7

Fault Detection and Location in Combinational Circuits

In recent years, the development of integrated-circuit technology has accelerated rapidly; MSI and LSI techniques promise to make today's functional-level devices tomorrow's (even today's) basic components. Accordingly, digital systems are built with more and more complexity; the fault testing and diagnosis of digital circuits becomes an important and indispensible part of the manufacturing process. The problem of fault testing and diagnosis consists of the following two subproblems:

1. The fault-detection problem.
2. The fault-location problem.

The first problem is the determination of whether or not any of a prescribed list of faults, which may include all possible single faults and multiple faults, has occurred. The second problem, in turn, consists of the following subproblems:

2(a) Determination of the location of the fault that has occurred.

2(b) Determination of the location of the fault within a module (package or subnetwork) in which it has occurred.

A number of basic analytic and heuristic methods, including the fault-table method, the path-sensitizing and equivalent-normal-form (ENF) method, the Karnaugh map and tabular method, the ENF–Karnaugh map method, the Boolean difference method, and the SPOOF method are presented. The fault-table method is the most classic approach to the problem. It is completely general and always yields a minimum set of diagnostic tests. However, it suffers from the fact that it requires a very large fault table to be constructed. To overcome the problem of not requiring the construction of a very large fault table, the concept of path sensitizing is introduced. The path-sensitizing method requires no fault table, basically only work for the class of tree-like circuits and some special non-tree-like circuits. A heuristic, systematic procedure derived from the concept of path sensitizing, known as the equivalent-normal-form (ENF) method, is then introduced. Although this method has eliminated the two imperfections that the fault table and path-sensitizing methods have, it introduces an unattractive new feature, the requirement of the cumbersome computation of a "score function" for

every literal in the ENF and the complemented ENF of the circuit. Finally, a ENF–Karnaugh map method is presented which eliminates the above three shortcomings. This method is a combination of the ENF method and the Karnaugh map and tabular method. Both the ENF method and the ENF–Karnaugh map method do not guarantee minimal experiments, nor is there a guarantee that a set of sensitized paths can be found for every circuit. Finally, the Boolean difference method and the SPOOF methos are two convenient general methods for deriving tests for detecting any single and/or multiple faults in any part of the circuit without using any fault tables or maps.

Based on these methods, computer programs can be written to generate complete fault-detection test sets for checking combinational circuits, such as those combinational TTL/MSI and MOS/LSI ROM IC chips. Two such computer programs DALG-II (D-Algorithm Version II) and TEST-DETECT are available.

7.1 Fault Detection and Fault Location: Classical Methods

After a digital circuit is designed and built, it is always desirable to know whether the circuit is constructed without any faults. If it is properly constructed and in use, it may be disabled by almost any internal failure. The process of applying tests and determining whether a digital circuit is fault-free or not is generally known as *fault detection*. If a known relationship exists between the various possible faults and the deviations of output patterns, it is possible to identify the failures and to classify them (at least) within the smallest possible set of components. This process is termed *fault location*. *Fault diagnosis* is generally referred to as the composite of both the above processes.

The diagnostic techniques of digital systems have progressed from the state of testing the operation codes of the machine instruction set in the early days to the current hardware-assisted software aids for speedy determination and identification of faulty elements. In the coming LSI (large-scale-integration) era, the problem of diagnosis is shifted from locating the faulty discrete component to the identification of the replaceable module (chip or flat pack) in which the faulty component is resident.

Testing strategies generally depend on the system design and the architecture. Generally, the diagnostic engineer partitions the system into levels of hierarchy (systems, functional subsystems, modules, chips, etc.) and uses these levels in an organized fashion to isolate faults. Test points and the like provide mutual isolations among the partitions at various levels and help to reduce the size and time of testing. Here we shall only be concerned with the fault detection and location problem at the circuit level.

A fault-detection test set is said to be *minimal* if it is a minimal set that can detect the existence of each of the faults under consideration. A fault-location test set is said to be *minimal* if it is a minimal set that can locate each of the faults under consideration to the smallest identifiable part of the logic circuit. To construct a minimal fault-detection test set or minimal fault-location test set for combinational logic circuits, it is

often necessary to have the complete test set of every fault under consideration; where a test of a fault is defined as the application of an input pattern to the primary input terminals and the observation of the corresponding outputs appearing at the primary output terminals so that the existence of the fault can be detected, the complete test set of a fault is defined as the set that contains all the tests of the fault. When it is not feasible to generate the complete test set of every fault under consideration because of a limitation on the amount of storage space, a near-optimal fault-detection test set or a near-optimal fault-diagnosis test set should be constructed. In this case the more tests that we can find, the more nearly optimal the fault-detection and fault-location test sets that can be constructed. The derivations of optimal fault-detection tests and optimal fault-location tests will be discussed.

Even when we are concerned only with the fault detection and location problem at the circuit level, without making certain realistic assumptions about the types of circuits and faults, the problem is still too broad to be discussed. The following are the assumptions made about the types of circuits and faults under investigation.

1. *Assumptions about the type of digital circuit to be considered*
   (a) It is assumed that the *digital* circuits under study are combinational circuits.
   (b) The circuits are assumed to be composed of exclusively AND, OR, NOT, NAND, and NOR gates. It should be pointed out that almost all the practical digital circuits in use are made up of these five types of gates. Furthermore, the methods derived for obtaining tests for this class of circuits are general enough to be applied to circuits consisting of gates other than these five types, such as the XOR (EXCLUSIVE-OR) gate, with only some slight modifications.
   (c) It is assumed that the response delays of all the gate elements are the same.
2. *Assumptions about the type of fault to be considered*
   (a) The faults considered here are assumed to be fixed or permanent or nontransient faults, by which we mean that without having them fixed or repaired, the fault will be permanently there.
   (b) Most of the faults occurred in currently used circuits, such as resistor–transistor logic circuits (RTL), diode–transistor logic circuits (DTL), and transistor–transistor logic circuits (TTL), are those which cause a wire to be (or appear logically to be) stuck-at-zero (s-a-0) or stuck-at-one (s-a-1). Restricting our consideration to just a class of faults is technically justified, since most circuit failures exhibit symptomatically identical effects, and this class of faults covers the faults in circuits with discrete components as well as integrated circuits. A multiple fault is defined as the simultaneous occurrence of any possible combination of s-a-0 and s-a-1 faults. In this chapter, both single and multiple fault detections will be considered.

Note that there are some faults in a circuit which are undetectable. For example, the s-a-0 fault on line α and s-a-1 fault on line β of the circuit of Fig. 7.1.1(a), denoted by α_0 and β_1, respectively, are undetectable because the output functions z^{α_0} and z^{β_1} of the faulty versions of the circuit are identical to the output function z of the normal circuit. Thus we define

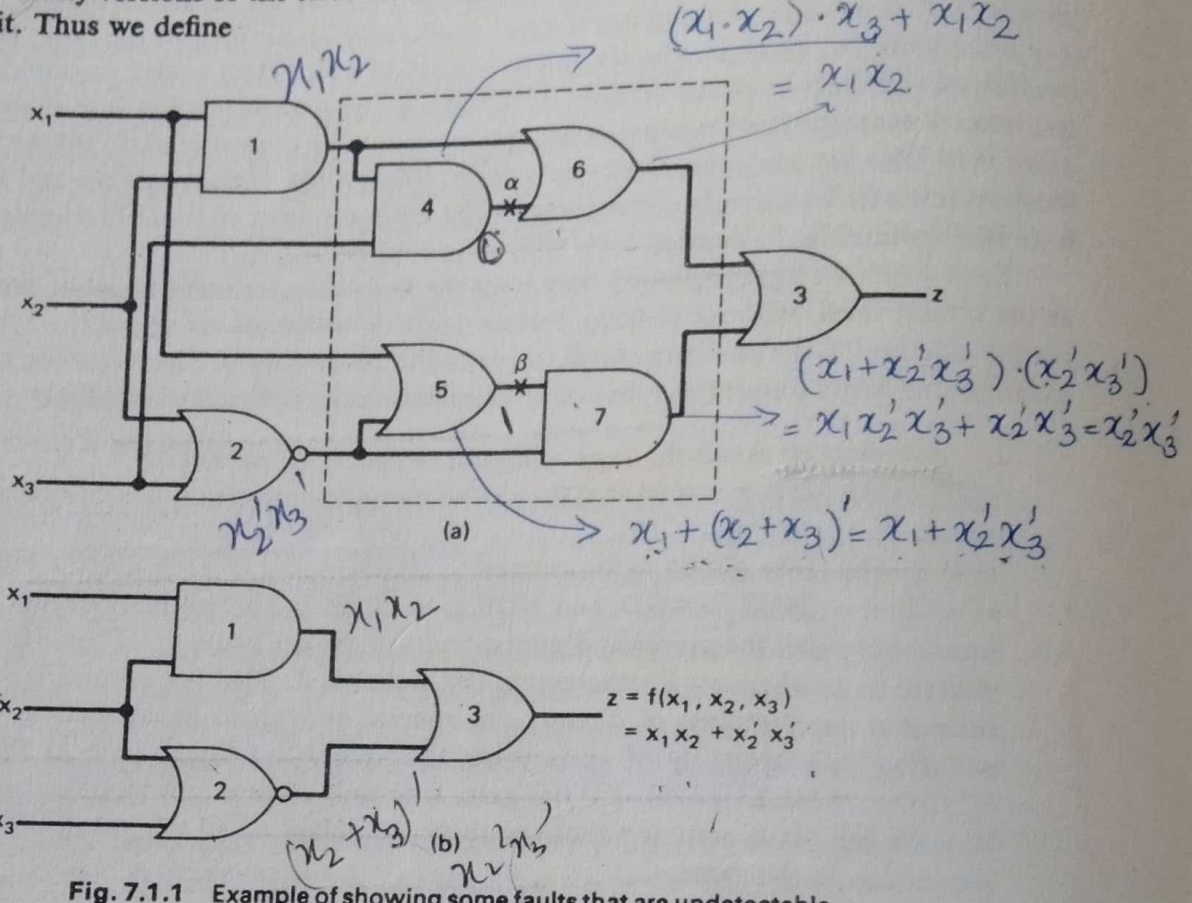


Fig. 7.1.1 Example of showing some faults that are undetectable.

DEFINITION 7.1.1

A fault of a combinational circuit is said to be *detectable* if there exists a test by which we can judge whether or not the circuit has such a fault; otherwise, we call the fault *undetectable*.

DEFINITION 7.1.2

A combinational circuit is said to be *irredundant* if any logic fault that occurred at any part of the circuit will cause a change in the switching function that the fault-free circuit realizes.

THEOREM 7.1.1

All s-a-1 and s-a-0 faults in a circuit are detectable if and only if the circuit is irredundant and if and only if the function that the circuit realizes is a minimized function.

The proof of this theorem is quite obvious and is omitted.
Consider the output function z of the circuit in Fig. 7.1.1(a):

$$z = x_1x_2 + \boxed{(x_1x_2)x_3 + (x_1 + x_2'x_3')}(x_2'x_3') \qquad (7.1.1)$$

Note that the terms $(x_1x_2)x_3$ and $x_1 + x_2'x_3'$ in the dashed frame can be deleted from the equation without changing the function. The deletion of these two terms corresponds to the removal of the four gates in the dashed frame in Fig. 7.1.1(a). It should also be noted that the function of Eq. (7.1.1) after the terms in the frame are removed is minimal, and the circuit of Fig. 7.1.1(a) after the four gates in the frame are removed (see Fig. 7.1.1(b)) is irredundant. Hence every s-a-0 and s-a-1 fault in the irredundant circuit of Fig. 7.1.1(b), wherever it occurs, will be detectable.

DEFINITION 7.1.3

A test set of a circuit is said to be *complete* if it detects every fault of the circuit under consideration. A *minimal complete* test set of a circuit is a complete test set that contains a minimum number of tests.

It is obvious that the set of all possible combinations of the values of the inputs to a combinational circuit constitute a complete test set for the circuit; that is to say that, by checking the entire truth table of a combinational circuit we can always determine whether the circuit is faculty. This method requires that every row of the truth table be tested. For a circuit with a large number of inputs n, the number of rows of the truth table of the circuit which is equal to 2^n is large. Now the question is: Can a subset of the set of the 2^n rows constitute a complete test set for the circuit? The answer is yes. This can be demonstrated by the following simple example.

Consider the NAND gate in Fig. 7.1.2. Let A and B denote the two inputs and C denote the output. The two input wires and the output wire are denoted by α, β, and γ, respectively. Suppose that we want to test a fault s-a-0 on wire α. The test that

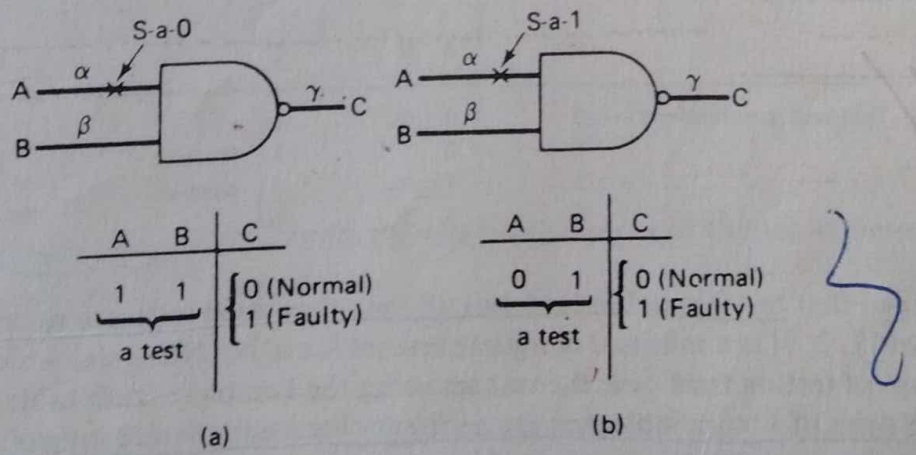


Fig. 7.1.2 Examples of illustrating the derivation of a test.

can detect this fault can be found as follows. First we find that the input B must be 1 because otherwise the output C will be 1 regardless of the value of the input A. Then we find that to test the fault s-a-0 on wire α, the input A must also be 1. Therefore the test is $A = 1$ and $B = 1$ which can be represented by a decimal number 3. By applying this test to the NAND gate and observing its output, we can determine whether the gate has the s-a-0 fault. The gate does not have the fault s-a-0 on wire α if the output is 0; otherwise, it is faulty. This is shown in Fig. 7.1.2(a). The test for detecting a s-a-1 fault on wire α which is $A = 0$ and $B = 1$ can be derived in a similar way. The normal and faulty outputs of this test are 1 and 0, respectively. This is shown in Fig. 7.1.2(b).

Now let us consider the problem of having a complete "check-up" for all possible logic faults that may occur in this gate. There are two possible faults (s-a-0 fault and s-a-1 fault) on each of the three wires which gives a total of six; namely, wire α s-a-0, wire α s-a-1, wire β s-a-0, wire β s-a-1, wire γ s-a-0, and wire γ s-a-1.

It is easy to show that the test $A = 1$ and $B = 1$ can detect the following logic faults:

wire α s-a-0

wire β s-a-0

wire γ s-a-1,

The test $A = 0$ and $B = 1$ can detect the following logic faults:

wire α s-a-1

wire γ s-a-0,

and the test $A = 1$ and $B = 0$ can detect the following logic fault:

wire β s-a-1.

The following table summarizes the above discussions.

	Test	A	B	C		C	
This test is redundant ⟶	0	0	0	1		or 0	
	1	0	1	1	Normal	or 0	Faculty
	2	1	0	1	output	or 0	output
	3	1	1	0		or 1	

It is seen that test 0 is redundant, but all the other three tests are necessary. Hence the test set {1, 2, 3} is a minimal complete test set for a NAND gate, which shows a 25% savings of testing time over the test set using the complete truth table. The percentage of the rows of a truth table that are necessary for a complete testing of a combinational circuit decreases as the number of inputs to the circuit increases. The main objective

of this chapter is to present methods for deriving a minimal complete test set (experiment) for a given combinational circuit.

The most classic method for constructing a set of tests to detect and locate a prescribed list of permanent logic faults, single or multiple, in a combinational circuit is the fault-table method. Basically, a test set (experiment) derived by this method is obtained by comparing the truth tables of the normal and faulty circuits. Two types of testing experiment will be presented. One is *fixed-scheduled*, which means that the choice of test schedules is completely independent of the outcome of the individual tests in the sequence, and the other is *adaptive (sequential)-scheduled*, which means that the choice of test schedules depends upon the outcome of the individual tests in the sequence. Derivations of fixed-scheduled and adaptive-scheduled experiments for fault detection and fault location will be discussed. In the sequel it will be shown that the length of a minimal adaptive fault-detection test set is the same as that of a minimal fixed-scheduled fault-detection test set. Hence there is no advantage in using an adaptive fault-detection experiment. But for fault location and fault location-to-within-modules, the possible reductions in test-schedule length are quite substantial when adaptive experiments are used.

Fixed-Scheduled Fault Detection The procedure of obtaining a minimal fixed-scheduled fault-detection test set (experiment) using a fault table consists of the following three steps:

Step 1 Construction of the fault table If $x_1, x_2, \ldots, x_n$ are the input variables to a single output circuit whose fault-free (correct) output is $z = z(x_1, \ldots, x_n)$ and $z^{\alpha_1}, z^{\alpha_2}, \ldots, z^{\alpha_l}$ are the erroneous outputs, each corresponding to one of the possible faults $\alpha_1, \alpha_2, \ldots, \alpha_l$, a multiple-output table of the combinations may be obtained:

Row Number r	x_1	x_2	...	x_n	z	z^{α_1}	z^{α_2}	...	z^{α_l}
0	0	0	...	0	0	1	0	...	0
1	0	0	...	1	1	1	0	...	1
.			.		.	.	.		.
.			.		.	.	.		.
.			.		.	.	.		.
$2^n - 1$	1	1	...	1	0	0	0	...	1

This is called a *fault table, F*. The complete test set for any α_i is the set of input combinations $x^j = (x_1^j, x_2^j, \ldots, x_n^j)$ such that

$$z(x^j) \oplus z^{\alpha_i}(x^j) = 1 \qquad \text{for all } 1 \leq i \leq l \tag{7.1.2}$$

DEFINITION 7.1.4

Two faults are said to be *indistinguishable* if the truth tables of the output functions of the circuits with these two faults are completely identical. In other words, we cannot

find a test such that the two faults can be distinguished based on the information of the output. If, on the other hand, there exists such a test, by applying it, the output values of the circuit having the two faults are different; we say that these two faults are *distinguishable*.

The multiple-output table above may be simplified somewhat as follows:

1. Delete the columns that correspond to undetectable faults (see Definition 7.1.1); that is, delete the columns z^{α_i} in the table that are identical to z.
2. Combine the columns that correspond to indistinguishable faults; that is, combine the columns z^{α_i} and z^{α_j} if $z^{\alpha_i} = z^{\alpha_j}$.

The reduced fault table F^* still contains the complete information about the complete test sets for detecting all the detectable faults. In fact, the complete test set for detecting each detectable fault α_i may be obtained by taking exclusive-or operations on column z^{α_i} with column z as in Eq. (7.1.2).

Step 2 Construction of the fault-detection table The fault-detection table or fault-detection matrix G_D is constructed as follows: Each column i, $i = 1, 2, \ldots, l$ of G_D presents a distinguishable fault under detection and the entries of the column are obtained from the reduced-fault table by $z(x^j) \oplus z^{\alpha_i}(x^j)$. The rows of G_D are the input combinations of the union of all the complete test sets for detecting the faults. If there are m distinguishable faults and the union of all the complete test sets contain p tests, the size of G_D is $p \times m$. An entry in row j and column i is 1 if row j is a test for fault i, and is 0 if row j is not a test for fault i. Since there is no need for displaying both the 0's and 1's entries in the table G_D, it is customary to display the 1's only and to leave all the 0 entries blank.

Step 3 Determination of a minimal test set The first step toward obtaining a minimal test set from a fault-detection table is simplification of the table by deleting certain superfluous rows and columns following the two rules:

RULE 1

Delete any row whose 1's all fall in the same columns as the 1's in some other row. That is, delete any row that is covered by, or is the same as, some other row.

RULE 2

Delete any column that has 1's in all the rows in which another column has 1's. That is, delete any column that covers, or is the same as, some other column.

These steps may be applied in any order until neither is applicable. The reduced table G_D^* has distinct rows and distinct columns; also, no row covers another row, and no column covers another column.

Let us observe the following analogies:

	Test-Set Minimization	Switching-Function Minimization
Rows	⟷ Tests	⟷ Implicants
Columns	⟷ Faults	⟷ Minterms
Table	⟷ Fault table	⟷ Implicant table
Reduced table	⟷ Reduced fault table	⟷ Prime-implicant table

Therefore, the problem of choosing a minimal set of tests that can detect all the faults under detection may be transformed into the familiar problem of minimizing a switching function, which has been solved in Sections 1.4–1.7.

Fixed-Scheduled Fault Location The difference between the fault-detection problem and the fault-location problem is that the former is concerned only with determining whether a circuit has a detectable fault, whereas the latter, in addition to the determination of whether the circuit is faulty or not, demands determination of the location of the fault that has occurred. There are two types of approaches to the latter: the fixed-scheduled experiment and the adaptive-scheduled experiment. The method for deriving a shortest fixed-scheduled fault-location experiment using the fault table is quite similar to the one for fault detection just described.

The fault-table method for constructing a shortest fixed-scheduled experiment for fault location consists of three steps:

Step 1 Construction of the fault table.

Step 2 Construction of the fault-location table. The fault-location table G_L is constructed from the reduced fault table F^*. A column of G_L is formed by taking the exclusive-or operation on a pair of columns of F^*. If we let f_0 be the output of an n-input single-output combinational circuit with no detectable faults, and $f_1, f_2, \ldots, f_m$ be the outputs of the circuit with m faults, respectively, then the fault-location table has 2^n rows denoted by 0, i, . . . , $2^n - 1$, and $m(m+1)/2$ columns, each of which is obtained by $f_i \oplus f_j$, $0 \le i < j \le m$. If it is only required to locate the faults within the limits of the modules of the circuit, the table G_M for fault location-to-within-modules is constructed from F in the same way as G_L, except for one column for each pair of columns of F^* that belong to different modules. It should be noted that G_D is always contained in G_L and G_M.

Step 3 Find a minimal cover of G_L or G_M.

The fault-table method for fault detection and fault location is illustrated by the following example.

Example 7.1.1

Consider the circuit of Fig. 7.1.3. The output function of this circuit is $z = f(x_1, x_2, x_3) = x_1x_2 + x_2x_3$. The fault table for detection and location of all the single faults of the circuit is shown in Table 7.1.1. In this table, f_0 denotes the output function of the faultless version of the circuit, and f_{ij} represents the output function of the circuit with its ith line stuck at 0 ($j = 0$) or 1 ($j = 1$). It is observed that all the faults are detectable, and four groups of faults $\{f_{10}, f_{20}, f_{50}\}, \{f_{11}, f_{41}\}, \{f_{30}, f_{40}, f_{60}\}$, and $\{f_{51}, f_{61}\}$ in this table are indistinguishable faults. We combine them, choose one function from each group, and delete the rest of them from the

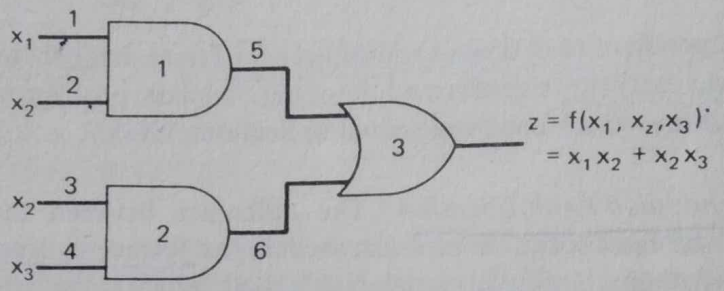


Fig. 7.1.3 Example to illustrate fault location using the fault table.

TABLE 7.1.1 Fault Table for Detection and Location of all Single Faults of the Circuit of Fig. 7.1.2

r	x_1	x_2	x_3	f_0	f_{10}	f_{11}	f_{20}	f_{21}	f_{30}	f_{31}	f_{40}	f_{41}	f_{50}	f_{51}	f_{60}	f_{61}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1
2	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	1
3	0	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1
4	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1
5	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1
6	1	1	0	1	0	1	0	1	1	1	1	1	0	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				f_0		f_1	f_2		f_3	f_4	f_5	f_6				

table. Name the six distinct output functions which represent six distinguishable faults to be $f_1, \ldots, f_6$, as indicated at the bottom the table. From the functions $f_0, \ldots, f_6$, we form the fault-location table G_L, which is shown in Table 7.1.2. The column label kl represents the functions f_k and f_l, from which the column is constructed ($f_k \oplus f_l$), where $0 \leq k < l \leq m$, and m denotes the total number of distinguishable faults. So there are $C_2^m = m(m-1)/2$ columns in G_L. Note that it is not necessary to show all the 0 entries of this table, and that the subtable G_D of this table, composed of the first six columns, is, in fact, the fault-detection table needed to detect the above-designated faults. From this subtable it is seen that tests 2, 3, and 6, which cover columns 01, 02, 04, and 06, are essential, and columns 03 and 05 can be covered by test 5. Thus the minimal complete test set for detection of all single faults

TABLE 7.1.2 Fault-Location Table G_L of Example 7.1.1, Which Includes the Fault-Detection Table as a Subtable

Test \ Fault	01	02	03	04	05	06	12	13	14	15	16	23	24	25	26	34	35	36	45	46	56
0						1					1				1			1		1	1
1					1	1				1	1			1	1		1	1	1	1	
*2		①				1	1				1	1	1	1				1		1	1
*3			①					1					1			1			1	1	
4		1			1			1			1	1			1	1	1			1	1
5		1		1	1			1		1	1	1		1	1	1			1	1	
*6	①						1	1	1	1	1										
7																					

Fault-detection table G_D

of this circuit is {2, 3, 5, 6}. Note that in this example we only need four out of a total of eight tests to completely check all the single faults of the circuit, which shows a 50% savings of testing time over the test time using the complete truth table. A close examination of the faults that are covered by each of these four tests shows that

Test	x_1	x_2	x_3	
3	0	1	1	} test all the s-a-0 faults
6	1	1	0	
2	0	1	0	} test all the s-a-1 faults
5	1	0	1	

Now let us examine the fault-location table, which is the entire table of Table 7.1.2. It is observed that tests 2, 3, and 6 are essential tests which cover all the columns except the four columns 03, 05, 26, and 35. With these columns and the nonessential rows (tests), we construct the fault-location table:

Test \ Fault	03	05	26	35
0			1	
1		1	1	1
4	1		1	1
5	1	1	1	

Test 0 is dominated by test 1 and column 26 dominates all the other three columns, so they can be removed from the table. The remaining table shows that any choice of two tests from the three tests 1, 4, and 5, plus the essential tests, will cover all the columns of G_L (i.e., will

locate all six distinguishable faults). Hence the minimal fault-location experiment for this circuit should consist of tests 2, 3, and 6, plus any two of the three tests 1, 4, and 5, as shown:

					Response of Faulty Circuits					
Test	x_1	x_2	x_3	f_0	f_1	f_2	f_3	f_4	f_5	f_6
2	0	1	0	0	0	1	0	0	0	1
3	0	1	1	1	1	1	1	0	1	1
6	1	1	0	1	0	1	1	1	1	1
plus 1	0	0	1	0	0	0	0	0	1	1
plus 4	1	0	0	0	0	0	1	0	0	1
plus 1 or 5	0	0	1	0	0	0	0	0	1	1
plus 5	1	0	1	0	0	0	1	0	1	1
plus 4 or 5	1	0	0	0	0	0	1	0	0	1
plus 5	1	0	1	0	0	0	1	0	1	1

Now let us examine these tests and the faults that they detect and locate. This is illustrated in Table 7.1.3. It is seen that the test set containing only three tests, 2, 3, and 6, cannot detect f_3 and f_5 because they are identical to f_0. A test set consisting of these three sets plus one additional test may be able to detect all the single faults. But this additional test must be test 5 and not test 1 or test 4, because test set {2, 3, 6, 1} cannot detect f_3, and test set {2, 3, 6, 4} cannot detect f_5. As a consequence, a test set consisting of five tests, tests 2, 3, and 6, plus two of the three tests 5, 1, and 4, will detect and locate all the six distinguishable single faults. The test set {2, 3, 6, 5, 1 or 4} and the responses of the circuit under no fault and a single (distinguishable) fault condition are shown in Table 7.1.3.

TABLE 7.1.3 Fault-Detection and Fault-Location Tests and the Responses of the Circuit of Fig. 7.1.3 with No Fault and a Single Distinguishable Fault

Test \ Fault	f_0	f_1	f_2	f_3	f_4	f_5	f_6
2	0	0	1	0	0	0	1
3	1	1	1	1	0	1	1
6	1	0	1	1	1	1	1
	(Test set {2, 3, 6} cannot detect f_3 and f_5)						
5	0	0	0	1	0	1	1
	[Test set {2, 3, 6, 5} can detect all the faults but cannot locate (distinguish) faults between f_3 and f_5]						
1 or 4	0	0	0	0	0	1	1
	0	0	0	1	0	0	1
	(All the faults are detected and located)						

Now suppose that instead of locating each individual fault, we only want to know which of the three gates of the circuit is faulty. By examining the definition of faults f_{ij} and distin-

guishable faults $f_1, \ldots, f_6$, we find that the following relation exists between the individual faults and the three gates:

Faults		Faulty Gate
f_1, f_3	indicate	gate 1
f_4, f_5	indicate	gate 2
f_2	indicates	gate 1 or gate 2
f_6	indicates	gate 3

The third row in this table demands an explanation. Fault f_2 represents two indistinguishable faults, f_{11} and f_{41}, which indicate a fault in gate 1 and gate 2, respectively. In other words, fault f_2 may come either from f_{11}, which indicates that gate 1 is faulty, or from f_{41}, which indicates that gate 2 is faulty. Thus the presence of f_2 may indicate either gate 1 or gate 2 being faulty (but not gate 3, of course). The G_M for this case is the same as G_L of Table 7.1.2, except that certain columns representing pairs of faults (i.e., 13 and 45) within the same module (gate) need not be included. It can easily be shown that the same test set for locating each individual single fault is required for detecting and locating a faulty gate of the circuit.

Suppose that the circuit is constructed by two modules, module 1, containing the two AND gates, and module 2, containing the OR gate. In this case the relation between the individual single fault and the faulty modules indicated by them is

Faults		Faulty Modules
f_1, f_2, f_3, f_4, f_5	indicate	module 1 (gates 1 and 2)
f_6	indicates	module 2 (gate 3)

By deleting columns 12, 13, 14, 15, 23, . . . , 45, from the G_L of Table 7.1.2 and finding the minimal cover for it, we find that the fault-detection test set {2, 3, 6, 5} obtained previously will locate any single fault within the two modules.

Let N_D, N_L, and N_M be the number of test inputs in the minimal experiments for fault detection, fault location, and fault location-to-within-modules. It should be remarked that for detecting or locating a set of faults of a circuit, the following relation always holds:

$$N_D \leq N_M \leq N_L$$

For example, in the example above, $N_D = 4$ and $N_L = 5$. $N_M = 5$ if each gate is considered as a module, and $N_M = 4$ if the two AND gates and the OR gate are considered as two modules of the circuit.

We have been discussing fixed-scheduled fault detection and location. In the following we will discuss the same problems but the test schedule will be adaptive. It will be shown that the use of an adaptive test schedule will be of no benefit in fault detection, but in fault location and fault location-to-within-modules, the possible

reductions in test-schedule length are substantial. A heuristic procedure is given which is easy to carry out and reasonably effective, although it does not necessarily lead to a test whose length is absolutely minimal.

Adaptive-Scheduled Fault Detection and Location It may be observed that the choice of test schedules derived above is completely independent of the outcome of the individual tests in the sequence; moreover, the length of the schedule is independent of the order in which the tests are performed. It is quite conceivable, however, that after the first test input of a schedule has been applied and the output noted, the residual test schedule, which is minimal with respect to a 0 output, is not the same as that which is minimal with respect to a 1 output. Similarly, after two test inputs have been applied, the four partial test schedules that should follow may be all different in content and length, and so on for successive test inputs. We consider here the economies that can be achieved by choosing each test input to be applied to the circuit on the basis of the outcomes of all previous tests in the schedule. A convenient way to present such a sequence of tests and their outcomes is to use a diagnosing tree, which is defined as follows:

DEFINITION 7.1.5

A *diagnosing tree* is a directed graph whose nodes are tests. The outgoing branches from a node represent the different outcomes of the particular test.

It is easy to show that adaptive-scheduled fault detection requires the same test length as that of fixed-scheduled fault detection. Take the fault detection of the circuit of Fig. 7.1.3, for example. A diagnosing tree of the fixed-scheduled fault-detection experiment {2, 3, 5, 6} for the circuit may be constructed from the fault table (Table 7.1.1), as shown in Fig. 7.1.4(a). From this diagnosing tree, we see that the circuit is fault-free if and only if the output sequence to the sequence (2, 3, 6, 5) is (0, 1, 1, 0), as indicated by the path in dark lines. Thus this tree can be simplified to the one shown in Fig. 7.1.4(b). Applying tests 2, 3, 5, 6 in any order will not shorten the length of the experiment. Another diagnosing tree for test set {2, 3, 5, 6} is shown in Fig. 7.1.4(c), which still requires all four tests. Since the adaptive-scheduled fault detection has no advantage over the fixed-scheduled fault detection, we can completely ignore it.

