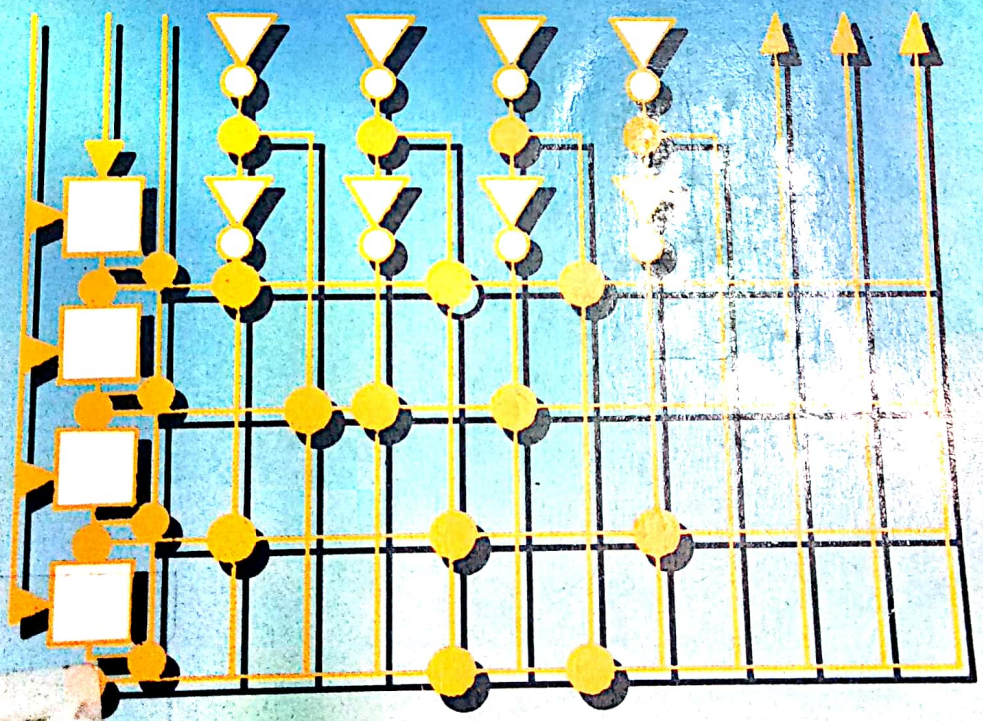
Eastern Economy Edition

LOGIC DESIGN THEORY



621.38195
B 623 L
C.2

NRIPENDRA N. BISWAS

PHI

Scanned by CamScanner

6

Fault Diagnosis and Tolerance

6.1 INTRODUCTION

The subject of fault modeling, diagnosis, testing, and fault tolerance of digital circuits is of crucial importance to the logic designer. The advent of VLSI circuits has resulted in a dramatic increase in the number of components on a single chip. As a consequence of the large circuit density, the probability of a fault occurring in the circuit also increases. Basically, the logic designer is confronted with two problems. First, it should be determined whether a digital circuit operates correctly and is free from faults. This involves the process of fault diagnosis and testing and is a necessary part of the manufacturing process. Second, its correct operation should be ensured even in the presence of faults. This is the topic of fault tolerance. In general, fault tolerance is obtained by introducing redundancy in the circuit. As the number of components on a chip increases, the fault detection and testing problem becomes more difficult and the techniques for fault tolerance become more expensive. Hence research on new and efficient techniques to handle the problem of faults has received considerable interest. In this chapter we discuss some basic concepts regarding faults in logic circuits, their diagnosis, testing, and techniques for fault tolerance.

6.2 FAULT CLASSES AND MODELS

We first discuss the different types of faults in a digital circuit and define the various terms used. A *fault* of a circuit is the physical defect of one or more components or connections of the circuit. Faults can be either permanent or

Scanned by CamScanner

temporary. *Permanent faults* are typically caused by the breaking or wearing out of a component. Such faults are always present and do not appear, disappear, or change their nature during operation. Permanent faults are also called *hard* and *solid faults*. *Temporary faults*, also known as *soft faults*, are those that occur only during certain intervals of time. These faults can be either transient or intermittent. A *transient fault* is usually caused by some externally induced signal perturbation, such as power-supply fluctuations. An *intermittent fault* is one that often occurs when a component is in the process of developing a permanent fault.

Another approach to classification is to classify a fault as logical or parametric, depending on its effect. A *logical fault* changes the Boolean function realized by the digital circuit, while a *parametric fault* alters the magnitude of a circuit parameter, causing a change in a factor, such as circuit speed, current, or voltage. An important type of parametric fault is the delay fault, which is caused by slow gates. This type of fault usually leads to problems of hazards or *critical races* (this topic is dealt with in Chapter 12).

Stuck-at, bridging, and crosspoint faults are three important types of logical faults.

Definition 6.2.1. A *stuck-at fault* is said to have occurred if a signal line appears to have its value fixed at either a logical 1 or a logical 0, irrespective of the input signals applied to the circuit. When the signal line is always at logical 1(0), the fault is known as a *stuck-at-one* or SA1 (*stuck-at-zero* or SA0) *fault*.

Stuck-at faults are one of the simplest faults to analyze. Further, they are proved to be very effective in modeling the fault behavior of actual devices since they represent the most commonly occurring circuit faults. Hence most of the proposed testing techniques cater to this type of fault. A stuck-at fault model generally assumes that the faults affect only the interconnections, especially inputs and outputs of the logic gates.

A *bridging fault* is said to have occurred if two signal lines are shorted together. It may be either an AND- or an OR-type of bridging fault.

Faults that occur in programmable logic arrays due to extra or missing devices (such as a diode or a transistor) are called *crosspoint faults*. Depending on their effect, crosspoint faults are classified as growth (G) faults, shrinkage (S) faults, appearance (A) faults, and disappearance (D) faults. Crosspoint faults are discussed in detail in Chapter 8.

Definition 6.2.2. If a circuit has only one fault at any given time, it is said to have a *single fault*. If there are two or more faults in the circuit, the circuit is said to have *multiple faults*.

Definition 6.2.3. Two faults are said to be *equivalent* or *indistinguishable* if they cause the circuit to malfunction in exactly the same way.

