12. LOGIC-LEVEL DIAGNOSIS

About This Chapter

In this chapter we analyze logic-level fault-location techniques. After discussing the basic concepts of diagnosis, we review the fault-dictionary method, guided-probe testing, diagnosis by progressive reduction of the unit under test, methods specialized for combinational circuits, expert systems for diagnosis, effect-cause analysis, and a reasoning technique based on structure and behavior.

12.1 Basic Concepts

A unit under test (UUT) *fails* when its observed behavior is different from its expected behavior. If the UUT is to be repaired, the cause of the observed error(s) must be diagnosed. Diagnosis consists of locating the physical fault(s) in a structural model of the UUT. In other words, *diagnosis maps the observed misbehavior of the UUT into physical faults affecting its components or their interconnections.*

The degree of accuracy to which faults can be located is referred to as *diagnostic resolution.* No external testing experiment can distinguish among functionally equivalent faults. The partition of all the possible faults into distinct sets of functionally equivalent faults defines the *maximal fault resolution*, which is an intrinsic characteristic of the system. The *fault resolution of a test sequence* reflects its capability of distinguishing among faults, and it is bounded by the maximal fault resolution. This is illustrated in Figure 12.1(a). A, B, C, D, E, and F represent the sets of equivalent faults of a system. Consider a test sequence T that does not distinguish between (the faults in) A and (those in) B, or between E and F. The maximal fault resolution is the partition $\{A,B,C,D,E,F\}$, while the resolution of T is $\{A \cup B, C, D, E \cup F\}$. A test (sequence) that achieves the maximal fault resolution is said to be a *complete fault-location test.*

Repairing the UUT often consists of substituting one of its *replaceable units* (RUs) identified as containing some faults, and referred to as a *faulty RU*, by a good unit. Hence usually we are interested only in locating a faulty RU, rather than in an accurate identification of the fault inside an RU. This diagnosis process is characterized by the *RU resolution.* Figure 12.1(b) illustrates the relation between fault resolution and RU resolution. U_1, U_2, U_3, and U_4 are the RUs of the system, and the faults are physically partitioned as shown. If the actual fault belongs to A or B, in either case we can identify U_1 as the faulty RU. But if the actual fault belongs to C, then we cannot determine whether the faulty RU is U_2 or U_3. However, distinguishing between U_2 and U_3 is feasible when the fault belongs to D or E.

Clearly, the location of the faulty RU is more difficult when equivalent faults span different RUs. The RU resolution corresponding to a fault resolution defined by a partition $\{F_1, F_2, ..., F_k\}$ of the set of faults, is obtained by replacing every F_i by the set of RUs spanned by the faults in F_i. For our example, the maximal RU resolution, corresponding to the maximal fault resolution, is given by $\{U_1, U_2, U_3, \{U_2, U_3\}, U_4\}$. The *RU resolution* of any test is bounded by the maximal RU resolution. For example, the RU resolution of T is $\{U_1, U_2, \{U_2, U_3\}, \{U_3, U_4\}\}$. A test that

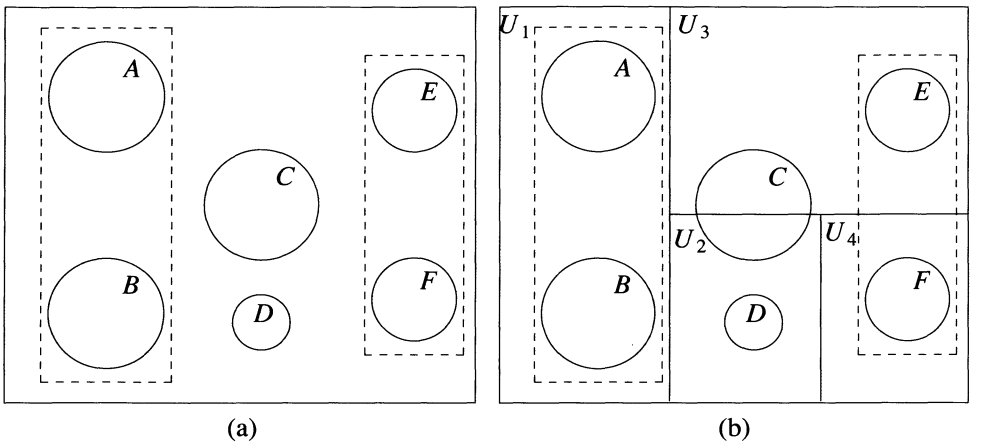


Figure 12.1 Fault resolution and RU resolution

achieves the maximal RU resolution (i.e., it distinguishes between every pair of nonequivalent faults that reside in different RUs) is said to be a *complete RU-location test.*

For the preceding example, suppose that the results of the test do not distinguish between U_3 and U_4. In such a case, it is sometimes possible to replace one of the suspected RUs, say U_3, with a good RU, and rerun the experiment. If the new results are correct, the faulty RU is the replaced one; otherwise, it is the remaining one (U_4). This type of approach is an example of a *sequential diagnosis procedure*, in which diagnosis and repair are interleaved.

The diagnosis process is often *hierarchical* such that the faulty RU identified at one level becomes the UUT at the next level. For example, to minimize the downtime of a computing system, first-level diagnosis deals with "large" RUs, such as boards containing many components; these are referred to as *field-replaceable units.* The faulty board is then tested in a maintenance center, where the objective is to locate a faulty component on the board; this is done to minimize the cost of the replaced unit. A typical RU at this level is an IC. Although further repair beyond the IC level is not possible, accurate location of faults inside a faulty IC may be useful for improving its manufacturing process.

The hierarchical diagnosis process described above proceeds *top-down*, starting with a system operating in the field. During the fabrication of a system, however, its testing proceeds *bottom-up* (e.g., ICs $\rightarrow$ boards $\rightarrow$ system), such that a higher level is assembled only from components already tested at a lower level. This is done to minimize the cost of diagnosis and repair, which increases substantially with the level at which the faults are detected. For example, if it costs \$1 to test an IC, the cost of locating the same defective IC when mounted on a board and of repairing the board may be about \$10; when the defective board is plugged into a system, the cost of finding the fault and repairing the system may be about \$100.

Note that in the bottom-up process, the faults most likely to occur are fabrication errors affecting the interconnections between components, while in the top-down process, the most likely faults are physical failures internal to components (because every UUT had been successfully tested in the past). Knowing the most likely class of faults is a definite help in fault location.

Fault diagnosis can be approached in two different ways. The first approach does most of the work before the testing experiment. It uses fault simulation to determine the possible responses to a given test in the presence of faults. The data base constructed in this step is called a *fault dictionary*. To locate faults, one tries to match the actual response obtained from the UUT with one of the precomputed responses stored in the fault dictionary. If this look-up process is successful, the dictionary indicates the corresponding fault(s) or faulty RU(s) in the UUT.

Fault diagnosis based on fault dictionaries can be characterized as a *cause-effect analysis* that starts with possible causes (faults) and determines their corresponding effects (responses). A second type of approach, employed by several diagnosis methods, relies on an *effect-cause analysis*, in which the effect (the actual response obtained from the UUT) is processed to determine its possible causes (faults).

12.2 Fault Dictionary

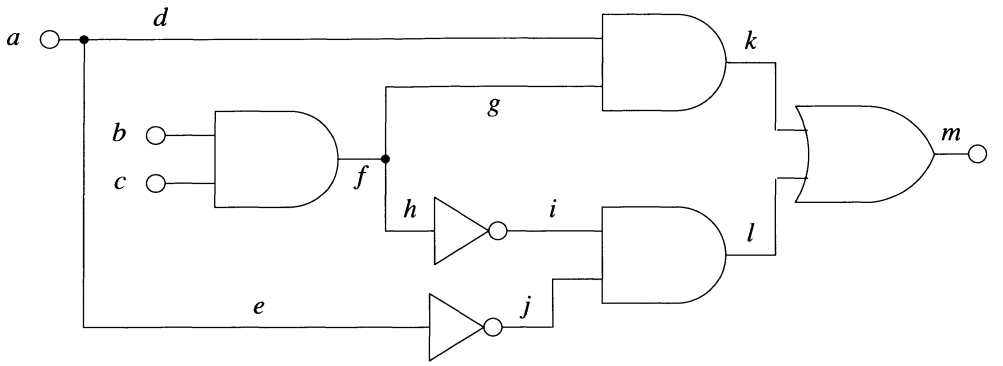
Example 12.1: To illustrate the concepts involved in building and using a fault dictionary, we will analyze the circuit shown in Figure 12.2(a). The circuit has 13 lines and 26 single stuck-at faults. Fault collapsing based on structural equivalence partitions the faults into the following 14 equivalence classes (x_i denotes x *s-a-i*):

1. $\{a_0\}$
2. $\{a_1\}$
3. $\{b_1\}$
4. $\{c_1\}$
5. $\{d_1\}$
6. $\{f_0, b_0, c_0\}$
7. $\{f_1\}$
8. $\{g_1\}$
9. $\{i_0, h_1, l_0, j_0, e_1\}$
10. $\{i_1, h_0\}$
11. $\{j_1, e_0\}$
12. $\{k_0, d_0, g_0\}$
13. $\{k_1, l_1, m_1\}$
14. $\{m_0\}$

We use the first fault in a class as the representative of its class. For convenience, we assume that the fault-free circuit has an empty fault Φ. Figure 12.2(b) shows all the possible responses of the circuit to the given test set under the SSF model. Error values are marked by "*". Note that the test set does not distinguish between d_1 and i_1, or between g_1 and j_1.