Unlike fault detection, adaptive-scheduled fault location in general yields a shorter experiment. For example, the fault location of the circuit of Fig. 7.1.3, using the fixed-scheduled experiment, requires a minimal length of 5 but needs only 4 when the adaptive-scheduled experiment is used. The diagnosing trees of the fixed-scheduled fault-location experiment {5, 4, 6, 3, 2} and an adaptive-scheduled fault-location experiment for this circuit are shown in Fig. 7.1.5(a) and (b), respectively.

Now the question is: How to find an adaptive schedule of tests such that the number of the test levels is minimal? First, several interesting observations are in order.

1. Since our goal has now been changed from finding a minimal test set to finding an (adaptive) test schedule with a minimal number of levels, in constructing such

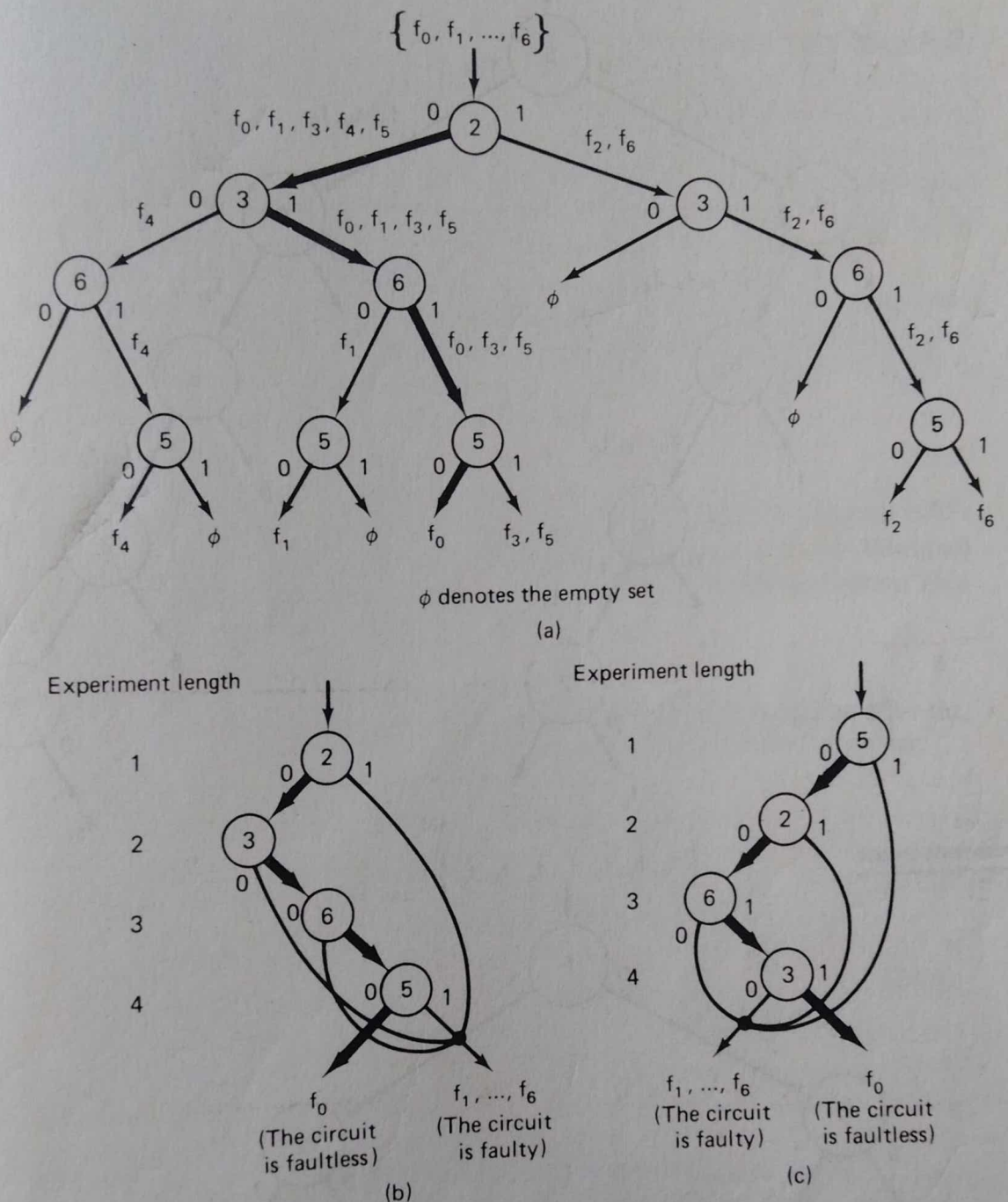


Fig. 7.1.4 (a) Diagnosing tree for fault detection of the circuit of Fig. 7.1.3; (b) simplified diagnosing tree; (c) another diagnosing tree for the same test set {2, 3, 5, 6}.

an experiment, tests chosen must not be confined to any particular given set of tests (e.g., any of the minimal fixed-schedule fault-location test sets). They must be chosen from the rows of the fault table.

2. The construction of an adaptive schedule of tests for fault location with a minimal number of levels needs at least N_L tests.

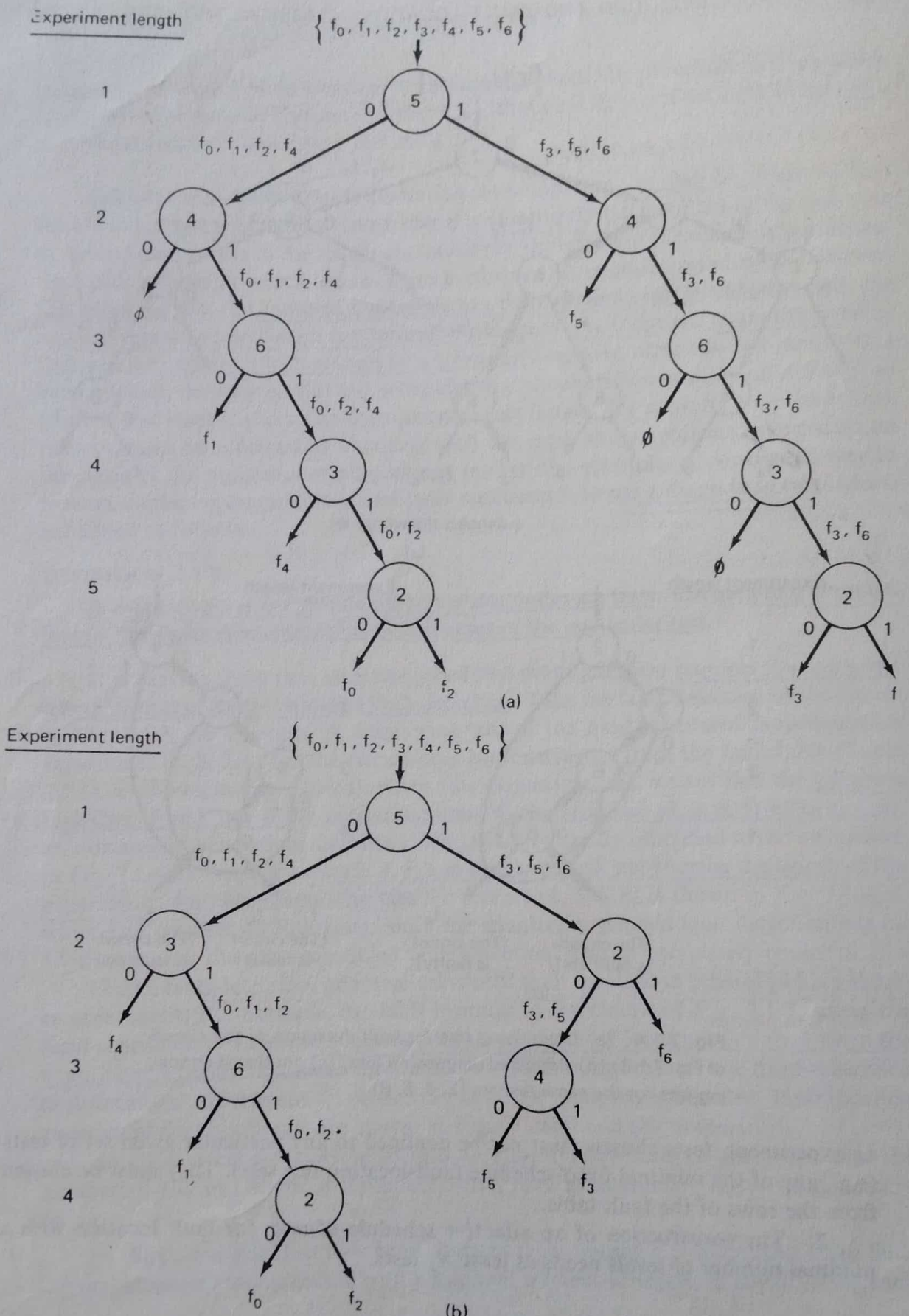


Fig. 7.1.5 Diagnosing tree of the fixed-schedule experiment {5, 4, 6, 3, 2}.

3. In constructing this experiment, at each step the test that will distinguish between the largest number of faults not already distinguished should be chosen.

4. One way to select the appropriate row (test) having the desired property described in observation 3 at each step of the procedure is to try all possible remaining rows (tests). For even a small fault table, however, the table number of possible graph labelings that must be tried to determine the minimal number of levels is astronomical. This approach is therefore impractical.

Let w_{i0} and w_{i1} denote the numbers of 0's and 1's in row i, respectively. A simple heuristic method for finding a nearly minimal adaptive-scheduled fault-location experiment is: Select row i, which maximizes the number of (0, 1) pairs between digits in that row, that is, which maximizes the exprsssion

$$R_i = w_{i0}w_{i1} \tag{7.1.3}$$

This number is optimized if the row that has the most nearly equal distribution of 0's and 1's, [i.e., the row (or one of the subset of rows) for which $|w_{i0} - w_{i1}|$ is minimal] is selected. The use of this criterion appears to work very well for many problems. This method is illustrated by the following example.

Example 7.1.1 (Continued)

Consider the construction of an adaptive-scheduled fault-location experiment for the circuit of Fig. 7.1.3 whose fault table (Table 7.1.1) may be presented in matrix form as

$$F^* = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 \end{matrix} \\ \left[\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \tag{7.1.4}$$

For this matrix, row 5 should be chosen first, since it has 4 0's and 3 1's and $|w_{50} - w_{51}| = 1$, and all the other rows of this matrix whose $|w_{i0} - w_{i1}|$ are greater than 1. Selection of row 5 yields the two submatrices

$$F_0^*(5) = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & f_4 \end{matrix} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3, \\ 4 \\ 6 \\ 7 \end{matrix} \qquad F_1^*(5) = \begin{array}{c} \begin{matrix} f_3 & f_5 & f_6 \end{matrix} \\ \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 6 \\ 7 \end{matrix} \tag{7.1.5}$$

The selection of subsequent rows for the two submatrices should be considered separately. For $F_0^*(5)$, there are three rows, rows 2, 3, and 6, whose $|w_{i0} - w_{i1}| = 2$, and those of the rest of the rows are 4. Thus any of the three rows may be chosen. Say that we choose row 3. Now consider $F_1^*(5)$. It is obvious that any rows of this matrix may be chosen except rows 3, 6, and 7. Suppose that we choose row 2. Then the two submatrices are further partitioned intc four smaller submatrices:

$$F_{00}^*(5,3) = \begin{matrix} f_4 \\ \left[\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{matrix}\right. \end{matrix} \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 4' \\ 6 \\ \left.\begin{matrix} \\ \\ \\ \\ \\ 7 \end{matrix}\right] \end{matrix} \qquad F_{01}^*(5,3) = \begin{matrix} f_0 & f_1 & f_2 \\ \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}\right. \end{matrix} \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 4' \\ 6 \\ \left.\begin{matrix} \\ \\ \\ \\ \\ 7 \end{matrix}\right] \end{matrix} \tag{7.1.6}$$

$$F_{10}^*(5,2) = \begin{matrix} f_3 & f_5 \\ \left[\begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{matrix}\right. \end{matrix} \begin{matrix} \\ 0 \\ 1 \\ 3 \\ 4' \\ 6 \\ \left.\begin{matrix} \\ \\ \\ \\ \\ 7 \end{matrix}\right] \end{matrix} \qquad F_{11}^*(5,2) = \begin{matrix} f_6 \\ \left[\begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{matrix}\right. \end{matrix} \begin{matrix} \\ 0 \\ 1 \\ 3 \\ 4 \\ 6 \\ \left.\begin{matrix} \\ \\ \\ \\ \\ 7 \end{matrix}\right] \end{matrix}$$

Now we have two submatrices, $F_{00}^*(5,3)$ and $F_{11}^*(5,2)$, which have only one column. This means that the faults denoted by f_4 and f_6 are already located by the sequences of tests (5, 3) and (5, 2), respectively. Continue this process until all the submatrices form a single-column matrix. One such adaptive experiment having the minimal number of levels obtained by this procedure is shown in Fig. 7.1.5(b).

The method for deriving the shortest test sequence to locate any individual faults just described applies with little change to fault location-to-within-modules. We now modify the criterion of row acceptance to count not all (0, 1) pairs, but only those pairs in each row in which the 0 and the 1 fall in different module classes. This quantity is most easily calculated by subtracting the sum of the number of (0, 1) pairs that fall entirely within the individual classes from the total number of (0, 1) pairs:

$$R_i = w_{i0}w_{i1} - \sum_{j=1}^{p} w_{ij0}w_{ij1} \tag{7.1.7}$$

where w_{ij0} and w_{ij1} are the number of 0's and 1's, respectively, in the jth module class in row i. The row to be selected is the one with the largest row count R_i.

Example 7.1.1 (Continued)

As an example for adaptive-scheduled fault location-to-within-modules, suppose that in the above F^* matrix, faults f_1, f_2, and f_4 are associated with a single module, as are faults

f_3, f_5, and f_6. The row counts on the eight rows of F^* are

0: $6 - 0 - 2 = 4$
1: $10 - 0 - 2 = 8$
2: $10 - 2 - 2 = 6$
3: $6 - 2 - 0 = 4$
4: $10 - 0 - 2 = 8$
5: 12
6: $6 - 2 - 0 = 4$
7: 0

Clearly, row 5 should be selected first, and the same two submatrices $F_0^*(5)$ and $F_1^*(5)$ that appeared in Eq. (7.1.5) arise here. $F_1^*(5)$ falls entirely within (and in fact covers exactly) module class (f_3, f_5, and f_6); therefore, it need not be reduced further. For the rows of $F_0^*(5)$, the row-count values are

0: 0
1: 0
2: $3 - 2 = 1$
3: $3 - 2 = 1$
4: 0
6: $3 - 2 = 1$
7: 0

Any of 2, 3, or 6 should be chosen. Selecting 2, only the $F_{00}^*(5, 2)$ need be further reduced, and any of the nonconstant rows 3 or 6 can be used. Figure 7.1.6(b) shows the resulting diagnosing tree. The diagnosing tree for fixed-scheduled fault location-to-within-modules for the same modules is shown in Fig. 7.1.6(a). It is seen that for locating faults f_3, f_5, and f_6 within the module, one test (test 5) is sufficient if the adaptive-scheduled experiment is used.

From the above discussion, the following bounds can readily be established.

$$1 \leq l_D = N_D \leq m$$
$$1 + \lceil \log_2 m \rceil \leq l_L \leq N_L \leq m$$
$$1 + \lceil \log_2 p \rceil \leq l_M \leq N_M \leq m$$

where m and p denote the number of distinguishable faults under investigation and the number of module classes, respectively; l denotes the number of levels in adaptive test schedules. For most problems, of course, we can expect l_L to be much smaller than N_L.

The heuristic adaptive-scheduled fault-location method provides a good but not necessarily optimal solution. The procedure is quite straightforward and easy to apply.

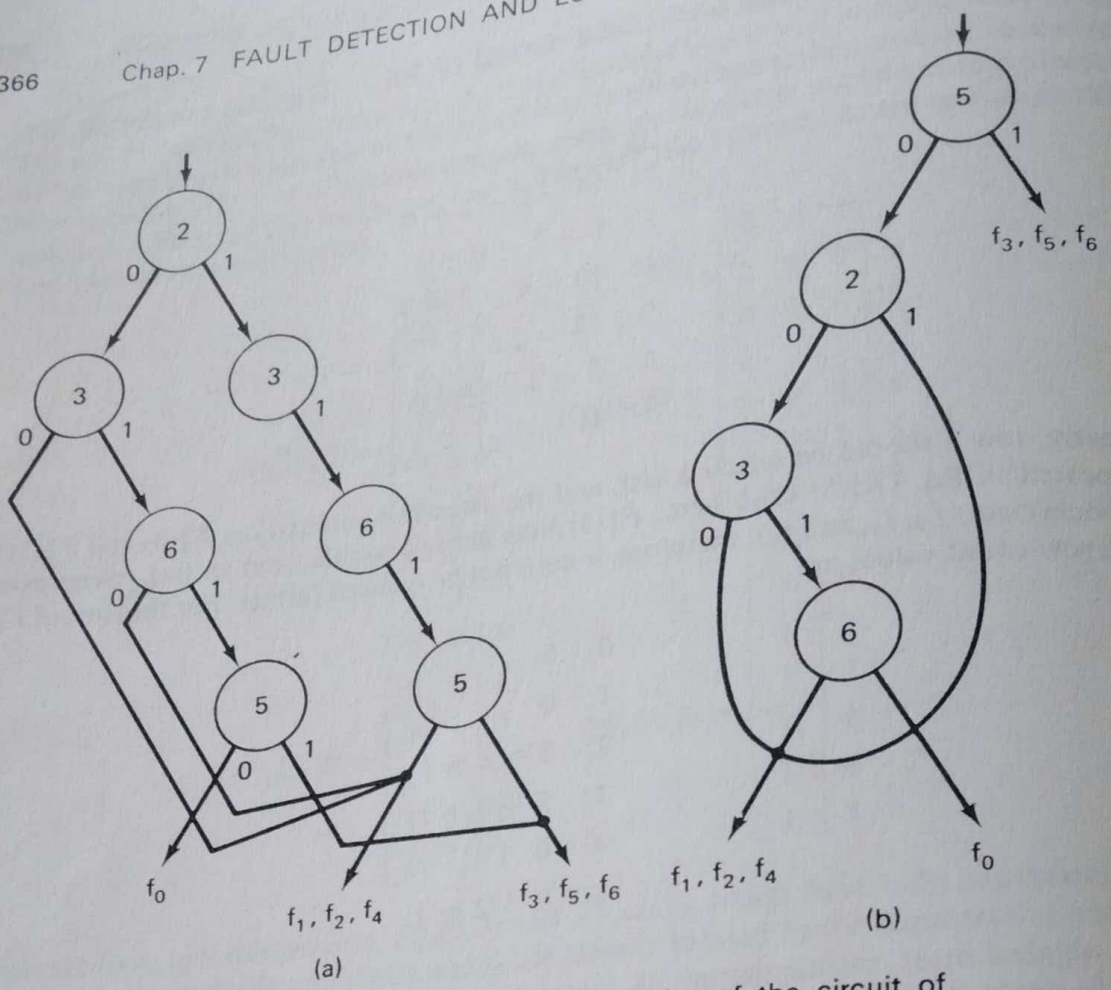


Fig. 7.1.6 Fault location-to-within-modules of the circuit of Fig. 7.1.3. (a) Diagnosing tree of the fixed-scheduled experiment {2, 3, 6, 5}. (b) Diagnosing tree of the adaptive-scheduled experiment {5, 2, 3, 6}.

The drawback of this method is that it requires a large amount of computer storage space to store the fault table. When the size of the fault table exceeds the amount available, we modify the above method to the *hybrid method*, which is described below.

Let n be the number of primary inputs of a circuit and m be the number of single faults of the circuit under consideration. Then the size of the fault table will be $2^n(m + 1)$. Suppose that the available memory of a computer can take only s columns of the $m + 1$ columns at a time. The hybrid method consists of the following steps:

1. Enter the memory with s arbitrary columns of the fault table and obtain an adaptive-scheduled experiment for it by the heuristic method described above. Record this experiment and then clean up the memory.

2. Enter the rest columns into the computer, s columns at a time, and apply the obtained adaptive-scheduled experiment to each of these $2_n \times s$ submatrices. Note that the application of the adaptive-scheduled experiment to these sets of faults is fixed-scheduled.

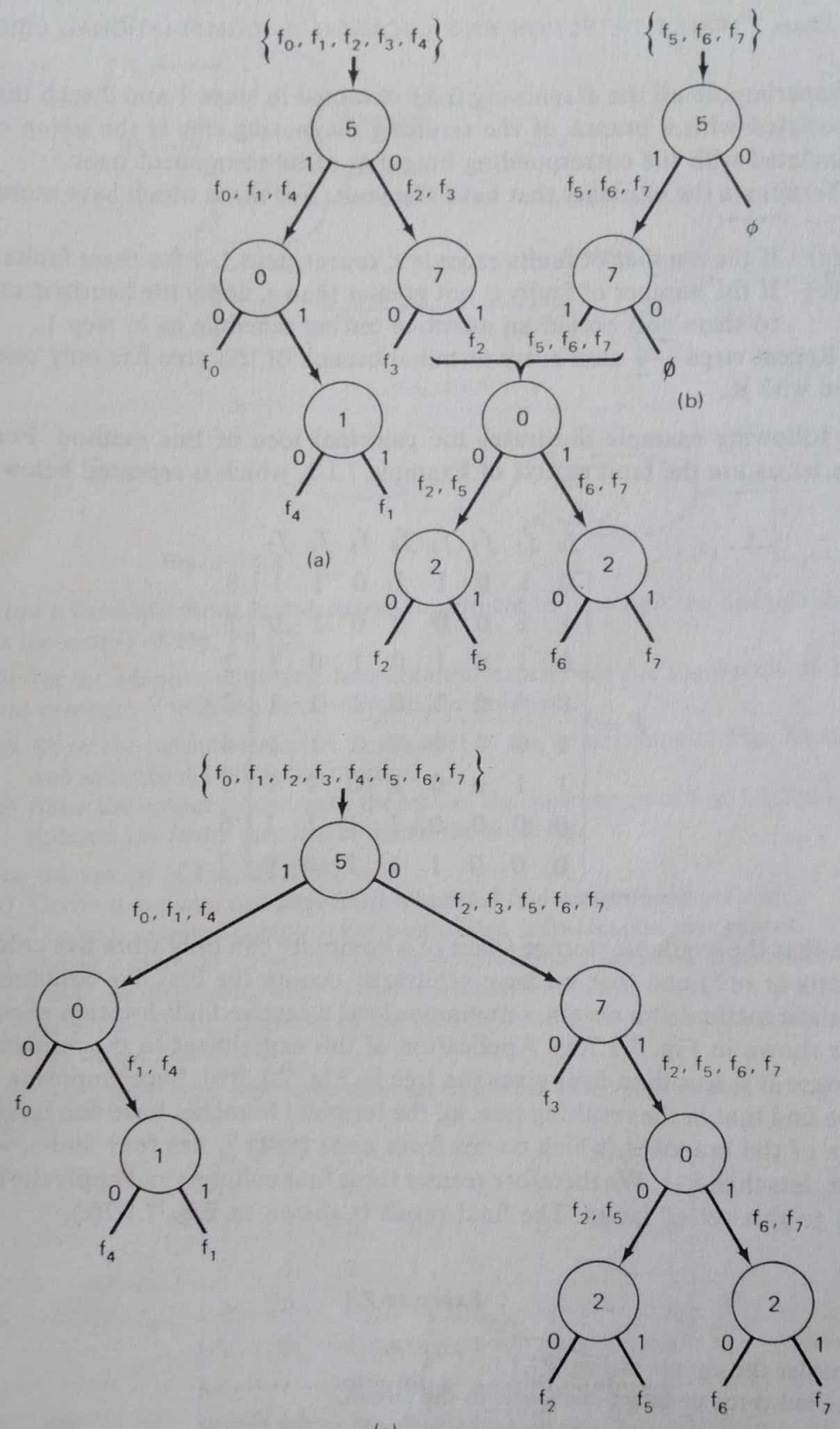


Fig. 7.1.7 Example to illustrate the hybrid method.

3. Superimpose all the diagnosing trees obtained in steps 1 and 2 such that the faults associated with a branch of the resulting diagnosing tree is the union of the faults associated with the corresponding branches of its component trees.

4. Terminate the branches that have one fault. For those which have more than one fault:
   (a) If the number of faults exceeds s, repeat steps 1–3 for these faults.
   (b) If the number of faults is not greater than s, apply the heuristic method to them and obtain an adaptive testing schedule as in step 1.

5. Repeat steps 1–4 until every terminal branch of this tree has only one fault associated with it.

The following example illustrates the principal idea of this method. For convenience, let us use the fault matrix of Example 7.1.1, which is repeated below.

$$F = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \end{matrix} \\ \left[\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

Suppose that the available storage space of a computer can only store five columns of this matrix ($s = 5$) and that we have arbitrarily chosen the first five columns. From the heuristic method, we obtain a minimum-level adaptive fault-location experiment, which is shown in Fig. 7.1.7(a). Application of this experiment to the remaining columns (since it is less than five) gives the tree in Fig. 7.1.7(b). Superimposing the two trees, we find that in the resulting tree, all the terminal branches have one fault, except that one of the branches, which comes from node (test) 7, has four faults, which is, however, less than five. We therefore reenter these four columns and apply the heuristic method to this set of faults. The final result is shown in Fig. 7.1.7(c).

Exercise 7.1

1. Consider the circuit of Fig. P7.1.1.
   (a) Find three undetectable faults in the circuit.
   (b) Find three pairs of indistinguishable faults in the circuit.
2. Find a minimal complete test set for detecting all distinguishable single faults in the irredundant circuit of Fig. P7.1.2 by the fault-table method.

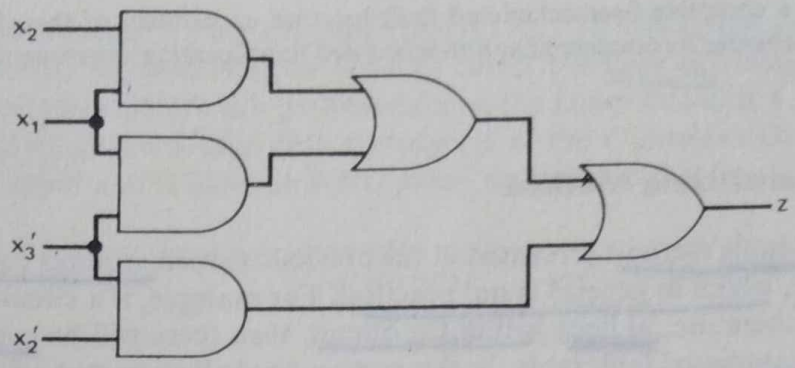


Fig. P7.1.1

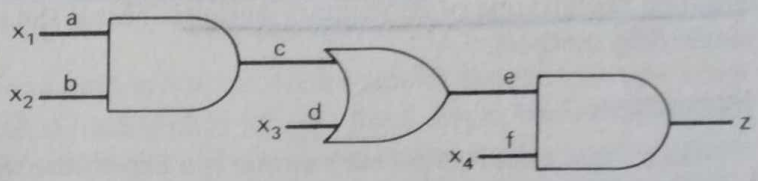


Fig. P7.1.2

3. Find a fixed-scheduled fault-location experiment to locate all the distinguishable faults in the circuit of Fig. P7.1.2.
4. Derive an adaptive-scheduled fault-location experiment for the circuit of Fig. P7.1.2 and compare it with the result obtained in problem 3.
5. (a) Show the output sequences to the tests of the experiments of Fig. 7.1.4(a) and (b) and indicate the faults they locate.
   (b) Show the output sequences to the tests of the experiments of Fig. 7.1.5(a) and (b) and indicate the faults they locate within the modules.
6. For the circuit of Fig. 7.1.1(b):
   (a) Derive a minimal complete fixed-scheduled fault-detection experiment.
   (b) Derive a minimal complete fixed-scheduled fault-location experiment.
   (c) Derive a minimal complete adaptive-scheduled fault-location experiment.
   (d) Draw the diagnosing trees of the experiments obtained in (a), (b), and (c).
   (e) Show the output sequences to the tests of these experiments and indicate the faults they detect and locate.
7. Consider a fault table F of some three-input circuit which is presented in matrix form as follows:

$$F = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \end{matrix} \\ \left[\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

Show that a complete fixed-scheduled fault-location experiment of this circuit requires four tests, whereas a complete adaptive-scheduled fault-location experiment requires only three levels.

7.2 Path-Sensitizing Method

The fault-table method presented in the previous section requires construction of the fault table, which in general is not practical. For example, if a circuit has, say, 20 inputs and if there are 30 lines within the circuit, then there will be over 30,000,000 entries in the associated fault table. In this section we shall show that a fault-detection test may be found by examining the paths of transmission from the location of an assumed fault to one of its primary outputs. This is the principal idea behind the path-sensitizing method.

DEFINITION 7.2.1

In a logic circuit, a *primary output* is a line whose signal output is accessible to the exterior of the circuit, and a *primary input* is a line that is not fed by any other line in the circuit.

DEFINITION 7.2.2

A *transmission path*, or simply path of a combinational circuit, is a connected, directed graph containing no loops from a primary input or internal line to one of its primary outputs.

DEFINITION 7.2.3

A path is said to be *sensitized* if the inputs to the gates along this path are assigned values so as to propagate any change on the faulty line along the chosen path to the output terminal of the path.

For example, consider the irredundant circuit of Fig. 7.1.1(b), which, for convenience, is repeated in Fig. 7.2.1. Each line of the circuit is labeled by a letter, $a, b,$

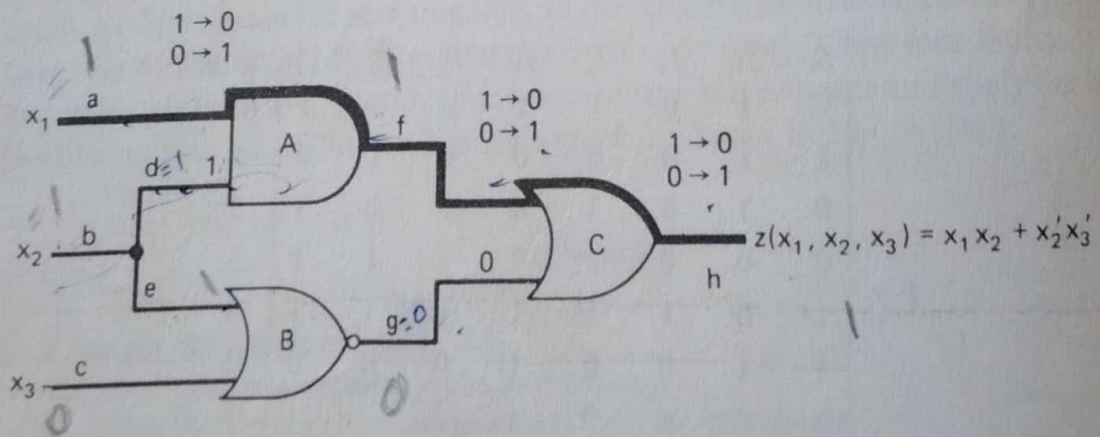


Fig. 7.2.1 Circuit to illustrate the concept of sensitized path.

$c, \ldots$. For convenience, we use the subscripts 0 and 1 to denote the s-a-0 and s-a-1 faults, respectively. Suppose that we want to detect the fault a_0 (line a s-a-0). The connected directed graph afh is a path containing the faulty line a. It is interesting to observe that if we assign a value 0 to the input g of the C gate (an OR gate) and a value 1 to the input d of A gate (an AND gate), we then see that

Value Change on Line a	*Corresponding Change in Value on Line f*	*Corresponding Change in Value on Line h*
$0 \longrightarrow 1$	$0 \longrightarrow 1$	$0 \longrightarrow 1$
$1 \longrightarrow 0$	$1 \longrightarrow 0$	$1 \longrightarrow 0$

The actual changes in values along path afh upon occurrence of the fault at line a are indicated in Fig. 7.2.1. Path afh is thus said to be *sensitized*. From the value of line $g = 0$ it is required that c or e or both equal to 1. If we choose to make $e = 1$ and $c = 0$ or 1 (don't care), then the values of d and e are 1 and thus are *consistent*; hence a test exists for detecting the fault a_0 using this method. Moreover, we observe that

Path	*Input Variable Values Required to Sensitize the Path*	*Value of the Input Variable of the Path*	*Response*
afh	$x_2 = 1$, $x_3 = 0$ or 1 (don't care)	$x_1 = 1$	1, if the circuit is faultless 0, if the circuit has fault a_0
		$x_1 = 0$	0, if the circuit is faultless 1, if the circuit has fault a_1

Furthermore, it is apparent that a test which detects fault a_0 also detects faults f_0 and h_0 and that a test which detects fault a_1 also detects faults f_1 and h_1. Thus the above two tests together would detect *all* the s-a-0 and s-a-1 faults on this path.

From the above simple example, a systematic procedure for detecting a single fault by path sensitizing may be described as follows:

Step 1 Choose a path from the faulty line to one of the primary outputs.

Step 2 Assign the faulty line a value 0 (1) if the fault is a s-a-0 (s-a-0) fault.

Step 3 Along the chosen path, except the lines of the path, assign a value 0 to each input to the OR and NOR gates in the path and a value 1 to each input to the AND and NAND gates in the path.

Step 4 Trace back from gates along the sensitized path toward the circuit inputs. If a consistent input combination (a test) exists, the procedure is terminated. If, on the other hand, a contradiction is encountered, choose another path which starts at the faulty line, and repeat the above procedure.

To appreciate the advantage of this procedure over the fault-table method, an example of deriving a complete test set using the path-sensitizing method is given below.

