[illegible]

ABSTRACT

FAULT ANALYSIS OF COMBINATIONAL LOGIC NETWORKS

By

Lung-Hsiung Chang

The algebraic model called the Complete Gate Equivalent Model (CGEM) which describes completely and precisely the physical and logical structures of a multiple output network is proposed for the study of faulty combinational logic networks.

In this thesis, the power of a CGEM is demonstrated, with its various forms, through its application in the generation of fault functions and complete test sets for a network with stuck-at type faults. An upper bound on the number of possible fault functions, hence the number of possible fault classes, is established. Then, a straightforward procedure to find, without enumeration, all possible fault functions and the associated fault sets is presented.

A complete test set can be found by covering the direct differences which are generated by enumerating all possible faults of the network. It is shown given a fault function and a complete test set of a logic network, one

8/2/32

can either find the associated fault set or decide that the network cannot have degenerated into the given fault function. Methods of computing the fault functions for several kinds of bridging faults also are demonstrated.

Four algorithms for generating near minimal complete test sets are presented. The first two are for multiple fault and single fault detection. The other two are the direct generalizations of the first two to fault location. A method for searching for a minimal complete detection test set for an irredundant network is also suggested. In general, one can find a smaller complete detection test set than that obtained by Poage's method. The generalizations of the method to cover the general networks parallel the development of the algorithms. Furthermore, the existence of undetectable faults in a redundant network impose no restrictions on the test generation since we can modify the CGEM according to the fault before it is used in the algorithms.

When detailed information about the logic network is not required, the Complete Gate Equivalent Model can be modified to describe LSI combinational networks in a manageable manner. Its use in generating tests for a sequential logic network has also been pointed out.

FAULT ANALYSIS OF COMBINATIONAL LOGIC NETWORKS

By

Lung-Hsiung Chang

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1974

ACKNOWLEDGMENTS

I am grateful to Dr. Carl V. Page, the chairman of my guidance committee, for his guidance and encouragement during the course of this thesis. My thanks also go to Dr. Richard J. Reid, Dr. Julian Kateley, Jr., Dr. Edward A. Nordhaus and to Dr. Berhard L. Weinberg for serving on my guidance committee and for reviewing this work.

Finally, I am deeply indebted to my wife Hsiu-Lan for her patience, understanding and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

In this chapter we present some background materials required for a study of faulty combinational logic networks. The basic assumption is that the combinational logic networks are comprised of single-output elementary gates. Further assumptions will be presented as the development of this thesis requires.

Most studies of faulty combinational logic networks have concentrated on the network level rather than on the system level and have been concerned only with the effects of faults on the logical behaviors of the networks. This thesis will follow in this tradition unless otherwise indicated.

The importance of the study of faulty combinational logic networks has been pointed out by several authors, e.g., Carter [6] and Kamal [22] and is evidenced by numerous related articles appearing in such publications as the IEEE Transactions on Computers. Furthermore, there have been three International Symposia on Fault-Tolerant

Computing in the past three years and two textbooks devoted exclusively to fault detection and diagnosis of digital systems.

The original contributions and organization of the thesis are discussed at the end of this chapter.

1.2 <u>COMBINATIONAL LOGIC NETWORKS</u>

A Boolean function is <u>elementary in argument X</u> if the function can be expressed in the form $X^* \cdot f_1$ or $X^* + F_2$, where $X^* = X$ or $\overline{X}$ and f_1 and f_2 are any Boolean functions independent of X. A function is <u>elementary</u> if it is elementary in all of its input arguments. An <u>elementary gate</u> is a gate whose function is elementary. The commonly used gates such as AND, OR, NOT, NAND and NOR are all elementary gates while Exclusive-OR is not. This thesis will deal with combinational logic networks which are realized by elementary gates. Figure 1.1 shows the schematic notion that will be used to represent these gates.

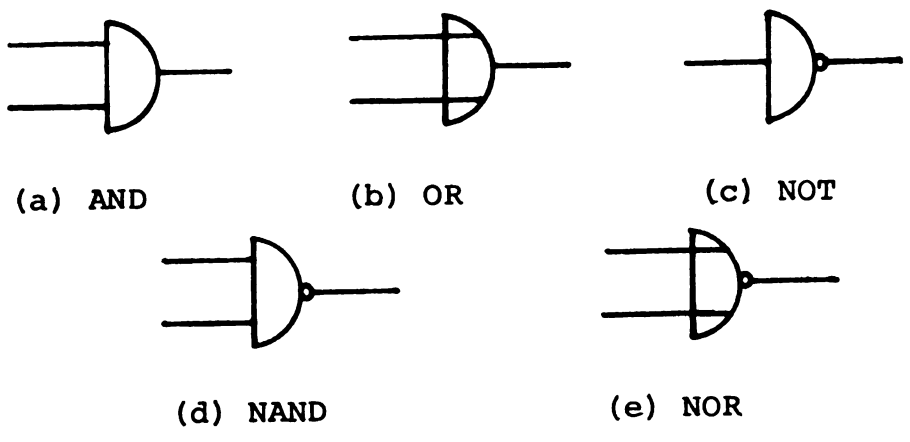
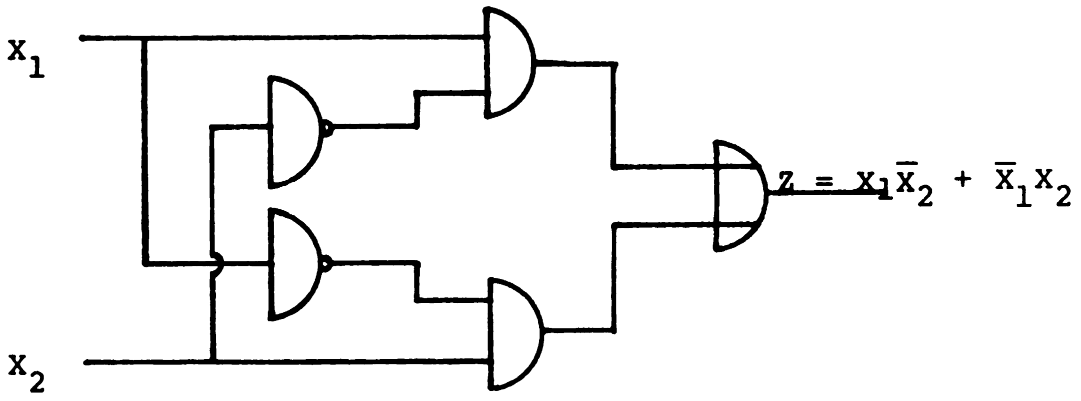


Figure 1.1. The Elementary Gates

Combinational logic networks are used to realize combinational functions and are characterized by the absence of feedback. A combinational function may be represented by (1) a Boolean expression of input literals, (2) a truth table which specifies the value of the function for each combination of input values or (3) a Karnaugh map. Figure 1.2 (a) is a combinational logic network for the Exclusive-OR function. Two different representations of the function are shown on Figure 1.2 (b) and (c).



(a) A Combinational Network for the Exclusive-OR Function.

X_1	X_2	Z
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth Table.

$$\begin{array}{cc|c|c|} & & \multicolumn{2}{c}{X_1} \\ & & 0 & 1 \\ \hline X_2 & 0 & 0 & 1 \\ \hline & 1 & 1 & 0 \\ \hline \end{array}$$

(c) Karnaugh Map.

Figure 1.2. A Network and the Representations for the Exclusive-OR Function.

1.3 LOGICAL FAULTS

The faults that we shall be concerned with are logical faults. A logical fault is a fault which produces some changes in the logical behavior of the network. Hereafter the term fault will mean logical fault unless otherwise indicated.

The class of faults to be considered in the study of faulty logic networks depends on the type of networks. However, most of the faults in contemporary networks can be represented by an input or output of some basic elements being stuck at one or stuck at zero. That is, an input or output of some basic elements may assume a fixed logical value, independent of the inputs supplied to the logic network. All but one section of this thesis deals with stuck-at type faults.

While stuck-at type faults deal with individual leads of the network, bridging faults are concerned with the faults due to the connection of two or more leads of the network. We shall investigate bridging faults exclusively in Section 3.4.

A fault may be permanent or intermittent. A permanent fault is a fault which will not occur, disappear or change its nature during testing and its effect will exist throughout the period of interest. An intermittent fault may show its effect at one time and not at other times. We shall deal only with permanent faults.

1.4 <u>FAULT DIAGNOSIS</u>

A combinational logic network is said to contain a fault if upon the application of an input vector T_i, the output vector of the network is different from that of fault-free network. We call such an input vector a <u>test</u>. A test T_i is said to <u>detect</u> fault F_j if the output vectors for the fault-free network and the same network containing F_j are different when T_i is applied.

A set of tests which detects all detectable single faults of the network is called a <u>complete single fault detection test set</u>. The <u>single fault assumption</u> implies the network cannot have more than one fault at any time. This assumption is justifiable only if testing is frequent enough so that the probability of the occurrence of more than one fault during the interval between test experiments is negligibly small. It is clear that the single fault assumption may not be valid for initial check-out of the networks. Even so, it does not mean the test generations under single fault assumption has to be simpler than that under the <u>multiple fault assumption</u>, as we shall demonstrate in Chapter 4.

Two faults are <u>distinguishable</u> if there is a test that generates different output vectors for two instances of the same network containing the different faults. A set of tests which distinguishes every pair of distinguishable faults of the network is called a <u>complete location test set</u>.

In general, a test can detect several faults and a fault can be detected by several tests. It takes several tests to locate a fault and it also takes several faults to specify a test.

1.5 CONTRIBUTIONS AND ORGANIZATION OF THE THESIS

An algebraic model called the Complete Gate Equivalent Model (CGEM) which describes completely and precisely the physical and logical structures of a multiple output network is proposed for the study of faulty combinational logic networks.

In this thesis, the power of a CGEM is demonstrated, with its various forms, through its application in the generation of fault functions and complete test sets for a network with stuck-at type faults. An upper bound on the number of possible fault functions, hence the number of possible fault classes, is established. Then, a straightforward procedure to find, without enumeration, all possible fault functions and the associated fault sets is presented.

A complete test set can be found by covering the direct differences which are generated by enumerating all possible faults of the network. It is shown given a fault function and a complete test set of a network one can either find the associated fault set or decide that the network can not have degenerated into the given fault function. Methods of computing the fault functions for several kinds of bridging faults also are demonstrated.

Four algorithms for generating near minimal complete test sets are presented. The first two are for multiple fault and single fault detection. The other two are the direct generalizations of the first two to fault location. A method of searching for a minimal complete detection test set for an irredundant network is also suggested. In general, one can find a smaller complete detection test set than that obtained by Poage's method [33].

In Chapter 2 the Complete Gate Equivalent Model is presented. Chapter 3 deals with fault functions and Chapter 4 describes the algorithms and the method of complete test generations. Chapter 5 concludes the thesis and recommends problems for further research.

CHAPTER 2

A COMPLETE GATE EQUIVALENT MODEL

2.1 INTRODUCTION

The problem of analyzing the relationships between network faults and output logic abberrations has already been studied by numerous models and various techniques. However, a large number of unanswered questions remain in the area of fault detection and location. In order to solve these and other related problems, a logic network model which makes structural and logical interdependencies easier to recognize and express formally is still needed.

The earliest model which embodies information about both structure and logical characteristics of an object network to aid in fault analysis was presented by Poage [33] in 1963. Poage's model uses three valued lead variables to explicitly represent the normal, stuck at one and stuck at zero conditions of each lead. This results in expressions which are notationally complicated and unwieldly even though the model is rigorously presented.

Armstrong's enf (equivalent normal from) [1] offers a much more tractable notation than that of Poage, but it lacks both the completeness and preciseness of the latter. Clegg [11] developed the SPOOF (Structure and Parity-Observing Output Function) model to more completely describe the network structure, by making internal network inversion explicit without providing a mathematically complete and rigorous basis. While Reese and McCluskey [36] utilized the SPOOF model to take advantage of its convenient notation and to provide mathematical rigor, it lacks both structural generality and logical completeness.

The model to be developed is an extension of Reese and McCluskey's model.

2.2 THE COMPLETE GATE EQUIVALENT ALGEBRAIC NETWORK MODEL (CGEM)

A labeling scheme that accurately describes the structure of the network will be used in our model. The CGEMs describe both the outputs and the complement of the outputs of the multiple-output network, whereas the GEM describes only the output of the single-output network. Furthermore, a formal definition of the functional equivalent of CGEM will be added to the model.

The logic networks we are going to study are finite, combinational networks comprised of single-output elementary gates with all their leads properly labeled. The labeling scheme is as follows:

<u>Procedure L (Lead-labeling)</u>

(1) Assign a distinct integer label to each primary input lead and the output lead of each elementary gate.

(2) To each of r fanout branches from a lead labeled p, assign a distinct label pv, where v is a letter in some alphabet other than the integers.

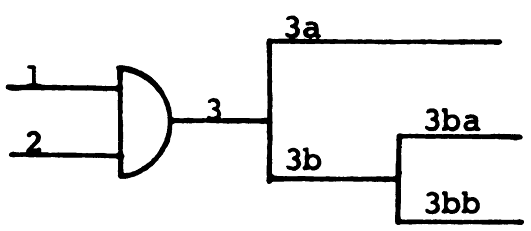
(3) Repeat step (2) until all of the leads are labeled.

Because the networks to be considered are finite, the labeling procedure will stop in a finite number of steps. The reasons for explicitly labeling fanout branches are:

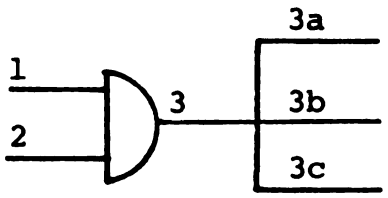
(1) Under the single fault assumption, if a network model that does not faithfully represent the actual wiring topology is used to generate test sets, some real single faults may remain untested. For example, a failure along lead 3b in Figure 2.1 (a) may be modeled as a single stuck-at fault. However, this failure can not be represented by any single fault in the network of Figure 2.1 (b).

(2) Under the multiple fault assumption, fanout branches are checkpoints which play an important role in the generation of complete detection test sets.

(3) Fault analysis for certain kinds of failures, e.g., bridging faults, require the identification of fanout branches.



(a) Labeling that Preserves Physical Wiring Topology.



(b) Labeling that Assumes Simple Fanout.

Figure 2.1. Two Different Labeling Schemes for the Same Portion of a Combinational Logic Network.

Example 2.1

Figure 2.2 is a network after applying Procedure L.

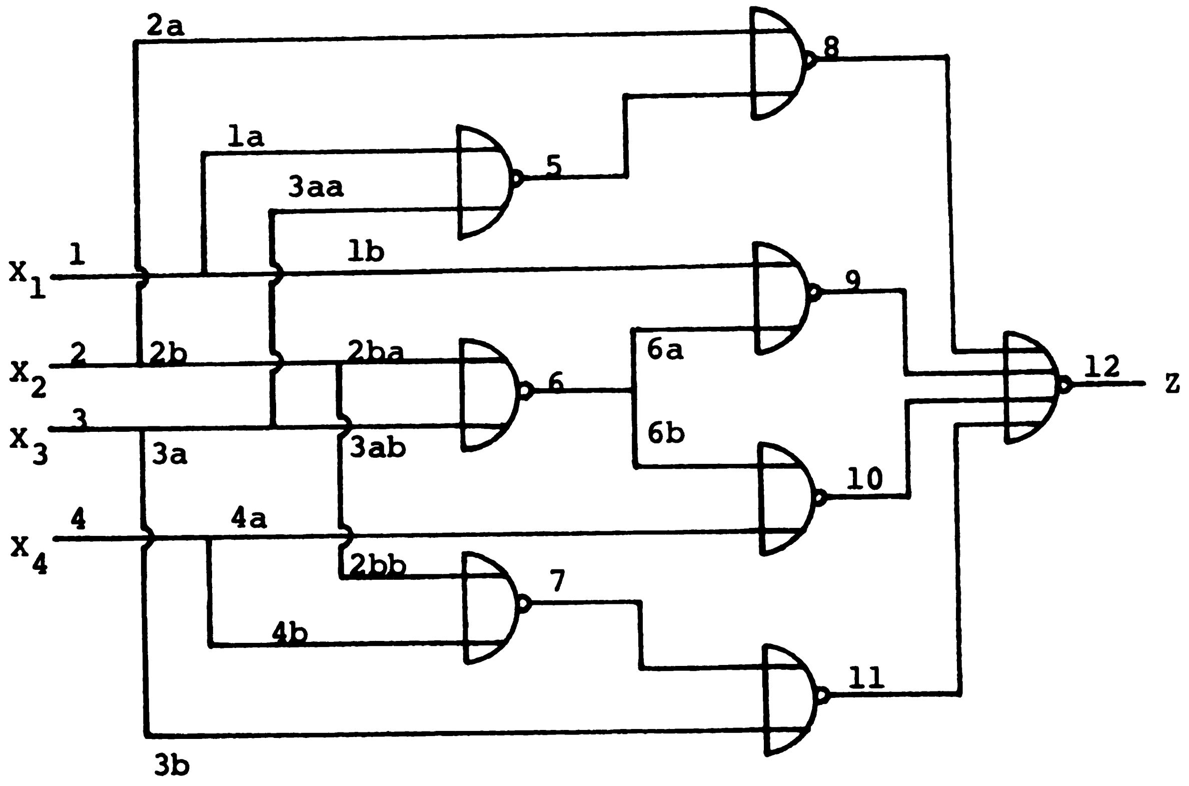


Figure 2.2. A Combinational Logic Network which Realizes $X_1X_2X_3X_4 + \overline{X}_1\overline{X}_2\overline{X}_3\overline{X}_4$.

Definition 2.1

The inversion parity with respect to a path h of lead g is the number (modulo 2) of inverting elementary gates between g and a network output Z_j along the signal path h. The inverting elementary gates are: NOT, NAND and NOR.

Since there is no inverting elementary gate between an input literal and the associated primary input lead, the inversion parity of an input literal will be the same as that of the associated input lead.

Definition 2.2

A path set $L = \langle a_k^*, \ldots , a_n^*, Z_j^* \rangle$ is an ordered set of labels describing some signal path h of a network with a network output literal as its last element. The ordering of the labels corresponds to the order in which a signal traveling along h would encounter them.

$a_r^* = a_r$ if the inversion parity of a_r in the path is even and $Z_j^* = Z_j$ or the inversion parity of a_r in the path is odd and $Z_j^* = \overline{Z}_j$.

$= \overline{a}_r$ otherwise.

Definition 2.3

A g-literal $L = \langle X_i^*, a_1^*, \ldots , a_n^*, Z_j^* \rangle$ is a path set which has an input literal X_i as its first element.

From the above definitions that we know we can divide the g-literals and the path sets of a network into two disjoint subsets, one which possesses overbarred output literals, while the other does not. We shall call the latter "g^1-literals" and "p^1-path sets", the former "g^0-literals" and "p^0-path sets".

Example 2.2

For the single output network of Figure 2.3 we list all p^1-path sets in Table 2.1(a). The p^1-path sets of (a) which are also g^1-literals are listed in Table 2.1(b). All p^0-path sets and all g^0-literals of the network are just their counterparts with all elements complemented. An input literal, an output literal or a lead label is overbarred if and only if it is complemented.

