AN ABSTRACT OF THE THESIS OF

LIN, PEI-CHEN for the degree of MASTER OF SCIENCE

in ELECTRICAL&COMPUTER ENGINEERING presented on June 8, 1979

Title: MULTIPLE OUTPUT COMBINATIONAL NETWORK MINIMIZATION

Abstract approved: Redacted for privacy

Dr. V. Michael Powers

An important step in the design of digital networks lies in the derivation of the switching formulas which describe the combinational logic networks in the system.

In most large systems the number of gates and the number of connections are major factors that affect the cost of the system. An algorithm MOMIN which minimizes these two factors according to a selected cost function in two-level, multiple-output combinational logic networks is presented in this thesis. Attempts have been made to solve the problem in two aspects: (1) Minimal solution, (2) sub-minimal solution. (2) is described in detail and (1) is implemented.

A computer program MOMIN implemented in FORTRAN has

been prepared which automatically derives a set of minimal cost switching expressions describing the given multiple-output combinational logic network.

Multiple-output Combinational Network Minimization

by

LIN, PEI-CHEN

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Completed June 8, 1979

Commencement June 1980

APPROVED:

Redacted for privacy

Professor of Electrical & Computer Engineering
in charge of major

Redacted for privacy

Head of Department of Electrical & Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented June 8, 1979

Typed by Elee Anita Ann for LIN, PEI-CHEN

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

Multiple-output Combinational Network Minimization

I. Intorduction

A. Statement of the problem

The design of digital networks involves many aspects, in which circuit minimization is one that has to be considered first before considering any hardware implementation if cost and volume of product are important factors. However, the minimization of the number of input connections as well as the number of gates is now most useful in the design of circuit compenents, such as MSI and LSI devices, and circuits that are not available in chip form. The traditional techniques are not very useful in the design of large circuits using MSI and/or LSI devices as components to achieve a low cost syatem.

Circuits minimization includes single-output and multiple-output networks. Since single-output networks are rarely seen (at least comparatively) in the real world, multiple-output network minimization becomes more important.

The main computational distinction between the minimization of single-output and multiple-output networks is that the definition of prime implicant must be broadened to contain multiple-output prime implicants (MOPI) which can cover cells in more than one output function. Thus the advantage of using MOPI's is that once generated they can be shared and combined to implement different output functions. It has been shown that the network cost of an m-output network may be significantly reduced(over a separate minimization) by considering the irredundant cover including MOPI's. In this thesie, a multiple-output function minimization algorithm MOMIN is presented to see its efficiency and applicability.

B. Organization of thesis

The remainder of this thesis is devided into four chapters, each of which contributes a portion of the understanding of the MOMIN algorithm. Chapter II contains an intorduction to the background concepts involved in developing MOMIN. Chapter III describes the MOMIN algorithm in detail. Chapter IV gives demonstration of the effectiveness of the algorithm; the minimized multiple-output network costs are compared to the summation of individual single-output network costs. Some conclusions are made at the end. A summary is in chapter V. Finally, the appendices contain a listing of the FORTRAN program MOMIN as well as forty-three sample problems all in computer output form.

II. Background Concepts

This chapter introdu ces some important concepts that involve switching algebra as well as some related works accomplished in the area of switching function minimization.

Switching algebra (or Boolean algebra), first studied by GEorge Boole[1], is the mathematical foundation of switching theory and logic design. Some of the important terminology is defined below to assure understanding of the information presented in the remainder of this thesis.

A switching function is a combination of a finite number of switching variables (A,B,C....) and constants (0,1) by means of the switching operators AND(AB), OR(A+B) and COMPLEMENT($\bar{A}$). e.g. $F(A,B,C)=\bar{A}B+AC$ indicates the operation of ((COMPLEMENT of A) AND B) OR (A AND C).

A literal is a switching variable in either true or complemented form. $\bar{A}$, B, A and C in the above example are all literals. A disjunctive canonical form is a switching function that is expressed as a sum of product terms, each of which contains all the switching variables of the function. For example, $F(A,B,C)=A\bar{B}C+\bar{A}\bar{B}\bar{C}$ is in disjunctive canonical form, but $F(A,B,C)=ABC+A\bar{C}$ is not.

[1]Boole, George, 1815-1864. An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilties. New York, Dover Pub. Co. 1951.

A minterm of a n-variable switching function is a switching product of n distinct literals. Any switching function F(A,B,C,...) can be expressed by a switching expression in the form of a sum of all those minterms which correspond to a true value for the function F.

A product term I of literals of a switching function F is an implicant if it implies F. I implies F if F is true whenever I is true. An implicant I is called a prime implicant (PI) if and only if there exists no other implicant I' such that I implies I'. An implicant I is said to cover a minterm m if and only if m implies I. A PI is said to be essential if it covers at least one minterm that is not covered by any other PI.

A true minterm must be covered in any equivalent of a switching function. A redundant minterm need not be covered, but may be covered if this allows reduction of the cost of the representation.

A switching function F of three variables may be expressed in canonical sum-of-product form, $F(A,B,C)= \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$, this function has the true value "1" for the combination of A, B and C, $F(A,B,C)=\{010,011,101,111\}$. Sometimes the minterm notation m_i is used to express the same function as $F(A,B,C)= m_2 + m_3 + m_5 + m_7$, where i is the binary value for 010,011,101 and 111. They may also be written as $F(A,B,C)= \Sigma(2,3,5,7)$.

For easy computation, numerical representations of minterms will in this thesis be in octal form. e.g. a 4-variable function $F(A,B,C,D) = \Sigma(0,2,4,8,12,13)$ in decimal becomes $\sum(0,2,4,10,14,15)$ in octal.

A minimal sum-of-product form of a switching function is a switching function in sum-of-product form logically equivalent to the given function, but implementable with a minimum of hardware, according to some accounting method such as;

1) The two-level AND-OR realization of the minimal sum-of-product form has a minimum number of AND gates.
2) No AND gate can be replaced by an AND gate with fewer inputs.

The problem of switching function minimization may be stated as: Given a switching function, find a minimal form. Some techniques are introducted below to serve as a probe into the problems.

A. Algebraic reduction (2)

Minimization of switching functions in canonical sum-of product form may be performed by application of the following Boolean algebra axioms:

A1	Idempotent	$XX=X$	$X+X=X$
A2	Commutative	$XY=YX$	$X+Y=Y+X$
A3	Associative	$X(YZ)=(XY)Z$	$X+(Y+Z)=(X+Y)+Z$
A4	Absorptive	$X(X+Y)=X$	$X+(XY)=X$
A5	Distributive	$X(Y+Z)=(XY)+(XZ)$	$X+(YZ)=(X+Y)(X+Z)$
A6	ZERO and ONE elements		
		$X1=1X=X$	$X+0=0+X=X$
A7	Complement	$X\bar{X}=0$	$X+\bar{X}=1$

The switching function $F(A,B,C)=\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}C+ABC$ may be reduced by applying the above axioms as:

$$\begin{aligned}\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}C+ABC&=\bar{A}B\bar{C}+\bar{A}BC+ABC+A\bar{B}C && \text{A2}\\ &=\bar{A}B(\bar{C}+C)+AC(B+\bar{B}) && \text{A5}\\ &=\bar{A}B+AC && \text{A2 and A7}\end{aligned}$$

B. Graphic simplification (5)

There are several graphic simplification methods, such as Venn diagram(6), Veitch diagram(5) and Karnaugh map(6), among which, the Karnaugh map is probably the most convenient one to use. Fig. 2.1 shows a 3-variable Karnaugh map. The Karnaugh map is constructed such that minterms combinable by Axiom 5 and Axiom 7 are adjacent on the map. The preceding 3-variable function is shown plotted on the Karnaugh map in Fig. 2.1(d) with the minterms of the function represented on the map by a one. It is noted in Fig. 2.1 (b) that no squares differ from any adjacent square by more

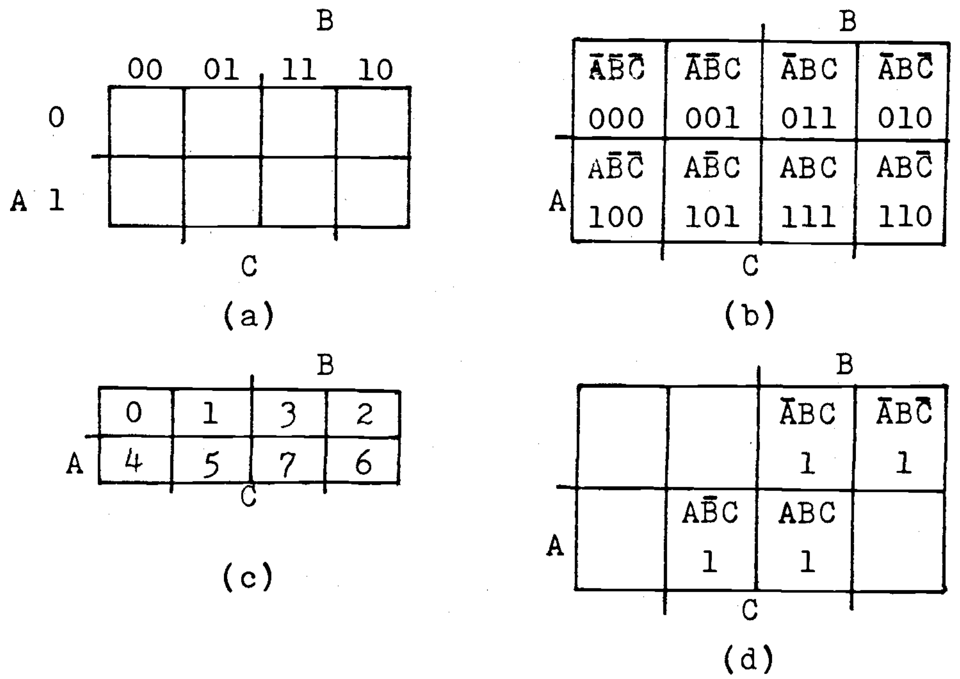


Figure 2.1 3-variable Karnaugh map (a) Original (b) Terms entered (c) Decimal equivalent (d) $F(A,B,C)=\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}C+ABC$

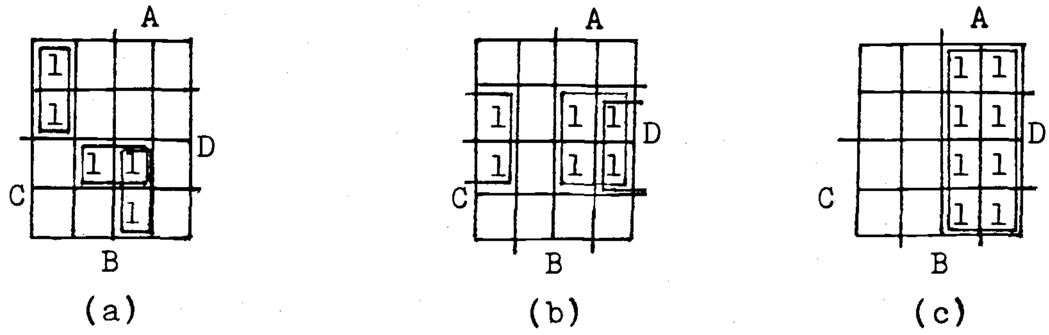


Figure 2.2 Simplification with Karnaugh map

(a) $F(A,B,C,D)=\sum(0,1,7,16,17)$

(b) $F(A,B,C,D)=\sum(1,3,11,13,15,17)$

(c) $F(A,B,C,D)=\sum(10,11,12,13,14,15,16,17)$

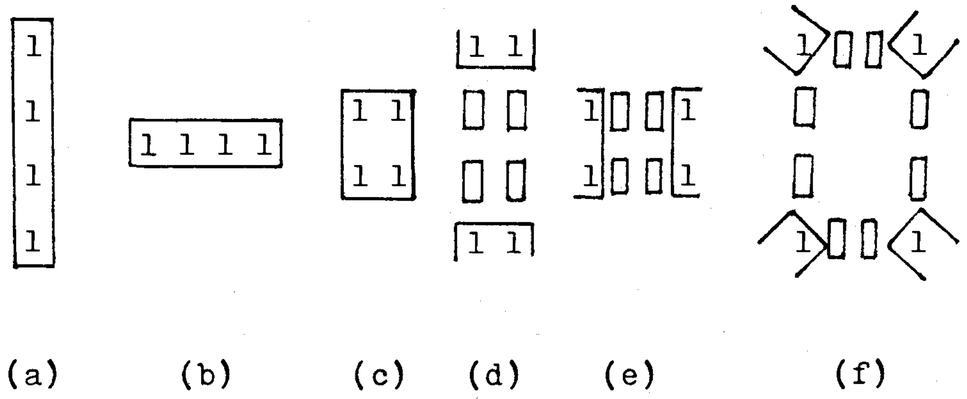
than one variable; thus makes the combination of any adjacent squares possible. In other words, the Karnaugh map shows all the adjacencies that exist. For convenience, we shall call each square in the Karnaugh map a o-cell.

Following are some basic properties of the Karnaugh map minimizations:

1). Any adjacent pair of o-cells marked by "1" in the Karnaugh map can be combined into one term, and one variable is eliminated. In Fig. 2.2(a), minterms 0,1; 16, 17 and 7, 17 are adjacent pairs, they are combined to form three terms (o, 1), (16,17) and (7,17), or equivalently $A\bar{B}\bar{C}$, ABC and BCD. They are called 1-cells.

2). If four o-cells that are marked by 1 form one of the six patterns shown in Fig. 2.3, they can be combined into one term, and two variables can be eliminated. In Fig. 2.2(b), patterns (c) and (e) of Fig. 2.3 are recognized, the combined terms are (1,3,11,13) and (11,13,15,17), which correspond to $\bar{B}D$ and AD, respectively. They are called 2-cells.

3). If eight o-cells that are marked by 1 form one of the four patterns shown in Fig. 2.4, they can be combined into one term, and three variables can be eliminated. In Fig. 2.2(c), pattern (a) of Fig. 2.4 is recognized, the combined term is



▯ indicates spacing

Figure 2.3 Six patterns for which four minterms may be combined

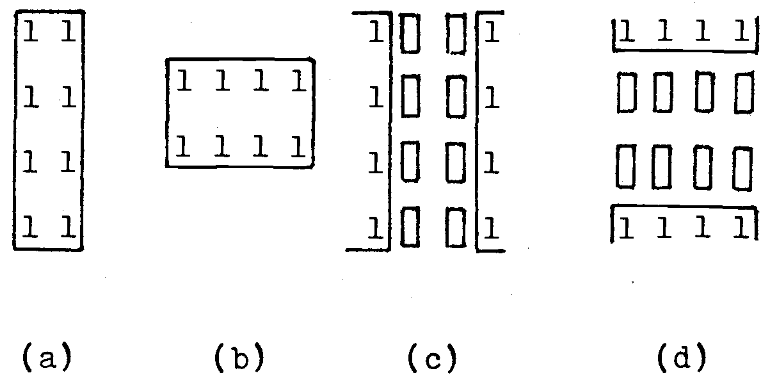


Figure 2.4 Four patterns for which eight minterms may be combined

(10,11,12,13,14,15,16,17), which corresponds to A. It is called a 3-cell.

