8

Adhoc Techniques

8.1 Introduction

In the first seven chapters we established the foundations of digital testing. In Part I, we discussed physical defects and their manifestation on the circuit and logic level. In Chapters 3 and 4 we placed testing within the design cycle and examined the various design representations that are appropriate for processing by different CAD tools. In Chapter 6, we found out that the process of test pattern generation is a NP-complete problem. In addition, we realized that the complexity of the test pattern is even more for sequential circuits. Because of the complexity, design approaches that facilitate testing have been developed and are very widely used at present. These approaches advocate introducing, into the circuit, constructs to make testing easier at the manufacturing level. The ease here extends to the processes, test pattern generation, and test application.

The design for test (DFT) discipline started with the insertion of test points into a design using an adhoc approach. Systematic techniques such as internal scan, built-in self-test (BIST) and boundary scan, are detailed in Chapters 9 to 11. They are all approaches to increase *testability* by embedding test structures in the circuit under test. There is no formal definition for testability. An interesting attempt was given as: "A digital IC is testable if test patterns can be generated, applied, and evaluated in such a way as to satisfy predefined levels of performance (e.g., detection, location, application) within a predefined cost budget and time scale" [Bennetts 1984]. One of the key words is "cost." It is probably the cost of testing that deters semiconductor manufacturers from doing as much testing as is really needed to ensure reliable products. Assessing the ease of testing used to be done for design described on the gate level. However, testability measures on higher levels of abstraction have been formulated. The benefit of attempting to assess testability of a circuit at such an abstract level is to correct for problems early in the design cycle.

In this chapter we first discuss why DFT structures are very particularly relevant in present technology. We discuss testability measures in more detail than we discussed in Chapter 1. In Section 8.4 we review ad hoc techniques that were followed before the systematic DFT methods. In addition, we revisit pseudoexhaustive

Copyright © 2000 by John Wiley & Sons

testing. Section 8.6 is devoted to regularly structured combinational designs that are tested either with a constant number of test patterns (C-testable) or with a test that depends on the number of repeated identical cells in the structure (scalable test).

8.2 Case for DFT

There are several reasons why it is becoming highly desirable, if not mandatory, to adopt DFT constructs in present VLSI circuits. These reasons pertain to the testing process itself and the nature of modern circuits as they affect testing cost.

8.2.1 Test Generation and Application

Unless we use exhaustive testing, test pattern generation is a complex problem. However, we realized in Chapter 2 that exhaustive testing is not realistic except in very few instances. Thus what was saved in test generation time is offset by the test application time. Random test pattern generation is also a viable feasible low-cost process; it yields long test sets but does not guarantee 100% SAF coverage. Fault-based test pattern generation, also called deterministic test pattern generation, has proven to be NP-complete [Ibarra 1975]. This means, as we mentioned in Chapter 4, that the problem of test pattern generation cannot be solved with a polynomial time algorithm. "To illustrate this," in [Fujiwara 1986]'s words, "suppose that we have three algorithms whose time complexities are *n,* n^3, and 2^n. Assuming that time complexity expresses execution time in microseconds, the *n,* n^3 and $\underline{2}^n$ algorithm can solve a problem of size 10 instantly in 0.00001, 0.001, and 0.001 seconds, respectively. To solve a problem of only size 60, the 2^n algorithm requires 366 centuries whereas the other two algorithms require only 0.00006 and 0.216 seconds, respectively." Some work was done to find circuits that can have a tractable solution [Fujiwara 1990, Chakradhar 1991]. However, since these works require more theoretical knowledge than we intend to present in this book, we leave it as a problem to explore further.

8.2.2 Characteristics of Present VLSI

Changes in technology have resulted in digital ICs that are characterized by:

1. Smaller devices that are capable of switching faster, thus allowing an increase in the operating frequency.

Copyright © 2000 by John Wiley & Sons

2. Higher device density that leads the designer to make more complex and larger circuits.

3. Lower power supply voltage: the supply voltage decreased from 5 V to 3.3 V and, very soon, 2 V. Thus the circuit is becoming less immune to noise.

4. Thinner and longer interconnect wires: The smaller wire cross sections have caused an increase in resistance, and capacitance. Other than capacitance to ground, there are fringing capacitances and mutual capacitance between interconnects on the same layer and on adjacent layers.

5. Design paradigm of a system on a chip (SOC). This approach to integrating several designs on the same chip involves the concept of reuse and brings new challenges to testing.

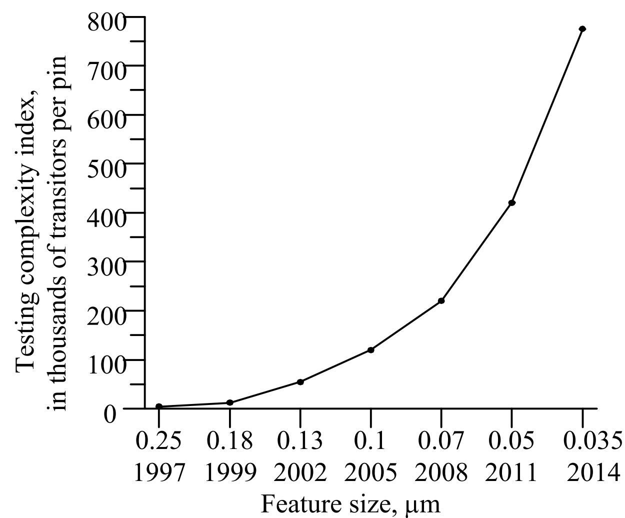


Figure 8.1 A Testability Index: Number of Transistors per Pin

The second and fifth characteristics of modern VLSI have resulted in designs that are too large and hence increase the complexity of test pattern generation. In addition, accessibility to the circuit is becoming more limited. The I/O pins recently have increased, but not at the same rate at which the size of the circuit has, to the extent that the pin-to-gate ratio is continuously decreasing. This is clear from Fig. 8.1. The increase in circuit size causes an increase in its processing time by various CAD tools. As stated in Chapter 5, new approaches to simulation, such as cycle-based simulation and static timing analysis, have been adopted to cope with complex circuits without delaying time-to-market. Another issue is that synthesis tools have not yet matured, and the handling of a large circuit requires more attention and skillful designers. To meet the challenge of testing such

Copyright © 2000 by John Wiley & Sons

circuits, more resources are needed – more powerful CAD tools, testers, and engineers trained in present technology. Next we revisit the testability measures that are still used as guides in test pattern generation.

8.3 Testability Analysis

An attempt to quantify testability, in the sense of fault detection, was proposed by [Goldstein 1979 and Grason 1979]. Two testability measures (TMs) were then defined: controllability and observability. *Controllability* reports the cost of placing a node in the circuit at a predetermined logic value. Placing a logic value on a primary input is "free." However, it is possible to assign a minimal cost, say 1, to any primary input. The cost increases as the node depth in the circuit increases. This cost will also depend on the type of gate, the logic value to be imposed on the line, and whether the circuit is combinational or sequential. For example, controlling the output of a multi-input AND gate to 1 requires the control of all its inputs to 1, whereas for an OR gate, it is sufficient to control only one of the inputs to 1. Thus controlling the output to 1 on an AND gate costs more than an OR gate. The most popular testability measures, SCOAP, list four controllability measures [Goldstein 1980]:

CC0	combinational 0-controllability
CC1	combinational 1-controllability
SC0	sequential 0-controllability
SC0	sequential 1-controllability