Definition 6.2.4. A fault is said to be *redundant* if the function realized by the circuit with the fault is exactly the same as that of a fault-free circuit.

In add
devices. Fo
(RAMs), ar
the RAM.
Throu
all testing
and perma

6.3 FAULT DIAG

An import
circuit co
be replac
fault dete
task of fa
The
logical ci
inputs of
externall

Def
produces
(TV).

De
the *test*

De
and loc
diagnos
that car
Fi
is *fault*
possibl
the test
in the f
A fault
faults a
T
difficul
detecti
of the
set. *E*
proces
proces
Test a

Scanned by CamScanner

In addition to the foregoing types, there are other faults that occur in specific devices. For example, *pattern-sensitive faults* occur in random-access memories (RAMs), and their effect is dependent on the particular input pattern applied to the RAM.

Throughout this chapter we assume the stuck-at fault model for presenting all testing and diagnosis techniques. Further, the fault is assumed to be logical and permanent.

6.3 FAULT DIAGNOSIS AND TESTING

An important task during the manufacturing process is to determine whether the circuit contains a fault, and if so, to locate it so that the faulty components can be replaced. The task of determining whether a fault is present or not is called *fault detection*, and the task of isolating the fault is *fault location*. The combined task of fault detection and location is referred to as *fault diagnosis*.

The technique adopted to diagnose faults is *testing*. Generally, testing of logical circuits consists of applying a set of input combinations to the *primary inputs* of the circuit (the primary inputs or outputs of a circuit are those that are externally accessible).

Definition 6.3.1. An input combination which in the presence of a fault produces an output different from the fault-free output is known as a *test vector* (TV).

Definition 6.3.2. The set of test vectors used for testing the circuit is called the *test set*.

Depending on whether the testing process results in detection or detection and location, the test set is classified as a *fault detection test set* or a *fault diagnostic test set*. *Test generation* is the process of finding the set of test vectors that can detect (or detect and locate) faults in a circuit.

Figure 6.3.1 shows the typical stages of the testing process. The first step is *fault modeling*, which consists of developing a fault dictionary, or list of possible faults, based on the assumed fault model. The next step is to generate the test vectors for the set of faults. The methods for *test generation* are described in the following sections. *Fault simulation* is the process of verifying the test set. A fault simulator computes the output of the circuit in the presence of various faults and verifies the detectability of the faults by the test set.

The increasing complexity of LSI and VLSI circuits has made it extremely difficult to detect all faults in a circuit. Hence the testing process usually aims at detecting or diagnosing only a fraction of the faults modeled. The *fault coverage* of the test set refers to the percentage of faults that can be detected by the test set. *Evaluation of the fault coverage* is the next important step in the testing process. If the coverage is inadequate, the test generation and fault simulation process is repeated until an adequate percentage of fault coverage is obtained. *Test application* forms the last

Scanned by CamScanner

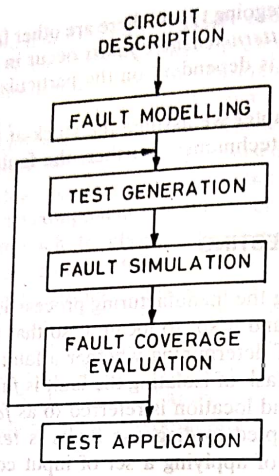


Figure 6.3.1 Testing process.

actual circuit and the responses are checked for correctness. An incorrect response implies the presence of a fault in the circuit. Further test vectors may need to be applied to locate the fault.

6.4 TEST GENERATION

The generation of test vectors forms one of the most important and difficult tasks in the testing process. The complexity arises mainly due to the reduction in the ratio of the number of primary inputs and outputs to the number of internal inaccessible points in the circuit. Consider a circuit with n signal lines. Then under the assumption of a multiple-stuck-at-fault model, the total number of possible states is 3^n, since each line can be fault-free, stuck-at-0, or stuck-at-1. It can now be computed easily that even for a circuit with about 20 lines, this gives rise to an astronomical number of possible states. Generating the test set for checking such an enormous number of states is infeasible. However, it has been found that the assumption of a single stuck-at-fault model provides a good practical basis for testing most circuits. For the circuit with n lines, there are at most $2n$ possible single stuck-at faults. However, even for the single-stuck-at-fault model, there is a need for good techniques for efficient test generation. The objective of test generation is to obtain a *minimal complete test set*.

Definition 6.4.1. A test set of a circuit is *complete* if it detects every fault (of the assumed fault model) in the circuit under consideration.

Definition 6.4.2. A *minimal complete test set* is a complete test set that contains the minimum number of test vectors.

Scanned by CamScanner

To illustrate the principles involved in test set generation, we describe below a technique called the fault-table method that gives the minimal complete test set.

6.5 FAULT-TABLE METHOD

Consider the circuit of Fig. 6.5.1, consisting of an AND and an OR gate. The primary inputs to the circuit are x_1', x_2, and x_3, while the primary output is z. Table 6.5.1 illustrates a fault table listing the outputs for each input pattern under fault-free and faulty conditions. The single-stuck-at-fault model is assumed. A stuck-at-0 fault on line a, for example, is denoted as a_0. The column under a_0 gives the output of the circuit for each input in the presence of this fault. Similarly, each of the other columns gives the response when a fault on the corresponding line is present. Now, a close look at the eight columns of the faulty outputs will reveal that the columns for faults a_0, b_0, and c_0 have identical responses. Hence these three SA0 faults are indistinguishable or equivalent faults. Similarly, the two SA1 faults c_1 and d_1 are also equivalent. The equivalence of faults may be ascertained by another observation. Note that a_0 and b_0 are the SA0 faults at the two inputs of the AND gate. When a SA0 fault occurs at the input of an AND gate, it forces the output of the gate to 0 irrespective of what happens at other inputs. Hence a SA0 fault at the input of an AND gate is classified as a *dominant* or *strong* fault. On the other hand, a SA1 fault at one of the inputs alone of an AND gate cannot force the output of the gate to 1. Hence a SA1 fault at the input of an AND gate is a *weak* fault. For similar reasons, a SA1 fault at the input of an OR gate is a dominant or strong fault, whereas an SA0 fault at the input of an OR gate is a weak fault. It is also obvious that either the SA0 or the SA1 fault at the output of any gate will belong to the dominant class of faults of that gate. It is now easy to see that all dominant faults pertaining to a gate will be indistinguishable and therefore equivalent.

