11. BUILT-IN SELF-TEST

About This Chapter

In previous chapters we have discussed algorithmic methods for test generation and techniques for design for testability (DFT). These methods are primarily used when external testing is employed. Built-in self-test (BIST) is a design technique in which parts of a circuit are used to test the circuit itself. The first part of this chapter covers the basic concepts associated with BIST. We then focus on the problem of built-in generation of test patterns. Various ways of partitioning a circuit for self-testing are described, as are ways of generating test patterns. Test-pattern generation techniques discussed include exhaustive testing, pseudorandom testing, and pseudoexhaustive testing; the latter includes the concepts of verification and segmentation testing.

Generic BIST architectures are described, including the major ways of characterizing such architectures in terms of centralized versus distributed BIST hardware and internal versus external BIST hardware. Next many specific BIST architectures are presented.

Finally several advanced BIST techniques are discussed, including the identification of a minimal number of test sessions, the control of BIST structures, and the notion of partial-intrusion BIST designs.

11.1 Introduction to BIST Concepts

Built-in self-test is the capability of a circuit (chip, board, or system) to test itself. BIST represents a merger of the concepts of *built-in test* (BIT) and *self-test*, and has come to be synonymous with these terms. The related term *built-in-test equipment* (BITE) refers to the hardware and/or software incorporated into a unit to provide DFT or BIST capability.

BIST techniques can be classified into two categories, namely on-line BIST, which includes concurrent and nonconcurrent techniques, and off-line BIST, which includes functional and structural approaches (see Figure 11.1). These terms were mentioned in Chapter 1 but will be briefly discussed here for ease of reference.

In *on-line* BIST, testing occurs during normal functional operating conditions; i.e., the circuit under test (CUT) is not placed into a test mode where normal functional operation is locked out. *Concurrent on-line* BIST is a form of testing that occurs simultaneously with normal functional operation. This form of testing is usually accomplished using coding techniques or duplication and comparison; these techniques will be described in more detail in Chapter 13. In *nonconcurrent on-line* BIST, testing is carried out while a system is in an idle state. This is often accomplished by executing diagnostic software routines (macrocode) or diagnostic firmware routines (microcode). The test process can be interrupted at any time so that normal operation can resume.

Off-line BIST deals with testing a system when it is not carrying out its normal functions. Systems, boards, and chips can be tested in this mode. This form of testing is also applicable at the manufacturing, field, depot, and operational levels. Often Off-line testing is carried out using on-chip or on-board test-pattern generators (TPGs) and output response analyzers (ORAs) or microdiagnostic routines. Off-line testing does not detect

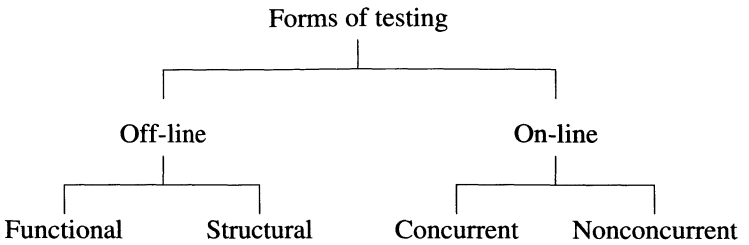


Figure 11.1 Forms of testing

errors in real time, i.e., when they first occur, as is possible with many on-line concurrent BIST techniques.

Functional off-line BIST deals with the execution of a test based on a functional description of the CUT and often employs a functional, or high-level, fault model. Normally such a test is implemented as diagnostic software or firmware.

Structural off-line BIST deals with the execution of a test based on the structure of the CUT. An explicit structural fault model may be used. Fault coverage is based on detecting structural faults. Usually tests are generated and responses are compressed using some form of an LFSR. Figure 11.2 lists several types of test structures used in BIST circuits. Two common TPG circuits exist. A *pseudorandom pattern generator* (PRPG) is a multioutput device normally implemented using an LFSR, while a *shift register pattern generator* (SRPG) is a single-output autonomous LFSR. For simplicity the reader can consider a PRPG to represent a "parallel random-pattern generator," and a SRPG to be a "serial random-pattern generator." Two common ORA circuits also exist. One is a *multiple-input signature register* (MISR), the other a *single-input signature register* (SISR). Both are implemented using an LFSR.

This chapter deals primarily with structural off-line BIST. Thus LFSRs will be used extensively. The reader is referred to Chapter 10 for a review of the theory related to LFSRs. Before discussing the structure of off-line BIST circuitry, some basic concepts will be reviewed.

11.1.1 Hardcore

Some parts of a circuit must be operational to execute a self-test. This circuitry is referred to as the *hardcore*. At a minimum the hardcore usually includes power, ground, and clock distribution circuitry. The hardcore is usually difficult to test explicitly. If faulty, the self-test normally fails. Thus detection is often easy to achieve, but little if any diagnostic capability exists. If a circuit fails during self-test, the problem may be in the hardcore rather than in the hardware presumably being tested. The hardcore is normally tested by external test equipment or is designed to be self-testable by using various forms of redundancy, such as duplication or self-checking checkers (see Chapter 13). Normally a designer attempts to minimize the complexity of the hardcore.

BILBO — built-in logic block observer (register)

LFSR — linear feedback shift register

MISR — multiple-input signature register

ORA — (generic) output response analyzer

PRPG — pseudorandom pattern generator, often referred to as a pseudorandom number generator

SISR — single-input signature register

SRSG — shift-register sequence generator; also a single-output PRPG

TPG — (generic) test-pattern generator

Figure 11.2 Glossary of key BIST test structures

11.1.2 Levels of Test

Production Testing

We refer to the testing of newly manufactured components as production testing. Production testing can occur at many levels, such as the chip, board, or system levels. Using BIST at these levels reduces the need for expensive ATE in go/no-go testing and simplifies some aspects of diagnostic testing. For example, the Intel 80386 microprocessor employs about 1.8 percent area overhead for BIST to test portions of the circuit that would be difficult to test by other means [Gelsinger 1987]. The BIST aspects of several other chips and/or boards are discussed in [Benowitz *et al.* 1975] [Boney and Rupp 1979], [Fasang 1982], [Kuban and Bruce 1984], [Beenker 1985], and [Karpovsky and Nagvajara 1989].

When implemented at the chip level along with boundary scan, BIST can be used effectively at all levels of a system's hierarchy. Since many BIST techniques can be run in real time, this method is superior to many non-BIST approaches and to some extent can be used for delay testing. It is not applicable, however, to parametric testing.

Field Testing

BIST can be used for field-level testing, eliminating the need for expensive special test equipment to diagnose faults down to field-replaceable units. This can have a great influence on the maintainability and thus life-cycle costs of both commercial and military hardware. For example, the U.S. military is attempting to implement the concept of *two-level maintenance*. Here a system must carry out a self-test and automatically diagnose a fault to a field-replaceable unit, such as a printed circuit board. This board is

then replaced "in the field" and the faulty board is either discarded or sent to a depot for further testing and repair.

11.2 Test-Pattern Generation for BIST

In Chapter 10 the design of a pseudorandom pattern generator based on the use of an LFSR was described. In this section various TPG designs will be described. We assume that the unit being tested is an n-input, m-output combinational circuit. The various forms of testing and related TPGs are summarized next.

Exhaustive testing
- Exhaustive test-pattern generators

Pseudorandom testing
- Weighted test generator
- Adaptive test generator

Pseudoexhaustive testing
- Syndrome driver counter
- Constant-weight counter
- Combined LFSR and shift register
- Combined LFSR and XOR gates
- Condensed LFSR
- Cyclic LFSR

11.2.1 Exhaustive Testing

Exhaustive testing deals with the testing of an n-input combinational circuit where all 2^n inputs are applied. A binary counter can be used as TPG. If a maximum-length autonomous LFSR is used, its design can be modified to include the all-zero state. Such an LFSR is referred to as a *complete LFSR*, and its design is described in [McCluskey 1981] and [Wang and McCluskey 1986d].

Exhaustive testing guarantees that all detectable faults that do not produce sequential behavior will be detected. Depending on the clock rate, this approach is usually not feasible if n is larger than about 22. Other techniques to be described are more practical when n is large. The concept of exhaustive testing is not generally applicable to sequential circuits.

11.2.2 Pseudorandom Testing

Pseudorandom testing deals with testing a circuit with test patterns that have many characteristics of random patterns but where the patterns are generated deterministically and hence are repeatable. Pseudorandom patterns can be generated with or without replacement. Generation with replacement implies that a test pattern may be generated more than once; without replacement implies that each pattern is unique. Not all 2^n test patterns need be generated. Pseudorandom test patterns without replacement can be generated by an autonomous LFSR. Pseudorandom testing is applicable to both combinational and sequential circuits. Fault coverage can be determined by fault simulation. The test length is selected to achieve an acceptable level of fault coverage. Unfortunately, some circuits contain random-pattern-resistant faults and thus require long

test lengths to insure a high fault coverage. Methods for estimating the probability of detecting a fault with random tests are described in Chapter 6. Further information on estimating test length as a function of fault coverage can be found in [Savir and Bardell 1984], [Williams 1985], [Chin and McCluskey 1987], and [Wagner *et al.* 1987].

The inherent attributes of an LFSR tend to produce test patterns having equal numbers of 0s and 1s on each output line. For many circuits it is better to bias the distribution of 0s and 1s to achieve a higher fault coverage with fewer test vectors. Consider, for example, a 4-input AND gate. When applying unbiased random inputs, the probability of applying at least one 0 to any input is 15/16. A 0 on any input makes it impossible to test any other input for *s-a*-0 or *s-a*-1. Thus there is a need to be able to generate test patterns having different distributions of 0s and 1s.

Some results relating the effectiveness of testing in terms of test length and fault coverage to the distribution characteristics of the test patterns have been reported in [Kumar 1980], [Archambeau and McCluskey 1984], [Lisanke *et al.* 1987], and [Wunderlich 1988].

Weighted Test Generation

A *weighted test generator* is a TPG where the distribution of 0s and 1s produced on the output lines is not necessarily uniform. Such a generator can be constructed using an autonomous LFSR and a combinational circuit. For example, the probability distribution of 0.5 for a 1 that is normally produced by a maximal-length LFSR can be easily changed to 0.25 or 0.75 to improve fault coverage. When testing a circuit using a weighted test generator, a preprocessing procedure is employed to determine one or more sets of weights. Different parts of a circuit may be tested more effectively than other parts by pseudorandom patterns having different distributions. Once these weights are determined, the appropriate circuitry can be designed to generate the pseudorandom patterns having the desired distributions. Further information on these test generators can be found in [Schnurmann *et al.* 1975], [Chin and McCluskey 1984], [Ha and Reddy 1986], and [Wunderlich 1987].

Adaptive Test Generation

Adaptive test generation also employs a weighted test-pattern generator. For this technique the results of fault simulation are used to modify the weights, thereby resulting in one or more probability distributions for the test patterns. Once these distributions are determined, an appropriate TPG can be designed. Tests generated by such a device tend to be efficient in terms of test length; however, the test-pattern-generation hardware can be complex. Further information can be found in [Parker 1976] and [Timoc *et al.* 1983].

11.2.3 Pseudoexhaustive Testing

Pseudoexhaustive testing achieves many of the benefits of exhaustive testing but usually requires far fewer test patterns. It relies on various forms of circuit segmentation and attempts to test each segment exhaustively. Because of the many subjects that are associated with pseudoexhaustive testing, we will first briefly outline the main topics to be discussed in this section.