Example 7.2.1

Consider the same circuit of Example 7.1.1 that was used to illustrate the fault-table method. By applying the above procedure to sensitize the four paths of the circuit, *afh*, *bdfh*, *begh*, and *cgh*, the required input-variable values, the assigned value of the input variable of the paths, and the faults detected by the tests are shown in Table 7.2.1. It is seen that the faults listed in the last column of this table include all the single faults of the circuit. From this table it is an easy matter to obtain a complete test set, which is {0, 2, 5, 7}. This set is one of the four minimal complete test sets obtained in Example 7.1.1, but the effort as well as the information storage space required here for obtaining it is much less compared to that required by the fault-table method.

TABLE 7.2.1 Construction of the Complete Test Set of the Circuit of Fig. 7.2.1 by Sensitizing Its Four Paths

Path Number	*Path*	*Input-Variable Values Required by Sensitizing the Path*	*Assigned Value of the Input Variable of the Path*	*Test*	*Faults Detected by the Test*
1	*afh*	$x_2 = 1$ and $x_3 = 0$ or 1	$x_1 = 1$	6 or 7	a_0, f_0, h_0
			$x_1 = 0$	2 or 3	a_1, f_1, h_1
2	*bdfh*	$x_1 = 1$ and $x_3 = 0$ or 1	$x_2 = 1$	6 or 7	b_0, d_0, f_0, h_0
		$x_1 = 1$ and $x_3 = 1$	$x_2 = 0$	5	b_1, d_1, f_1, h_1
3	*begh*	$x_1 = 0$ and $x_3 = 0$	$x_2 = 1$	2	b_0, e_0, g_1, h_1
		$x_1 = 0$ or 1 and $x_3 = 0$	$x_2 = 0$	0 or 4	b_1, e_1, g_0, h_0
4	*cgh*	$x_1 = 0$ or 1 and $x_2 = 0$	$x_3 = 1$	1 or 5	c_0, g_1, h_1
			$x_3 = 0$	0 or 4	c_1, g_0, h_0

An important question arises: Will this method always yield a test for detecting a fault? The answer is no. However, it always yields a test for a class of combinational circuits, known as tree-like circuits, which is defined as follows.

DEFINITION 7.2.4

A *tree-like circuit* is defined as a circuit in which (1) each input is an independent input line to the circuit and (2) the fan-out† of every gate is 1.

The circuits of Fig. 7.2.2(a) and (b) are examples of tree-like circuits. The reason that the path-sensitizing method always yields a test for detecting any single fault in a tree-like circuit is because that a consistent input combination always exists for any

†The definition of the fan-out of a gate was given on page 126.

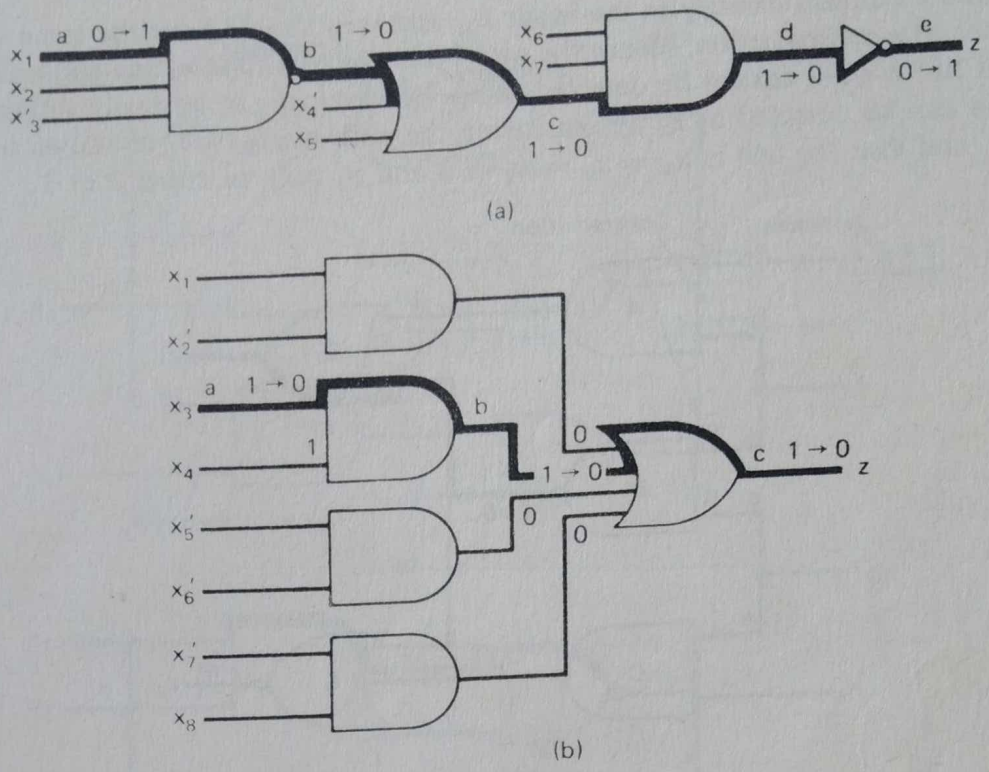


Fig. 7.2.2 Two tree-like circuits.

sensitized path in the circuit. For example, the faults a_1, b_0, c_0, d_0, and e_1 of the circuit of Fig. 7.2.2(a) can be detected by the test $x_1 = 0, x_2 = 1, x_3' = 1, x_4' = 0, x_5 = 0, x_6 = 1, x_7 = 1$, and the faults a_0, b_0, and c_0 of the circuit of Fig. 7.2.2(b) can be detected by the test $x_1 = x_2' = x_5' = x_6' = x_7' = x_8 = 0$ and $x_3 = x_4 = 1$.

From the above discussions, we have:

THEOREM 7.2.1

For any tree-like circuit, a complete test set can always be found by the path-sensitizing method.

Proof: Since every path of a tree-like circuit is sensitizable, we can find a complete test set for detecting any fault for every path. The union of the complete test sets for all the paths of the circuit is a complete test set for detecting any fault in the circuit. The determination of a minimal complete test set from this test set is similar to step 3 of the fault-table method for fault detection. ∎

For general combinational circuits, a contradiction of line value may occur at the gates, which are points of reconvergence for two or more fan-out paths from some preceding gate. For example, if we want to test whether the output of the NOR gate B of the circuit in Fig. 7.2.3(a) is s-a-0. In order to detect the s-a-0 fault, we require $h = 1$, which in turn requires $c = d = 0$. There are two paths from line h to the outputs z_1 and z_2, paths hjl and hkm. Let us first consider the path hjl. According to step 3, line g should be set to logic value 0, which requires $a = b = 1$. But both

lines b and c connect directly to the input x_2, and they should have the same value. This leads to a contradiction. Hence the path hjl is not sensitizable, and consequently the fault line h s-a-0 cannot be detected at z_1. However, it can be easily shown that this fault can be detected at z_2 by sensitizing the path along hkm, as shown in Fig. 7.2.3(b), and that the test is $x_2 = x_3 = x_4 = 0$ and x_1 may be either 0 or 1.

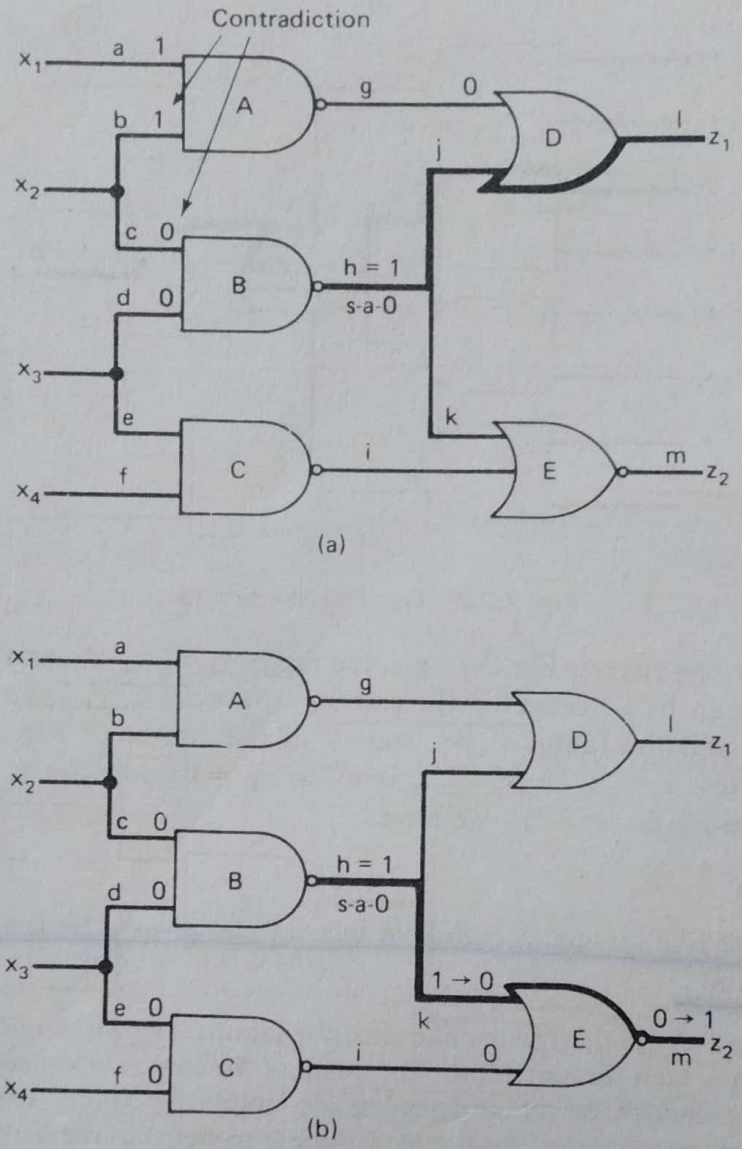


Fig. 7.2.3 Circuit of Example 7.2.2. (a) unsensitizable path hjl; (b) sensitizable path hkm.

It should be pointed out that there are cases where none of the individual paths of a circuit is sensitizable, but if we sensitize a group of them simultaneously, they become sensitizable. For example, none of the individual paths of the circuit of Fig. 7.2.4(a) from x_1 to z for detecting the fault line a s-a-0 is sensitizable, but if we sensitize *all the three paths* from x_1 to z as shown in Fig. 7.2.4(b) simultaneously, they become sensitizable.

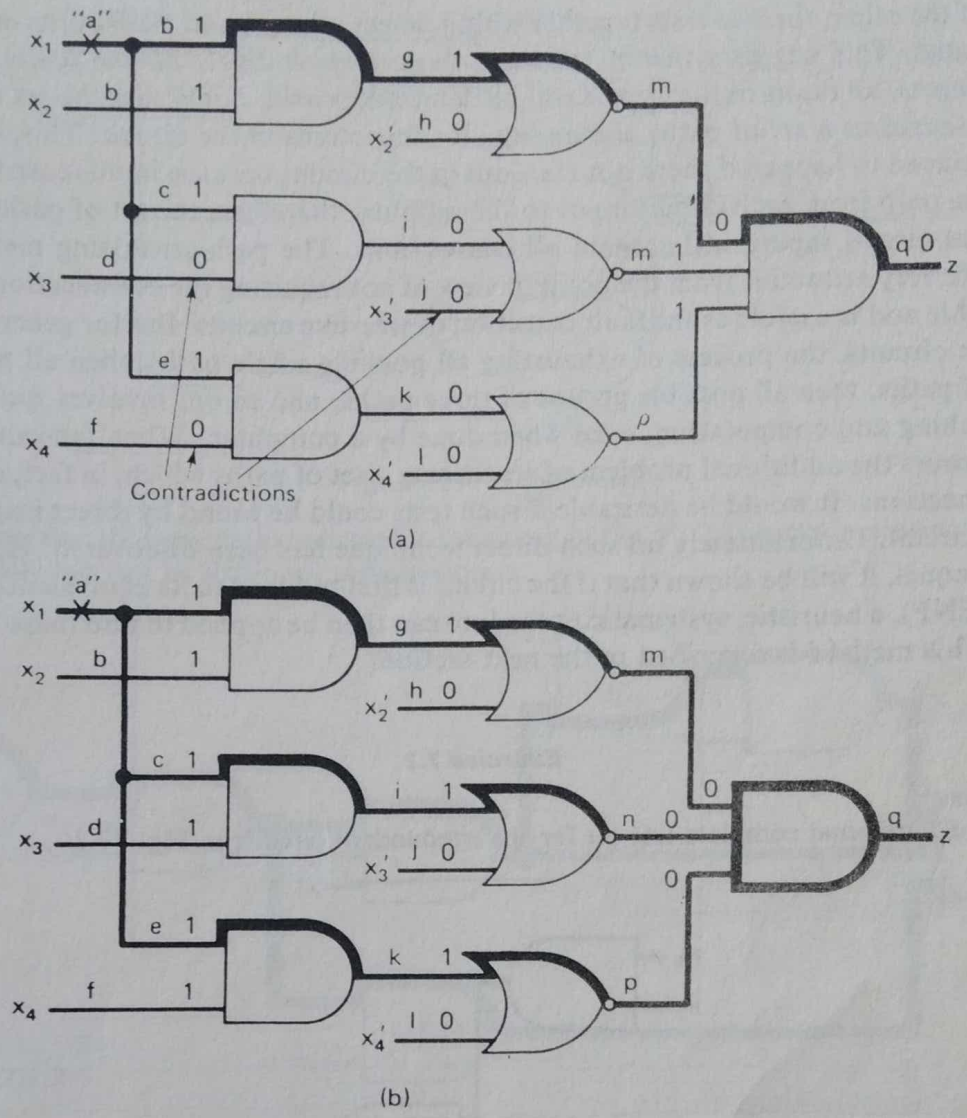


Fig. 7.2.4 Example of sensitizing several paths simultaneously.

This example shows that in searching out the sensitizable path for a particular fault, all possible combinations of single paths sensitized simultaneously must be included. We therefore add one additional step to the above procedure to make it complete.

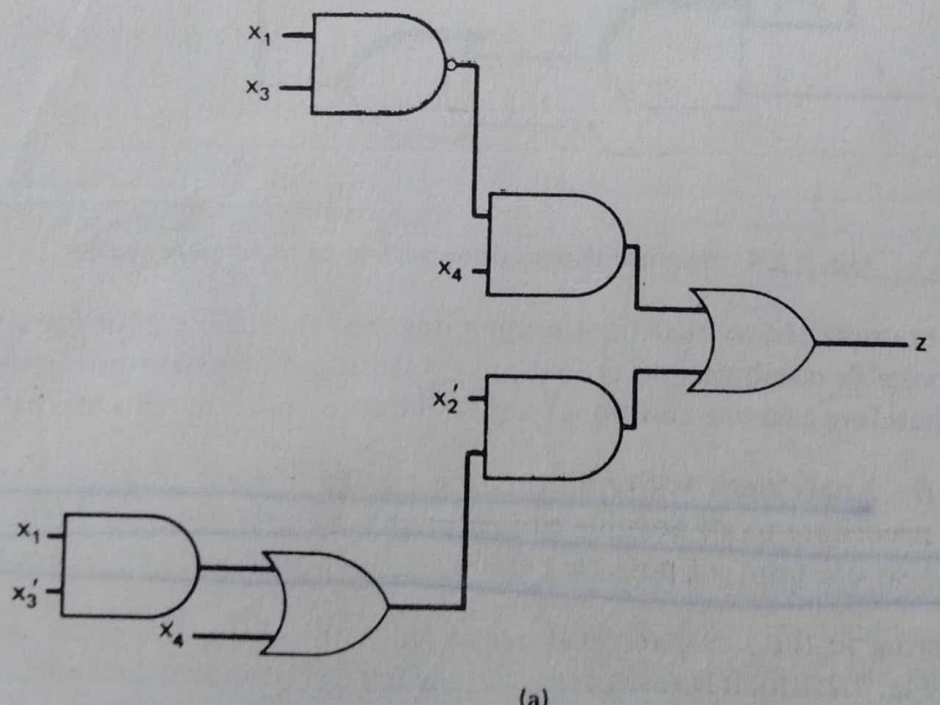
Step 5 Apply steps 1–4 to all the single paths. If none of them is sensitizable, apply the procedure to all possible pairs of paths, then to all possible groups of three paths, and so on, until all possible combinations of the single paths are tried.

Referring to the simultaneously sensitized paths *agmq*, *acinq*, and *aekpq* of the circuit of Fig. 7.2.4(b), it is easily seen that the test 1111 would detect a set of s-a-0 and s-a-1 faults on the lines along the paths, and the test 0111 would detect a set of s-a-0 and s-a-1 faults on the lines along the paths. Since the two tests detect the two sets of faults on the lines along the paths, and one is the complementary set of faults to the

faults of the other, the two tests together would detect *all* s-a-1 and s-a-0 faults on these three paths. This suggests that if tests are chosen *which detect all the faults on the circuit inputs*, all faults in the circuit will be detected, *provided* only that the set of tests chosen sensitizes a set of paths containing all connections in the circuit. This, in fact, is guaranteed to happen if there is no fan-out in the circuit, because in this case there is only one path from each circuit input to the outputs; therefore, the set of paths originating at circuit inputs will contain all connections. The path-sensitizing method is therefore very attractive from the point of view of not requiring the construction of the fault-table and is useful for the fault detection of tree-like circuits. But for general non-tree-like circuits, the process of exhausting all possible single paths, then all possible pairs of paths, then all possible groups of three paths, and so on, involves quite a lot of searching and computation, even when done by a computer. When fan-out exists, there occurs the additional problem of sensitizing a set of paths which, in fact, contain all connections. It would be desirable if such tests could be found by direct inspection of the circuit. Unfortunately no such direct technique has been discovered. However, in the sequel, it will be shown that if the circuit is first reduced to its equivalent normal form (ENF), a heuristic, systematic procedure can then be applied to find these "good" tests. This method is described in the next section.

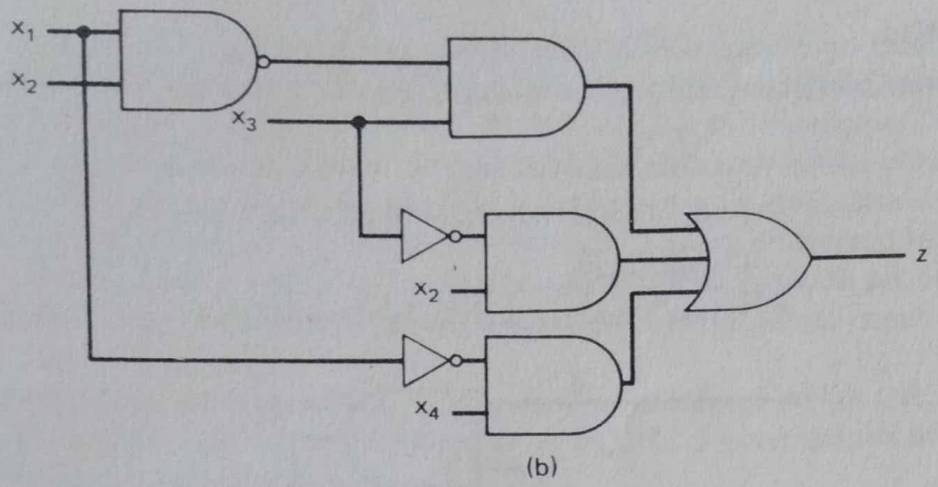
Exercise 7.2

1. Find a minimal complete test set for the irredundant circuits in Fig. P7.2.1.



(a)

Fig. P7.2.1



(b)

Fig. P7.2.1 (continued)

2. Show that the three paths indicated in the circuit of Fig. P7.2.2 cannot be sensitized individually but can be sensitized simultaneously.

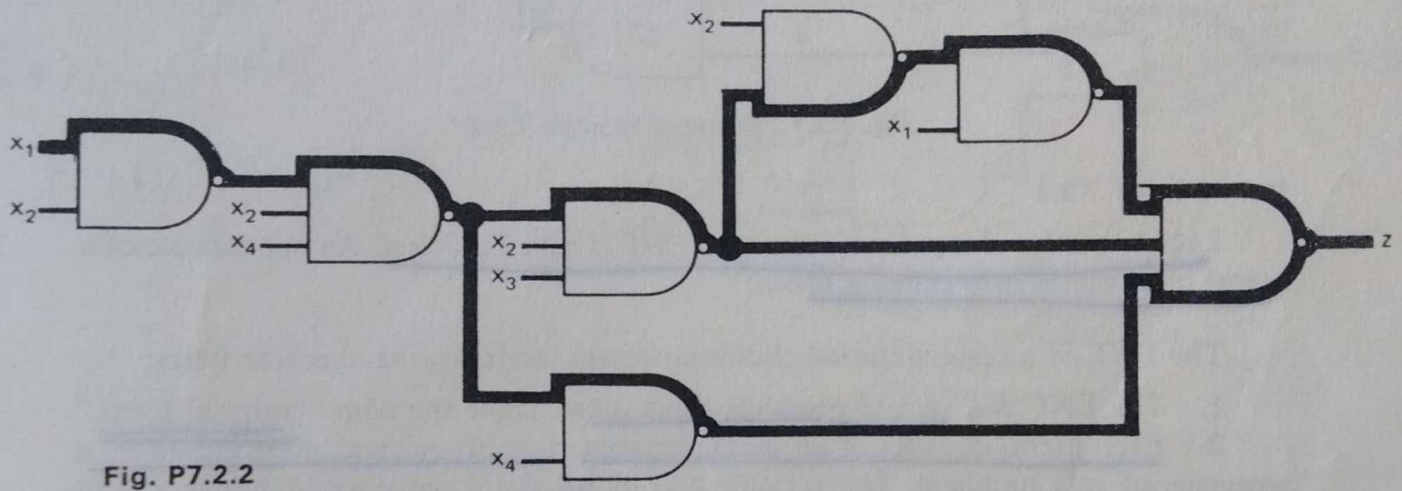


Fig. P7.2.2

7.3 Equivalent-Normal-Form Method

The equivalent normal form of a combinational circuit is defined below.

DEFINITION 7.3.1

The *equivalent normal form* (ENF) of a circuit is obtained by expressing the output of each gate as a sum-of-products expression of its inputs and preserving the identity of each gate by a suitable subscript.

Example 7.3.1

The ENF of the circuit of Fig. 7.2.1 is

$$\begin{aligned} h = (f+g)_h &= (ad)_{fh} + (e'c')_{gh} \\ &= (x_1)_{afh}(x_2)_{bdfh} + (x_2')_{begh}(x_3')_{cgh} \end{aligned} \tag{7.3.1}$$

Example 7.3.2

As another example, consider the circuit of Fig. 7.3.1. The ENF for the circuit is

$$h = (x_1)_{acfh}(x_1')_{acgh}(x_2')_{bcgh}(x_3)_{dfh} + (x_1')_{acgh}(x_2')_{bcgh} \cdot (x_2)_{bcfh}(x_3)_{dfh} + (x_1)_{acfh}(x_3)_{dfh}(x_4')_{egh} + (x_2)_{bcfh} \cdot (x_3)_{dfh}(x_4')_{egh} \tag{7.3.2.a}$$

$$= (x_1)_{acfh}(x_3)_{dfh}(x_4')_{egh} + (x_2)_{bcfh}(x_3)_{dfh}(x_4')_{egh} \tag{7.3.2}$$

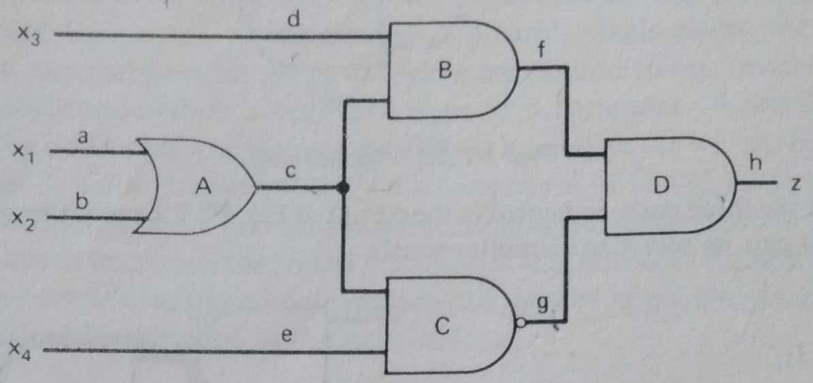


Fig. 7.3.1 Circuit of Example 7.3.2.

DEFINITION 7.3.2

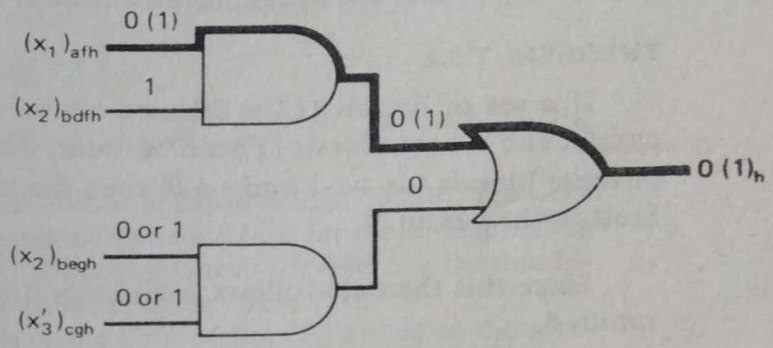
Each subscripted input variable in an ENF is called a *literal*. An appearance of a literal in a term is also called a *literal*.

The ENF of a combinational circuit possesses the following characteristics:

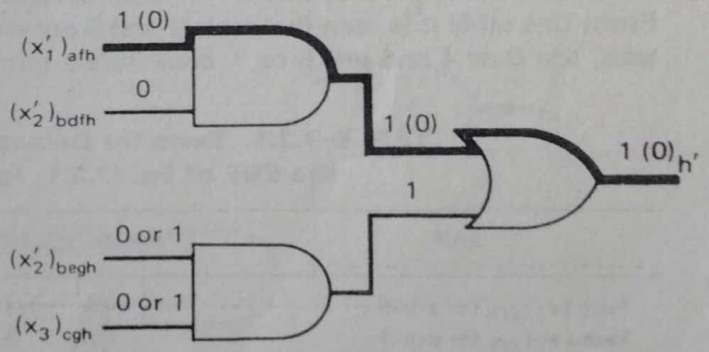
1. The ENF is a sum-of-products expression, hence the name "normal form."
2. Each literal consists of an input variable, possibly primed, subscripted by a sequence of gate numbers. The variable and its subscript sequence identifies a path from the corresponding circuit input to the output. For example, the literal $(x_1)_{afh}$ in the first term of Eq. (7.3.1) identifies the path from input x_1 through wires a, f, and h to the output of the circuit.
3. From the method of constructing the ENF, it follows that before any term of the ENF being discarded for the reason such as described in the next characterstic, there is at least one literal appearing in the ENF for every possible path from each input to the outputs.
4. Any term in the ENF may be discarded if it contains a variable and its complement. For example, the first two terms of Eq. (7.3.2a) may be discarded.
5. A literal may make several appearances in an ENF. For example, the literals $(x_3)_{dfh}$ and $(x_4')_{egh}$ appear in both terms of Eq. (7.3.2).
6. The ENF may contain redundant terms and literals even though the original circuit is irredundant. An example that illustrates this characteristic will be given later [see Eq. (7.3.3)].

7. If the path associated with a literal in the ENF contains an odd number of inversions (an inversion is produced by each NAND, NOR, and NOT element in the path), the priming on the literal will be opposite that on the corresponding input in the original circuit. If the number of inversions on the path is even, the priming on the literal will be the same as on the corresponding input in the original circuit. For example, the literals $(x_1)_{afh}$ and $(x_3')_{cgh}$ of the ENF of Eq. (7.3.1) are associated with the path *afh* with an even number (zero) of inversions and the path *cgh* with an odd number (one) of inversions, respectively. The two literals correspond to the input x_1 on gate *A* and the input x_3 on gate *B*.

8. Every ENF corresponds to a hypothetical two-level AND–OR equivalent circuit. For example, the ENF of the circuit of Fig. 7.2.1 corresponds to the hypothetical two-level equivalent circuits of Fig. 7.3.2.



(a) Hypothetical equivalent two-level AND-OR circuit of the circuit in Fig. 7.2.1 derived from the ENF.



(b) Hypothetical equivalent two-level OR-AND circuit of the complement of the circuit in Fig. 7.2.1 derived from the complemented ENF.

Fig. 7.3.2 Hypothetical equivalent circuits of Example 7.3.1.

According to the rules of path sensitizing and the eighth characteristic described above, to test a particular literal for s-a-1, that literal must be assigned value 0, while the remaining literals in the term are assigned value 1. Also, at least one literal in each remaining term must be 0, in order that all remaining inputs to the OR gate be 0. By similar reasoning, a test that assigns value 1 to all literals of a particular term and assigns at least one 0 to literal in each remaining term tests all literals in that particular term for s-a-0. These rules will be referred to as the *literal sensitizing rules*.

THEOREM 7.3.1

A test devised for a literal appearance in the ENF sensitizes the path in the original circuit associated with that literal appearance.

Proof: This theorem may be proved by mathematical induction. The complete proof is quite lengthy and will not be included here. It can be found in reference 7. ∎

THEOREM 7.3.2

If a set of literals $\{L\}$ is selected whose paths contain every connection in the circuit, and if a set of tests $\{T\}$ can be found which tests at least one appearance of each of these literals for s-a-1 and s-a-0, then the set of tests detects every s-a-1 and s-a-0 fault in the circuit.

Since this theorem follows sufficiently directly from Theorem 7.3.1, the proof is omitted.

Example 7.3.1 (Continued)

The four literals $(x_1)_{afh'}$, $(x_2)_{bdfh'}$, $(x'_2)_{begh'}$, and $(x'_3)_{cgh}$ identifying the four paths in the circuit of Fig. 7.2.1 can be tests for s-a-0 and s-a-1, which are shown in Table 7.3.1. Since the paths associated with these four literals containing all the connections of the circuit and the set of tests of the rightmost column of Table 7.3.1 tests each of the four literals for s-a-0 and s-a-1, by Theorem 7.3.2, this set of tests detects every s-a-0 and s-a-1 fault in the circuit. From this table it is seen that tests 2 and 5 are essential, and these two tests plus two other tests, test 0 or 4 and test 6 or 7, constitute a minimal complete test set for detecting every

TABLE 7.3.1 Tests for Detecting the Four Literals of the ENF of Eq. (7.3.1) for s-a-0 and s-a-1

ENF	$h = (x_1)_{afh}(x_2)_{bdfh} + (x'_2)_{begh}(x'_3)_{cgh}$				*Tests*
Test $(x_1)_{afh}$ for s-a-0	1	1	0	d	6 or 7
Test $(x_1)_{afh}$ for s-a-1	0	1	0	d	2 or 3
Test $(x_2)_{bdfh}$ for s-a-0	1	1	0	d	6 or 7
Test $(x_2)_{bdfh}$ for s-a-1	1	0	1	0	5
Test $(x'_2)_{begh}$ for s-a-0	d	0	1	1	0 or 4
Test $(x'_2)_{begh}$ for s-a-1	0	1	0	1	2
Test $(x'_3)_{cgh}$ for s-a-0	d	0	1	1	0 or 4
Test $(x'_3)_{cgh}$ for s-a-1	d	0	1	0	1 or 5

single s-a-0 and s-a-1 fault. It is left to the reader to show that the same sets of tests can be obtained by using the fault-table method, but the equivalent-normal-form method requires much less effort compared to the fault-table method.

DEFINITION 7.3.3

The *ENF of a multiple-output* circuit is defined to be the set of ENF's for the individual outputs of the circuit. These individual ENF's will be referred to as *sub-ENF's*.

From the above discussions, the following theorems are seen immediately.

THEOREM 7.3.3

A s-a-0 (s-a-1) test for a particular literal in the ENF of a single-output circuit (or sub-ENF of a multiple-output circuit) is a s-a-1 (s-a-0) test for the corresponding literal in the complemented ENF (or sub-ENF), denoted by ENF′.

Example 7.3.1 (Continued)

For example, the ENF′ of the circuit of Fig. 7.2.1 is

$$h' = (x_1')_{afh} + (x_2')_{bdfh}((x_2)_{begh} + (x_3)_{cgh} \tag{7.3.3}$$

whose hypothetical two-level OR–AND equivalent circuit is shown in Fig. 7.3.2(b). The s-a-0 and s-a-1 tests for literal $(x_1)_{afh}$ in the ENF are the s-a-1 and s-a-0 tests for the corresponding literal $(x_1')_{afh}$ in the ENF′, as shown in Fig. 7.3.3. The complete tests for detecting the four literals of the ENF′ of Eq. (7.3.3) for s-a-0 and s-a-1 are given in Table 7.3.2.

For brevity, replace the subscript sequences in the ENF of Eq. (7.3.1) and in the complemented ENF of Eq. (7.3.3) as follows: $afh \longrightarrow 1$, $bdfh \longrightarrow 2$, $begh \longrightarrow 3$, and $cgh \longrightarrow 4$. Denote the four paths *afh*, *bdfh*, *begh*, and *cgh* by P_1, P_2, P_3, and P_4, respectively.