The next step is to obtain a fault cover table that gives the test set for detecting each fault. In this table (Table 6.5.2) all the equivalent faults are grouped together and are treated as a single column. The test vectors for detecting each fault is determined by comparing the fault-free output with the faulty output of the fault.

Consider the first input pattern ($x_1x_2x_3 = 000$). The fault-free output z for this input is 0. All single stuck-at faults in the circuit except the faults b_1, c_1, and d_1 result in an output 0. The presence of any one of the faults b_1, c_1, or d_1 produces an output 1 that differs from the fault-free output. This implies that these faults can be detected by the test vector 000. Thus a × is placed under columns b_1 and (c_1d_1) in the first row corresponding to input 000. Considering another example,

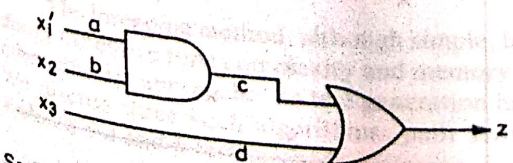


Figure 6.5.1 A circuit under test

Scanned by CamScanner

TABLE 6.5.1 SET OF ALL POSSIBLE SINGLE STUCK-AT-FAULTS AND THE FAULT-FREE AND FAULTY RESPONSES

				Faulty outputs							
				SA0 faults				SA1 faults			
x_1	x_2	x_3	Fault-free output z	a_0	b_0	c_0	d_0	a_1	b_1	c_1	d_1
0	0	0	0	0	0	0	0	0	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	1	1
1	0	1	1	1	1	1	0	1	1	1	1
1	1	0	0	0	0	0	0	1	0	1	1
1	1	1	1	1	1	1	0	1	1	1	1

input vector 001 gives a fault-free output of 1. The presence of fault d_0 gives an output 0. This means that test vector 010 is able to detect fault d_0. Hence a × is placed under column d_0 in row 001. The other rows are completed in a similar manner. It can also be observed from Table 6.5.1 that input vector 011 is not able to detect any fault.

Now the problem of obtaining a minimal complete test set is that of finding the smallest number of rows in the fault table such that all faults are detected. This is very similar to the problem of obtaining the minimum number of rows to cover all the minterms of a prime implicant table of the Quine–McCluskey algorithm for minimization of Boolean functions. Here also we can define three types of test vectors: essential, redundant, and selective. We define only the

TABLE 6.5.2 FAULT COVER TABLE

TVs			Faults				
x_1	x_2	x_3	$(a_0b_0c_0)$	d_0	a_1	b_1	(c_1d_1)
*0	0	0				×	×
0	0	1		×			
*0	1	0	×				
1	0	0					×
1	0	1		×			
*1	1	0			×		×
1	1	1		×			

Scanned by CamScanner

essential test vector below. The definitions of the other two types are similar to those of the redundant and selective prime cubes of Chapter 3.

Definition 6.5.1. If a fault is detected by one and only one test vector, this test vector is an *essential test vector* (ETV).

It is obvious that an ETV can be identified on the fault cover table (Table 6.5.2) by observing single-cross columns. Thus the ETVs 000, 010, and 110 (marked by an * in Table 6.5.2) are ETVs. The other faults covered by the ETVs are now checked (√) off. The remaining columns are to be covered by STVs. In this case, only fault d_0 remains to be covered. As can be seen from the table, it can be tested by any one of the TVs 001 or 101 or 111. Thus the minimum test set to test all SA0 and SA1 faults of the circuit is

$$\{000, 010, 110, (001 \text{ or } 101 \text{ or } 111)\}$$

Note that in Table 6.5.2, TV 100 is a redundant TV.

For larger cover tables, after ETVs have been chosen, a selective test vector (STV) table is derived from the fault cover table. In this table all rows that are ETVs and all columns that have been checked off are not included. This table is reduced either by deleting all dominated rows or by deleting all dominating columns (see the reduction of the SPC table of Chapter 4). Dominating and dominated rows and columns are defined in the same way as has been done in Chapter 4. The minimal subset of STVs to detect all faults is now determined from the reduced STV table.

It should be mentioned here that although the problem of determining a minimum cover from the fault cover table is very similar to the problem of determining a minimum cover of the prime implicant cover table, it is definitely not identical. In fact, there is an important difference between the two. In the rows of a prime implicant table, all the cubes may not be of the same dimension but may vary from a 0-cube to an $(n - 1)$-cube in an n-variable Boolean function. On the other hand, the rows, that is, TVs of a fault cover table, are of the same dimension, each being a minterm or a 0-cube. For this reason any one of the STVs covering a fault is a valid solution, whereas any one of the SPCs covering a minterm may not be a valid cover.

The test application process that follows test set generation consists of applying the four test vectors in sequence to the circuit and observing the output z. For example, if 000 results in output 0, we know that the circuit is free of faults b_1, c_1, and d_1, but other faults might be present. Hence the second test vector 001 needs to be applied. If the output z is 1, the circuit can immediately be declared as faulty. This process is repeated with other test vectors to diagnose other faults.

The foregoing method, although simple, becomes infeasible for large circuits, due to excessive time complexity and memory storage requirements. Hence many other efficient approaches to test generation have been developed over the years. We discuss three such algorithms: path sensitization, Boolean difference, and Kohavis's a and b test methods.

Scanned by CamScanner

6.6 PATH-SENSITIZATION METHOD

The main idea behind this approach is to create a sensitized path in order to move the effect of the fault at any line in the circuit to primary output. Suppose that it is required to detect a SA0 fault on a line. Then it is essential that a test vector create a change on that line and ensure that the change can be seen at a primary output. In other words, the test vector must produce a 1 on that line, and the path from the line to the output must be sensitized so that the output clearly shows whether the signal on the line under consideration is 0 or 1. Then the fault can be detected. This is the principle of the path-sensitization method.