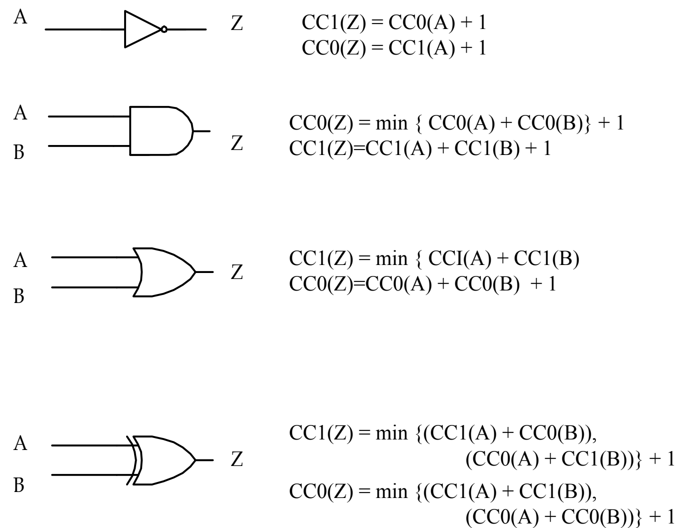


Figure 8.2 Controllability of Elementary Logic Functions

Copyright © 2000 by John Wiley & Sons

Here we concentrate on the combinational measures to illustrate the concepts and the use of the measures. The controllability measures of the outputs of elementary gates are given in Fig. 8.2, the gate within the circuit. If, however, they are primary inputs to the circuit, they are easily controllable and we can assume the cost to be 0 or 1. The cost at the output is increased by 1 to account for stepping one level over *A* and *B*. We can use these TMs to evaluate all nodes in a circuit based on such gates.

Example 8.1. As an example, let us calculate the TM of the nodes of circuit C1 shown in Fig. 8.3 as functions of the controllability of the input nodes. Although testability measure programs do not necessarily distinguish the branches from the stem, in this example, we will. To allow us to demonstrate how redundant logic that creates an undetectable fault will yield finite TM. Branches of the primary input, *A* and *B*, will be indexed with the number of the gate in which they fan-in. Thus we have branches, *A*1, *A* 2, *B*1, *B*2, *C*1, *C*3, and *H*4, and *H*5. Also, since the circuit has more than one primary output, we calculate observability measures for both outputs, *Y* and *Z*.

Table 8.1 Testability Measures of Circuit *C1* in Fig. 8.3.

Node	CC0	CC1	CO/*Y*	CO/*Z*	S1/*Y*	S1/*Z*	#P
A	1	1	6,8	7	7,9	8	2
A1	1	1	8	∝	9	∝	0
A2	1	1	6	7	7	8	2
B	1	1	6,8	7	7,9	8	2
B1	1	1	8	∝	9	∝	0
B2	1	1	6	7	7	8	2
C	1	1	8	5	9	6	3
C1	1	1	8	∝	9	∝	1
C3	1	1	∝	5	∝	6	2
G	2	2	∝	5	∝	7	2
H	3	2	4	4	7	7	1
H4	3	2	4	∝	7	∝	1
H5	3	2	∝	4	∝	7	1
F	2	4	5	∝	7	∝	1
Y	6	3	1	∝	7	∝	1
Z	5	3	∝	1	∝	6	3

To calculate the controllability, we need to proceed from the inputs to the outputs. Each logic level depends on the preceding levels. Here we assume that the controllability measures of the primary inputs, including branches, are all equal to 1. Then we calculate the numbers for the observability, this time starting from the most observable nodes, the primary outputs, and proceeding backward to the primary inputs. All results are summarized in Table 8.1. Initially, all TM values are set to infinity (∝). When the calculations are completed, the entries that

Copyright © 2000 by John Wiley & Sons

remain with $\propto$ indicate that the corresponding nodes are either uncontrollable or unobservable at the corresponding primary output.

We calculate the TM for the nodes on the second level, *F*, *G*, and *H*. The controllability of *H*4 and *H*5 are equal to that of their stem, *H*.

$$CC1(F) = CC1(A) + CC1(B) + CC1(C) + 1 = 4$$

$$CC0(F) = \min\{CC0(A), CC0(B), CC0(C)\} + 1 = 2$$

$$CC1(H) = \min\{CC0(A), CC0(B)\} + 1 = 2$$

$$CC0(H) = CC1(A) + CC1(B) + 1 = 3$$

$$CC1(G) = CC0(C) + 1 = 2$$

$$CC0(G) = CC1(C) + 1 = 2$$

These TM values are then used in calculating the primary output controllability:

$$CC1(Y) = \min\{CC1(F), CC1(H)\} + 1 = 3$$

$$CC0(Y) = CC0(F) + CC0(H) + 1 = 6$$

$$CC1(Z) = \min\{CC0(H), CC0(G)\} + 1 = 3$$

$$CC0(Z) = CC1(H) + CC1(G) + 1 = 5$$

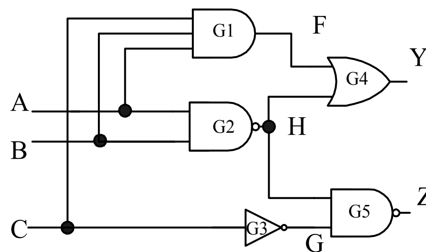


Figure 8.3 Circuit C1 to illustrate SCOAP Testability Measures

The *observability* of a node indicates the effort needed to observe the logic value on the node at a primary output. Again, we assume that the cost associated with observing a primary output is 1. To observe an internal

Copyright © 2000 by John Wiley & Sons

node, for example, node F of the circuit C1 in Fig. 8.3, it is necessary to control H to 0 in order to sensitize F to the primary output, Y. Thus the observability of F at Y is given by

$$CO_Y(F) = CO(Y) + CC0(H) + 1 = 5$$

Similarly, the observability of G is:

$$CO_Z(G) = CO(Z) + CC1(H) + 1 = 4$$

For H we can observe it as well as its branch $H4$ through Y. Also, we can observe it and its branch $H5$ through Z; consequently, we can find two observability numbers:

$$CO_Y(H) = CO(Y) + CC0(F) + 1 = 4$$

$$CO_Z(H) = CO(Z) + CC1(G) + 1 = 4$$

The observability measures of the primary inputs are calculated next. Since the primary inputs may be observable at either Y or Z, we have different values to evaluate. Let us first consider line C. Through Z, the observability of this node as well as its branch, $C3$, is

$$CO_Z(C) = CO_Z(G) + 1 = [CO(Z) + CC1(H) + 1] + 1 = 5$$

The observability of the same node and its branch, $C1$, through Y is given by

$$CO_Y(C) = CO_Y(F) + CC1(A) + CC1(B) + 1$$

$$= [CO(Y) + CC0(H) + 1] + CC1(A) + CC1(B) + 1 = 8$$

This result indicates that it is easier to observe C at node Z. In a similar manner, we can calculate the observability numbers for the other primary inputs, A and B. Actually, since the functions Y and Z are symmetrical in A and B, it is sufficient to calculate these numbers for one of these primary inputs. Notice also that the signals on these nodes may be sensitized to Y through gate G1 or G2. Sensitizing through G1 would yield the same observability number as for C through the same output, Y. Then $CO_{YF}(A) = CO_{YF}(B) = CO_Y(C) = 8$. Through G2 and observing at Y, the measure is

$$CO_{YH}(A) = CO_Y(H) + CC1(B) + 1 = 6$$

Observing through Z would yield

$$CO_Z(A) = CO_Z(H) + CC1(G) + 1 = 7$$

The TM numbers for all nodes in the circuit are summarized in Table 8.1. There are some entries in the table that still contain their initial value, infinity. This is an indication that the corresponding nodes are not observable at the

Copyright © 2000 by John Wiley & Sons

corresponding primary output. Although the SA1 faults on the branches of *A*1 and *B*1 feeding into gate G1 are not detectable because of redundancy, they have finite TMs that are comparable to other detectable faults, for example *C*.