Karnaugh maps can be used to simplify multiple-output switching functions. The essence lies in the usage of terms common to more than one output functions. A 3-output 4-variable switching function is shown in Fig. 2.5 with

$$F_1=\sum(7,11,13,15,16,17)+d(5)$$
$$F_2=\sum(0,1,2,3,6,7,17)$$
$$F_3=\sum(0,1,2,6,7,10,11,16,17)+d(3,4)$$

If we minimize them individually, we obtain

$$F_1=(11,13,15,17)+(5,7,15,17)+(16,17)$$
$$F_2=(0,1,2,3)+(2,3,6,7)+(7,17)$$
$$F_3=(0,1,2,3)+(0,1,10,11)+(6,7,16,17)$$

If individual networks are constructed for each output function, a total of 9 AND gates and 3 OR gates is required as shown in Fig. 2.6. It is noted that the PI(0.1.2.3) appears in the realization of F_2 and F_3. It may be shared to reduce the number of AND gates required to 8, Further, if we consider the simplification of F_1, F_2 and F_3 at the same time, as shown in Fig. 2.7 we obtain

$$F_1=(7,17)+(16,17)+(11,13,15,17)$$
$$F_2=(0,1,2,3)+(2,3,6,7)+(7,17)$$
$$F_3=(0,1,2,3)+(2,3,6,7)+(16,17)+(0,1,10,11)$$

It is clear that PI's (7,17), (16,17), (0,1,2,3) and (2,3, 6,7) are each shared by two output function, thus reducing

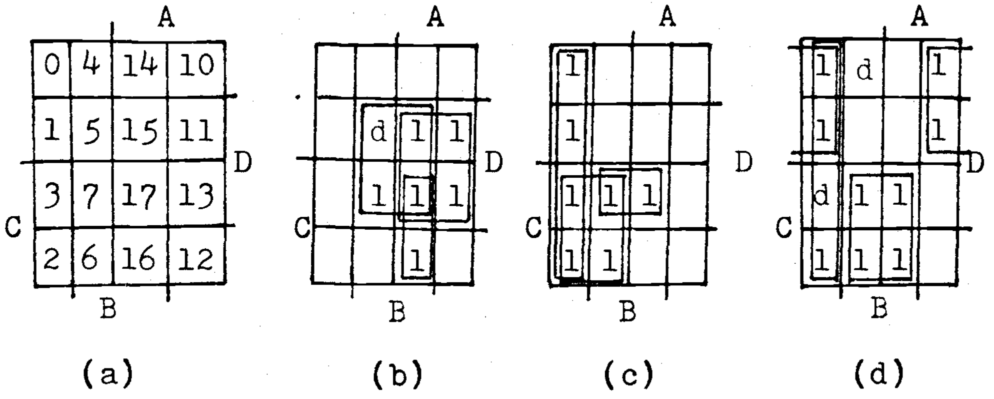


Figure 2.5 4-variable Karnaugh map for 3-output functions, (a) Number representation (b) F_1 (c) F_2 (d) F_3

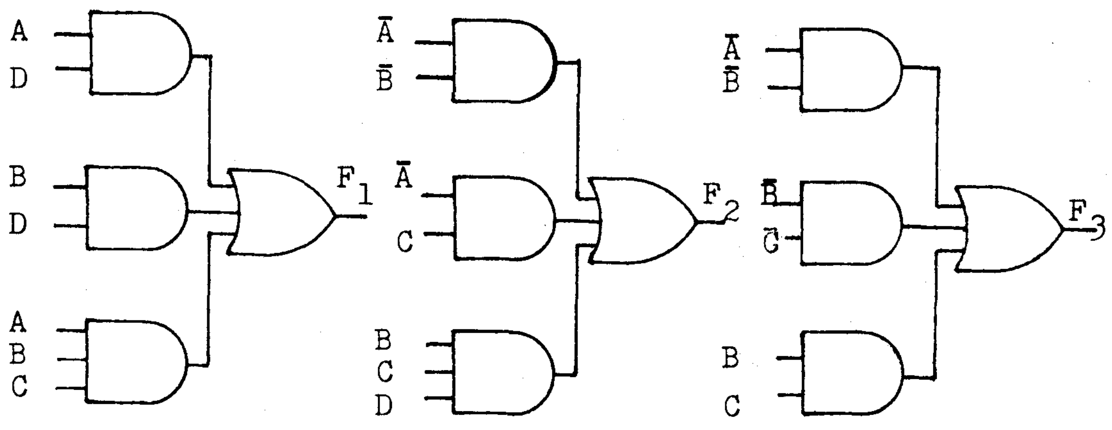


Figure 2.6 Single-output minimization results, 9 AND gates and 3 OR gates are required

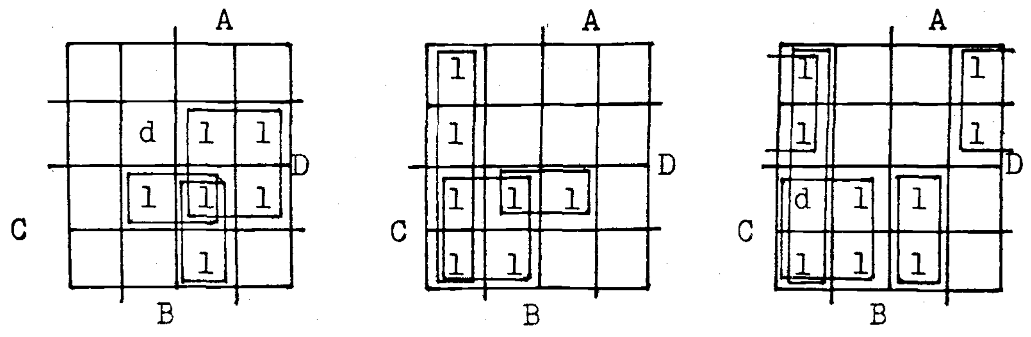


Figure 2.7 Multiple-output functions minimization using Karnaugh maps

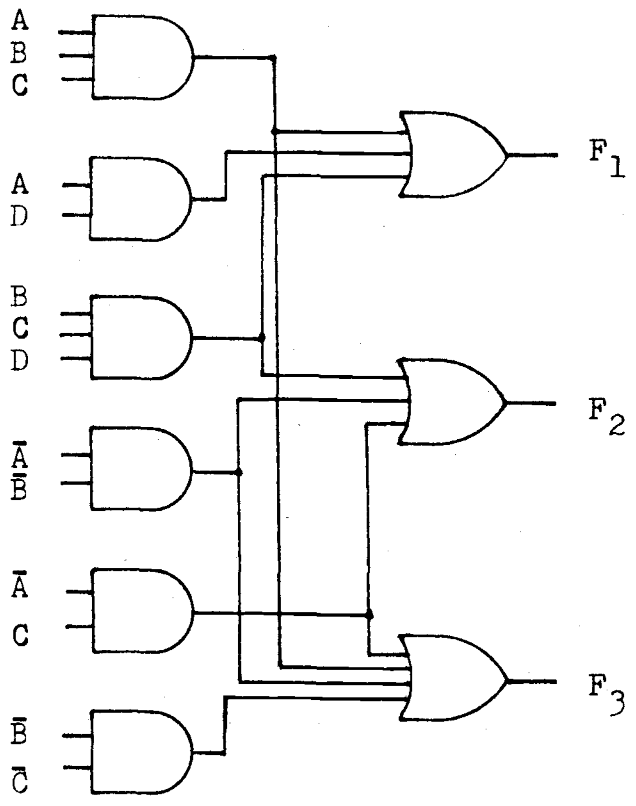


Figure 2.8 Multiple-output minimization result, only 6 AND gates and 3 OR gates are required

the total number of gates required to 6 AND gates and 3 OR gates, this is shown in Fig. 2.8.

C. Quine-McCluskey method

Graphic simplification method becomes less convenient and less effective as the number of variables increases. In practice we often encounter logic design problem that involves variables more than four, in this situation the Karnaugh map method is difficult to apply. The Quine-McCluskey (6) minimization method is applicable to both single-output and multiple-output switching functions. In particular, a digital computer can be used to aid the minimization processes of switching functions with more than four variables.

The essence of Quine-McCluskey tabular minimization method is:

a). All prime implicants are developed by iteratively examining all of the minterms and the reduced terms,

example:

$F(A,B,C)=\Sigma(0,1,4,6)=\Sigma(000,001,100,110)$

000 is examined with 001,100, since they differ in only one variable, two combined terms 00-and-00 are formed (-means 0 or 1), now 001 is examined with 110, a 1-0 is formed, thus

$F(A,B,C)=(0,1)+(0,4)+(4,6)$

b). Duplicates of prime implicants must be detected and deleted.

c). A prime implicant table is constructed. Essential PI's are removed.

D). A covering problem(5) must be solved which involves the deletion of dominated rows and dominating columns as well as the selection of essential or pseudo-essential PI's.

Since part of Quine-McCluskey's idea will be used, the details of which will be described in subsequent chapters.

D. Topological method (10)

Some investigators in the field of switching theory in the early fifty's, such as (10) and (12), viewed the problem of Boolean minimization as that of operating upon the cells of an n-cube complex. The 3-variable function, $F(A,B,C)=\Sigma(3,4,5,7)$ can be viewed as the cell system shown in Fig. 2-9. The minterms are viewed as <u>vertices</u> of an n-cube, where for the 3-variable function n=3. The vertices identified in Fig. 2-9 described three 1-cells. By definition (12), the unit n-cube is made up of cells described as follows:

0-cells or vertex	-- a point
1-cell	-- a line segment
2-cell	-- a quadrilateral
3-cell	-- a hexahedron
k-cell	-- a k-dimensional figure, $k \leq n$

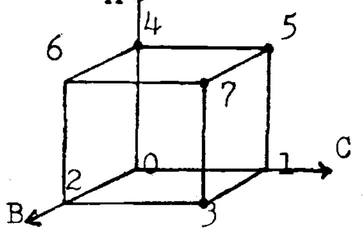


Figure 2.9

Two cells of the same dimension are said to be adjacent if their representations are identical in all but one coordinate position. For example, 0-cells 3 and 7 are adjacent. The basic minimization idea is the same as graphic simplification if the adjacency of each cell is identified. Thus in Fig. 2.9 the 0-cells 3,7;5,7 and 4,5 are adjacent pairs. They are combined to form 1-cells (4,5), (5,7), (3,7). Since (4,5) and (3,7) are each essential with respect to vertices 4 and 3, thus (5,7) is not required, the final solution for this function is F(A,B,C)= (3,7)+(4,5).

E. McKinney's new approach

Melvin Howard McKinney, Jr.(7) in his dissertation presented an algorithm as well as new ideas for the minimization of single-output switching function without finding all prime implicants. The algorithm is directed toward the determination of a minimum covering while performing the least possible processing. Only those prime implicants that have to be produced while finding one minimum-cost cover for the function are idemtified.

The input minterms are categorized into three subsets. Minterm m_i is a member of:

i) a true form (TF) if $F(m_i)=1$

ii) a false form (FF) if $F(m_i)=0$

iii) a redundant form (XF) if $F(m_i)$ is not specified.

The algorithm first derives RAD's (Required-Adjacency-Direction's) for each TF Select one TF as a starting point and combine it with these RAD's to form the largest PI that covers it. Once a PI is identified all TF's covered by this PI become XF's. Then select the next uncovered TF as a new starting point and perform the previous operations. This process continues until all TF's become XF's.

In an n-variable function every minterm is adjacent to n minterm. The <u>direction of adjancency</u> from a minterm m_i is defined as the set of signed integers $\{a_k\} = \{\pm 2^k \mid k=0,1,2,\ldots n-1\}$, which,when added to the value of i, gives the subscripts of the n adjacent minterms of m_i. Each a_k is positive if the kth bit of the binary form for i is zero and is negative if that bit is one. If m_i is a TF then the directions of adjacency leading to another TF or XF is a RAD of m_i. The number of RAD's immediately determines the order of largest PI that may possibly cover that TF. The search for PI's begins at a TF having the fewest RAD's for easy identification of selectable PI's.

example:

Minimize $F(A,B,C,D)=\sum(0,1,4,5,12,13,16,17)$. The function's RAD list is in Fig. 2.10 (a). Since all TF's have two RAD's, thus m_0 is arbitrarily selected as the first TF to be expanded. Its RAD tree is shown in Fig. 2.10(b). The search succeeds after all RAD's have been used. Thus PI1(0,1,4,5) is identified.

Now TF's 0, 1, 4 and 5 becomes XF's. The next available TF is 12, again search succeeds after all RAD's have been used, PI2(12,13,16,17) is identified. Since all TF's have been covered, the minimized form is

F(A,B,C,D)=(0,1,4,5)+(12,13,16,17)

It has been proved in McKinney's dissertation that the new technique is efficient in the minimization of single-output switching functions. Interested readers are suggested to read (7) for more information.

It is the goal of this thesis to modify this method to use it on multiple-output functions and see how well it performs.

TF	RAD
0	+1,+4
1	-1,+4
4	+1,-4
5	-1,-4
12	+1,+4
13	-1,+4
16	+1,-4
17	-1,-4

(a)

$$\begin{tikzcd}
(0) \arrow[d, "+1"'] & (12) \arrow[d, "+1"'] \\
(0,1) \arrow[d, "+4"'] & (12,13) \arrow[d, "+4"'] \\
(0,1,4,5)\ PI_1 & (12,13,16,17)\ PI_2
\end{tikzcd}$$

(b)

Figure 2.10 (a) Required-Adjacent-Directions (b) RAD tree

III. The Algorithm

This chapter explains the MOMIN algorithm in detail by first examining some definition and terminology in section A. In section B, a cost function is defined. Section C overviews each step used in the MOMIN algorithm. Sections D to H describe these steps in detail.

A. Definition and terminology

Following is a list of terminology used in subsequent sections.

1) An n-variable switching function possesses 2^n different canonical terms called minterms. Each minterm must fall into one of the three categories:

   (a) A <u>true form</u> (TF) is a minterm that must be covered.

   (b) A <u>false form</u> (FF) is a minterm that can not be covered.

   (c) A <u>redundant form</u> (XF) is a minterm that need not be covered, but may be covered to enlarge a cell if possible.

   In other words, any switching function is a combination of TFs, XFs and FFs. e.g. $F(A,B,C)=\sum(0,2,3,4)+d(1,6)$ expresses that a 3-variable switching function is the combination of

$$\begin{cases} \text{4 TFs-0,2,3 and 4} \\ \text{2 XFs-1 and 6} \\ \text{2 FFs-5 and 7} \end{cases}$$

Usually FFs are not listed, since once TFs and XFs of a switching function are specified, FFs become self-evident.

2) Output function: Any output expression of a multiple-output switching network is an output function.

3) Subfunction: The intersection of any 2,3,...or m outputs of a m-output switching functions, in addition to the m outputs, form the subfunction set. There are $m+\binom{m}{2}+\binom{m}{3}+,..\binom{m}{m}=2^m-1$ subfunctions for a m-output switching function. Example: A 2 outputs switching function; $F_1=\sum(0,1,3,4,6)$, $F_2=\sum(0,2,4,5,6)$, then F_{12}, the intersection of F_1 and F_2, is given by $F_{12}=\sum(0,4,6)$. F_1, F_2 and F_{12} are subfunctions of this multiple-output switching function.

4) Lower-level subfunction: A subfunction is said to be lower in level with respect to other subfunctions if it is the intersection of more outputs. Example, F_{123} is derived from the intersection

of F_1, F_2 and F_3, while F_{12} is derived from the intersection of F_1 and F_2, thus F_{123} is lower in level than F_{12}.

5) <u>Higher-level subfunction</u>: F_{12} is a higher-level subfunction of F_{123} in the above example.

6) <u>RAD</u>: Any minterm m_i is adjacent to another minterm m_j if the input combinations that they represent differ in only one variable. e.g. For a 3-variable function $F(X,Y,Z)$ there are eight minterms. They are:

XYZ, $\bar{X}YZ$, $X\bar{Y}Z$, $XY\bar{Z}$, $\bar{X}\bar{Y}Z$, $\bar{X}Y\bar{Z}$, $X\bar{Y}\bar{Z}$ and $\bar{X}\bar{Y}\bar{Z}$

XYZ is adjacent to $\bar{X}YZ$, $X\bar{Y}Z$ and $XY\bar{Z}$.