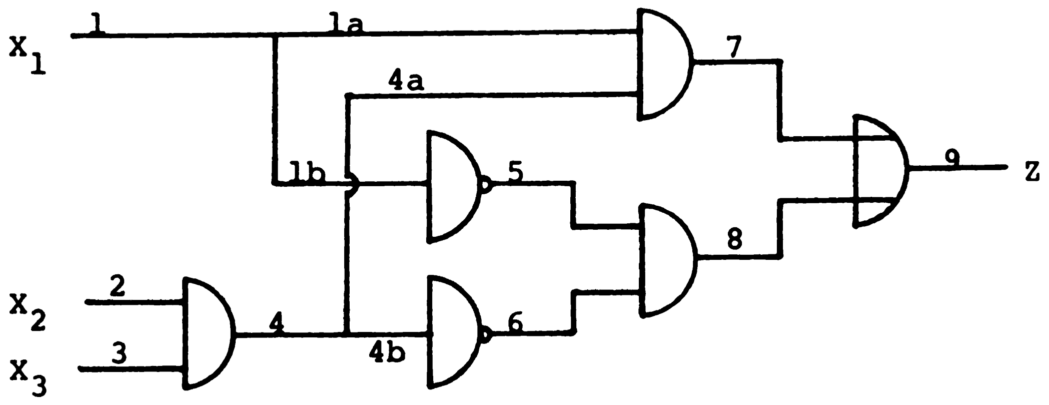


Figure 2.3. A Combinational Logic Network which Realizes $X_1X_2X_3 + \overline{X}_1\overline{X}_2 + \overline{X}_1\overline{X}_3$.

Note that a lead label may appear in several path sets and g-literals. Whether a particular label is complemented or uncomplemented depends on its inversion parity and the output literal of the path set or g-literal.

Table 2.1. All p^1-path Sets and g^1-literals for the Network of Figure 2.3.

(a) All p^1-path Sets for the Network of Figure 2.3.

$L_1 = <9,Z>$

$L_2 = <7,9,Z>$

$L_3 = <8,9,Z>$

$L_4 = <1a,7,9,Z>$

$L_5 = <4a,7,9,Z>$

$L_6 = <5,8,9,Z>$

$L_7 = <6,8,9,Z>$

$L_8 = <1,1a,7,9,Z>$

$L_9 = <4,4a,7,9,Z>$

$L_{10} = <\overline{1b},5,8,9,Z>$

$L_{11} = <\overline{4b},6,8,9,Z>$

$L_{12} = <X_1,1,1a,7,9,Z>$

$L_{13} = <2,4,4a,7,9,Z>$

$L_{14} = <3,4,4a,7,9,Z>$

$L_{15} = <\overline{1},\overline{1b},5,8,9,Z>$

$L_{16} = <\overline{4},\overline{4b},6,8,9,Z>$

$L_{17} = <X_2,2,4,4a,7,9,Z>$

$L_{18} = <X_3,3,4,4a,7,9,Z>$

$L_{19} = <\overline{X_1},\overline{1},\overline{1b},5,8,9,Z>$

$L_{20} = <\overline{2},\overline{4},\overline{4b},6,8,9,Z>$

$L_{21} = <\overline{3},\overline{4},\overline{4b},6,8,9,Z>$

$L_{22} = <\overline{X_2},\overline{2},\overline{4},\overline{4b},6,8,9,Z>$

$L_{23} = <\overline{X_3},\overline{3},\overline{4},\overline{4b},6,8,9,Z>$

(b) P^1-path Sets of (a) which are Also g^1-literals.

$L_{12} = <X_1,1,1a,7,9,Z>$

$L_{17} = <X_2,2,4,4a,7,9,Z>$

$L_{18} = <X_3,3,4,4a,7,9,Z>$

$L_{19} = <\overline{X_1},\overline{1},\overline{1b},5,8,9,Z>$

$L_{22} = <\overline{X_2},\overline{2},\overline{4},\overline{4b},6,8,9,Z>$

$L_{23} = <\overline{X_3},\overline{3},\overline{4},\overline{4b},6,8,9,Z>$

Since the purpose of introducing g-literals is to facilitate the study of faulty logic networks, it will be useful to assign the functional equivalent to a g-literal under the fault conditions of interest.

Definition 2.4

A g-literal $L = \langle X_i^*, a_1^*, \ldots, a_n^*, Z_j^* \rangle$ has a functional equivalent $L(F)$ which depends upon the fault F which is present:

$$\begin{aligned} L(F) = {} & X_i^* \cdot a_1/N \cdot a_2/N \cdot \ldots \cdot a_n/N \\ & + a_1^*/1 \cdot a_2/N \cdot \ldots \cdot a_n/N \\ & + a_1^*/0 \cdot a_2/N \cdot \ldots \cdot a_n/N \\ & + a_2^*/1 \cdot a_3/N \cdot \ldots \cdot a_n/N \\ & + a_2^*/0 \cdot a_3/N \cdot \ldots \cdot a_n/N \\ & + \ldots \\ & + a_n^*/1 \\ & + a_n^*/0 \end{aligned}$$

where $a_k/1$, $a_k/0$ and a_k/N represent lead a_k stuck-at 1, 0 and fault-free respectively. All faults are elements of F.

From the above formal definition, we can make following observations: The functional equivalent $L(F)$ of a g-literal L depends both on the current input vector and on the fault affecting the network and is

(1) The first element of L if L does not contain any lead affected by F.

(2) r if $a_k^* = a_k$ is the last element of L for which a_k/r is an element of F.

(3) $\bar{r}$ if $a_k^* = \bar{a}_k$ is the last element of L for which a_k/r is an element of F.

Table 2.2 illustrates this concept for the g-literals of Table 2.1(b).

Table 2.2. The Functional Equivalents of g^1-literals of Table 2.1(b).

(a) $F_1 = \emptyset$

$L_{12}(F_1) = X_1$	;	$L_{19}(F_1) = \bar{X}_1$
$L_{17}(F_1) = X_2$	;	$L_{22}(F_1) = \bar{X}_2$
$L_{18}(F_1) = X_3$	;	$L_{23}(F_1) = \bar{X}_3$

(b) $F_2 = [4a/0,5/1,7/0]$

$L_{12}(F_2) = 0$	;	$L_{19}(F_2) = 1$
$L_{17}(F_2) = 0$	;	$L_{22}(F_2) = \bar{X}_2$
$L_{18}(F_2) = 0$	;	$L^{23}(F_2) = \bar{X}_3$

<u>Definition 2.5</u>

A set Q of g-literals is a <u>Z_j-1</u> set of one of the given outputs Z_j of network N if:

(1) The output Z_j takes on the value 1 whenever the functional equivalents for every g^1-literal in Q has value 1, and

(2) If any member of Q is removed, then condition (1) no longer holds.

Definition 2.6

A set Q of g-literals is a $\underline{Z_j\text{-0 set}}$ of one of the given outputs Z_j of network N if:

(1) The output Z_j takes on the value 0 whenever the functional equivalents for every g^0-literal in Q has value 1, and

(2) If any member of Q is removed, then condition (1) no longer holds.

Table 2.3 lists the Z_j-1 and Z_j-0 sets for the example of Figure 2.4. The subscript j in $Z_{j,k}$-1 and $Z_{j,k}$-0 is used to indicate the set is one of the Z_j-1 and Z_j-0 sets, respectively, while k distinguishes between the elements in Q. Derivation of these sets will be illustrated in the next section.

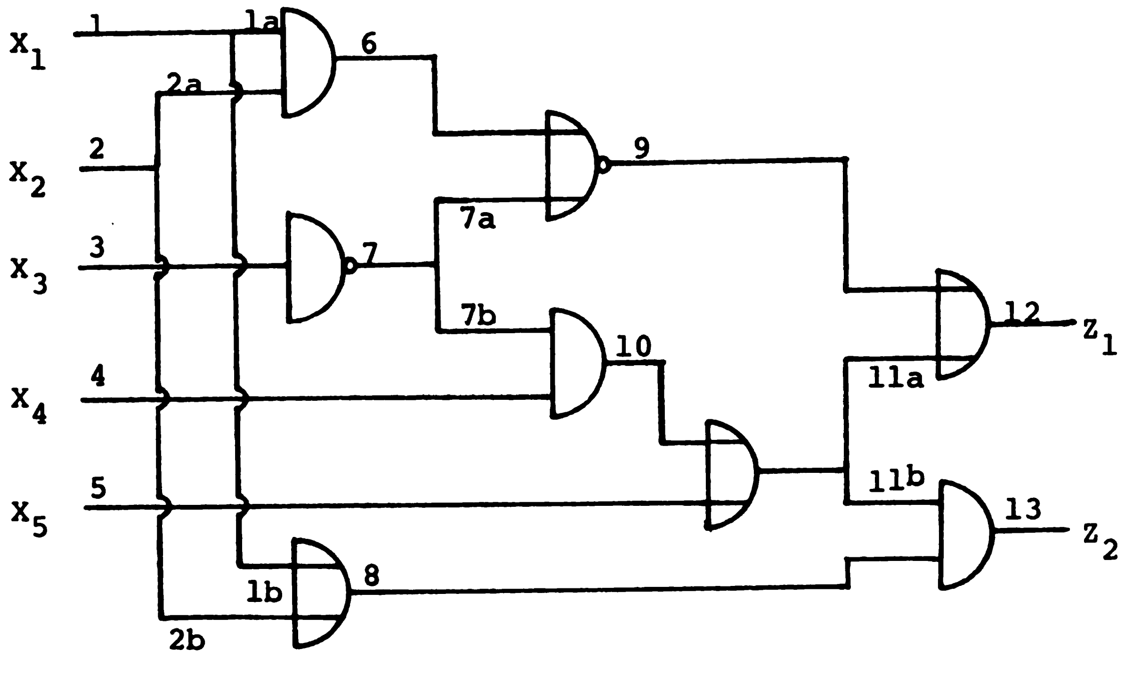


Figure 2.4. The Logic Network for Example 2.3.

Table 2.3. The Z_j-1 and Z_j-0 Sets for the Network of Figure 2.4.

(a) Z_j-1 Sets for the Network of Figure 2.4.

$$Z_{1,1}\text{-}1 = [\langle\overline{X_1},\overline{1},\overline{1a},\overline{6},9,12,Z_1\rangle,\ \langle X_3,3,\overline{7},\overline{7a},9,12,Z_1\rangle]$$

$$Z_{1,2}\text{-}1 = [\langle\overline{X_2},\overline{2},\overline{2a},\overline{6},9,12,Z_1\rangle,\ \langle X_3,3,\overline{7},\overline{7a},9,12,Z_1\rangle]$$

$$\begin{aligned}Z_{1,3}\text{-}1 = [&\langle\overline{X_3},\overline{3},7,7b,10,11,11a,12,Z_1\rangle,\\ &\langle X_4,4,10,11,11a,12,Z_1\rangle]\end{aligned}$$

$$Z_{1,4}\text{-}1 = [\langle X_5,5,11,11a,12,Z_1\rangle]$$

$$\begin{aligned}Z_{2,1}\text{-}1 = [&\langle X_1,1,1b,8,13,Z_2\rangle,\ \langle\overline{X_3},\overline{3},7,7b,11,11b,13,Z_2\rangle,\\ &\langle X_4,4,10,11,11b,13,Z_2\rangle]\end{aligned}$$

$$Z_{2,2}\text{-}1 = [\langle X_1,1,1b,8,13,Z_2\rangle,\ \langle X_5,5,11,11b,13,Z_2\rangle]$$

$$\begin{aligned}Z_{2,3}\text{-}1 = [&\langle X_2,2,2b,8,13,Z_2\rangle,\ \langle\overline{X_3},\overline{3},7,7b,11,11b,13,Z_2\rangle,\\ &\langle X_4,4,10,11,11b,13,Z_2\rangle]\end{aligned}$$

$$Z_{2,4}\text{-}1 = [\langle X_2,2,2b,8,13,Z_2\rangle,\ \langle X_5,5,11,11b,13,Z_2\rangle]$$

(b) Z_j-0 Sets for the Network of Figure 2.4.

$$\begin{aligned}Z_{1,1}\text{-}0 = [&\langle X_1,1,1a,6,\overline{9},\overline{12},\overline{Z_1}\rangle,\ \langle X_2,2,2a,6,\overline{9},\overline{12},\overline{Z_1}\rangle,\\ &\langle X_3,3,\overline{7},\overline{7b},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle,\\ &\langle\overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle]\end{aligned}$$

$$\begin{aligned}Z_{1,2}\text{-}0 = [&\langle X_1,1,1a,6,\overline{9},\overline{12},\overline{Z_1}\rangle,\ \langle\overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle,\\ &\langle X_2,2,2a,6,\overline{9},\overline{12},\overline{Z_1}\rangle,\ \langle\overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle]\end{aligned}$$

$$\begin{aligned}Z_{1,3}\text{-}0 = [&\langle\overline{X_3},\overline{3},7,7a,\overline{9},\overline{12},\overline{Z_1}\rangle,\ \langle\overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle,\\ &\langle X_3,3,\overline{7},\overline{7b},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle]\end{aligned}$$

$$\begin{aligned}Z_{1,4}\text{-}0 = [&\langle\overline{X_3},\overline{3},7,7a,\overline{9},\overline{12},\overline{Z_1}\rangle,\ \langle\overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle,\\ &\langle\overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle]\end{aligned}$$

$$Z_{2,1}\text{-}0 = [\langle\overline{X_1},\overline{1},\overline{1b},\overline{8},\overline{13},\overline{Z_2}\rangle,\ \langle\overline{X_2},\overline{2},\overline{2b},\overline{8},\overline{13},\overline{Z_2}\rangle]$$

$$Z_{2,2}\text{-}0 = [\langle X_3,3,\overline{7},\overline{7b},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle,\ \langle\overline{X_5},\overline{5},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle]$$

$$Z_{2,3}\text{-}0 = [\langle\overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle,\ \langle\overline{X_5},\overline{5},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle]$$

Now we are ready to define a gate equivalent model of the network (GEM).

<u>Definition 2.7</u>

The <u>GEM_j-1</u> set of an output Z_j of network N is the set G_j, where G_j contains all the Z_j-1 sets of N.

<u>Definition 2.8</u>

The <u>GEM_j-0</u> set of an output Z_j of network N is the set H_j, where H_j contains all the Z_j 0 sets of N.

Table 2.4 lists the GEM_j-1 and GEM_j-0 sets for the example of Table 2.3.

Table 2.4. The GEM_j-1 and GEM_j-0 Sets for the Example of Table 2.3.

(a) The GEM_j-1 Sets for the Example of Table 2.3(a).

GEM_1-1 = [$Z_{1,1}$-1, $Z_{1,2}$-1, $Z_{1,3}$-1, $Z_{1,4}$-1]

GEM_2-1 = [$Z_{2,1}$-1, $Z_{2,2}$-1, $Z_{2.3}$-1, $Z_{2,4}$-1]

(b) The GEM_j-0 Sets for the Example of Table 2.3(b).

GEM_1-0 = [$Z_{1,1}$-0, $Z_{1,2}$-0, $Z_{1,3}$-0, $Z_{1,4}$-0]

GEM_2-0 = [$Z_{2,1}$-0, $Z_{2,2}$-0, $Z_{2,3}$-0]

2.3 <u>CONSTRUCTION OF THE COMPLETE GATE EQUIVALENT MODEL FOR A COMBINATIONAL LOGIC NETWORK</u>

The construction of the Complete Gate Equivalent Model (CGEM) for a combinational logic network is accomplished in two steps. First, all g^1-literals of the network are formed and used to make a Boolean expression

E_j for each of the network outputs Z_j. Second, the E_j expressions are reduced to the desired CGEM forms.

2.3.1 Transformation from the Combinational Logic Network to an Equivalent Boolean Expression

In the following algorithm, the extension of a path set $L = \langle a_k^*, \ldots , a_n^*, Z_j^* \rangle$ to the path set $L' = \langle a_e^*, a_k^*, \ldots , a_n^*, Z_j^* \rangle$ will be denoted as $a_e^* \& L$, i.e., $L' = a_e^* \& L$. We shall treat each fanout branch as a single-input OR gate and each NOT gate as a single-input NOR gate.

Algorithm G (Generation of an Equivalent Boolean Expression)

G1: Given a network N with primary input and network output literals, assign distinct labels to each lead of the network using Procedure L.

G2: Form a p^1-path set E_j for each of the m network outputs Z_j. Each path set contains only two elements, the first element is a network output lead label and the second element is the corresponding output literal.

G3: Let $j = 1$.

G4: Choose all path sets which are not g-literals from the current E_j expression.

(1) If no such path set can be found from E_j for all j, where $j = 1,2, \ldots ,m$ then the collection of all E_j's is the desired network expression E.

(2) Process this selection of path sets one by one: Denote the path set under investigation by $L_r = \langle a^*_k, \ldots, a^*_n, Z^*_j \rangle$. Replace all path sets in current E expression which have the same a^*_k as L_r by a subexpression formed according to the appropriate rule as follows:

If a_k is the output from	and if a_k in L_r appears	
	Non-overbarred	Overbarred
AND gate	R1	R3
OR gate or fanout stem	R2	R4
NAND gate	R3	R1
NOR gate or NOT gate	R4	R2

Rules: In each of the following rules, let s be the number of leads which are the inputs to the gate or the fanout branches. The subexpression is

R1: The conjunct of s new path sets, $L_i = a_i \& L_r$.

R2: The disjunct of s new path sets, $L_i = a_i \& L_r$.

R3: The disjunct of s new path sets, $L_i = \bar{a}_i \& L_r$.

R4: The conjunct of s new path sets, $L_i = \bar{a}_i \& L_r$.

where $i = 1, 2, \ldots, s$.

(3) If $j = m$ go to G5, otherwise increase j by 1 and go to G4.

G5: Repeat G3 until all path sets in each E_j expression are g-literals.

Example 2.3

Let us use Algorithm G to construct an equivalent Boolean expression for the network of Figure 2.4.

Step G1: We label the leads of the network by Procedure L of Section 2.

Step G2: Form $<12,Z_1>$ for E_1 and $<13,Z_2>$ for E_2.

Step G3:

(a) At the end of first iteration, we have

$$E_1 = <9,12,Z_1> + <11a,12,Z_1>$$
$$E_2 = <11b,13,Z_2>\cdot<8,13,Z_2>$$

(b) The result of the second iteration is

$$E_1 = <\overline{6},9,12,Z_1>\cdot<\overline{7a},9,12,Z_1> + <11,11a,12,Z_1>$$
$$E_2 = <11,11b,13,Z_2>\cdot(<1b,8,13,Z_2> + <2b,8,13,Z_2>)$$

(c) Beginning with the third iteration, we can take advantage of the fact that Z_1 and Z_2 share some degree of input circuitry and save some effort in the path set expansion process. At the end of the seventh iteration, we get

$$\begin{aligned} E_1 = {} & (<\overline{X_1},\overline{1},\overline{1a},\overline{6},9,12,Z_1> + <\overline{X_2},\overline{2},\overline{2a},\overline{6},9,12,Z_1>) \\ & \cdot<X_3,3,\overline{7},\overline{7a},9,12,Z_1> \\ & + <\overline{X_3},\overline{3},7,7b,10,11,11a,12,Z_1> \\ & \cdot<X_4,4,10,11,11a,12,Z_1> \\ & + <X_5,5,11,11a,12,Z_1> \end{aligned} \qquad (1)$$

$$E_2 = (<\bar{X}_3,\bar{3},7,7b,10,11,11b,13,Z_2> \cdot <X_4,4,10,11,11b,13,Z_2> + <X_5,5,11,11b,13,Z_2>) \cdot <X_1,1,1b,8,13,Z_2> + <X_2,2,2b,8,13,Z_2>) \qquad (2)$$

Since all path sets in (1) and (2) are g-literals, the required equivalent Boolean expression saved as a vector of two E_j expressions is

$$E = (E_1 \; E_2)^T \qquad (3)$$

where $(V)^T$ denotes transpose of vector V.

Note that lead label 7 appears complemented in the third g-literal of Eq.(1) of Example 2.3, whereas it is not complemented when it appears in the fourth g-literal of the same equation. Physically, lead 7 is a fanout stem whose branches reconverge at the network output Z_1. Since the same lead has no branches which reconverge at the network output Z_2, it appears only once in Eq.(2).

In addition to the fanout stems, other labels also appear in different g-literals. In fact, the closer they are to the network output, the more they appear in the g-literals. The only difference is that they do not possess different inversion parities as some fanout stems do.