To illustrate the approach, consider Fig. 6.6.1, which shows a two-level circuit with four inputs, x_1, x_2, x_3, and x_4, and a single output, z. Suppose that it is required to detect if any fault is present on the input line a. Then we sensitize the path from a to the output in order to propagate its effect to the output. If the path from a is traced, the first condition we get is $x_2 = 1$. This is so because if x_2 is 0, the output y_1 of the NAND gate will be 1 irrespective of the signal on line a. Following a similar reasoning, the input y_2 to the NOR gate must be 0. This in turn gives the condition $x_3 = 1$ and $x_4 = 1$. Thus, to detect the fault SA0 on line a, the test vector 1111 is applied. If the output z is 1, the line a is free of an SA0 fault. If z is 0, line a has an SA0 fault. To detect fault SA1 on line a, the test vector 0111 is applied. If z is 0, a does not have an SA1 fault; if z is 1, a has the fault.

As another example, consider the three-level circuit of Fig. 6.6.2. Suppose that it is required to check if line g (shown by a ×) has an SA0 fault. Now it is required to form a test vector so that a 1 is produced on g. This gives $x_1 = 1$ and $x_2 = 1$. Next, the path $G_1G_4G_6$ from g to the output has to be sensitized. This

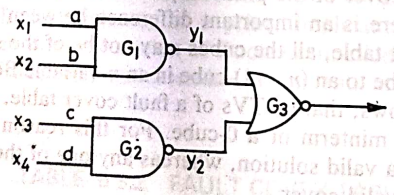


Figure 6.6.1 Path sensitization in a combinational circuit.

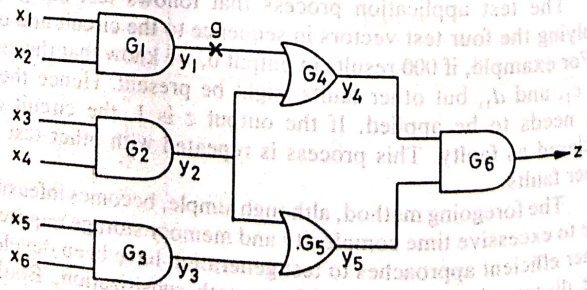


Figure 6.6.2 Path sensitization in a three-level combinational circuit.

Scanned by CamScanner

immediately requires that y_2 be 0, since if y_2 is 1, y_4 will be 1 irrespective of g being 0 or 1, and thus the path would be desensitized. The condition $y_2 = 0$ requires that $x_3 = 0$ and $x_4 = 2$ ("don't care") (alternatively, $x_3 = 2$ and $x_4 = 0$). Proceeding in a similar fashion, the next condition for path sensitization is $y_5 = 1$, which in turn gives $x_5 = 1$ and $x_6 = 1$. Thus any of the two test vectors within the cube 110211 detects the SA0 fault on g. If, on the application of the test vector, the output $z = 1$, the fault is not present; if $z = 0$, g has a SA0 fault.

In summary, the path-sensitization approach to test generation has the following steps (suppose that the test vector for a fault on line g is to be found).

1. A path is found from g to the output.
2. The input signals necessary to produce the appropriate signal at g (opposite the fault value) are determined.
3. The path from g to the output is sensitized and other inputs are determined by back propagation.

A general rule for path sensitization can be stated as follows: The output of an AND (NAND) gate is sensitized to one of its inputs (complement of the input) by placing a 1 on all other inputs. Similarly, the output of an OR (NOR) gate is sensitized to one of its inputs (complement of the input) by placing a 0 on all other inputs.

The single-path sensitization method, although simple and straightforward, has one major drawback. Although it works well for fanout-free circuits, it can sometimes fail in circuits with reconvergent fanout, as the following example demonstrates.

Consider the circuit shown in Fig. 6.6.3, which realizes the XOR function of variables x_1 and x_2. Let us try to derive a test vector for the SA1 fault on line y_1. There are two possible paths, $G_1G_2G_4$ and $G_1G_3G_4$, from the fault site to the primary output z. Let us first try to sensitize the path $G_1G_2G_4$. To excite the fault, we must set $y_1 = 0$, which implies that $x_1 = x_2 = 1$. Gate G_2 is already sensitized since $x_1 = 1$ and the fault effect reaches y_2. To sensitize gate G_4, we have to set $y_3 = 1$, and this in turn requires that $x_2 = 0$, leading to a contradiction, since the value of x_2 was determined earlier to be 1. Note that the logic value 0 on y_1 in the fault-free circuit cannot be used to justify the 1 required on y_3, as y_1 will change to 1 in the presence of the fault. Thus the path $G_1G_2G_4$ cannot be sensitized

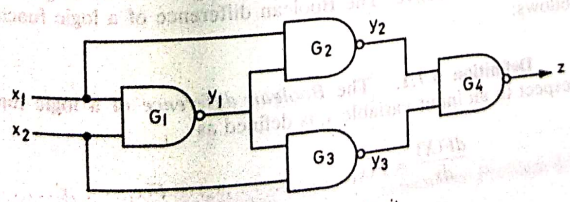


Figure 6.6.3 A reconvergent combinational circuit.

to detect the SA1 on y_1. Similarly, it can easily be verified that sensitizing the path $G_1G_3G_4$ will require setting x_1 to 0, which again leads to a contradiction. Thus single-path sensitization will fail to generate a test for y_1 SA1.

However, the fault above can be detected if we sensitize both paths from y_1 to the output simultaneously. If we set $x_1 = x_2 = 1$, the fault effect will propagate through gates G_2 and G_3, as both of these gates are sensitized. Under this condition, if the fault is present in the circuit, $y_2 = y_3 = 0$ and we get $z = 1$. However, if the circuit is fault-free, $y_2 = y_3 = 1$ and $z = 0$. Thus the fault is detected by the vector 11, through multiple-path sensitization, even though it is not possible to detect this fault via single-path sensitization. Thus any complete test generation algorithm should be capable of sensitizing all possible combinations of paths from the fault site to the primary output, so as to ensure detection of the fault. The *D*-algorithm developed by Roth (1966) is one such algorithm and guarantees to generate a test if a test exists for any given fault. The *D*-algorithm is a fairly complex algorithm and is not discussed in this book.