$\bar{X}YZ$ is adjacent to XYZ, $\bar{X}\bar{Y}Z$ and $\bar{X}Y\bar{Z}$.

In Fig. 3.1, they are represented in binary form, it is easy to find their adjacencies.

Every minterm of a n-variable switching function is adjacent to n other minterm, the <u>directions of adjacency</u> is defined(7) as the set of integers $\{a_k\}=\{2^k | k=0,1,2,\ldots n-1\}$, which, when exclusive-ored with the value of i, gives the subscripts of the n minterms that are adjacent to m_i. For example, with $n=3$, the $\{a_k\}$ set corresponding to $m_5=X\bar{Y}Z=101$ is $\{1,2,4\}$. When exclusive-ored with $i=5$, defines the subscripts of the 3 minterms that are adjacent to m_5; they are m_1, m_4 and m_7.

A direction of adjacency leading from a TF to another TF or XF is defined as a <u>Required-Adjacency Direction</u> or <u>RAD</u>.

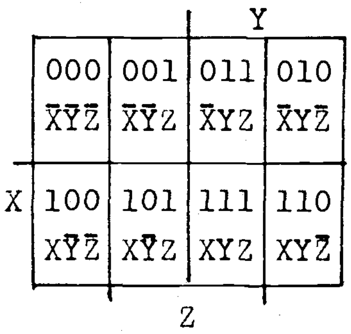
7) <u>Search tree</u> method: A new technique, developed by McKinney(7) used to find PIs in the minimization of single-output functions is used in the minimization of multiple-output switching functions. The RADs are used to expand the origin TF by successively exclusive-oring the RADs to the subscript of that TF. Initially, it produces a pair of TFs (or a TF-XF pair) that defines a 1-cell(6) containing the origin TF. Ex-oring a second RAD with the subscripts of each member of the pair generates the subscripts of additional vertices that, composed of a set of four minterms, defines a 2-cell that is subsumed by the origin TF. Adding additional RADs generates the subscripts of the minterms of higher ordered cubes covering the origin TF. Through this process, higher cells are defined. The Exclusive-or of each RAD doubles the number of minterms. To decide if a newly defined cell exists, it is only necessary to determine if the additional minterms produced when the last RAD was Exclusive-ored to the subscripts of the previous set of minterms are all TFs or XFs. Any FF appeared in the doubled set

indicates that the previous cell was a PI and terminates the search along that particular combination of RAD's. As shown in Fig. 3.2, the search starts from an uncovered TF. A 1-cell (0,1) which contains 0 , the origin TF, is derived by exclusive-or 0 with RAD 1, Exclusive-oring a second RAD 4 with (0,1), a 2-cell is derived. Since none of its minterms is FF, and since no more RAD's of TF 0 are uncircled, we have a PI (0,1,4,5).

8) <u>Related lower-level subfunction</u>: Subfunctions that are intersection of a subfunction, say F_s, and other output functions are related lower-level subfunctions to subfunction F_s. Example: For a four-output switching function; F_{123}, F_{124}, F_{1234} are related lower-level subfunctions of F_{12}.

9) <u>Related higher-level subfunction</u>: F_1, F_2, F_4, F_{12}, F_{14} and F_{24} are higher-level subfunctions of F_{124} for a four-output function.

10) <u>Circle RAD</u>: It is necessary to circle all used RAD's in the subfunction, for which PIs are being searched, and all related higher-level subfunctions to avoid repeated derivation of already derived PIs. In Fig. 3.3 RAD 2 of TFs 0 and 2



		Y	
000 $\bar{X}\bar{Y}\bar{Z}$	001 $\bar{X}\bar{Y}Z$	011 $\bar{X}YZ$	010 $\bar{X}Y\bar{Z}$
100 $X\bar{Y}\bar{Z}$	101 $X\bar{Y}Z$	111 XYZ	110 $XY\bar{Z}$

Figure 3.1 3-variable Karnaugh map

Fun	RAD
0	①,④
1	①,2,④
3	2,4
4	①,④
5	①,2,④
7	2,4

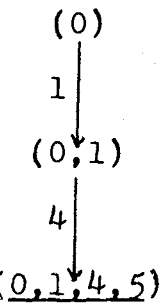
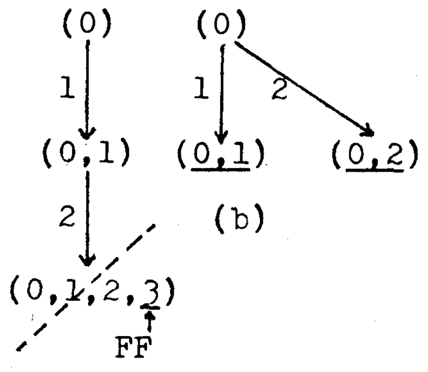


Figure 3.2 Search for a new PI

F_1	RAD	F_2	RAD	F_{12}	RAD
0	1,②	0	②,4	0	②
1	1,2	2	②,4	2	②
2	1,②	4	2,4		
3	1,2	6	2,4		

Figure 3.3 Circle RAD's

Fun	RAD
0	1,2
1	1,4
2	2
5	4



(a)

Figure 3. 4 Branching

F_{12}	RAD
0	②,4
2	②,4
4	②,4
6	②,4

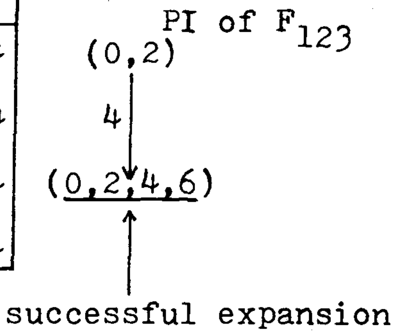


Figure 3.5 Expansion

in F_{12} have been used in deriving the PI (0,2), it is then circled as shown.

11) Covered TF: Any TF that is included in already selected essential/pseudo-essential PIs is a covered TF. Example: TF 0 is covered by essential /pseudo-essential PI (0,1,2,3).

12) Branching: Whenever FFs appear in searching for new PIs, branching technique is used. Fig. 3.4 shows the RAD column of a subfunction. False form 3 is found in searching and is so discarded as shown in Fig. 3.4 (a), branching is shown in Fig. 3.4 (b), where a new PI (0,2) is derived.

13) Empty: A subfunction is said to be empty if all of its TFs are covered.

14) Non-left: A subfunction is said to be non-left if all of its RAD's are circled.

15) Expansion: IF a subfunction is empty but not non-left, expand PIs from related lower-level subfunctions with uncircled RAD's. In Fig. 3.5, F_{12} is empty but not non-left, if (0,2) is an essential/pseudo-essential PI of F_{123}, then it is expanded by using RAD 2, the unused RAD of TF 0 in F_{12}. Expansion succeeded if all resultant minterms are either TF or XF, otherwise expansion failed.

16) Self-construct PI: If expansion failed, derive

new PI within the subfunction. Fig. 3.6 depicts this operation. Expansion from F_{123} failed, then self-construct the PI (1,5).

17) Pseudo-essential PI: Any non-essential PI is said to be pseudo-essential if after row/column deletions it cover at least one TF that is not covered by any other PI on the current PI table.

18) Dominated row: A row X of a PI table is said to dominate another row Y of that table if X covers every TF covered by Y. Row Y is called dominated row.

19) Dominating column: A column P in a PI table is said to dominate another column Q of that table if P has an "X" in every row in which Q has an "X".. Column P is called dominating column.

20) Same degree of minimum: Two minimized functions are said to have the same degree of minimum if they have either (i) The same switching expressions(same number of PIs and same PIs) or (ii) The same number of PI and the same size PIs. e.g. PI (0,1,2,3) and PI (3,7,13,17) have the same size but PI (0,4) and PI (0,2,4,6) have different size.

F_{12}	RAD
1	4,⑩
4	①
5	①,4
11	⑩

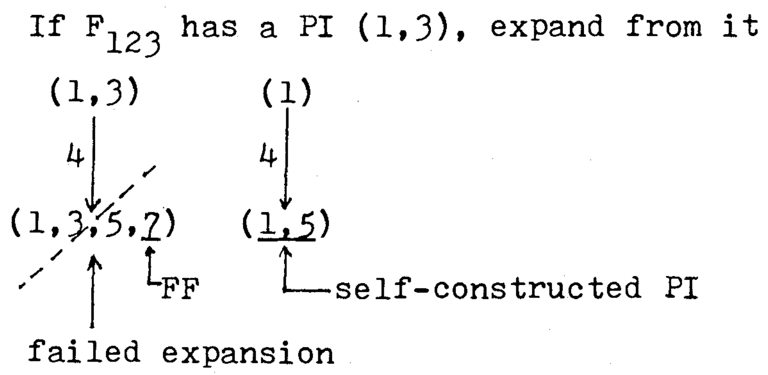


Figure 3.6 Self-construct PI

F_1	RAD	F_2	RAD	F_3	RAD
0	1,2,10	0	2,4,10	0	2
1	1,4,10	2	1,2,10	2	2,4
2	2,4,10	3	1,4	6	1,2,4,10
6	4	5	1,2	7	1,10
10	1,2,10	7	2,4,10	13	4
11	1,2,10	10	2,4,10	14	1,2,10
13	1,2,4	14	1,4,10	16	1,2,10
17	4	17	2,10	*4	
*5		*4		*15	
*12		*12		*17	

Figure 3.8 RAD's for a 3-output 4-variable function
* indicates don't care(XF)

B. Cost function

Before developing a minimization procedure for multiple-output switching circuits, it is necessary to define a criterion to measure the network cost. There are various definitions for cost functions. Three of those frequently used definitions (2) are introduced here.

Let F_1, F_2,.. F_m be the set of switching expressions describing a multiple-output switching circuit and let T_1, T_2, ... T_p be the set of all distinct terms appearing in the m output expressions.

1) If L_i denotes the number of literals in the term T_i, then the cost of the multiple-output switching circuit is given by the numerical quantity $C_1 = \sum_{i=1}^{p} L_i$

(This counts the number of inputs to AND gates in the two-level AND-OR network).

Example, A 3-output 3-variable switching circuit has the expressions:

$$F_1(A,B,C)=AB+\bar{B}C+\bar{A}\bar{B}\bar{C}$$
$$F_2(A,B,C)=\bar{A}\bar{B}\bar{C}$$
$$F_3(A,B,C)=B+\bar{A}\bar{B}\bar{C}$$

There are four distinct terms, i.e. $T_1=AB$ in
$T_2=\bar{B}C$
$T_3=\bar{A}\bar{B}\bar{C}$
$T_4=B$

these expressions.. Then $L_1=2$, $L_2=2$, $L_3=3$, $L_4=1$. The cost of this multiple-output switching circuit is given by

$$C_1 = \sum_{i=1}^{4} L_i=2+2+3+1=8$$

2) The cost of the multiple-output switching circuit is $C_2=p$ (The number of distinct AND gates in the representation).

Example, A 3-output 3-variable switching circuit has the expressions:

$$F_1(A,B,C)=AB+\bar{B}C+\bar{A}\bar{B}\bar{C}$$
$$F_2(A,B,C)=\bar{A}\bar{B}\bar{C}$$
$$F_3(A,B,C)=B+\bar{A}\bar{B}\bar{C}$$

The cost of this 3-output switching circuit is given by the number of distinct terms in the representation which is 4.

3) The cost of a multiple-output switching circuit is given by the number of gate inputs appearing in the representation. The number of gate inputs can be calculated as follows: Let A_i equal the number of terms in F_i unless there is only a single term, in which case let A_i equal 0. Also, let B_j equal the number of literals in the term T_i unless the term consists of a single literal, in which case let B_j equal 0. The number of gate inputs in the representation of the multiple-output switching circuit as given by the numerical quantity $C_3=\sum_{i=1}^{m} A_i+\sum_{j=1}^{p} B_j$. (The first term indicates total number of inputs to OR gates, the second term is the number of inputs to AND gates). Example, A 3-output 3-variable switching circuit has the expression:

$$F_1(A,B,C)=AC+\bar{B}C+\bar{A}\bar{B}\bar{C}$$
$$F_2(A,B,C)=\bar{A}\bar{B}\bar{C}$$
$$F_3(A,B,C)=B+\bar{A}\bar{B}\bar{C}$$

There are four distinct terms, i.e. $T_1=AC$

$T_2=\bar{B}C$

$T_3=\bar{A}\bar{B}\bar{C}$

$T_4=B$ in

these expressions. Then $A_1=3$, $A_2=0$, $A_3=2$; $B_1=2$, $B_2=2$, $B_3=3$, $B_4=0$. The cost of this 3-output switching circuit is given by $C_3=(3+0+2)+(2+2+3+0)$

$=5+7$

$=12$

Since the rapid growth of advanced IC technology assures the product cost increases slightly as the internal connection does, it is not necessary to include internal connection costs when designing chips. Comparatively, the external connection cost becomes important. Thus, a fourth cost function that measures the external connection cost is selected and used in this thesis.

4) The cost of a multiple-output network is given by $C_4=\sum_{i=1}^{p} L_i+m$ (where the first term indicates the number of inputs to AND gates and m is the number of outputs of OR gates).

Example, A 3-output 3-variable switching function has the expressions: $F_1(A,B,C)=AB+\bar{B}C+\bar{A}\bar{B}\bar{C}$

$F_2(A,B,C)=\bar{A}\bar{B}\bar{C}$

$F_3(A,B,C)=B+\bar{A}\bar{B}\bar{C}$

It has 4 distinct terms, i.e. $T_1=AB$, $T_2=\bar{B}C$, $T_3=\bar{A}\bar{B}\bar{C}$, $T_4=B$, then, $L_1=2$, $L_2=2$, $L_3=3$, $L_4=1$.

The cost of this multiple-output switching function is given by $C_4=(2+2+3+1)+3$

$=11$

The switching function in this thesis is represented in octal number, such as, $F_1(A,B,C,D)=(12,13,16,17)+(2,3,12,13)+(0,1)$, $F_2(A,B,C,D)=(0,1)$, $F_3(A,B,C,D)=(4,5,6,7,14,15,16,17)+(0,1)$. A formula is given to compute costs easily:

$$COST=\sum_{i=1}^{p}(N-\log_2 A_i)+m$$

N: number of input variables

A_i: number of TFs and XFs in T_i

m: number of outputs

In above example,	$T_1=(12,13,16,17)$	$A_1=4$
	$T_2=(2,3,12,13)$	$A_2=4$
	$T_3=(0,1)$	$A_3=2$
	$T_4=(4,5,6,7,14,15,16,17)$	$A_4=8$

$$\begin{aligned}COST&=2+2+3+1+3\\&=11\end{aligned}$$

C. Overview of the algorithm

The steps performed by the MOMIN algorithm are summarized below. They will be described in detail in subsequent sections. Fig. 3.7 is a general flow diagram of the steps taken and listed below. Some more flow diagrams of those major steps will be seen in later sections.

STEP 1: Obtain the multiple-output switching functions in numerical form.