For this simple example, we can arrange the fault dictionary as a mapping between the 12 distinct responses and the faults that can produce them. Thus if we obtain the response 00001, the dictionary will point to the faults $\{k_0, d_0, g_0\}$. □

To reduce the amount of data used for fault location, a fault dictionary does not store the entire response R_f caused by the fault f, but only a "signature" usually consisting of the list of errors contained in R_f. An error occurring in test t_i at output o_j is denoted by (i,j). In Example 12.1, the error caused by k_0 is (4,1). Other methods for reducing the size of a fault dictionary can be found in [Tulloss 1978, 1980].



(a)

	a b c	Φ	a_0	a_1	b_1	c_1	d_1	f_0	f_1	g_1	i_0	i_1	j_1	k_0	k_1	m_0
t_1	0 1 1	0	0	1*	0	0	1*	1*	0	0	0	1*	0	0	1*	0
t_2	1 1 0	0	1*	0	0	1*	0	0	1*	1*	0	0	1*	0	1*	0
t_3	1 0 1	0	1*	0	1*	0	0	0	1*	1*	0	0	1*	0	1*	0
t_4	1 1 1	1	0*	1	1	1	1	0*	1	1	1	1	1	0*	1	0*
t_5	0 0 1	1	1	0*	0*	1	1	1	0*	1	0*	1	1	1	1	0*

(b)

Figure 12.2 (a) Circuit (b) Applied tests and responses in the presence of faults

To reduce the large computational effort involved in building a fault dictionary, in fault simulation the detected faults are dropped from the set of simulated faults. Hence all the faults detected for the first time by the same vector at the same output will produce the same signature and will be included in the same equivalence class. In Example 12.1, a fault dictionary constructed in this way will not distinguish among the faults detected by t_1, namely $\{a_1, d_1, f_0, i_1, k_1\}$, even if most of them are further distinguished by subsequent tests. In this case the testing experiment can stop after the first failing test, because the information provided by the following tests is not used. Such a testing experiment achieves a lower diagnostic resolution. (A trade-off between computation time and diagnostic resolution can be achieved by dropping faults after $k>1$ detections.)

Figure 12.3 shows the possible results for Example 12.1 in the form of a *diagnostic tree*. The results of a test are indicated as *pass* (P) or *fail* (F). Every test distinguishes between the faults it detects and the ones it does not. (In a multioutput circuit, faults detected by the same test can be distinguished if they are detected at different outputs.) The set of faults shown in a rectangle are equivalent under the currently applied test set. Note that faults remaining undetected are equivalent to Φ.

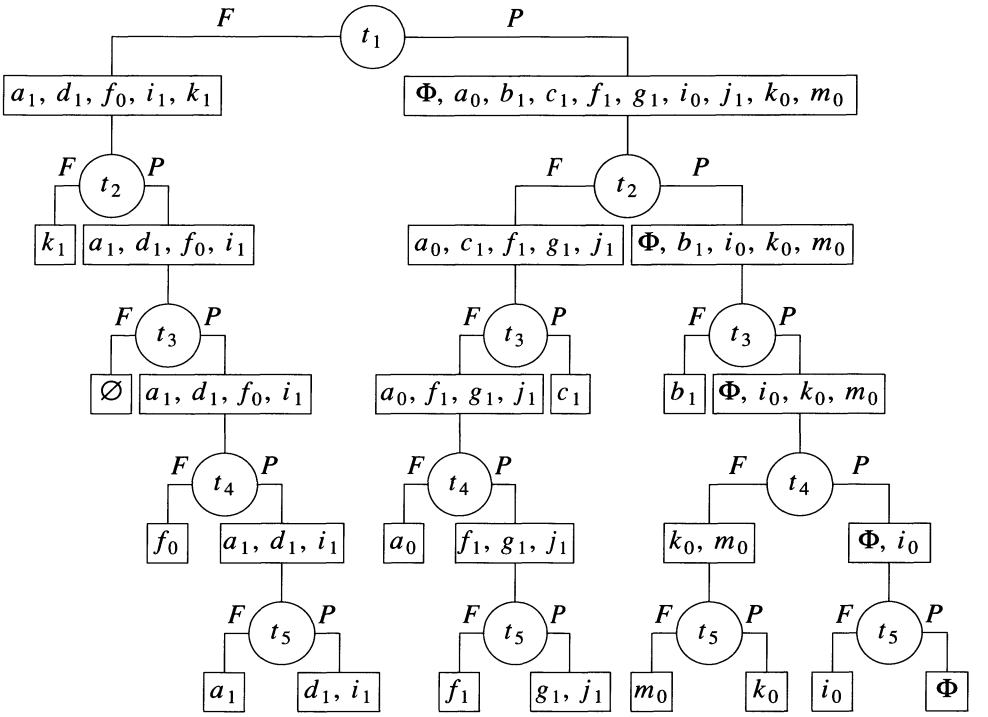


Figure 12.3 Diagnostic tree

From Figure 12.3 we can observe that some faults are uniquely identified before the entire test sequence is applied. For example, k_1 is the only fault detected in both t_1 and t_2; thus, if both t_1 and t_2 fail, the fault in the UUT is located to within the equivalence class $\{k_1, l_1, m_1\}$. Here the testing experiment can stop after the first two tests, because no more diagnostic information can be obtained from the following tests (according to the SSF model).

Rather than applying the entire test sequence in a fixed order, *adaptive testing* determines the next vector to be applied based on the results obtained by the preceding vectors. In our example, if t_1 fails and t_2 passes, the possible faults are $\{a_1, d_1, f_0, i_1\}$. At this point applying t_3 would be wasteful, because t_3 does not distinguish among these faults. The use of adaptive testing may substantially decrease the average number of tests required to locate a fault (see, for example, [Koren and Kohavi 1977]).

Generating Tests to Distinguish Faults

To improve the fault resolution of a test set T, it is necessary to generate tests to distinguish among faults equivalent under T. Note that the goal of having a high fault resolution is somehow at odds with the goal of minimizing the size of the test set, which requires every vector to detect as many faults as possible.

Consider the problem of generating a test to distinguish between two faults, f and g, in a combinational circuit. Such a test must detect f but not g on some output, or vice versa. The fault-oriented algorithms presented in Chapter 6 can be generalized to handle this problem, by using the following strategy:

Case 1: If f and g do not influence the same set of outputs, let O_f (O_g) be the set of outputs influenced by $f(g)$ and not by $g(f)$. Try to generate a test for f using only the circuit feeding the outputs O_f. If this fails, try to generate a test for g using only the circuit feeding the outputs O_g [Savir and Roth 1982].

Case 2: (Here $O_f = O_g = \emptyset$, or test generation for Case 1 has failed.) Try to generate a test for f without activating g (if g is the fault l *s-a-v*, this is simply done by imposing the additional constraint that the value of l should be v). If this fails, try to generate a test for g without activating f.

Case 3: (Here f and g are simultaneously activated.) Try to generate a test that propagates the effect of either f or g (but not both) to some primary output.

While Cases 1 and 2 require only minor changes to any fault-oriented test generation algorithm, Case 3 is more difficult, because the algorithm must process the fault effects of both f and g. We will present a generalization of the D-algorithm for this problem. Let us use the symbols D and $\overline{D}$ to represent effects of the fault f and the symbols E and $\overline{E}$ to represent effects of the fault g. Both these pairs of values are propagated in the same way when they interact with 0 and 1 signals and the objective is to propagate a D (or $\overline{D}$) or an E (or $\overline{E}$) but not both to some output. However, when these two values appear on different inputs to the same gate, some differences arise in defining how they propagate through the gate. Specifically it must be remembered that even though we are propagating two single faults, at most one fault is present. Thus if the inputs to a 2-input OR gate are D and $\overline{E}$, then the normal inputs are 1 and 0, the inputs with f present are 0 and 0, and the inputs with g present are 1 and 1. The normal gate output is 1, the output with f present is 0, and the output with g is 1. Hence only the D propagates to the output ($D + \overline{E} = D$). An AND gate with inputs D and E would have normal output 1 and output 0 with either f or g present, and hence both D and E propagate to the gate output; this is denoted by D,E ($D.E = D,E$). The tables of Figure 12.4 represent the signal propagations for AND and OR gates.

Note that when all the lines in the fault-propagation frontier have only D,E or $\overline{D},\overline{E}$ values (which means that the D-frontiers in the circuits N_f and N_g are identical), the algorithm must backtrack, because the fault effects cannot be propagated separately.