6.7 BOOLEAN DIFFERENCE METHOD

The Boolean difference method is an algebraic technique for test generation in which the test vectors are generated by utilizing the properties of Boolean algebra. Consider a logic circuit that realizes, under fault-free conditions, the Boolean function $F(X) = F(x_1x_2,\ldots,x_n)$ of n input variables, namely, $x_1, x_2, \ldots, x_n$. Suppose that a fault is present in the circuit. Let $F'(X)$ be the function realized by the circuit in the presence of the fault. Then the underlying basis for the Boolean difference method is stated in the following theorem.

Theorem 6.7.1. The complete set of test vectors for the fault is the set

$$\{X \mid F(X) \oplus F'(X) = 1\}$$

It is easy to see that any input vector $X = x_1 x_2 \cdots x_n$, that satisfies the equation $F(X) \oplus F'(X) = 1$ distinguishes between the two functions. Hence X is a test vector—hence the theorem.

The algebra of *Boolean differences* provides an elegant method for determining the set above. The Boolean difference of a logic function is defined as follows:

Definition 6.7.1. The *Boolean difference* of a logic function $F(X)$ with respect to an input variable x_i is defined as

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

Scanned by CamScanner

Let us denote the values of the function $F(X)$ when x_i assumes the values 0 and 1, respectively, as $F_i(0)$ and $F_i(1)$, that is,

$$F_i(0) = F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

$$F_i(1) = F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

Then it is easy to see that

$$\frac{dF(X)}{dx_i} = F_i(0) \oplus F_i(1) \tag{1}$$

First, we describe the method of test generation assuming a fault in one of the input lines. Later this result is generalized to a fault in any line. Consider a stuck-at-0 fault on an input line, say x_i. Then the function realized in the presence of the fault is $F'(X) = F_i(0)$. Hence, by Theorem 6.7.1, the test set for the fault can be obtained by solving the equation

$$F(X) \oplus F_i(0) = 1 \tag{2}$$

Similarly, it can be seen that the test set for a stuck-at-1 fault on input line x_i can be obtained by solving

$$F(X) \oplus F_i(1) = 1 \tag{3}$$

Now we state a theorem that provides a method for solving the equations above using Boolean differences.

Theorem 6.7.2. Equations (2) and (3) can be expressed, respectively, as

$$x_i \frac{dF(X)}{dx_i} = 1 \tag{4}$$

and

$$x_i' \frac{dF(X)}{dx_i} = 1 \tag{5}$$

Hence, the test sets for the faults SA0 and SA1 on input x_i are, respectively,

$$\left\{ X \mid x_i \frac{dF(X)}{dx_i} = 1 \right\}$$

and

$$\left\{ X \mid x_i' \frac{dF(X)}{dx_i} = 1 \right\}$$

The derivations of (4) and (5) are left as exercises (see Problem 6.6).

Scanned by CamScanner

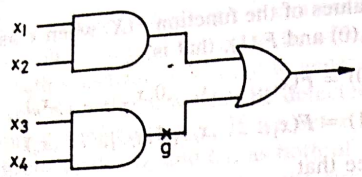


Figure 6.7.1 AND–OR circuit realizing $F(x) = x_1x_2 + x_3x_4$.

Example 6.7.1

In the circuit of Fig. 6.7.1, which realizes the function $F(X) = x_1 x_2 + x_3 x_4$ under fault-free conditions, find the test set for a SA0 fault at input line x_1 of the circuit.

Solution From Theorem 6.7.2 the test is

$$\left\{ X \mid x_i \frac{dF(X)}{dx_i} = 1 \right\}$$

Now

$$\begin{aligned} x_1 \frac{dF(X)}{dx_1} &= x_1(F_1(0) \oplus F_1(1)) && \text{from (1)} \\ &= x_1(x_3x_4 \oplus (x_2 + x_3x_4)) && \text{by substitution} \\ &= x_1x_2x_3' + x_1x_2x_4' && \text{after simplification} \end{aligned}$$

The equation $x_1x_2x_3' + x_1x_2x_4' = 1$ yields the two cubes 1102 and 1120 as solutions. Hence the test set is (1102, 1120) where the 2 represents a don't care; that is, the corresponding variable can be either 0 or 1.

The following theorem, which is an extension of Theorem 6.7.2, is used to obtain the test set for a fault in an internal signal line of the circuit.

Theorem 6.7.3. Let g be any signal line in the circuit. Let $G(x_1,x_2,\ldots,x_n)$ be the logic function giving the dependence of the value of the signal on g to the values of the input variables. Let $F(G,x_1,x_2,\ldots,x_n)$ be the function realized by the circuit as a function of G and the input variables. Then the test vectors for a SA0 on g are obtained by solving the equation

$$G = \frac{dF}{dG} = 1 \tag{6a}$$

and those for SA1 on g are obtained by the solution of

$$G' \frac{dF}{dG} = 1 \tag{6b}$$

The proof of this theorem is left as an exercise for the reader (see Problem 6.7).

Example 6.7.2

Find the test vectors for a SA0 fault on line g of the circuit of Fig. 6.7.1.

Scanned by CamScanner

Solution The signal on line g can be expressed as $G = x_3 x_4$ and the output of the circuit is $F(G,x_1,x_2) = G + x_1x_2$. By Theorem 6.7.3 the test vectors for a SA0 fault on g are obtained by solving

$$G = \frac{dF}{dG} = 1$$

The left-hand side of the equation above is

$$\begin{aligned} x_3x_4(F_G(0) \oplus F_G(1)) &= x_3x_4(x_1x_2 \oplus 1) \\ &= x_1'x_3x_4 + x_2'x_3x_4 \end{aligned}$$

Thus the test vectors are (0211, 2011).