Example: A 2-output 3-variable switching functions are $F_1(A,B,C)=\prod(0,1,4,7)+d(2,6)$

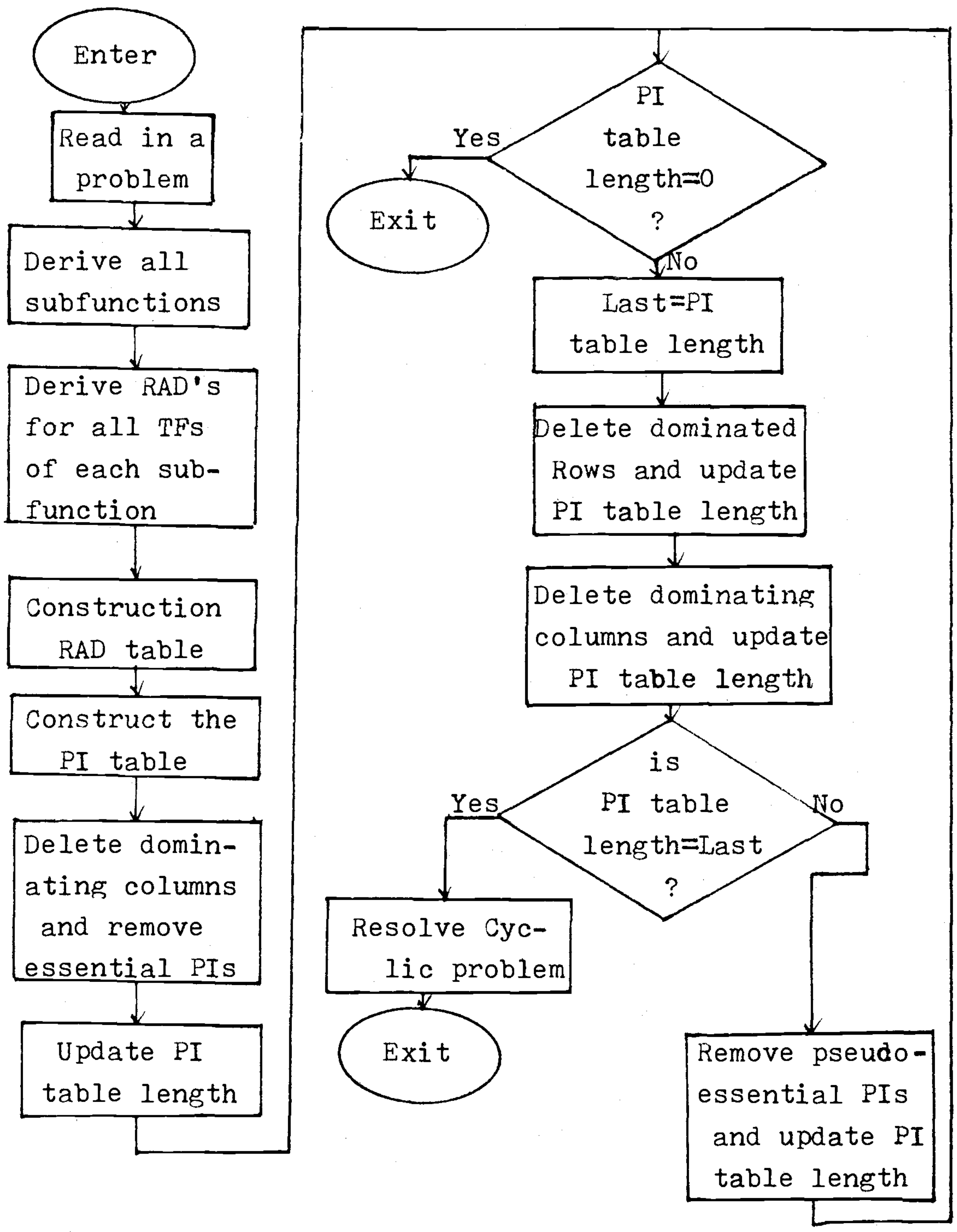

Enter
Read in a problem
Derive all subfunctions
Derive RAD's for all TFs of each sub-function
Construction RAD table
Construct the PI table
Delete domin-ating columns and remove essential PIs
Update PI table length
PI table length=0 ?
Yes
Exit
No
Last=PI table length
Delete dominated Rows and update PI table length
Delete dominating columns and update PI table length
is PI table length=Last ?
Yes
No
Resolve Cyc-lic problem
Exit
Remove pseudo-essential PIs and update PI table length

Figure 3.7 General flow of algorithm

$$F_2(A,B,C)=\sum(1,4,5,6)+d(0)$$

This expresses that (a) Output F_1 consists of four TFs and two XFs; they are 0, 1, 4, 7 and 2, 6, respectively. (b) Output F_2 consists of four TFs and one XF; they are 1, 4, 5, 6 and 0, respectively.

STEP 2: Determine all subfunctions and derive RAD's for all TFs of each subfunction. Note that output functions F_1 and F_2 are also subfunctions.

STEP 3: Construct a complete RAD table including all the subfunctions.

STEP 4: Construct the PI table with a search method.

STEP 5: Removing essential PIs and corresponding columns.

STEP 6: Recursive selection of pseudo-essential PIs. (This is essentially a covering problem)

STEP 7: If PI table is empty, GO TO STEP 10, otherwise GO TO step 8.

STEP 8: If any row or column deletion is possible, GO TO STEP 9.

STEP 10: Algorithm terminates.

D. Construction of the RAD table

As mentioned earlier, an RAD expresses a relation between a pair of adjacent TFs; or a TF-XF pair. They are numbers as $2^r(r=0,1,...,n-1)$ or in octal form RAD=1, 2,4,10,20,40,...;where n is the number of input variables.

1) RAD's for outputs F_i: Exclusive-or a TF with any other TF or XF in the same function and check the result. If it is equal to $2^r(r=0,1,2,...,n-1)$, then the TF has a RAD, 2^r.

Example, A 3-output, 4-variable switching circuit has the expression:

$$F_1=\sum m(0,1,2,6,10,11,13,17)+d(5,12)$$
$$F_2=\sum m(0,2,3,5,7,10,14,17)+d(4,12)$$
$$F_3=\sum m(0,2,6,7,13,14,16)+d(4,15,17)$$

In order to find RAD's for TF 1 of F_1, exclusive-or (X-OR) it with all other TFs and XFs in F_1:

TF 1 X-OR with $0=1=2^0$

TF 1 X-OR with $2=3$

TF 1 X-OR with $6=7$

TF 1 X-OR with $10=11$

TF 1 X-OR with $11=10=2^3$

TF 1 X-OR with $13=12$

TF 1 X-OR with $17=16$

TF 1 X-OR with $5=4=2^2$

TF 1 X-OR with $12=13$

TFs 0 and 11, XF 5 are adjacent to TF 1, thus 1, 10 and 4 are RAD's of TF 1; using the same logic operation, RAD's can be determined for all other TFs as shown in Fig. 3.8

2) RAD's for subfunctions $F_{ij}, F_{ijk}, \ldots$

If F_r and F_s are any multiple-output switching expressions, $r \neq s$, and if $TF(F_r)$, $TF(F_s)$, $TF(F_{rs})$, $XF(F_r)$, $XF(F_s)$ and $XF(F_{rs})$ represent the TF and XF of F_r, F_s, F_{rs}, respectively, then, $TF(F_{rs}) = TF(F_r)TF(F_s)+TF(F_r)XF(F_s)+XF(F_r)TF(F_s)$; $XF(F_{rs}) = XF(F_r)XF(F_s)$. Thus the subfunction $F_{rs} = TF(F_{rs}) + XF(F_{rs})$.

Example, Let r=1, s=2 for the example in section 1),

$$TF(F_1)=\sum(0,1,2,6,10,11,13,17)$$
$$XF(F_1)=\sum(5,12)$$
$$TF(F_2)=\sum(0,2,3,5,7,10,14,17)$$
$$XF(F_2)=\sum(4,12)$$

Then
$$\begin{aligned} TF(F_{12}) &= TF(F_1)TF(F_2)+TF(F_1)XF(F_2)+ \\ &\quad XF(F_1)TF(F_2) \\ &= \sum(0,2,5,10,17) \end{aligned}$$
$$\begin{aligned} XF(F_{12}) &= XF(F_1)XF(F_2) \\ &= (12) \end{aligned}$$

Thus subfunction $F_{12}=\sum(0,2,5,10,17)+d(12)$.

For a m-output network there are 2^m-1 subfunctions (Because there are one out of m, two out of m,....., m out of m choices, the total number of subfunctions is given by $\binom{m}{m}+\binom{m}{m-1}+\ldots\binom{m}{1}=2^m-1$).

Example, For m=3, there are:

1 out of 3 choices--$\binom{3}{1}=3$. i.e. F_1, F_2, F_3

2 out of 3 choices--$\binom{3}{2}=3$. i.e. F_{12}, F_{23} and F_{13}

3 out of 3 choices--$\binom{3}{3}=1$. i.e. F_{123}

A total of seven subfunctions are found.

Treating each subfunction as single output function and using the technique described in section (1), a complete RAD table can thus be formed as shown in Fig. 3.9.

F_1	RAD	F_2	RAD	F_3	RAD	F_{12}	RAD	F_{13}	RAD	F_{23}	RAD	F_{123}	RAD
0	1,2,10	0	2,4,10	0	2	0	2,10	0	2	0	2,4	0	2
1	1,4,10	2	1,2,10	2	2,4	2	2,10	2	2,4	2	2	2	2
2	2,4,10	3	1,4	6	1,2,4,10	5	$^{\#}$X	6	4	7	10	17	$^{\#}$X
6	4	5	1,2	7	1,10	10	2,10	13	4	14	10		
10	1,2,10	7	2,4,10	13	4	17	$^{\#}$X	17	4	17	2,10		
11	1,2,10	10	2,4,10	14	1,2,10	*12				*4			
13	1,2,4	14	2,4,10	16	1,2,10								
17	4	17	2,10	*4									
*5		*4		*15									
*12		*12		*17									

Figure 3.9 Complete RAD table for a 3-output 4-variable function

*: XF(don't care)

#: if a TF has no RAD, a "X" is entered in the RAD column

Note: XFs(don't cares) are also listed but no RAD is entered in the RAD columns for them, because RAD is meaningless and useless to a XF.

E. Construction of the PI table

The derivation of multiple-output prime implicants using search trees has two versions. (A) Derive necessary multiple-output prime implicants. (B) Derive all multiple-output prime implicants. The choice depends on circuit design requirements.

If size of product instead of design cost is more important, i.e. a minimal solution is desired, version (B) is best suited; if design cost is more important, version (A) is recommended, since it gives us a subminimal solution by saving labor spent in deriving a minimal solution and thus reduces design cost.

A typical PI table is of the form as in Fig. 3.10. The first column indicates all subfunctions related to PIs; the second column is a list of prime implicants derived; and the third column and after are TFs of each output. A "X" is entered if the corresponding TF is contained in one PI. The last column indicates cost of that PI. Example, row 2 indicates PI (1,11) is shared by F_1 and F_2, cost is 3.

(A) Derivation of necessary multiple-output prime implicants: The flow diagram in Fig. 3.11 is best suited to help understand the PI table construction algorithm. Necessary details are described below (numbers correspond to the flow diagram).

(1) Begin with the lowest level subfunction.

		F_1							F_2					
Fun	PI	0	1	4	5	6	7	11	1	3	11	13	14	COST
F_{12}	(1,11)		X					X	X		X			3
F_2	(1,3,11,13)								X	X	X	X		2
F_2	(14)												X	3
F_1	(0,1,4,5)	X	X	X	X									2
F_1	(4,5,6,7)			X	X	X	X							2

Figure 3.10 Typical PI table

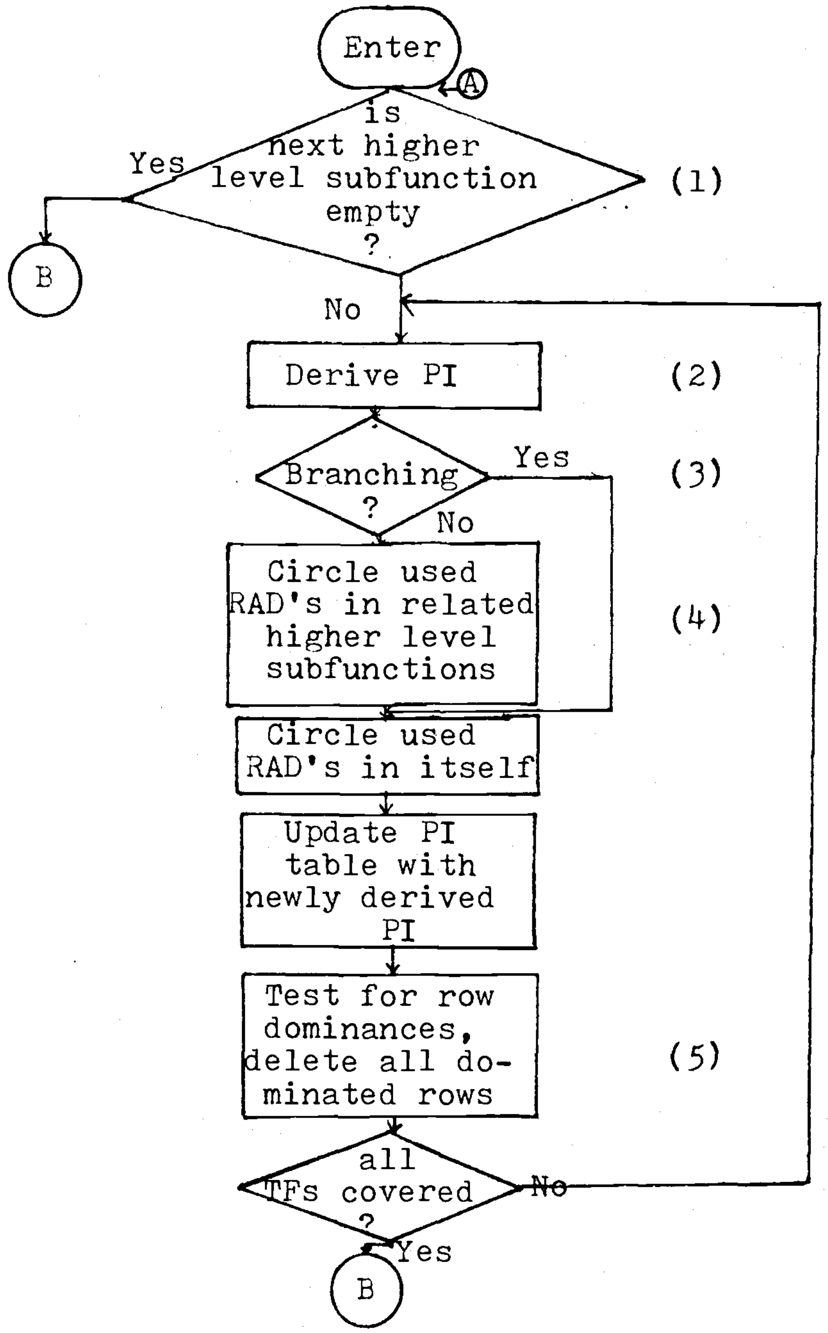

Enter
A
is next higher level subfunction empty ?
Yes
B
No
(1)
Derive PI
(2)
Branching ?
Yes
No
(3)
Circle used RAD's in related higher level subfunctions
(4)
Circle used RAD's in itself
Update PI table with newly derived PI
Test for row dominances, delete all dominated rows
(5)
all TFs covered ?
No
Yes
B

Figure 3.11 Flow diagram for PI table construction version (A)