A segment is a subcircuit of a circuit C. Segments need not be disjoint. There are several forms of segmentation, a few of which are listed below:

1. Logical segmentation
   a. Cone segmentation (verification testing)
   b. Sensitized path segmentation
2. Physical segmentation

When employing a pseudoexhaustive test to an n-input circuit, it is often possible to reconfigure the input lines so that tests need only be generated on m lines, where $m < n$, and these m lines can fanout and drive the n lines to the CUT. These m signals are referred to as *test signals*. A procedure for identifying these test signals will be presented.

Pseudoexhaustive testing can often be accomplished using constant-weight test patterns. Some theoretical results about such patterns will be presented. We will also describe several circuit structures that generate these patterns as well as other patterns used in pseudoexhaustive testing.

11.2.3.1 Logical Segmentation

In this section we will briefly describe two logical-segmentation techniques.

Cone Segmentation

In *cone segmentation* an m output circuit is logically segmented into m cones, each cone consisting of all logic associated with one output. Each cone is tested exhaustively, and all cones are tested concurrently. This form of testing was originally suggested by McCluskey [1984] and is called *verification testing*.

Consider a combinational circuit C with inputs $X = \{x_1, x_2, \ldots, x_n\}$ and outputs $Y = \{y_1, y_2, \ldots, y_m\}$. Let $y_i = f_i(X_i)$, where $X_i \subseteq X$. Let $w = \max_i\{|X_i|\}$. One form of a verification test produces all 2^w input patterns on all $\binom{n}{w}$ subsets of w inputs to C. The circuit under test is denoted as an (n,w)-CUT, where $w < n$. If $w = n$, then pseudoexhaustive testing simply becomes exhaustive testing. Figure 11.3 shows a (4,2)-CUT.

Sensitized-Path Segmentation

In Chapter 8 a segmentation technique based on circuit partitioning was presented. Some circuits can be segmented based on the concept of path sensitization. A trivial example is shown in Figure 11.4. To test C_1 exhaustively, 2^{n_1} patterns are applied to A while B is set to some value so that $D = 1$. Thus a sensitized path is established from C to F. C_2 is tested in a similar manner. By this process, the AND gate is also completely tested. Thus this circuit can be effectively tested using only $2^{n_1} + 2^{n_2} + 1$ test patterns, rather than $2^{n_1+n_2}$. More details on this form of testing can be found in [McCluskey and Bozorgui-Nesbat 1981], [Chandra *et al.* 1983], [Patashnik 1983], [Udell 1986], [Chen 1987], [Shperling and McCluskey 1987], and [Udell and McCluskey 1989].

11.2.3.2 Constant-Weight Patterns

Consider two positive integers n and k, where $k \leq n$. Let T be a set of binary n-tuples. Then T is said to *exhaustively cover all k-subspaces* if for all subsets of k bit positions, each of the 2^k binary patterns appears at least once among the $|T|$ n-tuples. For example, the set T shown below exhaustively covers all 2-spaces.

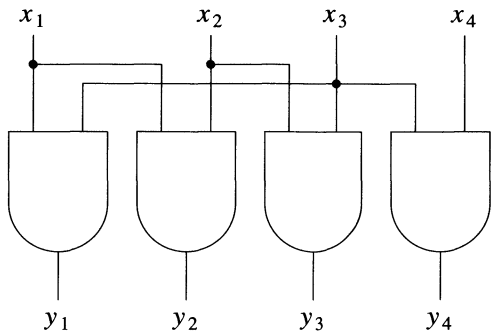


Figure 11.3 A (4,2)-CUT

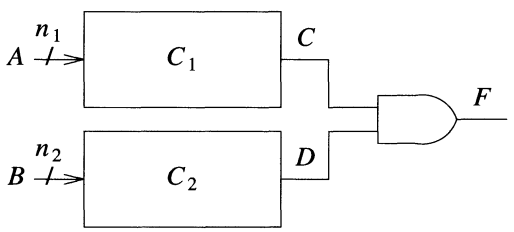


Figure 11.4 Segmentation testing via path sensitization

$$T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

If $|T|_{\min}$ is the smallest possible size for such a set T, then clearly $2^k \leq |T|_{\min} \leq 2^n$.

A binary n-tuple is said to be of weight k if it contains exactly k 1s. There are $\binom{n}{k}$ binary n-tuples having weight k.

The following results have been derived by [Tang and Woo 1983] and will be presented here without proof.

Theorem 11.1: Given n and k, then T exhaustively covers all binary k-subspaces if it contains all binary n-tuples of weight(s) w such that $w = c \bmod (n-k+1)$ for some integer constant c, where $0 \leq c \leq n-k$. □

Let T_c denote the set produced to using Theorem 11.1 for a specific value of c.

Example 11.1: $n = 20, k = 2, n - k + 1 = 19$.

Case 1: $c = 0$

Setting $w = 0 \bmod 19$ produces $w = 0$ and 19. Thus T_0 consists of the all-0 patterns and 20 patterns of weight 19. Hence

$$|T_0| = \binom{20}{0} + \binom{20}{19} = 1 + 20 = 21$$

and

$$T_0 = \left[\begin{array}{c|ccccc} 0\ 0 & \cdots & & & & 0 \\ \hline 0 & & & & & \\ & 0 & & & & \\ & & \ddots & & 1 & \\ & & & \iddots & & \\ & 1 & & & \ddots & \\ & & & & & 0 \end{array}\right]$$

Case 2: $c = 1$

Setting $w = 1 \bmod 19$ results in $w = 1, 20$. Therefore

$$|T_1| = \binom{20}{1} + \binom{20}{20} = 20 + 1 = 21$$

and

$$T_1 = \left[\begin{array}{c|ccccc} 1\ 1 & \cdots & & & & 1 \\ \hline 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & 0 & \\ & & & \iddots & & \\ & 0 & & & \ddots & \\ & & & & & 1 \end{array}\right]$$

Case 3: $2 \le c \le 18$

For $2 \le c \le 18$, $w = c \bmod 19$ implies that $w = c$. Thus in each case T_c consists of all weight-c binary n-tuples, and $|T_c| = \binom{n}{c}$. Note that T_0 and T_1 are the smallest among the 19 sets; in fact T_1 and T_0 are complements of each other. □

Example 11.2: $n = 20, k = 3, n - k + 1 = 18$

Case 1: $c = 0$

For $w = 0 \bmod 18$, $w = 0$ and 18. Thus

$$|T_0| = \binom{20}{0} + \binom{20}{18} = 1 + 190 = 191.$$

Case 2: $c = 1$

For $w = 1 \bmod 18$, $w = 1, 19$ and $|T_1| = \binom{20}{1} + \binom{20}{19} = 20 + 20 = 40.$

Case 3: $c = 2$

For $w = 2 \bmod 18$, $w = 2, 20$ and T_2 is the complement of T_0.

Case 4: $3 \le c \le 17$

For $w = c \bmod 8$, $3 \le c \le 17$, we have $w = c$. □

Note that for both examples, for any value of w, $0 \le w \le 20$, all n-tuples of weight w exist in exactly one case considered.

The general situation is covered by the following corollary.

Corollary 11.1: There are $(n-k+1)$ solution sets T_i, $0 \le i \le n-k$ obtained from Theorem 11.1, and these solution sets are disjoint and partition the set of all 2^n n-tuples into disjoint classes. □

Since these are $(n-k+1)$ solution sets that partition the set of 2^n distinct n-tuples into $(n-k+1)$ disjoint sets, and the smallest set cannot be larger than the average set, then an upper bound on the size of $|T|_{\min}$ is

$$|T|_{\min} \le \frac{2^n}{n-k+1} = B_n .$$

Theorem 11.2: Let T_c be a set generated according to Theorem 11.1. Then T_c is minimal; i.e., no proper subset of T_c also exhaustively covers all k-subspaces, if $c \le k$ or $c = n-k$. □

Example 11.3: $n = 6$, $k = 2$, $n - k + 1 = 5$.

For $c = 3$, $w = 3 \bmod 5$, thus $w = 3$ and $|T_3| = \binom{6}{3} = 20$. One subset of T_3 that exhaustively covers all 2-subspaces is shown below as T'_3.

$$T'_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

T'_3 is minimal and $|T'_3| = 6$. □

The upper bound of $|T_{\min}| \le B_n$ is tight when k is close to n and is loose when k is small. B_n grows exponentially as n increases, independent of the value of k.

For $k \le n/2$, the size of the minimum test set occurs when $w = \lfloor k/2 \rfloor$ and $\lfloor k/2 \rfloor + (n-k+1)$, resulting in

$$|T_{\min}| = \binom{n}{\lfloor k/2 \rfloor} + \binom{n}{k-\lfloor k/2 \rfloor - 1}$$

We will now consider two special cases.

Case 1: $n = k$

For this case $w = 0 \bmod 1$, hence $w = 0, 1, 2, \ldots, n$. Thus T_0 consists of the set of all 2^n binary n-tuples.

Case 2: $n = k + 1$

For this case $w = c \bmod 2$, and T_0 consists of the set of all binary n-tuples having odd parity, and T_1 consists of the set of all n-tuples having even parity. This situation is shown below for the case of $n = 4$ and $k = 3$.

$$T_0 = \left[\begin{array}{cccc} 0&0&0&0\\ 0&0&1&1\\ 0&1&0&1\\ 1&0&0&1\\ 0&1&1&0\\ 1&0&1&0\\ 1&1&0&0\\ 1&1&1&1 \end{array}\right] \text{ even parity} \qquad T_1 = \left[\begin{array}{cccc} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1\\ 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1 \end{array}\right] \text{ odd parity}$$

The following theorem verifies the correctness of this last case.

Theorem 11.3: In a matrix with 2^k distinct rows of $(k+1)$ binary values, where every row has the same parity, every set of k columns has all 2^k combinations of k values.

Proof: A set of k columns is obtained by removing one column of the matrix. We want to show that the 2^k rows of the remaining matrix are distinct. Let us assume the contrary, namely that two of the remaining rows — say, i and j — are identical. Because of the constant parity assumption, it follows that the bits of the removed column in the rows i and j of the original matrix must be equal. Hence rows i and j of the original matrix were also identical, which contradicts the assumption that all 2^k rows of $(k+1)$ bits are distinct. Therefore every set of k columns has all 2^k combinations of k values. □

11.2.3.3 Identification of Test Signal Inputs

Consider an n-input circuit. During testing it may be possible to apply inputs to p *test signal lines*, and have these p lines drive the n lines, where $p < n$. Clearly some of these p lines must fanout to two or more of the normal input lines. This section deals with the identification of these test signal lines and the tests associated with these lines.

Consider the circuit and test patterns shown in Figure 11.5. Note that f is a function of x and y, while g is a function of y and z. The four test vectors shown in Figure 11.5 test the individual functions f and g exhaustively and concurrently, even though to test the multiple-output function (f,g) exhaustively requires eight test vectors. Note that since no output is a function of both x and z, the same test data can be applied to both of these lines. Thus this circuit can be tested with only two test signals. A circuit is said to be a *maximal-test-concurrency* (MTC) *circuit*, if the minimal number of required test signals for the circuit is equal to the maximum number of inputs upon which any output depends. The circuit shown in Figure 11.5 is a MTC circuit.

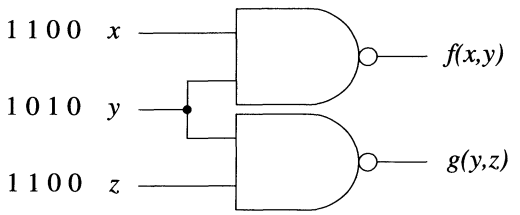


Figure 11.5 A maximal-test-concurrency circuit with verification test inputs $x = z$

Figure 11.6 shows a non-MTC circuit. Here every output is a function of only two inputs, but three test signals are required. However, each output can still be tested exhaustively by just four test patterns.