Example 12.2: For the circuit of Figure 12.5, we want to generate a test to distinguish the faults A *s-a*-1 and F *s-a*-0. These two faults influence the same output, so Case 1 does not apply. First we try to derive a test for A *s-a*-1 while setting F=0. By implication we obtain A=0, B=1, C=$\overline{D}$, H=$\overline{D}$. But if G=0, then J=$\overline{D}.D$=0; if G=1, then I=0 and J=0. Thus either case leads to a failure. Then we try to derive a test for F *s-a*-0 while setting A=1. By implication we obtain F=1, C=0, B=0, H=D, I=1, G=0, J=D. Thus the vector 1010 distinguishes the two faults. □

Example 12.3: For the circuit of Figure 12.5, we want to generate a test to distinguish the faults C_1 *s-a*-0 and C_2 *s-a*-0. Any attempt to activate the two faults separately fails, so Case 2 does not apply. The only way to activate the faults is C=1, which implies A=B=1. Then C_1=D and C_2=E. First we try to propagate the D by

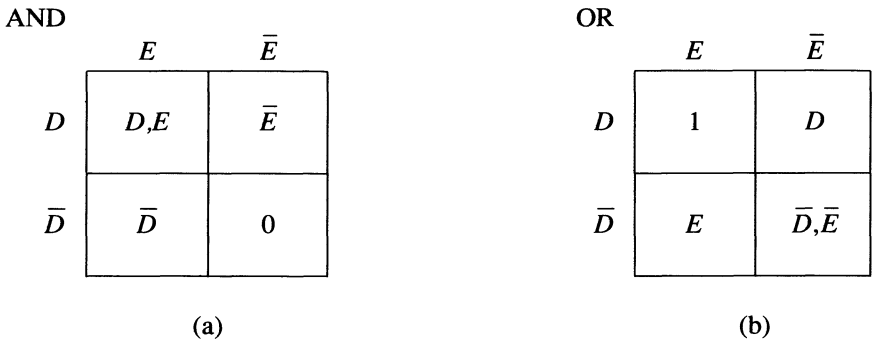


Figure 12.4

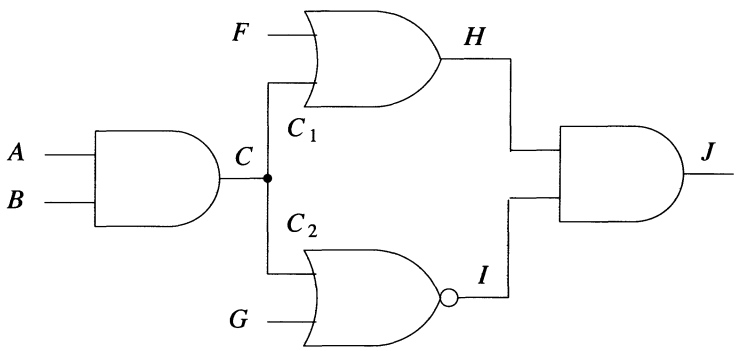


Figure 12.5

setting F=0. This implies H=D. Trying to inhibit the propagation of E by setting G=1 results in I=0 and J=0. We backtrack by setting G=0, which leads to I=$\bar{E}$ and J=$D.\bar{E}$=$\bar{E}$. Thus the vector 1100 distinguishes the two faults by propagating only one fault effect to the output. □

Problems with Fault Dictionaries

One problem with the fault-dictionary approach is the large computational effort involved in building fault dictionaries for large circuits tested with long test sequences. Fault simulation consumes much CPU time and storage, even when used with early fault dropping (typically on first detection). Moreover, early fault dropping results in lower diagnostic resolution.

A fault dictionary is constructed only for a specific fault universe, usually consisting of SSFs. A fault that is not equivalent under the applied test sequence to any of the simulated faults cannot be located via the fault dictionary, because its corresponding response does not match any response computed by fault simulation.

Example 12.4: Consider the AND bridging fault (*a.c*) in the circuit of Figure 12.2(a). The test set given in Figure 12.2(b) detects this fault in vectors t_1 and t_2. But none of the SSFs produces errors only in t_1 and t_2. Similarly, the multiple fault $\{b_1, i_1\}$ is detected in t_1 and t_3, but this does not match the signature of any SSF. □

When the signature of the UUT does not exactly match any of the precomputed signatures stored in the fault dictionary, location is based on closest-match analysis techniques. As the signature S_i derived from a response R_i is the set of errors contained in R_i, we can measure the closeness c_{12} of two signatures S_1 and S_2 by the amount of overlap between the two sets, that is,

$$c_{12} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

(Note that c_{12}=1 when S_1=S_2)

The diagnosis process selects as the most likely set of faults the one whose signature is closest to the signature of the UUT. While this heuristic may work in many cases, it is not guaranteed to produce a correct diagnosis. For example, the closest match for the signatures of both faults analyzed in Example 12.4 — the bridging fault (*a.c*) and the multiple fault $\{b_1, i_1\}$ — would be the signature of k_1; but this fault is totally unrelated to either (*a.c*) or $\{b_1, i_1\}$. (It is interesting to observe that the set of tests that detect $\{b_1, i_1\}$ is not the union of the sets of tests that detect b_1 and i_1 — $\{t_1,t_3,t_5\}$ — because i_1 masks b_1 under the test t_5.)

Another problem encountered by some testing systems based on the fault-dictionary approach is caused by potential detections. A potential detection occurs when the response generated by a fault contains an unknown (*u*) value, while the corresponding value in the fault-free circuit is binary. (This situation is typical for faults that prevent initialization). As the actual response of the UUT contains only binary values, an observed error should be interpreted as a match for a potential detection [Richman and Bowden 1985]. For example, consider a fault dictionary built from data including the values shown in Figure 12.6, and assume that the response of the UUT is 100. The signature of the UUT (with errors in t_1 and t_2) should match the signature of f_1 (error in t_1 and potential detection in t_2), but not that of f_2 (error only in t_1).

	Φ	f_1	f_2
t_1	0	1*	1*
t_2	1	u	1
t_3	0	0	u

Figure 12.6

12.3 Guided-Probe Testing

Guided-probe testing extends an edge-pin testing process by monitoring internal signals in the UUT via a *probe* which is moved (usually by an operator) following the guidance provided by the ATE. The principle of guided-probe testing is to backtrace an error from the primary output where it has been observed during edge-pin testing to its source (physical fault) in the UUT (see Figure 12.7). Typical faults located by guided-probe testing are opens and defective components. An open between two points A and B is identified by a mismatch between the error observed at B and the correct value measured at the signal source A. A faulty device is identified by detecting an error at one of its outputs, while only expected values are observed at its inputs.

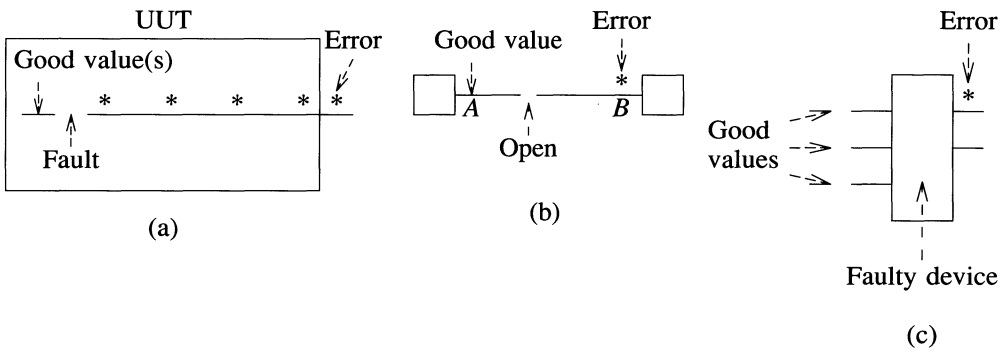


Figure 12.7 (a) Backtracing errors (b) Identifying an open (c) Identifying a faulty device

Unlike the fault-dictionary method, guided-probe testing is not limited to the SSF model. The concept of "defective component" covers any type of internal device fault detected by the applied test; this generality makes guided-probe testing independent of a particular fault model.

In addition to the applied stimuli and the expected response needed for edge-pin testing, ATE supporting guided-probe testing also needs the expected values of all the internal lines accessible for probing and the structural model of the UUT (see Figure 12.8). Because the backtrace is guided by the structural model, faults that create additional connections are inherently difficult to locate. For example, the faulty-device diagnosis illustrated in Figure 12.7(c) would be incorrect had the error on the device output been caused by a short.

The *Guided-probe* procedure outlined in Figure 12.9 recursively backtraces a given error along its propagation path and returns a diagnostic message when it reaches the location where the fault is activated. *Probe (j)* represents the action of moving the probe to line j. Initially, *Guided-probe* starts at a primary output and with the first test in which an error has been observed at that output. As long as the backtrace traverses a combinational region of the UUT, it would be sufficient to analyze only the values

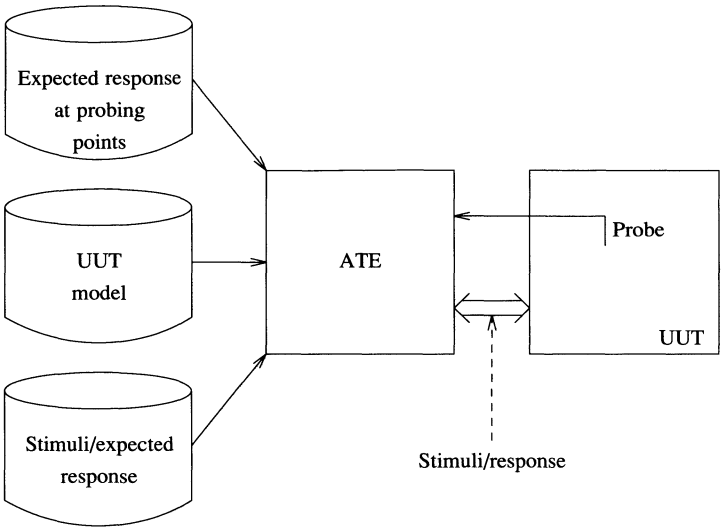


Figure 12.8 ATE with guided-probe testing

```
Guided-probe (i, t_r)
/* backtrace error observed at i in test t_r */
begin
    j = source of signal observed at i
    Probe (j)
    if no error at j in t_r then return ("open between j and i")
    /* error at j */
    if j is a PI then return ("faulty PI: j")
    m = the device whose output is j
    for every input k of m
        begin
            Probe (k)
            reapply (t_1, t_2, ..., t_r)
            if error at k in t_q (q≤r) then return Guided-probe (k, t_q)
        end
    /* no errors at the inputs of m */
    return ("faulty device: m")
end
```

Figure 12.9 Basic guided-probe procedure

existing after the last vector t_r has been applied, because in a combinational circuit the same vector activates the fault and propagates the resulting error. In a sequential circuit, however, fault activation and error propagation may be done by sequences.