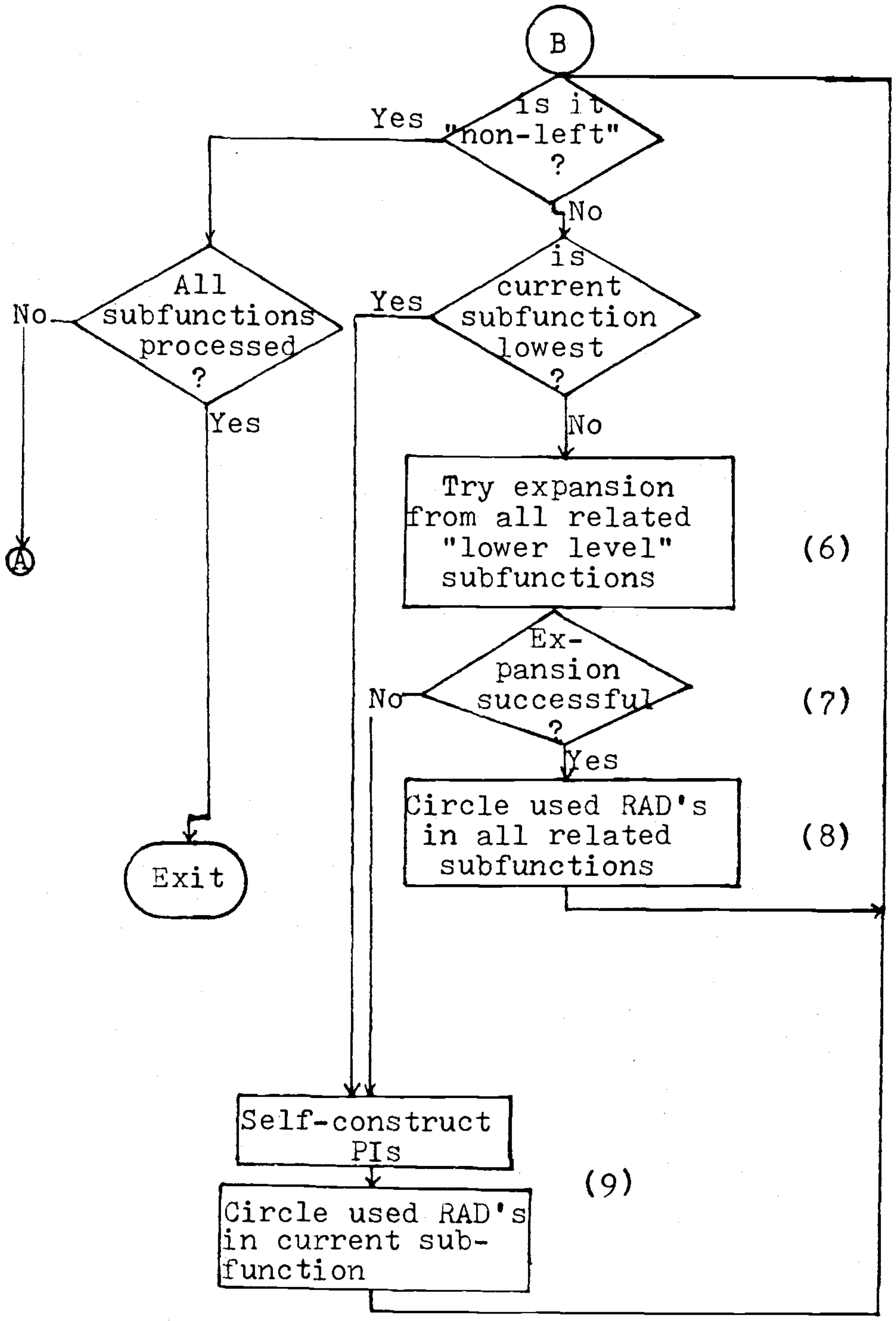

B
Is it "non-left" ?
Yes
No
is current subfunction lowest ?
Yes
No
All subfunctions processed ?
No
Yes
A
Try expansion from all related "lower level" subfunctions
(6)
Expansion successful ?
No
Yes
(7)
Circle used RAD's in all related subfunctions
(8)
Exit
Self-construct PIs
Circle used RAD's in current sub-function
(9)

Figure 3.11 Flow diagram for PI table construction version (A) (continued)

(2) Deriving PIs: If a subfunction has a list of RAD's as:

F_{123}	RAD
1	2
4	X
3	

search tree indicates two PIs

$$\begin{tikzcd}
(1) \arrow[d, "2"'] & (\underline{4}) \\
(\underline{1,3}) &
\end{tikzcd}$$

(since TF 4 has no RAD, it is a PI itself)

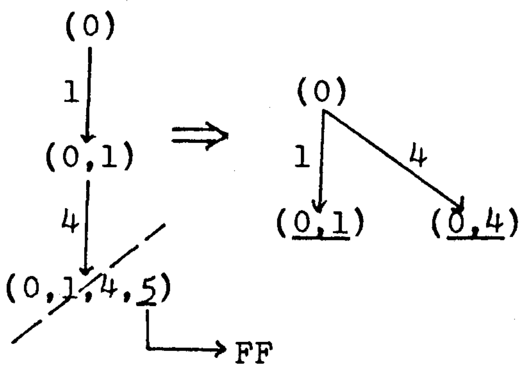
cost for PI $(1,3)=3+\log_2 2=2$

cost for PI $(4) \ =3+\log_2 1=3$

(3) Test for branching: If a subfunction F_{12} has a list of remaining RAD's (uncircled) as shown:

F_{12}	RAD
0	1,4
3	X
4	4
1	

a FF 5 appeared when deriving PI from TF 0. (0,1,4,5) is not a PI of F_{12}, thus the search tree becomes:



We call this branching. Whenever branching occurs, circles are placed only on those RAD's of the current subfunction, as shown above (dotted circles).

(4) Mark used RAD's : Circle RAD's in related higher level subfunctions and itself. In the example of (2), F_{123} has related higher level subfunctions F_1, F_2, F_3, F_{12}, F_{13} and F_{23}, RAD's 2 and "X" are circled in this subfunction and itself.

(5) First stage reduction of the PI table: Each time a new PI is derived, row dominance test is performed to eliminate rows and thus reduce the size of PI table.

Example:

	PI	1	2	3	6	7	0	2	4	6	COST
<u>a</u>	A		X		X			X		X	2
<u>b</u>	B		X	X	X	X		X		X	2

row <u>b</u> dominates row <u>a</u>, thus row <u>a</u> can be eliminated without the possibility of increasing the total cost of the final solution.

(6),(7) and(8) Expand PIs from lower level subfunctions to get larger dimension PIs: If a subfunction, say F_{12}, has the following RAD list, it is easily seen that F_{12} is empty i.e. at least one of the RAD's in each TF is circled. But since TFs 0,1 and 5 have unused RAD's, we try expansion from PIs of the related lower level subfunctions. If F_{123} has a PI

F_1	RAD	F_2	RAD	F_{12}	RAD
0	①,④	0	①,④	0	①,④
1	①,④	1	①,④	1	①,④
2	2	3	2,4	4	①,④
4	①,④	4	①,④	5	1,②,4
5	①,②,④	5	①,②,④	7	
7		7			

(0,4) then we expand it by using unused RAD 1 of TF 0.

$$\begin{tikzcd}(0,4) \arrow[d, "1"'] \\ (0,1,4,5)\end{tikzcd}$$

(successful expansion)

This is a successful expansion. Thus a new PI (0,1,4,5) is derived. Circle RAD's 1 and 4 in related higher level subfunctions and itself (dotted circles in the RAD table).

(9) Developing PIs by self-construction : If TF 5 in the above example does not exist in both F_1 and F_2, then the expansion above would fail, as shown below,

F_{12}	RAD
0	①,④
1	①
4	4
7	

$$\begin{tikzcd}(0,4) \arrow[d, "1"'] \\ (0,1,4,5)\end{tikzcd}$$

(failed expansion)

FF

Whenever expansion fails we derive PI by "self-construction"

$$\begin{tikzcd}(0) \arrow[d, "1"'] \\ (\underline{0,1})\end{tikzcd}$$

then we circle only RAD's in F_{12} (dotted circles in the RAD table).

The example in Fig. 3.12 illustrates an information loss during PI table construction, PI (2,3,12,13) in F_3 is not derivable after the above process, since it is impossible or at least fairly complicated (too much labor is required) to keep track of this kind of information loss. To save design cost, this kind of information is ignored. This is the essence of submnal solution. If this information loss does not occur then we can have a minimal solution through the usage of this subminimal solution process.

(B) Derivation of all multiple-output prime implicants: This approach differs from (A) in that (i) It derives all MOPIs and (ii) It has a simpler algorithm but is much more laborious.

Begin with highest level subfunction. A flow diagram in Fig. 3.13 helps explain the algorithm. Necessary details follow. (Numbers correspond to the flow diagram)

(1) Process subfunctions one at a time from high level to low level subfunctions: e.g.

$F_1 \dashrightarrow F_2 \dashrightarrow F_{12}$

F_3	RAD	F_{13}	RAD	F_{23}	RAD	F_{34}	RAD	F_{123}	RAD	F_{134}	RAD	F_{234}	RAD	F_{1234}	RAD
0	②,10	0	2,10	3	10	0	2	5	X	0	2	3	10	6	X
2	①,②,④,⑩	2	2,4,10	5	X	2	1,2,4	6	X	2	2,4	6	X		
3	①,④,⑩	5	X	6	X	3	1,4,10	10	2	6	4	13	10		
5	2	6	4,10	10	1,2	6	1,4	12	2						
6	①,④,⑩	10	2,10	11	1,2	7	1,4								
7	①,2,④	12	2,4,10	12	1,2	13	10								
10	①,②,4,10	16	4,10	13	1,2,10	14	X								
11	①,②														
12	①,②,④,⑩														
13	①,②,⑩														
14	2,4														
16	2,④,⑩														

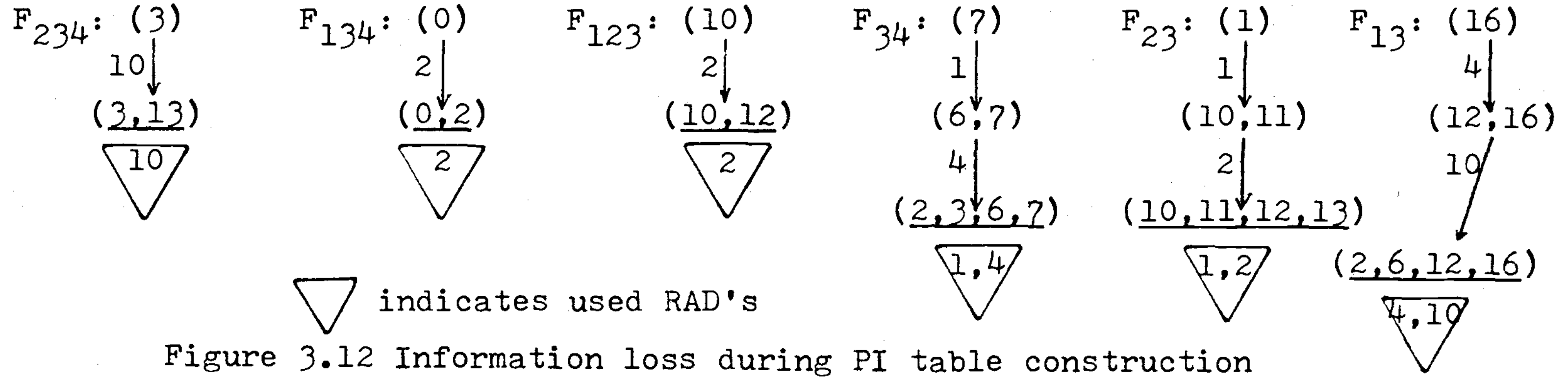


Figure 3.12 Information loss during PI table construction

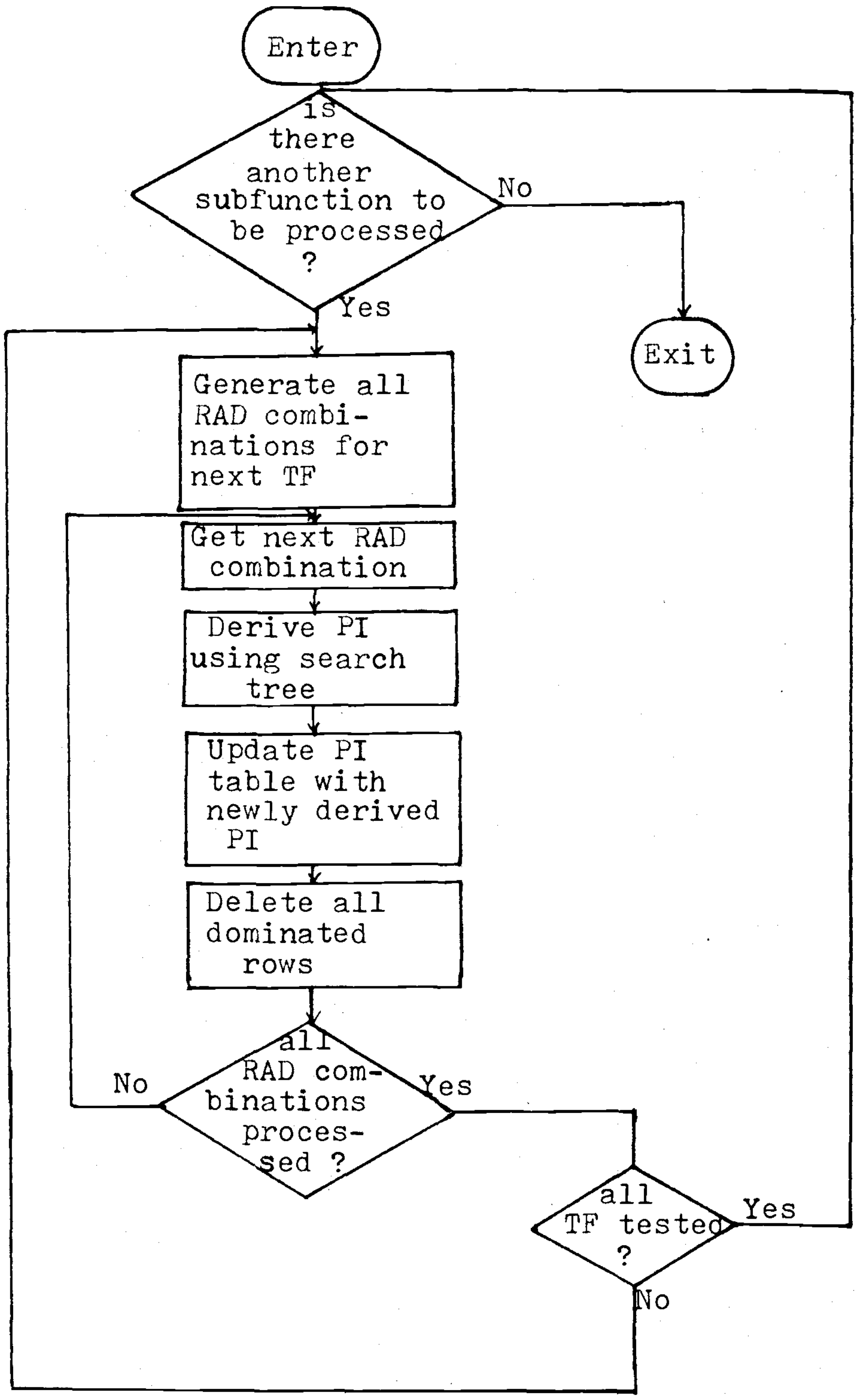

Enter
is there another subfunction to be processed ?
No
Yes
Exit
Generate all RAD combinations for next TF
Get next RAD combination
Derive PI using search tree
Update PI table with newly derived PI
Delete all dominated rows
all RAD combinations processed ?
No
Yes
all TF tested ?
Yes
No

Figure 3.13 Flow diagram for PI table construction version (B)

(2) Generate all RAD combinations for each TF:

If a TF has a list of RAD's as shown,

F_{12}	RAD
4	1,2,4

its combinations are 1; 2; 4; 1,2; 2,4; 1,2,4. They are listed in the order (1,2,4); (1,4); (2,4); (1,2); (4); (2); (1)(reasons follow)

(3) Begin with a new RAD combination.

(4), (5), and (6) Derive PI by using same technique described in version (A): TF 0 has a list of RAD combinations 1,4; 4 and 1. Each combination is tested to derive PIs.