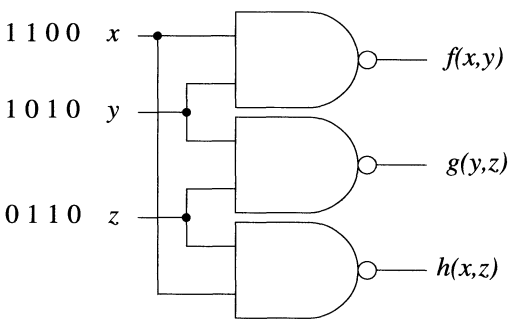


Figure 11.6 A nonmaximal-test-concurrency circuit with verification test inputs

Figure 11.7 shows a non-MTC circuit that requires four test signals; each output is a function of only two inputs, but five test patterns are required to exhaustively test all six outputs.

We next present a procedure for partitioning the inputs of a circuit to determine (1) the minimal number of signals required to test a circuit and (2) which inputs to the CUT can share the same test signal. We will also show how constant-weight patterns can be used to test the circuit. The various steps of the procedure will be illustrated as they are presented using the circuit C^* shown in functional form in Figure 11.8. From these results it will be possible to identify MTC circuits and to construct tests for MTC and non-MTC circuits.

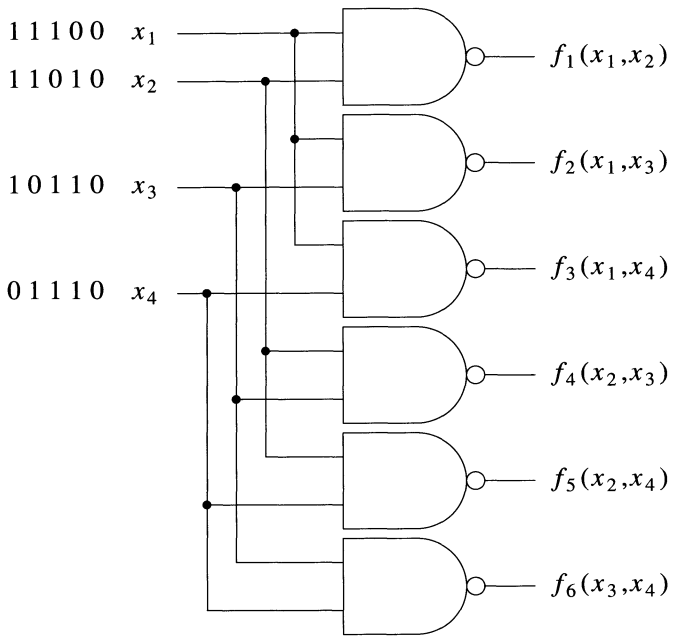


Figure 11.7 A nonmaximal-test-concurrency circuit with verification test inputs

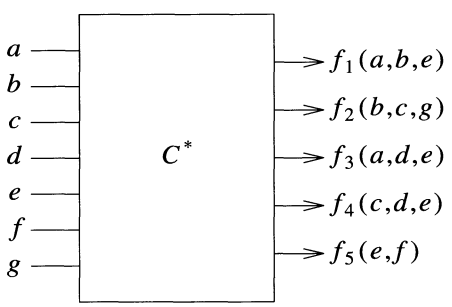


Figure 11.8 Circuit C^*

Procedure 11.1: Identification of Minimal Set of Test Signals

Step 1: Partition the circuit into disjoint subcircuits.

C^* consists of only one partition.

Step 2: For each disjoint subcircuit, carry out the following steps.

a. Generate a dependency matrix.

b. Partition the matrix into groups of inputs so that two or more inputs in a group do not affect the same output.

c. Collapse each group to form an equivalent input, called a test signal input.

For an n-input, m-output circuit, the *dependency matrix* $D = [d_{ij}]$ consists of m rows and n columns, where $d_{ij} = 1$ if output i depends on input j; otherwise $d_{ij} = 0$. For circuit C^*, we have

$$D = \begin{array}{c} \begin{matrix} a & b & c & d & e & f & g \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{matrix}$$

Reordering and grouping the inputs produce the modified matrix D_g.

$$D_g = \begin{array}{c} \begin{matrix} & & \textit{Group} & & & & \\ \textit{I} & & \textit{II} & & \textit{III} & \textit{IV} & \\ a & c & b & d & e & f & g \end{matrix} \\ \left[\begin{array}{cc:cc:c:cc} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{matrix}$$

In each group there must be less than two 1s in each row and the number of groups should be minimal. This insures that no output is driven by more than one input from each group.

Procedures for finding such a partition, which is a NP-complete problem, can be found in [McCluskey 1984] and [Barzilai *et al.* 1981].

The collapsed equivalent matrix D_c is obtained by ORing each row within a group to form a single column. The result for circuit C^* is

$$D_c = \begin{matrix} I & II & III & IV \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{matrix}$$

Step 3: Characterize the collapsed matrix D_c in terms of two parameters p and w, where p is the number of partitions in D_c, called the *width* of D_c, and w is the maximum number of 1s in any row, and is the *weight* of D_c. Note that p represents the maximum number of input signals required to test a disjoint subcircuit, and w represents the maximum number of signals upon which any output depends. Hence a pseudoexhaustive test must be of length 2^w or greater but need not exceed 2^p.

For D_c, we have $p = 4$ and $w = 3$.

A *universal minimal pseudoexhaustive test set* having parameters (p,w) is a minimal set of test patterns that contains, for all $\binom{p}{w}$ subsets consisting of w of the p signal lines, all 2^w test patterns. The properties of these test sets are determined by the relative values of p and w, where by definition, $p \geq w$. For a specific circuit, if all outputs are a function of w inputs, then this test set is minimal in length. Otherwise it may not be.

Step 4: Construct the test patterns for the circuit based upon the following three cases.

Case 1: $p = w$
Case 2: $p = w + 1$
Case 3: $p > w + 1$

Case 1: $p = w$

This case corresponds to MTC circuits and the test consists of all 2^p test patterns of p bits. The test can be easily generated by a counter or a complete LFSR. Clearly this is a universal minimal pseudoexhaustive test set. This case applies to Figure 11.5, where $p = w = 2$. Referring to the prior discussion on constant weight patterns, this case corresponds to the previous case where $k = n$ resulting in the test set T_0.

Case 2: $p = w + 1$

This case corresponds to the case where $n = k + 1$ in the previous discussion on constant weight patterns. The minimal test set consists of all possible patterns of p bits with either odd or even parity. There are $2^{p-1} = 2^w$ such patterns. Selecting odd parity, the tests for C^* are listed next.

A	B	C	D	
0	0	0	1	
0	0	1	0	
0	1	0	0	*parity* = 1
1	0	0	0	
0	1	1	1	
1	0	1	1	
1	1	0	1	*parity* = 3
1	1	1	0	

Note that for any subset of three columns, all 2^3 binary triplets occur.

This pseudoexhaustive test set consists of 8 patterns, while an exhaustive test would consist of 128 patterns. Each column represents an input to each line in a group; e.g., column A can be the input to lines a and c in the circuit shown in Figure 11.8.

This case also applies to the circuit shown in Figure 11.6, where $p = 3$ and test patterns of even parity are selected.

Case 3: $p > w + 1$

For this case, the test set consists of two or more pattern subsets, each of which contains all possible patterns of p bits having a specific constant weight. □

The total number of test patterns T is a function of p and w. Figure 11.9 shows the value of the constant weights and T for various values of p and w. Unfortunately, constant weights do not exist for all pairs of p and w. For such cases, w can be increased so as to achieve a constant-weight pseudoexhaustive test, but it may not be minimal in length.

The minimal test set for $p = 5$ and $w = 3$ corresponding to the constant-weight pair (1,4) (see Figure 11.9) is shown in Figure 11.10.

Unfortunately it is not always easy to construct a circuit to generate a pseudoexhaustive test set for $p > w + 1$, and the hardware overhead of some of these circuits is sometimes quite high. In the next subsection several techniques for designing circuits that generate pseudoexhaustive tests will be briefly described. Many of these designs do not generate minimal test sets, but the techniques lead to efficient hardware designs. Because most TPGs use some form of an LFSR, and since more than one test sequence is sometimes needed, often more than one seed value is required for initializing the state of the LFSR.

11.2.3.4 Test-Pattern Generators for Pseudoexhaustive Tests

Syndrome-Driver Counter

For the circuit shown in Figure 11.3, $y_1 = f_1(x_1,x_3)$, $y_2 = f_2(x_1,x_2)$, $y_3 = f_3(x_2,x_3)$, and $y_4 = f_4(x_3,x_4)$. Thus no output is a function of both x_1 and x_4, or of x_2 and x_4. (The width of this circuit is 3.) Hence x_1 and x_4 (or x_2 and x_4) can share the same input during testing. Thus this circuit can be pseudoexhaustively tested by applying all 2^3 input to x_1, x_2, and x_3, and by having the line driving x_1 or x_2 also drive x_4.

p	w	T	Constant weights
$p > 3$	2	$p + 1$	$(0,p-1)$ or $(1,p)$
$p > 4$	3	$2p$	$(1,p-1)$
$p > 5$	4	$\frac{1}{2}p(p+1)$	$(1,p-2)$ or $(2,p-1)$
$p > 6$	5	$p(p-1)$	$(2,p-2)$
$p = 8$	6	$\frac{1}{2}p(p+1)$	(1,4,7)
$p > 8$	6	$B(p+1,3)$	$(2,p-3)$ or $(3,p-2)$
$p = 9$	7	170	(0,3,6,9)
$p > 9$	7	$2B(p,3)$	$(3,p-3)$
$p = 10$	8	341	(0,3,6,9) or (1,4,7,10)
$p = 11$	8	496	(0,4,8) or (3,7,9)
$p > 11$	8	$B(p+1,4)$	$(3,p-4)$ or $(4,p-3)$
$p = 11$	9	682	(1,4,7,10)
$p = 12$	9	992	(0,4,8,12)
$p > 12$	9	$2B(p,4)$	$(4,p-4)$
$p = 12$	10	1365	(1,4,7,10) or (2,5,8,11)
$p = 13$	10	2016	(0,4,8,12) or (1,5,9,13)
$p = 14$	10	3004	(0,5,10) or (4,9,14)
$p > 14$	10	$B(p+1,5)$	$(4,p-5)$ or $(5,p-4)$

Figure 11.9 Constant weights for pseudoexhaustive tests. Note: $(\alpha_1,\alpha_2,...)$ represents a constant-weight vector, where α_i is the constant weight of one subset of patterns. $B(n,m)$ represents the binomial coefficient.

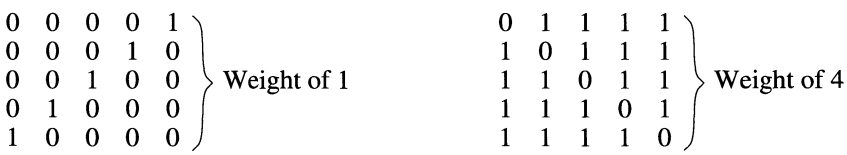


Figure 11.10 Constant-weight test set for (1,4)

In general if $(n-p)$ inputs, $p < n$, can share test signals with the p other inputs, then the circuit can be exhaustively tested using these p inputs. If $p = w$, only 2^p tests are required.

If $p > w$, $T = 2^p - 1$ tests are enough to test the circuit. T can be much greater than the lower-bound value of 2^w. For the example shown in Figure 11.3, $p = 3$ and

$2^3 - 1 = 7$. The reader can verify that the (1,1,1) test pattern is not required. In general the test patterns consisting of all 0s and all 1s are not required; hence $2^p - 2$ tests can be used.