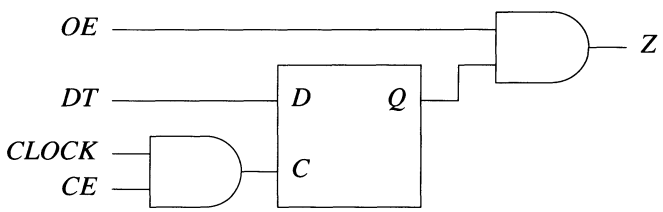
Hence when probing an input k of a device whose output has errors in t_r, the entire sequence (t_1, t_2, ..., t_r) must be reapplied, because the error could have been propagated to k in a vector t_q applied earlier ($q<r$). In such a case, backtracing continues with the earliest vector t_q that propagates an error to k. Although the reapplication of the sequence can be avoided for a combinational device, the test application time is usually negligible compared to the time involved in moving the probe.

Example 12.5: Consider the circuit and the test sequence given in Figure 12.10. The errors shown are caused by D *s-a-0*. (Note that the F/F input called D and the primary input DT are different probing points.) We assume that every signal is accessible for probing. Backtracing the error observed at Z in t_4, *Guided-probe* reaches the output Q of the F/F. Note that none of the F/F inputs has an error in t_4. Next the F/F inputs are probed while the entire sequence is reapplied, and the earliest error is found on the D input in t_2. The last line probed is the primary input DT. □

The procedure outlined in Figure 12.9 describes only the basic mechanism of guided-probe testing. In the following, we discuss the extensions it needs to handle feedback loops and wired logic.

Feedback Loops

The following example illustrates the problems encountered in circuits with feedback loops and the approach used to solve them.



(a)

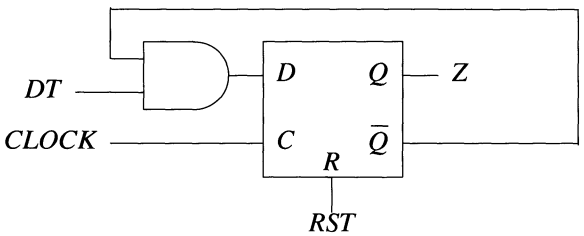
test	*CLOCK*	*CE*	*DT*	*OE*	*Q*	*Z*
t_1	↑	1	0	1	0	0
t_2	↑	0	1/0	1	0	0
t_3	↑	1	1/0	0	1/0	0
t_4	0	1	0	1	1/0	1/0

(b)

Figure 12.10

Example 12.6: Consider the circuit and the test sequence given in Figure 12.11. First assume that the fault is DT *s-a-1*, which causes errors in t_4 on DT, D, Z, and $\overline{Q}$. Backtracing the error observed at Z in t_4 leads to Q, D, and, assuming that the $\overline{Q}$ input

of the AND gate is probed before *DT*, the procedure again reaches the F/F. Hence to identify loops, one should keep track of the previously encountered devices. An identified loop is then retraced probing all the inputs of the devices in the loop (rather than following only the first input with an error). In this way, the procedure can determine whether the error originates within or outside the loop. In our example, retracing the loop and probing all the inputs of the AND gate will find *DT* to be the source of the errors along the loop.



(a)

test	*CLOCK*	*RST*	*DT*	*D*	*Z*
t_1	0	1	1	1	0
t_2	↑	0	1	1	1
t_3	↑	0	1	0	0
t_4	↑	0	0	0	0

(b)

Figure 12.11

Now assume that the fault is *D s-a-*1, which causes *D*, *Z* and $\overline{Q}$ to have errors in t_3 and t_4. After the loop is identified, probing all the inputs of the devices in the loop shows that the error originates within the loop. To further improve the diagnostic resolution, one needs to observe the values of the probed lines both before and after the CLOCK pulse is applied. This will show that the error at *D* in t_3 precedes the CLOCK pulse, which propagates it around the loop. (Alternatively, we can consider that the clock and data changes define separate vectors.) □

Wired Logic

One difficulty in backtracing errors through a "device" created by wired logic is that the inputs and the output of such a device form the same physical entity and hence cannot be separately probed. Also the previously used criterion for identifying a faulty device — error on output and good values on inputs — is no longer valid for a device driving wired logic, because the output error may propagate from another output connected to the same wire. For example, in Figure 12.12 it would be incorrect to identify gate *E* as faulty. The solution in such a case is to probe the inputs of all the

driving devices as if they were inputs of the wired gate. In Figure 12.12, this strategy will result in continuing the backtrace from C.

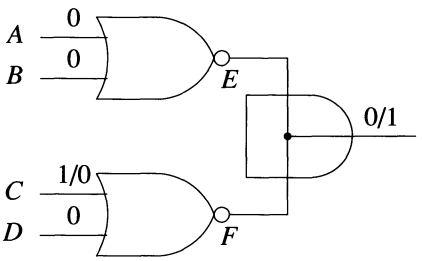


Figure 12.12

If no errors are found while probing the inputs of the driving devices, the conventional guided-probe testing strategy cannot distinguish among these devices. In the circuit of Figure 12.12, if E=F=1, while (A,B,C,D) have their expected values 0010, the obtained diagnostic resolution is $\{E,F\}$. This can be further improved by additional analysis. For example, we can observe that a failure of gate E (driving a 0 instead of 1) cannot produce the observed result, which can be explained only by a failure of gate F (outputting a 1 instead of 0).

Speed-up Techniques

The most time-consuming part of guided-probe testing is moving the probe. Hence to speed up the fault-diagnosis process, one needs to reduce the number of probed lines. Here are some techniques used to achieve this goal [Kochan *et al.* 1981]:

- Rather than probing one pin at a time, simultaneously probe all the pins of a device using an IC clip. If several inputs have errors, then follow the one with the earliest error (this also reduces the amount of probing for tracing feedback loops).
- Skip probing the output of a suspected device and directly probe its inputs. For the example in Figure 12.13, after observing an error at i, one would directly probe the inputs of m. The output j would be probed only if no errors are detected at the inputs, to distinguish between the faulty device m and an open between j and i.
- Probe only those device inputs that can affect the output with errors (see Figure 12.14).
- Among the inputs that can influence the output with errors, probe first the control lines; if no errors are detected at the control lines, then probe only those data inputs enabled by the values of the control lines. For the multiplexor shown in Figure 12.15, if no errors are found at the select inputs S_0 and S_1, the next line probed is D_3 (because it is the input selected by (S_0,S_1) = 11).
- Use a fault dictionary to provide a starting point for probing.

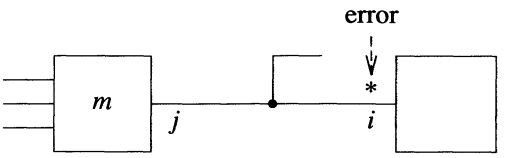


Figure 12.13

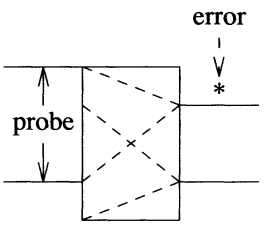


Figure 12.14

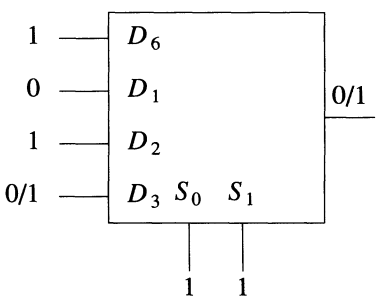


Figure 12.15

Techniques similar to those used in guided-probe testing are used in electron-beam testing [Tamama and Kuji 1986].

12.4 Diagnosis by UUT Reduction

The *C*ontrollability, *O*bservability and *M*aintenance *E*ngineering *T*echnique (COMET) developed by Chang and Heimbigner [1974] is based on a design for diagnosability technique which requires selective disabling of portions of the UUT, coupled with the

ability of testing only the enabled part. With these provisions, the testing and diagnosis process follows the diagram shown in Figure 12.16.