F_{23}	RAD
0	1,4
1	1
4	2,4
5	1,4

$$\begin{tikzcd}
(0) \arrow[d, "1"'] & (0) \arrow[d, "4"'] & (0) \arrow[d, "1"'] \\
(0,1) \arrow[d, "4"'] & \cancel{(\underline{0,4})} & \cancel{(\underline{0,1})} \\
(\underline{0,1,4,5}) & \text{(both are dominated by (0,1,4,5))} \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Since the RAD combinations are in such an order that larger PIs could be derived earlier as seen in the above example, and row dominance is tested each time a new PI is derived, thus eliminate the number of rows in the PI table. (0,1) and (0,4) are deleted right after their derivations.

F. Removal of essential/pseudo-essential PIs

Consider the multiple-output switching functions (6):

$F_1(A,B,C,D)=\sum(1,2,3,5,7,10,11,14,16)$

$F_2(A,B,C,D)=\sum(0,1,2,3,4,6,10,11,12,13)$

$F_3(A,B,C,D)=\sum(1,3,5,7,10,11,14,15,16,17)$

The PI table is shown in Fig. 3.14. Essential prime implicants C, F, G, H, M and all TFs covered by them have been checked off. The reduced PI table, which is obtained by removing the essential PIs and the columns covered by them, is shown in Fig. 3.15 Although none of the PIs in the table is essential, some of them may be removed. For example, row B has "X" in column 11 in both F_1 and F_3, while row D has "X" in columns 10 and 11 in both F_1 and F_3. Since B and D have the same cost, the the removal of row B can not prevent us from finding at least one minimal expression; and since row D covers the TFs covered by row B, it can replace row B in every expression for F_1 and F_3 without affecting its logical value. This is row dominance.

Definition: A row X of a PI table is said to dominate another row Y of that table if X covers every TF covered by Y and the cost of row X is less than or equal to row Y. If row X dominates row Y, then row Y can be deleted from the PI table.

Thus rows B and E in Fig. 3.15 are deleted because they are both dominated by row D. Similarly, row I is removed be-

			F$_1$									F$_2$										F$_3$										
			✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓		✓		
	Fun	PI	1	2	3	5	7	10	11	14	16	0	1	2	3	4	6	10	11	12	13	1	3	5	7	10	11	14	15	16	17	COST
	F_{123}	A	X		X								X		X							X	X									3
	F_{123}	B	X						X				X						X			X					X					3
✓	F_{12}	C		⊗	X									X	X																	3
	F_{123}	D						X	X									X	X							X	X					3
	F_{13}	E						X		X																X		X				3
✓	F_{13}	F								X	⊗																	X		X		3
✓	F_2	G										X		X		⊗	⊗															2
✓	F_{13}	H	X		X	⊗	⊗															X	X	X	X							2
	F_3	I																				X		X			X		X			2
	F_3	J																								X	X	X	X			2
	F_3	K																						X	X				X		X	2
	F_3	L																										X	X	X	X	2
✓	F_2	M										X	X	X	X			X	X	⊗	⊗											1

A=(1,3)	D=(10,11)	G=(0,2,4,6)	J=(10,11,14,15)
B=(1,11)	E=(10,14)	H=(1,3,5,7)	K=(5,7,15,17)
C=(2,3)	F=(14,16)	I=(1,5,11,15)	L=(14,15,16,17)
			M=(0,1,2,3,10,11,12,13)

Figure 3.14 PI table (i) circled X's indicates essential PIs
(ii)all essential PIs and covered TFs have been checked off

cause it is dominated by row J, row L is deleted because it is dominated by row K.

The PI table after deletion of rows B, E, I and L is shown in Fig. 3.16, clearly, row D and row K must be selected since they are the only remaining PIs that cover TFs 10, 11 of F_1, and TFs 15, 17 of F_3, respectively. After this removal of pseudo-essential PIs, the PI table becomes empty. Thus the minimal cost expression for this function consists of PIs C, D, F, G, H, K and M.

$$F_1=C+D+F+H=(2,3)+(10,11)+(14,16)+(1,3,5,7)$$

$$F_2=C+D+G+M=(2,3)+(10,11)+(0,2,4,6)+(0,1,2,3,10,11,12,13)$$

$$F_3=D+F+H+K=(10,11)+(14,16)+(1,3,5,7)+(5,7,15,17)$$

Note that there are two redundant PIs in F_2, (2,3) and (10,11). This is because they are shared PIs. It does not increase the total cost according to our cost function.

Total cost=16+13=19

The prime implicant table can also be reduced by removing certain columns. Consider, for example, column 10 of F_1 and column 10 of F_3 in Fig. 3.15. In order to cover column 10 of F_1, either row D or row E must be selected, then column 10 of F_3 will automatically be covered since it has X's in rows D and E. The reverse is not true, since column 10 of F_3 can also be covered by row J, but this will not cover column 10 of F_1.

		F_1		F_3				
Fun	PI	10	11	10	11	15	17	COST
F_{123}	B		X		X			3
F_{123}	D	X	X	X	X			3
F_{13}	E	X		X				3
F_3	I				X	X		2
F_3	J			X	X	X		2
F_3	K					X	X	2
F_3	L					X	X	2

Figure 3.15 Reduced PI table after all essential PIs and covered TFs have been removed

		F_1		F_3				
Fun	PI	10	11	10	11	15	17	COST
F_{123}	D	Ⓧ	Ⓧ	X	X			3
F_3	J			X	X			2
F_3	K					Ⓧ	Ⓧ	2

Figure 3.16 PI table after row deletion

		F_1		F_3	
Fun	PI	10	11	17	COST
F_{123}	B		X		3
F_{123}	D	X	X		3
F_{13}	E	X			3
F_3	K			X	2
F_3	L			X	2

Figure 3.17 PI table after column deletions

Definition: A column P in a PI table is said to <u>dominate</u> another column Q of that table if P has an "X" in every row in which Q has an "X". Then the dominating column P can be deleted without affecting the search for a minimal expression. In Fig. 3.15, it is clear that columns 10 and 11 of F_3 can be deleted according to the above definition. Also, column 15 of F_3 dominates column 17 of F_3; column 15 of F_3 is deleted. The reduced table is shown in Fig. 3.17. Again, rows B, E and L can be removed because they are dominated by row D and row K, respectively. Thus we have the same final minimal solution as directly derived from row deletions.

Following are a list of steps required to recursively select essential/pseudo-essential PIs:

(1) Remove all dominating columns from the PI table.

(2) Remove all dominated rows from the PI table.

(3) Remove all essential/pseudo-essential PIs from the PI table and include them in their respect function, or include them in two or more functions if they are shared.

(4) Repeat (1) to (3) as many times as needed until every TF of each function is covered by an essential/pseudo-essential PI, or until none of the three steps can be applied. In this case a cyclic problem is encountered.

G. Cyclic problem

It may occur that a PI table has no essential PIs, dominating columns or dominated rows. It can happen either at the beginning or at the end of PI table reduction. The remaining covering problem is known as a cyclic problem.

A typical cyclic problem example is shown in Fig. 3.18, a search tree(7) method is used to solve this problem.

First a search table is constructed and is shown in Fig. 3.19. Columns a, b and c represent TFs of F_1 or F_2. The coverage of PI is shown with a "1" where it covers a TF and a "0" where it does not. Each PI has a cover mask, for example, PI C has a cover mask 011, which indicates that TFs b and c are covered by it.

The search is shown in emanating from the start node "0" in Fig. 3.20. The nodes are shown with the PIs, the covering masks, and their cumulative cost. The cover mask at a node is the cumulative cover masks of all the PIs combined at that node(the OR of all these PI cover masks).

For example, the node AB-111-6 in the tree indicates that PIs A and B are combined at a total cost of six to obtain a full cover of the TFs. A perpendicular line accross a branch indicates terminating of that branch. The paths below a terminated branch are never reached.

The search is performed in a depth-first manner; as shown in Fig. 3.21, pursuing one branch from a node then one

Fun	PI	2	3	7	COST
F_1	A	X	X		4
F_2	B			X	2
F_{12}	C	X			3
F_1	D	X			1
F_{12}	E	X		X	4

Figure 3.18 A cyclic problem

PI	a	b	c	COST
A	1	1	0	4
B	0	0	1	2
C	0	1	1	3
D	1	0	0	1
E	1	0	1	4

Figure 3.19 Search table

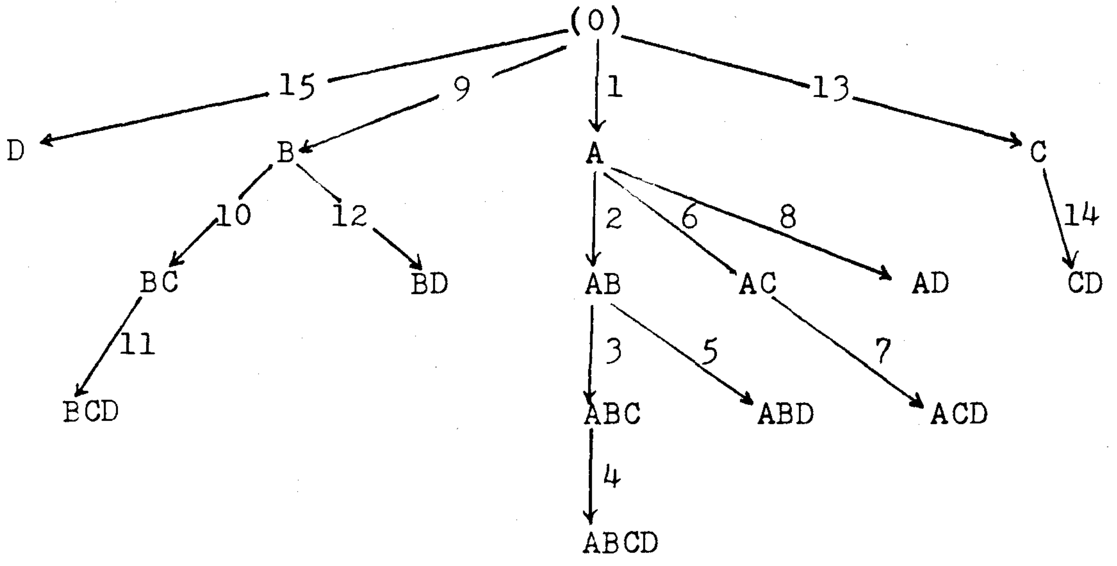


Figure 3.21 A complete search tree of a 4-PI cyclic problem(number correspond to searching sequences)

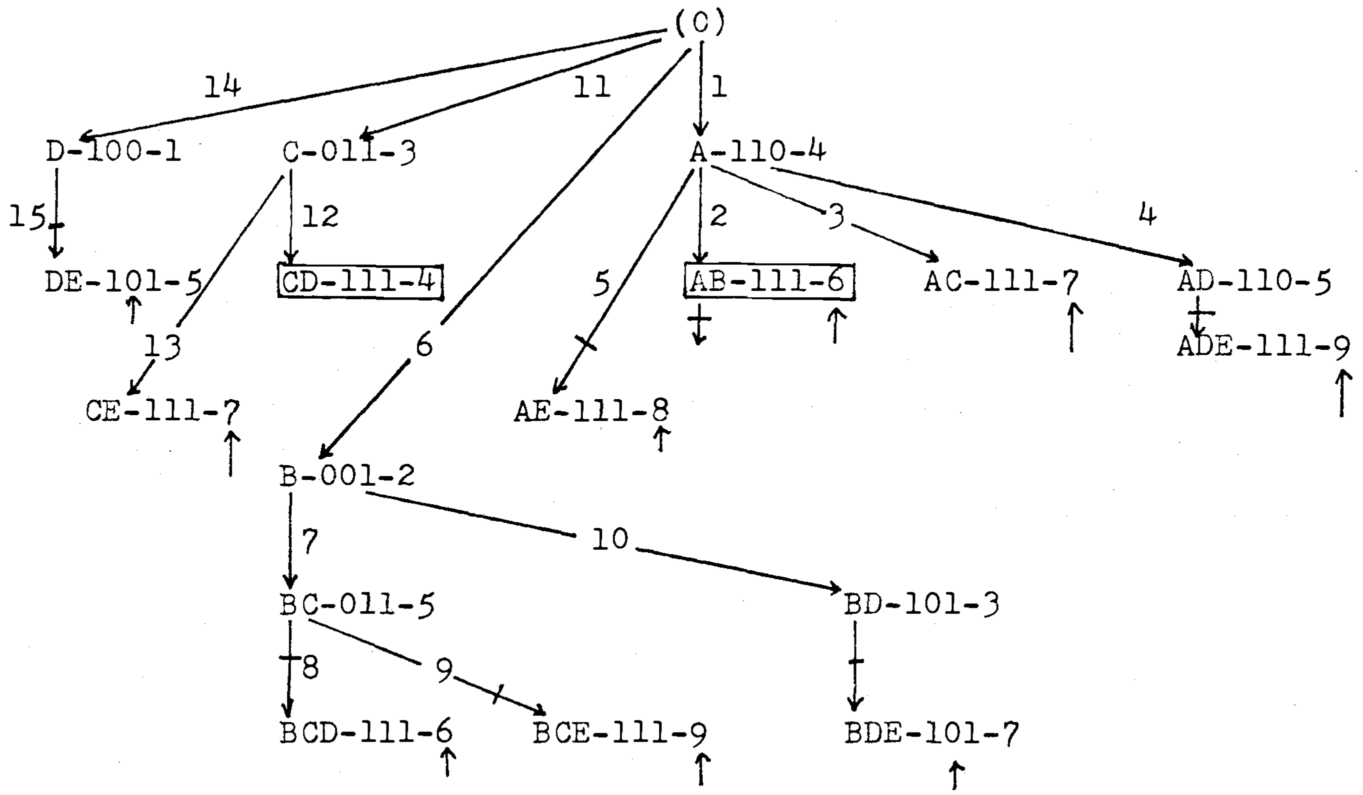


Figure 3.20 Search tree for minimum solution of cyclic problem.

(i) Upward arrows indicate reason of termination.

(ii) Numbers beside the arrows indicate sequence of searching.

from the node that branch leads to, etc. until a branch terminates. For example, nodes AB, ABC, ABCD,... etc.. A is selected first. It has a cover mask 110 and cost of four. Because the cover mask at the node is not 111, B is combined with A(because B is the next PI on list to be selected). Since B has a cost of two and a cover mask of 001, the cost and the cover of AB are six and 111, respectively. Because the cover mask includes all one's, i.e. it covers all TFs, AB is a complete cover and the combination of AB and the cost of six is the best cover thus far into the search.

The branch to AC is discarded because the cumulative cost of AC is seven which exceeds the better cost of six that is already known. Again, the branches to ADE and AE are discarded for the same reason. The search then backs up to node "0" and proceeds to examine the PI combinations not containing A. PI B has a cover mask of 001 and a cost of two. The combination of B and C has a cumulative cover mask of 011 and a cumulative cost of five. Since it does not exceed the already-known lower cost which is six, the search is continued downward. PI D has a cost of one and a cover mask of 100, the combination of BC and D is a complete cover, and the total cumulative cost is six. Since it is no better than the current lower cost cover, node BCD is discarded. Node BCE is also discarded, since it has a cumulative cost of nine. PI D is combined with PI B. Node BD has a cumulative cover mask of 101 and a cost of three, so PI E is added. Node

BDE has a cumulative cover mask of 101 and a cumulative cost of seven, which exceeds the already-known better(lower) cost. BDE is then discarded. Now the search tree backs up to node "0" to examine the PI combinations not containing A and B. PI C has a cover mask of 011 and a cost of three. Since the cumulative cover mask is not 111 but the cumulative cost does not exceed the current lower cost, PI D is combined with C. The cumulative cover mask of node CD is 111 andnd the cumulative cost is four, which is lower than the current best cover. Thus CD is selected as the new best cover. The search tree backs up to node "0" again and to examine PI combinations not containing A, B and C. The cover mask of D is not 111. E is combined with D, since E has a cover mask of 101, and a cost of four. The cost of DE is five, which exceeds the already-known lower cost of four. It is discarded. The search terminates since there are no PIs not already considered. A minimum cost covering of this cyclic problem is, therefore composed of PIs C and D with a total cost of four.