The major problem with this approach is that when p is close in value to n a large number of test patterns are still required.

This testing scheme was proposed by Barzilai *et al.* [1981] and uses a *syndrome-driver counter* (SDC) to generate test patterns. The SDC can be either a binary counter or an LFSR and contains only p storage cells.

Constant-Weight Counter

A *constant-weight code* (CWC), also known as an N-out-of-M code, consists of the set of codewords of M binary bits, where each codeword has exactly N 1s. A constant-weight code is analogous to the constant-weight patterns discussed previously. The 2-out-of-4 CWC is shown below.

1 1 0 0
1 0 1 0
1 0 0 1
0 1 1 0
0 1 0 1
0 0 1 1

Note that for any two columns all possible pairs of binary values appear. The same is not true of a 2-out-of-3 CWC.

Any (n,w)-circuit can be pseudoexhaustively tested by a constant-weight counter implementing a w-out-of-K code, for an appropriate value of K. For large values of N and M, the complexity of the associated constant-weight counter is often high. More information on CWCs and the design of constant-weight counters can be found in [Ichikawa 1982], [Tang and Woo 1983], and [McCluskey 1984].

Combined LFSR/SR

Another way to produce a pseudoexhaustive test, proposed in [Barzilai *et al.* 1983] and [Tang and Chen 1984a], is to employ a combination of an LFSR and a shift register (SR), as shown in Figure 11.11. A 4-stage combined LFSR/SR for testing a (4,2)-CUT is shown in Figure 11.12. The implementation cost for this design approach is less than for a constant-weight counter, but the resulting circuit usually generates more test vectors. Also, the LFSR usually requires at least two seed values. The number of test patterns generated is near minimal when w is much less than n; e.g., $w < n/2$. LFSRs are usually designed to have a shift mode of operation. This mode is used for loading seed values into the registers. If the register is also used as a signature analyzer, the shift mode is also used for scanning out the final signature.

Combined LFSR/XOR Gates

Still another way to produce a pseudoexhaustive test, shown in Figure 11.13, uses a combination of an LFSR and a XOR (linear) network. The design of these networks is based on the use of linear sums [Akers 1985] or linear codes [Vasanthavada and Marinos

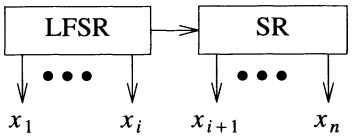


Figure 11.11 An LFSR/SR verification test generator

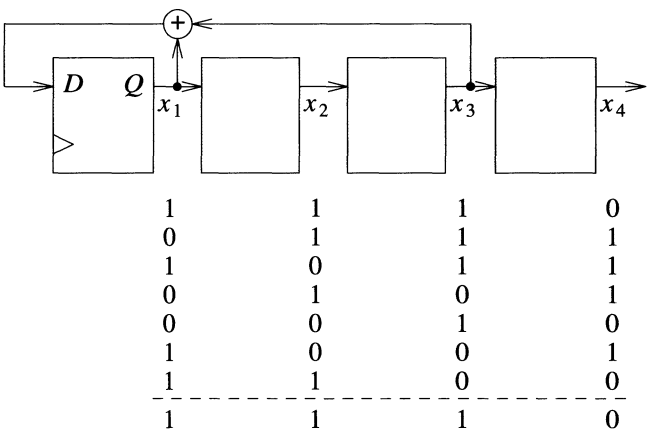


Figure 11.12 A 4-stage LFSR/SR for a (4,2)-CUT

1985]. These designs require at most two seeds, and the number of test patterns needed to ensure pseudoexhaustive testing is close to that required for LFSR/SR designs.

Figure 11.14 shows a combined LFSR/XOR TPG along with the patterns it produced. This device can test a (4,2)-CUT.

Condensed LFSR

Another design approach, proposed by Wang and McCluskey [1984, 1986b] and referred to as *condensed LFSR*, uses at most two seeds, leads to simple designs, and produces a very efficient test set when $w \geq n/2$. When $w < n/2$ this technique uses more tests than the LFSR/SR approach. Condensed LFSRs are based on the concept of linear codes [Peterson and Weldon 1972, Lin and Costello 1983]. An *(n,k)-linear code* over a Galois field of 2 generates a set S of n-tuples containing 2^k distinct code words, where if $c_1 \in S$ and $c_2 \in S$, then $c_1 \oplus c_2 \in S$.

Using a type 2 LFSR having a characteristic polynomial $p(x)$, a condensed LFSR for a (n,w)-CUT can be constructed as follows. Let k be the smallest integer such that

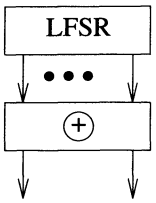


Figure 11.13 A combined LFSR/XOR verification-test generator

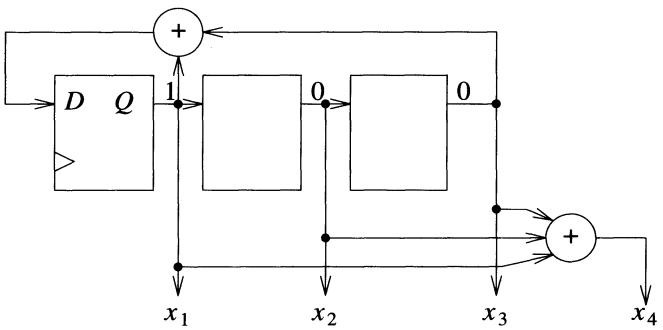


Figure 11.14 A combined LFSR/XOR TPG for a (4,2)-CUT

$$w \leq \lceil k/(n-k+1) \rceil + \lfloor k/(n-k+1) \rfloor$$

Then a type 2 LFSR realizes a condensed LFSR if

$$p(x) = (1+x+x^2+ \ldots +x^{n-k})\,q(x)$$

where $q(x)$ is a primitive polynomial of degree k. Also the seed polynomial $S_0(x)$ must be divisible by $(1+x+x^2+ \ldots +x^{n-k})$.

Again consider the case $(n,w) = (4,2)$: then $k = 3$ and $(n-k) = 1$. Selecting $q(x) = 1 + x + x^3$, we obtain

$$p(x) = (1+x)(1+x+x^3) = 1 + x^2 + x^3 + x^4$$

Figure 11.15 shows the resulting design and initial seed. Although a condensed LFSR has n stages, the feedback circuitry is usually simple.

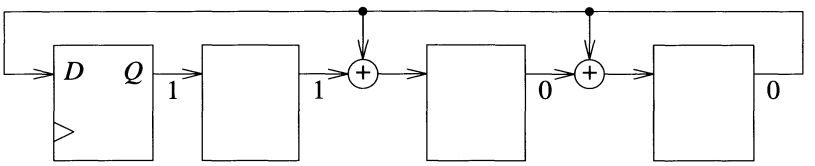


Figure 11.15 A condensed LFSR for a (4,2)-CUT

Cyclic LFSR

When $w < n/2$, condensed LFSR designs produce long tests for (n,w)-CUTs. LFSR/XOR designs reduce this test length but have a high hardware overhead. For $w < n/2$, *cyclic LFSRs* lead to both efficient tests and low hardware overhead. Cyclic LFSRs are based on cyclic codes [Peterson and Weldon 1972, Lin and Costello 1983]. An *(n,k)-cyclic code* over the Galois field of 2 contains a set of 2^k distinct codewords, each of which is an n-tuple satisfying the following property: if c is a codeword, then the n-tuple obtained by rotating c one place to the right is also a code word. Cyclic codes are a subclass of linear codes. The design of cyclic LFSRs and details for obtaining the characteristic polynomial for a cyclic LFSR are presented in [Wang 1982] and [Wang and McCluskey 1986f, 1986g, 1987a, 1987c].

11.2.3.5 Physical Segmentation

For very large circuits, the techniques described for pseudoexhaustive testing often lead to large test sets. In these cases, pseudoexhaustive testing can still be achieved by employing the concept of *physical segmentation.* Here a circuit is divided or partitioned into subcircuits by employing hardware-segmentation techniques.

One such technique is shown in Figure 9.11. Various ways for segmenting a circuit based on this type of structure are presented in [Patashnik 1983], [Archambeau 1985], and [Shperling and McCluskey 1987].

More details on this form of testing can be found in [McCluskey and Bozorgui-Nesbat 1981], [Chandra *et al.* 1983], [Udell 1986], [Chen 1987], and [Udell and McCluskey 1989].

Physical segmentation can also be achieved by inserting bypass storage cells in various signal lines. A *bypass storage cell* is a storage cell that in normal mode acts as wire, but in the test mode can be part of an LFSR circuit. It is similar to a cell used in boundary-scan designs, such as the one shown in Figure 9.14. If inserted into line x, then the associated LFSR can be used as a MISR and hence to detect errors occurring on line x, or it can be used as a PRPG and hence to generate test patterns on line x.

Example 11.4: Consider the circuit C shown in Figure 11.16(a), where the logic blocks G_i, $i = 1, 2, ..., 9$ are represented by circles. Next to each block is an integer indicating the number of primary inputs that can affect the output of the block. Assume it is desired

to segment this circuit using bypass storage cells so that no signal is a function of more than w variables. Figure 11.16(b) shows how three bypass storage cells can be added to this circuit to achieve a value of $w = 4$.

Figure 11.17 shows the four segments of C created by the bypass storage cells. Figure 11.17(a) depicts the subcircuit C_1; here G_3 is tested along with its associated interconnecting lines, and the storage cells associated with x_4, x_5, x_6 and x_7, labeled P, are part of a TPG; the bypass storage cell on the output of G_3, labeled S, is part of a ORA. Similarly, Figure 11.17(b) shows the subcircuit C_2 where G_4 and G_7 are tested. G_1 and G_5 are tested as part of C_3 (see Figure 11.17(c)); G_2, G_6, and G_8 are tested as part of C_4 (see Figure 11.17(d)). One additional subcircuit C_5 (not shown) is needed to test G_9, along with G_2 and G_6. □

Procedures for this form of segmentation and that also take into account the design of the corresponding TPG and ORA circuitry are discussed in [Jone and Papachristou 1989].

11.3 Generic Off-Line BIST Architectures

In the previous section we considered BIST approaches for testing a single block of combinational logic. In this section we will consider general off-line BIST structures that are applicable to chips and boards consisting of blocks of combinational logic interconnected by storage cells.

Off-line BIST architectures at the chip and board level can be classified according to the following criteria:

1. centralized or distributed BIST circuitry;
2. embedded or separate BIST elements.

BIST architectures consist of several key elements, namely

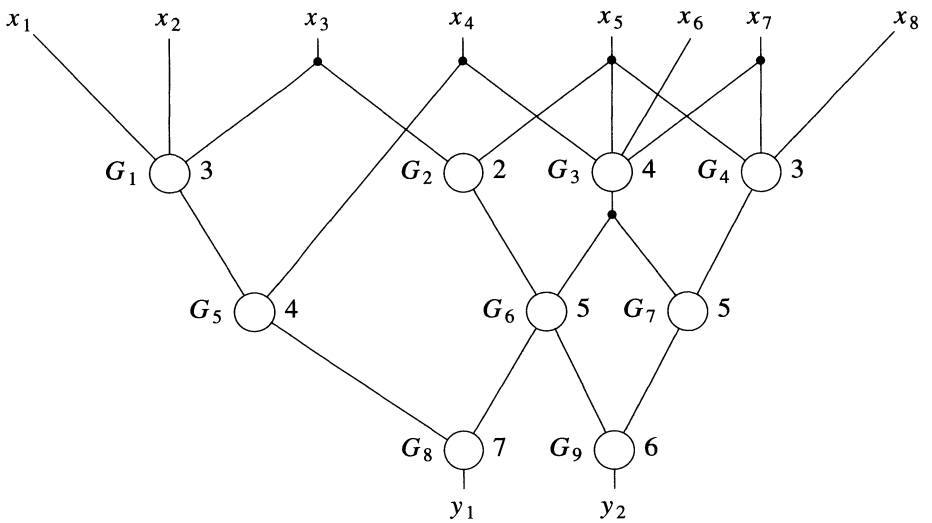
1. test-pattern generators;
2. output-response analyzers;
3. the circuit under test;
4. a distribution system (DIST) for transmitting data from TPGs to CUTs and from CUTs to ORAs;
5. a BIST controller for controlling the BIST circuitry and CUT during self-test.