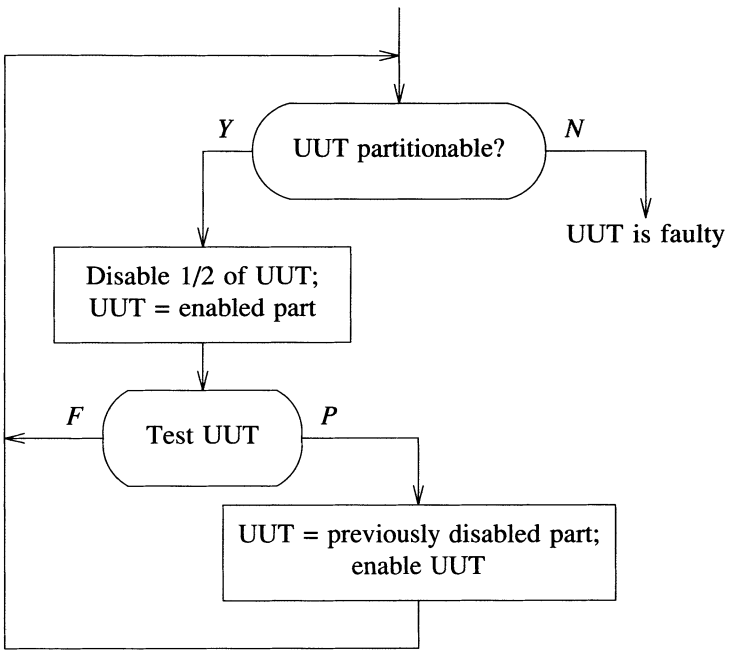


Figure 12.16 Testing and diagnosis process in COMET

Initially, the UUT is the entire circuit and the process starts when its test fails. While the failing UUT can be partitioned, half of the UUT is disabled and the remaining half is tested. If the test passes, the fault must be in the disabled part, which then becomes the UUT.

COMET has been applied to processor-level testing, where every board in the processor can be independently disabled, either by physically disconnecting it or by setting its outputs to a logically inactive state.

Example 12.7: Consider a processor composed of four boards, referred to as A, B, C, and D. Assume that B is the faulty board. After the fault-detection tests for the entire processor fail, C and D are disabled and fault-detection tests for A and B are run. These tests fail and then B is disabled and the tests for A are run. These tests pass and implicate B as the faulty unit. □

Note that COMET does not require fault-location tests. Applying fault-detection tests to the enabled part of the UUT, and the ability to reduce the UUT progressively, are sufficient to locate faults.

12.5 Fault Diagnosis for Combinational Circuits

The fault-location methods discussed in this section are applicable to combinational circuits and also to sequential circuits that are transformed into combinational ones during testing using one of the scan-design techniques presented in Chapter 9.

Structural Analysis

If we assume a single fault that does not create new connections, then there exists a path from the site of the fault to each of the outputs where errors have been detected. Hence the fault site belongs to the intersection of the cones of all the failing outputs (Figure 12.17).

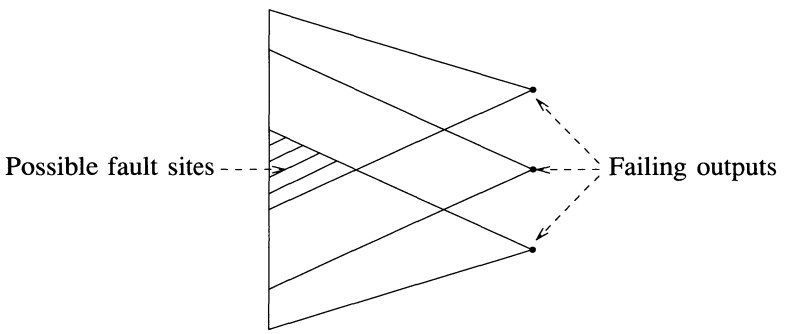


Figure 12.17 Cone intersection

(This observation is also true in a sequential circuit, but its application is more difficult because there a propagation path may span several time frames). Unlike fault dictionaries, which rely only on the first error, this simple structural analysis tries to find faults that can explain all the observed errors, and it often yields acceptable diagnostic resolution [Hsu *et al.* 1981].

If we restrict ourselves to the SSF model, the set of suspected faults can be further reduced by taking into account the inversion parities of the paths between the fault site and the failing POs. Let l be a possible fault site. The fault l *s-a-v* is a *plausible* fault only if for every error value v_o obtained at an output o, there exists a path with inversion parity $v \oplus v_o$ between l and o.

Example 12.8: When testing the circuit shown in Figure 12.18, assume that 1/0 errors are observed at E and F. Then the fault site must be in the subcircuit whose output is D (note that this does not include $C2$). The fault C *s-a*-0 is excluded because the unique path between C and E has odd inversion. The plausible faults are C *s-a*-1, B *s-a*-1, $C1$ *s-a*-1, and D *s-a*-0. □

Fault Simulation

Once a set of plausible faults has been identified by structural analysis, these faults can then be simulated to select the ones that generate a response identical to the actual response of the UUT [Arzoumanian and Waicukauski 1981]. In this simulation, one compares the output errors produced by every fault with the errors observed during the

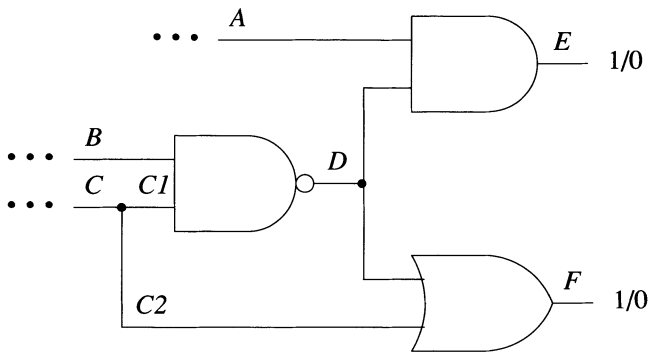


Figure 12.18

testing. The comparison is done vector by vector, and every fault that mismatches is dropped from the set of simulated faults. The faults remaining after all vectors have been simulated are consistent with the observed symptom, and they are equivalent under the applied test. The actual fault belongs to this equivalence class. Compared to building a fault dictionary, this fault simulation has the advantage of processing a much smaller set of faults. Also the resulting fault resolution is generally higher because now faults are required to account for all the observed errors.

Example 12.9: Consider again the circuit and the test given in Figure 12.2. Assume that tests t_1 and t_4 failed. Because the error value is 1 in t_1 and 0 in t_4, there must exist two propagation paths with different inversion parities from the fault site to the output m. Hence the fault site is one of the lines $\{a,b,c,f\}$ and the set of plausible faults is $\{a_0,a_1,b_1,c_1,f_0,f_1\}$. Only a_1 and f_0 make t_1 fail, so the other faults are dropped after simulating t_1. Then only f_0 produces an error in t_4, and we conclude that f_0 is the actual fault. By contrast, the result obtained using a fault dictionary built with fault dropping on the first error would be the class $\{a_1,d_1,f_0,i_1,k_1\}$. □

Note that it is possible that none of the plausible faults can reproduce the response obtained from the UUT. Such a result shows that the actual fault contradicts the initial assumptions made about its nature (single fault that does not create new connections). For example, the actual fault could be a multiple stuck fault or a bridging fault that is not equivalent under the applied test with any SSF in the circuit.

12.6 Expert Systems for Diagnosis

Expert systems are programs applying Artificial Intelligence techniques in an attempt to emulate the problem-solving ability of a human expert in a specialized field [Nau 1983]. An expert system relies on a *knowledge base*, which is a data base representing the knowledge applied by a human expert, coupled with an *inference engine* which searches the knowledge base to retrieve and apply the information pertinent to the problem to be solved.

Expert systems for diagnosis [Hartley 1984, Laffey *et al.* 1986, Mullis 1984, Wilkinson 1984] are built by encoding the techniques followed by expert troubleshooters to locate a fault, given certain error symptoms. Because these diagnosis techniques are specific to one target system, such expert systems have been developed mainly for high-priced, complex digital systems, where they can provide substantial economic benefits accumulated during the lifetime of the system. An important advantage of an expert system is that it makes the expert's knowledge instantly available in the field, thus allowing diagnosis to be performed by less skilled personnel. In general, the use of an expert system leads to a faster and more accurate fault-location process, and hence it minimizes the downtime of the target system.

Most of the knowledge of an expert system is a collection of *if-then rules* (such systems are also referred to as *rule-based* systems). The general form of a rule is

if (condition) **then** action(s)

In the following example, the condition is based on some of the observed symptoms and the action suggests specific RUs as suspects (likely to be faulty):

if (*ALU-test* failed **and** *register-test* passed) **then** (suspect board *A* or board *B*)

Usually the tests are grouped according to the part of the system they exercise (i.e., *ALU-test*) and the symptoms relate to the overall results of groups of tests, rather than individual vectors. Faults are located only to large field-RUs (typically boards).

The conditions can also relate to the current state of the diagnosis process (that is, which RUs are currently suspected). Other actions may rule out a previously suspected RU, or they may request the application of additional tests or the replacement of a suspected RU with a fault-free one and rerunning of some tests. For example:

if (board *A* is suspect **and** board *B* is suspect)
then (replace board *A*; apply *ALU-test*)
if (*Data-Path-test* passed) **then** (assume board *C* is fault-free).

Additional knowledge may be included in the knowledge base to represent

- structural information about the target system (its RUs and their interconnections);
- relations between RUs and tests (which RUs are checked by which tests);
- general assumptions about the nature of the fault (no more than one faulty RU, etc.)