6.8 THE KOHAVI ALGORITHM

The path sensitization and the Boolean difference methods become practically infeasible for multiple faults, even for circuits of moderate size. This is because both methods consider only one fault at a time, and the total number of states to be tested in a circuit with n lines becomes $3^n - 1$ if multiple faults are allowed. Kohavi and Kohavi (1972) have proposed a method that overcomes this difficulty for test generation for multiple faults in two-level networks. The technique, which consists of determining two sets of tests, the a-tests and b-tests, considers the altered Boolean function realized by the circuit due to the presence of a single fault rather than considering the faults themselves. Kohavi and Kohavi have then proved that this single fault test set detects all multiple faults (see Theorem 6.8.1). However, to achieve the objective of providing a single fault test set with the capability of detecting all multiple stuck-at faults, Kohavi and Kohavi have put three severe restrictions on the networks, where alone the algorithm has to be applied. First, the network must be a two-level AND–OR or (OR–AND) network. Second, each AND gate must realize a prime cube. Third, the AND–OR network must implement a Boolean function which is a sum of irredundant prime implicants.* The Kohavi algorithm is quite simple and straightforward. But as we discussed in Chapter 5, the programmable logic arrays (PLAs) are two-level AND–OR networks, and extensive research has been done in finding efficient algorithms for PLA test generation. Consequently, quite a few excellent CAD packages are available to test PLAs. These are discussed in Chapter 8. It will then be apparent that Kohavis's a and b test set becomes a subset of the PLA test set. Hence this algorithm has lost much of its practical importance. Nevertheless, we shall discuss this algorithm, as it clearly brings out some of the fundamental concepts, which will be very useful in understanding faults in PLAs.

We first describe the algorithm for a two-level AND–OR circuit. As mentioned earlier, it is assumed that each AND gate of the circuit realizes an

*By "a sum of irredundant prime implicants," Kohavi and Kohavi (1972) mean a sum that does not contain either a redundant prime implicant or a redundant literal. However, it is not equivalent to the term "minimum sum of product" (MSOP), as we discussed in Chapter 3. For example, the function $f = \Sigma\,(0,1,3,5,7,8,12,13)$ has two irredundant sum forms, $f_1 = 0221 + 2000 + 1102$ and $f_2 = 0221 + 0002 + 2101 + 1200$. But only one of these, f_1, is the MSOP form.

Scanned by CamScanner

irredundant prime implicant. Thus the output function of the circuit can be expressed as a sum of irredundant prime implicants or cubes. Consider a SA0 fault on any of the inputs of an AND gate. The effect of this fault on the output function is the elimination of the prime implicant realized by the AND gate. This can be checked by any one of the distinguished minterms of the prime cube (see Definition 3.4.3).

Definition 6.8.1. The set of distinguished minterms that tests each AND gate for SA0 faults is called the *set of a-tests*.

Consider a SA1 fault on any of the inputs of an AND gate. To illustrate the effect of such a fault, consider the circuit of Fig. 6.8.1(a), which realizes the function $f = x_1x_2' + x_3x_4$. The prime implicants of the function are $x_1x_2' = 1022$ and $x_3x_4 = 2211$, which are shown in the Karnaugh map of Fig. 6.8.1(b). Suppose that there is an SA1 fault on the input line x_2'. The effect of this fault is to change the prime implicant x_1x_2' to x_1. In other words, the cube 1022 expands to the cube 1222 (shown by the dashed line in the figure). Thus the fault can be detected choosing, as a test vector, a minterm that is in the subcube 1122, which is adjacent to the original cube. Another important consideration is that the minterm should be in the offset of the function. To test an SA1 fault in *any* input of an AND gate, it is necessary to select a test minterm in *each* of the adjacent subcubes of the prime implicant realized by the AND gate. The set of all such minterms constitutes the set of *b*-tests.

Definition 6.8.2. The set of minterms that test each AND gate for SA1 faults is called the set of *b*-tests.

The following theorem can now be stated.

Theorem 6.8.1. (Kohavi and Kohavi, 1972). The set of *a*- and *b*-tests detects all multiple faults in the two-level AND–OR network.

For proof, see Theorem 2 of Kohavi and Kohavi (1972) or Theorem 8-1 of Kohavi (1978).

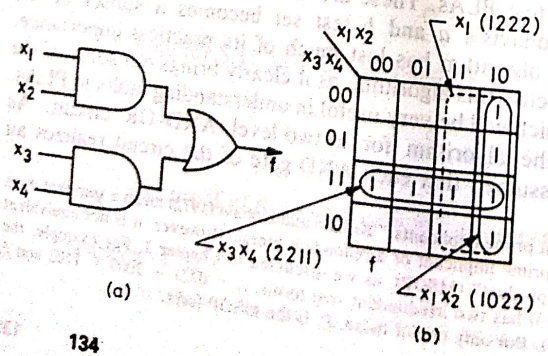


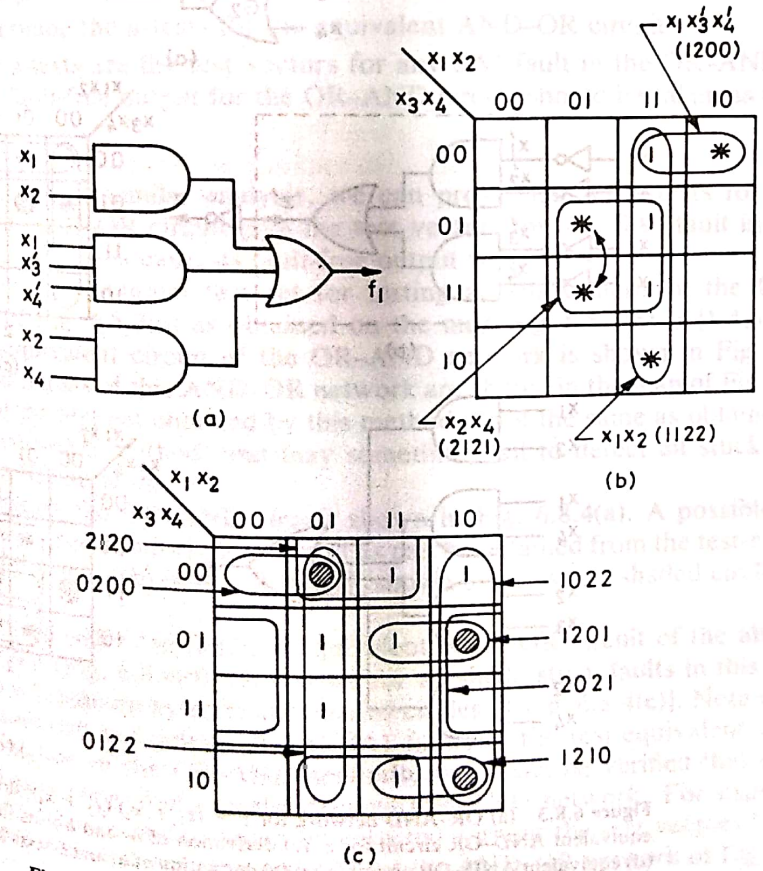
Figure 6.8.1 (a) AND–OR network realizing the function $f = x_1x_2' + x_3x_4$; (b) generation of *a*-tests and *b*-tests on the map.