H. Removal of redundancy

Redundancy occurs when one PI is contained in 2 or more other PIs, e.g. $F_1=(4,5)+(1,3,5,7)+(0,2,4,6)$. The PI (4,5) is redundant. It is caused by the selection of essential PIs as well as pseudo-essential PIs. Fig. 3.22 depicts the PI table of a 2-output 4-variable switching circuit after the first-stage deletion of dominated rows during table construction. After deleting the dominating columns, the PI table is shown in Fig. 3.23.

The PI (4,5,14,15) is an essential PI of F_1. It is included in the cover of F_1. The PI (4,14) is an essential PI of F_2. It is included in the cover of F_2, but since the PI (4,14) is shared by F_1, it is also included in the cover of F_1. The cover of F_1 and F_2 becomes:

$$F_1=(4,5,14,15)+(4,14)+\dots$$
$$F_2=(4,14)+\dots$$

(4,14) is redundant to (4,5,14,15) in the cover of F_1.

If a PI is essential or pseudo-essential with respect to both outputs then it may not be redundant. This can be seen from the following examples:

		F_1				F_2						
Fun	PI	4	5	14	15	1	3	4	5	7	14	COST
F_{12}	(4,14)	X		X				X			X	3
F_1	(4,5,14,15)	X	X	X	X							2
F_2	(3,7)						X			X		3
F_2	(1,3,5,7)					X	X		X	X		4
F_{12}	(1,5)		X			X			X			3

Figure 3.22 PI table of a 2-output function,

with $\begin{cases} F_1=\sum(4,5,14,15)+ d(1) \\ F_2=\sum(1,3,4,5,7,14) \end{cases}$

		F_1	F_2			
Fun	PI	15	1	3	4	COST
F_{12}	(4,14)				$\otimes$	3
F_1	(4,5,14,15)	$\otimes$				2
F_2	(3,7)			X		3
F_2	(1,3,5,7)		X	X		4
F_{12}	(1,5)		X			3

Figure 3.23 PI table after first-stage deletion of dominating columns

Example 1 .

		F_1						F_2					
Fun	PI	0	2	4	10	12	14	0	1	2	3	4	COST
F_{12}	(0,2)	X	X					X		X			3
F_{12}	(0,4)	X		X				X				X	3
F_2	(1,3)								X		X		3
F_1	(0,2,10,12)	X	X		X	X							2
F_1	(0,4,10,14)	X		X	X		X						2

After deleting dominating columns this table is reduced to:

		F_1		F_2				
Fun	PI	12	14	1	2	3	4	COST
F_{12}	(0,2)				⊗			3
F_{12}	(0,4)						⊗	3
F_2	(1,3)			⊗		⊗		3
F_1	(0,2,10,12)	⊗						2
F_1	(0,4,10,14)		⊗					2

Now all five PIs become essential i.e.

F_1=(0,2)+(0,4)+(0,2,10,12)+(0,4,10,14)

F_2=(0,2)+(0,4)+(1,3)

Obviously, (0,2),(0,4) in F_1 are redundant.

Example 2: TFs 10,12 and 14 in example 1 are eliminated to show a non-redundant situation.

		F_1			F_2					
Fun	PI	0	2	4	0	1	2	3	4	COST
F_{12}	(0,2)	X	X		X		X			3
F_{12}	(0,4)	X		X	X				X	3
F_2	(1,3)					X		X		3

After deleting dominating columns 0,4 of F_1 and 2 of F_2, the PI table becomes:

		F_1	F_2		
Fun	PI	2	3	4	COST
F_{12}	(0,2)	$\otimes$			3
F_{12}	(0,4)			$\otimes$	3
F_2	(1,3)		$\otimes$		3

We have 3 essential PIs and the final covering is:

$$F_1=(0,2)+(0,4)$$

$$F_2=(0,2)+(0,4)+(1,3)$$

None of the PIs in the covering is redundant.

We usually remove redundancies from the final covering although they do not cost more according to the definition of our cost function.

IV. Implementation

A. Computer-aided design

This chapter demonstrates the effectiveness of the MOMIN algorithm by testing run forty-three examples and comparing their results.

As mentioned early in chapter III, two approaches: (A) Minimal solution, and (B) Subminimal solution were presented in this thesis, but only the first one was programmed for computer-aided design. The current implementation is on the CDC CYBER model 73. To save memory, several data items were often stored in a single word.

The program structure is shown in Fig. 4.1. The general flow of the program is shown in Fig. 4.2. Supporting subroutines are described below:

1) Subroutine DATA reads a multiple-output switching function in decimal form, converts TFs and XFs into octal numbers 01 and 11, respectively, and stores them in array SUBFN.

2) Subfunction SUBFUN derives all subfunctions. The ID codes are generated by subroutine IOCODE and stored in a single word. Example: A subfunction F_{123} has an ID code 111 in binary form. It is stored in the first column of array SUBFUN as $\begin{array}{lccc} 59 & 2 & 1 & 0 \\ \hline \multicolumn{1}{|c|}{\qquad\qquad 0 \qquad\qquad} & \multicolumn{1}{|c|}{1} & \multicolumn{1}{|c|}{1} & \multicolumn{1}{|c|}{1} \\ \hline \end{array}$, where 0,1,2,

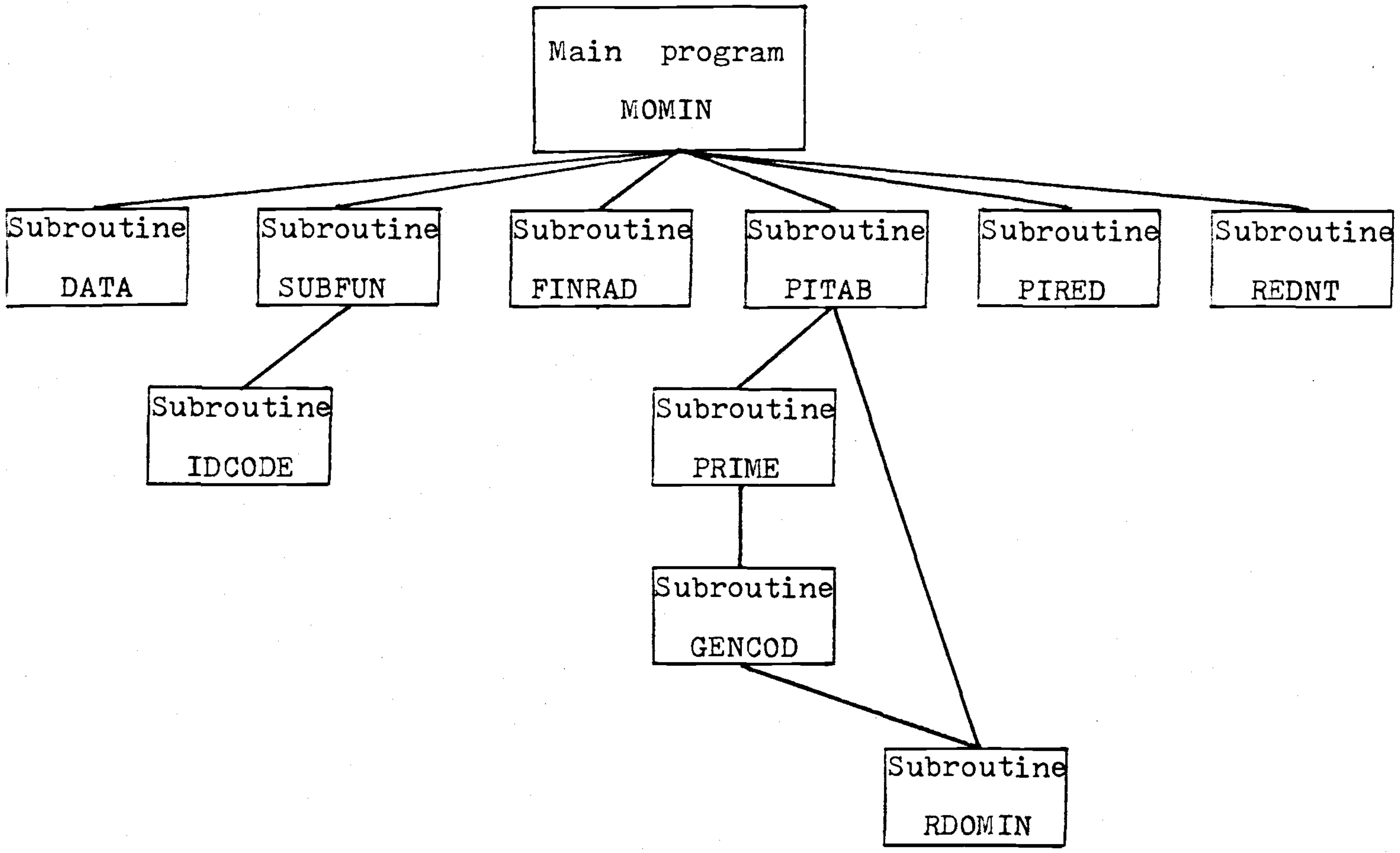


Figure 4.1 Program Structure

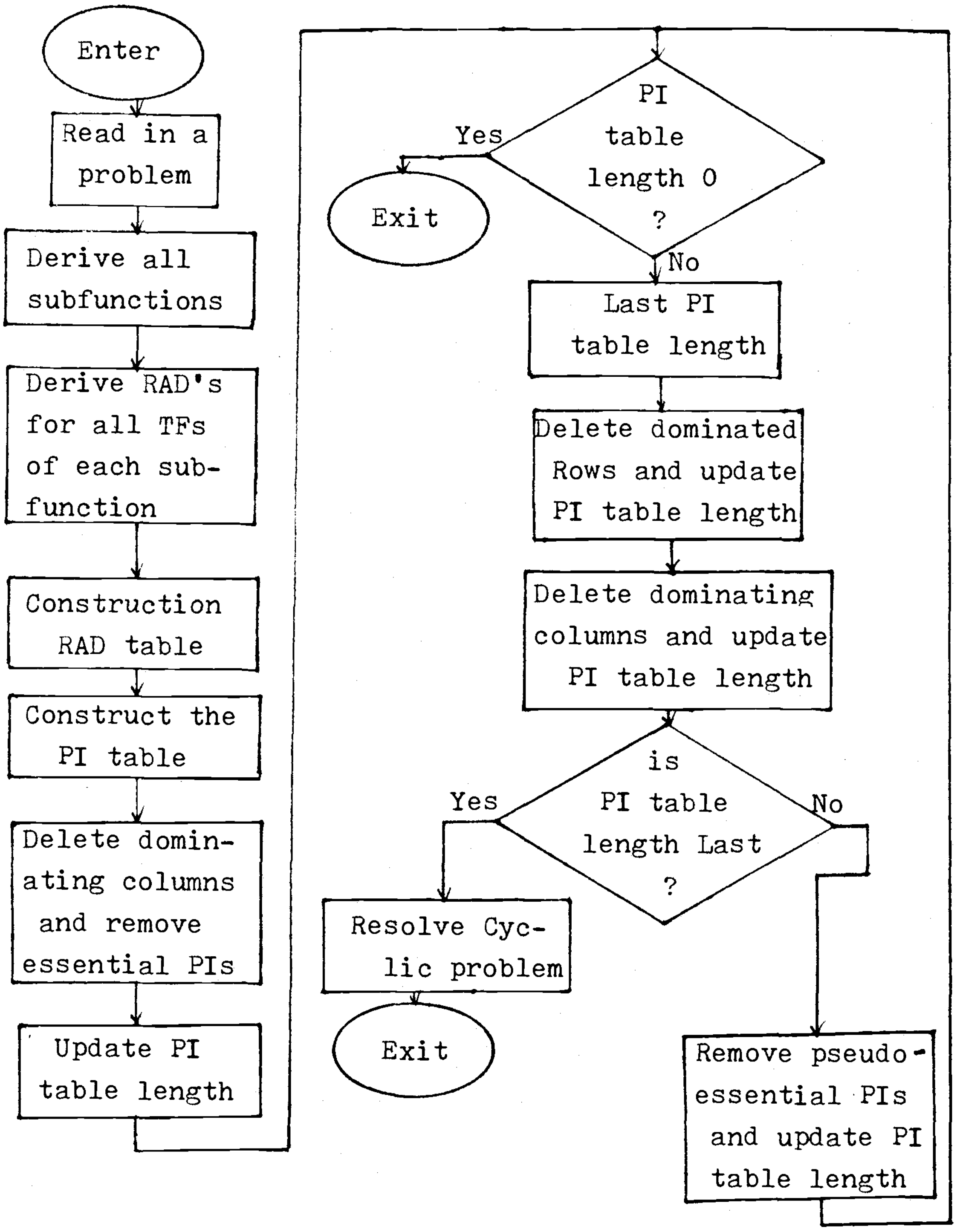
...59 indicate bit numbers. Fig. 4.3 depicts how information is stored in array SUBFN. The subfunction ID code ($F_1 \dashrightarrow 1$; $F_2 \dashrightarrow 2$; $F_3 \dashrightarrow 4$; $F_{123} \dashrightarrow 7$;... ...) is stored in first column, subscripts of minterm m_i is stored in the first row; "00" indicates FF, "01" indicates TF and "11" indicates XF. For example, $F_2 = \sum(1,2,6,7)+d(0,5)$ "01" is entered in columns 1, 2, 6 and 7; "11" is entered in columns 0 and 5.

3) Subroutine FINRAD derives RAD's for each TF. This is done by exclusive-or each TF with adjacency-directions to see if the result is a TF or XF. Since RAD's are of the form 2^r($r=0,1,...n-1$; n is the number of variables), they are stored by their exponents. A "1" is stored in bit number 1 if 2^1 is a RAD. Example: If a TF has 3 RAD's, say 1,2 and 8(or in exponential form 2^0, 2^1 and 2^3), they will be stored in a single word as

59	3	2	1	0
0	1	0	1	1

. If a TF has no RAD, a -2 is stored instead to avoid misinterpreting stored information(any number that can achieve this requirement may be used).

4) Subroutine PITAB construct a complete PI table by searching down the RAD tree. Each time a new PI is derived, subroutine RDOMIN is referenced to delete any dominated row(PI).



Enter
Read in a problem
Derive all subfunctions
Derive RAD's for all TFs of each sub-function
Construction RAD table
Construct the PI table
Delete dominating columns and remove essential PIs
Update PI table length
PI table length 0 ?
Yes
Exit
No
Last PI table length
Delete dominated Rows and update PI table length
Delete dominating columns and update PI table length
is PI table length Last ?
Yes
No
Resolve Cyclic problem
Exit
Remove pseudo-essential PIs and update PI table length

Figure 4.2 General flow of algorithm

TF Fun	0	1	2	3	4	5	6	7
F_1	00	01	11	11	01	01	00	00
F_2	11	01	01	00	00	11	01	01
F_3	01	00	01	01	00	00	00	11
F_{12}	00	01	01	00	00	01	00	00
F_{13}	00	00	01	01	00	00	00	00
F_{23}	01	00	01	00	00	00	00	01
F_{123}	00	00	01	00	00	00	00	00

(a)

	0	1	2	3	4	5	6	7
1	00	01	11	11	01	01	00	00
2	11	01	01	00	00	11	01	01
4	01	00	01	01	00	00	00	11
3	00	01	01	00	00	01	00	00
5	00	00	01	01	00	00	00	00
6	01	00	01	00	00	00	00	01
7	00	00	01	00	00	00	00	00

(b)

Figure 4.3 Typical array SUBFN for a 3-output 3-variable function (a) Written form (b) Computer storage form

5) Subroutine PRIME searches for a new PI each time it is referenced by subroutine PITAB.
6) Subroutine GENCOD generates an RAD tree sequence. If a TF has RAD's 1 and 4, then a 14, 1 and 4 sequence is generated to order the PI search in this order.
7) Subroutine RDOMIN performs row dominance checking and deletes these dominated rows.
8) Subroutine PIRED first removes essential PI's from the PI table, then reduces the PI table by deleting dominated rows and dominating columns, as well as selecting pseudo-essential PI's.
9) Subroutine REDNT removes redundancy from each output function by chedking if TFs in a PI are all covered by other PI's.