A data structure for the machine implementation of Algorithm G will be discussed in the Appendix.

Theorem 2.1

All g^1-literals for the combinational network are generated by Algorithm G. Each appears in the resultant expression E exactly once, and all path sets appearing in the final expression are g^1-literals.

Proof

Suppose there exists a g^1-literal $L_i = <X_i^*, a^*, \ldots, a_{n-1}^*, a_n, Z_j>$ which is not generated by Algorithm G. At Step G2 we form a set of two-element path sets which includes path set $L_s = <a_n, Z_j>$. By Step G4, L_s will be chosen and expanded into three-element path set. Since no input lead has been neglected, one of them must be path set $<a_{n-1}^*, a_n, Z_j>$. If this is a g-literal we are done, otherwise repeating the above argument we will generage g^1-literal L_i which contradicts our assumption. Hence all g^1-literals of the network are generated by Algorithm G.

Since each lead is labeled distinctly by Step G1, and a u-element path set can only be expanded to (u+1)-element path sets in one of the following three cases: (1) Expanded to distinct labels from the same path set for AND, OR, NAND and NOR gates. (2) Expanded to the same label from distinct path sets for fanout. (3) Expanded to a label from a path set for NOT gate. Hence all the path sets generated by Algorithm G are distinct from one another which implies each g^1-literal in the resultant expression E appears exactly once.

The fact that all path sets appearing in the final expression are g^1-literals is guaranteed by Step G4.

Q.E.D.

Theorem 2.2

Algorithm G transforms a given network structure into a Boolean expression which accurately and completely describes the logical structure of the network.

Proof

The path set expansion rules used in Algorithm G are exactly the logical operations of elementary gates. The logical structure of the network is accurately described in each step of the algorithm by applying the proper rule to each elementary gate and fanout point.

The completeness of the final Boolean expression is guaranteed by Theorem 2.1.

Q.E.D.

Theorem 2.3

The Boolean expression E generated by Algorithm G is isomorphic to the physical structure and its corresponding input-output literals of the network to the level of interconnected gates.

Proof

Our labeling procedure completely and distinctly labels the physical structure of the network to the level of fanout points and interconnected elementary gates. The rules used in the generation of E are the logical operations of fanout point and elementary gates. For each g^1-literal $L_r = <X_i^*, a_1^*, \ldots, a_{n-1}^*, a_n, Z_j>$ we can find an input literal X_i together with primary input lead a_1 which is followed by lead a_2. The lead a_2 must be a fanout branch of a_1 or the output lead of an elementary gate for which a_1 is one of the input leads. Trace the elements of L_r one by one, we will finally find that a_n is a network output lead and Z_j is the corresponding output literal. Thus, we have a physical path for each g^1-literal in Expression E.

For each physical path we can also find, using a similar argument as the proof of Theorem 2.1, a distinct g^1-literal which describes uniquely and completely each lead in the path together with the input and output literals associated with the path.

Q.E.D.

2.3.2 Reduction of Boolean Expression E to CGEM Form

The second step of the construction of CGEM is to change the Boolean expression E of the network to a two-level sum of products expression E'.

Example 2.4

Find the sum of products expression E' for the network of Figure 2.4.

Starting with Equation (3) of Example 2.3, only carrying it one step further using distributivity, we have

$$\begin{aligned} E_1' = {} & \langle \overline{X_3},\overline{3},7,7b,10,11,11a,12,Z_1\rangle\cdot\langle X_4,4,10,11,11a,12,Z_2\rangle \\ & + \langle \overline{X_1},\overline{1},\overline{1a},6,9,12,Z_1\rangle\cdot\langle X_3,3,\overline{7},\overline{7a},9,12,Z_1\rangle \\ & + \langle \overline{X_2},\overline{2},\overline{2a},6,9,12,Z_1\rangle\cdot\langle X_3,3,\overline{7},\overline{7a},9,12,Z_1\rangle \\ & + \langle X_5,5,11,11a,12,Z_1\rangle \end{aligned}$$

$$\begin{aligned} E_2' = {} & \langle X_1,1,1b,8,13,Z_2\rangle\cdot\langle \overline{X_3},\overline{3},7,7b,10,11b,13,Z_2\rangle \\ & \cdot\langle X_4,4,10,11,11b,13,Z_2\rangle \\ & + \langle X_1,1,1b,8,13,Z_2\rangle\cdot\langle X_5,5,11,11b,13,Z_2\rangle \\ & + \langle X_2,2,2b,8,13,Z_2\rangle\cdot\langle \overline{X_3},\overline{3},7,7b,10,11,11b,13,Z_2\rangle \\ & \cdot\langle X_4,4,10,11,11b,13,Z_2\rangle \\ & + \langle X_2,2,2b,8,13,Z_2\rangle\cdot\langle X_5,5,11,11b,13,Z_2\rangle \end{aligned}$$

and $E' = (E_1' \; E_2')^T$.

Z_j-1 sets and GEM_j-1 sets of Figure 2.4 are listed in Table 2.3(a) and Table 2.4(a), respectively.

2.4 THE COMPLEMENT OF THE BOOLEAN EXPRESSION E OF A COMBINATIONAL LOGIC NETWORK

From Algorithm G we can construct a Boolean equivalent expression E for a logic network. For the completeness of the CGEM, we would like to know how to obtain $\overline{E}$, the complement of E, and some of its properties.

Definition 2.9

The complement of a g-literal $L = \langle X_i^*, a_1^*, \ldots, a_n^*, Z_j^* \rangle$ is $\overline{L} = \langle \overline{X_i^*}, \overline{a_1^*}, \ldots, \overline{a_n^*}, \overline{Z_j^*} \rangle$ which is also a g-literal.

Note that Definition 2.9 does not conflict with Definition 2.4 of Section 2.2. Since the complement of a g-literal L is a g-literal $\overline{L}$ with all elements of L complemented, the complement of a g^1-literal is a g^0-literal and vice versa.

We have defined the functional equivalent for a g-literal and it is not difficult to see that the functional equivalent of a Boolean expression of g-literals is just the Boolean expression of the functional equivalents of individual g-literals.

Lemma 2.1

For a g-literal $L = \langle X_i^*, a_1^*, \ldots, a_n^*, Z_j^* \rangle$ we have $(L + \overline{L})(F) = 1$ and $(L \cdot \overline{L})(F) = 0$.

Proof

$$\begin{aligned}
(L + \overline{L})(F) &= L(F) + \overline{L}(F) \\
&= (X_i^* + \overline{X_i^*}) . a_1/N . a_2/N . \ \ldots \ . a_n/N \\
&\quad + (a_1^*/1 + \overline{a_1^*}/1) \cdot a_2/N \cdot a_3/N \cdot \ \ldots \ \cdot a_n/N \\
&\quad + \cdot \ \cdot \ \cdot \\
&\quad + (a_n^*/1 + \overline{a_n^*}/1) \\
&\quad + (a_n^*/0 + \overline{a_n^*}/0)
\end{aligned}$$

$$\begin{aligned}
= &(a_1/N + a_1^*/1 + \overline{a_1^*}/1 + a_1^*/0 + \overline{a_1^*}/0) \\
&\cdot a_2/N\cdot \ \ldots \ \cdot a_n/N \\
&+ a_2^*/1 + \overline{a_2^*}/1 + a_2^*/0 + \overline{a_2^*}/0) \\
&\cdot a_3/N\cdot \ \ldots \ \cdot a_n/N \\
&+ \cdot \cdot \cdot \\
&+ (a_n^*/1 + \overline{a_n^*}/1 + a_n^*/0 + \overline{a_n^*}/0)
\end{aligned} \qquad (1)$$

The first term of (1) equals to $a_2/N\cdot \ \ldots \ \cdot a_n/N$, this term disjuncted with the second term makes it $a_3N\cdot a_4/N\cdot \ \ldots \ \cdot a_n/N$. Continuing the operation we will have $a_n/N + a_n^*/1 + \overline{a_n^*}/1 + a_n^*/0 + \overline{a_n^*}/0$ which is 1.

$$(L\cdot\overline{L})(F) = L(F)\cdot\overline{L}(F) \qquad (2)$$

Since a lead can be in only one of the three states, i.e., stuck at one, stuck at zero and fault free the conjunct of different faulty states of the same lead vanishes. We have

$$\begin{aligned}
(L\cdot\overline{L})(F) = &(X_i^*\cdot\overline{X_i^*})\cdot a_1/N\cdot a_2/N\cdot \ \ldots \ \cdot a_n/N \\
&+ (a_1^*/1\cdot\overline{a_1^*}/1 + a_1^*/0\cdot\overline{a_1^*}/0)\cdot a_2/N\cdot \ \ldots \ \cdot a_n/N \\
&+ \cdot \cdot \cdot \\
&+ (a^*/1\cdot\overline{a^*}/1 + a^*/0\cdot\overline{a^*}/0)
\end{aligned} \qquad (3)$$

Each term in the right hand side of (3) is logical zero. Hence we have the second part of the lemma.

Q.E.D.

<u>Definition 2.10</u>

Let E be a Boolean expression of g-literals, <u>the complement of E</u>, denoted as $\overline{E}$, is also a Boolean expression of g-literals with the operations of conjunct and disjunct in E interchanged and all g-literals in E complemented.

<u>Example 2.5</u>

Find $\overline{E}'$ for the network of Figure 2.4.

Using Equation (1) of Example 2.3 of Section 2.3 we first find $\overline{E}_1$ then reduce $\overline{E}_1$ to two-level sum of products form, i.e.,

$$
\begin{aligned}
E_1' = {} & \langle X_1,1,1a,6,\overline{9},\overline{12},\overline{Z_1}\rangle\cdot\langle X_2,2,2a,6,\overline{9},\overline{12},\overline{Z_1}\rangle \\
& \cdot\langle X_3,3,\overline{7},\overline{7b},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle\cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle \\
& + \langle X_1,1,1a,6,\overline{9},\overline{12},\overline{Z_1}\rangle\cdot\langle X_2,2,2a,6,\overline{9},\overline{12},\overline{Z_1}\rangle \\
& \cdot\langle \overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle\cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle \\
& + \langle X_3,3,\overline{7},\overline{7b},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle\cdot\langle \overline{X_3},\overline{3},7,7a,\overline{9},\overline{12},\overline{Z_1}\rangle \\
& \cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle \\
& + \langle \overline{X_3},\overline{3},7,7a,\overline{9},\overline{12},\overline{Z_1}\rangle\cdot\langle \overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle \\
& \cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11a},\overline{12},\overline{Z_1}\rangle
\end{aligned}
$$

Complementing Eq.(2) of Example 2.3 of Section 2.3 and reducing it to the sum of products form, we get

$$
\begin{aligned}
\overline{E}_2' = {} & \langle \overline{X_1},\overline{1},\overline{1b},\overline{8},\overline{13},\overline{Z_2}\rangle\cdot\langle \overline{X_2},\overline{2},\overline{2b},\overline{8},\overline{13},\overline{Z_2}\rangle \\
& + \langle X_3,3,\overline{7},\overline{7b},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle\cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle \\
& + \langle \overline{X_4},\overline{4},\overline{10},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle\cdot\langle \overline{X_5},\overline{5},\overline{11},\overline{11b},\overline{13},\overline{Z_2}\rangle
\end{aligned}
$$

The required expression is

$$\overline{E}' = (\overline{E}_1' \; \overline{E}_2')^T$$

Z_j-0 sets and GEM_j-0 sets of Figure 2.4 are listed in Table 2.3(b) and Table 2.4(b), respectively.

Inspecting Algorithm G we can find that if we start with two-element p^0-path sets at Step G3, call it Algorithm G', we will end up with Boolean expression $\overline{E}$. The following three theorems are the duals of Theorems 2.1-2.3 which we state without proof.

Theorem 2.1'

All g^0-literals for the combinational network are generated by Algorithm G'. Each appears in the resultant expression $\overline{E}$ exactly once, and all path sets appearing in the final expression are g^0-literals.

Theorem 2.2'

Algorithm G' transforms a given network structure into a Boolean expression which accurately describes the complement of the logical structure of the network.

Theorem 2.3'

The Boolean expression $\overline{E}$ generated by Algorithm G' is isomorphic to the physical structure and its corresponding input-output literals of the network to the level of interconnected gates.

2.5 CHAPTER SUMMARY AND REMARKS

An algebraic network model which describes accurately and completely the physical and logical structure of an object logic network has been rigorously presented. It is a substantial extension of GEM of Reese and McCluskey, even though the name of their model is adopted.

The differences between our model and theirs are: (1) Our model is for multiple-output logic networks, theirs for single-output networks only. (2) We added the rigorous definition of the complement of the Boolean expression of g-literals, without which the other side of the logical structure remains untold. (3) The functional equivalent of a g-literal is formally defined in our model. With this, the detailed information of a faulty network can be rigorously investigated.

One may note that we mentioned nothing about the complemented input literals. This would not impose a serious restriction on the model since either the complemented inputs can be generated by the network or we can rename the input provided by the system.

By substituting the functional equivalent of Definition 2.4 for the corresponding g-literals in the expressions E and $\overline{E}$, one can have more information about the logic network than that obtained by Poage's method. Finally, with all lead labels and output literals dropped from the Boolean expressions of g-literals we have the simplified cause-effect equations of Bossen and Hong [3].

CHAPTER 3

FAULT FUNCTIONS

3.1 INTRODUCTION

In this Chapter we shall deal with fault functions and related topics.

A fault function is a Boolean expression of input literals that a logic network realizes under the influence of a fault.

In addition to stuck-at type fault we shall discuss how to find fault functions for several kinds of bridging faults.

The difference between the fault-free function and fault function provides the tests. So, for a given fault we can first find the fault function, then a set of tests that detects the fault.

If we have a complete test set and a fault function, we can also either find the corresponding fault set or decide that the network can not realize the given fault function.

3.2 FAULT EQUIVALENCE

The immediate application of the model developed in Chapter 2 is the determination of the fault function E(F) realized by the network in the presence of fault F.

Since two different faults F_i and F_j may result in the same fault function, it will be convenient to put these two faults in a set and deal with the whole set or its representative in the studies of faulty logical networks.

<u>Definition 3.1</u>

Two faults F_i and F_j in a logical network are said to be <u>functionally equivalent</u>, written as $F_i \sim F_j$, if and only if their fault functions for that network are identical, i.e.,

$$F_i \sim F_j \iff E(F_i) = E(F_j).$$

<u>Example 3.1</u>

For the network of Figure 2.3 we have

$$\begin{aligned} E = {} & \langle X_1,1,1a,7,9,Z\rangle\cdot\langle X_2,2,4,4a,7,9,Z\rangle\cdot\langle X_3,3,4,4a,7,9,Z\rangle \\ & + \langle \overline{X_1},\overline{1},\overline{1b},5,8,9,Z\rangle\cdot(\langle \overline{X_2},\overline{2},\overline{4},\overline{4b},6,8,9,Z\rangle \\ & + \langle \overline{X}_3,\overline{3},\overline{4},\overline{4b},6,8,9,Z\rangle) \end{aligned}$$

Consider two faults: $F_1 = 4b/0$ and $F_2 = 6/1$. The fault functions are

$$E(F_1) = X_1X_2X_3 + \overline{X}_1$$

and

$E(F_2) = X_1X_2X_3 + \overline{X}_1$

Hence F_1 and F_2 are functionally equivalent in the network.

The functional equivalence of faults is an equivalence relation which partitions the collection of all possible faults F_I of a logic network into disjoint fault classes.

Definition 3.2

A logic network is said to contain a redundant fault if and only if there exists a fault $F_i \varepsilon F_I$ such that $i \neq 0$ and $F_i \sim F_0$, where F_0 contains no fault.

Example 3.2

In Figure 3.1 the logic network is a realization of Boolean function $Z_B = X_1X_2 + \overline{X}_2\overline{X}_3\overline{X}_4$ which has

$$\begin{aligned} E' = {} & <X_1,1,7,9,Z>\cdot<\overline{X}_2,\overline{2},\overline{2a},5,7,9,Z>\cdot<X_2,2,2b,6,6a,7,9,Z> \\ & + <X_1,1,7,9,Z>\cdot<\overline{X}_2,\overline{2},\overline{2a},7,9,Z>\cdot<X_3,3,6,6a,7,9,Z> \\ & + <X_1,1,7,9,Z>\cdot<\overline{X}_2,\overline{2},\overline{2a},7,9,Z>\cdot<X_4,4,6,6a,7,9,Z> \\ & + <\overline{X}_2,\overline{2},\overline{2b},\overline{6},\overline{6b},8,9,Z>\cdot<\overline{X}_3,\overline{3},\overline{6},\overline{6b},8,9,Z> \\ & \quad \cdot<\overline{X}_4,\overline{4},\overline{6},\overline{6b},8,9,Z> \end{aligned}$$

For fault $F_1 = 6a/1$ the fault function is

$E'(F_1) = X_1\overline{X}_2 + \overline{X}_2\overline{X}_3\overline{X}_4$

Since $E'(F_1) = X_1\overline{X}_2X_3 + X_1\overline{X}_2X_4 + \overline{X}_2\overline{X}_3\overline{X}_4 = Z_B$ we have $F_1 \sim F_0$ which implies the logic network is redundant, i.e., it contains redundant faults.

The redundant faults cannot be detected. This fault-masking property has been used in designing logic networks where continuous operation is required for a specified length of time and repair is impossible. In general, the existence of redundancy can not be readily identified. The detection and location of redundancy usually are the by-products of test generation.

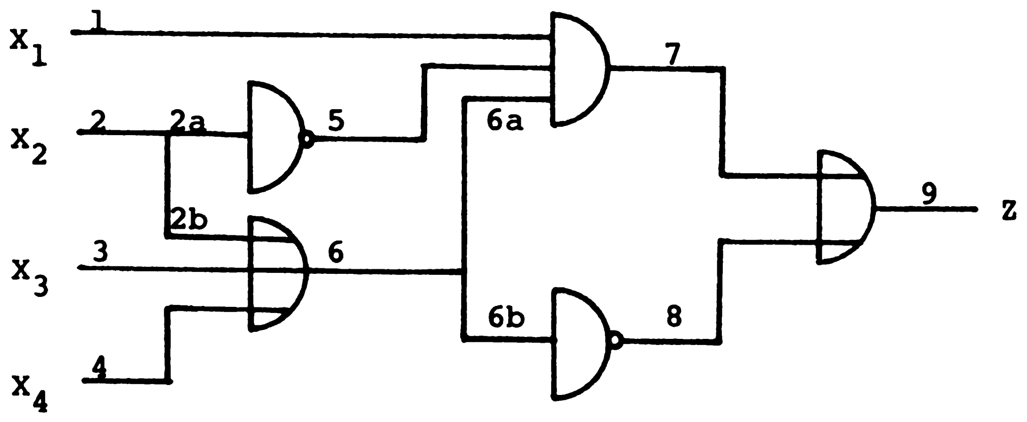


Figure 3.1. The Logic Network for Example 3.2.

Note that the fault 6a/0 in the above logic network can be detected.

3.3 GENERATION OF ALL POSSIBLE FAULT FUNCTIONS

Given a logic network we can find all possible fault functions and associated fault classes by assigning all combinations of the single faults of the network. The possible fault functions are the collection of all distinct functions that the network degenerates into and each fault class contains all faults of the same fault function. But, the astronomical number of all possible combinations of the single faults makes the enumeration method impractical.

Before the presentation of a straightforward method for the generation of all possible fault functions, we first establish an upper bound on the number of fault functions a given logic network can possibly have.

Theorem 3.1

The number of fault functions N_F for a logic network is bounded by

$$N_F \leq \text{Min}\ [(2^{2^{N_X}})^m,\ \prod_{i=1}^{m} (2^{G^i} + 1)]$$

where N_X is the number of input literals, m the number of output literals and G^i is the smaller number of the numbers of g-literals in E_i' and $\bar{E}_i'$.