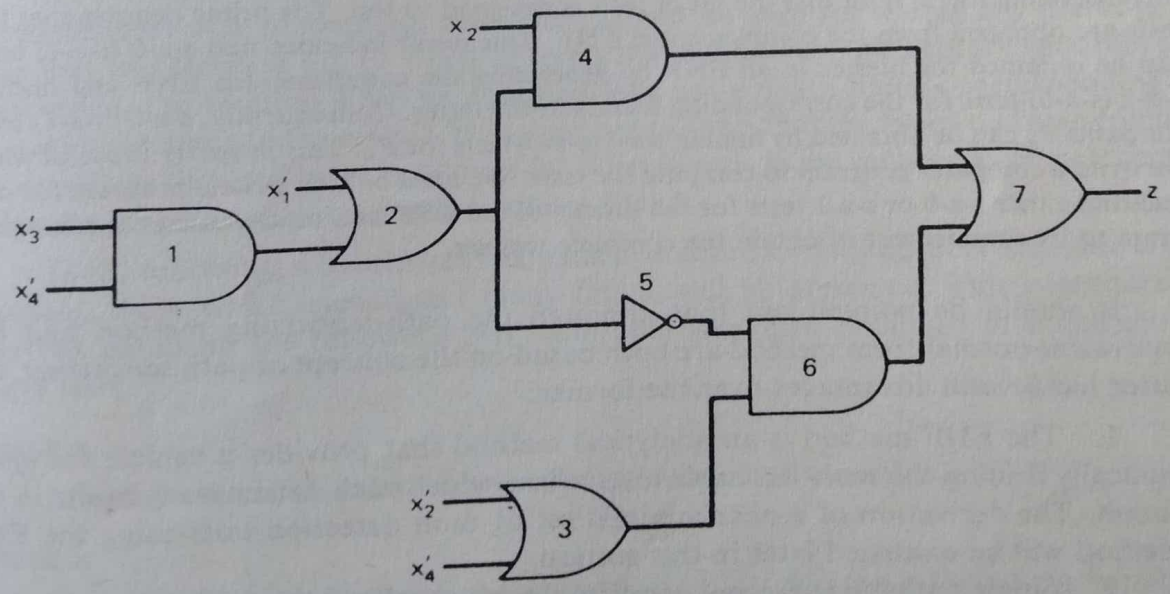


Fig. 7.3.3 Circuit of Example 7.3.2.

TABLE 7.3.2 Tests for Detecting the Four Literals of the ENF′ of Eq. (7.3.3) for s-a-0 and s-a-1

ENF′	$h' = ((x_1')_{afh} + (x_2)_{bdfh})((x_2)_{begh} + (x_3)_{cgh})$				*Tests*
Test $(x_1')_{afh}$ for s-a-0	1	1	0	d	2 or 3
Test $(x_1')_{afh}$ for s-a-1	0	1	0	d	6 or 7
Test $(x_2')_{bdfh}$ for s-a-0	1	1	0	d	5
Test $(x_2')_{bdfh}$ for s-a-1	1	0	1	0	6 or 7
Test $(x_2)_{begh}$ for s-a-0	d	0	1	1	2
Test $(x_2)_{begh}$ for s-a-1	0	1	0	1	0 or 4
Test $(x_3)_{cgh}$ for s-a-0	d	0	1	1	1 or 5
Test $(x_3)_{cgh}$ for s-a-1	d	0	1	0	0 or 4

From Tables 7.3.1 and 7.3.2, we obtain the following relation:

$$T(P_{10}) \longleftrightarrow T((x_1)_{afh,0}) = \{11d\} = \{6, 7\} = T((x_1')_{afh,1}) \longleftrightarrow T(P_{11}')$$
$$T(P_{20}) \longleftrightarrow T((x_2)_{bdfh,0}) = \{11d\} = \{6, 7\} = T((x_2')_{bdfh,1}) \longleftrightarrow T(P_{21}')$$
$$T(P_{30}) \longleftrightarrow T((x_2')_{begh,0}) = \{d00\} = \{0, 4\} = T((x_2)_{begh,1}) \longleftrightarrow T(P_{31}')$$
$$T(P_{40}) \longleftrightarrow T((x_3')_{cgh,0}) = \{d00\} = \{0, 4\} = T((x_3)_{cgh,1}) \longleftrightarrow T(P_{41}')$$
$$T(P_{11}) \longleftrightarrow T((x_1)_{afh,1}) = \{01d\} = \{2, 3\} = T((x_1')_{afh,0}) \longleftrightarrow T(P_{10}')$$
$$T(P_{21}) \longleftrightarrow T((x_2)_{bdfh,1}) = \{101\} = \{5\} = T((x_2')_{bdfh,0}) \longleftrightarrow T(P_{20}')$$
$$T(P_{31}) \longleftrightarrow T((x_2')_{begh,1}) = \{010\} = \{2\} = T((x_2)_{begh,0}) \longleftrightarrow T(P_{30}')$$
$$T(P_{41}) \longleftrightarrow T((x_3')_{cgh,1}) = \{d01\} = \{1, 5\} = T((x_3)_{cgh,0}) \longleftrightarrow T(P_{40}')$$

$T(\cdot)$ denotes the test set for testing $\cdot$. The second subscript of each path denotes the type of corresponding literal fault that the set of tests is designed to test. The prime denotes that the tests are obtained from the complemented ENF. This result indicates that s-a-0 (s-a-1) tests can be obtained for literals in an ENF by generating the complemented ENF and finding s-a-1 (s-a-0) tests for the corresponding literals in the latter. Consequently, s-a-0 (s-a-1) tests for paths P_k can be obtained by finding s-a-1 (s-a-0) tests for P_k'. This property is useful when we write a computer program to compute the tests. We need only to write a program for calculating either s-a-0 or s-a-1 tests for the literals of the ENF and repeat using the same program to its complement to obtain the complete test set.

It should be pointed out that although the path-sensitizing method and the equivalent-normal-form method are both based on the concept of path sensitizing, the latter has several advantages over the former:

1. The ENF method is an analytical method that provides a vehicle for systematically finding the most desirable tests, those which each detect many faults in the circuit. The derivation of a near-minimal set of fault detection tests using the ENF method will be discussed later in this section.
2. Single paths that are not sensitizable are usually easily seen from the ENF. For instance, when both a variable x and its complement x' are contained in the same

term of an ENF, it indicates that the paths represented by the corresponding two literals of x and x' cannot be sensitized. Thus these literals cannot be tested for either s-a-0 or s-a-1 fault. The reason for this is that due to reconvergent fan-out, no matter how we try to sensitize (i.e., to sensitize) one of the two paths or both paths simultaneously and what faults we sensitize, the s-a-0 or s-a-1 fault, a contradiction will always result. Another example is that when two or more literals pertaining to the same input variable are contained in the same term, the s-a-1 test for testing either of the literals individually is impossible.

3. The way to derive tests using the ENF is very simple.

Note that Theorem 7.3.2 does not guarantee finding a set $\{T\}$ which tests every literal in $\{L\}$ for s-a-1 and s-a-0. However, Armstrong (reference 7) has conjectured that such a set of tests can be found. More precisely, it is believed that the following result holds; 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. As yet this result has not been proved; however, efforts to find a counterexample have been unsuccessful.

Several important remarks about the equivalent-normal-form method should be made here.

1. The path-diagnostic test set for a path of a circuit obtained from the ENF or its complement should not be expected to be complete (problem 3).

2. Although Theorem 7.3.3 states that a s-a-0 (s-a-1) test for a particular literal in the ENF of a single-output circuit (or sub-ENF of a multioutput circuit) is a s-a-1 (s-a-0) test for the corresponding literal in the complemented ENF (or sub-ENF), not every test obtainable from one of two functions (the ENF and its complement) is obtainable from the other.

3. If what Armstrong has conjectured is true, certain terms in the ENF (or its complement) may be omitted, since they cannot be used for testing any literal [e.g., the first two terms in Eq. (7.3.2a)]. If terms that contain complementary variables are discarded, the reduced ENF so obtained may not contain literals corresponding to some connections in the circuit. It has not been shown that the faults on these connections will be detected by tests derived for other literals in the reduced ENF or its complement, but no counterexample has been found.

In the following, a heuristic, systematic procedure for finding most desirable tests, namely, those which each detect many faults, will be presented. This procedure is derived based on the following two main heuristic rules employed in constructing s-a-1 tests:

RULE 1

A term in the ENF exhibiting the fewest number of variables is chosen.

RULE 2

Then that literal in the term chosen by rule 1 such that the prime of the variable appearing in this literal has the most number of appearances in the remaining terms of

the ENF is chosen. Value 0 is then assigned to this literal and value 1 to the remaining literals in this term.

The first heuristic rule assures that when the selected term has these values assigned, as many variables as possible remain unassigned, thus giving maximum flexibility in assigning values in the remaining terms. The second heuristic rule results in many 1's being assigned in all terms, a desirable goal since the ideal condition is to have just a single 0 assigned in each term and the remaining literals assigned 1's. Together rules 1 and 2 result in constructing a test that simultaneously tests a literal in each of many terms for s-a-1.

In practice, the order of testing literals is determined by assigning a numerical score to each literal appearance. At any stage of formation of a test, variable values are assigned so as to test the as-yet untested literal appearance of highest score, provided no other appearance of this literal has already been tested. Otherwise, this literal appearance is passed over. The scoring function assigns appropriate weights to rules 1 and 2, which is

$$(Sc_1)_k = \left(1 - \frac{V_j}{V}\right) + \frac{\lambda_k}{L} \tag{7.3.4}$$

(provides for rule 1) (provides for rule 2)

where $(Sc_1)_k \equiv$ s-a-1 score for the kth literal in the ENF

$V_j \equiv$ number of variables exhibited in the jth term of the ENF, where this term contains the kth literal

$V \equiv$ total number of variables

$L \equiv$ total number of literal appearances in the ENF

$\lambda_k \equiv \begin{cases} Ai', \text{ if the } k\text{th literal is unprimed} \\ Ai, \text{ if the } k\text{th literal is primed} \end{cases}$ †

$Ai \equiv$ number of unprimed appearances of the ith variable in the ENF

$Ai' \equiv$ number of primed appearances of the ith variable in the ENF

The construction of s-a-0 tests requires somewhat different heuristics from the two used for s-a-1 tests, thus requiring a new scoring function. Theorem 7.3.3 states that s-a-0 tests can be obtained for literals in an ENF by generating the complemented ENF and finding s-a-1 tests for the corresponding literals in the latter. Therefore, both the ENF and the complemented ENF will be employed and only s-a-1 tests will be constructed, using the s-a-1 scoring function for both ENF's. During construction of tests, the next test to be constructed will start with the as-yet-untested literal appearance of highest score in either the ENF or the complemented ENF. The remainder of the construction of this test will be confined to the particular ENF in which it started.

†It is assumed that the kth literal represents an appearance of the ith variable.

To illustrate this procedure, the following example was given by Armstrong (reference 7).

Example 7.3.3

Consider the irredundant circuit of Fig. 7.3.3 whose ENF and ENF′ are

$$\begin{aligned} z = {} & (x_1')_{247}(x_2)_{47} + (x_2)_{47}(x_3')_{1247}(x_4)_{1247} + (x_1)_{2567}(x_3)_{12567}(x_2')_{367} \\ & + (x_1)_{2567}(x_3)_{12567}(x_4')_{367} + (x_1)_{2567}(x_4')_{12567}(x_2')_{367} \\ & + (x_1)_{2567}(x_4')_{12567}(x_4')_{367} \end{aligned} \tag{7.3.5}$$

and

$$\begin{aligned} z' = {} & {}_{2567}(x_2')_{47} + (x_1)_{247}(x_1')_{2567}(x_3)_{1247} + (x_1)_{247}(x_1')_{2567}(x_4')_{1247} \\ & + (x_2')_{47}(x_3')_{12567}(x_4)_{12567} + (x_1)_{247}(x_3)_{1247}(x_3')_{12567}(x_4)_{12567} \\ & + (x_1)_{247}(x_3')_{12567}(x_4')_{1247}(x_4)_{12567} + (x_2')_{47}(x_2)_{367}(x_4)_{367} \\ & + (x_1)_{247}(x_2)_{367}(x_3)_{1247}(x_4)_{367} + (x_1)_{247}(x_2)_{367}(x_4')_{1247}(x_4)_{367} \end{aligned} \tag{7.3.6}$$

Note that in the above expressions, instead of line labels, gate labels are used to describe the subscript sequence of a literal. We purposely choose to do so to show this alternative. Again for brevity, replace the subscript sequences in the ENF [Eq. (7.3.5)] and in the complemented ENF [Eq. (7.3.6)] as follows: $247 \longrightarrow \alpha$, $47 \longrightarrow \beta$, $1247 \longrightarrow \gamma$, $2567 \longrightarrow \delta$, $367 \longrightarrow \epsilon$, and $12567 \longrightarrow \lambda$. Two tables are constructed, Table 7.3.3(a) for the ENF and Table 7.3.3(b) for the complemented ENF. Row 1 in each table exhibits the relevant ENF. Only five of the nine terms comprising the complemented ENF appear in Table 7.3.3(b). The remaining four terms were eliminated because none of their literals are testable for s-a-1.

To illustrate the use of the scoring function, the score for a particular literal in the ENF of Table 7.3.3(a) is computed in the following table. First, the values of Ai and Ai' for each variable in the ENF are tabulated:

Variable	*Ai*	*Ai′*
x_1	4	1
x_2	2	2
x_3	2	1
x_4	1	4

For example, the first entry in column Ai indicates that there are four appearances of variable x_1 in the ENF. These appearances are in the literal $x_{1\delta}$, which occurs in each of the last four terms. In this example the number of variables V is 4. The total number of literal appearances L in the ENF is 17.

To compute the score of the literal $x'_{1\alpha}$ in the first term of the ENF in Table 7.3.3(a), observe that the number of variables exhibited in this term is 2, and λ_α for

TABLE 7.3.3

(a) Tests for ENF

ENF	*1* $x_{1x'}$	*4* x_{1x}	✓ x_{1x}	*1* $x_{1x'}$	× x_{4x}	✓ x_{1x}	× $x_{1x'}$	*8* $x_{4x'}$	✓ x_{1x}	*1* $x_{2x'}$	*8* x_{2x}	✓ x_{1x}	✓ x_{2x}	*1* $x_{4x'}$	*4* x_{1x}	× $x_{4x'}$	× x_{4x}
Sa_1	200		100		132	84		84	84		84	84		84	152	—	—
		168		100			100			100			84				
Ordering of literals	1	2	5	6	4	9	7	10	11	8	12	13	14	15	3	—	—
Test 1	0	1	1	0	1	1	0	0	1	0	1	1	1	0	1	0	0
Test 4	1	0	0	0	0	0	1	1	0	1	1	0	1	1	0	1	1
Test 8	0	0	0	1	1	1	1	0	1	1	0	1	0	0	1	0	0

(b) Tests for Complemented ENF

Complemented ENF	*3* $x_{1x'}$	*2* $x_{1x'}$	✓ $x_{1x'}$	*3* $x_{1x'}$	*7* x_{4x}	✓ $x_{1x'}$	*3* x_{2x}	× x_{4x}	*2* x_{1x}	✓ x_{2x}	*5* x_{1x}	✓ x_{4x}	× x_{1x}	× x_{2x}	× $x_{4x'}$	*6* x_{4x}
Sa_1	170		119		85	187		—	17		17		—	—	136	
		187		85			187			51		17				85
Ordering of literals	4	1	6	7	8	2	3	—	11	10	12	13	—	—	5	9
Test 2	1	0	0	0	1	0	1	1	0	1	1	1	0	1	0	1
Test 3	0	1	1	0	1	1	0	1	1	0	1	1	1	0	0	1
Test 5	0	0	0	1	1	0	1	1	1	1	0	1	1	1	0	1
Test 6	0	0	0	0	0	0	1	0	1	1	1	0	1	1	1	0
Test 7	0	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0

this literal is 4. Therefore, the score for this literal is

$$Sc_1 = \left(1 - \frac{2}{4}\right) + \frac{4}{17} = \frac{50}{68} = \frac{200}{272}$$

All literal scores in Table 7.3.3(a) and (b) have been brought to the common denominator 272, and this denominator is omitted throughout.

The third row gives an ordering of the literal appearances within the particular table according to their scores. The ordering is arbitrary in case of tie scores. Henceforth, any literal appearance is referred to by the number assigned in this row. The remaining rows contain the binary values assigned to each literal by s-a-1 tests generated by the algorithm. Derivation of the first two tests is now described.

First, the literal appearance of highest score in either table is selected. This is literal 1 of Table 7.3.3(a); therefore, the first test will be constructed entirely within Table 7.3.3(a). Literal 1 is assigned value 0, and the remaining literal in this term is assigned value 1. This assigns value 1 to variable x_1 and x_2. Next, assign all other appearances of these variables their appropriate values in Table 7.3.3(a). Now check to see if the values of additional variables are forced to prevent the test from aborting. A test aborts if the variable assignment produces one of more terms of all 1's. Variable x_4 falls in this category. It must be assigned 1, because if it is assigned 0, all literals in the last term will be 1. It is now apparent that variable x_3 must be assigned 1; otherwise, all literals in the second term will be 1, again aborting the test. The test is now complete, because all variables have been assigned.

Above those literal appearances in Table 7.3.3(a) which were tested by this test, the number of the test is written. Four distinct literals are seen to have been tested. A check mark is also placed above all other appearances in the ENF of these literals, since only one appearance of each need be tested.

To start the second test, the highest-scoring literal appearance in Table 7.3.3(a) or (b), which does not represent an already-tested literal, is chosen. In this case it is literal 1 of Table 7.3.3(b), so the second test is constructed entirely in Table 7.3.3(b). Literal 1 is assigned value 0, and the other literal in this term is assigned 1. This assigns values 0 and 1 to variables x_1 and x_2, respectively. Next, assign all other appearances of these variables their appropriate values in Table 7.3.3(b). It is now observed that no further variables need be assigned at this time solely for the purpose of avoiding aborting the test. Therefore, the as-yet-untested literal appearance of highest score, which is literal 2 appearing in the third term, is sought. Since another appearance of this literal has already been tested by this test, it is passed over. Further search indicates that only one other literal can be tested by the current test, literal 11 in the fourth term. To test it, variables x_3 and x_4 must both be assigned value 1, completing the construction of this test. The literals tested are $x_{1\alpha}$ and $x'_{2\beta}$. If this test is applied to the ENF, it will be seen that it tests literals $x'_{1\alpha}$ and $x_{2\beta}$ in the first term of the latter for s-a-0, as expected.

Proceeding as previously stated, eight tests, which test every distinct, testable literal in both the ENF and complemented ENF for s-a-1, were constructed. [Literals

having a × above them in Table 7.3.3(a) and (b) are untestable.] It is found, by constructing and reducing the complete fault table for this circuit, that six of these eight tests form a minimal fault-detection set. Two such minimal test sets are {1, 3, 4, 5, 6, 7} and {1, 2, 3, 6, 7, 8}. The underlined tests are essential; they must be included since they each detect faults not detected by any other test.

Exercise 7.3

1. Show that the test set described in Table P7.3.1 obtained from Eq. (7.3.2) will detect all the detectable single faults along the four paths of the circuit of Fig. 7.3.1.

TABLE P7.3.1 Test Sets for Detecting the s-a-0 and s-a-1 Faults along the Paths Represented by the Literals in Eq. (7.3.2)

Literal	*Path Represented by the Literal*	*Test Sets That Detect s-a-0 and s-a-1 Faults on the Path*
$(x_1)_{acfh}$	*acfh*	{1010, 0010}
$(x_2)_{bcfh}$	*bcfh*	{0110, 0010}
$(x_3)_{dfh}$	*dfh*	{1010, 1*d*00} or {0110, *d*100}
$(x_4')_{egh}$	*egh*	{1010, 1*d*11} or {0110, *d*111}

2. (a) Find the ENF of the circuit of Fig. 7.2.4.
   (b) Show that the test set {3, 5, 6, 7, 15} obtained from this ENF will detect all the detectable single faults in the circuit.
3. (a) Show that the path-diagnostic test for the path abcd of the circuit in Fig. P7.3.3 obtained from the ENF is {1010, 1110}.
   (b) Show that there are other tests which can be used to detect faults occurring along the path abcd.

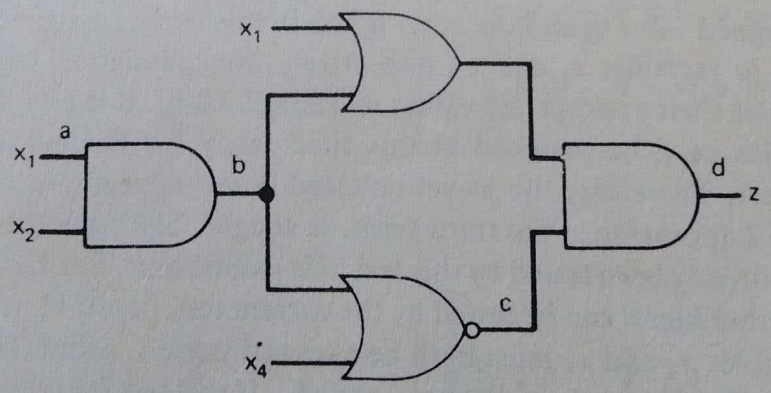


Fig. P7.3.3

4. Using the scoring function, find a near-minimal complete test set for the circuit in Fig. P7.3.4 using the ENF method. Note that only the faults on the labeled lines are considered.

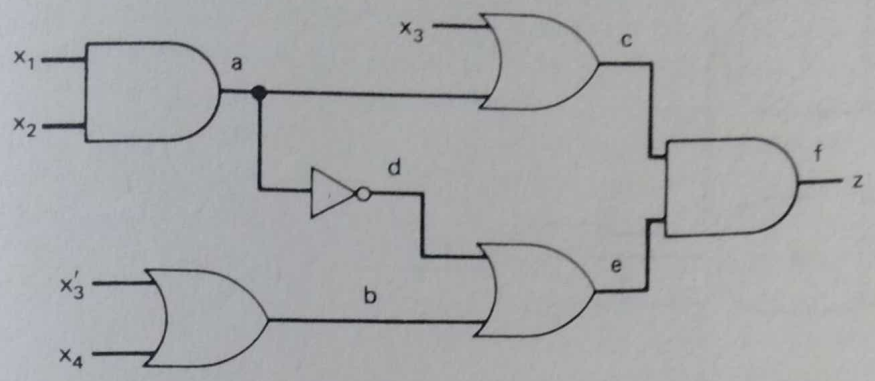


Fig. P7.3.4

7.4 Two-Level-Circuit Fault Detection

So far we have discussed methods of construction of a complete fault-detection test set for a combinational circuit using the two basic approaches: (1) examining each "individual fault" (the fault-table method) and (2) examining each "path" (the path-sensitizing method and the ENF method). In this section, a third approach to the problem is introduced. Instead of examining each individual fault or each path, it is proposed to examine each gate of the circuit. A very simple and direct method for constructing a minimal complete fault-detection test set for any two-level AND–OR (OR–AND, NAND–OR, etc.) irredundant circuit using this approach is presented. It utilizes certain basic properties of the prime implicants of a minimized function when displayed on the Karnaugh map. By using these properties, not only s-a-0 and s-a-1 tests for each gate can be directly "read out" from the map, but the information about the effect of the variable perturbation on the output (i.e., whether the output changes from a 1 to a zero or a zero to a 1) is also provided. In practice, it is necessary to know what the output should be for a specific test so that the result of the applied test can be monitored at the circuit output.

This method may be considered to have two versions: a graphical and a tabular. The graphical version will first be presented which uses the Karnaugh map, hence is convenient to apply to circuits with a small number of input variables, say no more than six, but preferably no more than four. Then, just as what was done in the minimization of switching functions (Sections 1.4–1.7), this Karnaugh map version is extended to a tabular method in a similar manner as that of Quine–McCluskey (Section 1.5). It uses exactly the same principles but without maps, allows the circuit to have any finite number of input variables, and hence is particularly attractive from a machine-computation point of view. First, we present the graphical version of this method to two-level AND–OR circuits with input variables no greater than four.

Karnaugh Map Method Consider the irredundant two-level AND–OR circuit shown in Fig. 7.4.1. Since it is assumed to be an irredundant circuit, the algebraic

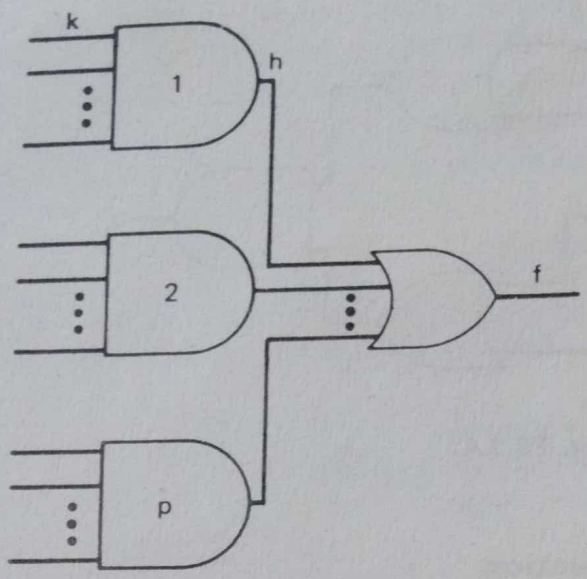


Fig. 7.4.1 Gereral two-level AND-OR circuit.

expression describing the circuit is an irredundant sum of prime implicants. In fact, each AND gate realizes one prime implicant,

$$F = f(X) = \Pi_1(X) + \Pi_2(X) + \ldots + \Pi_p(X) \qquad (7.4.1)$$

where $X = (x_1, x_2, \ldots, x_n)$ and $\Pi_i(X)$, $i = 1, 2, \ldots, p$, are p prime implicants. The subscripts $1, 2, \ldots, p$ denote p AND gates. In analyzing the possible faults on the internal wires connecting the AND gates with the OR gate, it is evident that each such fault has an equivalent fault (or faults) on the inputs of the circuit (i.e., the inputs of the AND gates). For example, an s-a-0 fault on wire h is equivalent to an s-a-0 fault on one or more of the inputs of gate 1. Consequently, an experiment designed to detect all faults on the external inputs of the two-level AND–OR circuit will detect all the faults within the circuit as well.

To detect an s-a-0 fault on wire k at the input to gate 1, it is necessary to assign the value 1 to the variable associated with this input and apply 1's to all the remaining inputs of gate 1. The yet-unassigned variables must be assigned values in such a way that the outputs of all the other AND gates will be 0. Then, if the circuit output is 1, no s-a-0 fault exist on wire k. However, if the output is 0, the gate is faulty, because one (or more) of its inputs or its output is s-a-0. Clearly, such a set of values assigned to the variables will detect s-a-0 faults at *any* input of gate 1. Consequently, it constitutes an s-a-0 test for gate 1.

An s-a-1 fault in a gate input is detected in a similar manner. The input in question is assigned the value 0, while all other inputs to that AND gate are assigned 1's. The remaining variables are again assigned values in such a way that the output of each AND gate will be 0. Then, if the circuit output is 0, no s-a-1 fault exists on the wire being tested. But if the output is 1, the circuit has an s-a-1 fault. The identification of the particular faulty wire, however, is not always possible without additional tests, since any of the AND gate outputs may be s-a-1, not necessarily the AND gate in question. This point will be discussed further.

An s-a-0 fault in an input of an AND gate causes the output of this gate to be s-a-0, regardless of the values of the remaining variables. *Such a fault in effect eliminates one prime implicant from the function realized by the circuit.* Thus, to check whether a given prime implicant has completely "vanished," it is sufficient to test one minterm which is covered by that prime implicant and by no other prime implicant, and to verify that this minterm is indeed realized by the circuit. (Clearly, if the prime implicant is not redundant, such a minterm always exists.) The requirement that the minterm chosen for testing must be one that is not covered by more than one prime implicant is essential, since a minterm covered by two (or more) prime implicants is realized by two (or more) AND gates, and the failure of one gate will not be detected if at least one of the remaining gates is operating correctly. From a path-sensitizing point of view, this is to say that each path from a primary input to the circuit output must be sensitized with a test condition that does not simultaneously sensitize another path. *A single unique minterm from a prime implicant will sufficiently describe an s-a-0 test for the AND gate that realizes the prime implicant.* If the output of the block is s-a-0 when the test is applied, the output of the circuit will immediately go to zero since, in effect, that prime implicant has now been eliminated from the circuit. The same is also true if any of the input wires (literals) to the AND gate should fail to a zero. Thus a complete set of tests for s-a-0 faults for a two-level AND–OR circuit consists of p tests, corresponding to the p prime implicants in f. This set of tests can be easily determined from the Karnaugh map of the function. For example, consider

$$f(x_1, x_2, x_3, x_4) = x_2x_4 + x_3'x_4 + x_1x_2x_3 \tag{7.4.2}$$

whose two-level AND–OR realization is shown in Fig. 7.4.2(a) and whose Karnaugh-map representation is shown in Fig. 7.4.2(b). The three subcubes describing the three prime implicants that are realized by the AND gates are indicated in the map.

DEFINITION 7.4.1

A *true subcube* is a subcube in the map describing a prime implicant for which the function $f = 1$.

For example, the three true subcubes of the function of Eq. (7.4.2) are (1, 5, 9, 13), (5, 7, 13, 15), and (14, 15). To test these three true subcubes for s-a-0 faults we select three minterms, one for each subcube, such that each minterm is contained in *only one* true subcube. If more than one minterm is exclusively contained in a true subcube, any one of them may be chosen to test the s-a-0 fault of the AND gate that realizes this true subcube (prime implicant). From Fig. 7.4.2(b) we see that the minimal complete test set T_0 for detecting all s-a-0 faults is therefore $T_0 = \{7, 1 \text{ or } 9, 14\}$. It should be noted that since the circuit is assumed as having no redundancy (i.e., the function that it realizes is minimal), any prime implicant of the function has at least one minterm that is not contained in any other true subcubes. Therefore, *such an s-a-0 test set always exists.*

Now, consider the effect on the circuit output of an s-a-1 fault in one of the AND-gate inputs. The output of the faulty AND gate will be independent of the variable

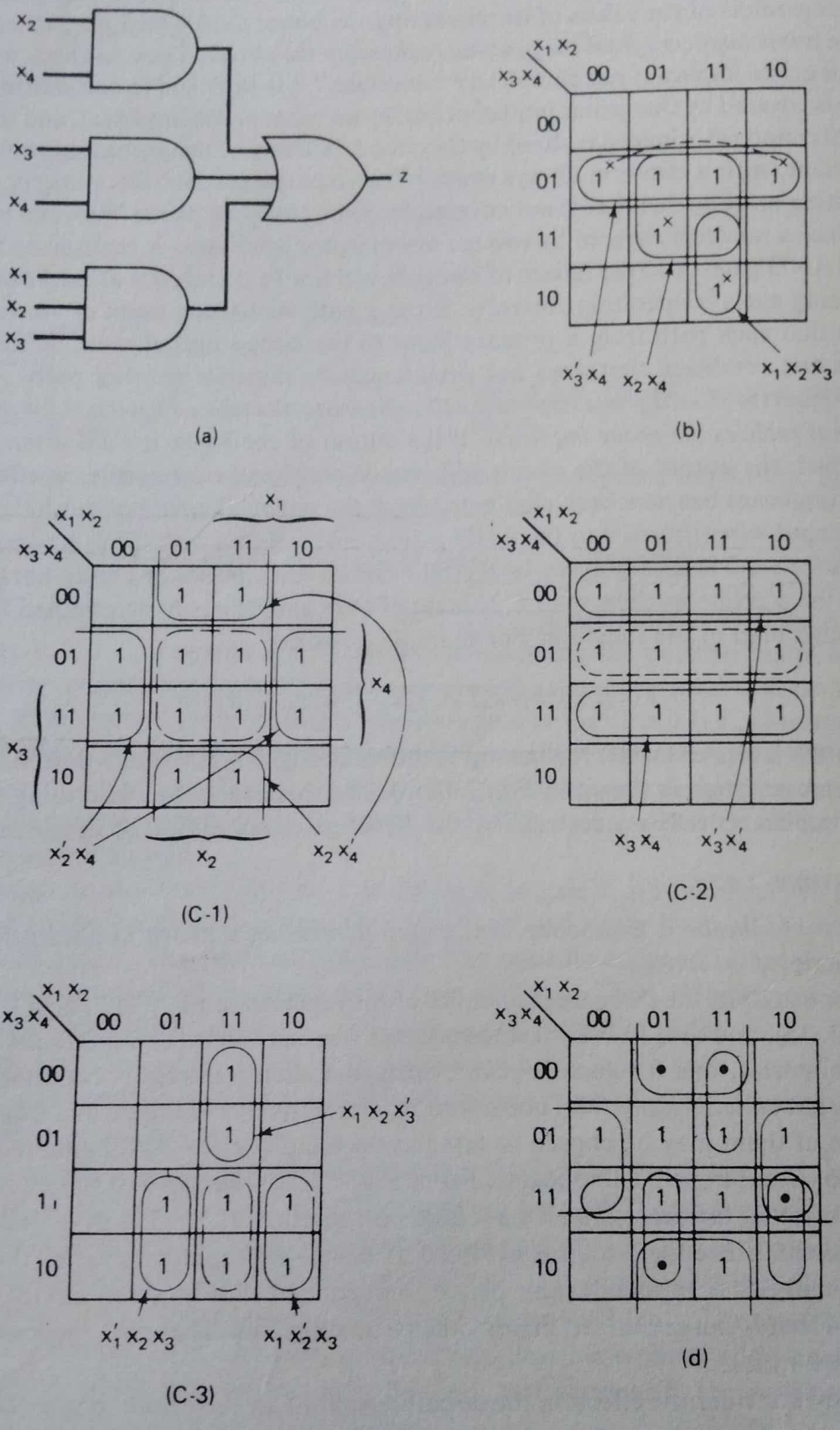


Fig. 7.4.2 Two-level AND-OR circuit for illustrating the true subcubes and adjacent subcubes associated with a circuit.

associated with the s-a-1 input. Consequently, the gate will realize a product that covers the original prime implicant and is independent of one of its variables.

DEFINITION 7.4.1(a)

Two product terms of the same variables are said to be *adjacent* if they differ in the value of just one variable.