B. Results and comparison

All examples presented in this section were used to test various aspects of the multiple-output minimization algorithm.

A typical design example would be: A Sine generator(2) which accepts input angle(X) in digital form and generates approximate results Sin(X) in digital form.

Let us first express an angle as $X=A_1X_1+A_2X_2+...+A_nX_n$, where $A_1=2^{-1}$, $A_2=2^{-2}$,...$A_n=2^{-n}$, X_i is 0 or 1, thus X will be in the range $0 \leq X < 1$. Next express, the result $\mathrm{Sin}(X\pi)=Z=B_1Z_1 +B_2Z_2+...+B_mZ_m$, where $B_1=2^{-1}$, $B_2=2^{-2}$,..$B_m=2^{-m}$, Z_i is 0 or 1, thus Z will be in the range $0 \leq Z < 1$. Assume that m=n=4(note that with this assumption we can have only a limited application for its poor accuracy) Fig. 4.4(b) shows such an approximation. The truth table is shown in Fig. 4.4(a), we now have a multiple-output combinational circuit design problem. The switching function of this problem is translated directly from the table.

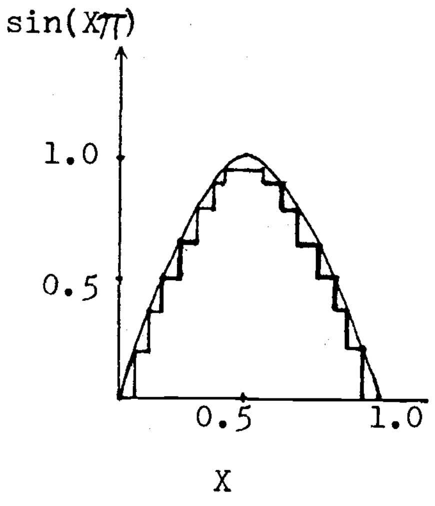
$$Z_1(X_1,X_2,X_3,X_4)=\sum(3,4,5,6,7,10,11,12,13,14,15)$$
$$Z_2(X_1,X_2,X_3,X_4)=\sum(2,5,6,7,10,11,12,13,16)$$
$$Z_3(X_1,X_2,X_3,X_4)=\sum(1,2,4,6,7,10,11,12,14,16,17)$$
$$Z_4(X_1,X_2,X_3,X_4)=\sum(1,4,5,7,10,11,13,14,17)$$

If we minimize each function individually, we can have the following solutions(Refer to examples No, 1 to No. 4 in appendix B for computer-aided design output).

$$Z_1=(4,5,6,7)+(3,7)+(4,5,14,15)+(10,11,12,13)$$
$$Z_2=(2,6,12,16)+(5,7)+(10,11,12,13)$$
$$Z_3=(1,11)+(6,7,16,17)+(2,6,12,16)+(10,12,14,16)$$
$$+(4,6,14,16)$$
$$Z_4=(1,5)+(4,14)+(10,11)+(11,13)+(7,17)$$

Note that Z_4 is a cyclic problem. Two minimal solutions are shown in Fig. 4.5. If we make no attempt to share PI's, the

X_1	X_2	X_3	X_4	Z_1	Z_2	Z_3	Z_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	1	1
1	1	0	1	1	0	0	0
1	1	1	0	0	1	1	0
1	1	1	1	0	0	1	1

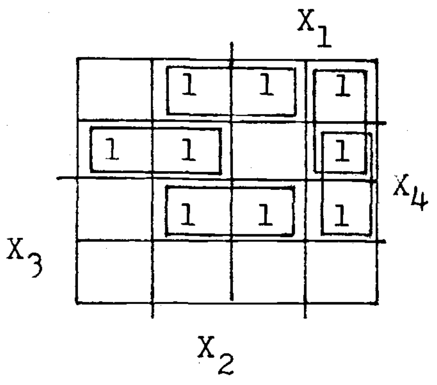


(b)

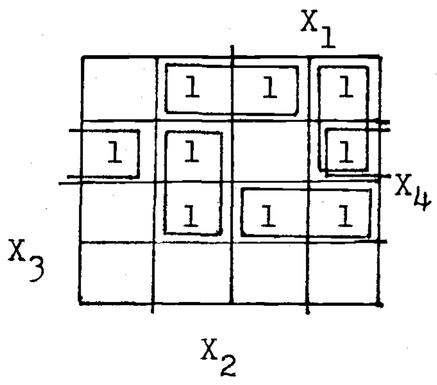
(a)

Figure 4.4 Sine generator (a) Truth table

(b) Approximated Sine function



(a)



(b)

Figure 4.5 Two possible same cost solutions for Z_4

(a) $Z_4=(1,5)+(4,14)+(10,11)+(11,13)+(7,17)$

(b) $Z_4=(1,11)+(5,7)+(4,14)+(10,11)+(13,17)$

total cost is 46. But if the multiple-output minimization algorithm is used as shown in appendix B example No. 5, some PI's are shared by more than 1 output function, and the total cost becomes 32, which is 14 lower. This is the major advantage of multiple-output minimization. The minimal solution of this example is somewhat different from that of Dietmeyer(2), but the degree of minimality is the same.

Solution from Dietmeyer is:

$Z_1=(\underline{3,13})+(10,11,12,13)+(\underline{10,11,14,15})+(4,5,6,7)$

$Z_2=(5,7)+(2,6,12,16)+(10,11,12,13)$

$Z_3=(4,14)+(1,11)+(\underline{10,11})+(2,6,12,16)+(6,7,16,17)$

$Z_4=(4,14)+(5,7)+(\underline{13,17})+(1,11)+(\underline{10,11})$

Solution from MOMIN is:

$Z_1=(\underline{3,7})+(10,11,12,13)+(\underline{4,5,14,15})+(4,5,6,7)$

$Z_2=(5,7)+(2,6,12,16)+(10,11,12,13)$

$Z_3=(4,14)+(1,11)+(\underline{10,14})+(2,6,12,16)+(6,7,16,17)$

$Z_4=(4,14)+(5,7)+(\underline{11,17})+(1,11)+(\underline{10,14})$

Note that: (i) PIs (3,7) and (4,5,14,15) are selected in Z_1 instead of PIs (3,13) and (10,11,14,15), respectively.

(ii) PI (10,14) is selected in Z_3 instead of PI (10,11).

(iii) PIs (11,17) and (10,14) are selected in Z_4 instead of PIs (13,17) and (10,11), respectively.

Although different cost function are used, we have the same degree of minimality.

Example No. 6 to No. 12 in Appendix B correspond to example 1, 2, 5, 7, 8, 9 and 10 in McKinney's dissertation (7). It is seen that all seven examples have the same minimal cost results(even though there are some different essential/pseudo-essential PIs, they have the same cost, e.g. The PI (8,10,24,26) in example No. 6 instead of the PI (10,11,26,27) is selected. The comparison is shown in Table 4.1.

Two sets of single-output functions were used to reveal the advantage of multiple-output minimization.

(1) Example No. 13 to 18 in Appendix B are six 4-variable single-output functions, they are minimized individually as shown. They were combined to form ten m-output 4-variable functions(where m is 2, 3, 4, 5 or 6) as shown in Table 4.2 column four. It is seen that the execution time increases as the number of TFs/XFs increases. Examples No. 19 and 20 are the combinations of examples No. 13 and No. 14 and No. 15 and No. 16, respectively. Because there are no shared PIs, the multiple-output minimization result costs are the same as the single-output minimization result costs. Example No. 21 is the combination of examples No. 14 and 18. The PI (3,7) instead of the PI (3,7,11,15) is selected to share with F_1. This reduces the total cost by two. Example No. 25 is the combination of examples No. 15, 16, 17 and 18. The PI (3,7) instead of (6,7) is selected in F_2 since it can be shared by F_1. F_3 has

the same result as example No. 17; although in F_4 the PI (6, 7,14,15) has a lower cost than PI (6,14), (6,14) is selected since it is shared by F_2 and F_4. The PI (3,7,11,15) is shared by F_3 and F_4. The total cost is reduced by 3+2+2=7. Other examples show same kind of cost reductions. The P.C. ratio (product cost ratio; multiple-output minimization cost/single - output minimization costs) shown in Table 4.2 indicates that the network cost decreases(compared with summation of each single-output cost) as the number of outputs increases.

(2) Examples No. 29 to 34 are six 5-variable single-output functions,minimized individually. The same kind of comparisons are made as shown in Table 4.3. Note that the execution time is related to (i) the number of outputs, (ii) the number of variables and (iii) the number of TFs/XFs. The MOMIN algorithm has a fairly slow increasing execution time with the above factors. Example No. 43 , the combination of examples No. 29 to 34, is a 6-output, 5-variable function. It is a cyclic problem. In general it is more time consuming to solve cyclic problems, because a cyclic chain(all remaining PIs in the PI table) has to be resolved. In order to find a minimal solution for a cyclic problem, most combinations of remaining PIs must be examined. This has been done by many previous investigators and thus it is not repeated in this thesis.

Table 4.1 MOMIN and DSA performance comparison for single-output minimization

Example	No. of variables	No. of TF	No. of XF	Execution time(sec.) DSA	Execution time(sec.) MOMIN	Product cost DSA	Product cost MOMIN
6	4	2	11	0.43	0.275	3	3
7	4	9	0	0.58	0.406	12	12
8	4	9	0	0.59	0.362	15	15
9	5	4	12	0.74	0.310	13	13
10	5	10	7	0.99	0.485	21	21
11	5	15	0	0.88	0.638	13	13
12	6	16	9	5.03	0.759	47	47

Note: Execution time of DSA includes both POS and SOP solutions, while MOMIN finds only SOP solution

Different computer systems were used in DSA and MOMIN

Table 4.2 Performance of multiple-output minimization

Example	No. of outputs	No. of TF	No. of XF	Combined s-o examples	Execution time(sec)	Product cost m-o	Product cost s-o's	P.C. ratio m/s
19	2	8	6	13,14	0.323	15	15	1
20	2	8	4	15,16	0.299	19	19	1
21	2	9	5	14,18	0.350	11	14	0.785
22	3	12	9	13,14,15	0.503	18	24	0.750
23	3	12	5	16,17,18	0.545	14	21	0.666
24	4	16	10	13,14,15,16	0.745	25	34	0.735
25	4	12	6	15,16,17,18	0.788	22	30	0.733
26	5	21	11	14,15,16,17,18	1.410	28	38	0.736
27	5	20	12	13,14,15,16,17	1.183	28	39	0.710
28	6	24	14	13,14,15,16,17,18	2.051	31	45	0.689

Notes: Variable number n=4 P.C. means product cost

m-o means multiple-output, s-o means single-output

Table 4.3 Performance of multiple-output minimization

Example	No. of outputs	No. of TF	No. of XF	Combined s-o examples	Execution time(sec)	Product cost m-o	Product cost s-o's	P.C. ratio
35	2	34	3	29,30	1.939	38	38	1
36	2	16	5	31,32	0.705	38	38	1
37	2	13	5	31,33	0.456	32	32	1
38	3	41	3	29,30,31	2.597	52	60	0.866
39	3	27	18	32,33,34	2.240	27	30	0.900
40	4	34	18	31,32,33,34	3.030	47	52	0.903
41	4	43	21	29,32,33,34	4.269	43	51	0.843
42	5	50	21	29,31,32,33,34	5.816	57	73	0.780
43	6	68	21	29,30,31,32,33,34	Cyclic problem			

Note: Variable number n=5

V. Summary and Conclusion

McKinney's new technique for minimizing single-output switching functions emphasizes that those PIs that must be covered in the final solution are generated first. This avoids the possibility of creating non-essential PIs except in cyclic problems. It differs from the conventional approach in that not all PIs are generated. However, since in the minimization of multiple-output switching functions multiple-output prime implicants must be generated and tested to see if sharing is possible, this new technique becomes less applicable(since McKinney's method is not directed toward the derivation of all PIs, thus the derivation of all MOPIs becomes difficult).

From the computer-aided design outputs we see that multiple-output minimization is less useful in comparison with individual minimization if the number of outputs is small. But if the number of outputs increases, the advantage of multiple-output minimization becomes evident as discussed in chapter II.

The MOMIN algorithm presents the applicability of RAD's and RAD trees in the minimization of multiple-output functions. Although it is basically a conventional approach, i.e. derive all PIs and find a cover for the function, the algorithm shows a different way in generating PIs as well as the advantage of multiple-output minimization.

A faster and more efficient algorithm for the minimization of multiple-output circuits is the current trend of study and is also the goal of this thesis. It has been found that McKinney's directed search algorithm DSA is difficult to apply directly to multiple-output circuits if an absolute minimal solution is required(since MOPIs must be considered). But if a subminimal solution is desired, the ideas of DSA algorithm(not finding all PIs) may be applied.

Although the hardware costs of a multiple-output circuit may be significantly reduced by finding a minimal solution, yet the hardware costs saved may be less than the design costs saved by finding an easily found sub-minimal solution. A subminimal solution is sometimes more economical. In designing logic circuits, these two factors are to be compromised. It is why the two versions of the minimization algorithm are presented in this thesis.

BIBLIOGRAPHY

1. Bartee T. C., "Computer Design of Multiple-output Logical Networks, " IRE Transactions on Electronic Computers, vol. EC-10, pp. 21-30, March 1961.

2. Dietmeyer D. L., Logic Design of Digital Systems, Boston, Mass.: Allyn and Bacon, 1971.

3. Douglas L., Computer-Aided Design of Digital Systems, New York: Crane Russak, 1977.

4. Hong S. J., Cain R. G. and Ostapko D. L., "Mini: A Heuristic Approach for Logic Minimization," IBM Technical Journal, pp. 443-458, September 1974.

5. Kohavi Z., Switching and Finite Automata Theory, New York: McGraw-Hill, 1970.

6. Lee S. C., Digital circuits and Logical Design, Englewood, N. J.: Prentice-Hall, 1976.

7. McKinney M. H., "A Directed Search Algorithm for the Canonical Minimization of Switching Function," Ph. D. dissertation, Texas A&M Univ., College Station, TX. 1974.

8. Nagle H. T. Jr., Carroll B. D. and Irwin J. D., An Introduction to Computer Logic, Englewood Cliffs, N. J.: Prentice-Hall, 1975, Chapter four.

9. Rhyne V. T., Noe P. S., McKinney M. H. and Pooch U. W., "A New Technique for the Fast Minimization of Switching Functions," IEEE Transactions on Electronic Computers, vol. C-26, pp. 757-764, August 1977.

10. Roth J. P., "Algebraic topological methods for the synthesis of switching systems," Transaction of the American Mathematical Society, vol. 88, pp. 301-326, July 1958.

11. Su Y. H. and Dietmeyer D. L., "Computer Reduction of Two-level, Multiple-Output Switching Circuits," IEEE Transactions on Electronic Computers, vol. C-18, pp. 58-63, January 1969.

12. Urbano R. H. and Mueller R. K., "A topological method for the determination of the minimal forms of a Boolean function," IRE Transactions on Computers vol. ED-5, pp. 126-132, September 1956.