Example 6.8.1

Figure 6.8.2(a) shows a two-level AND–OR circuit that realizes the function $f = x_1x_2 + x_1x_3'x_4' + x_2x_4$. Find a minimum test set for this network.

Solution The prime implicants of the function x_1x_2 (1122), $x_1x_3'x_4'$ (1200), and x_2x_4 (2121) are shown in the map in Fig. 6.8.2(b). To derive the *a*-tests, consider each cube and mark a minterm that belongs to that and only that cube. Thus minterm 14 is chosen for 1122, minterm 8 for 1200, and either of minterms 5 or 7 can be chosen for 2121; these minterms are marked * in Fig. 6.8.2(b). Thus the set of *a*-tests is {14,8,5 or 7}.

Next we consider the derivation of *b*-tests. Figure 6.8.2(c) shows the subcubes that are adjacent to each of the prime implicants. The adjacent subcubes are 0122 and 1022 (adjacent to 1122), 0200, 1210, 1201 (adjacent to 1200), 2021, and 2120 (adjacent to 2121). Now we choose a set of minterms that are present in each of these adjacent subcubes but not present in the function. One such selection is shown in the figure (hatched circles). Minterm 9 covers 1022, 1201, and 2021. Minterm 4 covers 0122, 2120, and 0200. Next, minterm 10 is chosen to cover the remaining



Figu... derivations of

subcube 1210. Thus the set of b-tests is {4,9,10}. Thus the set {14,8,5 or 7,4,9,10} is the test set for the circuit of Fig. 6.8.2(a).

Now let us consider the derivation of a- and b-tests for a two-level OR–AND circuit. The basic strategy of the Kohavi algorithm in this case is to find a test equivalent AND–OR circuit for the given OR–AND network. For the sake of illustration, consider the two-level OR–AND circuit of Fig. 6.8.3(a). Let us find the test vectors for a SA1 fault on an input line of the OR gate G_1. Then, from the path-sensitization principle, it is easy to see that this requires $x_1 = 0$, $x_2 = 0$, and among x_3 and x_4, at least one of them must be 1. Thus the test vector is within the cube 0021 or 0012, and we can choose any of the minterms 1, 2, and

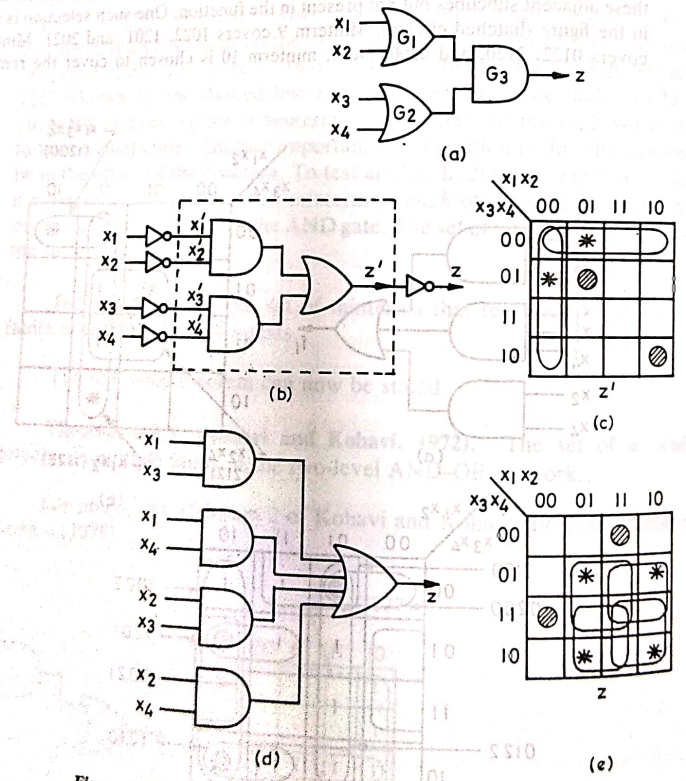


Figure 6.8.3 (a) OR–AND network for $z = (x_1 + x_2)(x_3 + x_4)$; (b) test-equivalent AND–OR circuit for z; (c) derivation of a- and b-tests for z; (d) equivalent AND–OR circuit for z; (e) derivation of a- and b-tests for circuit (d).

Scanned by CamScanner

3 to detect the SA1 fault. By following a similar procedure for G_2, we find that any of the minterms 4, 8, and 12 can detect a SA1 fault on the input line of G_2. Now if we consider both gates G_1 and G_2, the SA1 fault can be detected by minterms 1 or 2 or 3 (for G_1) and 4 or 8 or 12 (for G_2). An interesting observation can be made by comparing this result with the Kohavi algorithm for an AND–OR network. Convert the OR–AND network into an AND–OR network with inverters as discussed in Section 5.3. This is shown in Fig. 6.8.3(b). Now find the a-tests for the AND–OR network of the equivalent circuit shown in the dashed box of Fig. 6.8.3(b). This AND–OR circuit realizes the sum of the two cubes 0022 and 2200. It can be seen from Fig. 6.8.3(c) that the a-test vectors are the same as those obtained by the path-sensitization method.