Proof

There are only $2^{2^{N_X}}$ distinct functions of N_X binary variables which any single output logic network can possibly realize. Hence, for a m-output logic network $N_F \leq (2^{2^{N_X}})^m$.

For a $Z_{i,k}$-1 term which has G_k^i g^1-literals we can delete none, 1, 2, ... up to $G_k^i - 1$ g^1-literals by substituting 1 for the g^1-literals under consideration. We can also delete a $Z_{i,k}$-1 term by assigning 0 to it. If all the g^1-literals in the term are distinct we have $\prod_k 2^{G_k^i} = 2^{G^i}$ ways of deleting g^1-literals from a

E_i' expression. In addition, any term in E_i' which has a functional equivalent of 1 will drive output Z_i to 1. Thus, we have $(2^{G^i} + 1)$ distinct fault functions for output Z_i.

Since there is a one-to-one correspondence between the fault functions of E_i' and $\overline{E}_i'$, the total number of fault functions for a network will be bounded by using the smaller number of G^i's from each pair of E_i''s in our computation. The arguments for $\overline{E}_i'$ parallel those of E_i' in preceding two paragraphs.

Q.E.D.

Example 3.3

For the logic network of Figure 2.4 we have $N_X = 5$; $m = 2$; $G^1 = \text{Min}\ (7,10) = 7$ and $G^2 = \text{Min}\ (14,6) = 6$. Since $(2^{2^{N_X}}) \cong 2.97 \times 10^{19}$ and $(2^{G^1} + 1)(2^{G^2} + 1) = 8,385$, the upper bound for the number of fault functions for the network is 8,385.

The bound computed from the number of input literals is very loose even for a moderate size, single output logic network. It is interesting to note that the number of all possible combinations of the single faults of the illustrative network is 3^{21}, approximately 10^{10}.

Directly from Definition 2.4 we can find functional equivalents of a g-literal and the Boolean expression of g-literals under a given fault set. When we want to find all possible functional equivalents, i.e, fault functions, of a logic network, we simply let fault set F be unspecified and substitute the functional equivalent of each g-literal into the expression E, then reduce it into the sum of products form.

Definition 3.3

The range of a Boolean expression of g-literals, written as R(·), is the disjunct of the conjunct of all possible fault functions of the expression over their corresponding fault sets.

Example 3.4

The range of the term

$z_{1,1}\text{-}1 = \langle \overline{X}_1, \overline{1}, \overline{1a}, 6, 9, 12, z_1\rangle \cdot \langle X_3, 3, \overline{7}, \overline{7a}, 9, 12, z_1\rangle$ is

$$\begin{aligned}
&\overline{X}_1 X_3 \cdot (1/N \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N) \\
&+ \overline{X}_1 \cdot (1/N \cdot 1a/N \cdot 3/1 \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/N \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/0 \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/N \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/0 \cdot 9/N \cdot 12/N) \\
&+ X_3 \cdot (1/0 \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/N \cdot 1a/0 \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/N \cdot 1a/N \cdot 3/N \cdot 6/0 \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N) \\
&+ 1 \cdot (1/0 \cdot 1a/N \cdot 3/1 \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/0 \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/0 \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1/0 \cdot 1a/N \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/0 \cdot 9/N \cdot 12/N \\
&\qquad + 1a/0 \cdot 3/1 \cdot 6/N \cdot 7/N \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1a/0 \cdot 3/N \cdot 6/N \cdot 7/0 \cdot 7a/N \cdot 9/N \cdot 12/N \\
&\qquad + 1a/0 \cdot 3/N \cdot 6/N \cdot 7/N \cdot 7a/0 \cdot 9/N \cdot 12/N
\end{aligned}$$

+ 3/1•6/0•7/N•7a/N•9/N•12/N

+ 3/N•6/0•7/0•7a/N•9/N•12/N

+ 3/N•6/0•7/N•7a/0•9/N•12/N + 9/1•12/N + 12/1)

+ (1/1•1a/N•3/N•6/N•7/N•7a/N•9/N•12/N

+ 1a/1•3/N•6/N•7/N•7a/N•9/N•12/N + 7/1•7a/N•9/N•12/N

+ 3/0•6/N•7/N•7a/N•9/N•12/N + 7a/1•9/N•12/N

+ 6/1•7/N•7a/N•9/N•12/N + 9/0•12/N + 12/0)

Investigating the above result we conclude that any fault that affects the $Z_{1,1}$-1 term will transform it into one and only one of the five possible fault functions. The computation of the ranges for E_i' and E' is a trivial extension.

The fault set associated with each fault function contains the representatives of the fault class. Any combination of the single faults of a logic network that satisfies one fault in the fault set will transform the function realized by the network into the associated fault function. For example, there are approximately 3.5×10^9 combinations of the single faults which include 12/1 that cause output Z_1 of the network of Figure 2.4 to be stuck-at 1.

3.4 BRIDGING FAULTS

While the stuck-at type faults deal with individual leads, bridging faults are concerned with the connection of two or more leads of the logic network.

Definition 3.4

A bridging fault is a short circuit between two or more leads of a logic network and wired logic is performed at the point of connection.

We shall limit our studies to single two-lead bridging faults. The wired logic function is assumed to be either a wired AND or a wired OR.

If a bridging fault affects two leads on the same path in a logic network, it is a feedback bridging fault. A non-feedback bridging fault can affect at most one lead on one path.

In order to find the fault function of a logic network under a non-feedback bridging fault, we can (1) express the network output $E'(F_0)$ in terms of primary inputs and the lead variables of faulty leads. We call this the decomposition of the network output with respect to the lead variables; (2) express the lead variables of faulty leads in terms of primary inputs; (3) combine these lead variables with proper wired logic and substitute the result into the lead variables in the decomposed network output.

The first two steps of the preceding procedure can be carried out by first treating the faulty leads as primary input leads to the succeeding part of the network and letting the unaffected portion remain unchanged, then as the network output leads of the preceding part of the network and dropping the remaining portion from consideration.

Since the Boolean expression E' of the g-literals of a logic network can be regarded as a network output expression, one may want to know what is the decomposition of E' and how it helps us in finding the fault function realized by a logic network under the influence of a bridging fault.

Definition 3.5

The decomposition of the Boolean expression E' of the g-literals of a logic network with respect to the lead variable Y_i of lead i is obtained by: (1) Replacing the g-literals which have the form $<X_k^*, a_1^*, \ldots, i^* \ldots, a_n, Z_j>$ by $<Y_i^*, i^*, \ldots, a_n, Z_j>$, where the inversion parity of Y_i is the same as that of lead i. (2) Keeping the other g-literals in the expression unchanged.

Let E'_{Y_i} denote the decomposition of E' with respect to Y_i. It is not difficult to see that $E'_{Y_i}(F_0)$ is indeed the decomposition of the network output $E'(F_0)$ with respect to Y_i. Since for a g-literal which contains no i* as one of its elements, it cannot be decomposed with respect to Y_i and its effect on the network output remains unchanged. For a g-literal containing i*, the effect of lead variable Y_i on the network output depends on the inversion parity of lead i.

To express the lead variable in terms of the network primary inputs we can construct a Boolean expression for an isolated portion of the network as pointed out, or obtain it from the E' expression of the network as follows:

Lemma 3.1

The lead function Y_i for lead i can be determined from the Boolean expression E' of the g-literals of the logic network by the following procedure:

(1) For each $Z_{i,j}$-1 term which contains no g-literal which has i* as one of its elements, replace this term by 0.

(2) For each $Z_{i,j}$-1 term which has at least one g-literal contains i*, replace each such g-literal by the corresponding input literal, replace the other g-literals in this term by 1. Set the result equal to Y_i if i is non-overbarred, $\overline{Y}_i$ otherwise.

(3) Simplify the resultant expression by Boolean rules.

Proof

Since only the inputs of the g-literals which have i* as one of their elements can have the effect on the lead variable Y_i, all other g-literals can be neglected in the evaluation.

We have to find Y_i and $\overline{Y}_i$ separately and treat i and $\overline{i}$ independently because by definition i cannot generate $\overline{Y}_i$ and $\overline{i}$ cannot generate Y_i.

Q.E.D.

While the stuck-at type faults have the effect of simplifying the logic topology, the bridging faults introduce connections and wired gates to the network and complicate the network topology.

Definition 3.5 and Lemma 3.2 were established for a single lead. When two leads are involved as the case of bridging faults, we can deal with both the affected leads at one time if no g-literal contains more than one faulty lead, or decompose E' with respect to the lead variable whose lead is closer to the primary inputs, then subject the decomposed expression to be decomposed with respect to another lead variable if two leads in the same g-literal are connected.

Example 3.5

We denote wired AND between lead i and lead j by $B(i \cdot j)$ and wired OR by $B(i + j)$. For the logic network of Figure 2.3 the fault function due to the presence of bridging fault $F_1 = B(1b \cdot 6)$ can be computed as follows:

From Example 3.1 the sum of products expression of g-literals is

$$\begin{aligned} E' = {} & \langle X_1,1,1a,7,9,Z\rangle \cdot \langle X_2,2,4,4a,7,9,Z\rangle \\ & \cdot \langle X_3,3,4,4a,7,9,Z\rangle \\ + {} & \langle \overline{X}_1,\overline{1},\overline{1b},5,8,9,Z\rangle \cdot \langle \overline{X}_2,\overline{2},\overline{4},\overline{4b},6,8,9,Z\rangle \\ + {} & \langle \overline{X}_1,\overline{1},\overline{1b},5,8,9,Z\rangle \cdot \langle \overline{X}_3,\overline{3},\overline{4},\overline{4b},6,8,9,Z\rangle \end{aligned}$$

(a) Since no g-literal contains both lead 1b and lead 6, we can decompose E' with respect to Y_{1b} and Y_6 at one time. The result is

$$E'_{Y_{1b},Y_6} = \langle X_1,1,1a,7,9,Z\rangle\cdot\langle X_2,2,4,4a,7,9,Z\rangle \cdot\langle X_3,3,4,4a,7,9,Z\rangle + \langle Y_{1b},\overline{1b},5,8,9,Z\rangle\cdot\langle Y_6,6,8,9,Z\rangle.$$

The decomposed network output is

$$E'_{Y_{1b},Y_6}(F_0) = X_1X_2X_3 + \overline{Y}_{1b}Y_6 \qquad (1)$$

The lead variables in terms of primary inputs are found to be

$$Y_{1b} = X_1 \text{ and } Y_6 = \overline{X}_2 + \overline{X}_3.$$

The output of the wired gate is

$$Y_g = Y_{1b}\cdot Y_6 = X_1\overline{X}_2 + X_1\overline{X}_3 \qquad (2)$$

Substituting (2) into Y_{1b} and Y_6 of (1), we get the fault function

$$E'_{Y_{1b},Y_6}(F_1) = X_1X_2X_3 \qquad (3)$$

(b) We can also treat one lead at one time. The results are

$$E'_{Y_{1b}} = \langle X_1,1,1a,7,9,Z\rangle\cdot\langle X_2,2,4,4a,7,9,Z\rangle \cdot\langle X_3,3,4,4a,7,9,Z\rangle$$

$$+ \langle Y_{1b}, \overline{1b}, 5, 8, 9, Z\rangle \cdot \langle \overline{X}_2, \overline{2}, \overline{4}, \overline{4b}, 6, 8, 9, Z\rangle$$
$$+ \langle \overline{Y}_{1b}, \overline{1b}, 5, 8, 9, Z\rangle \cdot \langle \overline{X}_3, \overline{3}, \overline{4}, \overline{4b}, 6, 8, 9, Z\rangle \qquad (4)$$

$$E'_{Y_6} = \langle Y_1, 1, 1a, 7, 9, Z\rangle \cdot \langle X_2, 2, 4, 4a, 7, 9, Z\rangle \cdot \langle X_3, 3, 4, 4a, 7, 9, Z\rangle$$
$$+ \langle \overline{X}_1, \overline{1}, \overline{1b}, 5, 8, 9, Z\rangle \cdot \langle Y_6, 6, 8, 9, Z\rangle$$
$$+ \langle \overline{X}_1, \overline{1}, \overline{1b}, 5, 8, 9, Z\rangle \cdot \langle Y_6, 6, 8, 9, Z\rangle \qquad (5)$$

If we either decompose (4) with respect to Y_6 or decompose (5) with respect to Y_{1b}, we will obtain the same result as that of (a). The order of decomposition in the case of non-feedback bridging is of no importance, e.g., $E'_{Y_{1b},Y_6} = E'_{Y_6,Y_{1b}}$.

A feedback bridging fault can be classified as <u>inverting</u> or <u>non-inverting</u> depending on whether the lead labels in the g-literal have different inversion parities or not. The following example will demonstrate how to find the fault functions caused by feedback bridging faults.

<u>Example 3.6</u>

The logic network of Figure 2.3 will be investigated further in this example.

(a) The bridging fault $F_2 = B(4 \cdot 7)$ is non-inverting because both lead labels in the second and the third g-literals of the first term of expression E' (see Example 3.5) have the same inversion parity.

Since lead 4 is closer to the primary input, we have to decompose E' with respect to Y_4 first.

$$E'_{Y_4} = <X_1,1,1a,7,9,Z>\cdot<Y_4,4,4a,7,9,Z> \cdot<Y_4,4,4a,7,9,Z> + <\overline{X}_1,\overline{1},\overline{1b},5,8,9,Z>\cdot<\overline{Y}_4,\overline{4},\overline{4b},6,8,9,Z> \quad (1)$$

$$E'_{Y_4}\ (F_0) = X_1Y_4 + \overline{X}_1\overline{Y}_4 \quad (2)$$

$$Y_4 = X_2X_3 \quad (3)$$

Then, we decompose E'_{Y_4} with respect to Y_7

$$E'_{Y_4,Y_7} = <Y_7,7,9,Z> + <\overline{X}_1,\overline{1},\overline{1b},5,8,9,Z>\cdot<\overline{Y}_4,\overline{4},\overline{4b},6,8,9,Z> \quad (4)$$

$$E'_{Y_4,Y_7}(F_0) = Y_7 + \overline{X}_1\overline{Y}_4 \quad (5)$$

$$Y_7 = X_1Y_4 \quad (6)$$

The first wired gate output after the application of the primary inputs is

$$Y_{g1} = X_1X_2X_3 \quad (7)$$

Let the superscript s of $E'_{Y_4,Y_7}(F_2^s)$ denote the sequence of output under the influence of bridging fault F_2. Substitute (3) into Y_4 of (5) and (6), then (6) into Y_7 of (5), we have

$$E'_{Y_4,Y_7}(F_2^1) = X_1X_2X_3 + \overline{X}_1\overline{X}_2 + \overline{X}_1\overline{X}_3 \quad (8)$$

which is the fault free output of the network.

Since the wired gate output will remain at (7), substituting (7) into both Y_4 and Y_7 of (5), we get the right hand side of (8) again. This implies (8) can be regarded as the required fault function. The fault F_2 happens to be undetectable.

(b) The fault $F_3 = B(1b \cdot 8)$ is an inverting feedback bridging fault since lead 1b and lead 8 have different inversion parities in a g-literal of E'.

After a few computations, we have

$$E'_{Y_{1b}}(F_0) = X_1X_2X_3 + \bar{Y}_{1b}\bar{X}_2 + \bar{Y}_{1b}\bar{X}_2 \tag{1}$$

$$Y_{1b} = X_1 \tag{2}$$

$$E'_{Y_{1b},Y_8}(F_0) = X_1X_2X_3 + Y_8 \tag{3}$$

$$Y_8 = \bar{Y}_{1b}\bar{X}_2 + \bar{Y}_{1b}\bar{X}_3 \tag{4}$$

$$Y_g = Y_{1b}Y_8 \tag{5}$$

The first output of the wired gate is obtained by first substituting (2) into Y_{1b} of (4) and (5), then (4) into Y_8 of (5), i.e.,

$$Y_{g1} = X_1(\bar{X}_1\bar{X}_2 + \bar{X}_1\bar{X}_3) = 0 \tag{6}$$

Substitute (6) into Y_8 of (3), we have

$$E'_{Y_{1b},Y_8}(F_3^1) = X_1X_2X_3 \tag{7}$$

The wired gate output will remain at (6), which in turn keep the network out at (7). (7) is the required fault function.

In the above examples, all three bridging faults generated no sequential dependency in the logic network. In some cases, we may have to compute the network output for a number of feedback periods to establish the fault functions.

It is interesting to note that bridging fault F_1 of Example 3.5 is equivalent to bridging fault F_3 of Example 3.6. Furthermore, the logic network of Figure 2.3 is redundant with respect to bridging faults since fault F_2 of Example 3.6 cannot be detected, even though it is irredundant when only stuck-at type faults are considered.

3.5 DIRECT DIFFERENCE AND TEST GENERATION

A fault in a logic network can be detected only if it causes the network to realize a function, called the fault function, which is different from the intended function, i.e., fault-free function.

A fault can be located, to the level of interest, if it can be detected and the differences between this fault and all other faults of the network can be identified.

Definition 3.6

The direct difference D_{ij} of fault F_i and fault F_j is $E(F_i) \oplus E(F_j)$, the exclusive-OR of the respective fault functions.

The solution of $D_{0j} = 1$ is the test set which detects the existence of fault F_j. A set of input vectors which covers the solutions of $D_{0j} = 1$ for all $j \neq 0$ of the network is a complete detection test set. A set of input vectors which covers the solutions of $D_{ij} = 1$ for all i,j of the network and distinguishes each pair of the solutions is a complete location test set.

Example 3.7

The logic network of Figure 2.2 (p. 11) realizes the function $E'(F_0) = X_1X_2X_3X_4 + \overline{X}_1\overline{X}_2\overline{X}_3\overline{X}_4$. The Boolean expression E of the network is

$$\begin{aligned}
E = {} & (\langle \overline{X}_1, \overline{1}, \overline{1a}, 5, \overline{8}, 12, Z\rangle \cdot \langle \overline{X}_3, \overline{3}, \overline{3a}, 5, 8, 12, Z\rangle \\
& \quad + \langle X_2, 2, 2a, \overline{8}, 12, Z\rangle)(X_1, 1, 1b, \overline{9}, 12, Z\rangle + \\
& \quad\quad \langle \overline{X}_2, \overline{2}, \overline{2b}, \overline{2ba}, 6, 6a, \overline{9}, 12, Z\rangle \cdot \langle \overline{X}_3, \overline{3}, \overline{3a}, \overline{3ab}, 6, 6a, \overline{9}, 1], Z\rangle) \\
& \quad\quad (\langle \overline{X}_2, \overline{2}, \overline{2b}, \overline{2ba}, 6, 6b, 10, 12, Z\rangle \\
& \quad\quad \cdot \langle \overline{X}_3, \overline{3}, \overline{3a}, \overline{3ab}, 6, 6b, 10, 12, Z\rangle + \langle X_4, 4, 4a, \overline{10}, 12, Z\rangle) \\
& \quad\quad (\langle \overline{X}_2, \overline{2}, \overline{2b}, \overline{2bb}, 7, \overline{11}, 12, Z\rangle \cdot \langle \overline{X}_4, \overline{4}, \overline{4b}, 7, 11, 12, Z\rangle \\
& \quad + \langle X_3, 3, 3b, 11, 12, Z\rangle)
\end{aligned}$$

For the fault $F_1 = 6/0$, we have $E(F_1) = X_1X_2X_3X_4$. The solution of $D_{01} = E(F_0) \oplus E(F_1) = \overline{X}_1\overline{X}_2\overline{X}_3\overline{X}_4 = 1$ is

0000. This is the only input vector which can detect the existence of the 6/0 fault.

To use the direct difference method in test generation, we have to know the fault sets, the fault functions, and the corresponding direct differences. This makes it impractical for the generation of complete test set. However, this method is still valuable in generating test set for a smaller number of the faults of particular interest, such as the 6/0 fault of above example and the bridging faults discussed in Section 3.4.

The generation of a complete test set for a logic network will be discussed in Chapter 4.