For example, the adjacent product terms of the prime implicants of the function Eq. (7.4.2) are:

Prime Implicants	*Adjacent Product Terms of the Prime Implicant*
x_2x_4	$x_2'x_4$ and x_2x_4'
$x_3'x_4$	x_3x_4 and $x_3'x_4'$
$x_1x_2x_3$	$x_1'x_2x_3$, $x_1x_2'x_3$, and $x_1x_2x_3'$

Suppose that input x_2 in gate 1 is s-a-1; the output of that gate will depend only on input x_4. The output of gate 1 under the fault s-a-1 on the x_2 input can be represented by the expression

$$1 \cdot x_4 = (x_2 + x_2')x_4 = x_2x_4 + \underbrace{x_2'x_4}_{\text{an additional term}}$$

This indicates that an s-a-1 fault on the x_2 input is equivalent to that an additional term $x_2'x_4$, that is adjacent to x_2x_4, is realized by gate 1. Since any of the inputs to gate 1 may be s-a-1, a test must be made to verify that none of the subcubes adjacent to x_2x_4 (i.e., $x_2'x_4$ and x_2x_4') is realized by gate 1. Similarly, to test whether gates 2 and 3 have any s-a-1 faults, we must verify that they do not realize any (additional) product terms adjacent to the prime implicants that they realize.

DEFINITION 7.4.1(b)

Two subcubes are said to be *adjacent* if their corresponding product terms are functions of the same variables and differ in the value of just one variable.

For example, the adjacent product terms of the three prime implicants are shown in Fig. 7.4.2(c). Figure 7.4.2(d) shows all the adjacent subcubes to the three prime implicants of the function $f = x_2x_4 + x_3'x_4 + x_1x_2x_3$. In this map, only the 1's of the true subcubes of the function are shown. This map offers a convenient way to select the cells (tests) to test all the adjacent subcubes. A set of s-a-1 tests should be selected with the following properties:

1. Each adjacent subcube must be tested in at least one of its cells.
2. The value of f in the cell selected for testing the adjacent subcube must be zero.
3. It is allowed (and generally desirable) to select cells common to a number of adjacent subcubes.

The reason that each adjacent subcube must be tested in at least one of its cells is that every input of every AND gate must be checked to see whether it is s-a-1, and the

reason for having the third property is simply for minimizing the number of tests. The need for the second property demands an explanation. Note that an s-a-1 fault is detected by a 1 at the circuit output. If some other AND gate is set to 1, the circuit output will be 1, regardless of the fault at the gate being tested. For example, if we choose input combination 13 (i.e., 1101) to detect the s-a-1 of input x_3 of gate 3 [this is equivalent to saying that we choose an input combination to test whether gate 3, in addition to realizing the prime implicant $x_1x_2x_3$, also realizes its adjacent product term $x_1x_2x_3'$ represented by the adjacent subcube (12, 13) shown in Fig. 7.4.2(c–3), for which the circuit output will be 1], then there is no way to tell whether input x_3 is s-a-1. If, on the other hand, input combination 12 is selected, then the s-a-1 fault on input x_3 of gate 3 will be detected, since the circuit is 0 if the fault is not present and is 1 if the fault is present. It is obvious that *if the circuit is irredundant, it is always possible to find in each subcube adjacent to a true subcube at least one cell for which the value of f is zero.*

The problem now is to select a set of s-a-1 tests such that each adjacent subcube is checked at least once and the total number of tests is minimal.

DEFINITION 7.4.2

An input combination is said to be an *essential* test if it is the only input combination that can be used to check an adjacent subcube.

For example, the combination 6 and 12 are essential, since they are the only tests that can be used to test the subcubes (6, 7) and (12, 13), respectively; whereas the rest of the subcubes can be tested by at least two input combinations. For example, the subcube (1, 3, 9, 11) can be tested by either combination 3 or 11. Thus combinations 3 and 11 are not essential. It is clear that input combinations 6 and 12 must be selected in our s-a-1 test set. Referring to Fig. 7.4.2, it is seen that combinations 6 and 12 test not only subcubes (6, 7) and (12, 13), but also two other subcubes, (4, 6, 12, 14) and (0, 4, 8, 12). The subcubes left to be tested are (1, 3, 9, 11), (3, 7, 11, 15), and (10, 11). Based on the consideration that the total number of tests be as small possible, combination 11 should be chosen, since it is contained in all three subcubes. The set of tests composed of combinations 6, 11, and 12 constitute the s-a-1 test set T_1 (i.e., $T_1 = \{6, 11, 12\}$). Hence the minimal complete test set T is $T = T_0 \cup T_1 = \{7, 1 \text{ or } 9, 14, 6, 11, 12\}$, which will detect any number of s-a-0 and s-a-1 faults within the circuit.

Several final remarks are made here.

1. *The upper bound of the length of the experiment.* Suppose a two-level AND–OR irredundant circuit that realizes a function having p prime implicants. Let $q_1, q_2, \ldots, q_k$ be the numbers of the literals of the p prime implicants. A minimal complete test set can always be found by the method described in this section, and the upper bound of the length of the experiment $l_{u.b.}$ using the method described above is

$$l_{u.b.} = p + \sum_{i=1}^{p} q_i$$

2. An s-a-0 fault is detected by a 0 at the circuit output (i.e., for detecting any s-a-0 fault in a two-level AND–OR circuit by a test obtained by the method described above, the circuit output is 1 if the s-a-0 fault under detection is not present and is 0 if the fault is present), and an s-a-1 fault is detected by a 1 at the circuit output (i.e., for detecting any s-a-1 fault by a test obtained by the method, the circuit output is 0 if the s-a-1 fault under detection is not present and is 1 if the fault is present).

3. Although this method was demonstrated for the case of AND–OR logic, it can be easily extended to OR–AND logic.

Tabular Method The procedure just outlined for the determination of the set of tests in a fault-detection experiment suffers from the limitation inherent in any operation with a map. It is very useful for circuits with a small number of variables, and becomes complicated when the number of variables increases. To overcome this difficulty, the tabular method can be used, which is described as follows:

Step 1 Determination of a minimal complete s-a-0 test set T_0. From the given input implicants, find all the combinations for which each AND gate output is 1. Then find those combinations that are *not* common to two or more gates. Let $S_1, S_2, \ldots, S_p$ be the p sets of input combinations for which the p prime implicants or the p AND-gate outputs are 1. The s-a-0 test set T_{0j} for AND gate j can be found by the equation

$$T_{0j} = S_j - \sum_{\substack{k=1 \\ k \neq j}}^{p} (S_j \cap S_k) \qquad \text{for } j = 1, 2, \ldots, p \tag{7.4.3}$$

Note that $T_{0j} \neq \varnothing$ for all j, where $\varnothing$ denotes the empty set. Choose a combination from each set T_{0j} that will form a minimal complete s-a-0 test set.

Step 2 Determination of a minimal complete s-a-1 test set. For each prime implicant m_i, find all its adjacent product terms $m_{i1}^*, m_{i2}^*, \ldots, m_{iq}^*$, where q is the number of literals in the prime implicant m_i^*. Then for each m_{ij}^*, find the set R_{ij} of all the combinations for which m_{ij}^* is 1. The complete s-a-1 test set for the jth input of ith AND gate

$$T_{1ij} = R_{ij} - \sum_{k=1}^{p} S_k \qquad \text{for } i = 1, 2, \ldots, p;\ j = 1, 2, \ldots, q_i \tag{7.4.4}$$

From the test sets T_{1ij}, for $i = 1, 2, \ldots, p$, and $j = 1, 2, \ldots, q_i$, we can form an s-a-1 fault-detection table. A minimal cover T_1 can be obtained from it using the standard procedure described in the previous section.

Step 3 The minimal complete s-a-0 and s-a-1 test set T is the union of T_0 and T_1.

To illustrate this method, consider the construction of a minimal complete test set of the function of Eq. (7.4.2) using the tabular method.

TABLE 7.4.1

Prime Implicant	Binary-Number Representation	S_j	s-a-0 Test	Input at Which an s-a-0 Fault Will Be Detected
x_2x_4	$d1d1$	$S_1 = \{5, 7, 13, 15\}$	$T_{01} = S_1 - \sum_{k=1}^{3} S_1 \cap S_k = \{7\}$	At any input of gate 1
$x_3'x_4$	$dd01$	$S_2 = \{1, 5, 9, 13\}$	$T_{02} = S_2 - \sum_{k=1,3} S_2 \cap S_k = \{1, 9\}$	Any input of gate 2
$x_1x_2x_3$	$111d$	$S_3 = \{14, 15\}$	$T_{03} = S_3 - \sum_{k=1}^{2} S_3 \cap S_k = \{14\}$	Any input of gate 3

TABLE 7.4.2

Prime Implicant	Adjacent Product Terms	Binary-Number Representation	R_{ij}	s-a-1 Test	Input at Which an s-a-1 Fault Will Be Detected
x_2x_4	$x_2'x^4$	$d0d1$	$R_{11} = \{1, 3, 9, 11\}$	$T_{111} = \{3, 11\}$	Input x_2 of gate 1
	x_2x_4'	$d1d0$	$R_{12} = \{4, 6, 12, 14\}$	$T_{112} = \{4, 6, 12\}$	Input x_4 of gate 1
$x_3'x_4$	$x_3'x_4'$	$dd00$	$R_{21} = \{3, 7, 11, 15\}$	$T_{121} = \{3, 11\}$	Input x_3' of gate 2
	$x_3'x_4'$	$dd00$	$R_{22} = \{0, 4, 8, 12\}$	$T_{122} = \{0, 4, 8, 12\}$	Input x_4 of gate 2
$x_1x_2x_3$	$x_1'x_2x_3$	$011d$	$R_{31} = \{6, 7\}$	$T_{131} = \{6\}$	Input x_1 of gate 3
	$x_1x_2'x_3$	$101d$	$R_{32} = \{10, 11\}$	$T_{132} = \{10, 11\}$	Input x_2 of gate 3
	$x_1x_2x_3'$	$110d$	$R_{33} = \{12, 13\}$	$T_{132} = \{12\}$	Input x_3 of gate 3

Step 1 From the prime implicants x_2x_4, $x_3'x_4$, and $x_1x_2x_3$, whose binary-number representation are $d1d1$, $dd01$, and $111d$, respectively, we can find all input combinations (minterms) for which $f = 1$.

$$d1d1 = \{0101,\ 0111,\ 1101,\ 1111\} = \{5, 7, 13, 15\} = S_1$$
$$dd01 = \{0001,\ 0101,\ 0101,\ 1101\} = \{1, 5, 9, 13\} = S_2$$
$$111d = \{1110,\ 1111\} = \{14, 15\} = S_3$$

We then use Eq. (7.4.3) to obtain the s-a-0 tests, as shown in Table 7.4.1.

Step 2 The adjacent product terms of the prime implicants x_2x_4, $x_3'x_4$, and $x_1x_2x_3$ are found to be $x_2'x_4$, x_2x_4', x_3x_4, $x_3'x_4'$, $x_1'x_2x_3$, $x_1x_2'x_3$, and $x_1x_2x_3'$, which are shown in the second column of Table 7.4.2. As before, from the adjacent product terms, we obtain all input combinations for which adjacent product terms equal 1. They are tabulated in the fourth column of the table. By applying Eq. (7.4.4), the s-a-1 tests are obtained and shown in the fifth column of the table. For example, the s-a-1 tests for detecting input the x_2 s-a-1 fault of gate 1 is obtained by

$$T_{111} = R_{11} - \sum_{k=1}^{P} S_k = \{1, 3, 9, 11\} - \{1, 5, 7, 9, 13, 14, 15\}$$
$$= \{3, 11\}$$

The rest of the T_{1ij} are obtained similarly.

An s-a-1 fault-detection table constructed from T_{1ij} is shown in Table 7.4.3. It is observed that tests 6 and 12 are essential and must be included in the test set. However, these two tests also detect the s-a-1 faults of x_4 of gates 1 and 2. The remaining gate inputs, x_2 of gate 1, x_3' of gate 2, and x_2 of gate 3, can be detected by test 11. Therefore, $T_1 = \{6, 11, 12\}$.

TABLE 7.4.3 s-a-1 Fault-Detection Table

Test \ Gate and Input	Gate 1		Gate 2		Gate 3		
	x_2	x_4 ✓	x_3'	x_4 ✓	x_1 ✓	x_2	x_3 ✓
0				1			
3	1		1				
4		1		1			
*6		1			1		
8				1			
10						1	
11	1		1			1	
*12		1		1			1

Step 3 Combining T_0 and T_1, we obtain the minimal complete test set $T =$ {7, 1 or 9, 14, 6, 11, 12}, which is exactly the same as obtained previously.

Exercise 7.4

1. Find a minimal complete test set for detecting any fault in the minimal two-level AND–OR circuit in Fig. P7.4.1 using the Karnaugh map method.

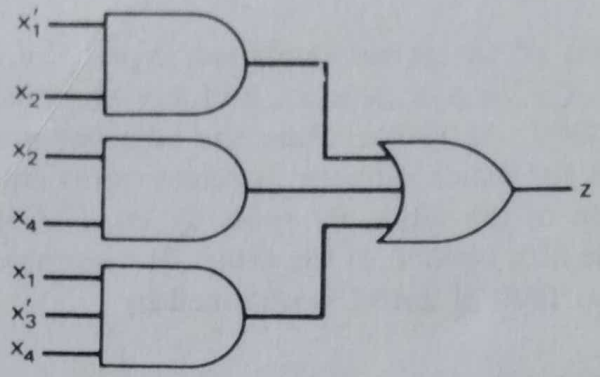


Fig. P7.4.1

2. Modify the Karnaugh map method so that it will work for the minimal two-level OR–AND circuit. Apply this modified version to the circuit of Fig. P7.4.2.

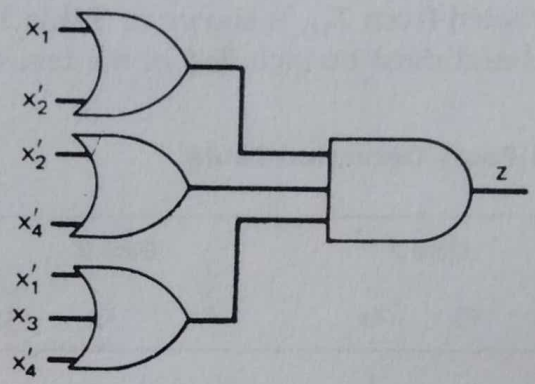


Fig. P7.4.2

3. Modify the Karnaugh map method so that it will work for the minimal two-level NAND circuit. Apply this modified version to the circuit of Fig. P7.4.3.

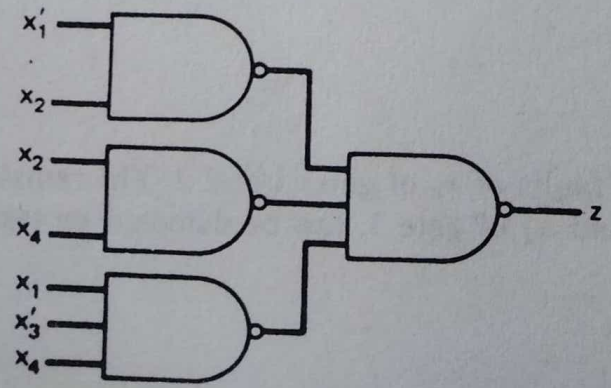


Fig. P7.4.3

4. Modify the Karnaugh map method so that it will work for the minimal two-level NOR circuit. Apply this modified version to the circuit of Fig. P7.4.4.

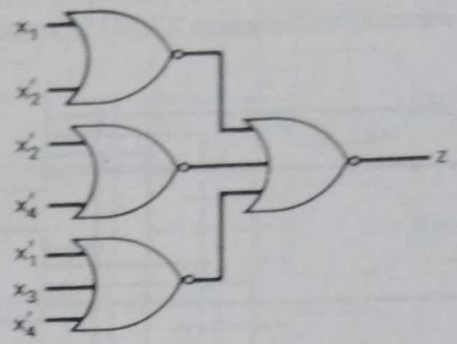
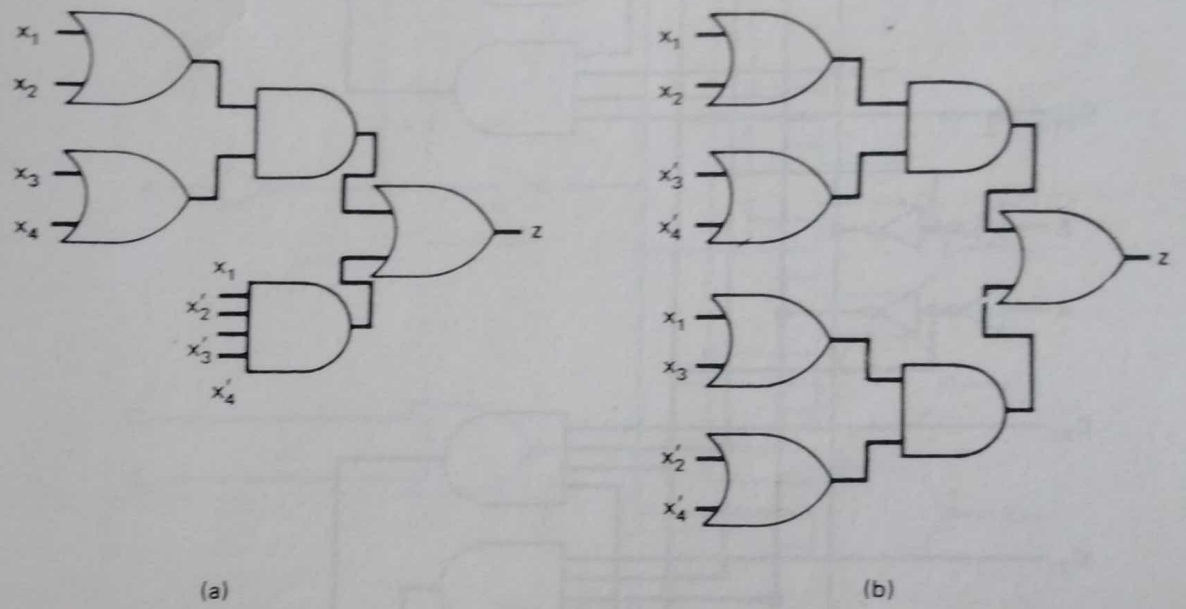


Fig. P7.4.4

5. Apply the Karnaugh map method to derive a minimal complete test set for each of the multilevel circuits of Fig. P7.4.5.



(a) (b)

Fig. P7.4.5

6. Use the tabular method to derive a minimal complete test set for detecting any faults in the circuit of Fig. P7.4.6.

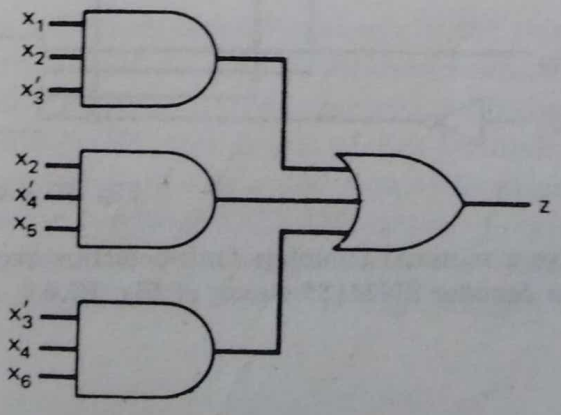


Fig. P7.4.6

7. Derive a minimal complete fault-detection experiment for checking the true/complement, zero/one device SN74H87, whose circuit diagram was shown in Fig. 2.4.2.
8. Derive a minimal complete fault-detection experiment for checking the dual 4-line-to-1-line data selector/multiplexers SN74153 circuit of Fig. P7.4.8.

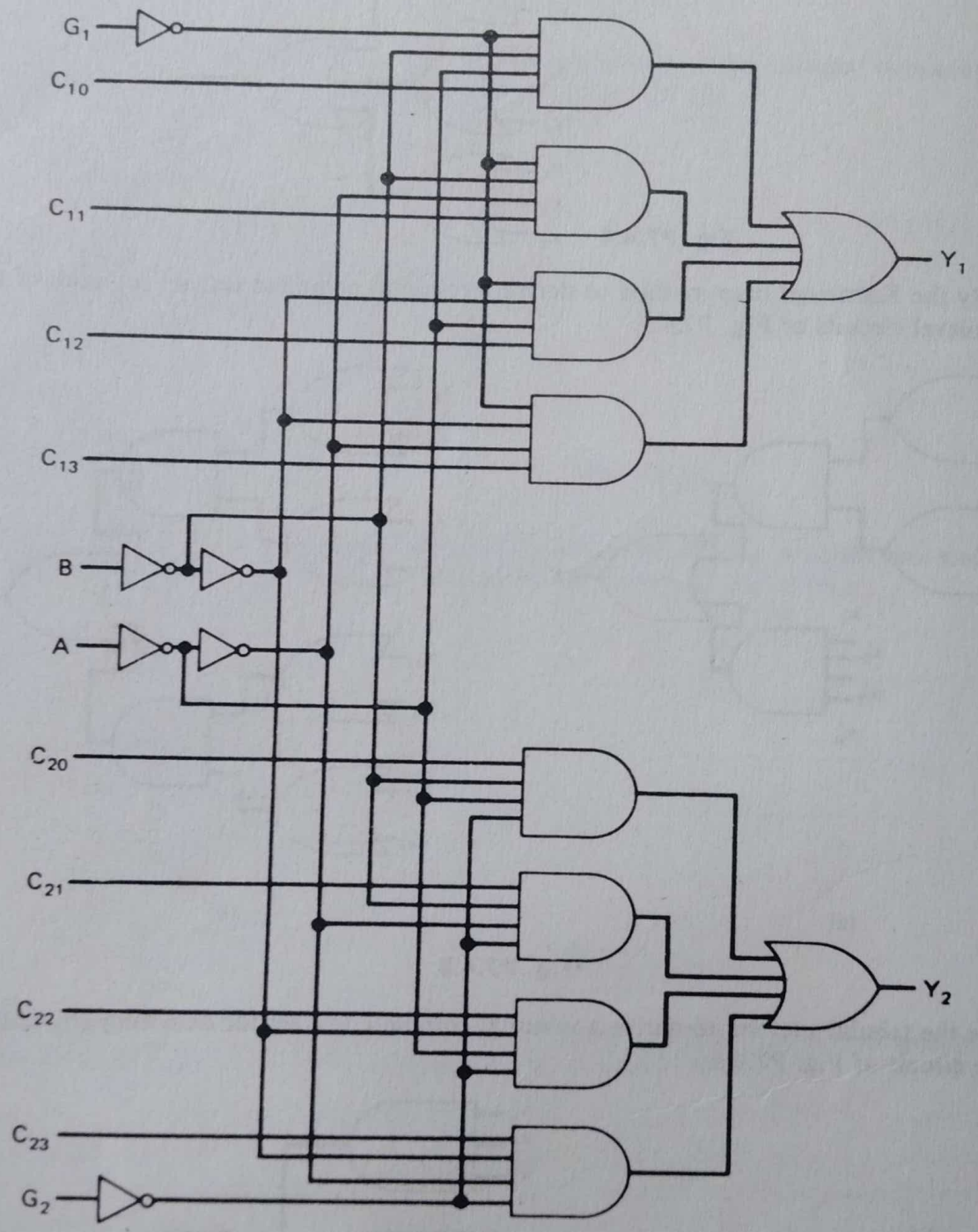


Fig. P7.4.8

9. Derive a minimal complete fault-detection experiment for checking the dual 2-line-to-4-line decoder SN74155 circuit of Fig. P7.4.9.

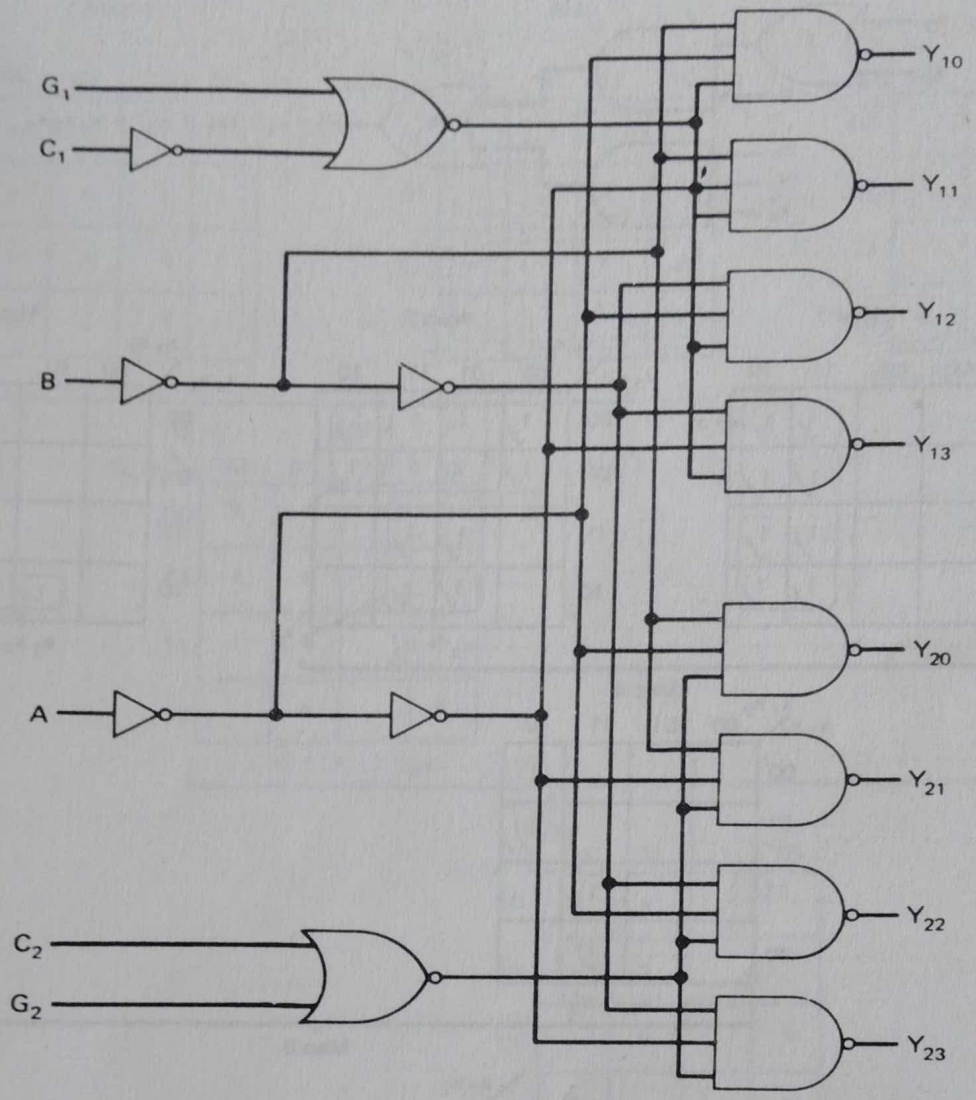


Fig. P7.4.9

7.5 Multilevel-Circuit Fault Detection

Multilevel AND–OR Circuits The minimum sum-of-products or the minimum product-of-sums form of a Boolean expression is not always the most economical to implement. Consequently, a common term or terms is often factored, with the result that a cheaper realization is obtained. AND–OR circuits that realize factored forms are multilevel AND–OR circuits. The purpose of this subsection is to extend the fault-detection methods described above for two-level AND–OR circuits to this class of multilevel circuits. The circuits considered are again assumed to be irredunant.

Let us consider the simple three-level AND–OR circuit of Fig. 7.5.1(a), whose

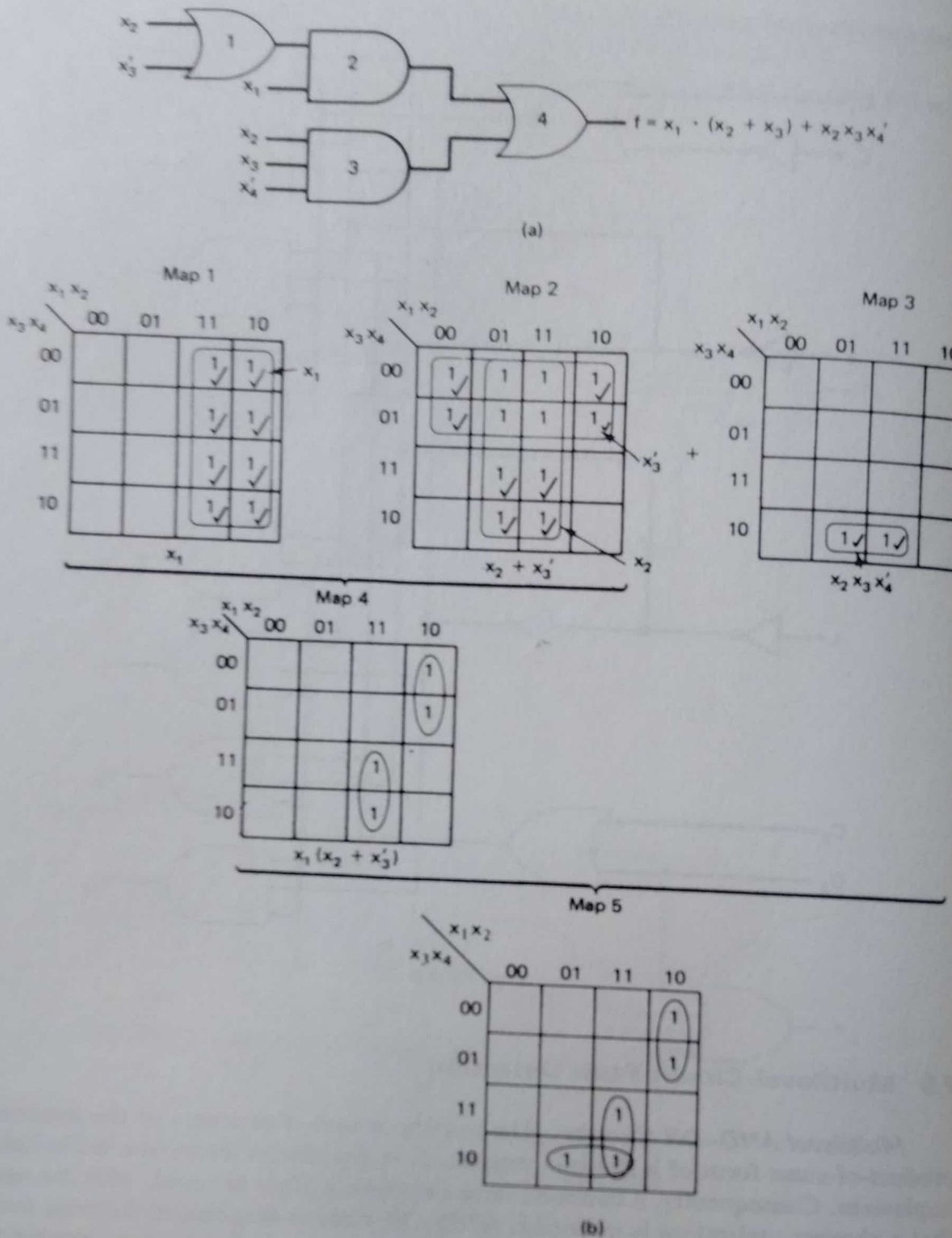
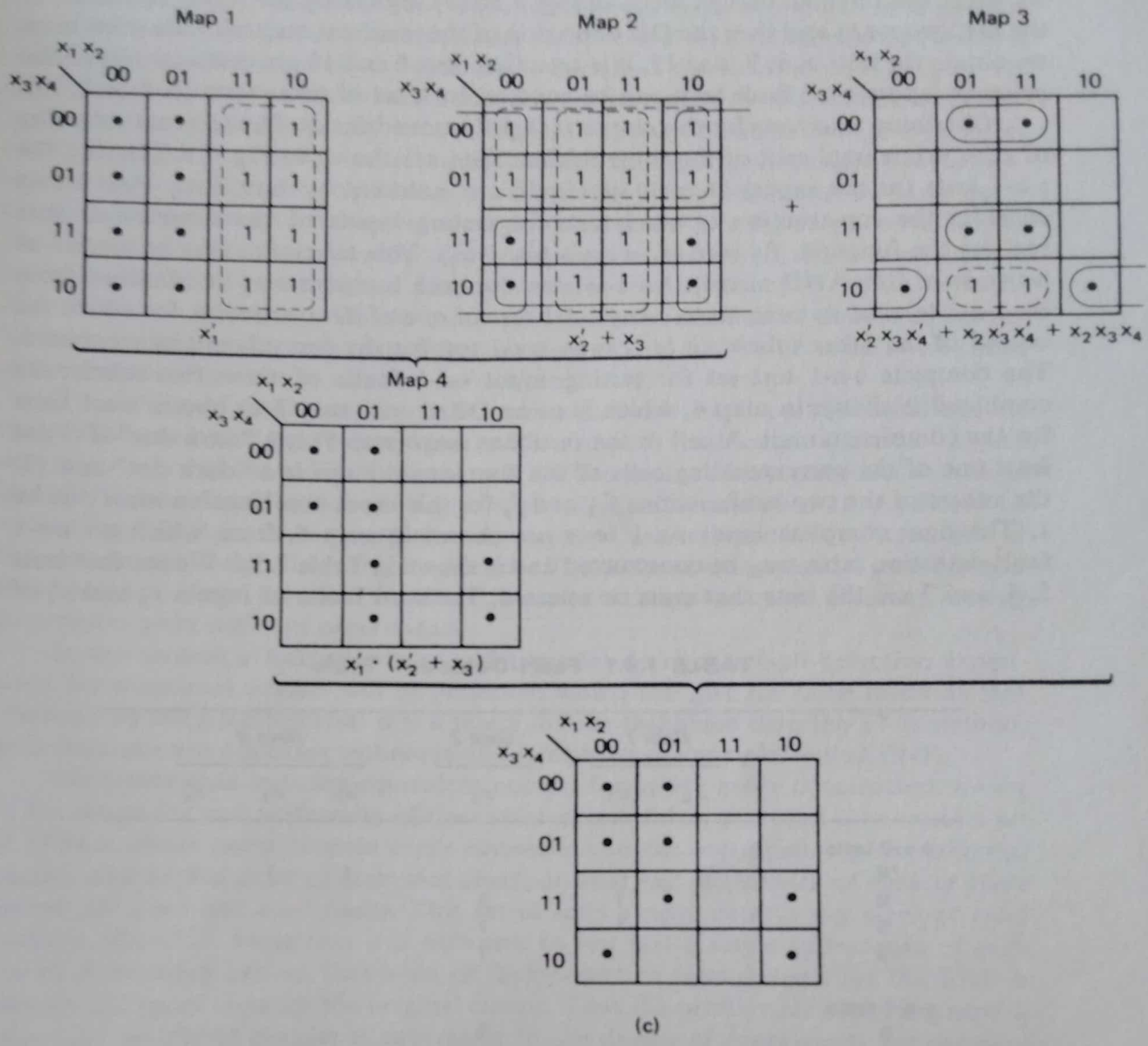


Fig. 7.5.1 (a) Multilevel AND-OR circuit; (b) maps for finding s-a-0 tests.

output function is

$$f(x_1, x_2, x_3, x_4) = x_1 \cdot (x_2 + x_3') + x_2 x_3 x_4' \qquad (7.5.1)$$



(c)

Fig. 7.5.1 (continued) (c) maps for finding s-a-1 tests.