The results indicate that the TMs vary with the depth of the circuit. The controllability numbers are higher for the primary outputs, while the observability numbers are higher at the primary inputs. Both measures are equally important since an easily controllable node is not necessarily observable. Similarly, the ease of observability is useless unless we can control the node. Several attempts have been made to combine the two numbers to correlate with the ease of test pattern generation. For example, one can add (or multiply) the controllability 0 to the observability and use the resulting number as a measure of the testability of stuck-at-1 faults. Such values are listed in the sixth and seventh columns of Table 8.1. To assess these measures, it is possible to generate an exhaustive test set for the circuit and determine the number of the patterns that detect each fault. These numbers are shown in the eigth column of the table.

The merit of testability measures in testing was assessed in several works [Agrawal 1982, Savir 1983, Mercer 1984]. SCOAP testability measures are a quick estimate of the degree of difficulty in generating test patterns without actually generating them. Calculating TM is not as time consuming as test pattern generation. The complexity is only $O(n)$, where n is the circuit size (number of lines). There are several variations of testability measures [Kovijanic 1979, Bennetts 1981, Ratiu 1982]. In addition, statistical testability measures have also been developed [Agrawal 1985]. They have been generalized to the design described on RTL and functional levels [Jamoussi 1991, Gu 1995]. As we learned in Chapter 6, these numbers have been used successfully as an aid in test pattern generation algorithms such as FAN and SOCRATES [Fujiwara 1982, Schultz 1988] and in determining test insertion points to facilitate controllability and observability for built-in self-test (BIST) [Lin 1993].

8.4 Initialization and Test Points

Ad hoc techniques tend to be heuristic rather than applicable to all circuits and low cost. On the other hand, structured techniques tend to solve general problems. Before DFT had been established formally, test engineers have always approached testing in a commonsense way. For example, they initialized a sequential circuit before testing it, included observation and control points to increase testability, and then divided the circuit into smaller

Copyright © 2000 by John Wiley & Sons

partitions to make each part more manageable. These various ad hoc techniques are the topic of this section. Partitioning for testability is discussed in Section 8.5.

8.4.1 Initialization

Initialization of a circuit is not always necessary for proper operation. However, it is necessary to place the circuit in a known position in order to test it [McCluskey 1986]. We learned in Chapter 3 how to generate an initializing or homing sequence. Such sequences may be too long and may be computationally costly to develop. In addition, not every sequential circuit may have a synchronizing or homing sequence.

8.4.2 Observation Points

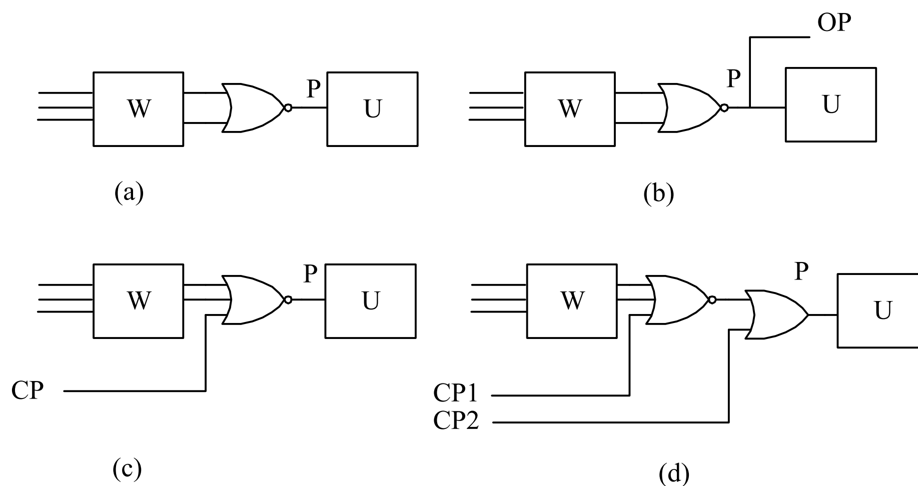


Figure 8.4 Test Point Insertion a) Original Circuit (b) Observation Point (c) Controlling P to 0, (d) Controlling P to 0 and 1

From the knowledge acquired on test pattern generation in previous chapters and the concepts of controllability and observability just discussed, it is clear that increasing the number of observation points facilitates the testing. This is illustrated with the circuit construct shown in Fig. 8.4, which consists of three main components: *W*, a NOR gate, and *U*. For example, if the node *P* is difficult to observe through *U*, then an observation point, OP, can be placed at this location as shown in Fig. 8.4*b*.

Copyright © 2000 by John Wiley & Sons

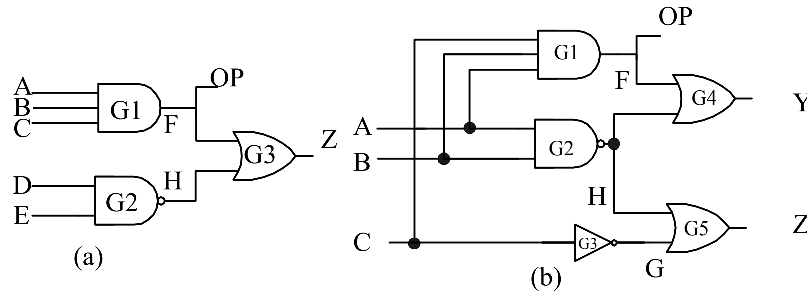


Figure 8.5 Examples to Illustrate Observation Points

Example 8.2. Let us consider circuit C2 in Fig. 8.5*a*, which is taken from [Breuer 1973]. This circuit requires five test patterns to detect all stuck-at faults. However, if we insert an observation point at *F*, only four patterns are necessary (or required) to detect the faults. If this shorter test set were used while observing only through the primary output, *Z*, the fault *B*/1 would not be observable. Thus adding this observation point improves testing by shortening the test length. However, these points cannot be added in a haphazard manner [Hayes 1975]. For example, an observation point at *H* would not have helped in reducing the test set.

Example 8.3. In circuit C1 of Fig. 8.3, the branches of *A* and *B* feeding into gate G1 have undetectable SA1 faults because of redundancy. These faults are observable at node *F*, but they are masked by *H*. Using F as an observation point, as illustrated in Fig. 8.5*b*, would make these faults detectable. Often, it is assumed that since a fault is redundant, there is no need to uncover it. However, this might cause unpredictable behavior by the circuit.

8.4.3 Control Points

As effective as inserting an observation point is the insertion of control points to ease forcing nodes to specific logic values. If, say, a zero is not obtainable from the NOR gate in Fig. 8.4*a*, it is possible to introduce the control point, CP1, as an additional input to the NOR gate and hold it high, as shown in Fig. 8.4*c*. If *P* is to be controlled also to 1, the structure in Fig. 8.4*d* would achieve this goal whenever CP2 is held high. It is worth noticing, however, that the insertion of a control point affects the controllability of the nodes influenced by the controlled point (all nodes of *U*). But the observability of the nodes whose signals go through the controlled point will also be affected. Thus CP1 and CP2 which ease the control of *P* in Fig. 8.4*d* affect the observability of the subcircuit, *W*. The observability of this subcircuit has more constraints – holding CP1 and CP2 low.

Copyright © 2000 by John Wiley & Sons

Test point insertion is not limited to combinational logic; it is actually needed more in sequential circuits [Gundhach 1990]. Sometimes, the reset of some storage devices is controlled by a complex logic and this makes initialization of the devices difficult. Using a control point can facilitate this task, as illustrated in Fig. 8.6*a*. Test points can be very helpful in enhancing the testability of long counters. As was pointed out in Section 1.7, to test a SA0 on the ripple-carry out of a 32-bit counter, 2^{32} patterns must be applied. Inserting control points as shown in Fig. 8.6*b* makes it possible to test the two parts, *M* and *N*, independently. Whenever possible, it is preferable to design modular counters. For example, a Mod 24 counter may be designed as two freely running Mod 3 and Mod 8 counters. In this fashion the test length can be reduced from 2^{24} to 2^{8}. Also, the ultimate of test point insertion in sequential logic is the introduction of scan path which is discussed in the Chapter 9 [Williams 1973].