3.6 IDENTIFICATION OF FAULTS

Given a fault in a logic network we can find the corresponding fault function simply by substituting the functional equivalents of the g-literals into the Boolean expression E of the network. Conversely, given a fault function we can find the associated fault set if we have a complete test set for the network.

Before we proceed to describe a procedure to find the required fault set, it is worth noting that if the range of the Boolean expression E is already found, the problem reduces to the identification of the fault function which conjoined with the associated fault set is a term of the range. If we want to know whether there is a fault which

generates a particular fault function, the procedure is exactly the same except if there is no such fault the search of the fault function in the range will fail.

Instead of computing the range of the Boolean expression E then identifying the fault function to find the fault set, we can: (1) Substitute each g-literal in E_i by the range of the corresponding g-literal, then set the result equal to the fault function $E_i(F)$. (2) Evaluate both sides of the m equations by a complete test set which contains t input vectors. (3) Solve the set of mt simultaneous equations. The solution is the required fault set. If there is no solution, we conclude that the logic network can not degenerate into the given fault function.

The above procedure is straightforward and can be machine implemented. But to deal with every lead of the logic network the job will be enormous even for one of moderate size. Looking for the ways of simplification, we have the following theorem.

<u>Theorem 3.2</u>

Any multiple fault in a combinational logic network can be represented by a combination of the faults in the distinct g-literals of the network.

<u>Proof</u>

From Chapter 2 we know each g-literal of the Boolean expression E generated by Algorithm G is

distinct. The expression E describes accurately and completely the logical and physical structures of the network. The effects of a fault on E, hence the network, is the Boolean expression of the effects of the fault on the distinct g-literals.

Q.E.D.

Since a g-literal can assume only one of the three states as a lead in the network, it will be convenient to assign a distinct number to a distinct g-literal and treat the simplified g-literal as a single lead g-literal.

Definition 3.7

The simplified g-literal of a g-literal $L = \langle X_i^*, a_1^*, \ldots, a_n^*, Z_j^* \rangle$ is the three element g-literal $L_s = \langle X_i^*, A^*, Z_j^* \rangle$, where A is a number representing the sequence of lead labels in L and it is overbarred if $Z_j^* = \overline{Z}_j$, non-overbarred otherwise.

In the above definition we treated the distinct g-literals in $\overline{E}$ as the complements of their counterparts in E. The Boolean expressions of the simplified g-literals will be denoted by the corresponding notations, with subscript s, used for the g-literals, e.g., E_s, E_{is}.

The range of the simplified g-literal $L_s = \langle X_i^*, A^*, Z_j^* \rangle$ is $R(L_s) = X_i^* \cdot A/N + A^*/1 + A^*/0$ where $A/N = a_1/N \cdot \ldots \cdot a_n/N$, $A/1 = a_1^*/B_1 \cdot a_2/N \cdot \ldots \cdot a_n/N + \ldots + a_{n-1}^*/B_{n-1} \cdot a_n/N + a_n^*/B_n$ where $B_k = 1$ if $a_k^* = a_k$, 0 otherwise. $A/0 = a_1^*/C_1 \cdot a_2/N \cdot \ldots \cdot a_n/N + \cdots + a_{n-1}^*/C_{n-1} \cdot a_n/N + a_n^*/C_n$ where $C_k = 1$ if $a_k^* = \overline{a}_k$, 0 otherwise. Note that $\overline{A}/1 = A/0$ and $\overline{A}/0 = A/1$.

Example 3.8

For the logic network of Figure 2.3, we find the fault set associated with the fault function $E(F) = X_1$.

Referring to Example 3.1 for the E expression of the network, assign 1 to 6 sequentially to each of the six distinct g-literals. Substitute the range of each simplified g-literal in the expression and set it equal to the fault function. We have

$$(X_1\cdot 1/N + 1/1)(X_2\cdot 2/N + 2/1)(X_3\cdot 3/N + 3/1) + (\overline{X}_1\cdot 4/N + 4/1)((\overline{X}_2\cdot 5/N + 5/1) + (\overline{X}_3\cdot 6/N + 5/1)) = X_1 \qquad (1)$$

The set T = (001, 010, 011, 101, 111) of input vectors is a minimal complete detection test set for the network. Evaluating (1) by the elements of T, we get

$$1/1\cdot 2/1\cdot 3/N + 1/1\cdot 2/1\cdot 3/1 + 4/N\cdot 5/N + 4/N\cdot 5/1 + 4/1\cdot 5/N + 4/1\cdot 5/1 + 4/N\cdot 6/1 + 4/1\cdot 6/1 = 0 \qquad (2)$$

$$1/1\cdot 2/N\cdot 3/1 + 1/1\cdot 2/1\cdot 3/1 + 4/N\cdot 5/1 + 4/1\cdot 5/1 + 4/N\cdot 6/N + 4/N\cdot 6/1 + 4/1\cdot 6/N + 4/1\cdot 6/1 = 0 \qquad (3)$$

$$1/1\cdot 2/N\cdot 3/N + 1/1\cdot 2/N\cdot 3/1 + 1/1\cdot 2/1\cdot 3/N + 1/1\cdot 2/1\cdot 3/1 + 4/N\cdot 5/1 + 4/1\cdot 5/1 + 4/N\cdot 6/1 + 4/1\cdot 6/1 = 0 \qquad (4)$$

$$1/N \cdot 2/1 \cdot 3/N + 1/N \cdot 2/1 \cdot 3/1 + 1/1 \cdot 2/1 \cdot 3/N + 1/1 \cdot 2/1 \cdot 3/1$$
$$+ 4/1 \cdot 5/N + 4/1 \cdot 5/1 + 4/1 \cdot 6/1$$
$$= 1 \tag{5}$$

$$1/N \cdot 2/N \cdot 3/N + 1/N \cdot 2/N \cdot 3/1 + 1/N \cdot 2/1 \cdot 3/N + 1/1 \cdot 2/N \cdot 3/N$$
$$+ 1/N \cdot 2/1 \cdot 3/1 + 1/1 \cdot 2/N \cdot 3/1 + 1/1 \cdot 2/1 \cdot 3/N + 1/1 \cdot 2/1 \cdot 3/1$$
$$+ 4/1 \cdot 5/1 + 4/1 \cdot 6/1$$
$$= 1 \tag{6}$$

The simultaneous solution of (2) to (6) is

$$F_s = 1/N \cdot 2/1 \cdot 3/1 \cdot (4/0 + 5/0 \cdot 6/0) \tag{7}$$

for the simplified g-literals which implies

$$F = 4/1 \cdot (1b/1 + 5/0 + 8/0 + 4b/1 + 6/0)$$
$$+ 4a/1 \cdot (1b/1 + 5/0 + 8/0 + 2/1 \cdot 3/1 + 4b/1 + 6/0)$$

for the leads of the logic network.

If the fault function is $E(F_0)$, the fault free function, the solution for the simultaneous equations may contain just one situation, i.e., all leads are normal, which implies the network is irredundant. If the solution contains situations other than all leads are normal, the network is redundant and the extra solutions tell exactly where the redundancies are.

3.7 CHAPTER SUMMARY AND REMARKS

Through the fault equivalence defined in Section 3.2, we can treat the faults in fault classes instead of as individual faults. The upper bound on the number of possible

fault functions established in Section 3.3 gives us a rough idea how tedious it will be if we have to consider each fault function separately. However, we have a machine implementable procedure to find all possible fault functions and the associated fault sets.

The feedback bridging faults not only can be identified as suggested by Flomenhoft et al. [15], but the fault functions can also be computed as demonstrated in Section 3.4.

In Section 3.5, we discussed how to find a test set if the fault function can be found. From a given fault we can find the fault function, conversely, given a fault function we can identify the associated fault set.

For the faults which are essentially indistinguishable, we certainly do not have to compute their fault functions individually. In a network comprised of the elementary gates, they are:

Gate	Input leads a_is	Output lead a_j
AND	any $a_i/0$	$a_j/0$
NAND	any $a_i/0$	$a_j/1$
NOT	$a_i/0$	$a_j/1$
OR	any $a_i/1$	$a_j/1$
NOR	any $a_i/1$	$a_j/0$
NOT	$a_i/1$	$a_j/0$

In solving the simultaneous equations of Section 3.6 we can use the cubic intersection:

	a/0	a/1	a/N	X
a/0	a/0	∅	∅	a/0
a/1	∅	a/1	∅	a/1
a/N	∅	∅	a/N	a/N
X	a/0	a/1	a/N	X

where ∅ denotes empty and X denotes don't care. We can use the machine to solve a large set of simultaneous equations, but for a small number of equations solution by inspection may be faster.

CHAPTER 4

TEST GENERATION

4.1 INTRODUCTION

The Complete Gate Equivalent Model and its various forms will be used in this Chapter for the generation of complete test sets for the combinational logic network.

Four algorithms for generating near minimal complete test sets will be discussed. The first two will be for multiple and single fault detection. The other two are the direct generalizations of the first two to fault location.

A method of searching for a minimal complete test set for an irredundant combinational logic network also will be presented.

4.2 GENERATION OF NEAR MINIMAL COMPLETE DETECTION TEST SETS

The generation of a complete test set for detecting stuck-at type faults in a combinational logic network will be accomplished in two steps: (1) Generation of a near minimal set of terms that detects all detectable faults. (2) Finding a minimal covering of the terms generated.

4.2.1 Generation of a Near Minimal Test Set which Detects All Faults in a Combinational Logic Network

In the sequel we shall call E' under the fault free condition, but without any Boolean simplifications a simplified E' expression and denoted by E''. The notation $\overline{E}''$, E_i'', $\overline{E}_i''$, $Z_{i,\overline{k}}''$-1 and $Z_{i,\overline{k}}''$-0 will be used for their counterparts in E' and $\overline{E}'$. A Z''-1 or Z''-0 term is said to contain a conflict pair if it has $X_i\overline{X}_i$ as its member.

A growth in a $Z_{i,\overline{k}}''$-1 or $Z_{i,\overline{k}}''$-0 term is the enlargement of the set of nodes covered by the term on the N-cube, caused by the presence of some faults. A shrinkage of a $Z_{i,\overline{k}}''$-1 or $Z_{i,\overline{k}}''$-0 term is the reduction of the set of nodes covered by the term on the N-cube, caused by the presence of some faults.

In the descriptions of the algorithms and the procedure in this Chapter, the alternative arguments will be put in the parentheses.

Algorithm MFD (Multiple Fault Detection)

Data required for this algorithm are the expressions E'', $\overline{E}''$, E'(F_0) and $\overline{E}$'(F_0).

1: Select either E_i'' or $\overline{E}_i''$ which contains the smaller number of input literals from each of the m primary outputs of the network. If both E_i'' and $\overline{E}_i''$ have the same number of input literals, choose one arbitrarily.

2: Separate $Z''_{i,k}=1$ ($Z''_{i,k}=0$) terms in E'' ($\overline{E}''$) into two different sets such that one contains the terms which have conflict pairs and the other does not.

3: For those $Z''_{i,k}=1$ ($Z''_{i,k}=0$) terms which contain no conflict pairs

(a) Let i = 1.

(b) Compute

$$M_u = (Z''_{i,k}=1) \prod_{\text{all } j\neq k} (\overline{Z''_{i,j}=1})$$

$$(M_u = (Z''_{i,k}=0) \prod_{\text{all } j\neq k} (\overline{Z''=0}))$$

If M_u is nonempty put it in set M.

(c) Repeat Step 3(b) for all $Z''_{i,k}=1$ ($Z''_{i,k}=0$) terms in E''_i ($\overline{E}''_i$).

4: (a) If no M_u computed in Step 3 for E''_i ($\overline{E}''_i$) is empty, increase i by 1 otherwise go to Step 5.

(b) If not all E''_i ($\overline{E}''_i$) selected in Step 1 are investigated, go to Step 3(b) otherwise go to Step 6. Set M now contains the terms which detect any shrinkage of the coverings of the selected E''_i's and $\overline{E}''_i$'s.

5: Let there by s $Z''_{i,k}=1$ ($Z''_{i,k}=0$) terms in set S whose M_u computed in Step 3 for E''_i ($\overline{E}''_i$) are empty.

(a) Set n = 2.

(b) If n is greater than the number of terms in S, increase i by 1 and go to Step 4(b).

(c) For all combinations of n $Z''_{i,k}$-1 ($Z''_{i,k}$-0) terms from set S compute

$$M_u = \sum_{k \text{ selected}} (Z''_{i,k}\text{-}1) \prod_{j \text{ not selected}} \overline{(Z''_{i,j}\text{-}1)}$$

$$(M_u = \sum_{k \text{ selected}} (Z''_{i,k}\text{-}0) \prod_{j \text{ not selected}} \overline{(Z''_{i,j}\text{-}0)})$$

(d) Put all nonempty M_u's in set M, delete the corresponding Z''-1 (Z''-0) terms from set S and increase n by 1 go to (b).

6: Set i = 1 and j = 1.

7: Now we compute the terms which detect the growths caused by the faults. Let W denote a $Z''_{i,j}$-1 ($Z''_{i,j}$-0) term with one or more of its input literals replaced by 1.

(a) Replace one input literal in $Z''_{i,j}$-1 ($Z''_{i,j}$-0) by 1.

(b) Compute

$$M_j = W \cap \overline{E}'(F_0) \qquad (1)$$

$$(M_j = W \cap E'(F_0))$$

If M_u is not empty put it in set M.

(c) Repeat (a) through (b) for all input literals in this term.

8: (a) If no M_u computed in Step 7 for a term is empty increase j by 1 otherwise go to Step 9.

(b) If not all terms in E_i'' ($\bar{E}_i''$) are investigated go to Step 7 otherwise increase i by 1 and set j = 1.

(c) If all selected E_i'' ($\bar{E}_i''$) are processed, stop. Set M now contains the terms which detect the shrinkages and the growths of the coverings of the selected E_i'''s and $\bar{E}_i''$'s or else go to Step 7.

9: Let there be t input literals in set T whose M_u's are empty.

(a) Let q = 2.

(b) If q is greater than the number of input literals in T, delete all the literals from T, increase j by 1 and go to Step 8(b).

(c) Replace all combinations of q input literals by 1 and compute M_u by Eq. (1) of Step 7.

(d) Put all nonempty M_u in set M, delete the corresponding input literals from set T, increase q by 1 go to (b).

<u>Theorem 4.1</u>

A minimal single covering of all M_u's in set M generated by Algorithm MFD is a near minimal complete detection test set that detects all detectable faults in the logic network.

Proof

The functional equivalent of a g-literal affected by a stuck-at type fault can assume only one of the two values 1 or 0.

The functional equivalent of a $Z_{i,k}$-1 or $Z_{i,k}$-0 term will be 0, i.e. false if at least one of its g-literals has functional equivalent 0. This results in a smaller number of vertices covered by the output Z_i or the complement of Z_i. Hence it is a shrinkage.

The shrinkage can be detected only if it is not covered by the fault free realization and the growth caused by the same fault due to the common lead label appearing with different inversion parities in different g-literals. We computed the terms that detect the shrinkages which are not covered by the fault free realization in Step 3 through Step 5. The result, more than the least number of terms required for detecting all detectable shrinkages is generated.

A term which contains conflict pairs covers no vertices originally, so we can neglect it in the generation of the terms which detect the faults that cause shrinkages.

If a g-literal in a term has functional equivalent of 1, it causes this term to cover more vertices than the fault free situation. Hence it is a growth. A growth will be detected if it is not covered by the fault free realization. This implies a detectable growth has a nonempty intersection with the complement of the original function.

A fault will be detected if it contains a detectable fault of smaller size on the N-cube, whether it is a growth or a shrinkage. We took advantage of this property to terminate the algorithm.

Step 1 is to ensure that a minimal number of computations is required. The fact that a minimal single covering is a near minimal complete test set is because the terms for shrinkages are more than necessary and the terms for growths of the same input literal in different terms are not merged into one subset. Hence they may be covered more than once.

Q.E.D.

Example 4.1

Find a complete test set that detects all faults in the logic network of Figure 2.4.

First, we use Algorithm MFD to generate the terms which detect all faults in the network.

Step 1: Referring to Examples 2.4 and 2.5, we select E_1'' and $\overline{E}_2''$ for the computations, where

$$E_1'' = \overline{X}_1X_3 + \overline{X}_2X_3 + \overline{X}_3X_4 + X_5 \qquad (1)$$

and

$$\overline{E}_2'' = \overline{X}_1\overline{X}_2 + X_3\overline{X}_5 + \overline{X}_4\overline{X}_5 \qquad (2)$$

Step 2: Since no term in Eqs. (1) and (2) contains conflict pairs, all the terms have to be investigated in the generation of terms which detect shrinkages.

Step 3: The first iteration processes 4 terms of E_1''.

$$M_1 = \bar{X}_1 X_3 (\overline{\bar{X}_2 X_3}) (\overline{\bar{X}_3 X_4}) \bar{X}_5 = \bar{X}_1 X_2 X_3 \bar{X}_5$$

$$M_2 = \bar{X}_2 X_2 (\overline{\bar{X}_1 X_3}) (\overline{\bar{X}_3 X_4}) \bar{X}_5 = X_1 \bar{X}_2 X_3 \bar{X}_5$$

$$M_3 = \bar{X}_3 X_4 (\overline{\bar{X}_1 X_3}) (\overline{\bar{X}_2 X_3}) \bar{X}_5 = \bar{X}_3 X_4 \bar{X}_5$$

$$M^4 = X_5 (\overline{\bar{X}_1 X_3}) (\overline{\bar{X}_2 X_3}) (\overline{\bar{X}_3 X_4}) = X_1 X_2 X_3 X_4 + \bar{X}_3 \bar{X}_4 X_5$$

Step 4: All M_u's computed for E_1'' are nonempty, so we can proceed to consider $\bar{E}_2''$. The result of the second iteration of Step 3 is:

$$M_5 = \bar{X}_1 \bar{X}_2 \bar{X}_3 X_4 + \bar{X}_1 \bar{X}_2 X_5 \quad ; \quad M_6 = X_1 X_3 X_4 \bar{X}_5 + X_2 X_3 X_4 \bar{X}_5$$

$$M_7 = X_1 \bar{X}_3 \bar{X}_4 \bar{X}_5 + X_2 \bar{X}_3 \bar{X}_4 \bar{X}_5$$

All M_u's computed in Step 3 for $\bar{E}_2''$ also are nonempty, so put them in set M and go to Step 6.

Step 6: In order to compute the terms that detect the growth of the network function, we have to consider all terms regardless of whether they contain conflict pairs or not. For the problem concerned they happen to be all the terms we investigated so far.

Step 7: The fault free functions to be used in the following computations are

$$\overline{E}'(F_0) = X_1X_2X_3\overline{X}_5 + X_1X_2\overline{X}_4\overline{X}_5 + \overline{X}_3\overline{X}_4\overline{X}_5$$

and

$$E_2'(F_0) = X_1\overline{X}_3X_4 + X_1X_5 + X_2\overline{X}_3X_4 + X_2X_5$$

Step 8: Since all terms computed from each single growth of all terms selected are nonempty, the algorithm terminates without going to Step 9. From $\overline{X}_1X_3$ we have

$$M_8 = X_3 \cap \overline{E}'(F_0) = X_1X_2X_3\overline{X}_5$$

$$M_9 = \overline{X}_1 \cap \overline{E}'(F_0) = \overline{X}_1\overline{X}_3\overline{X}_4\overline{X}_5$$

and

From this term	The M_u's obtained
$\overline{X}_2X_3$	$M_{10} = X_1X_2X_3\overline{X}_5$; $M_{11} = \overline{X}_2\overline{X}_3\overline{X}_4\overline{X}_5$
$\overline{X}_3X_4$	$M_{12} = X_1X_2X_3X_4\overline{X}_5$; $M_{13} = \overline{X}_3\overline{X}_4\overline{X}_5$
X_5	$M_{14} = \overline{E}_1'(F_0)$
$\overline{X}_1\overline{X}_2$	$M_{15} = X_1\overline{X}_2\overline{X}_3X_4 + X_1\overline{X}_2X_5$;
	$M_{16} = \overline{X}_1X_2\overline{X}_3X_4 + X_1\overline{X}_2X_5$
$X_3\overline{X}_5$	$M_{17} = X_1\overline{X}_3X_4\overline{X}_5 + X_2\overline{X}_3X_4\overline{X}_5$
	$M_{18} = X_1X_3X_5 + X_2X_3X_5$
$\overline{X}_4\overline{X}_5$	$M_{19} = X_1\overline{X}_3X_4\overline{X}_5 + X_2\overline{X}_3X_4\overline{X}_5$
	$M_{20} = X_1\overline{X}_4X_5 + X_2\overline{X}_4X_5$

A minimal set of input vectors that covers all M_u's generated is (00101, 01000, 01010, 01110, 10000, 10110, 11101, 1110) which can be found by row-column dominance and other covering techniques.