Let

$$f_1(x_1, x_2, x_3, x_4) = x_1 \tag{7.5.2}$$

$$f_2(x_1, x_2, x_3, x_4) = x_2 + x_3' \tag{7.5.3}$$

$$f_3(x_1, x_2, x_3, x_4) = x_2 x_3 x_4' \tag{7.5.4}$$

As the first step we find the s-a-0 and s-a-1 tests for each individual subfunction f_1, f_2, and f_3 by using either the map method or the tabular method when the number of variables is large. For this example, since there are four variables, it is convenient to use the map method. The true subcubes of the three subfunctions f_1, f_2, and f_3 are

shown in the three Karnaugh maps in Fig. 7.5.1(b). By taking the AND operation of the first two maps and then the OR operation of the resultant map with the third map, we obtain the tests 6, 8, 9, and 15. It is seen that tests 6 and 15 are essential and all the primary inputs s-a-0 fault tests can be covered by a set of three tests {6, 8 or 9, 15}.

Obtaining s-a-1 tests for this circuit is slightly more difficult. The adjacent subcubes of each prime implicant of the three subfunctions are shown in Fig. 7.5.1(c), and the s-a-1 tests for the inputs of each subcircuit are indicated by dark dots. First let us consider the construction of s-a-1 tests for testing inputs of the subnetwork that realizes the function $f_{12} = f_1 \cdot f_2 = (x_1) \cdot (x_2 + x_3')$. This subcircuit may be viewed as a two-level OR–AND circuit. An s-a-1 test for such a circuit may be obtained from the s-a-1 tests of its subcircuits. Any s-a-1 test of one of its subcircuits, for which the output of the other subcircuit is 1, is an s-a-1 test for the two subcircuits combined. The complete s-a-1 test set for testing input s-a-1 faults of these two subcircuits combined is shown in map 4, which is to be ORed with map 3 to obtain s-a-1 tests for the complete circuit. A cell in the resultant map (map 5) is a "dark dot" if (1) at least one of the corresponding cells of the component maps is a "dark dot" and (2) the values of the two subfunctions f_{12} and f_3 for this input combination must not be 1. The final complete input s-a-1 tests are shown in map 5, from which an s-a-1 fault-detection table may be constructed and is shown in Table 7.5.1. We see that tests 2, 4, and 7 are the tests that must be selected. The s-a-1 faults of inputs x_2 and x_3' of

TABLE 7.5.1 Fault-Detection Table

	Gate 1		*Gate 2*	*Gate 3*		
	x_2	x_3	x_1	x_2	x_3	x_4'
s-a-0 tests						
*6				1	1	1
8		1	1			
9		1	1			
*15	1		1			
s-a-1 tests						
0			1			
1			1			
*2				1		
*4			1		1	
5			1			
*7			1			1
10	1	1				
11	1	1				

gate 1 may be detected by either test 10 or test 11. The minimal complete s-a-1 test set for the circuit is [2, 4, 7, 10, or 11]. Thus the minimal complete experiment for detecting s-a-0 and s-a-1 faults in the circuit of Fig. 7.5.1(a) consists of the following seven tests: [6, 8 or 9, 15, 2, 4, 7, 10 or 11].

ENF–Karnaugh Map Method The problem of detecting faults in a general combinational circuit is considerably more complicated than in the case of two-level AND–OR circuits described in Section 7.4. Except in the case where each gate has only a fan-out of 1 (tree-like circuits), and those multilevel AND–OR circuits discussed in the previous subsection, it is not true that testing only the inputs will always detect all the faults within the circuit. If we try to apply an experiment designed for a two-level circuit to a logically equivalent multilevel circuit in which two or more paths emanate from a certain gate and reconverge at a level closer to the output terminal, we shall find that a fault in one path may not always be detectable if the other path is faultless. We shall be concerned with procedures for the detection of single faults This limitation does not, of course, exclude the detection of most double and other multiple faults, but it emphasizes that only single faults will be detected in all cases, while some multiple faults may not be detected.

The design of fault-detection experiments for multilevel combinational circuits that will be presented here is based on the ideas of path sensitization (Section 7.2) and equivalent normal form (ENF) (Section 7.3). The general fault-detection technique using ENF introduced by Armstrong (reference 7) is rather complicated. As described in Section 7.3, it involves separate computations for s-a-0 and s-a-1 faults, and it depends on the order of testing the literals. The order is determined by assigning a score to each of the inputs of the ENF, and, as is often the case, different scoring procedures yield different experiments.

In this section a Karnaugh map technique for deriving a fault-detection experiment for multilevel circuits will be presented which will give the same result as that obtained by the ENF method. It is a much simpler technique than the ENF method, as it does not use a scoring technique and is without the complemented ENF.

Fault-detection tests for equivalent normal forms are easily constructed, owing to the simplicity and uniformity of their structure. All that is needed is to select a set of literals whose paths contain every connection in the corresponding hypothetical circuit, and *to find a set of tests that check at least one appearance of each of these literals for s-a-0 and s-a-1 faults.* This set of tests clearly detects any s-a-0 or s-a-1 faults in the ENF. Note that it is sufficient to test just a single appearance of each literal. Armstrong proved that a set of fault-detection tests devised for the ENF is also a valid set of tests for the original circuit. Thus the problem of designing experiments for multilevel circuits is equivalent to the design of experiments for the two-level equivalent normal forms.

An ENF corresponds to a two-level AND–OR circuit. Consequently, the techniques developed above for two-level circuits can now be applied, with several modifications, to the ENF's.

1. Any literal that appears in two or more terms in the ENF need only be tested once.

2. When there are two literals $(x_i)_\alpha$ and $(x_i)_\beta$ associated with the same input (*not* variable) but different paths in the ENF, it means that in the original circuit the signal x_i diverges into two different paths α and β and then recoverges. There are four possible cases:

(a) The literals are both complemented or both uncomplemented and appear in different terms of the ENF. This corresponds to the case where both paths α and β have either an even or an odd number of inversions. (An inversion is obtained by a NOT, a NAND, and a NOR gate.) The literals associated with these two paths must then have identical polarities, that is, both complemented or both uncomplemented. When these literals appear in different terms in the ENF, *they may be tested simultaneously for s-a-1 faults.* In this case the output will change incorrectly to 1, regardless of whether the fault is common to both paths or is just one of the paths and will be detected. However, *simultaneous s-a-0 tests should be avoided because they may leave some faults undetected.* The correctly operating path will provide the required 1 output, and a fault in the second path will not be detected.

(b) One of the literals is complemented and the other is not, and they appear in different terms of the ENF. This corresponds to the case where one path has an even number of inversions and the other has an odd number of inversions. The literals in the ENF associated with these paths must be complementary. Suppose now that we test one of these literals for an s-a-0 fault. Then if the input combination is such that the other literal is tested for s-a-1 fault, the output will be 1 and the fault will go undetected. Hence *no input combination should be selected so as to test two complementary literals simultaneously for s-a-0 and s-a-1 faults.*

(c) The literals are both complemented or both uncomplemented and they appear in the same term of the ENF. In this case *the literals cannot be tested for s-a-1 faults individually but can be tested simultaneously. Of course, they can be tested individually for s-a-0 faults. Simultaneous s-a-0 tests should be avoided* for the reason stated in (a).

(d) One of the literals is primed and the other is not and they both appear in the same term of the ENF. In this case no test can be found.

For easiness of applying these principles and rules they are summarized as follows:

RULE 1

Any literal that appears in two or more terms in the ENF need only be tested once. Thus the complete s-a-0 and s-a-1 test sets for this literal are the unions of all s-a-0 tests and s-a-1 tests for each appearance of the literal in the ENF, respectively. s-a-0 tests and s-a-1 tests for appearance of the literal are obtained based on the following rules:

(a) Individual and simultaneous testing of two or more appearances of the literal for s-a-0 and s-a-1 faults is valid.

(b) Simultaneous testing of one appearance of the literal for s-a-0 faults and the other for s-a-1 faults is invalid.

(c) Simultaneous s-a-0 testing for the same literal contained in different terms

is valid, as is simultaneous s-a-1 testing for the same literal contained in different terms.

RULE 2

When two literals associated with the same input but different paths are both uncomplemented [$(x_i)_\alpha$ and $(x_i)_\beta$] or both complemented [$(x_i')_\alpha$ and $(x_i')_\beta$] and appear in different terms in the ENF:

(a) Individual s-a-0 and s-a-1 tests for each literal is valid.
(b) Simultaneous testing of the two literals for s-a-1 faults is valid.
(c) Simultaneous testing of the two literals for s-a-1 faults should be avoided.
(d) Simultaneous testing of the two literals in complementary forms [i.e., $(x_i)_\alpha$ and $(x_i')_\beta$ or $(x_i')_\alpha$ and $(x_i)_\beta$] for s-a-0 and s-a-1 faults is invalid.

RULE 3

When two literals associated with the same input but different paths are complementary to each other and appear in different forms in the ENF, s-a-1 tests for an appearance of the literal are obtained based on the following rules:

(a) Individual and simultaneous testing of two or more appearances of the literal for s-a-0 and s-a-1 faults is valid.
(b) Simultaneous testing of one appearance of the literal for s-a-0 faults and the other appearance for s-a-1 faults is invalid.
(c) Simultaneous s-a-0 testing for the same literal contained in different terms is valid, as is simultaneous s-a-1 testing for the same literal contained in different terms.

RULE 4

When two literals associated with the same input but different paths are both uncomplemented or both complemented and appear in the same term in the ENF:

(a) Individual s-a-0 testing for the literals is valid.
(b) Individual s-a-1 testing for the literals is invalid.
(c) Simultaneous s-a-0 testing for the literals should be avoided.
(d) Simultaneous s-a-1 testing for the literals is valid.

RULE 5

When two complementary literals associated with the same original circuit input appear in the same term in the ENF, no test (either s-a-0 or s-a-1) can be found for them or for any other literals of the term. Hence the term should be discarded completely.

This method for obtaining s-a-0 and s-a-1 tests for any general combinational circuits may be systematically described as follows:

Step 1 Find the equivalent normal form of a given multiple circuit as described in Section 7.3.

Step 2 Draw the map of the ENF of the circuits and label all subcubes in the map. (The map is needed here only for clarity. Actually, the adjacencies can be determined in a systematic manner without the use of map, as described in Section 7.4.)

Step 3 Construct the testing table. Each distinct literal of the ENF appears once as a column heading. Literals corresponding to the same circuit input are grouped together. (This will later help to ensure that no simultaneous s-a-0 and s-a-1 tests are made on complementary inputs.) All possible s-a-0 and s-a-1 tests are made row headings.

Step 4 Complete the testing table. A literal can be tested for s-a-0 faults in any one of the true subcubes in which it is contained. It can be tested for s-a-1 faults by means of a subcube adjacent to any one of the true subcubes.

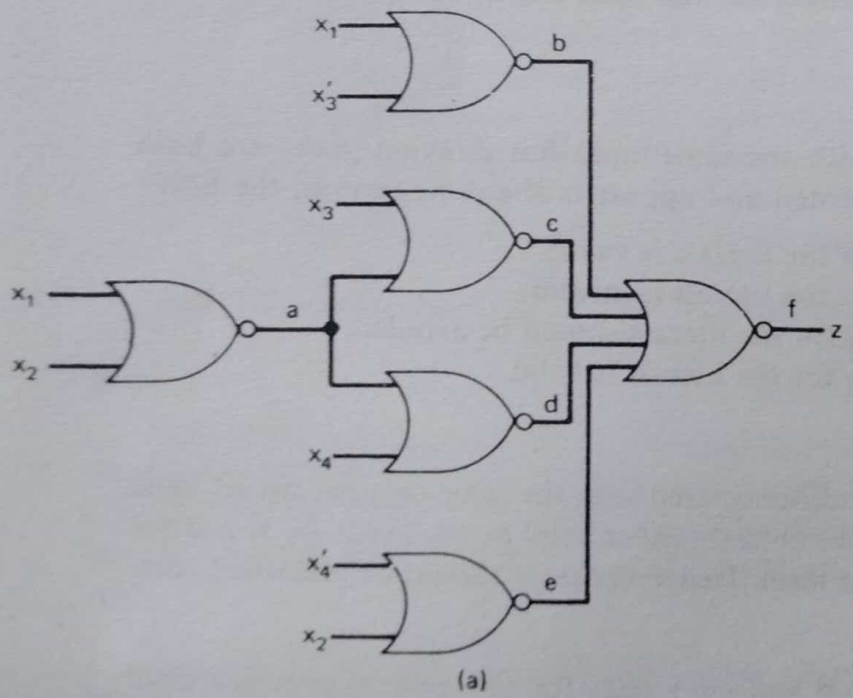
Step 5 Find an s-a-0 cover and an s-a-1 cover.

This method is illustrated by the following examples.

Example 7.5.1

The ENF for the circuit of Fig. 7.5.2(a), after all the terms containing one or more products of complementary literals associated with the same variable but different paths (there are 14 such terms by rule 5), is

$$z = (x_1')_{acf}(x_1')_{adf}(x_2')_{acf}(x_2')_{adf}(x_3')_{bf}(x_4')_{ef} + (x_1)_{bf}(x_2)_{ef}(x_3)_{cf}(x_4)_{df}$$



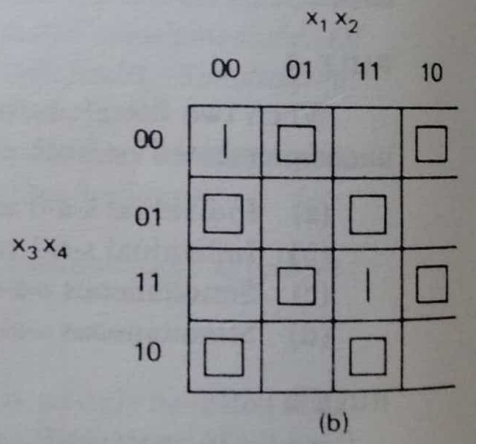


Fig. 7.5.2 (a) Multilevel NOR circuit; (b) true subcubes and adjacent subcubes.

Again, to simplify the notation, let $acf \longrightarrow \alpha$, $adf \longrightarrow \beta$, $bf \longrightarrow \gamma$, $ef \longrightarrow \delta$, $cf \longrightarrow \epsilon$, $df \longrightarrow \sigma$. Then

$$z = x'_{1\alpha}x'_{1\beta}x'_{2\alpha}x'_{2\beta}x'_{3\gamma}x'_{4\delta} + x_{1\gamma}x_{2\delta}x_{3\epsilon}x_{4\sigma}$$

From rule 4(b), the literals $x'_{1\alpha}$ and $x'_{1\beta}$ in the first term cannot be tested for s-a-1 individually, nor can the literals $x'_{2\alpha}$ and $x'_{2\beta}$. However, by rule 4(d), simultaneous s-a-1 testing for them is valid. Figure 7.5.2(b) shows the true subcubes and the adjacent subcubes of the prime implicants of z. There are two true subcubes {0} and {15}, which will detect all s-a-0 faults for all 10 literals. Note that individual s-a-0 tests for literals $x'_{1\alpha}$ and $x'_{1\beta}$ and for literals $x'_{2\alpha}$ and $x'_{2\beta}$ are valid [rule 4(a)]. The s-a-1 tests for these 10 literals are the eight disjoint adjacent subcubes: 1, 2, 4, 7, 8, 11, 13, and 14. For this example the construction of testing table is not needed, and the (minimal) fault-detection experiment is {0, 15, 1, 2, 4, 7, 8, 11, 13, 14}.

Example 7.5.2

The ENF for the circuit of Fig. 7.2.1 was found to be [Eq. (7.3.1)]

$$h = (x_1)_{afh}(x_2)_{bdfh} + (x'_2)_{begh}(x'_3)_{cgh}$$

In order to simplify the notation, let us replace the subscript sequences as follows: $afh \longrightarrow \alpha$, $bdfh \longrightarrow \beta$, $begh \longrightarrow \gamma$, and $cgh \longrightarrow \delta$. Then we write

$$h = x_{1\alpha}x_{2\beta} + x'_{2\gamma}x_{3\delta}$$

and its true subcubes map and adjacent subcubes map are, respectively, shown in Fig. 7.5.3. The literals associated with each original circuit input x_i (see Fig. 7.5.1) are first grouped and the s-a-0 and s-a-1 tests for each group of literals associated with the same input are considered separately.

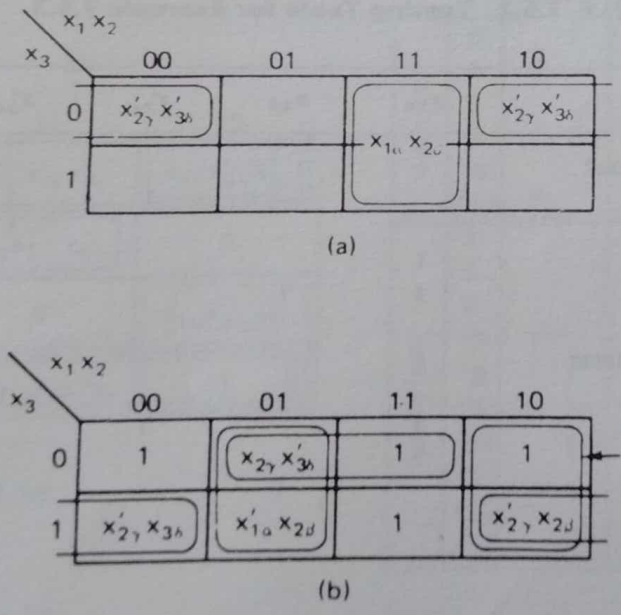


Fig. 7.5.3 (a) True subcubes; (b) adjacent subcubes of the ENF of the circuit of Fig. 7.2.1.

(1) Literal $x_{1\alpha}$. Since $x_{1\alpha}$ is the only literal associated with x_1 and is contained in only one subcube, the s-a-0 and s-a-1 tests are readily found to be

$$T_0(x_{1\alpha}) = \{6, 7\}, \qquad T_1(x_{1\alpha}) = \{2, 3\}$$

(2) Literals $x_{2\beta}$ and $x'_{2\gamma}$. There are two literals associated with original circuit input x_2, $x_{2\beta}$, and $x'_{2\gamma}$, which are complementary to each other and are contained in different terms of the ENF. Since by rule 3(c) no input combination should be selected so as to test $x_{2\beta}$ and $x'_{2\gamma}$ simultaneously for s-a-0 and s-a-1 faults, input combination 6 (s-a-0 test for $x_{2\beta}$ and s-a-1 test for $x'_{2\gamma}$) and input combination 4 (s-a-0 test for $x'_{2\gamma}$ and s-a-1 test for $x_{2\beta}$) should not be selected for testing literals $x_{2\beta}$ and $x'_{2\gamma}$. Thus

$$T_0(x_{2\beta}) = \{7\}, \qquad T_0(x'_{2\gamma}) = \{0\}$$
$$T_1(x_{2\beta}) = \{5\}, \qquad T_1(x'_{2\gamma}) = \{2\}$$

(3) Literal $x'_{3\delta}$. Again, $x'_{3\delta}$ is the only literal associated with x_3 and is contained in only one true subcube. Thus the s-a-0 and s-a-1 tests for $x'_{3\delta}$ can be obtained in a straightforward manner.

$$T_0(x'_{3\delta}) = \{0, 4\}, \qquad T_1(x'_{3\delta}) = \{1, 5\}$$

A testing table may now be constructed and is shown in Table 7.5.2. It is observed that tests 0 and 7 are essential s-a-0 tests and constitute an s-a-0, and tests 2 and 5 are essential s-a-1 tests and constitute an s-a-1 cover. Therefore, a minimal complete test set for testing any s-a-0 and s-a-1 faults is the circuit of Fig. 7.2.1(a). This is the same result obtained in Example 7.3.1 using the ENF method.

TABLE 7.5.2 Testing Table for Example 7.5.3

	$x_{1\alpha}$	$x_{2\beta}$	$x'_{2\gamma}$	$x'_{3\delta}$
s-a-0 test				
*0			1	1
4				1
6	1			
*7	1	1		
s-a-1 tests				
1				1
*2	1		1	
3	1			
*5		1		1

Example 7.5.3

As a third example, consider the circuit of Fig. 7.5.3(a), which has four input variables, x_1, x_2, x_3, and x_4. The minimal complete test set for this circuit was obtained in Example

7.3.2 using a scoring technique and the complemented ENF. Here we would like to construct it using the map technique. The ENF for this circuit was found to be

$$z = (x_1')_{247}(x_2)_{47} + (x_2)_{47}(x_3')_{1247}(x_4)_{1247} + (x_1)_{2567}(x_3)_{12567}(x_2')_{367} + (x_1)_{2567}(x_3)_{12567}(x_4')_{367} + (x_1)_{2567}(x_4')_{12567}(x_2')_{367} + (x_1)_{2567}(x_4')_{12567}(x_4')_{367}$$

To simplify the notation, let us replace the subscript sequences as follows: $247 \longrightarrow \alpha$, $47 \longrightarrow \beta$, $1247 \longrightarrow \gamma$, $2567 \longrightarrow \delta$, $367 \longrightarrow \epsilon$, $12567 \longrightarrow \lambda$. The above equation can now be written

$$z = x'_{1\alpha}x_{2\beta} + x_{2\beta}x'_{3\gamma}x_{4\gamma} + x_{1\alpha}x'_{2\epsilon}x_{3\lambda} + x_{1\delta}x_{3\lambda}x'_{4\epsilon} + x_{1\delta}x'_{2\epsilon}x'_{4\lambda} + x_{1\delta}x'_{4\lambda}x'_{4\epsilon}$$

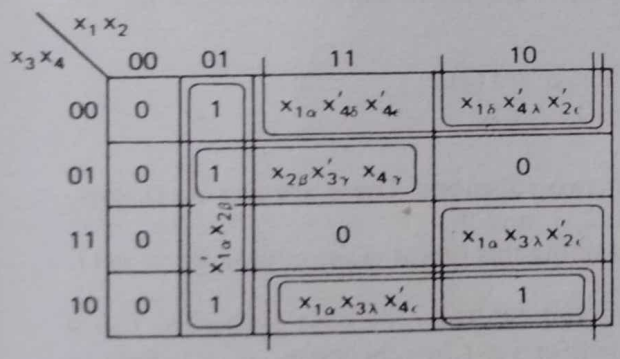
The literals of z of the ENF are grouped into six groups:

$$\{x'_{1\alpha}, x_{1\delta}\}, \{x_{3\lambda}, x'_{3\gamma}\}, \{x'_{4\lambda}, x_{4\gamma}\}, \{x_{2\beta}\}, \{x'_{2\epsilon}\}, \{x'_{4\epsilon}\}$$

The literals in each group are associated with the same original circuit input of the circuit of Fig. 7.3.2.

The ENF map of z is shown in Fig. 7.5.4(a). The s-a-0 and s-a-1 tests for each of the literals are constructed as follows:

1. *Literals $x'_{1\alpha}$ and $x_{1\delta}$.* Since $x'_{1\alpha}$ and $x_{1\delta}$ are complementary to each other and are contained in different terms, by rule 3(c) any input combinations that simultaneously test s-a-0 and s-a-1 faults of the two literals cannot be selected. Tests 4 and 6 cannot be selected because they are s-a-0 tests for $x'_{1\alpha}$ and also s-a-1 tests for $x_{1\delta}$. The s-a-1 tests are determined by observing that 15 is the only adjacent cell for which $z = 0$ and in which the value of x_1 is complementary to its value in the subcube being tested.



(a)

		$x'_{1\alpha}x_{1\delta}$	$x_{3\lambda}x'_{3\gamma}$	$x'_{4\lambda}x_{4\gamma}$	$x_{2\beta}$	$x'_{2\epsilon}$	$x'_{4\epsilon}$
a-a-1 tests	0	1			1		
	1				1		
	2	1			1		
	3	1			1		
	*9		1	1	1		
	*15	1	1			1	1
a-a-0 tests	4				1		
	5				1		
	6				1		
	*7	1			1		
	*8	1		1			
	10	1					
	*11	1	1			1	
	12						1
	*13		1	1	1		
	14						1

(b)

Fig. 7.5.4 (a) ENF map; (b) testing table of Example 7.5.4.

The literal $x_{1\delta}$ is contained in four subcubes. It may be tested in any one of these subcubes. Again, rule 3(c), tests 12 and 14, are invalid as s-a-0 tests for $x_{1\alpha}$ because they are also s-a-1 tests for $x'_{1\alpha}$ (i.e., they are adjacent to $x'_{1\alpha}x_{2\beta}$); the valid s-a-0 tests for $x_{1\delta}$ are therefore tests 8, 10, and 11. Note that tests 8 and 10 are valid [by rule 1(c)], although they belong to more than one subcube. The s-a-1 tests for $x_{1\delta}$ are those adjacent cells in which $z = 0$ and the value of x_1 is complementary. Clearly, 0, 2, and 3 satisfy these conditions. Hence

$$T_0(x'_{1\alpha}) = \{7\}, \qquad T_0(x_{1\delta}) = \{8, 10, 11\}$$
$$T_1(x'_{1\alpha}) = \{15\}, \qquad T_1(x_{1\delta}) = \{0, 2, 3\}$$

2. *Literals $x'_{3\gamma}$ and $x_{3\lambda}$.* From the ENF map we see once again that there exist no tests which test these two literals simultaneously; hence the construction of s-a-0 and s-a-1 tests for one literal is completely independent of that for the other. $x'_{3\gamma}$ is contained in only one subcube. Thus its s-a-0 test is 13 and its s-a-1 test (where the value of x_3 is complementary) is 15. The literal $x_{3\lambda}$ is contained in two subcubes. The only s-a-0 test is 11. Tests 10 and 14 are invalid since they belong to true subcubes that do not contain $x_{3\lambda}$. The s-a-1 test is 9. In summary, the following test sets are found.

$$T_0(x'_{3\gamma}) = \{13\}, \qquad T_0(x_{3\lambda}) = \{11\}$$
$$T_1(x'_{3\gamma}) = \{15\}, \qquad T_1(x_{3\lambda}) = \{9\}$$

3. *Literals $x'_{4\lambda}$ and $x_{4\gamma}$.* The literal $x'_{4\lambda}$ is contained in two terms (true subcubes). One of these subcubes is $x_{1\delta}x'_{4\lambda}x'_{4\epsilon}$. But because this term contains two entries of the same variable (i.e., $x'_{4\lambda}$), which correspond to different inputs in the original circuit, none of the x'_4 inputs in this term can be tested for s-a-1 faults. Test 15 is invalid because it tests only the "untestable" $x_{1\delta}x'_{4\lambda}x'_{4\epsilon}$. Hence the s-a-1 test is 9. The candidates for the s-a-0 test for $x'_{4\lambda}$ are 8, 10, 12, and 14. Tests 10 and 14 are contained in other true subcubes, and test 12 is adjacent to subcube $x_{2\beta}x'_{3\gamma}x_{4\gamma}$ and will simultaneously test $x_{4\gamma}$ for s-a-1, so they are invalid. Thus the s-a-0 test is 8. The only s-a-0 test for $x_{4\gamma}$ is 13, and there are no s-a-1 tests, since there are no adjacent cells in which $z = 0$ and the value of x_4 is complementary. To summarize, we have

$$T_0(x'_{4\lambda}) = \{8\}, \qquad T_0(x_{4\gamma}) = \{13\}$$
$$T_1(x'_{4\lambda}) = \{9\}, \qquad T_1(x_{4\gamma}) = \phi$$

4. *Literal $x_{2\beta}$.* Literal $x_{2\beta}$ can be tested in two subcubes. There are five s-a-0 tests, 4, 5, 6, 7, and 13. There are five s-a-1 tests, 0, 1, 2, 3, and 9.

5. *Literal $x'_{2\epsilon}$.* There are two subcubes containing this literal. The only s-a-0 test is 11. The s-a-1 test is 15.

6. *Literal $x'_{4\epsilon}$.* There are two s-a-0 tests, 12 and 14. The only s-a-1 test is 15. Test 9 is invalid as an s-a-1 test, since $x'_{4\epsilon}$ cannot be tested for s-a-1 in subcube $x_{1\delta}x'_{4\lambda}x'_{4\epsilon}$, only in subcube $x_{1\delta}x_{3\lambda}x'_{4\epsilon}$.

After the complete s-a-0 and s-a-1 test sets for each literal are found, we construct the testing table [Fig. 7.5.4(b)], from which it is seen that tests 9 and 15 are the essential s-a-1 tests, and tests 7, 8, 11, and 13 are the essential s-a-0 tests. After the s-a-1 essential tests which cover columns $x'_{1\alpha}$, $x_{3\lambda}$, $x'_{4\lambda}$, $x_{2\beta}$, $x'_{2\epsilon}$, and $x'_{4\epsilon}$ are selected, left uncovered is only $x_{1\delta}$, which can be covered by 0, 2, or 3. Similarly, we find that the s-a-0 essential tests plus one

additional test (which can be either 12 or 14) will cover every column of the table. Hence the complete test set T is

$$T = \{0 \text{ or } 2 \text{ or } 3, 9, 15, 7, 8, 11, 13, 12 \text{ or } 14\}$$

It can be easily shown that this experiment is identical with the one obtained in Example 7.3.3. However, it has been obtained without a scoring technique and without using the complemented ENF. This method and the scoring technique obtain the same result, but the former is less complicated and easier to comprehend.

As a final remark, this method, like the ENF method, does not guarantee minimal experiments, nor is there a guarantee that a set of sensitized paths can be found for every circuit.

Exercise 7.5

1. Use the ENF–Karnaugh map method to determine a near-minimal complete test set for the circuit of Fig. P7.3.3 and compare the result with that obtained previously.

7.6 Boolean Difference Method

The need for a conceptually simple and straightforward ways of deriving test sequences for combinational circuits is the impetus behind the Boolean difference methods. Boolean difference is defined as being the exclusive-or operation between two Boolean functions, one representing the normal circuit and the other representing the faulty circuit. Thus if the Boolean difference is a 1, a fault is indicated.

Assume that there is a switching function that has one output F and n inputs $x_1, x_2, \ldots, x_n$, so $F(X) = F(x_1, x_2, \ldots, x_n)$. If one of the inputs to the switching function was in error, say input x_i, then the output would be $F(x_1, \ldots, x_i', \ldots, x_n)$. To analyze the action of the circuit when an error occurs, it is desirable to know under what circumstances the two outputs are the same. For this purpose, we define the following function.

DEFINITION 7.6.1

$$\frac{dF(X)}{dx_i} = F(x_1, \ldots, x_i, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_n)$$

The function $dF(X)/dx_i$ is called the *Boolean difference* of $F(X)$ with respect to x_i.

One may ask: Why do we use the derivative notation to represent the Boolean difference? Recall that the meaning of the derivative of a real continuous function $F(x)$ of a real continuous variable x may simply be stated as: the rate of change of $F(x)$ with respect to an infinitesimal change in x. In switching algebra, both F and x can only take on values 0 and 1. Thus, if the value of x is 0(1), it can only change to 1(0). This

change can be described by: x changes to x'. The change of x is therefore always 1, which may be seen from the following expression:

$$\Delta x = x \oplus x' = 1$$

Now let us examine the change of a binary function $F(x_1, \ldots, x_i, \ldots, x_n)$ due to the change of a binary variable x_i ($\Delta x_i = 1$). The functions F before and after the change of value of x_i may be represented by $F(x_1, \ldots, x_i, \ldots, x_n)$ and $F(x_1, \ldots, x_i', \ldots, x_n)$, which may have the following four possible cases:

F before the change $F(x_1, \ldots, x_i, \ldots, x_n)$	*F after the change* $F(x_1, \ldots, x_i', \ldots, x_n)$	$\Delta F = F(x_1, \ldots, x_i, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_n)$	Δx_i	$\frac{\Delta F}{\Delta x_i}$ or $\frac{dF(X)}{dx_i}$
0	0	0	1	0
1	1	0	1	0
0	1	1	1	1
1	0	1	1	1

It is seen that when $dF(X)/dx_i = 1$ ($dF(X)/dx_i = 0$), it means that the value of $F(X)$ has (has not) changed with x changed to x'. Our interest here is to find input combinations (tests) for detecting detectable faults on a wire x_i of a combinational circuit such that whenever x_i changes to x_i' (caused by a fault on wire x_i), $F(x_1, \ldots, x_i', \ldots, x_n)$ will immediately be different from $F(x_1, \ldots, x_i, \ldots, x_n)$. In other words, we are interested in finding input combinations for each fault occurring on wire x_i under investigation such that $dF(X)/dx_i = 1$. Before showing how a fault-detection test may be conveniently derived by using the Boolean difference, we first present some useful properties of the Boolean difference.