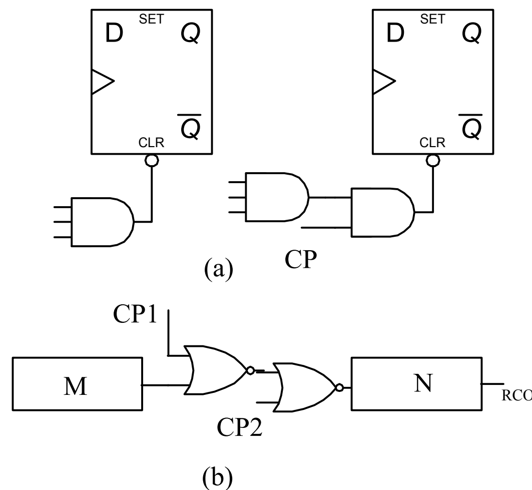


Figure 8.6 Initialization Through Test Point

Test point insertion for BIST application is also very important. We alluded to random pattern resistance (RPR) faults in Chapter 1. Those are the faults that are not easily detectable by PR patterns. To facilitate their testing, control points are inserted in the circuit and they are considered as primary inputs to the circuit. This topic is discussed in Chapters 11 and 14.

8.5 Partitioning for Testability

The testability measures of a circuit are functions of its depth. Smaller circuits are easier to test. If a large circuit is broken down into smaller subcircuits, it is possible to include some controllability and observability points, as

Copyright © 2000 by John Wiley & Sons

illustrated earlier. For this, the circuit needs to be modified by inserting some hardware to facilitate insertion of these points.

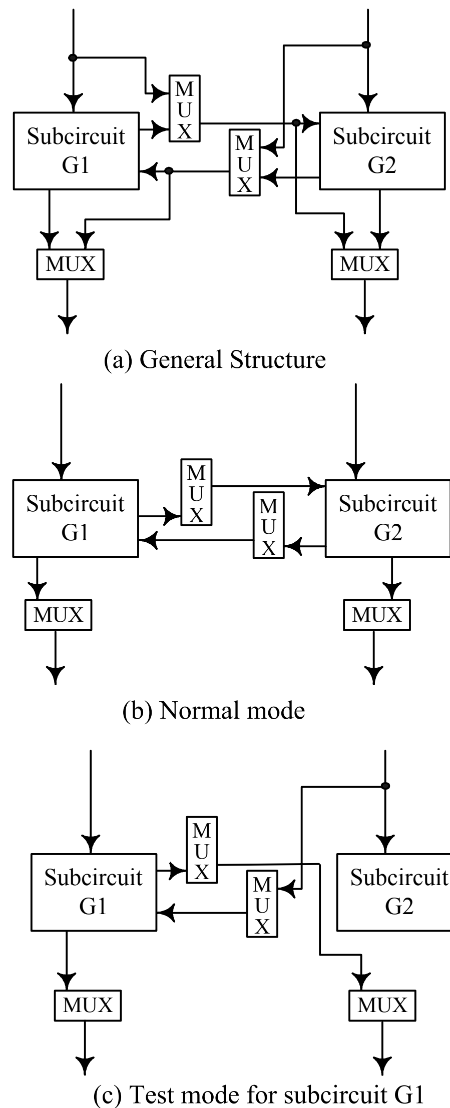


Figure 8.7 Partitioning Scheme Using Multiplexers

A general hardware partitioning technique using multiplexers is illustrated in Fig. 8.7. The subcircuits, G1 and G2, are not two independent partitions. They are two segments that have common circuitry. Using the multiplexers, it is possible to isolate each segment and improve its controllability and observability. The added multiplexers are transparent during normal operation as illustrated in Fig. 8.7*b*. To test one of the segments, say G1, the circuit is configured as shown in Fig. 8.7*c*. Some input to G2 will be used to supply the test data directly to G1. Therefore, the lines from G2 feeding into G1 become test points that are easily controllable. Similarly, the lines from G1

Copyright © 2000 by John Wiley & Sons

feeding into G2 are diverted through the multiplexer to the outputs of the circuit. They become easily observable at these outputs. We apply the concept to a concrete example, circuit, C3, in Fig 8.8, which has been divided into three subsections, α, β, and γ.

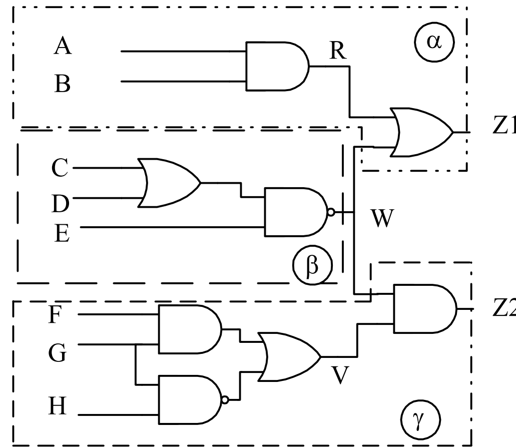


Figure 8.8 Circuit C3

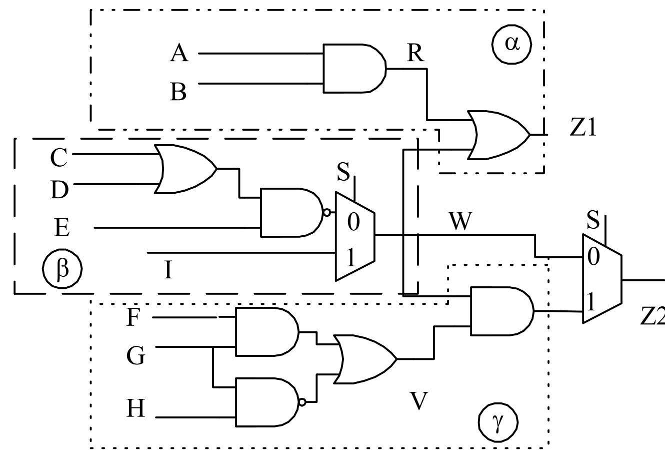


Figure 8.9 Hardware Segmentation

For example, to test the β partition of the same circuit, it is possible to insert two 2-to-1 MUXs as shown in Fig. 8.9 and to observe this partition directly ($S = 0$) or to synthesize the two other segments through their respective outputs ($S = 1$). The appropriate value is applied to I for testing each of these segments.

It is also possible to do this segmentation of the circuit by sensitization of the test data, and it is known as *sensitized partitioning*. For this we will revisit an approach that was proposed as an alternative to exhaustive testing [McCluskey 1981]. An N-input circuit is partitioned into M segments each of which has K_i primary inputs such that

Copyright © 2000 by John Wiley & Sons

$K_i < N$, for all values of i. Testing all partitions exhaustively would require a test set of length: $\sum_{i=1}^{i=M} 2^{K_i} \leq 2^N$.

Thus by testing each partition, rather than the whole circuit exhaustively, the complexity of the test is reduced while guaranteeing 100% fault coverage. We illustrate this approach with the circuit, C3, in Fig. 8.8.

Example 8.4. The circuit has eight primary inputs. The length of an exhaustive test is 256 patterns. Instead, the circuit is partitioned into three partitions: α, β, and γ. It is possible to test the first partition exhaustively using eight patterns and observing through the primary output, Z_1. The second partition, β, also depends on three of the inputs. It can be sensitized through Z_1 or Z_2. Partition γ can be tested exhaustively with 16 patterns since it depends on four inputs. The total length of the test is the sum of the three individual tests, 32 patterns instead of 256 patterns. To sensitize the test for partition α, we need to keep $W = 0$. But we have to have $W = 1$ to observe the response to the test for γ. For partition β, any time a pattern for α that makes $R = 0$ or a pattern for γ that makes $V = 0$, the response for β will be observed through Z_1 or Z_2, respectively.