In reading the example above, note that it is not necessary to compute M_{14} if we recorded the growth of each term and found there is a detectable growth in E_1''. We also do not have to compute M_{10} and M_{19} if we note that the growths into the X_3 direction and the $\overline{X}_5$ direction have been investigated. Although these observations are feasible for hand computations the tradeoffs necessary for machine implementation remain to be assessed. Since the covering set is found for all M_u's, some vectors of the set may detect both the growth and the shrinkage.

4.2.2 Generation of a Near Minimal Test Set which Detects All Single Faults in a Combinational Logic Network

Algorithm MFD certainly will generate all terms that detect all single faults in the logic network, but for a possibly smaller set of terms and a smaller complete test set, we can simplify it by means of the single fault assumption.

In order to obtain some simplifications in Algorithm MFD we need more information about the structure of the logic network. The algorithm requires as data the lead labels in the g-literals.

<u>Algorithm SFD (Single Fault Detection)</u>

Data required for this algorithm are the expressions E', $\overline{E}'$, E'', $\overline{E}''$, $E'(F_0)$ and $\overline{E}'(F_0)$.

1: Select either E_i' or $\overline{E}_i'$ which contains the smaller number of g-literals for each of the m primary outputs of the network. If both have same number of g-literals, choose one arbitrarily.

2: Separate $Z_{i,\overline{k}}'\overline{}1$ $(Z_{i,\overline{k}}'\overline{}0)$ terms in E_i' $(\overline{E}_i')$ into two different sets. One contains the terms which contain g-literals have the same input literals with different inversion parities and the other does not.

3: For those terms in E_i'' $(\overline{E}_i'')$ which contains no conflict pairs:

(a) Let i = 1.

(b) Compute

$$M_u = (Z_{i,\overline{k}}''\overline{}1) \prod_{\text{all } j\neq k} \overline{(Z_{i,\overline{j}}''\overline{}1)}$$

$$(M_u = (Z_{i,\overline{k}}''\overline{}0) \prod_{\text{all } j\neq k} \overline{(Z_{i,\overline{j}}''\overline{}0)})$$

If M_u is nonempty, put it in set M.

(c) Repeat (b) for all terms in E_i'' $(\overline{E}_i'')$.

4: (a) If no M_u computed for each term is empty, increase i by 1 otherwise go to Step 5.

(b) If not all outputs of the network are considered to to Step 3(b) otherwise go to Step 6. Now set M contains the terms which detect any shrinkage of the coverings of the selected $E_i^{\prime\prime}$'s and $E_i^{\prime\prime}$'s caused by any single fault in the network.

5: Let there be s terms whose M_u's are nonempty and t terms whose M_u's are empty. Put their corresponding g-literals in set S and set T, respectively.

(a) If all g-literals in a term of T are covered by some terms of S, delete this term from T. Repeat this investigation for every term of T.

(b) For each common lead label, with same inversion parity, contained in two or more terms in T: Compute

$$M_u = \sum_{k\varepsilon K} (Z_{i,k}^{\prime\prime}\neq 1) \prod_{j\not\varepsilon K} \overline{(Z_{i,j}^{\prime\prime}\neq 1)}$$

$$(M_u = \sum_{k\varepsilon K} (Z_{i,k}^{\prime\prime}\neq 0) \prod_{j\not\varepsilon K} \overline{(Z_{i,j}^{\prime\prime}\neq 0)})$$

where K contains the numbers of the terms which have the common lead label.

(c) Increase i by 1, go to Step 3(b).

6: Remove all g-literals from set S and T. Set i = 1 and j = 1.

7: Let W denote a $Z_{i,j}^{\prime\prime}\neq 1$ $(Z_{i,j}^{\prime\prime}\neq 0)$ term with one or more of its input literals replaced by 1.

(a) Replace one input literal in the $Z_{i,j}^{\prime\prime}\neq 1$ $(Z_{i,j}^{\prime\prime}\neq 0)$ term by 1. Compute

$$M_u = W \cap \overline{E}'_i(F_0)$$

$$(M_u = W \cap E'_i(F_0))$$

If M_u is nonempty, put it in set M.

(b) Repeat (a) for every input literal in the term.

8: (a) If no M_u computed for each input literal in a term is empty, increase j by 1 otherwise go to Step 9.

(b) If not all terms in $E''_i(\overline{E}''_i)$ are investigated, go to Step 7 otherwise increase i by 1 and set j = 1.

(c) If not all selected E''_i's and $\overline{E}''_i$'s are processed, go to Step 7 otherwise stop. Set M now contains the terms which detect all single faults in the logic network.

9: Let there be s input literals whose deletion generated nonempty M_u's and t input literals whose deletion generated empty M_u's. Put the corresponding g-literals in set S and set T, respectively.

(a) Delete all the g-literals which are in both S and T from T.

(b) For each common lead label contained in two or more g-literals, replace the corresponding input literals in the $Z''_{i,j}{=}1$ $(Z''_{i,j}{=}0)$ term by 1 then compute M_u by the proper equation of Step 7. Put each nonempty M_u in set M, delete the corresponding g-literals from T.

(c) Increase j by 1, go to Step 8(b).

There is certainly much more information that can be obtained from Algorithm SFD, but since this information is useless in detection test generation, we delay its extraction until the generation of the complete location test set.

<u>Theorem 4.2</u>

A minimal single covering of each M_u in set M generated by Algorithm SFD is a near minimal complete test set that detects all detectable single faults in the logic network.

<u>Proof</u>

A single fault can cause multiple shrinkages in several terms only if this fault is in the common lead which has the same inversion parity in the corresponding terms of E' or $\overline{E}$' expression. If a term is jointly covered by other terms of the same output, the shrinkage of this term alone will not be detected. Furthermore, if a covered term has its g-literals covered by those of uncovered terms, this term need not be investigated for the possible joint multiple shrinkage. Because all the single shrinkages due to its member g-literals can be detected by the terms which detect the shrinkages of the covering terms.

A multiple growth in a term is due to a single fault in the common lead label of several g-literals in the corresponding terms of E' or $\overline{E}$' expression. A growth of a g-literal in a term may be masked while the same growth in other terms may not. Each g-literal which is not masked in any one term need not be considered further for multiple growth, since the multiple growth caused by a single fault in the g-literal will also be detected by the term generated to detect the single growth.

Q.E.D.

<u>Example 4.2</u>

If we use algorithm SFD to generate a near minimal set of terms that detects all single faults in the logic network of Figure 2.4, we will end up with the same set of terms that was generated in Example 4.1 to detect all faults of the network. Hence a complete detection test set for single faults is also a complete test set for multiple faults in this particular network.

In a multiple output network, a fault may be detected in one, two, ... up to all of the network outputs. If a fault is not detected by a complete detection test set we say that fault is undetectable and the network contains redundancy. An undetectable fault is a redundant fault as defined in Definition 3.2.

4.3 GENERATION OF NEAR MINIMAL COMPLETE LOCATION TEST SETS

The generation of a complete test set that locates stuck-at type faults in a combinational logic network will also be accomplished in two steps: (1) the generation of a near minimal set of terms and the fault set detected by the terms. (2) Finding a minimal multiple covering of the terms such that every pair of distinguishable fault sets will be distinguished by the covering.

4.3.1 Generation of a Near Minimal Complete Test Set Which Locates all Faults in a Combinational Logic Network

We shall call a collection of the subsets of single faults a multiple fault set if each fault in the set will cause (1) at least one g-literal to have functional equivalent of 1, or (2) at least one $Z''_{i,j}\neg 1$ or $Z''_{i,j}\neg 0$ term to have a functional equivalent of 0.

A multiple fault of a multiple fault set is a non-conflicting combination of at least one single fault from one of the single fault subsets. For convenience, a multiple fault $F = [a_1/0, a_2/0, a_3/1, a_4/0, a_5/1]$ will be written as $[(a_1, a_2, a_4)/0, (a_3, a_5)/1]$.

Algorithm MFL (Multiple Fault Location)

Data required for this algorithm are the expressions E', $\overline{E}'$, E'', $\overline{E}''$, $E'(F_0)$ and $\overline{E}'(F_0)$.

1: Select either E'_i or $\overline{E}'_i$ which contains the smaller number of g-literals for each of the m primary outputs of the network. If both have same number of g-literals, choose one arbitrarily.

2: For all $Z''_{i,k}=1$ $(Z''_{i,k}=0)$ terms which contain no conflicting pairs: Let i = 1.

(a) For all possible combinations of the terms of E''_i $(\bar{E}''_i)$, compute

$$M_u = \sum_{k \text{ selected}} (Z''_{i,k}=1) \prod_{j \text{ not selected}} \overline{(Z''_{i,j}=1)}$$

$$(M_u = \sum_{k \text{ selected}} (Z''_{i,k}=0) \prod_{j \text{ not selected}} \overline{(Z''_{i,j}=0)})$$

(b) Put all nonempty M_u's and the associated multiple fault sets in set M. The multiple fault set at this stage is the collection of the subsets of single faults in the corresponding terms. These subsets contain the single faults whose effects on the shrinkages of their terms are not offset by the appearances of the faulty leads in different g-literals with different inversion parities.

(c) Repeat (a) through (b) for all selected E''_i $(\bar{E}''_i)$.

3: For all terms in all selected E''_i's and $\bar{E}''_i$'s. Let W denote a degenerate $Z''_{i,k}=1$ $(Z''_{i,k}=0)$ term.

(a) Set i = 1 and k = 1.

(b) For all possible combinations of the growths of input literals in a term, compute

$$M_u = W \cap \bar{E}'_i(F_0) \qquad \text{for growths in } E''_i$$

or

$$M_j = W \cap E_i'(F_0) \qquad \text{for growths in } \bar{E}_i''$$

(c) Put all nonempty M_u's and the associated multiple fault sets in set M. The single fault subsets of the multiple fault set now contains all single faults in the corresponding g-literals whose effect on the growth of the term is not masked by the fault free function.

(d) Repeat (b) through (c) for all terms of the same output.

(e) Repeat (b) through (d) for all selected E_i'' $(\bar{E}_i'')$.

4: Group M_u's which have the same multiple fault set and multiple fault sets which have the same M_u. Stop.

Theorem 4.3

A minimal multiple covering of M_u's generated by the Algorithm MFL which distinguishes each pair of multiple fault sets is a near minimal complete fault location test set of the logic network.

Proof

Algorithm MFL generates a set of terms that detects any detectable combinations of shrinkages and growths of the covering of the network function.

A single covering of each M_u of M is a complete detection test set follows directly from Theorem 4.1. A multiple covering which distinguishes every pair of M_u's will distinguish all distinguishable multiple faults. Because any multiple fault of the logic network is a combination of faults in the generated multiple fault sets, if any two combinations are distinguishable the multiple fault will be distinguished by the covering which distinguishes between the elements of the combination.

The non-minimal property of a minimal covering follows the same arguments in the proof of Theorem 4.1. Step 4 ensures there are no overlaps in the M_u's and in the multiple fault sets.

Q.E.D.

When a single fault is investigated to determine whether its effect on the shrinkage of a term is offset by the possible growths in other terms, each vertex in the shrinking term has to be considered separately because the offset may be partial. The M_u and the multiple fault set then can be adjusted accordingly.

Example 4.3

Find a near minimal complete test set that locates all multiple faults in the logic network of Figure 2.3.

First, we use Algorithm MFL to generate a near minimal set of terms which will distinguish every distinguishable pair of the multiple faults.

Step 1: Refer to Example 3.1, E' is selected to be processed, where

$$\begin{aligned} E' = &<X_1,1,1a,7,9,Z>\cdot<X_2,2,4,4a,7,9,Z> \\ &\cdot<X_3,3,4,4a,7,9,Z> \\ &+ <\bar{X}_1,\bar{1},\overline{1b},5,8,9,Z>\cdot<\bar{X}_2,\bar{2},\bar{4},\overline{4b},6,8,9,Z> \\ &+ <\bar{X}_1,\bar{1},\overline{1b},5,8,9,Z>\cdot<\bar{X}_3,\bar{3},\bar{4},\overline{4b},6,8,9,Z> \end{aligned}$$

Step 2: The result of this step is:

Terms investigated	M_u	Multiple fault set
$X_1X_2X_3$	$M_1 = X_1X_2X_3$	$F_1 = [(1,1a,2,3,4\ 4a,7,9)/0]$
$\bar{X}_1\bar{X}_2$	$M_2 = \bar{X}_1\bar{X}_2X_3$	$F_2 = [(1,1b,2,4b)/1, (5,6,8,9)/0]$
$\bar{X}_1\bar{X}_3$	$M_3 = \bar{X}_1X_2\bar{X}_3$	$F_3 = [(1,1b,3,4,4b)/1,(5,6,8,9)/0]$
$X_1X_2X_3$, $\bar{X}_1\bar{X}_2$	$M_4 = M_1 + M_2$	$F_4 = [F_1, F_2]$
$X_1X_2X_3$, $\bar{X}_1\bar{X}_3$	$M_5 = M_1 + M_3$	$F_5 = [F_1, F_3]$
$\bar{X}_1\bar{X}_2$, $\bar{X}_1\bar{X}_3$	$M_6 = \bar{X}_1\bar{X}_2 + \bar{X}_1\bar{X}_3$	$F_6 = [F_2, F_3]$
$X_1X_2X_3$, $\bar{X}_1\bar{X}_2$, $\bar{X}_1\bar{X}_3$	$M_7 = E'(F_0)$	$F_7 = [F_1, F_2, F_3]$

Step 3: We need $\overline{E}'(F_0) = X_1\overline{X}_2 + X_1\overline{X}_3 + \overline{X}_1X_2X_3$ for computing the M_u's. The result is:

Term	Inputs replaced	M_u	Multiple fault set
$X_1X_2X_3$	X_1	$M_8 = \overline{X}_1X_2X_3$	$F_8 = [(1,1a,7,9)/1]$
	X_2	$M_9 = X_1\overline{X}_2X_3$	$F_9 = [(2,4,4a,7,9)/1]$
	X_3	$M_{10} = X_1X_2\overline{X}_3$	$F_{10} = [(3,4,4a,7,9)/1]$
	X_1, X_2	$M_{11} = M_8 + M_9$	$F_{11} = [F_8, F_9]$
	X_1, X_3	$M_{12} = M_8 + M_{10}$	$F_{12} = [F_8, F_{10}]$
	X_2, X_3	$M_{13} = X_1\overline{X}_2 + X_1\overline{X}_3$	$F_{13} = [F_9, F_{10}]$
	All	$M_{14} = \overline{E}'(F_0)$	$F_{14} = [F_8, F_9, F_{10}]$
$\overline{X}_1\overline{X}_2$	$\overline{X}_1$	$M_{15} = X_1\overline{X}_2$	$F_{15} = [(1,1b)/0, (5,8,9)/1]$
	$\overline{X}_2$	$M_{16} = \overline{X}_1X_2X_3$	$F_{16} = [(2,4,4b)/0, (6,8,9)/1]$
	All	$M_{17} = \overline{E}'(F_0)$	$F_{17} = [F_{15}, F_{16}]$
$\overline{X}_1\overline{X}_3$	$\overline{X}_1$	$M_{18} = X_1\overline{X}_3$	$F_{18} = F_{15}$
	$\overline{X}_3$	$M_{19} = M_{16}$	$F_{19} = [(3,4,4b)/0, (6,8,9)/1]$
	All	$M_{20} = \overline{E}'(F_0)$	$F_{20} = [F_{18}, F_{19}]$

Step 4: Since $F_{18} = F_{15}$ we can delete M_{18} and substitute M_{15} by $X_1\overline{X}_2 + X_1\overline{X}_3$, which in turn equals M_{13}. So M_{15} can be deleted and F_{13} replaced by $[F_9, F_{10}, F_{15}]$. We can also remove M_{17} and M_{20} from set M, then replace F_{14} by $[F_8, F_9, F_{10}, F_{15}, F_{16}, F_{18}, F_{19}]$.

Finally, since M_8 and M_{16} are equal, we can delete M_{16} and replace F_8 by [(1,1a,6,7,8,9)/1, (2,4,4b)/0].

Now,a minimal multiple covering of remaining M_u's which distinguishes each pair of remaining multiple fault sets is found to be the all possible combinations of input literals, i.e. (000, 001, 010, 011, 100, 101, 110, 111) which is a near minimal complete location test set.

4.3.2 Generation of a Near Minimal Test Set which Locates All Single Faults in a Combinational Logic Network

The algorithm to be discussed is a direct extension of Algorithm SFD and bears some similarities with Algorithm MFL of preceding subsection.

Algorithm SFL (Single Fault Location)

Data required for this algorithm are the same as Algorithm MFL.

1: Select either E'_i or $\bar{E}'_i$ which contains the smaller number of g-literals for each of the m primary outputs of the network. If both have same number of g-literals, choose one arbitrarily.

2: For all $Z''_{i,\bar{k}}=1$ ($Z''_{i,\bar{k}}=0$) terms which contain no conflicting pairs: Let i = 1.

(a) Compute

$$M_u = (Z''_{i,k}=1) \prod_{\text{all } j\neq k} \overline{(Z''_{i,j}=1)}$$

$$(M_u = (Z''_{i,k}=0) \prod_{\text{all } j\neq k} \overline{(Z''_{i,j}=0)})$$

for each term in E''_i ($\bar{E}''_i$).

(b) Put all nonempty M_u's and the associated set of single faults in set M. The set of single faults contains all single faults in the term that cause such a term to have the functional equivalent of 0 without considering the possibly masking growths caused by the same single faults.

3: For the terms of Step 2 which contain common lead labels.

(a) Compute

$$M_u = \sum_{k\varepsilon K} (Z''_{i,k}=1) \prod_{j\not\varepsilon K} \overline{(Z''_{i,j}=1)}$$

$$(M_u = \sum_{k\varepsilon K} (Z''_{i,k}=0) \prod_{j\not\varepsilon K} \overline{(Z''_{i,j}=0)})$$

where K contains the numbers of the terms which have common lead labels under investigation.

(b) Put all nonempty M_u's and associated single faults in set M.

4: Repeat Step 2(b) through Step 3 for each E''_i ($\bar{E}''_i$) selected.

5: For all terms selected:

Let W denote a degenerate $Z''_{i,\bar{k}}1$ $(Z''_{i,\bar{k}}0)$ term.

(a) Set i = 1 and j = 1.

(b) Compute

$$M_u = W \cap \bar{E}_i(F_0) \qquad \text{for growths in } E''_i$$

or

$$M_u = W \cap E'_i(F_0) \qquad \text{for growths in } \bar{E}''_i$$

for each single and possible multiple growth in a term.

(c) Repeat (b) for all terms in E''_i $(\bar{E}''_i)$.

(d) Repeat (b) through (c) for all E''_i's and $\bar{E}''_i$'s selected.

6: Group M_u's which have the same fault set and group fault sets which have the same M_u.

Theorem 4.4

A minimal multiple covering of M_u's generated by Algorithm SFL which distinguishes each pair of single fault sets is a near minimal complete test set which locates all single faults of the logic network.