Often all or parts of the controller are off-chip. The distribution system consists primarily of direct interconnections (wires), busses, multiplexers, and scan paths.

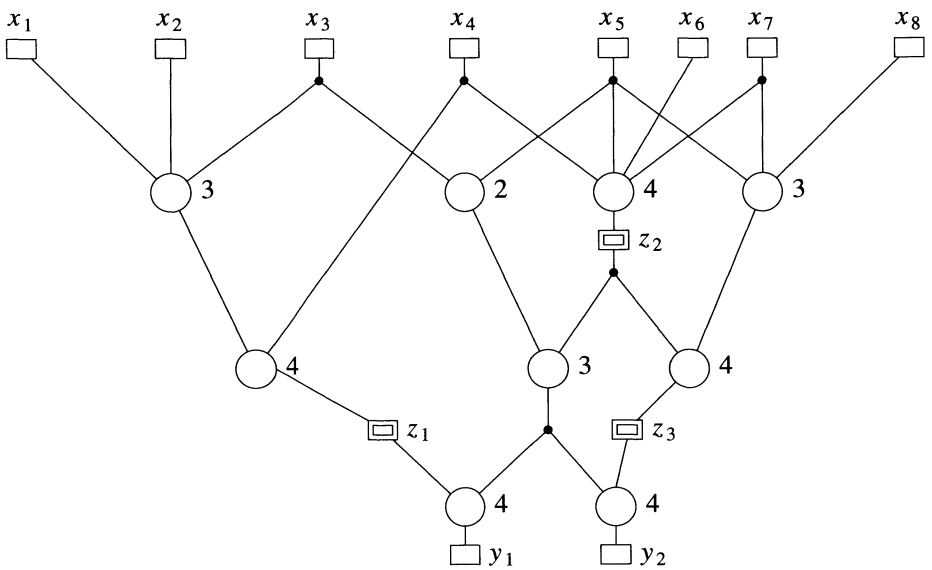
The general form of a centralized BIST architecture is shown in Figure 11.18. Here several CUTs share TPG and ORA circuitry. This leads to reduced overhead but increased test time.

During testing, the BIST controller may carry out one or more of the following functions:

1. Single-step the CUTs through some test sequence.
2. Inhibit system clocks and control test clocks.



(a)

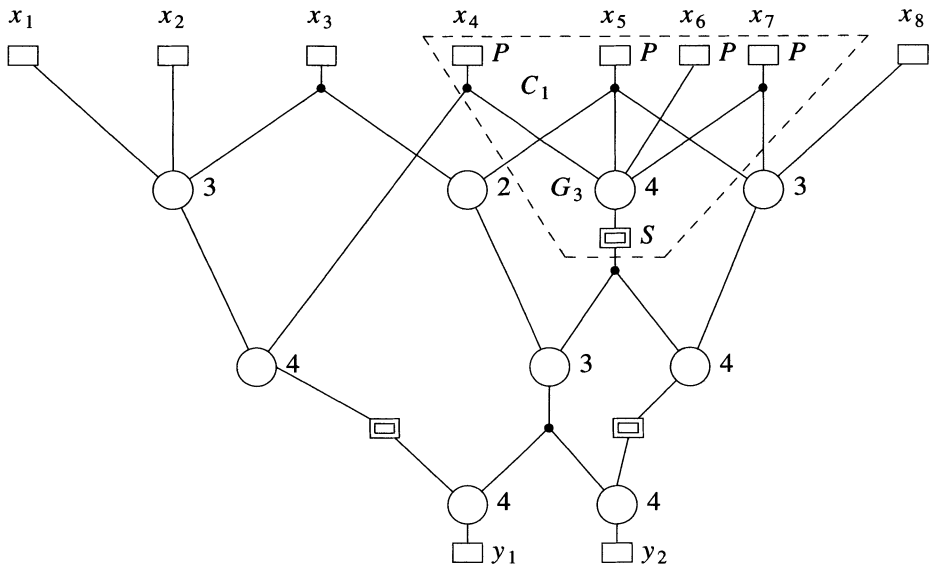


(b)

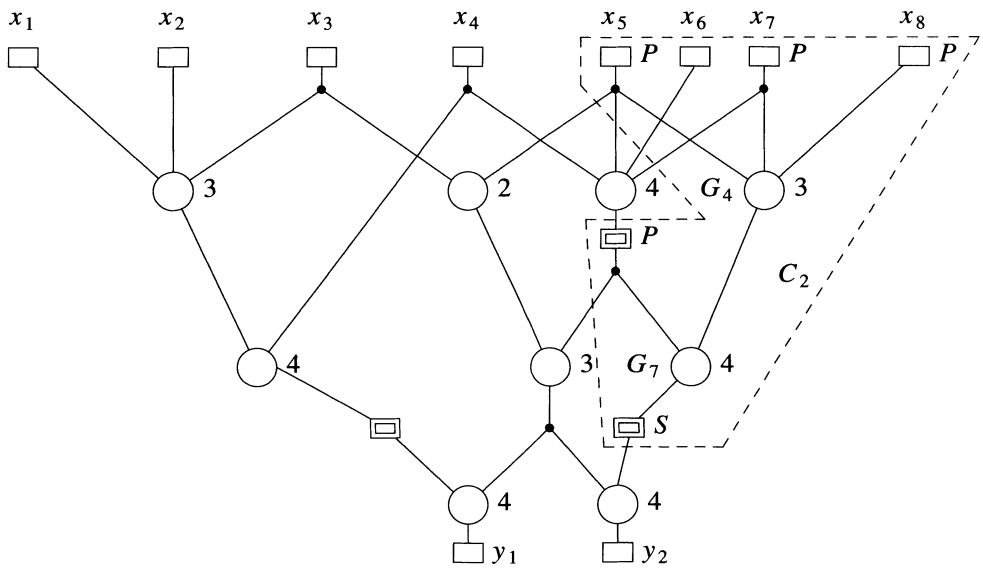
Key

□ — normal I/O storage cell ▣ — bypass storage cell ○ — a logic block

Figure 11.16 Inserting bypass storage cells to achieve $w = 4$



(a)

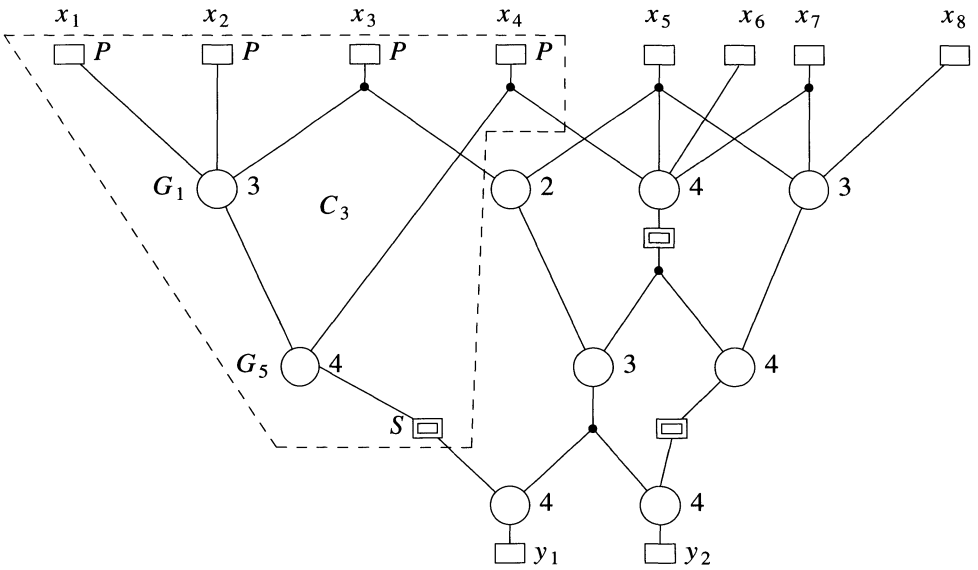


(b)

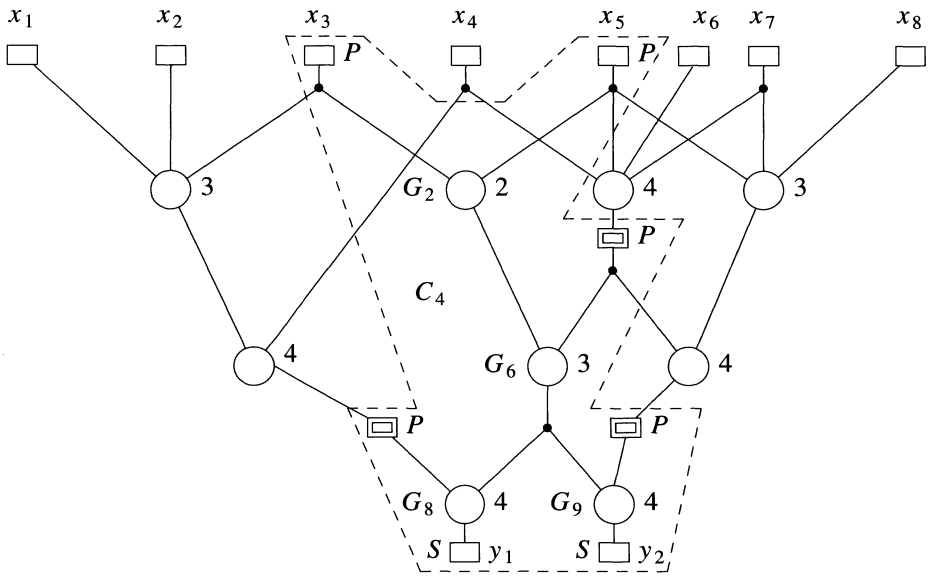
Key

□ — normal I/O storage cell ▣ — bypass storage cell ○ — a logic block

Figure 11.17 Four segments formed by the bypass storage cells



(c)



(d)

Key

□ — normal I/O storage cell ▣ — bypass storage cell ○ — a logic block

Figure 11.17 (Continued)

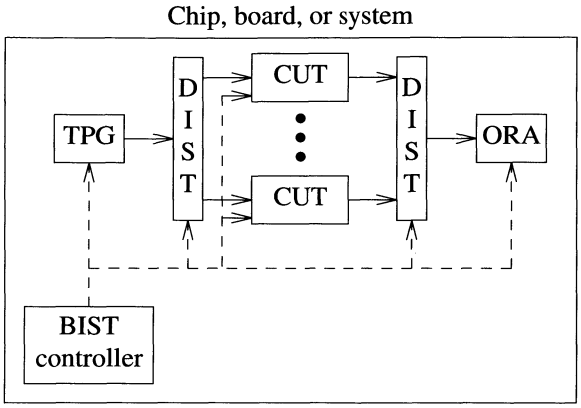


Figure 11.18 Generic form of centralized and separate BIST architecture

3. Communicate with other test controllers, possibly using test busses.
4. Control the operation of a self-test, including seeding of registers, keeping track of the number of shift commands required in a scan operation, and keeping track of the number of test patterns that have been processed.