Table 8.2 Pseudo-exhaustive Testing for Circuit in Fig. 8.6

	A	*B*	*R*	*C*	*D*	*E*	*W*	*F*	*GA*	*H*	*V*
α	0	0	0				0				
	0	1	0				0				
	1	0	0				0				
	1	1	1				0				
	0	0	0				1				
	0	1	0				1				
	1	0	0				1				
	1	1	1				1				
β			0	0	0	0	1				
			1	0	0	1	1				
			0	0	1	0	1				
			1	0	1	1	0				
				1	0	0	1				
				1	0	1	0				
				1	1	0	1				
				1	1	1	0				
γ							0	0	0	0	1
							0	0	0	1	1
							0	0	1	0	1
							0	0	1	1	0
							0	1	0	0	1
							0	1	0	1	1
							0	1	1	0	1
							0	1	1	1	1
							1	0	0	0	1
							1	0	0	1	1
							1	0	1	0	1
							1	0	1	1	0
							1	1	0	0	1
							1	1	0	1	1
							1	1	1	0	1
							1	1	1	1	1

Copyright © 2000 by John Wiley & Sons

It is possible to minimize the test set by testing the partitions concurrently. For example, patterns for α that result in $R = 0$ can be combined with patterns for β that make $W = 0$. Similarly, patterns for β that make $W = 1$ can be applied concurrently with those patterns for γ that make $V = 1$. Considering these facts, we can compact the test from 32 to 16 patterns. All patterns are listed in Table 8.2. It might be possible to compact the test further by taking advantage of the *don't care*, *x*, signals.

Another special case of exhaustive testing is known as *verification testing* [McCluskey 1984]. It is applicable to circuits where each primary output is a function of only a subset of primary inputs. Thus the circuit is divided into segments with overlapping components. We illustrate the concepts with three different circuits.

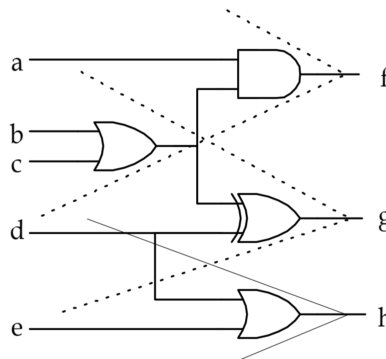


Figure 8.10 Circuit C4

Table 8.3 Verification Testing for Circuit C4.

(a) Dependency Matrix

	a	b	c	d	e
F	1	1	1	0	0
G	0	1	1	1	0
H	0	0	0	1	1

(b) Two Different Input Partitioning

a	d	b	c	e
1	0	1	1	0
0	1	1	1	0
0	1	0	0	1

a	e	b	c	d
1	0	1	1	0
0	0	1	1	1
0	1	0	0	1

Example 8.5. Consider first the simple example, circuit C4, shown in Fig. 8.10. This circuit has five inputs ($n = 5$) and three outputs, $f\,(a,b,c) = a\,(b + c)$, $g\,(b,c,d) = d \oplus (b + c)$, and $h\,(e,d) = e + d$. Each output is a function of a subset of the primary inputs set. The three outputs have 3, 3, and 2 inputs, respectively. The largest number of input will be denoted by w and here $w = 3$. An exhaustive test set would consist of 32 patterns. If we exhaustively test one output at a time, the combined test will consist of 8, 8, and 4 patterns, a total of 20 patterns. Although this is

Copyright © 2000 by John Wiley & Sons

shorter than the exhaustive test, it can still be reduced. It has been proven that if two inputs never appear in the same output function, they can have the same test signal applied to both [McCluskey 1984]. This results in reducing the number of required test signals, a scheme called *maximum test concurrency* (MTC). We follow a technique formulated by McCluskey. For circuit C4 we develop the *dependency matrix* shown in Table 8.3. This matrix consists of *m* rows and *n* columns, where *m* is the number of primary outputs and *n* is the number of primary inputs of the circuits. For this example, $m = 3$ and $n = 5$.

Table 8.4. Test Set for Circuit C4.

a	b	c	d	e
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

The first column lists the various functions. In this matrix, for each function, a 1 entry in any of the subsequent columns indicates the dependency of this function on the corresponding variable, otherwise, the entry in the matrix is 0. Next, we partition the primary inputs so that any partition is such that two or more of the inputs do not affect the same output. In other words, no partition would have more than one 1 in a row. For example, we combined *a* and *d* to form one partition since *F* depends on *a* and not *d*, and vice versa for *G* and *H*. Similarly, we combined *c* (or *b*) and *e*. This partitioning is not unique since we could have grouped *a* and *e* instead. Actually, there are three different ways of partitioning. In general, such partitioning is a NP-complete problem. The information provided by the partitioned matrix determines the total number of test patterns, 2^K, where *K* is the maximum number of 1's in a row. This is called the *maximum dependency*. Also, variables in the same partition can assume the same values in the test. For this example, $K = 3$ and the total number of patterns is only 2^3. This implies that only eight patterns are needed to test the entire circuit pseudoexhaustively! This is a 75% saving over the exhaustive test. The test is shown in Table 8.4. Notice that values of *a* and *d* are identical since the two inputs are in the same partition. The same istrue for *c* and *e*.

Copyright © 2000 by John Wiley & Sons

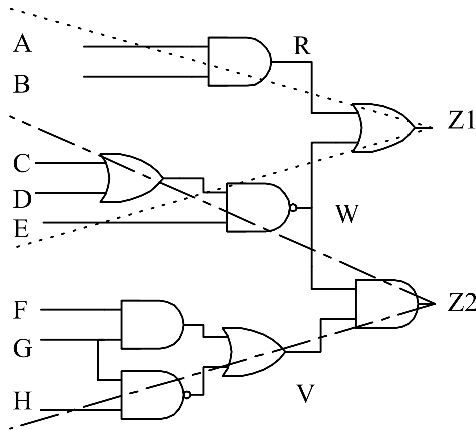


Figure 8.11 Verification Testing on Circuit C3

Example 8.6. Circuit C3 has eight primary inputs and two primary outputs, Z_1 and Z_2. Each output is dependent on a subset of the primary inputs, 5 and 6, respectively, as can be determined from Fig. 8.11.

Exhaustive test sets for the partitions are of lengths 32 and 64; the total length of the test for the circuit is then 96, compared to the 256 patterns for an exhaustive test for the entire circuit. However, here again we can compact the test set because some of the signals on the shared inputs may be common to patterns for both partitions and therefore these patterns may be applied simultaneously.

Table 8.5 Verification Testing for C3

(a) Dependency Matrix

	A	B	C	D	E	F	G	H
Z1	1	1	1	1	1	0	0	0
Z2	0	0	1	1	1	1	1	1

(b) Input Partitioning

A	F	B	G	C	D	E	H
1	0	1	0	1	1	1	0
0	1	0	1	1	1	1	1

To partition the matrix we combined A and F to form one partition, as shown in Table 8.5. Similarly, we combined B and G. This partitioning is not unique since we could have grouped A and H instead. Actually, there are six different ways of partitioning the matrix, since A and B can be combined with any of the inputs G, H, and F. The maximum dependency is $K = 6$. This implies that the number of test patterns needed is only $2^6 = 64$ instead of 256.