Proof

The statements for the proof parallel those of Theorem 4.2 and Theorem 4.3.

Q.E.D.

Example 4.4

Find a near minimal complete test set which locates all single faults in the logic network of Figure 2.3.

Refer to Example 4.3 and use Algorithm SFL to generate the required M_u's and the single fault sets.

Step 1: Select E' to be processed.

Step 2: The result of this step is the same as the first three M_u's and F_u's of Example 4.3.

Step 3: After this step, we obtain the M_6 and M_7 of the preceding example, but the single fault sets are F_6 = [(5,6,8,9)/0] and F_7 = [9/0].

Step 4: The network has only one output, so this step is skipped.

Step 5: We get M_8, M_9, M_{10}, M_{13} and M_{14} of Example 4.3 for the $X_1X_2X_3$ term except that F_{13} = [(4,4a,7,9)/1] and F_{14} = [(7,9)/1]. For the term of $\overline{X}_1\overline{X}_2$, there are M_{15}, M_{16} and M_{17} = [(8,9)/1]. From $\overline{X}_1\overline{X}_3$ we have M_{18}, M_{19} and M_{20} = [(8,9)/1].

Step 6: Delete M_{15} and M_{18} as before except the new F_{13} is [(4,4a,5,7,8,9)/1, (1,1b)/0]. For the removal of M_{17} and M_{20}, the new F_{14} = [(7,8,9)/1] is introduced. Finally, delete M_{16} and make F_8 equal to [(1,1a,6,7,8,9)/1, (2,4,4b)/0].

We do have a smaller set of M_u's for single faults than that for multiple faults. However, the near minimal complete location test set found is no smaller for this particular logic network.

4.4 SEARCH OF MINIMAL COMPLETE DETECTION TEST SETS

In this Section we shall search for a minimal set of terms such that a minimal single covering of these terms is a minimal complete test set which detects all faults in the combinational logic network.

The logic networks will be assumed to be irredundant. The generalization of the procedure to be presented in this Section to general networks parallels the algorithms discussed in Section 2 and will not be repeated in this Section.

Lemma 4.4.1

Any multiple fault in a combinational logic network can be represented by a combination of faults in non-fanout primary input leads and the fanout branches of the network.

Proof

Any multiple fault that includes the output lead of the elementary gate is equivalent to the single fault in the output lead only. But a stuck output of an elementary gate is equivalent to a stuck input leads or lead of the gate. For a stuck fanout stem lead the fault is equivalent to stuck branches.

Tracing back from the primary output leads of the network and repeating the above arguments, we have the lemma.

Q.E.D.

We shall call the non-fanout primary input leads and the fanout branches the <u>checkpoints</u> of the network.

<u>Procedure MTG (Minimal Test Generation)</u>

(1) Find a minimal number of $Z'_{i,\overline{k}}1$ $(Z'_{i,\overline{k}}0)$ terms that covers all the checkpoints of the network.

(2) (a) Compute

$$M_u = (Z''_{i,\overline{k}}1) \prod_{\text{all } j \neq k} \overline{(Z''_{i,\overline{j}}1)} \quad \text{for terms from } E''_i$$

$$M_u = (Z''_{i,\overline{k}}0) \prod_{\text{all } j \neq k} (Z''_{i,\overline{j}}0) \quad \text{for terms from } \overline{E}''_i$$

(b) If M_u is empty, go to Step 1.

(c) If M_u is nonempty, but at least one checkpoint is a selected term is masked by a growth and the deletion of this checkpoint destroys the property of coverage obtained in Step 1, go to Step 1.

(3) (a) Find a minimal number of g-literals in E'_i $(\overline{E}'_i)$ which covers the complement of lead labels not covered by the terms selected in Step 1.

(b) Compute

$M_u = W \cap \overline{E}_i'(F_0)$

$(M_u = W \cap E_i'(F_0))$

for each g-literal selected in (a), where W is the disjunct of the degenerate $Z_{i,\overline{k}}'' 1$ $(Z_{i,\overline{k}}'' 0)$ terms of which the input literal of the g-literal in question is replaced by 1.

(4) (a) Find a minimal number of terms in $\overline{E}_i'$ (E_i') which covers all the checkpoint labels that are not covered by the terms selected in Step 1.

(b) Compute M_u by the proper equation of Step 2(a).

(c) If M_u is empty, go to (a).

(d) If M_u is nonempty, but at least one checkpoint label in a term is covered by a growth and the deletion of this checkpoint destroys the property of coverage obtained in (a), go to (a).

(5) Repeat Step 1 through Step 4, start Step 1 for the terms in $\overline{E}'$.

(6) For both E' and $\overline{E}'$:

(a) Find a minimal single covering for M_u's generated in Step 2 and Step 3.

(b) Find a minimal single covering for M_u's generated in Step 2 and Step 4,

(c) Select one which contains a minimal number of input vectors from the four coverings found in (a) and (b). Stop.

Theorem 4.5

The set of input vectors selected in Step 6(c) of Procedure MTG is a minimal complete detection test set for an irredundant network.

Proof

The purpose of Procedure MTG is to detect the stuck-at type faults at all checkpoints of the network.

Since the network is irredundant, every shrinkage caused by a g-literal will be detected. In particular, the shrinkages of the g-literals in the selected terms of Step 1 will be detected, hence Step 1 can be satisfied.

For a checkpoint, both stuck-at 1 and stuck-at 0 faults can cause a shrinkage in the network function if the checkpoint has different inversion parities in the different terms selected in Step 1. It is necessary and sufficient to cover the checkpoint labels which are not covered by Step 1. Step 3 and Step 4 serve this purpose.

Step 5 searches the complement of the network function. After four possible minimalities are computed, we get a minimal complete detection test set for the network by selection.

Q.E.D.

Example 4.5

Find a minimal test set which detects all faults of the logic network of Figure 4.1.

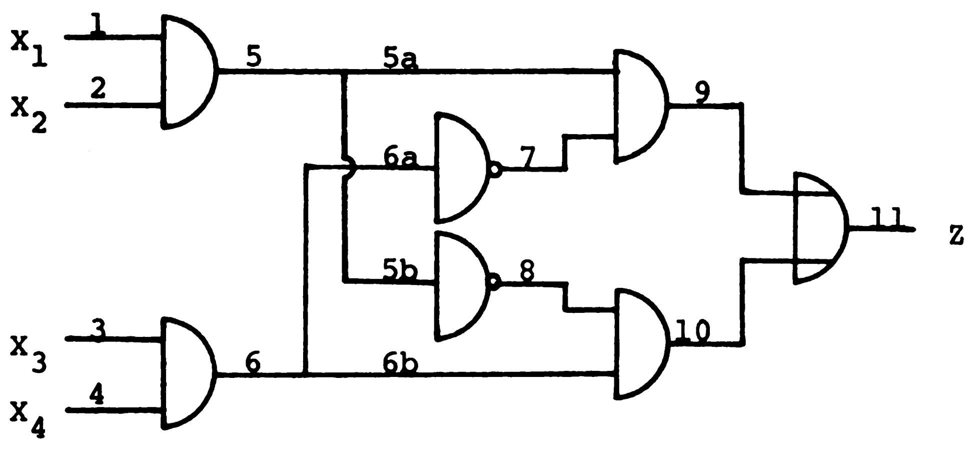


Figure 4.1 The Irredundant Network for Example 4.5.

The E' and $\bar{E}'$ expressions of the network can be found by using Algorithm G of Chapter 2. They are:

$$
\begin{aligned}
E' = {} & \langle X_1,\underline{1},5,\underline{5a},9,11,Z\rangle\cdot\langle X_2,\underline{2},5,\underline{5a},9,11,Z\rangle \\
& \cdot\langle \bar{X}_3,3,6,\underline{6a},7,9,11,Z\rangle \\
& + \langle X_1,\underline{1},5,\underline{5a},9,11,Z\rangle\cdot\langle X_2,\underline{2},5,\underline{5a},9,11,Z\rangle \\
& \cdot\langle \bar{X}_4,\underline{4},6,\underline{6a},7,9,11,Z\rangle \\
& + \langle \bar{X}_1,\underline{\bar{1}},\bar{5},\underline{\overline{5b}},8,10,11,Z\rangle\cdot\langle X_3,\underline{3},6,\underline{6b},10,11,Z\rangle \\
& \cdot\langle X_4,\underline{4},6,\underline{6b},10,11,Z\rangle \\
& + \langle \bar{X}_2,\underline{\bar{2}},\bar{5},\underline{\overline{5b}},8,10,11,Z\rangle\cdot\langle X_3,\underline{3},6,\underline{6b},10,11,Z\rangle \\
& \cdot\langle X_4,\underline{4},6,\underline{6b},10,11,Z\rangle
\end{aligned}
$$

and

$$
\begin{aligned}
\bar{E}' = {} & \langle \bar{X}_1,\underline{\bar{1}},\bar{5},\underline{\overline{5a}},\bar{9},\overline{11},\bar{Z}\rangle\cdot\langle \bar{X}_3,\underline{\bar{3}},\bar{6},\underline{\overline{6b}},\overline{10},\overline{11},\bar{Z}\rangle \\
& + \langle \bar{X}_1,\underline{\bar{1}},\bar{5},\underline{\overline{5a}},\bar{9},\overline{11},\bar{Z}\rangle\cdot\langle \bar{X}_4,\underline{\bar{4}},\bar{6},\underline{\overline{6b}},\overline{10},\overline{11},\bar{Z}\rangle \\
& + \langle \bar{X}_2,\underline{\bar{2}},\bar{5},\underline{\overline{5a}},\bar{9},\overline{11},\bar{Z}\rangle\cdot\langle \bar{X}_3,\underline{\bar{3}},\bar{6},\underline{\overline{6b}},\overline{10},\overline{11},\bar{Z}\rangle
\end{aligned}
$$

$$+ <\overline{X}_2,\underline{\overline{2}},\overline{5},\underline{\overline{5a}},\overline{9},\overline{11},\overline{Z}>\cdot<\overline{X}_4,\underline{\overline{4}},\overline{6},\underline{\overline{6b}},\overline{10},\overline{11},\overline{Z}>$$

$$+ <X_1,\underline{1},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>\cdot<X_2,\underline{2},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>$$
$$\cdot<X_3,\underline{3},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>\cdot<X_4,\underline{4},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>$$

$$+ <X_1,\underline{1},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>\cdot<\overline{X}_1,\underline{\overline{1}},\overline{5},\underline{\overline{5a}},\overline{9},\overline{11},\overline{Z}>$$
$$\cdot<X_2,\underline{2},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>$$

$$+ <X_1,\underline{1},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>\cdot<X_2,\underline{2},5,\underline{5b},\overline{8},\overline{10},\overline{11},\overline{Z}>$$
$$\cdot<\overline{X}_2,\underline{\overline{2}},\overline{5},\underline{\overline{5a}},\overline{9},\overline{11},\overline{Z}>$$

$$+ <X_3,\underline{3},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>\cdot<\overline{X}_3,\underline{\overline{3}},\overline{6},\underline{\overline{6b}},\overline{10},\overline{11},\overline{Z}>$$
$$\cdot<X_4,\underline{4},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>$$

$$+ <X_3,\underline{3},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>\cdot<X_4,\underline{4},6,\underline{6a},\overline{7},\overline{9},\overline{11},\overline{Z}>$$
$$\cdot<\overline{X}_4,\underline{\overline{4}},\overline{6},\underline{\overline{6b}},\overline{10},\overline{11},\overline{Z}>$$

where the checkpoints are underlined.

Step 1: Select the first and the third terms from E'. The checkpoint labels covered by this selection are: $1,\overline{1},2,3,\overline{3},4,5a,\overline{5b},\overline{6a},6b$.

Step 2: The M_u's obtained are: $M_1 = X_1X_2\overline{X}_3X_4$ and $M_2 = \overline{X}_1X_2X_3X_4$. Both terms are nonempty and no checkpoint is covered by any corresponding growth.

Step 3: One of the minimal number of g-literals which covers the checkpoint labels $2,4,5a,\overline{5b},\overline{6a},6b$ contains the second and the third g-literals of the first term and the first and the third g-literals of the third term. The M_u's computed are:

$$M_3 = X_1\overline{X}_2\overline{X}_3 + X_1\overline{X}_2\overline{X}_4;\ M_4 = X_1X_2X_3X_4;$$

$$M_5 = \overline{X}_1X_3\overline{X}_4 + \overline{X}_2X_3\overline{X}_4$$

Step 4: One of the minimal number of terms in $\overline{E}'$ that covers the checkpoint labels $\overline{2},\overline{4},\overline{5a},5b,6a,\overline{6b}$ contains the fourth and the fifth terms of $\overline{E}'$. The M_u's that detect the shrinkages of these terms are: $M_6 = X_1\overline{X}_2X_3\overline{X}_4$ and $M_7 = X_1X_2X_3X_4$. There is no checkpoint covered by the corresponding growth.

Step 5: Start from Step 1 by picking the first and the fifth terms of $\overline{E}'$. These two terms cover the checkpoint labels $1,\overline{1},2,3,\overline{3},4,\overline{5a},5b,6a,\overline{6b}$. The M_u's are $M_8 = \overline{X}_1X_2\overline{X}_3X_4$ and $M_9 = X_1X_2X_3X_4$. Then we choose the first and the second g-literals of the first term and the second and the fourth g-literals of the fifth term. The result is: $M_{10} = X_1X_2\overline{X}_3 + X_1X_2\overline{X}_4$, $M_{11} = \overline{X}_1X_3X_4 + \overline{X}_2X_3X_4$, $M_{12} = X_1\overline{X}_2X_3X_4$ and $M_{13} = X_1X_2X_3\overline{X}_4$. Finally, use the second and the fourth terms of E' to get $M_{14} = X_1X_2X_3\overline{X}_4$ and $M_{15} = X_1\overline{X}_2X_3X_4$.

Step 6: The result of this step is:

The M_u's to be covered	A minimal covering
$(M_1, M_2, M_3, M_4, M_5)$	(0010, 0111, 1001, 1101, 1111)
(M_1, M_2, M_6, M_7)	(0111, 1010, 1101, 1111)
$(M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13})$	(0101, 1011, 1110, 1111)
$(M_8, M_9, M_{14}, M_{15})$	(0101, 1011, 1110, 1111)

We can choose either the second or the third set of input vectors as a minimal complete detection test set for the network. Note that the fourth test set is the same as the third set.

4.5 CHAPTER SUMMARY AND REMARKS

The power of the CGEMs, with its various modified forms, in generating the complete test sets was demonstrated in this Chapter.

Two algorithms for generating near minimal complete test sets and two algorithms for generating near minimal complete location test sets were presented. As the examples illustrated, the generation of complete test sets under the single fault assumption is sometimes no simpler than that with the multiple fault assumption.

A method of searching for a minimal complete detection test set for an irredundant network was also suggested. Applying this method to a network, i.e. The network of Figure 4.1 we found a smaller minimal complete detection set than that obtained by Poage's method. This is due to the fact that we used a "better" model than that of Poage's.

In reading the searching method, i.e. Procedure MTG, one may note that all the algorithms discussed were not intended for the generation of minimal complete test sets. Rather, we tried to simplify the computations as much as possible and at the same time kept the test sets at a reasonable small size. Therefore, these algorithms can easily be modified, with the aid of Procedure MTG, to conform to one's particular situation.

When applied to a redundant network, the algorithms presume no redundant faults exist. For a redundant network with the presence of redundant faults, we can: (1) Substitute the g-literals which are affected by each of the possible combinations of the redundant faults by appropriate functional equivalents. (2) Find a complete test set T_i for each of the modified E''s or $\overline{E}'$'s. The disjunction of T_i's is a complete test set.

CHAPTER 5

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

5.1 CONCLUSIONS

This thesis presents an algebraic model which describes accurately and completely the physical and logical structures of a combinational logic network comprised of elementary gates. The model also provides a large degree of flexibility for expressing the interdependencies of the physical and logical structures of an object network.

After the model is constructed by an algorithm, it is used for (1) analyzing stuck-at type faults and bridging faults and (2) generating complete test sets. The fault functions realized by a network under the influence of the stuck-at type faults or the bridging faults were rigorously investigated. The generation of complete test sets was studied by four algorithms and one procedure. The algorithms and the procedure are straightforward and can be modified to satisfy one's particular situation. The memory requirements are minimized by taking advantage of the flexibility of the model.

5.2 SUGGESTIONS FOR FUTURE WORK

Many conjectures from the articles of fault diagnosis and test generation can be confirmed or disproved using the model discussed in this thesis. For example:

> Armstrong's [1] conjecture: There exist at least one set of literals [L] and an associated set of tests [T] that tests an appearance of every literal in [L] for s-a-1 and s-a-0 and also detects every fault in the net.

This is true if we consider the set of literals of Armstrong is the set of all distinct g-literals of the network. We may have an even smaller set of g-literals by Theorem 4.5.

> Bossen and Hong's [3] conjecture: Any strategy based on checkpoint verification seems infeasible for redundant networks.

This can easily be disproved by viewing the detection of a shrinkage and a growth in Algorithm MFD is the verification of checkpoints in a g-literal of the network.

There are some interesting problems which remain to be solved.

The problem of detecting and locating stuck-at type faults, both permanent and intermittent, in sequential logic networks has not been satisfactorily solved. This is due to both the large number of state transitions and the length of test sequence which brings the network to a required state. I conjecture that by cascading the combinational equivalent of a sequential network and generating test set for first, first two, first three, ... time frame sequentially, removing the checkpoints which are investigated and adding checkpoints from the new time frame the computational burden may remain manageable and the problem can be more satisfactorily solved.

The design of easily testable networks without adding logic circuits for testing purpose is another interesting problem. From Chapter 4 we know if the shrinkage of each term and the growth of each g-literal are detectable, then the algorithms can be terminated earlier and the test set will be smaller. A more general network structure other than two-level and tree realizations remains to be discovered. A retrospective study of this thesis may shed a little light on this problem.

Even if all functional units can be modeled in terms of elementary gates and the checkpoints can be used adequately in the model to describe a MSI or a LSI logic network, one may still want to develop a model in terms of other small functional units for a LSI network. The possible consequences are (1) The model cannot be used to detect redundancy in the network, (2) the preciseness of the parity of each label in an information path will be lost, hence the computations for test generation will be much more complicated.

APPENDIX

APPENDIX

A DATA STRUCTURE FOR ALGORITHM G OF SECTION 2.3

For the machine implementation of Algorithm G a doubly linked list structure can be used to represent the equivalent Boolean expression of a logic network. Each node of the list represents a distinct lead label together with the addresses of its predecessors and successors.

The size of a node depends on the properties of the logic network. In terms of number of bits, it is the sum of the following four items:

1. For the lead label

   $\lceil \log_2$ (The largest number of the lead labels)$\rceil$ + $\lceil \log_2[$(Maximum numb er of fanout branches with the same labeling integer) + 1$]\rceil$ bits.

   where $\lceil x \rceil$ is the smallest integer $\geq x$.

2. For indicators

   a. The label is barred or non-barred: 1 bit.

   b. The lead is a network output, a primary input or an internal lead: 2 bits.

c. The lead is the output of an OR gate or an AND gate: 1 bit.

3. For the addresses of predecessors

[(Maximum number of input leads for an elementary gate) + 1(for endmarker)]

X [$\lceil \log_2$ (Total number of leads of the network)$\rceil$
+ 1(indicating whether the label is barred)
+ 1(indicating whether it is a predecessor or a successor)] bits.

4. For the addresses of successors

[(Maximum number of fanout branches from a fanout stem) + 1(for endmarker)]

X [$\lceil \log_2$ (Total number of leads of the network)$\rceil$
+ 1(indicating whether the label is barred)
+ 1(indicating whether it is a predecessor or successor)] bits.

For a network of 2,000 gates with 40 primary inputs, let the maximum number of fanout branches with the same labeling integer be 6, the maximum number of fanout branches from a fanout stem be 3 and the total number of distinct leads of the network be 5,000. It requires 168 bits to represent a node. Counting both barred and non-barred labels, a memory size of 30K 60-bit words is needed.