Further information on the design of controllers for BIST circuitry can be found in [Breuer *et al.* 1988].

The distributed BIST architecture is shown in Figure 11.19. Here each CUT is associated with its own TPG and ORA circuitry. This leads to more overhead but less test time and usually more accurate diagnosis. The BIST control circuitry is not shown. The designs shown in Figures 11.18 and 11.19 are examples of the separate BIST architecture, since the TPG and ORA circuitry is external to the CUT and hence not part of the functional circuitry.

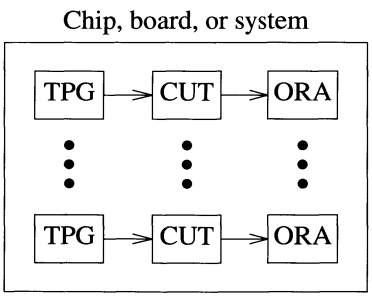


Figure 11.19 Generic form of distributed and separate BIST architecture

Figure 11.20 shows the general form of the distributed and embedded BIST architecture. Here the TPG and ORA elements are configured from functional elements within the CUT, such as registers. This leads to a more complex design to control, but has less hardware than distributed and separate architectures have.

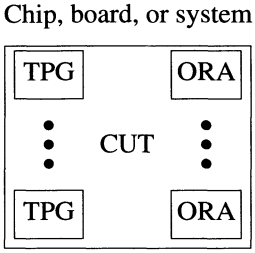


Figure 11.20 Generic form of distributed and embedded BIST architecture

The choice of a BIST architecture is a function of several factors, some of which are listed next.

1. *Degree of test parallelism:* Distributed BIST provides a higher degree of test parallelism, since several CUTs can be tested at the same time.
2. *Fault coverage:* Distributed BIST usually leads to a higher fault coverage since TPG and ORA circuits can be customized to each CUT. For example, a BIST technique for a block of combinational logic may not be suitable for a RAM.
3. *Level of packaging:* At higher levels, centralized BIST becomes more natural. For example, microdiagnostics testing is only applicable at the level where a microprogrammable controller exists. This controller can then be used to test many components of a system.
4. *Test time:* Distributed BIST usually leads to a reduction in test time.
5. *Physical constraints:* Size, weight, power, cooling costs, and other factors influence the design. Often embedded and separate BIST architectures require more hardware and degrade performance.
6. *Complexity of replaceable units:* If a board is a replaceable unit and is to be self-testable, then it must contain TPG and ORA circuitry. If a system is the lowest level of replaceable unit, then its constituent boards need not have TPG and ORA circuitry and a more centralized BIST architecture can be used.
7. *Factory and field test-and-repair strategy:* The type of ATE and degree to which it is used to test and diagnose failures influences and is influenced by BIST. For example, because of the increasing use of surface-mounted devices, in-circuit or bed-of-nails testing is becoming more difficult to use and hence the need for BIST and the use of boundary scan.
8. *Performance degradation:* Adding BIST hardware in critical timing paths of a circuit may require a reduction in the system clock rate.

11.4 Specific BIST Architectures

In this section several BIST architectures proposed by various research and development groups will be described. We will denote sequential blocks of logic by S and combinational blocks by C.

11.4.1 A Centralized and Separate Board-Level BIST Architecture (CSBL)

Figure 11.21 illustrates a *centralized and separate board-level* (CSBL) BIST architecture proposed by Benowitz *et al.* [1975]. It has the following attributes:

- centralized and separate BIST architecture;
- no boundary scan;
- combinational or sequential CUT.

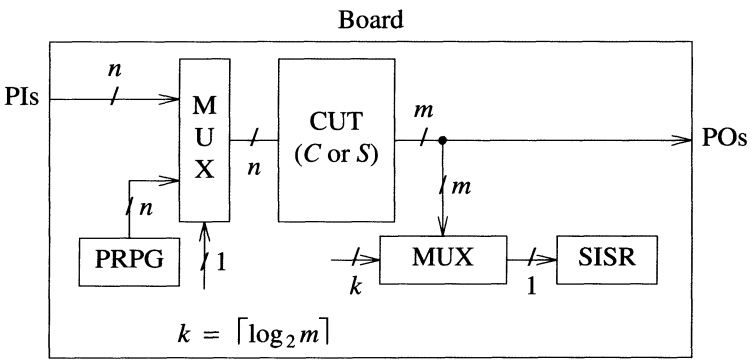


Figure 11.21 A centralized and separate BIST architecture (CSBL)

During the off-line test mode, the inputs are driven by a PRPG and the outputs are monitored using a single-input signature analyzer. To reduce hardware costs, the test is repeated m times, once for each output; hence only one signature analyzer is required.

This method is best suited for pipeline circuits with limited feedback. General finite-state controllers can be more complex to test. Microprocessors represent a very complex test situation. Extensive fault simulation is required to determine the number of test vectors required to achieve an adequate level of fault coverage.

11.4.2 Built-In Evaluation and Self-Test (BEST)

The *built-in evaluation and self-test* (BEST) architecture is an application of the CSBL design to chips (see Figure 11.22). In general the logic being tested is a sequential circuit. The inputs to the CUT are driven by a PRPG and the outputs are compressed using a MISR. The details of switching between the primary inputs and the output of the PRPG when applying the normal or the test inputs to the CUT are not shown. Either a MUX can be used, or the primary inputs can first be loaded into the PRPG and then

applied to the CUT. The same concepts apply to the outputs. Both an embedded and a separate version of this architecture exist. For example, if the primary inputs to the CUT go directly to registers, and the primary outputs are driven by registers, then an embedded BIST design can be used. That is, these I/O registers can be modified for use as the PRPG and MISR. Otherwise the PRPG and MISR need to be added to the CUT, which results in a separate architecture. In the latter case, the PRPG and MISR can be made part of the boundary-scan registers.

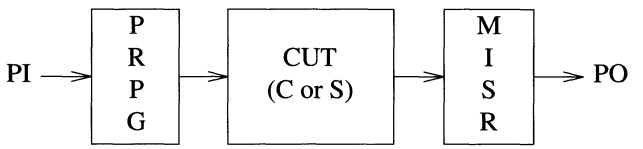


Figure 11.22 The built-in evaluation and self-test (BEST) BIST architecture

The hardware overhead for the BEST architecture is low. Again, extensive fault simulation is required to determine an acceptable balance between fault coverage and test length. For some circuits this technique can be ineffective in achieving an acceptable level of fault coverage. Further information can be found in [Resnick 1983] and [Lake 1986].

The previous BIST architectures often result in low fault coverage because they rely on the use of pseudorandom patterns for testing a sequential circuit. To circumvent this problem, an internal scan path can be used within the CUT so that the testing of the CUT can be reduced to the problem of testing combinational logic. The next few BIST architectures will illustrate this concept.

11.4.3 Random-Test Socket (RTS)

The *random-test socket* (RTS) [Bardell and McAnney 1982] is not a true BIST architecture because the test circuitry is external to the CUT. The RTS architecture, shown in Figure 11.23, has the following attributes:

- distributed and separate test hardware;
- no boundary scan;
- scan path (LSSD) CUT architecture.

The CUT is tested as follows:

1. Initialize the LFSRs.
2. Load a pseudorandom test pattern into the scan path using R_2.
3. Generate a new pseudorandom test pattern using R_1. This takes one clock cycle. The vector is applied to the primary inputs of the CUT.
4. Capture the response on the primary outputs of the CUT by applying one clock pulse to R_3.

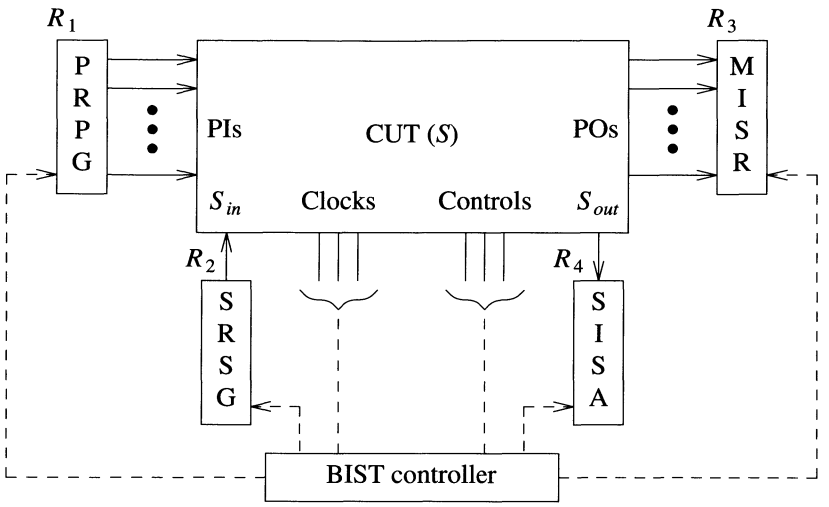


Figure 11.23 Random-test socket (RTS)

5. Execute a parallel-load operation on the system storage cells to capture the response to the random test pattern.
6. Scan out the data in the internal scan path of the CUT and compress these data in R_4.

Steps 2-6 are repeated until either an adequate fault coverage is achieved or the maximum allowable test time is reached. Also, steps 2 and 6 can be carried out simultaneously. The circuit is considered to be fault-free if the final values in R_3 and R_4 are correct.

The RTS test approach can suffer from problems related to fault coverage and test time. Testing is inherently slow, since for each test pattern the entire scan path must be loaded. Test time can be reduced by partitioning the storage cells into several scan paths, but this requires additional I/O pins.

Testing with pseudorandom test patterns requires more patterns than testing with deterministic test patterns. The number of test patterns is determined by the desired fault coverage, the circuit itself, and the characteristics of the LFSRs used to generate the test data and produce the signatures. For a high fault coverage, the population of random-pattern-resistant faults will dictate the number of tests required. Test points can be effectively used to reduce this number.

Fault coverage can be adversely affected by linear correlations among the bits generated by LFSRs and by the periodicity of their sequences [Bardell and McAnney 1986, Chen 1986]. Assume, for example, that the LFSR R_2 has a period p and that the length of the scan path is k, where either p divides k, or k divides p. Then, in the former case only k/p unique patterns can be loaded in the scan path; in the latter case only p/k unique patterns exist. Thus it is important that neither of these situations occur. Also, if a primitive

polynomial is not used, then all possible patterns are not generated. Some of these patterns may be required to detect specific faults.

Even when $p >> k$, a complete load of the scan path between each test vector implies that large blocks of combinational logic in the CUT will probably not be tested exhaustively.