Copyright © 2000 by John Wiley & Sons

8.6 Easily Testable Circuits

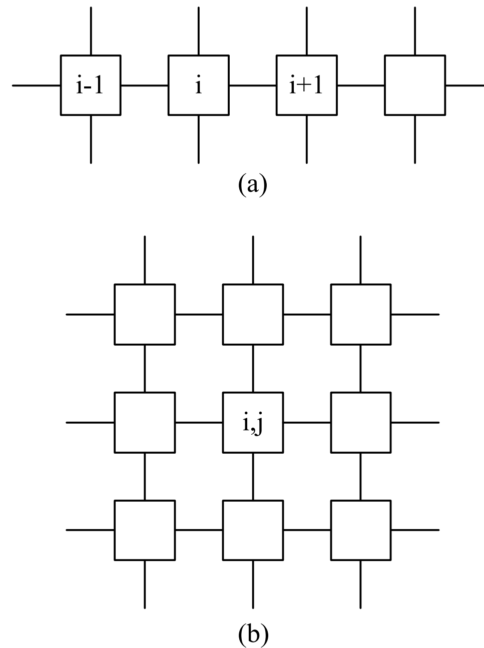


Figure 8.12 Regular Structures
(a) One-Dimension, (b) Two-Dimension

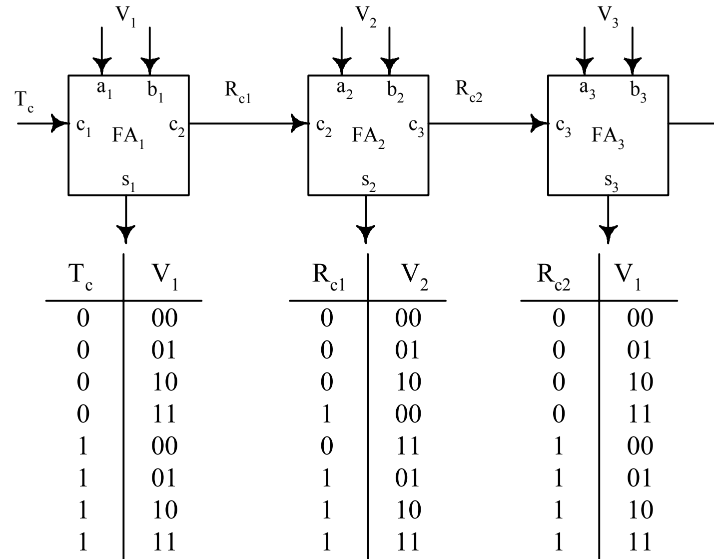
Regularly structured circuits consist of an array of identical cells. They may be arranged in one- or two-dimensional arrays as illustrated in Fig. 8.12. Examples of such circuits are RAMs, FPGAs, array multipliers or circuits that are deliberately designed in a modular fashion to facilitate layout on a chip. Although these circuits may be combinational or sequential, it is on the first category that we focus our attention in this section.

Combinational regular structures are usually referred to as iterative logic arrays (ILAs). Often, n-bit comparators are organized in a one-dimension array and each compares the corresponding bit from two numbers. Another example may be a parity tree where each cell is a two-input XOR. Array multipliers consist of some type of full adders that are usually arranged in a two-dimensional array. Some of these structures are easily testable with a constant number of test patterns that is independent of the size of the circuit – the number of identical cells in the circuit [Friedman 1973]. Friedman called these ILAs C-testable. We examine two examples: first, a parity tree, and second, an array multiplier that is redesigned to make it C-testable. Other C-testable structures have a scalable test length. That is, the test length grows proportionately to size. In this section we examine the two types.

Copyright © 2000 by John Wiley & Sons

Testing array structure will be encountered in Chapters 11 and 13 when we study the testing of memories and FPGAs, respectively. C-testability and scalable testing are applicable to some of these structures.

8.6.1 C-Testability



T_c	V_1
0	00
0	01
0	10
0	11
1	00
1	01
1	10
1	11

R_{c1}	V_2
0	00
0	01
0	10
1	00
0	11
1	01
1	10
1	11

R_{c2}	V_1
0	00
0	01
0	10
0	11
1	00
1	01
1	10
1	11

Figure 8.13 C-testability Developing the Test Set

Consider a one-dimensional array that is organized as shown in Fig. 8.13. Let T_c be an exhaustive test set for any cell *j* in the array, and let R_c be the response to this test. This test will detect all faults that keep the circuit combinational [Friedman 1973]. That is the faults in cell *j* are sensitized to its outputs. If $R_c \subseteq T_c$, R_c will be an exhaustive test for cell $j + 1$. Notice that the order of the patterns in R_c is not necessarily the same as T_c. The test set for any cell *j* is the concatenation of T_c and V_j. If again $R_c \subseteq T_c$, then R_c concatenated with vector V_{j+1} form the test for cell $j + 1$. Suppose that the cells are full adders, then an eight-pattern exhaustive test applied is sufficient to test the array as indicated in the figure. We will show two more example circuits.

Example 8.7. An *N*-input parity tree is easily testable with four patterns [Bossen 1970]. A parity tree is a fan-out free circuit. A test set for such a circuit consists of those patterns detecting faults on the primary inputs. The four patterns for an exhaustive test for each two-input XOR gate is {00,01,10,11}.

Copyright © 2000 by John Wiley & Sons

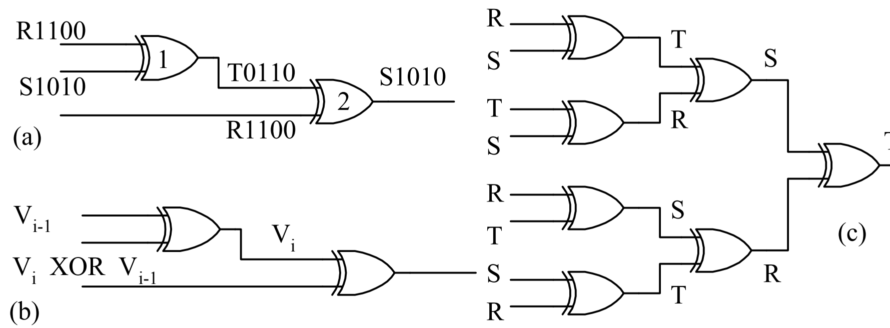


Figure 8.14 A Parity Tree
a) A Labeling Scheme, b) Labeling Process,
c) A Labeled 8-input Tree

Consider gate 1 of the circuit shown in Fig. 8.14*a*. We apply an exhaustive test on it and we call the two vectors at the primary pins *R* and *S*. The result of the application of this test at the output of the gate is vector *T*. To observe the result of the test at the primary output, the output of gate 2, we can apply any signal, 1 or 0, at the other input of this gate. However, we apply the sequence *R*, which together with *T*, form an exhaustive test for gate 2. The output sequence is *S*. The three vectors have the following property: $V_i \oplus V_j = V_k$, where V_i, V_j, and V_k are selected from the set of vectors *R*, *S*, and *T*, in any order, provided that they are three different vectors. This property allows us to generalize the application of the test on any *N*-input tree. The labeling procedure illustrated in Fig. 8.14*b* of an eight-input tree is shown in Fig. 8.14*c*. You might be interested to verify the fault coverage of this test using a fault simulator. The test set for the eight-input parity tree is then:

	Primary inputs
Pattern #	**RS TS RT SR**
1	1 1 0 1 1 0 1 1
2	1 0 1 1 0 1 0 1
3	0 1 1 0 1 1 1 0
4	0 0 0 0 0 0 0 0

This approach was used successfully to determine the testability of parity checkers [Mourad 1989].