Figure 12.19 illustrates the general structure of an expert system for diagnosis. The diagnosis data consist of the known facts (symptoms obtained so far, currently suspected RUs). Based on these facts, the inference engine repeatedly searches the knowledge base to find the applicable rules, which are then used to establish new facts or to control the testing experiment to obtain new facts. The expert system controls the ATE directly and/or by interacting with an operator.

The inference engine may reason using *forward chaining* — reaching conclusions by finding and applying the rules whose conditions are true in the current state — or *backward chaining* — starting with a hypothesis and searching for supporting facts to confirm it. Some systems can combine the two strategies.

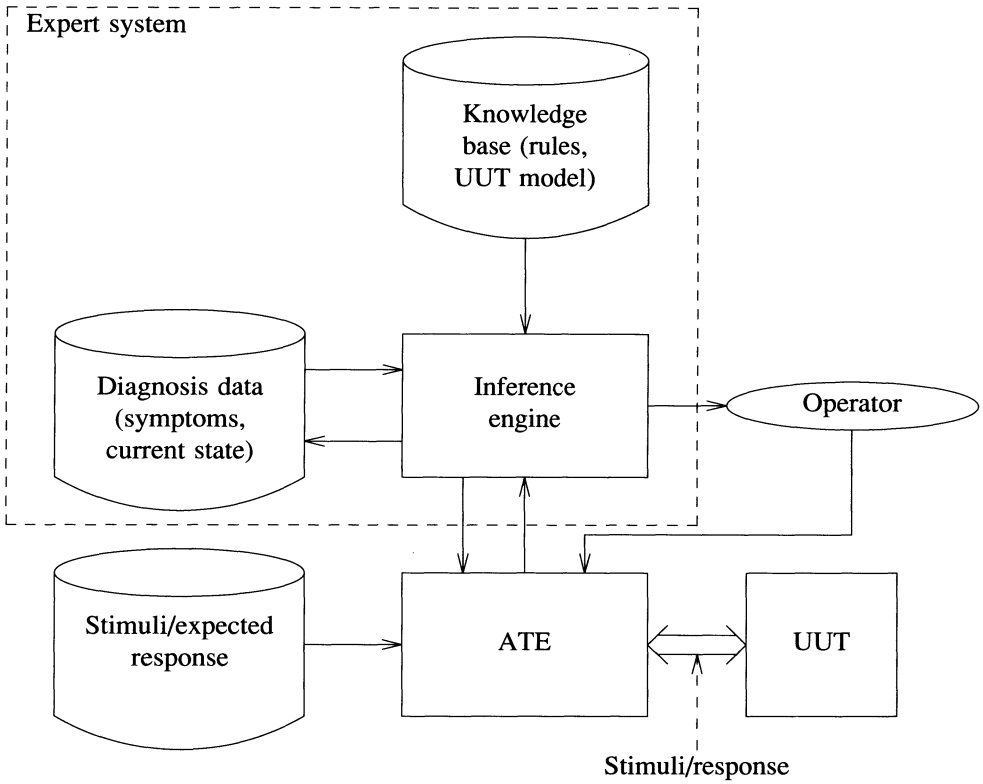


Figure 12.19 Expert system for diagnosis

A limitation of rule-based expert systems for diagnosis is their use of *shallow knowledge* — a collection of empirical associations that are not directly related to the intended behavior of the target system. This is why such expert systems cannot successfully process a new symptom that does not match the ones stored in the knowledge base.

12.7 Effect-Cause Analysis

The *effect-cause analysis* [Abramovici and Breuer 1980] processes the response obtained from the UUT to determine the possible stuck-at faults that can generate that response, based on *deducing internal values* in the UUT (Figure 12.20).

Any line for which both 0 and 1 values are deduced can be neither *s-a*-1 nor *s-a*-0, and it is identified as *normal* (fault-free). Faults are located on some of the lines that could not be proved normal. The ensemble of normal and faulty lines is referred to as a *fault situation*. The effect-cause analysis employs an *implicit multiple stuck-fault model*, and it does not require fault enumeration or precomputing faulty responses.

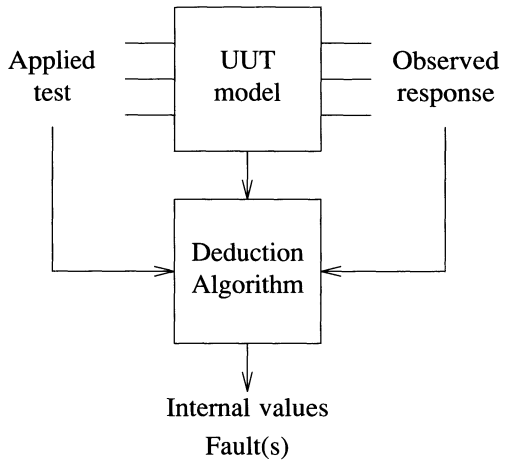


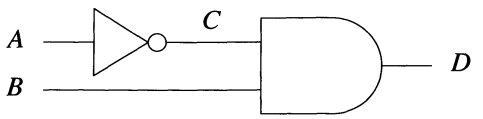
Figure 12.20 Effect-cause analysis

Internal values are computed by the *Deduction Algorithm*, which implements a line-justification process whose primary goal is to justify all the values obtained at the POs, given the tests applied at the PIs. The values of a normal PI are the values set by the applied tests, and the values of every other normal line must be justified by values of its predecessors. The following example introduces the basic concepts of the Deduction Algorithm.

Example 12.10: Consider the circuit and the test shown in Figure 12.21.

Assume that the observed response is D=101. Because D has both 0 and 1 values, D is normal. Then its values must be justified by those of B and C. First D=1 in t_1 implies B=C=1 in t_1. The value 1 deduced for B shows that B is not s-a-0; therefore in every test that applies 1 to the PI B, B will have value 1. Hence B=1 in t_1 implies B=1 in t_2. This, in turn, implies C=0 in t_2 (because D=0 in t_2 and D is normal). At this point, both 0 and 1 values have been deduced for C, hence C is a normal line. Then all the known values of C (in t_1 and t_2) are complemented and assigned to A. Because A is a PI, A=0 in t_1 shows that A will have value 0 in every test that sets A=0; hence A=0 in t_3. Then C=1 in t_3. From D=1 in t_3 we determine B=1 in t_3. Because t_3 applies a 0 value to B, this shows that B is s-a-1. □

In this simple example, all internal values have been deduced via a chain of implications. The implications involving values of different lines in the same test — called *horizontal implications* — are similar to the backward and the forward implications described in Chapter 6, except that they apply only to lines identified as normal. The example also illustrates a new type of implications — called *vertical implications* — which involve values of the same line in different tests. Vertical implications are based on the concept of *forced value* (FVs).



(a)

	A	B
t_1	0	1
t_2	1	1
t_3	0	0

(b)

	D
t_1	1
t_2	0
t_3	1

(c)

	A	B	C	D
t_1	0	1	1	1
t_2	1	1	0	0
t_3	0	1	1	1

(d)

Figure 12.21 (a) Circuit (b) Applied test (c) Obtained response (d) Deduced values

Definition 12.1: Let v be the expected value of line l in test t. We say that v is a *forced value*, if for every possible fault situation, either l has value v in t, or else l has value $\overline{v}$ in every test.

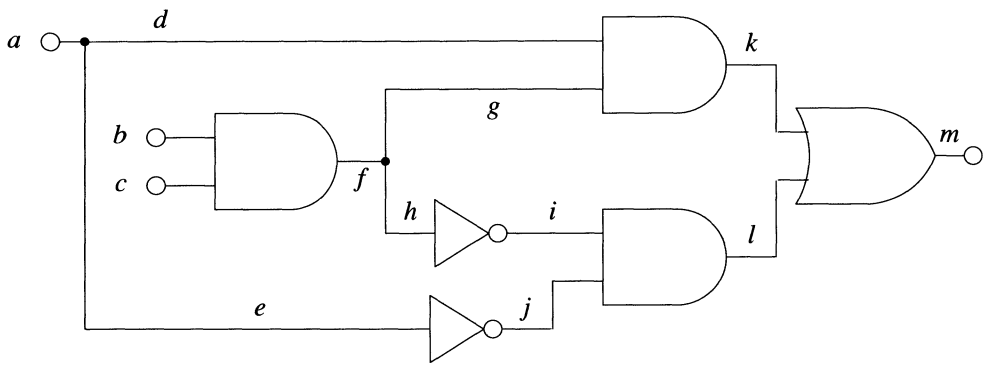
Clearly, all the expected values of a PI are FVs. In the previous example, either A=0 in t_1 (if A is normal or s-a-0), or else A=1 in every test (if A is s-a-1). The FVs are not restricted only to PIs. If C is normal, C propagates the values from A; hence either C=1 in t_1, or else C=0 in every test. The statement "C=1 in t_1, or else C=0 in every test" is also true when C is s-a-1 or s-a-0 and therefore is true for any fault situation involving A and C. Since the fault situation of the other lines does not affect C, the statement is true for any fault situation in the circuit; hence C has FV 1 in t_1. The same reasoning can be applied for t_2 or t_3; thus we can conclude that the FVs of the output of an inverter are the complements of the FVs of its input. Because a normal fanout branch takes all the values of its stem, it is easy to show that a fanout branch has the same FVs as its stem. The propagation of FVs through multi-input gates is based on the following theorem.