Operation Properties of Boolean Differences Based on Definition 7.6.1, a set of important operation properties can be derived as follows:

PROPERTY 1

$$\frac{d\overline{F(X)}}{dx_i} = \frac{dF(X)}{dx_i}$$

PROPERTY 2

$$\frac{dF(X)}{dx_i} = \frac{dF(X)}{dx_i'}$$

PROPERTY 3

$$\frac{d}{dx_i}\frac{dF(X)}{dx_j} = \frac{d}{dx_j}\frac{dF(X)}{dx_i}$$

PROPERTY 4

$$\frac{d[F(X)G(X)]}{dx_i} = F(X)\frac{dG(X)}{dx_i} \oplus G(X)\frac{dF(X)}{dx_i} \oplus \frac{dF(X)}{dx_i}\frac{dG(X)}{dx_i}$$

PROPERTY 5

$$\frac{d[F(X) + G(X)]}{dx_i} = \overline{F(X)}\frac{dG(X)}{dx_i} \oplus \overline{G(X)}\frac{dF(X)}{dx_i} \oplus \frac{dF(X)}{dx_i}\frac{dG(X)}{dx_i}$$

PROPERTY 6

$$\frac{d[F(X) \oplus G(X)]}{dx_i} = \frac{dF(X)}{dx_i} \oplus \frac{dG(X)}{dx_i}$$

The bar in $\overline{F(X)}$ denotes the complement of $F(X)$.

These properties can be derived in a straightforward manner and are useful in computing the Boolean difference. The derivations are left to the reader (problem 1).

The most important property of the Boolean difference is that it is equal to 1 when the outputs are different for normal and erroneous settings of input x_i, and equal to 0 if the logic output is the same for both normal and erroneous settings of input x_i. This is the basis for the use of the Boolean difference in the analysis of logic faults. The following definition relates directly to the problem of fault detection.

DEFINITION 7.6.2

A Boolean function $F(X)$ is said to be *independent of* x_i if and only if $F(X)$ is logically invariant under complementation of x_i [i.e., if $F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, x_i', \ldots, x_n)$].

If we consider $F(X)$ as an output function of a combinational circuit, then we say that $F(X)$ is independent of variable x_i if and only if for any values of the other variables, the output $F(X)$ is independent of the value of x_i. This implies a very important point, that a fault in x_i will not affect the final output $F(X)$.

In the following theorem, the independence relationship can be easily described using the Boolean difference.

THEOREM 7.6.1

A necessary and sufficient condition that a function $F(X)$ be independent of x_i is that $dF(X)/dx_i = 0$.

Proof: By Definition 7.6.2 and the condition $F \oplus F = 0$, the theorem immediately follows. ∎

Based on the result of Theorem 7.6.1, we now have additional operation properties to add to the original set.

PROPERTY 7

$$\frac{dF(X)}{dx_i} = 0 \qquad [\text{if } F(X) \text{ is independent of } x_i]$$

PROPERTY 8

$$\frac{dF(X)}{dx_i} = 1 \qquad [\text{if } F(X) \text{ depends only on } x_i]$$

PROPERTY 9

$$\frac{d[F(X)G(X)]}{dx_i} = F(X)\frac{dG(X)}{dx_i} \qquad [\text{if } F(X) \text{ is independent of } x_i]$$

PROPERTY 10

$$\frac{d[F(X) + G(X)]}{dx_i} = \overline{F(X)}\frac{dG(X)}{dx_i} \qquad [\text{if } F(X) \text{ is independent of } x_i]$$

The derivations can again be easily obtained and thus are left to the reader (problem 1).

Six Methods for Obtaining a Boolean Difference of Switching Function There are at least six methods that can be used to obtain the Boolean difference of a function. One of them is the one presented above, the analytic method using properties 1–10, termed method 1. Method 2 is based on the following theorem.

THEOREM 7.6.2

$$\frac{dF(X)}{dx_i} = F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \oplus F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

for any x_i, $1 \le i \le n$

Proof:

$$\begin{aligned}\frac{dF(X)}{dx_i} &= [x_i F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \oplus x_i' F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)] \\ &\quad \oplus [x_i' F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \oplus x_i F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)] \\ &= (x_i \oplus x_i')[F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)] \\ &\quad \oplus (x_i' \oplus x_i)[F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)] \\ &= F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \oplus F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \quad \blacksquare\end{aligned}$$

Method 3 is based on the following theorem.

THEOREM 7.6.3

If $F(X)$ is expressed in the form

$$F(X) = A(X) + x_i \cdot B(X) + x_i' C(X)$$

where $A(X)$, $B(X)$, and $C(X)$ are not functions of x_i, then

$$\frac{dF(X)}{dx_i} = [B(X) \oplus C(X)]\overline{A(X)}$$

The proof of this theorem is left to the reader as an exercise.

In addition to the above three analytical methods, there are three map methods. By Definition 7.6.1, $dF(X)/dx_i = F(x_1, \ldots, x_i, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_n)$, we can construct two Karnaugh maps, one for $F(x_1, \ldots, x_i, \ldots, x_n)$ and the other for $F(x_1, \ldots, x_i', \ldots, x_n)$. The maps are then added using modulo-2 addition in the corresponding entries. The resultant map is the map for $dF(X)/dx_i$ and can be simplified directly. This method will be referred to as method 4.

Another map method, method 5, is derived based on the expression of Boolean functions $F(X)$ in terms of a set of prime implicants summed by exclusive–or operators. The resultant equation is called the exclusive–or form. This prime exclusive–or form of $F(X)$ can be obtained easily by first writing $F(X)$ on the Karnaugh map and then regrouping the map entries. The grouping rule is to allow every entry to be used only for an odd number of times. The steps toward finding $dF(X)/dx$ are summarized as follows:

1. Convert $F(X)$ into its prime exclusive–or form by grouping every entry of $F(X)$ on the map an odd number of times.
2. Apply properties 1 through 6 to the prime exclusive–or form of $F(X)$; the result of $dF(X)/dx_i$ can be readily obtained.

The six method, which is probably the most convenient one, is to obtain the Boolean difference dF/dx_i by rotating the Karnaugh map of F about the axis of x_i using the following two rules.

RULE 1:

A 1 (minterm) is generated and added to the Karnaugh map of F if (1) it is a minor image of a 1 on the map with respect to the axis x_i, and (2) the cell where the 1 is to be added is not occupied by a 1 of the function.

RULE 2:

A 1 (minterm) of the function is deleted from the map if it is a minor image of another 1 of the function with respect to the axis x_i.

The map so obtained is the Karnaugh map of the Boolean difference dF/dx_i, from which the minimized dF/dx_i can be readily obtained.

The following example illustrate these six methods for obtaining the Boolean difference of a switching function.

Example 7.6.1

Find the Boolean difference of $F(X) = x_1x_2 + x_3$ with respect to x_1.

METHOD 1

$$\frac{dF}{dx_1} = \frac{d(x_3 + x_1x_2)}{dx_1} = x_3' \frac{d(x_1x_2)}{dx_1} \oplus (x_1x_2)' \frac{dx_3}{dx_1} \oplus \frac{dx_3}{dx_1} \cdot \frac{d(x_1x_2)}{dx_1} \qquad \text{(by property 5)}$$

$$= x_3' \frac{d(x_1x_2)}{dx_1} \qquad \text{(by property 7)}$$

$$= x_3' \left(x_1 \frac{dx_2}{dx_1} \oplus x_2 \frac{dx_1}{dx_1} + \frac{dx_1}{dx_1} \frac{dx_2}{dx_1} \right) \qquad \text{(by property 4)}$$

$$= x_3'x_2 \qquad \text{(by properties 7 and 8)}$$

METHOD 2

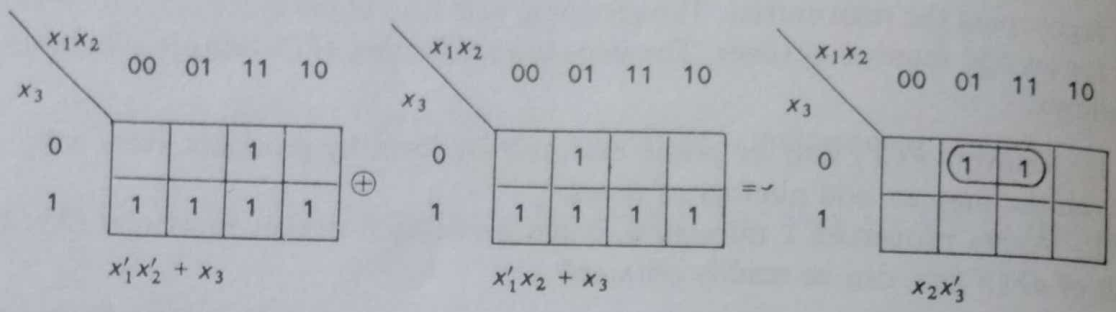
$$\frac{dF}{dx_1} = (1 \cdot x_2 + x_3) \oplus (0 \cdot x_2 + x_3) = (x_2 + x_3) \oplus x_3$$
$$= (x_2 + x_3)x_3' + (x_2 + x_3)'x_3 = x_2x_3'$$

METHOD 3

It is seen that $A(X) = x_3$, $B(X) = x_2$, and $C(X) = 0$. The $dF(x)/dx_1$ using method 3 is

$$\frac{dF(X)}{dx_1} = \overline{A(X)}[B(X) \oplus C(X)] = x_3'(x_2 \oplus 0) = x_3'x_2$$

METHOD 4



METHOD 5

From the Karnaugh map of $F(X) = x_1x_2 + x_3$ (see Fig. 7.6.1(a)), one prime EXCLUSIVE-OR form of $F(X)$ is

$$F(X) = x_1x_2x_3' \oplus x_3$$

It is easy to verify that $x_1x_2 + x_3 = x_1x_2x_3' \oplus x_3$. Using property 6 we obtain

$$\frac{dF(X)}{dx_1} = \frac{d(x_1x_2x_3')}{dx_1} \oplus \frac{dx_3}{dx_1} = \frac{d(x_1x_2x_3')}{dx_1} = x_2x_3'$$

METHOD 6

Referring to Fig. 7.6.1(b), one minterm (minterm 2, indicated by a **1**) is added and four minterms (minterms 1, 3, 5, and 7, indicated by 1) are deleted; thus the minimized $dF/dx_1 = x_2x_3'$.

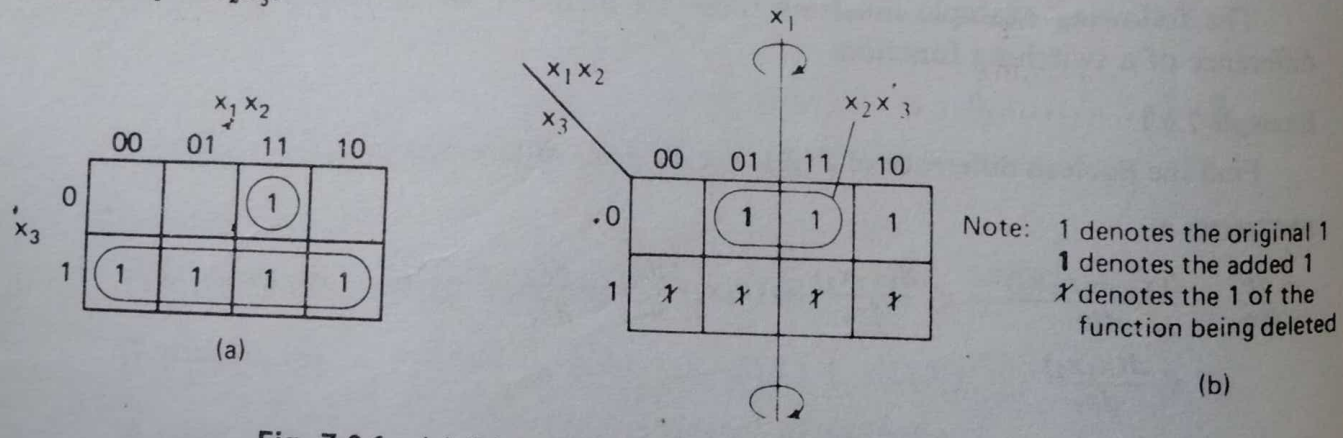


Fig. 7.6.1 (a) Prime EXLUSIVE-OR form of $F(X) = x_1x_2 + x_3$; (b) rotation of the Karnaugh map of F about x_1.

Derivation of a Complete Fault-Detection Test Set for a Single Fault To derive a test of a fault on line j, line j is "cut" and is considered as applied a "pseudo-input" x_j. The primary output z is then expressed in terms of the primary input variables $x_1, \ldots, x_n$, and this pseudo-input x_j [i.e., $z_{x_j} = F(x_1, \ldots, x_n, x_j)$]. It is noted that the actual value on j depends on the values $x_1, x_2, \ldots, x_n$, and we let this dependence relation be denoted by the function $x_j(x_1, \ldots, x_n)$. In order to find a test of the s-a-0 (or s-a-1) fault on line j, we must specify an input pattern applied to the primary input terminals such that the value of f depends on the value of x_j and that $X_j = 1$ (or 0) under the fault-free condition. Let a_i be the binary value of the primary input variable x_i. Then it is obvious that $(a_1, \ldots, a_n)$ is a test of the s-a-0 fault on line j if and only if

$$X_j(x_1, \ldots, x_n) \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j}\bigg|_{a_1, \ldots, a_n} = 1 \qquad (7.6.1)$$

Similarly, $(a_1, \ldots, a_n)$ is a test of the s-a-1 fault on line j if and only if

$$\overline{X_j(x_1, \ldots, x_n)} \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j}\bigg|_{a_1, \ldots, a_n} = 1 \qquad (7.6.2)$$

Thus we have the following theorem:

THEOREM 7.6.4

Let $x_i^0 = x_i'$ and $x_i^1 = x_i$. Let a_i be the binary value of input variable x_i. Then $(a_1, \ldots, a_n)$ is a test of the s-a-0 (or s-a-1) fault on line j if and only if $x_1^{a_1} \ldots x_n^{a_n}$ is a minterm in the canonical sum-of-products form of

$$X_j(x_1, \ldots, x_n) \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j} \qquad \left[\text{or } \overline{X_j(x_1, \ldots, x_n)} \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j}\right]$$

Following the above theorem, we have:

COROLLARY 7.6.1

The s-a-0 (or s-a-1) fault on line j is undetectable if and only if

$$X_j(x_1, \ldots, x_n) \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j} = 0 \qquad \left[\text{or } \overline{X_j(x_1, \ldots, x_n)} \frac{dF(x_1, \ldots, x_n, x_j)}{dx_j} = 0\right]$$

Example 7.6.2

Consider the circuit of Fig. 7.6.2. By using the truth-table method, we could determine that the faults e_0 and h_1 are undetectable. Now we would like to use the Boolean difference method described in Corollary 7.6.1.

The output functions expressed in terms of primary input variables and the pseudo-inputs e and h are

$$F(x_3, x_4, e) = (e + x_2) + x_3(x_3 + x_4)$$

and

$$F(x_1, x_2, h) = (x_1x_2 + x_2) + hx_3$$

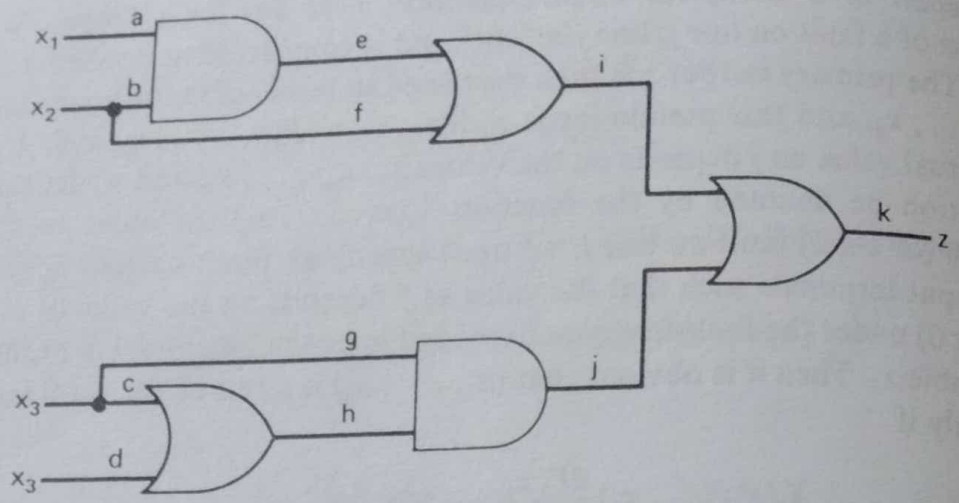


Fig. 7.6.2 Circuit of Example 7.6.2.

respectively, and the Boolean differences with respect to e and h are

$$\frac{dF(x_3, x_4, e)}{de} = [(1 + x_2) + x_3] \oplus [(0 + x_2) + x_3] = x_2'x_3'$$

and

$$\frac{dF(x_1, x_2, h)}{dh} = (x_2 + 1 \cdot x_3) \oplus (x_2 + 0 \cdot x_3) = x_2'x_3$$

It is easy to see that $X_e = x_1x_2$ and $\bar{X}_h = x_3'x_4'$. We find that

$$X_e \frac{dF(x_3, x_4, e)}{de} = 0 \quad \text{and} \quad \bar{X}_h \frac{dF(x_1, x_2, h)}{dh} = 0$$

According to Corollary 7.6.1, e_0 and h_1 are undetectable faults. We also find tha

$$\bar{X}_e \frac{dF(x_3, x_4, e)}{de} = (x_1x_2)'x_2'x_3' = x_1'x_2'x_3'$$

and

$$X_h \frac{dF(x_1, x_2, h)}{dh} = (x_3 + x_4)x_2'x_3 = x_2'x_3 + x_2'x_3x_4$$

which indicate that faults e_1 and h_0 are detectable and the complete test sets for detecting e_1 and h_0 are {0000 and 0001} and {0010, 0011, 1010, 1011}, respectively.

Derivation of Complete Fault-Detection Tests for Multiple Faults The derivation of tests for detecting multiple faults are derived from multiple Boolean differences. First let us derive the Boolean difference of a function $F(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ simultaneously with respect to two variables x_i and x_j, which we call a double Boolean difference.

DEFINITION 7.6.3

$$\frac{dF(X)}{d(x_ix_j)} = F(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_j', \ldots, x_n)$$

Note that

$$\frac{dF(X)}{d(x_i x_j)} \neq \frac{d}{dx_i}\frac{dF(X)}{dx_j}$$

Definition 7.6.4 defines independence as applied to the double Boolean difference.

DEFINITION 7.6.4

A Boolean function $F(X)$ is said to be *independent of two variables* x_i *and* x_j if and only if $F(X)$ is logically invariant under complementation of x_i and x_j [i.e., if $F(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = F(x_1, \ldots, x_i', \ldots, x_j', \ldots, x_n)$].

Now from Definition 7.6.3, we have Theorem 7.6 5.

THEOREM 7.6.5

A necessary and sufficient condition that a function $F(X)$ be independent of variables x_i and x_j is that $dF/d(x_i, x_j) = 0$.

Note that Definition 7.6.4 and Theorem 7.6.5 refer strictly to those cases when a multiple fault x_i and x_j occurs simultaneously. It differs from the convenient definition of multiple fault detection. In the latter case, not only do faults in x_i and x_j occur at the same time, but cases of fault in either x_i or x_j are also included. The following cases may be of interest.

CASE 1

$$\frac{dF(X)}{d(x_i x_j)} = F(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_j', \ldots, x_n)$$

Case 1 is the Boolean difference for studying the effect of multiple faults on a circuit.

CASE 2

$$\frac{dF(X)}{d(x_i \oplus x_j)} = \frac{dF(X)}{dx_i} + \frac{dF(X)}{dx_j}$$

Case 2 is the Boolean difference for studying the effect of a fault in either x_i or x_j.

CASE 3

$$\frac{dF(X)}{d(x_i + x_j)} = \frac{dF(X)}{d(x_i \oplus x_j)} + \frac{dF(X)}{d(x_i x_j)}$$

Case 3 is the Boolean difference for studying a fault in either x_i or x_j or both.

All the preceding double Boolean difference are readily extended to higher-order multiple Boolean differences.

CASE 1'

$$\frac{dF(X)}{d(x_i x_1 \ldots x_k)} = F(x_1, \ldots, x_i, \ldots, x_j, \ldots x_k, \ldots, x_n) \oplus F(x_1, \ldots, x_i', \ldots, x_j', \ldots, x_k', \ldots, x_n)$$

CASE 2′

$$\frac{dF(X)}{d(x_i \oplus x_j \oplus \cdots \oplus x_k)} = \frac{dF(X)}{dx_i} + \frac{dF(X)}{dx_j} + \cdots + \frac{dF(X)}{dx_k}$$

CASE 3′

$$\frac{dF(X)}{d(x_i + x_j + \cdots + x_k)} = \frac{dF(X)}{d(x_i \oplus x_j \oplus \cdots \oplus x_k)} + \frac{dF(X)}{d(x_i x_j \cdots x_k)}$$

Parallel to Theorem 7.6.4 and Corollary 7.6.1, we have the following theorem and corollary for test derivation for detecting multiple faults.

THEOREM 7.6.6

Let $x_i^0 = x_i'$ and $x_i^1 = x_i$. Let a_i be the binary value of input variable x_i. Then $(a_1, \ldots, a_n)$ is a test of the s-a-0 (or s-a-1) faults on lines $i, j, \ldots, k$ simultaneously if and only if $x_1^{a_1} \cdots x_n^{a_n}$ is a minterm in the standard sum of

$$X_{i,j,\ldots,k}(x_1, \ldots, x_n) \frac{dF(X)}{d(x_i, x_j, \ldots, x_k)} \qquad \left[\text{or } \overline{X_{ij\ldots k}(x_1, \ldots, x_n)} \frac{dF(X)}{d(x_i x_j \cdots x_k)}\right]$$

COROLLARY 7.6.2

The s-a-0 (s-a-1) faults simultaneously occurring on lines $i, j, \ldots,$ and k are undetectable if and only if

$$X_{ij\ldots k}(x_1, \ldots, x_n) \frac{dF(X)}{d(x_i x_j \cdots x_k)} = 0 \qquad \left[\text{or } \overline{X_{ij\ldots k}(x_1, \ldots, x_n)} \frac{dF(X)}{d(x_i x_j \cdots x_k)} = 0\right]$$

It is important to point out that

$$\frac{dF(X)}{d(x_i x_j)} \neq$$
$$F(x_1, \ldots, x_i = 1, \ldots, x_j = 1, \ldots, x_n) \oplus F(x_1, \ldots, x_i = 0, \ldots, x_j = 0, \ldots, x_n)$$

This may be seen from the following example.

Example 7.6.3

Consider the OR gate of Fig. 7.6.3. The output function is $F(x_1, x_2, x_3, x_4) = x_1 + x_2 + x_3 + x_4$. Suppose that we want to derive a test for detecting the multiple fault x_1 s-a-0

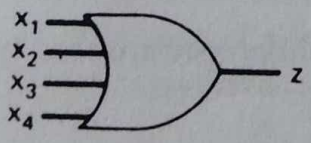


Fig. 7.6.3 OR gate.

and x_2 s-a-0 simultaneously. Then we need first to find the second-order Boolean difference with respect to x_1 and x_2:

$$\frac{dF}{d(x_1x_2)} = (x_1 + x_2 + x_3 + x_4) \oplus (x_1' + x_2' + x_3 + x_4)$$
$$= x_1x_2x_3'x_4' + x_1'x_2'x_3'x_4'$$

The test for detecting the fault x_1 s-a-0 and x_2 s-a-0 may be obtained from

$$X_{x_1x_2} \cdot \frac{dF}{d(x_1x_2)} = x_1x_2(x_1x_2x_3'x_4' + x_1'x_2'x_3'x_4') = x_1x_2x_3'x_4'$$

which agrees with our intuition that when the input combination 1100 is applied to the OR gate, the faults x_1 s-a-0 and x_2 s-a-0 may be detected by observing the gate's output. It is fault-free if the output is 1 and has the fault x_1 s-a-0 and x_2 s-a-0 if the output is 0.

Fault-Detection Test-Generation Algorithm Using Boolean Differences

THEOREM 7.6.7

Let f be a switching function realized by a combinational circuit. Let j be an internal line and i be either an internal line or a primary input line of the circuit such that every path from i to the primary output line passes through j. Then

$$\frac{df}{dx_i} = \frac{df}{dx_j}\frac{dx_j}{dx_i} \tag{7.6.3}$$

where x_i and x_j are the variables representing the logic values on lines i and j, respectively.

Proof: Let $x_1, \ldots, x_n$ be the primary input variables. When all the paths from line i to the primary output line passing through line j, we can write

$$x_j = J(x_1, \ldots, x_n, x_i) \tag{7.6.4}$$

and

$$\begin{aligned} f &= F_j(x_1, \ldots, x_n, x_j) \\ &= F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i)] \\ &= F_i(x_1, \ldots, x_n, x_i) \end{aligned} \tag{7.6.5}$$

Consider the following three cases:

CASE 1

Suppose that $dx_j/dx_i = 1$ and $df/dx_j = 1$. Then

$$\begin{aligned} \frac{df}{dx_i} &= F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i)] \\ &\quad \oplus F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i')] \\ &= F_j(x_1, \ldots, x_n, \delta) \oplus F_j(x_1, \ldots, x_n, \delta) \\ &= 1 \end{aligned} \tag{7.6.6}$$

CASE 2

Suppose that $dx_j/dx_i = 0$. Then

$$\begin{aligned}\frac{df}{dx_i} &= F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i)] \\ &\quad \oplus F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i')] \\ &= F_j(x_1, \ldots, x_n, \delta) \oplus F_j(x_1, \ldots, x_n, \delta) \\ &= 0\end{aligned} \tag{7.6.7}$$

CASE 3

Suppose that $df/dx_j = 0$. Then

$$\begin{aligned}\frac{df}{dx_i} &= F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i)] \\ &\quad \oplus F_j[x_1, \ldots, x_n, J(x_1, \ldots, x_n, x_i')] \\ &= F_j(x_1, \ldots, x_n, \delta_1) \oplus F_j(x_1, \ldots, x_n, \delta_2) \\ &= 0\end{aligned} \tag{7.6.8}$$

It is noted that each of δ, δ_1, and δ_2 can be either 0 or 1. Combining the above three cases, we have Eq. (7.6.3). ∎

To avoid ambiguity, we define a line i of a combinational circuit as a *fan-out line* if line i is either a primary input line or a gate-output line such that line i is connected to more than one gate-input terminal. For example, line 2 of the circuit shown in Fig. 7.6.4 is a fan-out line.

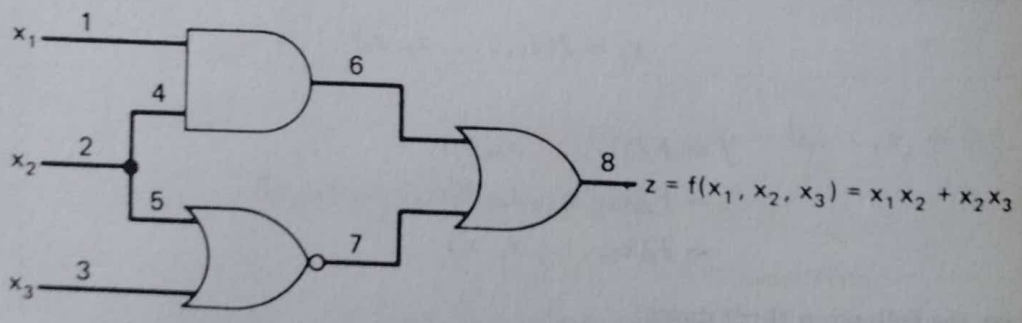
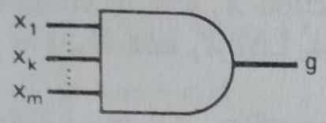
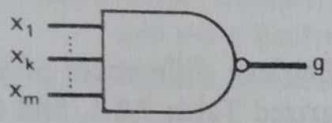
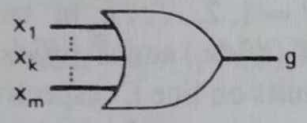
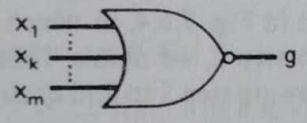
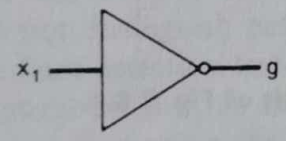
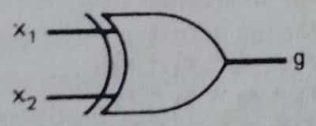


Fig. 7.6.4 Circuit to illustrate the algorithm for the genera of tests based on Boolean differences.

The algorithm for generating the complete test sets of all s-a-0 and s-a-1 single faults using the Boolean difference is described as follows:

Step 1 Number all the lines of the logic circuit under consideration as follows:
(1) Number all the lines using positive integers in ascending order starting with prime inputs from top to bottom.
(2) Draw the circuit diagram in such a way that all the gates are lined up into levels of

TABLE 7.6.1 Boolean Difference of Six Most Commonly Used Gates

Gate	Boolean difference
 AND gate	$\frac{dg}{dx_k} = x_1 x_2 \cdots x_{k-1} x_{k+1} \cdots x_m$
 NAND gate	
 OR gate	$\frac{dg}{dx_k} = x_1' x_2' \cdots x_{k-1}' x_{k+1}' \cdots x_m'$
 NOR gate	
 NOT gate	$\frac{dg}{dx_1} = 1$
 EXCLUSIVE-OR gate	$\frac{dg}{dx_1} = \frac{dg}{dx_2} = 1$

gates. The NOT gates, if any, are drawn into separate columns from those of other types. But a NOT gate will be treated the same as other types if its input is one of the fan-out branches.

(3) Label the lines from left to right. The last ones to be numbered are the primary output lines.

Step 2 Starting from the line with the smallest number 1 (one of the primary input lines) to the line with the largest number l (the primary output line), where l is the number of lines in the circuit, express the function X_j and its complement $\bar{X}_j$ of each line j in terms of the primary input variables. List X_j and $\bar{X}_j$, $j = 1, 2, \ldots, l$.

Step 3 Find the Boolean differences $df/dx_j = dF(x_1, \ldots, x_n, x_j)/dx_j$ according to (2), where f is the logic function realized by the circuit and j ranges over all fan-out lines.

Step 4 Based on Theorem 7.6.7 and the Boolean differences of various gate outputs with respect to their gate inputs as summarized Table 7.6.1, find the Boolean differences df/dx_j, where j starts from the largest number to the smallest number that is not a fan-out line. List all the df/dx_j, $1, 2, \ldots, l$.

Step 5 Express $X_j(df/dx_j)$, and $\bar{X}_j(df/dx_j)$, $j = 1, 2, \ldots, l$, in terms of the primary input variables. Then the sets of terms of $X_j(df/dx_j)$ and $\bar{X}_j(df/dx_j)$ represent the complete test sets of the s-a-0 and s-a-1 single faults on line j, respectively.

Example 7.6.4

This algorithm is now applied to the circuit shown in Fig. 7.6.4, in which each line is labeled using the rules described in step 1. According to step 2, we obtain Table 7.6.2. It is seen that there is only one fan-out line that is line 2. Following step 3 to find df/dx_2, we obtain

$$\begin{aligned}\frac{df}{dx_2} &= (x_1 \cdot 0 + 0' x_3') \oplus (x_1 \cdot 1 + 1' x_3') \\ &= x_1 x_3 + x_1' x_3'\end{aligned}$$

TABLE 7.6.2 X_j and $\bar{X}_j$ of the Circuit of Fig. 7.6.4

$X_j = x_j$ and $\bar{X}_j = x_j'$, $j = 1, 2, 3$	
$X_4 = x_2$	$\bar{X}_4 = x_2'$
$X_5 = x_2$	$\bar{X}_5 = x_2'$
$X_6 = x_1 x_4 = x_1 x_2$	$\bar{X}_6 = x_1' + x_4' = x_1' + x_2'$
$X_7 = x_3' x_5' = x_2' x_3'$	$\bar{X}_7 = x_3 + x_5 = x_2 + x_3$
$X_8 = x_6 + x_7 = x_1 x_2 + x_2' x_3'$	$\bar{X}_8 = x_6' x_7' = x_1' x_2 + x_1' x_3 + x_2' x_3$

According to step 4, find df/dx_j in the order of $j = 8, 7, \ldots, 2, 1$. All the df/dx_j, $j = 1, 2, \ldots 8$, are listed in Table 7.6.3. According to step 4, the complete test set of the s-a-0 or s-a-1 fault on line j is obtained by taking $X_j(df/dx_j)$ or $\bar{X}_j(df/dx_j)$, respectively.

The complete test sets for all s-a-0 and s-a-1 faults of this circuit are shown in Table 7.6.4.