Copyright © 2000 by John Wiley & Sons

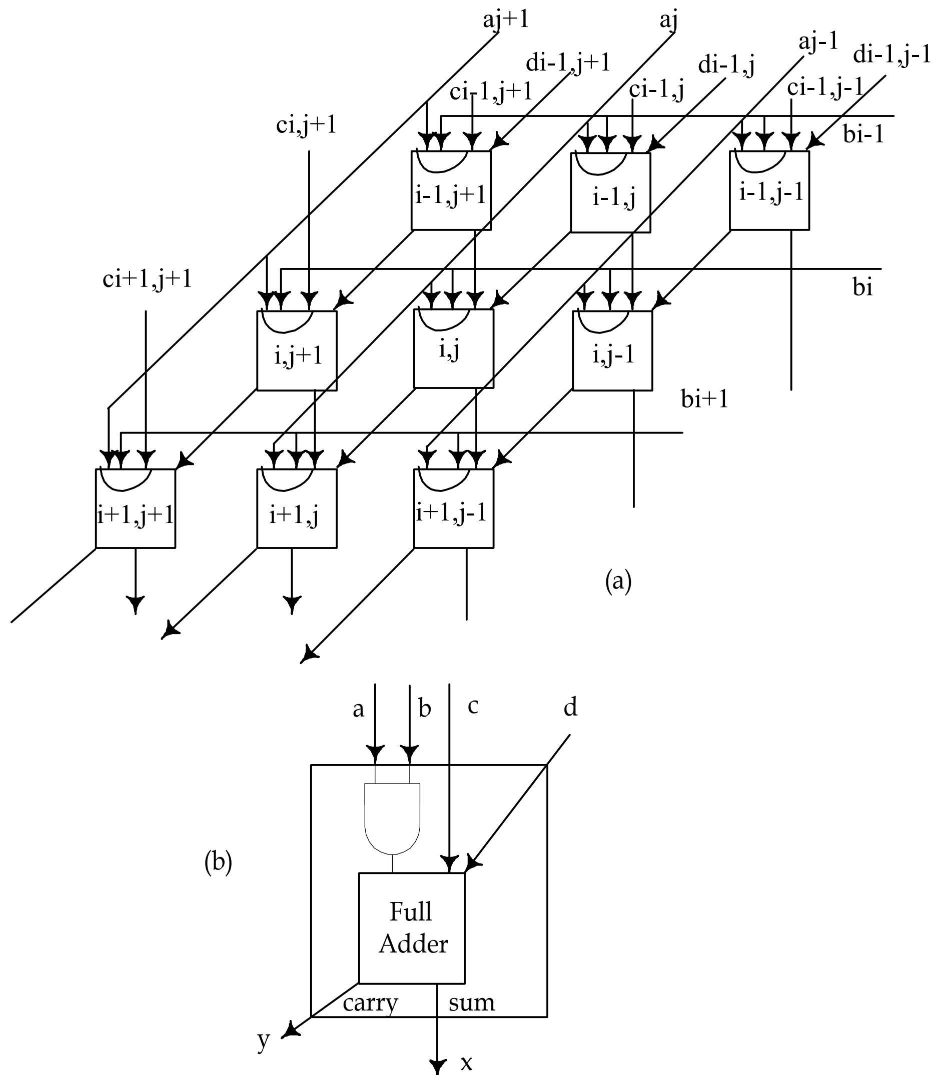


Figure 8.15 Array Multiplier (a) General Structure
(b) Carry & Save Basic Cell [Shen 1984]

Example 8.8. Some circuits need just a slight modification to make them C-testable. An interesting example is the modification of the cell of an array multiplier that made it testable with only 16 patterns [Shen 1984]. Figure 8.15 shows a carry-save array multiplier and the basic unit, a full adder cell. This cell has four inputs, a, b, c, and d, and two outputs, x and y:

$$x = (ab) \oplus c \oplus d$$

$$y = (ab)c + (ab)d + cd$$

As in the case of the parity tree, the aim is to use an exhaustive test for every cell and test all cells simultaneously.

Copyright © 2000 by John Wiley & Sons

Let {*a,b,c,d*} be a test pattern applied to any of the cells. To be detected by the test, the fault has to be observable on either *x, y*, or both. A fault will cause any or both of the two outputs to be inverted. We notice also that the test patterns (*a,b,c′,d*) and (*a,b,c,d′*) yield the same results since *x* is a party of (*ab*), *c,* and *d*. Thus a fault appearing at *c* or *d* will be propagated to the output of the cell.

Table 8.6 The Truth Tables for the Original and the Modified Array Multiplier Cell [Shen 1984]

	cd			
ab	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

	cd			
ab	00	01	11	10
00	0	1*	1	0
01	0	1*	1	0
11	0	1	1	1
10	0	0	1	0

Next, we need the values on *x* and *y* to replicate a test pattern of the cell. That is, if {*abcd*} is a pattern applied on cell (*j,k*) and $\{a_1b_1c_1d_1\}$ is a pattern applied on cell (*j, k+1*), then the $\{ab_2x_1y\}$ is a test for cell (*j + 1,k + 1*). This is possible only if the main carry-save cell is slightly modified as indicated by the truth tables in Table 8.6. The proposed change, which is indicated by an asterisk (*), does not alter the functionality of the cell as a multiplier since, during normal operation, *c* and *d* on the first row of the array are both 0. Consequently, the patterns {*abcd*} = {0001} and {0101} can never appear on the input of any cell in the array. The array may thus be tested by 16 patterns regardless of its size. For details on constructing the test, you are referred to Shen's work [Shen 1984].

Table 8.7 An Array Comparator

(a) Definition

gs	GS
	ab 00 01 11 10
00	00 01 00 10
01	01 01 01 10
11	Xx xx xx xx
10	10 01 10 10

(b) Test Set (Tc)

	gs ab	GS
1	11 00	11
2	00 00	00
3	10 11	10
4	01 11	01
5	10 01	01
6	01 10	10

(c) Augmented Test. (Ta)

	Cell I	Cell i+1
	gs ab GS	gs ab GS
1	11 00 11	11 00 11
2	00 00 00	00 00 00
3	10 11 10	10 10 10
4	01 11 01	01 01 01
5	10 01 01	01 10 10
6	01 10 10	10 01 01
7	10 01 01	01 01 01
8	01 10 10	10 10 10

Copyright © 2000 by John Wiley & Sons

8.6.2 Scalable Testing

For arrays that are not C-testable the response, R_c, of cell *j* to the test, T_c, is such *that* $T_c \cap R_c \subset T_c$. For the response from cell *j* to be a test for cell *j* + 1, we need to apply on cell *j* an augmented test set, $T_a = T_c + T_p$, such *that* $T_c \subset R_a$, where R_a is the response of the cell to T_a. We can then write $R_a = T_c + T_q$. This is the test for *j* + 1 and hence it has to be augmented by T_p. In this fashion, when we reach the *n*th cell in the array, the test set will be $T_c + p\ (n - 1)$, where *p* is the length of the augmented patterns.

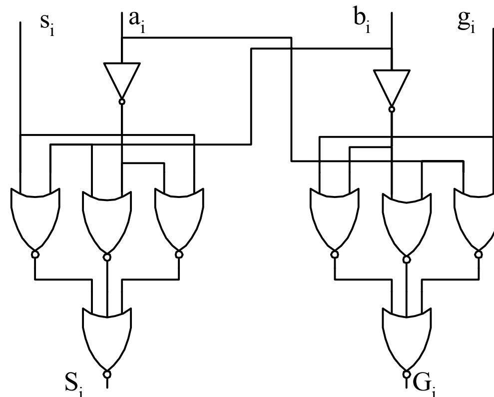


Figure 8.16 A 2-bit Magnitude Comparator

A scalable test can be obtained for the array comparator whose cell is shown in Fig. 8.16. The outputs, *S* and *G*, of this cell are defined in Table 8.7*a*. The minimal test set and the augmented set are shown in Table 8.7*b* and *c*. For any number of bits *N*, the test length *is* $K + p(N + 1)$, where *K* is the minimal test for one unit (K = 6) and *p* the number of added patterns (p = 2). Test scalability is becoming more relevant for core-based testing [Al-Assad 1998] and will be revisited in a later chapter.

References

Agrawal, V. D. and R. M. Mercer (1982), Testability measures what do they tell us, *Proc. IEEE Semiconductor Test Conference,* pp. 391–396.