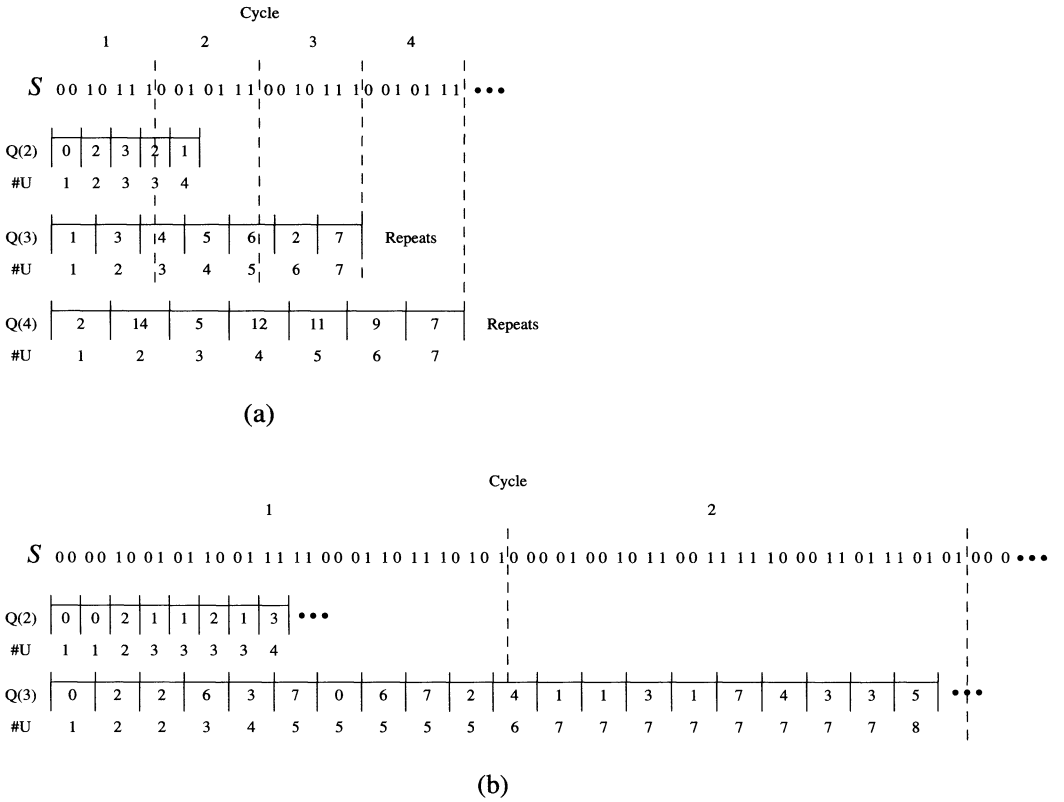


Figure 11.24 Test patterns generated by LFSRs

Figure 11.24 illustrates the test patterns generated by two LFSRs for different values of k and p. S refers to the sequence of bits generated by the LFSR. $Q(k)$ indicates the integer (base 10) value of the k-bit test pattern generated, and # U indicates the number of unique test patterns generated. For example, Figure 11.24(a) shows the result for a type 1 LFSR with characteristic polynomial $P(x) = 1 + x^2 + x^3$ and initial state 100. The sequence generated is 0010111 and $p = 7$. For $k = 2$, five test patterns must be generated before all four unique test patterns are produced. For $k = 3$ after three cycles seven unique test patterns have been generated. The test patterns now repeat, hence the (000) pattern is never generated. For $k = 4$ the patterns repeat after four cycles and only 7 of the 16 possible test patterns are generated. Figure 11.24(b) shows comparable results for the case $P(x) = 1 + x^3 + x^5$, where $p = 31$. For $k = 3$, 20 test patterns must be generated before all 8 possible patterns are produced.

By incorporating the LFSRs and controller within the CUT, and by multiplexing between test inputs and normal system inputs, the RTS architecture can be transformed into a BIST architecture.

11.4.4 LSSD On-Chip Self-Test (LOCST)

Figure 11.25 shows the general form of the *LSSD on-chip self-test* (LOCST) architecture, which is an extension of a BIST version of the RTS design. This architecture has the following attributes:

- centralized and separate BIST architecture;
- scan path (LSSD) CUT architecture;
- boundary scan;
- on-chip test controller.

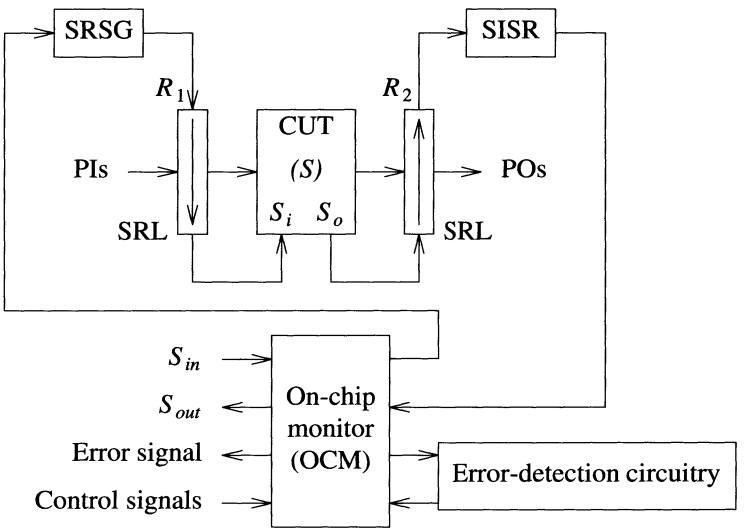


Figure 11.25 The LOCST architecture

The inputs and outputs are buffered through two boundary scan registers, denoted by R_1 and R_2 in Figure 11.25. The design includes an on-chip controller, referred to as an on-chip monitor (OCM), which is driven by a test bus and is used to partially control the test process. One implementation of LOCST uses a 20-bit SRSG having the characteristic polynomial $x^{20} + x^{17} + 1$, and a 16-bit SISR with characteristic polynomial $x^{16} + x^9 + x^7 + x^4 + 1$.

The test process is as follows:

1. *Initialize:* The scan path is loaded with seed data via the S_{in} line.
2. *Activate self-test mode:* Disable system clocks on R_1, R_2; enable LFSR operation.

3. *Execute self-test operation:*
   a. Load the scan path with a pseudorandom test pattern. Data in the scan path is compressed in the SISR. This step requires multiple clock cycles and is controlled by off-chip signals.
   b. Activate the system clocks for one cycle; register R_2 and the internal scan path will load data.
   c. Steps a and b are repeated until an adequate level of fault coverage is achieved.
4. *Check result:* Compare the final value in the SISR with the known good signature.

Data in the scan path can also be scanned out of the chip using the S_{out} line. The design can be embedded rather than separate by implementing the TPG and ORA using the system latches. For this case, the SRSG becomes a PRPG, and the SISR a MISR.

Some versions of this form of BIST architecture do not employ boundary-scan registers. Here some logic, referred to as *external logic*, does not get fully tested unless external test equipment is used. The external logic consists of (a) those blocks of logic driven by primary inputs and (b) blocks of logic that drive primary outputs.

More details on this architecture can be found in [Eichelberger and Lindbloom 1983] and [LeBlanc 1984].

11.4.5 Self-Testing Using MISR and Parallel SRSG* (STUMPS)

The *self-testing using MISR and parallel SRSG* (STUMPS) architecture is shown in Figure 11.26 [Bardell and McAnney 1982, 1984]. It was originally applied at the board level, and subsequently at the chip level. It has the following attributes:

- centralized and separate BIST architecture;
- multiple scan paths;
- no boundary scan.

The scan paths are driven in parallel by a PRPG, and the signature is generated in parallel from each scan path using a MISR. At the board level, each scan path corresponds to the scan path in a separate chip; at the chip level each scan path is just one segment of the entire scan path of a chip.

The use of multiple scan paths leads to a significant reduction in test time. Since the scan paths may be of different lengths, the PRPG is run for K clock cycles to load up the scan paths, where K is the length of the longest scan path. For short scan paths, some of the data generated by the PRPG flow over into the MISR.

When this approach is applied at the board level to chips designed with a scan path, then the PRPG and the MISR can be combined into a special-purpose test chip, which must be added to the board.

---

* A SRSG is equivalent to a PRPG.

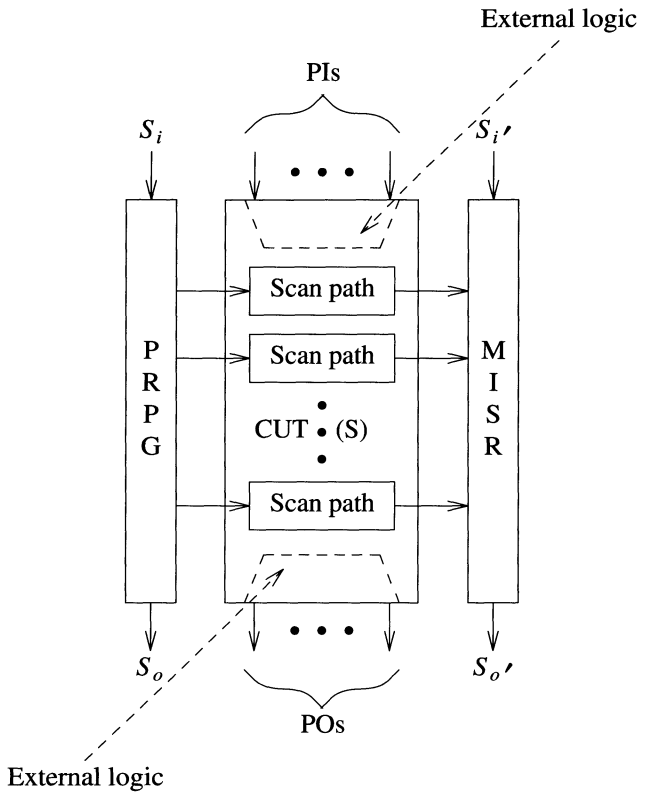


Figure 11.26 Self-test using MISR and parallel SRSG (STUMPS)

As before, problems related to linear correlation of data and periodicity can adversely affect the performance of this architecture. Note that if a type 1 LFSR is used in the PRPG, then $Q_i(t) = Q_{i-1}(t-1)$. Hence data in one scan path are a shifted version of data in another scan path. To avoid this situation, a type 2 LFSR can be used. The external logic must be tested via ATE or by adding boundary-scan registers.

11.4.6 A Concurrent BIST Architecture (CBIST)

Saluja *et al.* [1988] have extended the concepts of off-line BIST to include on-line BIST. One version of their proposed *concurrent BIST* (CBIST) architecture is shown in Figure 11.27. The CUT must be combinational logic. The BIST hardware is separate from the normal hardware. The technique can be employed in sequential circuits, if the circuitry is partitioned into blocks of combinational logic that can be tested as separate entities. No boundary or internal scan paths need to be used. To reduce hardware overhead, a centralized BIST architecture can be employed. During off-line testing, the PRPG drives the CUT, whose response is compressed in the MISR. For on-line testing, the PRPG and MISR are first initialized and held in their initial state until enabled by the *EN* signal. Normal inputs are applied to the CUT. When a match exists between the

normal inputs and the state of the PRPG, an enable signal is generated allowing the PRPG to advance to its next state, while the MISR samples the output and also advances to its next state. This process is repeated whenever a match occurs between the normal input data and the current state of the PRPG. When the state of the PRPG reaches some prespecified final state, the signature in the MISR is verified.