TABLE 7.6.3 df/dx_j **of the Circuit of Fig. 7.6.4**

$$\frac{df}{dx_8} = 1$$
$$\frac{df}{dx_6} = x_7' = x_2 + x_3$$
$$\frac{df}{dx_4} = \frac{df}{dx_6}\frac{dx_6}{dx_4} = x_1x_2 + x_1x_3$$
$$\frac{df}{dx_2} = x_1x_3 + x_1'x_3'$$
$$\frac{df}{dx_7} = x_6' = x_1' + x_2'$$
$$\frac{df}{dx_5} = \frac{df}{dx_7}\frac{dx_7}{dx_5} = x_1'x_3' + x_2'x_3'$$
$$\frac{df}{dx_3} = \frac{df}{dx_6}\frac{dx_7}{dx_3} = x_2'$$
$$\frac{df}{dx_1} = \frac{df}{dx_6}\frac{dx_6}{dx_1} = x_2$$

TABLE 7.6.4 Complete Test Sets for Detecting All s-a-0 and s-a-1 Faults of the Circuit of Fig. 7.6.4

$X_1\frac{df}{dx_1} = x_1x_2 \longrightarrow T(a_0) = \{6, 7\}$
$X_2\frac{df}{dx_2} = x_1x_2x_3 + x_1'x_2x_3' \longrightarrow T(b_0) = \{2, 7\}$
$X_3\frac{df}{dx_3} = x_2'x_3 \longrightarrow T(c_0) = \{1, 5\}$
$X_4\frac{df}{dx_4} = x_1x_2 \longrightarrow T(d_0) = \{6, 7\}$
$X_5\frac{df}{dx_5} = x_1'x_2x_3' \longrightarrow T(e_0) = \{2\}$
$X_6\frac{df}{dx_6} = x_1x_2 \longrightarrow T(f_0) = \{6, 7\}$
$X_7\frac{df}{dx_7} = x_1'x_2' \longrightarrow T(g_0) = \{0, 4\}$
$X_8\frac{df}{dx_8} = x_1x_2 + x_2'x_3' \longrightarrow T(h_0) = \{0, 4, 6, 7\}$

$\bar{X}_1\frac{df}{dx_1} = x_1'x_2 \longrightarrow T(a_1) = \{2, 3\}$
$\bar{X}_2\frac{df}{dx_2} = x_1x_2'x_3 + x_1'x_2'x_3' \longrightarrow T(b_1) = \{0, 5\}$
$\bar{X}_3\frac{df}{dx_3} = x_2'x_3' \longrightarrow T(c_1) = \{0, 4\}$
$\bar{X}_4\frac{df}{dx_4} = x_1x_2'x_3 \longrightarrow T(d_1) = \{5\}$
$\bar{X}_5\frac{df}{dx_5} = x_1'x_2' \longrightarrow T(e_1) = \{0, 4\}$
$\bar{X}_6\frac{df}{dx_6} = x_1'x_2 + x_1'x_3 + x_2'x_3\ T(f_1) = \{1, 2, 3, 5\}$
$\bar{X}_7\frac{df}{dx_7} = x_1'x_2 + x_1'x_3 + x_2'x_3 \longrightarrow T(g_1) = \{1, 2, 3, 5\}$
$\bar{X}_8\frac{df}{dx_8} = x_1'x_2 + x_1'x_3 + x_2'x_3 \longrightarrow T(h_1) = \{1, 2, 3, 5\}$

Owing to the fact that the Boolean difference of a function with respect to a variable does not distinguish between the effects of that variable changing from a 0 to a 1, the fault-detection test-generation method using Boolean differences, like many other methods, such as the fault-table method, the path-sensitizing method, the ENF method, and so on, does not give information about the effect of the variable perturbation on the output. But it has been shown that the methods (basically, the methods using the Karnaugh map) described in Sections 7.4 and 7.5 not only generate tests for fault detection but also provide the information about whether the output changes from a 1 to a zero or a zero to a 1. In the following it will be shown that, with slight modification, the fault-detection test-generation method using Boolean differences described above may also provide this information whenever it is needed.

The problem of determining what the correct output should be for a given test can be solved by intersecting the tests for a s-a-1 and a s-a-0 with both the function and its inverse. We now define two equations that define completely the test for an individual error:

$$\nabla_{x_i}F = \left(x_i\frac{dF}{dx_i}\right)[F, \bar{F}] \tag{7.6.9}$$

$$\Delta_{x_i}F = \left(x_i' \frac{dF}{dx_i}\right)[F, \bar{F}] \tag{7.6.10}$$

In these two equations, the tests for the variables x_i s-a-1 and s-a-0 are intersected with the ordered pair $[F, \bar{F}]$. Equation (7.6.9) gives the tests that will detect a s-a-0 fault in variable x_i, and it partitions these tests into tests corresponding to a 1 or a 0 output for the overall circuit output F. Equation (7.6.10) yields the tests for variable i s-a-1 and partitions them according to their effect on the circuit output. Since each block and line within the total circuit must be tested for a s-a-1 and s-a-0 condition, a complete test for a circuit will consist of one test from Eq. (7.6.9) and one from Eq. (7.6.10), where the index i represents all input lines, internal blocks, and interconnecting lines. All the circuits considered in this section are assumed to have no redundancy.

Example 7.6.5

To illustrate Eqs. (7.6.9) and (7.6.10), let us consider the circuit of Fig. 7.6.4. The complete test sets for detecting all s-a-0 and s-a-1 faults of the circuit by Boolean differences were given in Table 7.6.4, from which we can easily obtain $\nabla_{x_i}F$ and $\Delta_{x_i}F$ for all x_i shown in Table 7.6.5. Take the s-a-0 test for detecting fault a_0, for example, which consists of 110 and 111. From Table 7.6.5 it is seen that $\nabla_{x_1}F = (x_1x_2, 0)$, which means that with the application

TABLE 7.6.5 $\nabla_{x_i}F$ and $\Delta_{x_i}F$ of the Circuit of Fig. 7.6.4(b)

$\nabla_{x_i}F = \left(x_i \frac{dF}{dx_i}\right) \cdot [F, \bar{F}]$	$\Delta_{x_i}F = \left(\bar{x}_i \frac{dF}{dx_i}\right) \cdot [F, \bar{F}]$
$\nabla_{x_1}F = (x_1x_2, 0)$	$\Delta_{x_1}F = (0, x_1'x_2)$
$\nabla_{x_2}F = (x_1x_2x_3, x_1'x_2x_3')$	$\Delta_{x_2}F = (x_1'x_2'x_3', x_1x_2'x_3)$
$\nabla_{x_3}F = (0, x_2'x_3)$	$\Delta_{x_3}F = (x_2'x_3', 0)$
$\nabla_{x_4}F = (x_1x_2, 0)$	$\Delta_{x_4}F = (0, x_1x_2'x_3)$
$\nabla_{x_5}F = (0, x_1'x_2x_3')$	$\Delta_{x_5}F = (x_2'x_3', 0)$
$\nabla_{x_6}F = (x_1x_2, 0)$	$\Delta_{x_6}F = (0, x_1'x_2 + x_2'x_3 + x_1'x_3)$
$\nabla_{x_7}F = (x_2'x_3', 0)$	$\Delta_{x_7}F = (0, x_1'x_2 + x_2'x_3 + x_1'x_3)$
$\nabla_{x_8}F = (x_1x_2 + x_2'x_3', 0)$	$\Delta_{x_8}F = (0, x_1'x_2 + x_2'x_3 + x_1'x_3)$

of input combination 110 or 111, the output is 1 if the circuit does not have any fault and is 0 if the circuit has fault a_0. The reader can easily verify this from the circuit. Similarly, for the s-a-1 tests 010 and 011 for detecting fault a_1, the $\Delta_{x_1}F = (0, x_1'x_2)$ indicates that with the application of input combination 010 or 011, the output is 0 if the circuit does not have any fault and is 1 if the circuit has fault a_1.

As a final remark, the Boolean-difference method presented in this section can easily be extended to fault detection of multioutput circuits (problems 11 and 12).

Exercise 7.6

1. Supply the proofs for properties 1–10 of the Boolean difference.
2. Let $F(X) = \prod_{k=1}^{m} G_k(X)$, where $X = (x_1, \ldots, x_i, \ldots, x_n)$.

Prove that

$$\frac{dF(X)}{dx_i} = \left[\prod_{k=1}^{m} G_k(X) + \prod_{k=1}^{m} G_k(X^*)\right] \cdot \sum_{k=1}^{m} \frac{dG_k(X)}{dx_i}$$

where $X^* = (x_1, \ldots, x_i', \ldots, x_n)$.

3. Let $F(X) = \sum_{k=1}^{m} G_k(X)$. Prove that

$$\frac{dF(X)}{dx_i} = \left[\prod_{k=1}^{m} \overline{G_k(X)} + \prod_{k=1}^{m} \overline{G_k(X^*)}\right] \cdot \sum_{k=1}^{m} \frac{dG_k(X)}{dx_i}$$

where X and X^* are defined as in problem 2.

4. Find $dF(X)/dx_1$ for
   (a) $F(X) = x_1x_2 + x_2x_3 + x_1x_3$
   (b) $F(X) = (x_1 + x_2)(x_2 + x_3)(x_1 + x_3)$
   using the six methods given in this section.

5. For the same functions of problem 4, find $dF(X)/dx_1$ using the formulas given in problems 2 and 3.

6. Derive a complete fault-detection test set for the following faults of the NAND circuit in Fig. P7.6.6:
   (a) Single fault: line α s-a-0.
   (b) Single fault: line β s-a-1.
   (c) Single fault: line γ s-a-0.
   (d) Multiple fault: line α s-a-0, line β s-a-1.
   (e) Multiple fault: line β s-a-1, line γ s-a-0.

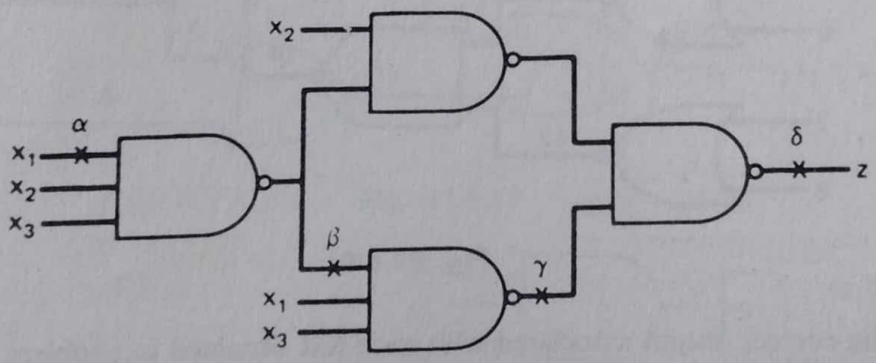


Fig. P7.6.6

7. Find a complete fault-detection test set for the following faults of the NOR circuit of Fig. P7.6.7:
   (a) Single fault: line *a* s-a-0.
   (b) Multiple fault: line *b* s-a-1 and line *c* s-a-0.
   (c) Multiple fault: line *d* s-a-0, line *e* s-a-1, and line *f* s-a-0.

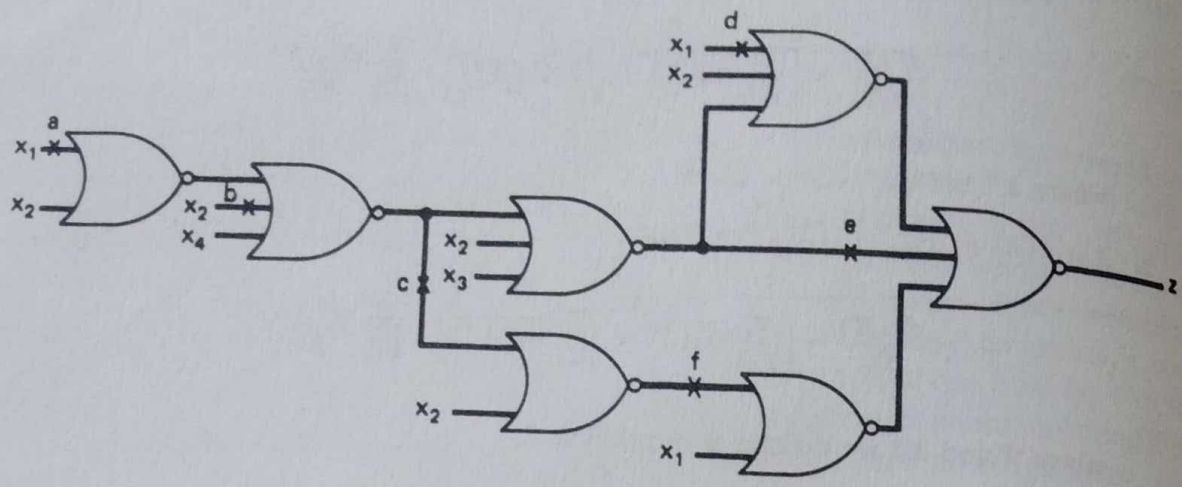


Fig. P7.6.7

8. Construct a complete test set for detecting each single fault in the irredundant tree-like circuit of Fig. P7.6.8 using the method described in this section.

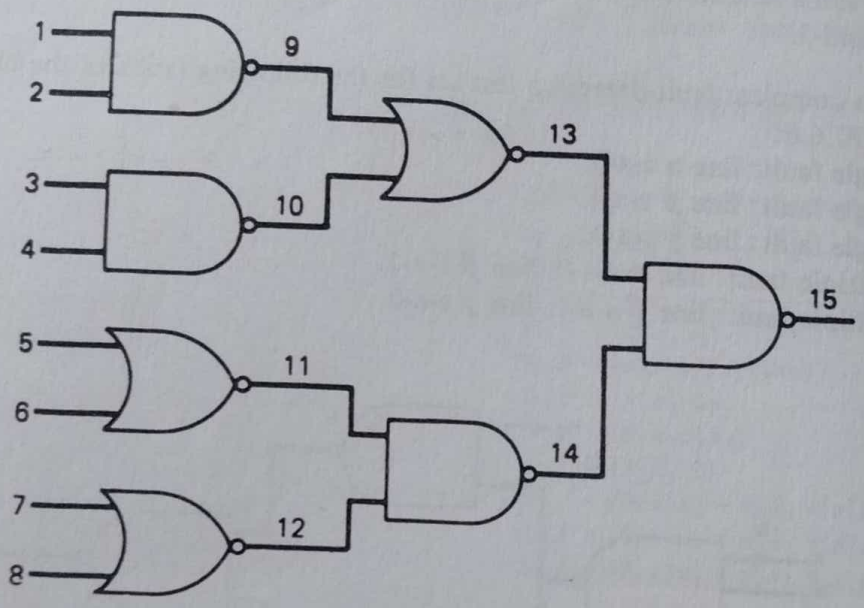


Fig. P7.6.8

9. Find the correct output associated with each test obtained in problem 8.

10. Find a complete fault-detection test for each single fault of the irredundant circuit of Fig. P7.6.10.

11. Consider the fault detection of the multiple-output circuit of Fig. P7.6.11.
(a) Construct dz_i/dx_j of the circuit.
(b) Construct the complete sets for detecting s-a-0 and s-a-1 faults on lines 1, 2, 3, and 4.

12. Use the Boolean difference method to construct complete tests for detecting single faults for the ir undant multiple-output circuit of Fig. P7.6.12.

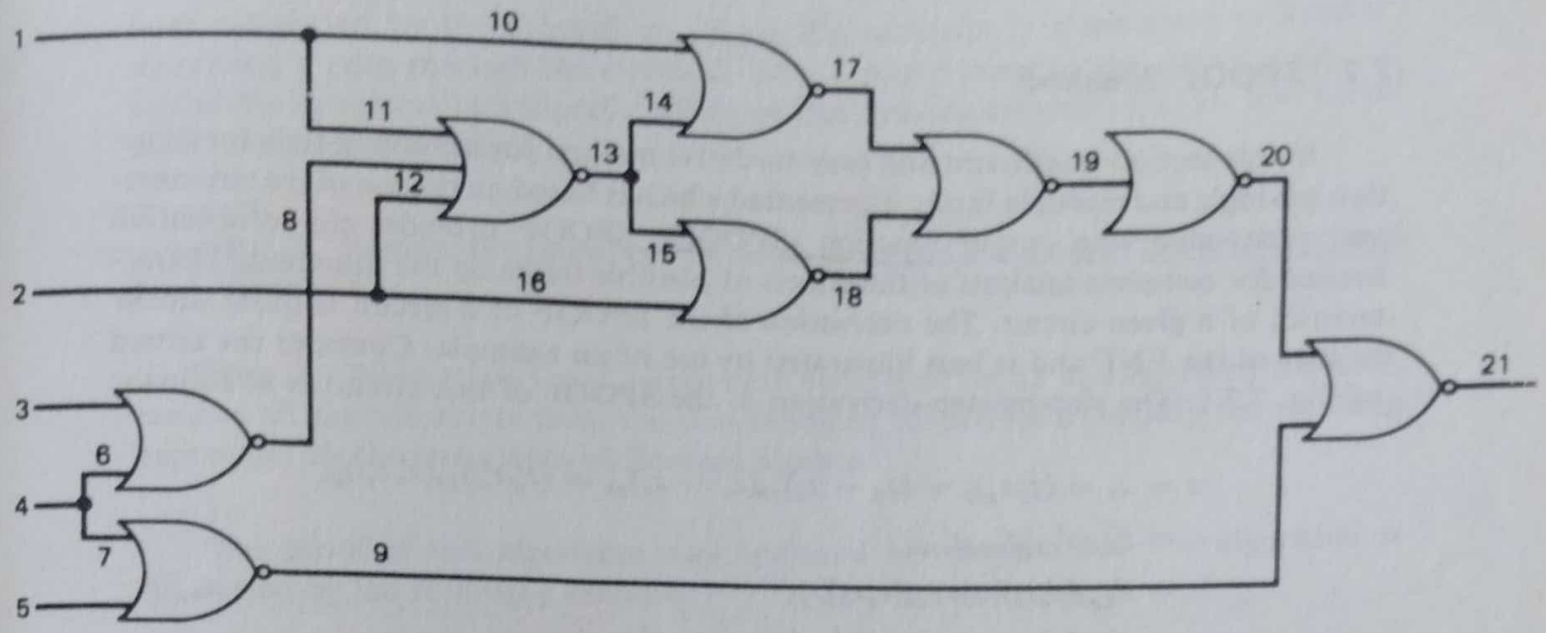


Fig. P7.6.10

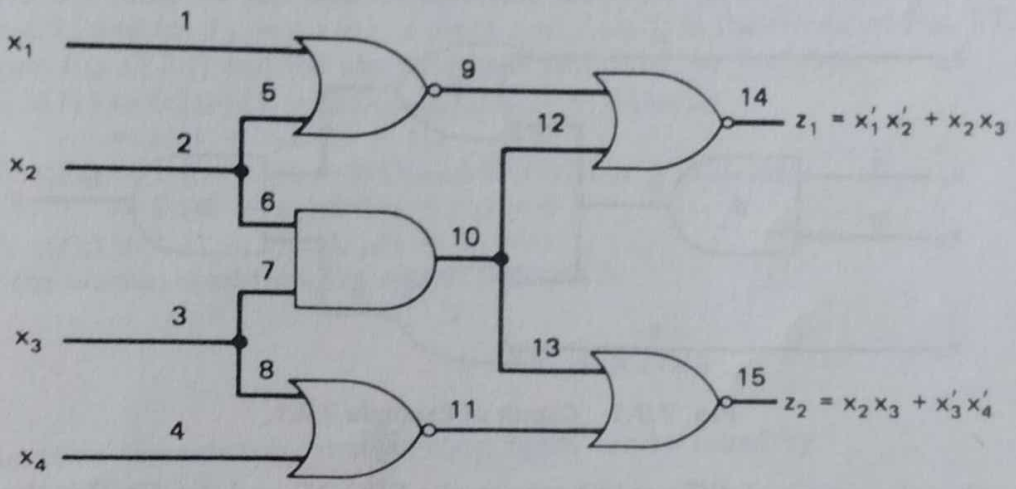


Fig. P7.6.11

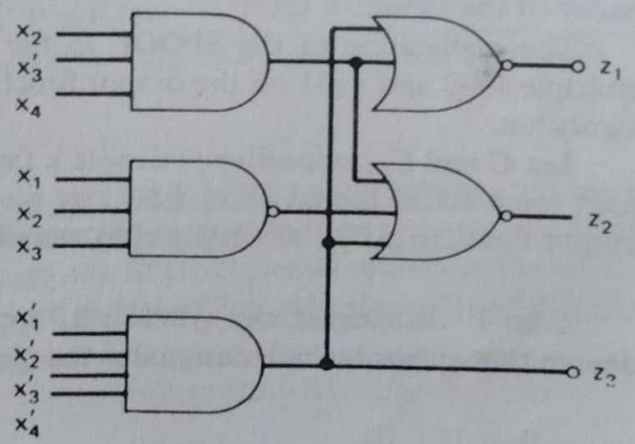


Fig. P7.6.12

7.7 SPOOF Method

In this section an efficient and easy-to-derive method for obtaining tests for detection of single and multiple faults is presented which is based on the use of the structure- and *p*arity-*o*bserving *o*utput *f*unction (SPOOF). SPOOF provides the information needed for complete analysis of the effects of possible faults on the functional characteristics of a given circuit. The derivation of the SPOOF of a circuit is quite similar to that of the ENF and is best illustrated by use of an example. Consider the circuit of Fig. 7.7.1. The step-by-step derivation of the SPOOF of this circuit is as follows:

$$
\begin{aligned}
z = z_h &= (z_f z_g)_h = (z_d + z_c)'_{fh}(z_e + z_c)'_{gh} = (z'_{d'}z'_{c'})_{fh}(z'_{e'}z'_{c'})_{gh} \\
&= z'_{d'fh}z'_{c'fh}z'_{e'gh}z'_{c'gh} \\
&= z'_{d'gh}(z_a z_b)'_{c'fh}z'_{e'gh}(z_a z_b)'_{c'gh} \\
&= z'_{d'gh}(z'_{a'c'fh} + z'_{b'c'fh})z'_{e'gh}(z'_{a'c'gh} + z'_{b'c'gh}) \\
&= (x'_3)_{d'fh}[(x'_1)_{a'c'fh} + (x'_2)_{b'c'fh}](x'_4)_{e'gh}[(x'_1)_{a'c'gh} + (x'_2)_{b'c'gh}] \qquad (7.7.1)
\end{aligned}
$$

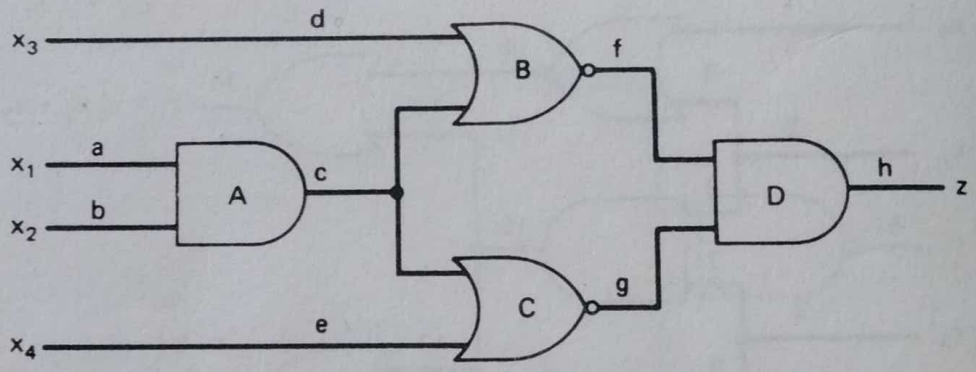


Fig. 7.7.1 Circuit of Example 7.7.1.

It is seen that the essential difference between the SPOOF and the ENF is the presence or absence of primes over each of the subscript symbols of each literal to indicate the parity of the signal.

The application of the SPOOF to the analysis of the effect of any single and multiple s-a-0 and s-a-1 on the output function of a circuit is based on the following algorithm.

Let C and C_F, respectively, denote a fault-free circuit and a faulty circuit with a fault (or a set of faults) F: a_1 s-a-i_1, a_2 s-a-i_2, . . . , a_m s-a-i_m, where $i_k \in \{0, 1\}$. The output function $z(C_F)$, or simply $z(F)$, may be determined as follows.

Step 1 Whenever any symbols a_k appear as a subscript to a literal in z, let a_j denote that symbol which designates the closest line to the output terminal of all the

lines designated by the symbols a_k. Since the subscript is a sequence of symbols describing a path through the circuit C from a prime input to the circuit output, if any of the a_k appear in a literal's subscript, the symbol designated a_j for that literal must necessarily be unique.

Step 2 Replace the literal by i_j if no prime appears over a_j in the subscript, and by i'_j otherwise.

Step 3 When steps 1 and 2 have been completed for all a_k affected by the fault, remove all the subscripts from the thus-modified SPOOF and simplify the resulting expression by the techniques of Boolean algebra.

The proof of this algorithm may be found in reference 21. The algorithm is illustrated by the following example.

Example 7.7.1

Find the complete test sets for detecting faults for (a) $F_1 = a$ s-a-0; (b) $F_2 = d$ s-a-0 and e s-a-1; and (c) $F_3 = a$ s-a-1, c s-a-0, and f s-a-1, in the circuit of Fig. 7.7.1.

From Eq. (7.7.1) and the algorithm just described, we find that

(a) $z(F_1) = (x'_3)_{d'fh}[1 + (x'_2)_{b'c'fh}](x'_4)_{e'gh}[1 + (x'_2)_{b'c'gh}]$
$= x'_3(1 + x'_2)x'_4(1 + x'_2) = x'_3x'_4$

(b) $z(F_2) = 1 \cdot [(x'_1)_{a'c'fh} + (x'_2)_{b'c'fh}] \cdot 0 \cdot [(x'_1)_{a'c'gh} + (x'_2)_{b'c'gh}]$
$= 1 \cdot (x'_1 + x'_2) \cdot 0 \cdot (x_1 + x_2) = 0$

(c) $z(F_3) = 1 \cdot (1 + 1)(x'_4)_{e'gh}(1 + 1) = x'_4$

Under the normal condition, the output function is

$$z = (x'_1 + x'_2)x'_3x'_4$$

The complete test sets for detecting these faults can be found by

$$T(F_1) = (x'_1 + x'_2)x'_3x'_4 \oplus x'_3x'_4 = x_1x_2x'_3x'_4 \longrightarrow T(F_1) = \{1100\}$$

$$T(F_2) = (x'_1 + x'_2)x'_3x'_4 \oplus 0 = (x'_1 + x'_2)x'_3x'_4 \longrightarrow T(F_2) = \{0000, 0100, 1000\}$$

$$T(F_3) = (x'_1 + x'_2)x'_3x'_4 \oplus x'_4 = x_1x_2x'_4 + x_3x'_4 \longrightarrow T(F_3) = \{1100, 1110, 0010, 0110, 1010\}$$

As a final remark, recall that if both a variable and its complement are contained in a term in the ENF, that term may be discarded (see the fourth characteristic of the ENF described in Section 7.3). Although the SPOOF greatly resembles the ENF, this rule is not applicable to the SPOOF. As a matter of fact, no terms of the SPOOF may be discarded under any circumstances.

Exercise 7.7

1. For the circuit in Fig. P7.7.1, find the complete set of tests for detecting the following faults:
   (a) Single fault α s-a-0.
   (b) Multiple fault α s-a-0 and β s-a-1.
   (c) Multiple fault α s-a-0, β s-a-1, γ s-a-0, and δ s-a-1.
   Use (1) the path-sensitizing method, (2) the ENF method, and (3) the SPOOF method.

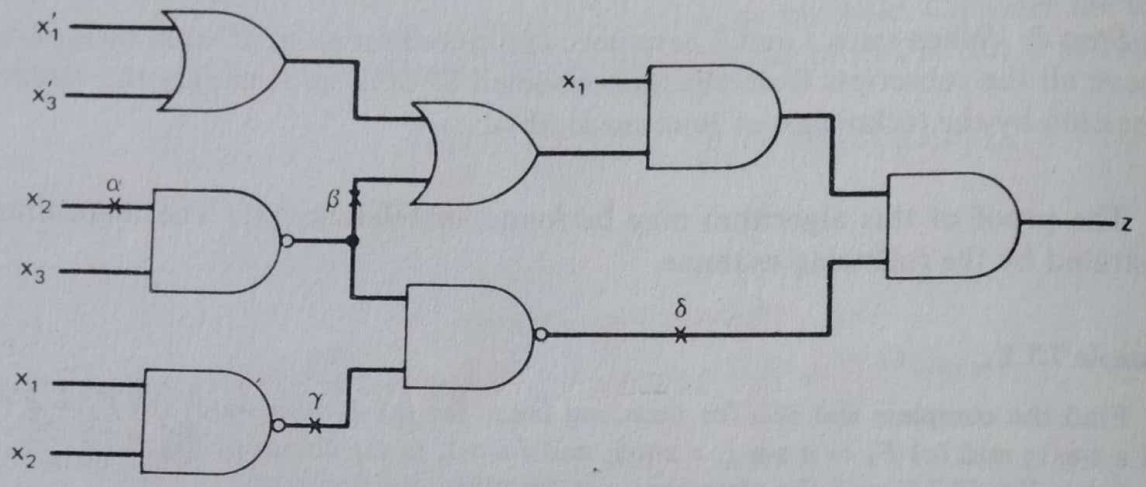


Fig. P7.7.1

2. For the following combinational circuit, find the complete set of tests to detect the multiple fault, all five lines α, β, γ, δ, and ϵ being s-a-0 using the SPOOF method.

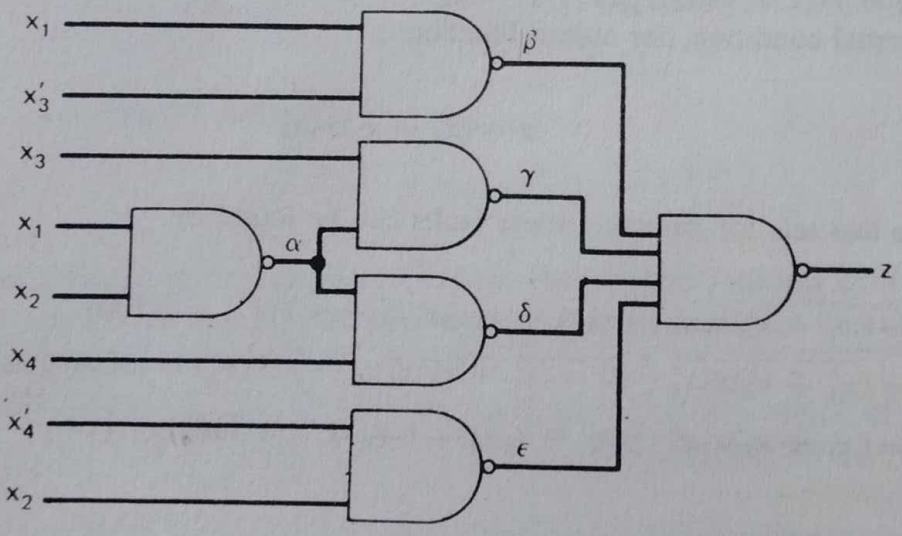


Fig. P7.7.2

3. (a) The normal circuit of Fig. P7.7.3 is a realization of the EXCLUSIVE-OR gate. Suppose that, owing to the presence of single faults in the circuit, the output of the circuit is changed from $z = x_1 \oplus x_2$ to $z = x_1' x_2$. Show that the multiple fault line b s-a-1, line c s-a-1, and line e s-a-1 is the only fault that can cause this change.
   (b) Find all the logic faults that can cause the circuit output to become $z = x_1' + x_2'$.

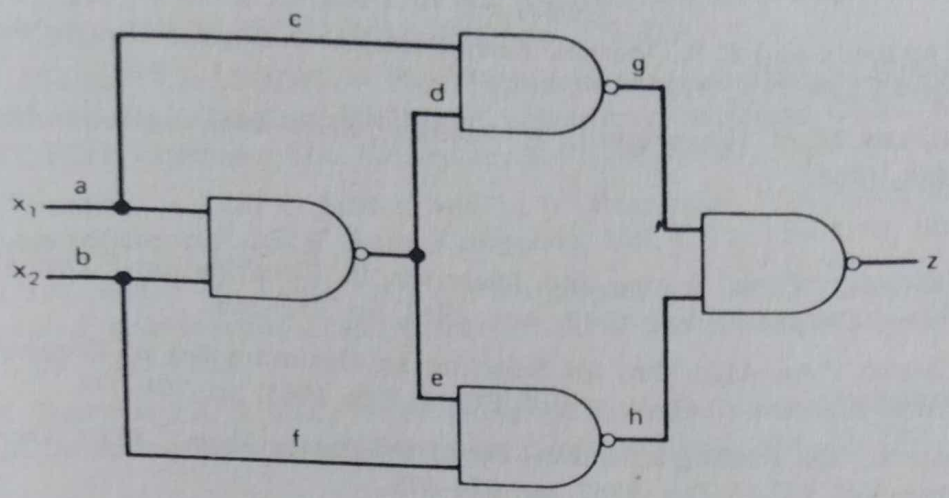


Fig. P7.7.3

Bibliographical Remarks

Three books (references 1–3) on fault detection and diagnosis in digital circuits and systems and two *IEEE Transactions on Computers* special issues on fault-tolerant computing (references 30 and 31) are excellent references for most material covered in this chapter. Readers interested in fault testing and diagnosis in combinational circuits are advised to read them. In recent years interest in this subject has been growing, which results in the publication of a large number of technical papers and the presentation of many papers at conferences. We list here only those which are referred to in the text.

The fault-table method is extensively treated by Kautz (reference 4), Chang (reference 5), and Hadlock (reference 6) .The path-sensitizing and ENF methods are contained in the article by Armstrong (reference 7). The Karnaugh map and tabular method was discovered by Kohavi and Spires (reference 8) and Bearnson and Carroll (reference 9) independently. The ENF–Karnaugh map method is also due to Kohavi and Spires (reference 8). Many techniques for deriving and selecting single-fault diagnostic tests can be found in references 10–16. The problem of fault equivalence in combinational circuits was studied by McCluskey and Clegg (reference 17). Two excellent papers on the analysis of multiple faults in combinational circuits are those of Gault et al. (reference 18) and Bossen and Hong (reference 19). References 20 and 21 are the original papers of the SPOOF method, and references 22–25 are references for the Boolean difference method. A computer-oriented fault-detection test generation algorithm known as the *D*-algorithm which is widely used in industry for large combinational circuits can be found in references 26–29. This algorithm is derived based on the *D*-Calculus (reference 26) and has been implemented in APL language (references 28 and 29). The latest versions of the computer programs are called DALG-II (*D*-Algorithm Version II) and TEST–DETECT (reference 29). An application of the *D*-algorithm to the fault location in large combinational circuits was investigated by Su and Cho (reference 30).

References

Books

1. H. Y. Chang, E. G. Manning, and G. Metze, *Fault Diagnosis of Digital Systems*, Wiley, New York, 1970.