Now the algorithm to find the test vectors for any SA1 fault in a two-level OR–AND circuit can be written as follows:

1. Replace all OR gates in the OR–AND circuit by AND gates, and vice versa. Replace all primary inputs and the outputs by their respective complements. This gives the test-equivalent AND–OR circuit.
2. Determine the a-tests for the equivalent AND–OR circuit.
3. The a-tests are the test vectors for any SA1 fault in the OR–AND circuit. The fault-free output for the OR–AND circuit should be taken as 0 and not 1.

Following a similar analysis, we can prove that the b-tests for the test-equivalent AND–OR circuit give the test vectors for any SA0 fault in the OR–AND circuit. In this case, its fault-free output will be 1.

Thus the minimum test set for testing all stuck faults of the OR–AND network of Fig. 6.8.3(a) as obtained on the map Fig. 6.8.3(c) is {1,4,5,10}. The function-equivalent circuit of the OR–AND network is shown in Fig. 6.8.3(d). The a and b tests of this AND–OR network are shown in the map of Fig. 6.8.3(e). Note that the test set obtained by this method is not the same as obtained by the path-sensitization method, and may sometimes fail to detect all stuck faults of the OR–AND network.

Consider the OR–AND circuit shown in Fig. 6.8.4(a). A possible test set for all single stuck faults in this circuit as can be obtained from the test-equivalent circuit of Fig. 6.8.4(b) is $T = \{0,2,5,7\}$ [shown by stars and shaded circles in Fig. 6.8.4(c)].

Now consider the function-equivalent AND–OR circuit of the above OR–AND circuit [Fig. 6.8.4(d)]. A test set for all single stuck faults in this circuit is $T = \{0,4,6,7\}$ [shown by stars and shaded circles of Fig. 6.8.4(e)]. Note that these test vectors will not detect all SA0/SA1 faults of the test-equivalent AND–OR implementation of the OR–AND network, and it can be verified that these will fail to detect some faults in the original OR–AND network. For example, the SA0 faults for G_1 and G_2 cannot be detected without the test vectors 2(010) and 5(101). But these are not included in T for the AND–OR network of Fig. 6.8.4(d).

Scanned by CamScanner

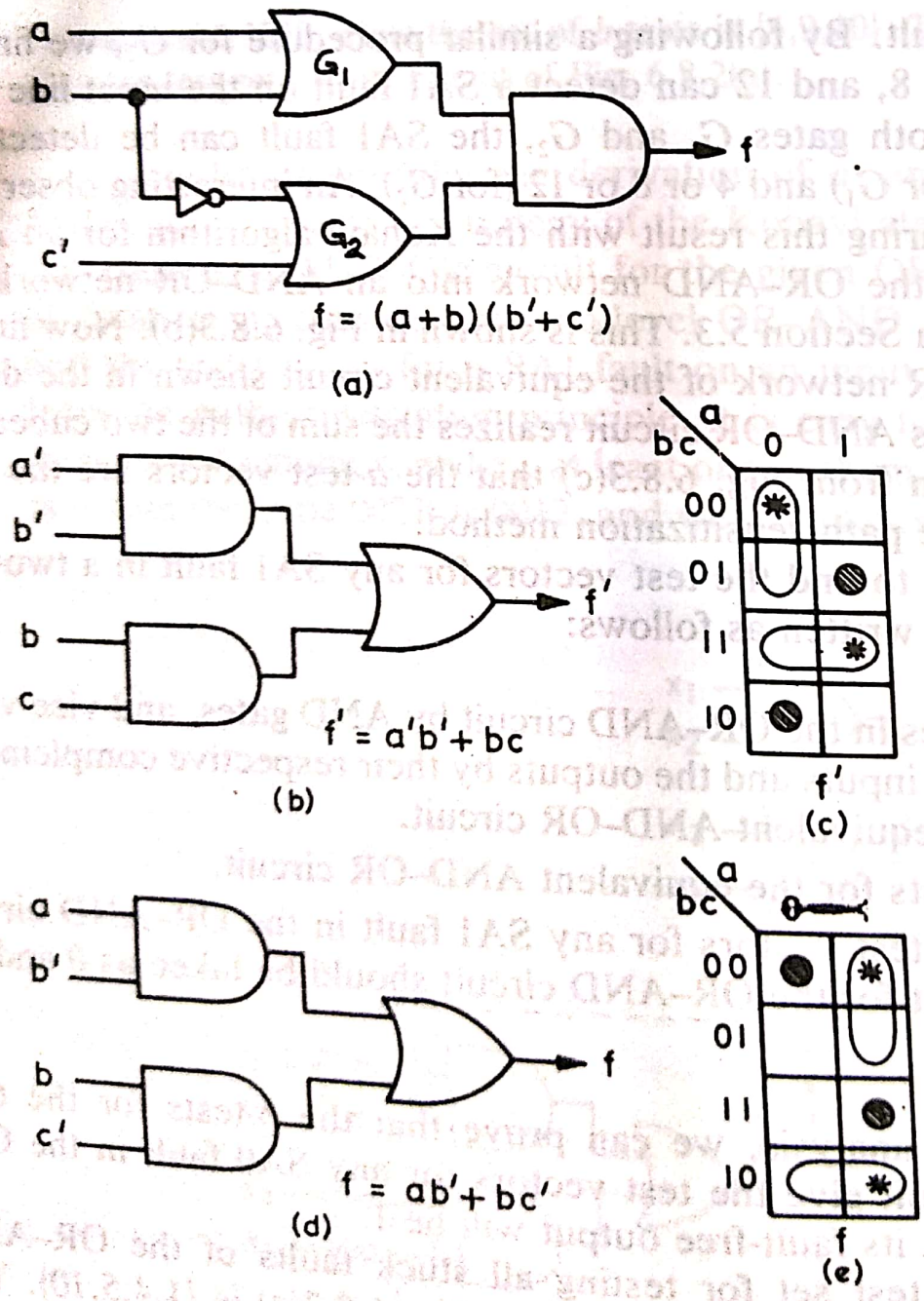


Figure 6.8.4 (a) An OR–AND circuit; (b) its test-equivalent AND–OR circuit; (c) its *a*- and *b*-test vectors; (d) functional equivalent AND–OR circuit; (e) its *a*- and *b*-test vectors.

Scanned by CamScanner