In order to illustrate the data structure discussed, Example 2.3 is recomputed as follows:

For the network of Figure 2.4 the node of List requires 53 bits which is divided into 11 fields:

Field	Length (in bits)	Descriptions
1	5	Represents the integer part of lead label.
2	2	0 for non-fanout, 1 for fanout a and 2 for fanout b.
3	1	0 for non-barred, 1 for barred.
4	2	0 for network output, 2 for primary input and 1 for internal lead.
5	1	0 for OR gate, 1 for AND gate. (primary input lead use 0.)
6-8	7 each	For the addresses of predecessors which will be represented by the labels with or without overbar. E for endmarker and ɓ for not used.
9-11	7 each	For the addresses of successors which will be represented by the labels with or without overbar. E for endmarker and ɓ for not used.

The node describing label p will be denoted by Np where p either barred or non-barred. Two more Lists will be introduced to keep track of path expansion process. List A stores the non-primary input label and its address from the end and its head element is to be removed for expansion. After expansion the label and the primary input label introduced together with the corresponding addresses are stored in List B.

The starting state after the network is labeled by Procedure L is

N12: 12,0,0,0,0,9,11a,E,E,b̸,b̸
N13: 13,0,0,0,1,11b,8,E,E,b̸,b̸
A(0): 12,13
B(0): (empty)

The first step is to remove 12 from List A then generate N9 and N11. Since 9 and 11 are not primary input lead labels, they are put at the tail of List A. 12 is put at List B. Note the integer in the parentheses after A or B denotes the number of expansions completed.

N9: 9,0,0,1,1,$\overline{6}$,$\overline{7a}$,E,12,E,b̸
N11a: 11,1,0,1,0,11,E,b̸,12,E,b̸
A(1): 13,9,11a
B(1): 12

The further results are:

N11b: 11,2,0,1,0,11,E,b̸,13,E,b̸
N8: 8,0,0,1,0,1b,2b,E,13,E,b̸
A(2): 9,11a,11b,8
B(2): 12,13

N$\overline{6}$: 6,0,1,1,0,$\overline{1a}$,$\overline{2a}$,E,9,E,b̸
N$\overline{7a}$: 7,1,1,1,0,$\overline{7}$,E,b̸,9,E,b̸

A(3): 11a,11b,8,$\overline{6}$,$\overline{7a}$,

B(3): 12,13,9

N11: 11,0,0,1,0,10,5,E,11a,E,$\cancel{b}$

A(4): 11b,8,$\overline{6}$,$\overline{7a}$,11

B(4): 12,13,9,11a

For the fifth step 11b is extended to 11 which is in A(4), so we update N11:

N11: 11,0,0,1,0,10,5,E,11a,11b,E

A(5): 8,$\overline{6}$,$\overline{7a}$,11

B(5): 12,13,9,11a,11b

N1b: 1,2,0,1,0,1,E,$\cancel{b}$,8,E,$\cancel{b}$

N2b: 2,2,0,1,0,2,E,$\cancel{b}$,8,E,$\cancel{b}$

A(6): $\overline{6}$,$\overline{7a}$,11,1b,2b

B(6): 12,13,9,11a,11b,8

N$\overline{1a}$: 1,1,1,1,0,$\overline{1}$,E,$\cancel{b}$,$\overline{6}$,E,$\cancel{b}$

N$\overline{2a}$: 2,1,1,1,0,$\overline{2}$,E,$\cancel{b}$,$\overline{6}$,E,$\cancel{b}$

A(7): $\overline{7a}$,11,1b,2b,$\overline{1a}$,$\overline{2a}$

B(7): 12,13,9,11a,11b,8,$\overline{6}$

N$\overline{7}$: 7,0,1,1,0,3,E,$\cancel{b}$,$\overline{7a}$,E,$\cancel{b}$

A(8): 11,1b,2b,$\overline{1a}$,$\overline{2a}$,$\overline{7}$

B(8): 12,13,9,11a,11b,8,$\overline{6}$,$\overline{7a}$

N10: 10,0,0,1,1,7b,5,E,11,E,$\cancel{b}$

N5: 5,0,0,2,0,E,$\cancel{b}$,$\cancel{b}$,11,E,$\cancel{b}$

A(9): 1b,2b,$\overline{1a}$,$\overline{2a}$,$\overline{7}$,10

B(9): 12,13,9,11a,11b,8,$\overline{6}$,$\overline{7a}$,11,5

Note that lead label 5 is a primary input lead label. It can be extended, so put it into B(9) rather than attached it to A(9).

N1: 1,0,0,2,0,E,b̸,b̸,1b

A(10): 2b,$\overline{1a}$,$\overline{2a}$,7,10

B(10): B(9),1b,1

where the notation B(9),1b,1 is used to represent the insertion of 1b and 1 into List B(9).

N2: 2,0,0,2,0,E,b̸,b̸,2b,E,b̸

A(11): $\overline{1a}$,$\overline{2a}$,$\overline{7}$,10

B(11): B(9),1b,1,2b,2

N$\overline{1}$: 1,0,1,2,0,E,b̸,b̸,$\overline{1a}$,E,b̸

A(12): $\overline{2a}$,$\overline{7}$,10

B(12): B(9),1b,1,2b,2,$\overline{1a}$,$\overline{1}$

N$\overline{2}$: 2,0,1,2,0,E,b̸,b̸,$\overline{2a}$,E,b̸

A(13): $\overline{7}$,10

B(13): B(9),1b,1,2b,2,$\overline{1a}$,$\overline{1}$,$\overline{2a}$,$\overline{2}$

N3: 3,0,0,2,0,E,b̸,b̸,$\overline{7}$,E,b̸

A(14): 10

B(14): B(9),1b,1,2b,2,$\overline{1a}$,$\overline{1}$,$\overline{2a}$,$\overline{2}$,$\overline{7}$,3

N7b: 7,2,0,1,0,7,E,b̸,10,E,b̸

N4: 4,0,0,2,0,E,b̸,b̸,10,E,b̸

A(15): 7b

B(15): B(9),1b,1,2b,2,$\overline{1a}$,$\overline{1}$,$\overline{2a}$,$\overline{2}$,$\overline{7}$,3,10,4

N7: 7,0,0,1,0,$\bar{3}$,E,b̸,7b,E,b̸

A(16): 7

B(16): B(15),7b

N$\bar{3}$: 3,0,1,2,0,E,b̸,b̸,7,E,b̸

A(17): (empty)

B(17): B(15),7b,7,$\bar{3}$

The program will stop at the eighteenth step, since the empty A(17) implies the path sets generated are g-literals.

In the above computations the input to the program is assumed to have all the information needed. The network information can be compactly stored and easily fetched if they are arranged in an appropriate order.

For a large network one may like to sort the nodes in List B in the manner that will save the searching time. The procession of List A is similar to the breadth-first search method in artificial intelligence. The size of List A may be reduced by using the depth-first method but, in general, for a network of various lengths of g-literals the reduction will not be appreciable.

The AND/OR graph representation of the network of Figure 2.4 is depicted on the next page. One may note with a little effort the sum of products form of the g-literals and the functional equivalent of the network can be obtained. Furthermore, with the interpretations of AND and OR, barred and non-barred interchanged it is the complemented form of the network.

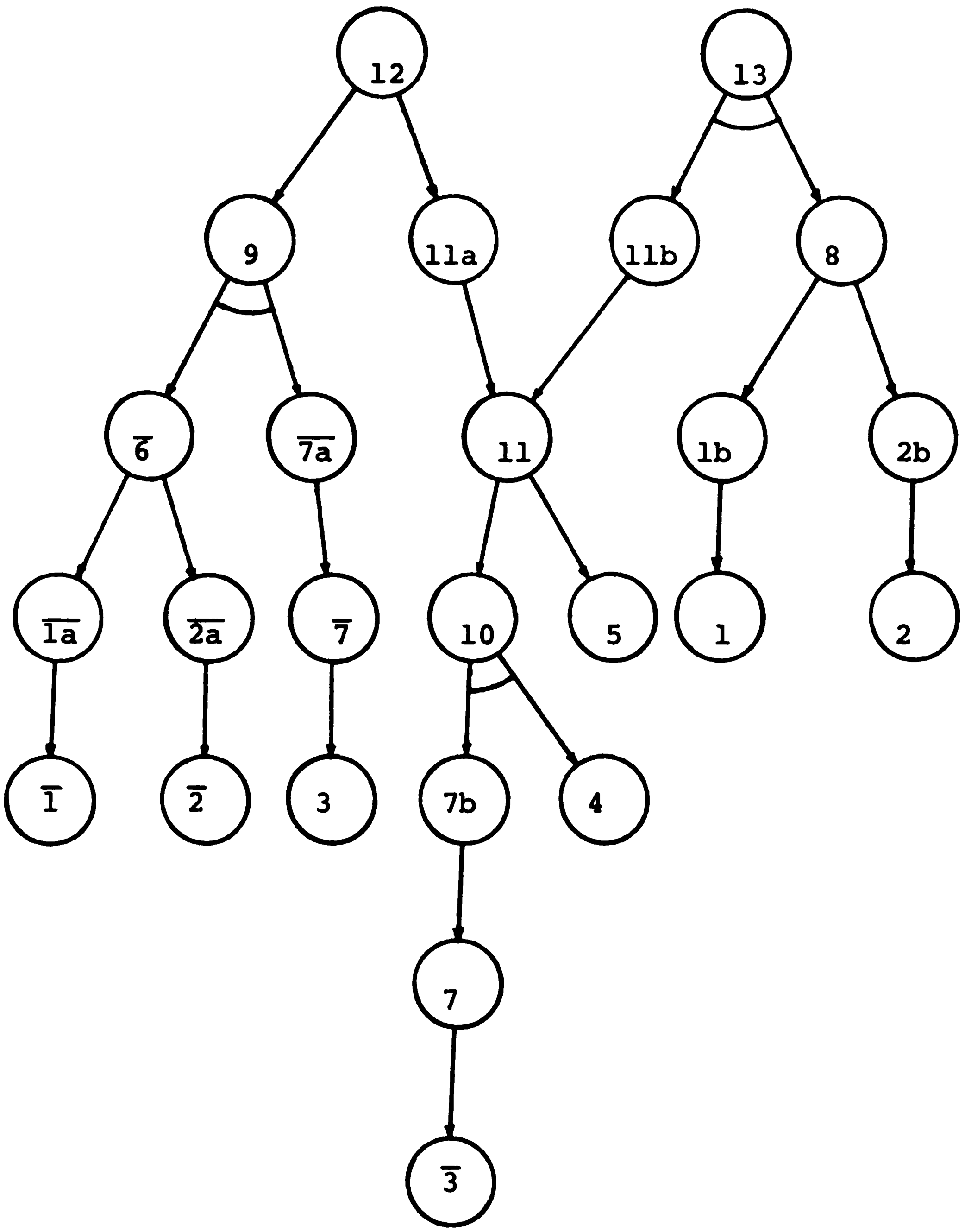


Figure A.1. The AND/OR Graph representation of Figure 2.4.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Armstrong, D. B. "On Finding a Near Minimal Set of Fault Detection Tests for Combinational Logic Nets," IEEE Trans. Electron. Comput., Vol. EC-15, pp. 66-73, Feb. 1966.

[2] Arnold, T. F.; Tan, C. J.; and Newborn, M. M. "Iteratively Realized Sequential Circuits," IEEE Trans. Comput., Vol. C-19, pp. 54-66, Jan. 1970.

[3] Bossen, D. C.; and Hong, S. J. "Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks," IEEE Trans. Comput., Vol. C-20, pp. 1252-1257, Nov. 1971.

[4] Breuer, M. A. "Testing for Intermittent Faults in Digital Circuits," IEEE Trans. Comput., Vol. C-22, pp. 241-246, Mar. 1973.

[5] Ball, M.; and Hardie, F. "Effects and Detection of Intermittent Failures in Digital Systems," in 1969 Fall Joint Comput. Conf., AFIPS Conf. Proc., Vol. 35, pp. 329-355. Montvale, N. J.: AFIPS Press, 1969.

[6] Carter, W. C. "Fault-Tolerant Computing: An Introduction and a Viewpoint," IEEE Trans. Comput., Vol. C-22, pp. 225-229, Mar. 1973.

[7] Chang, H. Y. "An Algorithm for Selecting an Optimum Set of Diagnostic Tests," IEEE Trans. Electron. Comput., Vol. EC-14, pp. 706-711, Oct. 1965.

[8] __________. "A Distinguishability Criterion for Selecting Efficient Diagnostic Tests," in 1968 Spring Joint Comput. Conf., AFIPS Conf. Proc., Vol. 32, pp. 529-534. Washington, D.C.: Thompson, 1968.

[9] Chang, H. Y.; Manning, E.; and Metze, G. Fault Diagnosis of Digital Systems. New York: Wiley-Interscience, 1970.

[10] Clegg, F. W. "Algebraic Properties of Faults in Logic Networks," Ph.D. thesis, Stanford Univ., 1970.

[11] __________. "Use of SPOOF's in the Analysis of Faulty Logic Networks," IEEE Trans. Comput., Vol. C-22, pp. 229-234, Mar. 1973.

[12] Dwyer, T. F. "Comments on 'Fault Testing and Diagnosis in Combinational Digital Circuits,'" IEEE Trans. Comput., (Corresp.), Vol. C-18, p. 760, Aug. 1969.

[13] Farmer, D. E. "Algorithms for Designing Fault-Detection Experiments for Sequential Machines," IEEE Trans. Comput., Vol. C-22, pp. 159-167, Feb. 1973.

[14] __________. "A Strategy for Detecting Faults in Sequential Machines Not Possessing Distinguishing Sequences," in 1970 Fall Joint Comput. Conf., AFIPS Conf. Proc., Vol. 37, pp. 493-501. Montvale, N. J.: AFIPS Press, 1970.

[15] Flomenhoft, M. J.; Si, S. C.; and Susskind, A. K. "Algebraic Techniques for Finding Tests for Several Fault Types," in Digest 1973 International Symp. on Fault-Tolerant Computing, pp. 85-99.

[16] Friedman, A. D. "Fault Detection in Redundant Circuits," IEEE Trans. Electron. Comput., Vol. EC-16, pp. 99-100, Feb. 1967.

[17] __________. "Diagnosis of Short Faults in Digital Circuits," in Digest 1973 International Symp. on Fault-Tolerant Computing, pp. 95-99.

[18] Friedman, A. D.; and Menon, P. R. Fault Detection in Digital Circuits. Englewood Cliffs, N. J.: Prentice-Hall, 1971.

[19] Gault, J. W.; Robinson, J. P.; and Reddy, S. M. "Multiple Fault Detection in Combinational Networks," IEEE Trans. Comput., Vol. C-21, pp. 31-36, Jan. 1972.

[20] Hannigan, J. M.; and Master, C. G., Jr. "Redundant System Test Point Allocation and Mission Reliability Estimation Procedures," IEEE Trans. Comput., Vol. EC-16, pp. 591-596, Oct. 1967.

[21] Hsieh, E. P. "Checking Experiments for Sequential Machines," IEEE Trans. Comput., Vol. C-20, pp. 1152-1166, Oct. 1971.

[22] Kamal, S. "Diagnosis of Intermittent Faults in Digital Systems," Ph.D. thesis, Michigan State Univ., Jan. 1973.

[23] Kautz, W. H. "Fault Testing and Diagnosis in Combinational Digital Circuits," IEEE Trans. Comput., Vol. C-17, pp. 352-366, Apr. 1968.

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

[25] Kohavi, I.; and Kohavi, Z. "Detection of Multiple Faults in Combinational Logic Networks," IEEE Trans. Comput., Vol. C-21, pp. 556-568, Jun. 1972.

[26] Kohavi, Z.; and Spires, D. A. "Designing Sets of Fault-Detection Tests for Combinational Logic Circuits," IEEE Trans. Comput., Vol. C-20, pp. 1463-1469, Dec. 1971.

[27] Kubo, H. "A Procedure for Generating Test Sequences to Detect Sequential Circuit Failures," NEC Res. Develop., Oct. 1968.

[28] Marinos, P. N. "Derivation of Minimal Complete Sets of Test-Input Sequence Using Boolean Differences," IEEE Trans. Comput., Vol. C-20, pp. 25-32, Jan. 1971.

[29] McCluskey, E. J.; and Clegg, F. W. "Fault Equivalence in Combinational Logical Networks," IEEE Trans. Comput., Vol. C-20, pp. 1286-1293, Nov. 1971.

[30] Mealy, G. H. "A Method for Synthesizing Sequential Circuits," Bell System Tech. J., Vol. 34, pp. 1045-1080, 1955.

[31] Mei, K. C. Y. "Bridging and Stuck-at Faults," in Digest 1973 International Symp. on Fault-Tolerant Computing, pp. 91-94.

[32] Moore, E. F. "Gedangen Experiments on Sequential Machines," in Automata Studies, Annals of Math. Studies, No. 34, Shannon, C. E.; and McCarthy, J., eds., pp. 129-153. Princeton, N. J.: Princeton Univ. Press, 1956.

[33] Poage, J. F. "Derivation of Optimal Tests to Detect Faults in Combinational Circuits," in Proc. Symp. on Mathematical Theory of Automata, Polytechnic Inst. of Brooklyn, pp. 483-528, 1963.

[34] Reddy, S. M. "Easily Testable Realizations for Logic Functions," IEEE Trans. Comput., Vol. C-21, pp. 1183-1188, Nov. 1972.

[35] __________. "A Design Procedure for Fault-Locatable Switching Circuits," IEEE Trans. Comput., Vol. C-21, (Short Notes), pp. 1421-1426, Dec. 1972.

[36] Reese, R. D.; and McCluskey, E. J., Jr. "A Gate Equivalent Model for Combinational Logic Network Analysis," in Digest 1973 International Symp. on Fault-Tolerant Computing, pp. 79-84.

[37] Roth, J. P. "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Res. Develop., Vol. 10, pp. 278-291, Jul. 1966.

[38] Roth, J. P.; Bouricius, W. G.; and Schneider, P. R. "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," IEEE Trans. Electron. Comput., Vol. EC-16, pp. 567-580, Oct. 1967.

[39] Russel, J. D.; and Kime, C. R. "Structure Factors in the Fault Diagnosis of Combinational Networks," IEEE Trans. Comput., Vol. C-20, pp. 1276-1285, Nov. 1971.

[40] Schertz, P. R.; and Metze, G. "A New Representation for Faults in Combinational Digital Circuits," IEEE Trans. Comput., Vol. C-21, pp. 858-866, Aug. 1972.

[41] Schneider, P. R. "On the Necessity to Examine D-Chain in Diagnostic Test Generation--An Example," IBM J. Res. Develop., Vol. 11, p. 114, Jan. 1967.

[42] Sellers, F. F.; Hsiao, M. Y.; and Bearnson, L. W. "Analyzing Errors with the Boolean Difference," IEEE Trans. Comput., Vol. C-17, pp. 676-683, Jul. 1967.

[43] Seshu, S.; and Freeman, D. N. "The Diagnosis of Asynchronous Sequential Switching Systems," IRE Trans. Electron. Comput., Vol. EC-11, pp. 459-465, Aug. 1962.

[44] Su, S. Y. H.; Chang, S. J.; and Breuer, M. A. "Location of Multiple Stuck-Type Faults in Combinational Networks," IEEE Comput. Soc. Reposit. R-72-196, 1972.

[45] Su, S. Y. H.; and Cho, Y.-C. "A New Approach to Fault Location of Combinational Circuits," IEEE Trans. Comput., Vol. C-21, pp. 21-30, Jan. 1972.

[46] Williams, M. J. Y.; and Angell, J. B. "Enhancing Testability of LSI Circuits and Additional Logic," IEEE Trans. Comput., Vol. C-22, pp. 46-60, Jan. 1973.