Theorem 12.1: If all inputs of a gate G have the same FV in a test t, than the corresponding output value of G is a FV in t.

Proof: Without loss of generality, assume that G is an AND gate. First assume that all inputs of G have FV 1 in t. If G is normal, its value is determined by its inputs; then either G=1 in t (if all its inputs are 1), or else G=0 in every test (if at least one input is 0 in every test). The same is true if G is s-a-1 or s-a-0, hence G has FV 1 in t. Now assume that all inputs of G have FV 0 in t. Then either at least one input is 0 in t, or else all inputs are 1 in every test. If G is normal, then either G=0 in t, or else G=1 in every test. The same is true if G is s-a-0 or s-a-1, hence G has FV 0 in t. Therefore, if all inputs of G have the same FV in t, the corresponding output value of G is also a FV in t. □

The following example illustrates the computation and the use of FVs by the Deduction Algorithm.

Example 12.11: Consider the circuit and tests given in Figure 12.22. Figure 12.22(c) shows the FVs computed as a preprocessing step, starting with the FVs of the PIs (ignore now the values in parentheses).



(a)

	a	b	c
t_1	0	1	1
t_2	1	1	0
t_3	1	0	1
t_4	1	1	1
t_5	0	0	1

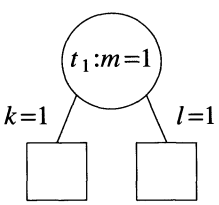
(b)

	a	b	c	d	e	f	g	h	i	j	k	l	m
t_1	0	1	1	0	0	1	1	1	0	1			
t_2	1	1	0	1	1			(0)	(1)	0			
t_3	1	0	1	1	1			(0)	(1)	0			
t_4	1	1	1	1	1	1	1	1	0	0	1	0	
t_5	0	0	1	0	0			(0)	(1)	1		(1)	

(c)

	a	b	c	d	e	f	g	h	i	j	k	l	m
t_1	0	1	1		0	1	1			1			1
t_2	1	1	0	1	1	0	0		1	0	0	0	0
t_3	1	0	1	1	1	0	0		1	0	0	0	0
t_4	1	1	1	1	1	1	1			0	1	0	1
t_5	0	0	1		0	0	0		1	1	0	1	1

(d)



(e)

Figure 12.22 (a) Circuit (b) Applied tests (c) Forced values (d) Obtained response and initial implications (e) Decision tree

Assume that the obtained response is 10011. Figure 12.22(d) shows the implications obtained by processing this response. (A good way to follow this example is to mark values as they are deduced.) First we determine that m is normal. In t_2 and t_3, m=0

implies k=l=0. The FV 0 of l in t_4 means that either l=0 in t_4, or else l=1 in every test. Since 0 values have been deduced for l, it follows that l=0 in t_4 (this is an instance of vertical implication). This in turn implies k=1 in t_4. Hence k is normal. From k=1 in t_4, we deduce d=g=1, both of which generate vertical implications, namely d=1 in t_2 and t_3, and g=1 in t_1. In t_2 and t_3, d=1 implies g=0 (to justify k=0). Then g becomes normal, and all its known values are assigned to its stem f, which is also identified as normal. Next, f=1 in t_1 implies b=c=1, which, by vertical implications, show that b=1 in t_2 and t_4, and c=1 in t_3, t_4 and t_5. Then f=0 in t_2 is justified by c=0, and f=0 in t_3 by b=0. The latter implies b=0 in t_5, which propagates forward to make f=g=k=0. From k=0 and m=1 in t_5 we deduce l=1, which shows that l is normal. Then l=1 implies i=j=1 in t_5, and j=1 in t_5 implies j=1 in t_1.

Known values of a stem generate new FVs on its fanout branches. For example, f=0 in t_2 shows that either h=0 in t_2 or else h=1 in every test; hence h has FV 0 in t_2. Then the deduced values of f in t_2, t_3 and t_5 propagate as FVs on h and further on i. (The new FVs — enclosed in parentheses — are added to those in Figure 12.22(c)). Because i=1 in t_5, i must also have 1 values in all tests where it has FV 1; hence i=1 in t_2 and t_3. From these we deduce j=0 in t_2 and t_3; then j=0 in t_4 is obtained by vertical implication. Now j becomes normal, and its values are propagated to e and to a.

Now all the values given in Figure 12.22(d) have been deduced and no more implications can be made. The only value not yet justified is m=1 in t_1. First we try k=1 (see the decision tree in Figure 12.22(e)); this implies d=1 in t_1, which results in d=1 in t_5 (because d has FV 0 in t_1). At this point we have obtained a *solution*, i.e., a consistent set of values where all known values of the normal lines are justified. The only lines not reported as normal are d, i, and h. Because d=1 in every test, d is identified as s-a-1. (Moreover, because a is normal, we can locate an *open* between a and d). In addition, i may also be s-a-1, as only 1 values have been deduced for i. A second solution is obtained by justifying m=1 in t_1 by l=1; this identifies i s-a-1 as another fault that can generate the analyzed response. □

Unlike conventional diagnosis methods, *the effect-cause analysis does not use the concept of error*; in the above examples, faults were located without knowing the expected output values. The effect-cause analysis has also been extended to sequential circuits [Abramovici and Breuer 1982], where, because it does not require predictable errors, it can diagnose even faults that prevent initialization.

The Deduction Algorithm tries to justify simultaneously all the values obtained at the POs. Like the line justification algorithms described in Chapter 6, it alternates between implications and decisions. Backtracking is used either to recover from incorrect decisions, or to generate all solutions. Every solution identifies a possible fault situation in the circuit under test. When the Deduction Algorithm fails to obtain any solution, the processed response cannot be generated by any (single or multiple) stuck-at fault. Here the effect-cause analysis recognizes the presence of a fault not consistent with the assumed stuck-fault fault model, such as a bridging fault or a functional fault.

If the processed response happens to be the expected one, then one solution identifies every line as normal. If the Deduction Algorithm generates additional solutions, they *identify faults not detected by the applied tests.*

The effect-cause analysis can also help guided-probe testing [Abramovici 1981]. This application uses only the implication part of the Deduction Algorithm and replaces the decision process (and its associated backtracking) with probing. In other words, instead of making decisions about values on the inputs of a gate whose value is not justified, we probe them (if possible). This guided-probe testing algorithm achieves a higher diagnostic resolution than conventional methods and requires less probing points.

12.8 Diagnostic Reasoning Based on Structure and Behavior

Like the effect-cause analysis, the diagnosis technique described by Davis [1984] analyzes the observed response of the UUT, but it is more general since it is not limited to a particular fault model. Unlike a rule-based expert system using shallow knowledge about a specific digital system, this method is general and it uses *deep knowledge* based on the structure and the behavior of the circuit under test. Such a diagnosis approach relying on *model-based reasoning* is also referred to as a *first principles* approach.

The behavior of the components in the structural model of the UUT is described by *simulation rules* and *inference rules*. Simulation rules represent input-to-output value mapping, while inference rules correspond to backward implication. The behavior of a component is considered as introducing a *constraint* on the values of its inputs and outputs, and the entire circuit is regarded as a *constraint network*.

The initial analysis makes the following assumptions.

- The interconnections are fault-free and only the components can be faulty.
- No more than one module is faulty.
- Faulty components do not increase their number of states.

(These simplifying assumptions are explicit and they can be relaxed later.)

The first step is to determine an initial set of suspected components, that is, components whose incorrect behavior may explain the output errors. Based on the single faulty-module assumption, this step is done by intersecting the sets of components affecting the outputs with errors.

The next step is checking the validity of every suspected component by a technique called *constraint suppression*. It consists of temporarily suspending all the behavior rules of a suspected component and checking whether the resulting behavior of the circuit is consistent with the observed response.

Example 12.12: Consider the circuit in Figure 12.23, composed of three multipliers (M1, M2, and M3) and two adders (A1 and A2) interconnected by busses. The values shown are the decimal representations of the bus values.

Assume that instead of the expected value $F = 12$, in the first test we obtain the erroneous response $F = 10$ (Figure 12.23(a)). The suspected components affecting the output F are A1, M2, and M1. First we check whether A1 can be the faulty component by suspending the constraints introduced by A1 (i.e., we disable the

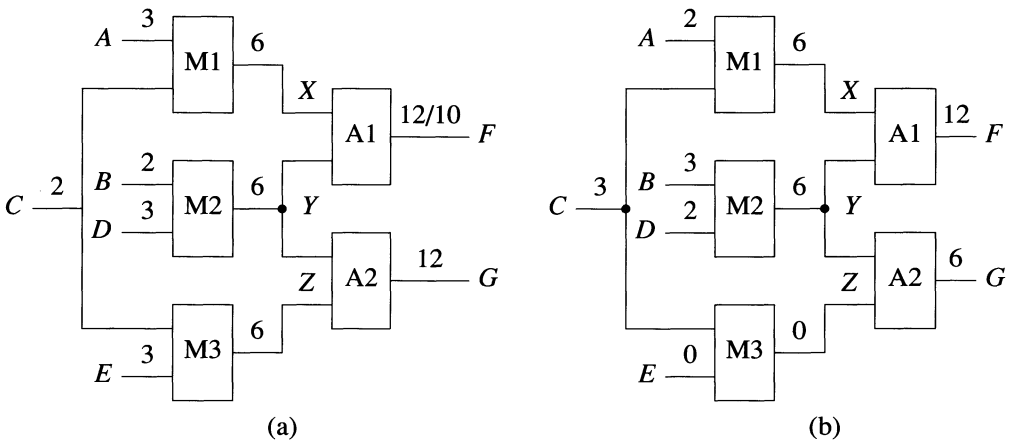


Figure 12.23

relation $F = X + Y$). Since the other components work correctly, the symptom describing the misbehavior of A1 is $\{X=6, Y=6, F=10\}$. In the second test (Figure 2.23(b)), A1 receives the same inputs ($X=6$, $Y=6$), and based on its behavior observed in the first test, we expect $F = 10$. The observed value, however, is the correct one ($F=12$). Based on the assumption that the faulty A1 is still a combinational component, we reached a contradiction which proves that A1 cannot be faulty.