Agrawal, V. D. and S. C. Seth (1985), Probabilistic testability, *Proc. International Conference on Comput. Des.*, pp. 562–565.

Al-Assad, J. P. Hayes, and B. T. Murray (1998), Scalable test generators for high-speed datapath circuits, J. of Elec. Testing, Vol. 12, No. ½, pp 111 – 126, *Kluwer Academic*, Norwell, MA.

Bennetts, R. G. (1984), *Design of Testable Logic Circuits,* Addison-Wesley, Reading, MA, p. 164.

Copyright © 2000 by John Wiley & Sons

Bennetts, R. G., C. M. Maunder, and G. D. Robinson (1981), CAMELOT: A Computer-Aided Measure for Logic Testability, *IEEE Proc.*, Vol.128, Part E, No.5, pp.177–189.

Bossen, D. C., D. L. Ostapko, and A. M. Patel (1970), Optimum test patterns for parity network, *Proc. AFIPS Fall 1970 Joint Computer Conference*, Vol. 37, pp. 63–83.

Bozorgui-Nesbat, S., and E. J. McCluskey (1980), Structured design for testability to eliminate test pattern generation. *Proc. International Symp. Fault-Tolerant Computing*, pp. 158–163.

Breuer M. A., and A. D. Friedman (1976), *Diagnosis and Reliable Design of Digital Systems,* Computer Science Press, New York.

Chakradhar, S.T. et al. (1991), *Neural Models and Algorithms for Digital Testing*, Kluwer Academic, Norwell, MA.

Friedman, A. D. (1973), Easily testable iterative systems, *IEEE Trans. Comput,* Vol. C-22, No. 12, pp. 1061–1064.

Fujiwara, H. and S. Toida (1982), The complexity of fault detection problem for combination circuits, *IEEE Trans. Comput*. Vol. C-31, No. 6, pp. 555–560.

Fujiwara, H. (1986), *Logic Testing and Design for Testability*, MIT Press, Cambridge, MA.

Fujiwara, H. (1990), Computational complexity of controllability/observability problems for combination circuits, *IEEE Trans. Comput*. Vol. C-39, No. 7, pp. 762–767.

Goldstein, L. H. (1979), Controllabilty/observability analysis of digital circuits, *IEEE Trans. Circuits and Sys.*, Vol. CAS-26, No. 9, pp. 683–693

Goldstein, L. H., and E. L. Thigpen (1990), SCOAP Santdia controllability / Observability analysis program, *Proc. Des. Automation Conference.* , pp. 190–194.

Grason, J. (1979), TEMAS, a testability measure program, *Proc. Des. Automation Conf*erence, pp. 156–161.

Gu, X., K Kuchcinski, and Z. Peng (1995), An efficient and economic partitioning approach for testability, *Proc. IEEE International Test Conference*., pp. 403 – 412.

Gundhach H. H. S., and K. D. Muller-Glaser (1990), On automatic test-point Insertion in sequential circuits, *Proc. International Test Conference*, Philadelphia, PA, Oct., pp. 1072–1079.

Hayes J. P. (1974), On modifying logic networks to improve their diagnosability, *IEEE Trans. Comput.* Vol. C-22, No. 1, pp. 56–63.

Hayes J. P., and A. D. Friedman (1975), Test points placement to simplify fault detection, *IEEE Trans. Comput*. Vol. C-23, No. 7, pp. 727–735.

Ibarra, O. H. and S. K. Sahni (1975), Polynomially Complete Fault Detection Problems, *IEEE Trans. Comput*., Vol. C-24, pp. 242–249.

Jamoussi, M, B. Kaminska, and D. Mukhekar (1991), A new variable testability measure: a concept for data-flow testability evalaution, *Proc.IEEE International Test Conference* , pp. 239–243.

Kovijanic, P. G. (1979), Computer aided testability analysis, *Proc. IEEE Automatic Test Conference*, pp. 292–294.

Lin, C. J., Y. Zorian, and S. Bhawmik (1993), PSBIST: a partial-scan based BIST scheme, *Proc. IEEE International Test Conference*, pp. 507–516.

Copyright © 2000 by John Wiley & Sons

McCluskey E. J. (1984), Verification Testing – A pseudo-exhaustive test technique, *IEEE Trans. Comput*., Vol. C-33, No. 6, pp 541–546.

McCluskey E.J., (1986), *Principles of Logic Design with Emphasis on Semicustom Circuits*, Prentice Hall, Upper Saddle River, NJ.

Mercer, M. R. and B. Underwood (1984), Correlating testability with fault detection,

Mourad, S. and E. J. McCluskey (1989), Testability of parity checkers, *IEEE Trans. Ind. Electron.*, Vol. IE- 36, No. 2, pp. 254–260.

Ratiu, I. M., A. Sangiovanni-Vincentelli, and D.O. Peterson (1982), VICTOR: A Fast VLSI Testability Analysis Program, *Proc. IEEE International. Test Conference*, pp.397–401.

Savir, J. (1983), Good controllability and observability do not guarantee good testability, *IEEE Trans.Comput*., Vol. C-32, pp 1198–1200.

Schultz, M. H. et al. (1988), SOCRATES: A Highly Efficient Automatic Test Pattern Generation System, *IEEE Trans. Comput.-Aided Des.*, Vol. CAD-8, No.1, pp. 126–137.

Shen J. P. and F. J. Ferguson F. J. (1984), The design of easily testable VLSI array multipliers, *IEEE Trans. Comput.*, Vol. C-33, No. 6, pp. 554–560.

Sridhar, T. and J. P. Hayes (1981) Design of easiliy testable bit-sliced systems, *IEEE Trans. Comput.*, Vol. C-30, No. 11, pp 842–854

Williams M. J.Y., and Angell B. (1973), Enhancing testability of large scale integrated circuit via test points and additional logic, *IEEE Trans. Comput.*, Vol. C-22, No. 1, pp. 46-60.

Wulderlich, H.-J. and S. Hellebrand (1989), The pseudo-exhaustive test of sequenctial circuits, *Proc. IEEE International Conference,* pp. 19–27.

8.8 Problems

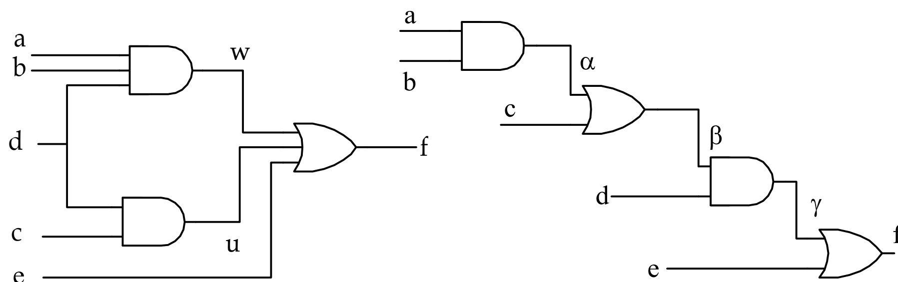


Fig. 8.P1

Copyright © 2000 by John Wiley & Sons

8.1. The two circuits in Fig. P8.1 are functionally equivalent. Compare their SCOAP testability measures.

8.2. Verify the information presented in Example 8.2 about circuit C2 in Fig. 8.5*b* by developing a SA test for the circuit using (a) only Z as the output, (b) F and Z as outputs, and (c) H and Z as outputs.

8.3. Consider the logic block of ACT1 and generate a pseudoexhaustive test with a minimal number of patterns. For this circuit, see Fig. 13.2*c*.

8.4. Use the labeling technique recommended in Section 8.6 and develop a four-pattern test set for a 16-input parity tree and verify the fault coverage using a fault simulator.

8.5. Develop a maximal concurrency test for (a) the parity generator network, part SN74630 and (b) the dual multiplexer network, part SN74153.

Copyright © 2000 by John Wiley & Sons