

8
B. Arden, B. Gauer, R. Graham, "The Michigan Algorithm Decoder," The University of Michigan Publishing Center, 1962.

BIBLIOGRAPHY

1. Abhyankar, S. "Minimal 'Sum of Products of Sums' Expressions of Boolean Functions," IRE Transactions on Electronic Computers, Vol. EC-7, No. 4, (December 1958).

2. Burgess, R. Charles "Boolean Algebra Minimizer," SHARE PROGRAM LIBRARY (September 15, 1961)

3. Caldwell, S.H. "Switching Circuits and Logical Design," John Wiley & Sons, Inc. (1958).

4. Curtis, Allen H. "A New Approach to the Design of Switching Circuits," D. Van Nostrand Company, Inc. (1962).

5. Ewing, Ann "PK MIN 4," SHARE PROGRAM LIBRARY (March 3, 1961).

6. Hurley, R.B. "Transistor Logic Circuits," John Wiley & Sons, Inc. (1961).

7. Maley and Earle "The Logic Design of Transistor Digital Computers," Englewood Cliffs, N.Y. (1963).

8. Markus, M. "Switching Circuits for Engineers," Prentice-Hall, Inc. (1962).

9. Naslin, P. "Circuits a Relais et Automatismes a Sequences," Dunod, Paris (1958).

10. Quine, W.V. "The Problem of Simplifying Truth Functions." American Mathematical Monthly, Vol. 59, No. 8, (October 1952).

11. Roth, J.P. "Algebraic Topological Methods for the Synthesis of Switching Systems I," Transactions American Mathematical Society, Vol. 88 No. 2, (July 1958).

12. Rothmeier, J.J. "An Algorithm for Boolean Simplification," Cornell Aeronautical Lab., Inc., Box 235, Buffalo 21, New York.

13. Urbano, R.H., and Mueller, R.K.A. "A Topological Method for the Determination of the Minimal Forms of a Boolean Function," IRE TRANSACTIONS, Vol. EC-5, (September 1956).