Next, we restore the correct rules of A1 and check whether M2 can be faulty. Since M1 is fault-free, we deduce $X = 6$. Since A1 is fault-free, from $X = 6$ and $F = 10$ we infer that $Y = F - X = 4$. But $Y = 4$, $Z = 6$, and A2 fault-free imply $G = 10$, which contradicts the observed value $G = 12$. Therefore M2 cannot be faulty.

The last suspected component is M1. Since M2 is fault-free, we have $Y = 6$ in the first test. From $F = 10$, $Y = 6$, and A1 fault-free we infer $X = F - Y = 4$. Thus the symptom describing the misbehavior of M1 is $\{A=3, C=2, X=4\}$. (In the second test, however, M1 works correctly). Thus we conclude that M1 is faulty. If we have a hierarchical structural model, the analysis can continue inside M1 using a lower-level structural model of the multiplier. □

If no consistent explanation of the observed response can be found, the reasoning system is capable of relaxing the initial assumptions and repeat the analysis. For example, if no single faulty component can generate the observed response, the system may assume that two components are simultaneously faulty, and repeat the constraint suppression analyzing pairs of suspected components.

Other systems performing model-based diagnosis are described in [Genesereth 1984] and [Purcell 1988].

REFERENCES

[Abramovici 1981] M. Abramovici, "A Maximal Resolution Guided-Probe Testing Algorithm," *Proc. 18th Design Automation Conf.*, pp. 189-195, June, 1981.

[Abramovici 1982] M. Abramovici, "A Hierarchical, Path-Oriented Approach to Fault Diagnosis in Modular Combinational Circuits," *IEEE Trans. on Computers*, Vol. C-31, No. 7, pp. 672-677, July, 1982.

[Abramovici and Breuer 1980] M. Abramovici and M. A. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," *IEEE Trans. on Computers*, Vol. C-29, No. 6, pp. 451-460, June, 1980.

[Abramovici and Breuer 1982] M. Abramovici and M. A. Breuer, "Fault Diagnosis in Synchronous Sequential Circuits Based on an Effect-Cause Analysis," *IEEE Trans. on Computers*, Vol. C-31, No. 12, pp. 1165-1172, December, 1982.

[Arzoumanian and Waicukauski 1981] Y. Arzoumanian and J. Waicukauski, "Fault Diagnosis in an LSSD Environment," *Digest of Papers 1981 Intn'l. Test Conf.*, pp. 86-88, October, 1981.

[Chang and Heimbigner 1974] H. Y. Chang and G. W. Heimbigner, "LAMP: Controllability, Observability, and Maintenance Engineering Technique (COMET)," *Bell System Technical Journal*, Vol. 53, No. 8, pp. 1505-1534, October, 1974.

[Davis 1984] R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, Vol. 24, pp. 347-410, December, 1984.

[Genesereth 1984] M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence*, Vol. 24, pp. 411-436, December, 1984.

[Groves 1979] W. A. Groves, "Rapid Digital Fault Isolation with FASTRACE," *Hewlett-Packard Journal*, pp. 8-13, March, 1979.

[Hartley 1984] R. T. Hartley, "CRIB: Computer Fault-Finding Through Knowledge Engineering," *Computer*, Vol. 17, No. 3, pp. 76-83, March, 1984.

[Havlicsek 1986] B. L. Havlicsek, "A Knowledge Based Diagnostic System for Automatic Test Equipment," *Proc. Intn'l. Test Conf.*, pp. 930-938, September, 1986.

[Hsu *et al.* 1981] F. Hsu, P. Solecky, and R. Beaudoin, "Structured Trace Diagnosis for LSSD Board Testing — An Alternative to Full Fault Simulated Diagnosis," *Proc. 18th Design Automation Conf.*, pp. 891-897, June, 1981.

[Kime 1979] C. R. Kime, "An Abstract Model for Digital System Fault Diagnosis," *IEEE Trans. on Computers*, Vol. C-28, No. 10, pp. 754-767, October, 1979.

[Kochan *et al.* 1981] S. Kochan, N. Landis, and D. Monson, "Computer-Guided Probing Techniques," *Digest of Papers 1981 Intn'l. Test Conf.*, pp. 253-268, October, 1981.

[Koren and Kohavi 1977] I. Koren and Z. Kohavi, "Sequential Fault Diagnosis in Combinational Networks," *IEEE Trans. on Computers*, Vol. C-26, No. 4, pp. 334-342, April, 1977.

[Laffey *et al.* 1986] T. J. Laffey, W. A. Perkins, and T. A. Nguyen, "Reasoning About Fault Diagnosis with LES," *IEEE Expert*, pp. 13-20, Spring, 1986.

[Mullis 1984] R. Mullis, "An Expert System for VLSI Tester Diagnostics," *Proc. Intn'l. Test Conf.*, pp. 196-199, October, 1984.

[Nau 1983] D. S. Nau, "Expert Computer Systems," *Computer*, Vol. 16, No. 2, pp. 63-85, February, 1983.

[Purcell 1988] E. T. Purcell, "Fault Diagnosis Assistant," *IEEE Circuits and Devices Magazine*, Vol. 4, No. 1, pp. 47-59, January, 1988.

[Rajski 1988] J. Rajski, "GEMINI — A Logic System for Fault Diagnosis Based on Set Functions," *Digest of Papers 18th Intn'l. Symp. on Fault-Tolerant Computing*, pp. 292-297, June, 1988.

[Richman and Bowden 1985] J. Richman and K. R. Bowden, "The Modern Fault Dictionary," *Proc. Intn'l. Test Conf.*, pp. 696-702, November, 1985.

[Savir and Roth 1982] J. Savir and J. P. Roth, "Testing for, and Distinguishing Between Failures," *Digest of Papers 12th Annual Intn'l. Symp. on Fault-Tolerant Computing*, pp. 165-172, June, 1982.

[Shirley and Davis 1983] M. Shirley and R. Davis, "Generating Distinguishing Tests Based on Hierarchical Models and Symptom Information," *Proc. Intn'l. Conf. on Computer Design*, pp. 455-458, October, 1983.

[Tamama and Kuji 1986] T. Tamama and N. Kuji, "Integrating an Electron-Beam System into VLSI Fault Diagnosis," *IEEE Design & Test of Computers*, Vol. 3, No. 4, pp. 23-29, August, 1986.

[Tendolkar and Swann 1982] N. N. Tendolkar and R. L. Swann, "Automated Diagnostic Methodology for the IBM 3081 Processor Complex," *IBM Journal of Research and Development*, Vol. 26, No. 1, pp. 78-88, January, 1982.

[Tulloss 1978] R. E. Tulloss, "Size Optimization of Fault Dictionaries," *Digest of Papers 1978 Semiconductor Test Symp.*, pp. 264-265, October, 1978.

[Tulloss 1980] R. E. Tulloss, "Fault Dictionary Compression: Recognizing When a Fault May Be Unambiguously Represented by a Single Failure Detection," *Digest of Papers 1980 Test Conf.*, pp. 368-370, November, 1980.

[Wilkinson 1984] A. J. Wilkinson, "A Method for Test System Diagnostics Based on the Principles of Artificial Intelligence," *Proc. Intn'l. Test Conf.*, pp. 188-195, October, 1984.

[Yau 1987] C. W. Yau, "ILIAD: A Computer-Aided Diagnosis and Repair System," *Proc. Intn'l. Test Conf.*, pp. 890-898, September, 1987.

PROBLEMS

12.1 For the circuit shown in Figure 12.2(a), show that the faults *d s-a-*1 and *i s-a-*1 are indistinguishable.

12.2 For the circuit and the test set shown in Figure 12.2, determine the fault resolution of a fault dictionary built

a. without fault dropping;

b. with fault dropping after the first detection;

c. with fault dropping after two detections.

12.3 Construct a diagnosis tree for the circuit shown in Figure 12.2(a), assuming that the tests given in Figure 12.2(b) are applied in reverse order (i.e., t_5, ..., t_1).

12.4 For the circuit of Figure 12.2(a), try to generate a test that distinguishes between the faults (a) *g s-a-1* and *h s-a-*1 (b) *g s-a-*1 and *j s-a-*1.

12.5 For the circuit and the faulty responses shown in Figure 12.2, find the closest match for the response produced by the bridging fault (*a.f*).

12.6 For the circuit and the test shown in Figure 12.21, try to deduce the internal values corresponding to the response $D = 010$.