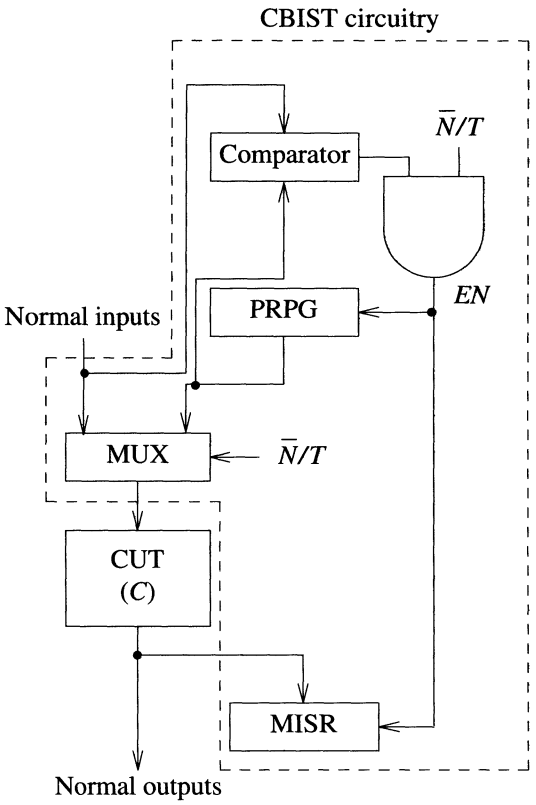


Figure 11.27 Concurrent BIST for combinational logic

The *test latency* for a circuit is the time required for the PRPG to go through all states corresponding to the desired test sequence. The expected value of the test latency can be estimated if certain assumptions on the distribution of inputs are made. For example, assume that the CUT has n inputs and that all input patterns are equally likely. Then the probability that an input will match the state of the PRPG, denoted by p, is $1/2^n$. Let $q = 1 - p$. Assume that L test patterns are applied to the CUT, that the CUT is to be tested exhaustively, and that $L \geq 2^n$. Let P_s denote the probability that the PRPG reaches its final state during the time these patterns are being applied. Then

$$P_s = 1 - \sum_{k=0}^{2^n - 1} \binom{L}{k} p^k \, q^{L-k}$$

Let the concurrent test latency for exhaustive testing, denoted by $C(\alpha)$, be the time (in clock cycles) during normal operation that it takes the PRPG to go through all 2^n states with a probability α. Figure 11.28 shows a few values of $C(\alpha)$ for various values of α and n. Here $C(\alpha)$ is expressed in terms of both L and time, where a 10 MHz system clock is assumed.

n	C(0.90)		C(0.99)	
	L	sec.	L	sec.
10	109089	0.11	1127044	0.11
12	17114557	1.71	17395982	1.74
14	271129177	27.11	273349519	27.33

Figure 11.28 C(α) for different values of n and α

11.4.7 A Centralized and Embedded BIST Architecture with Boundary Scan (CEBS)

Figure 11.29 shows a *centralized and embedded BIST architecture with boundary scan* (CEBS); this is a version of the architecture depicted in Figure 11.25. Here the first r bits of the input boundary scan register labeled RPG act as a PRPG and a SRSG, and the last s bits of the output boundary scan register act as both a MISR and a SISR. Testing proceeds as follows. A test mode signal is set and the scan registers are seeded. Then the RPG loads the scan path with pseudorandom test data. A system clock is issued and the scan-path registers are loaded in parallel with system data, except for the signature register SR, which operates in the MISR mode. The scan path is again loaded with pseudorandom data while the signature register operates in the SISR mode, compressing data that were in the scan path.

More details on this type of design can be found in [Komanytsky 1982, 1983] and [Butt and El-Ziq 1984].

11.4.8 Random Test Data (RTD)

One major problem with many of the techniques presented previously is that to apply a single test pattern to the CUT requires that the entire scan path be loaded with new data. This problem can be eliminated by using the *random test data* (RTD) BIST architecture, shown in Figure 11.30, which has the following attributes:

- distributed and embedded BIST architecture;
- boundary scan.

In this architecture, signature analyzers are used for both compression and test pattern generation [Bardell *et al.* 1987].

The test process operates as follows. The registers R_1, R_2, and R_3 are first set to their scan mode so that they can be loaded with a seed pattern. R_1 cannot be loaded with the all-0 pattern. The registers are then put into their test mode and held in this state while the circuit is tested. For each clock cycle, R_1 and R_2 generate a new test pattern, and R_2 and R_3 operate as a MISR.

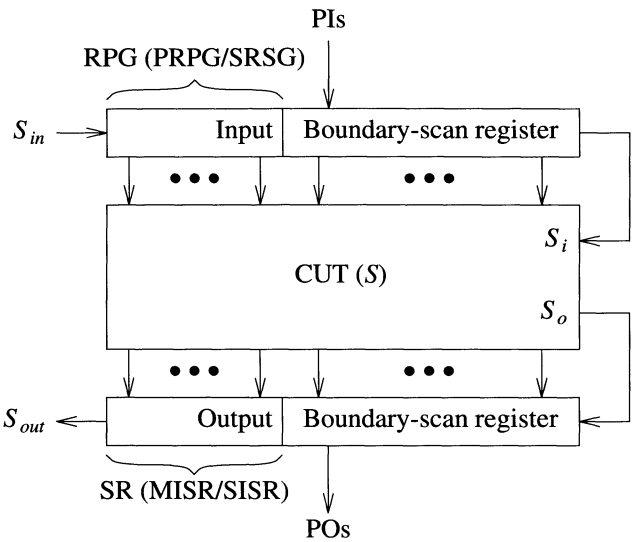


Figure 11.29 A centralized and embedded architecture with boundary scan (CEBS)

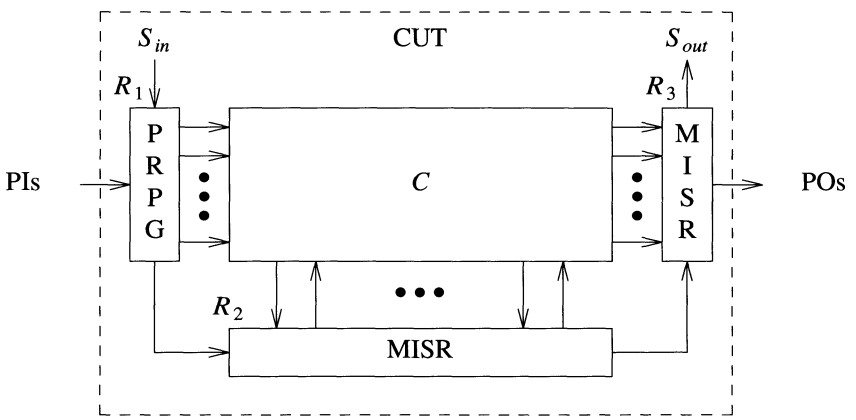


Figure 11.30 Random test data (RTD) BIST architecture

Little can be said for the tests generated by R_2 since these patterns are the state of a MISR. Clearly the patterns generated are a function of the logic C being tested. One problem with this technique is that some binary patterns may be repeated, and others may never be generated. Fault simulation is required to determine the number of clock cycles to run in the test mode to achieve a desired level of fault coverage.

Most of the previous BIST approaches use some form of LFSRs for generating test patterns and a signature. The RTD technique assumes that the normal data in a circuit, if suitably combined with some form of error capturing mechanism, is sufficient to test a circuit. This concept is the basis of the next three BIST architectures.

11.4.9 Simultaneous Self-Test (SST)

Simultaneous self-test (SST) is a BIST approach with the following attributes:

- distributed and embedded BIST architecture;
- scan CUT architecture;
- no boundary scan;
- no LFSRs used.

In this approach each storage cell in the CUT is modified to be a *self-test storage cell.* Such a storage cell has three modes of operation, namely normal mode, scan mode and test mode. A self-test SRL storage cell is shown in Figure 11.31. During normal operation, the value of the system data signal D is loaded into the flip-flop by *CK1*. To scan data into or out of the scan path, T is set to 0 and clock *CK2* is used. During self-test, T is set to 1 and again clock *CK2* is used. The new state of the flip-flop is equal to $D \oplus S_i$. A part of a circuit employing simultaneous self-test is shown in Figure 11.32. Note that the self-test scan register is not explicitly configured as an LFSR. Feedback can exist from the output to the input of this register by way of the combinational logic block C.

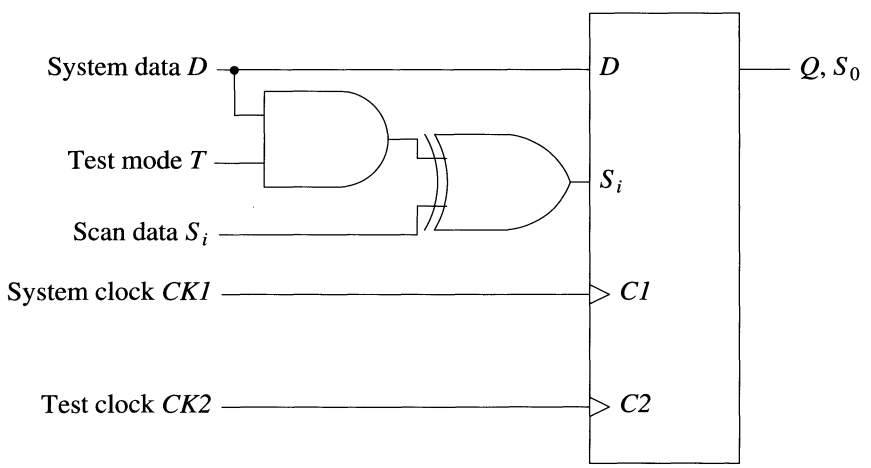


Figure 11.31 A self-test storage cell

To test a SST design, the scan path is first loaded with seed data. The circuit is then put into the test mode and clocked a sufficient number of times (N) to be adequately tested. After N clock cycles, the circuit is put back into the scan mode and the contents of the scan path shifted out. Faulty operation is detected if during the test process an output

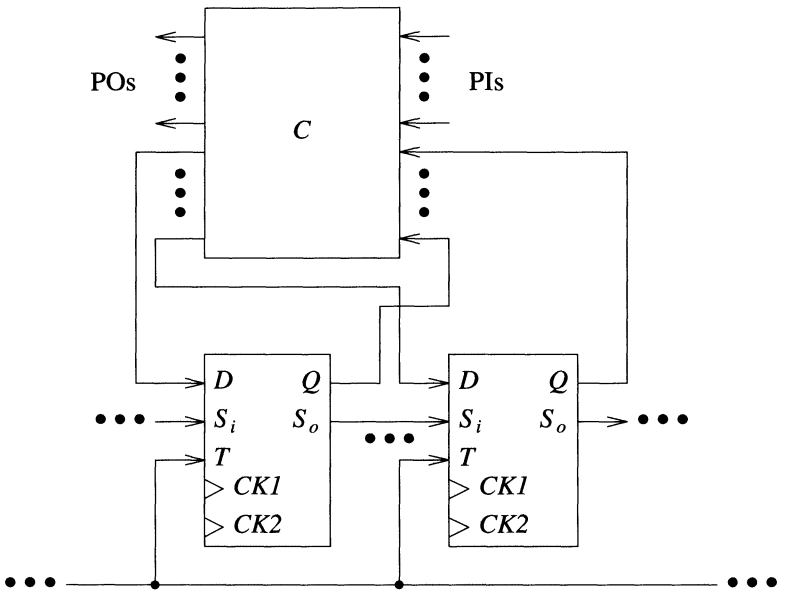


Figure 11.32 BIST architecture using self-test storage cells

error occurs, or if the final contents of the scan path is incorrect. During the test process the self-test scan path is simultaneously collecting test results from C and supplying test values to C. This procedure leads to a very fast test process since test data need not be shifted into the scan path.

Unfortunately, there are several problems associated with this test method. One problem again deals with testing external logic. For production testing, this can be solved by means of the ATE. For field test, some other means is required, such as the use of boundary-scan registers. Another solution is to connect the outputs of the CUT back to its inputs during self-test. In general, the number of POs and PIs are not the same. If there are more POs than PIs, then groups of POs can be XORed together to drive a single PI; if there are more PIs than POs, then some PIs can be driven by a unique PO, while other PIs are driven by the XOR of several POs. Another alternative is to let the ICs connected to the CUT drive (observe) inputs (outputs) which cannot be driven (observed) by the chip's own scan path.

Another problem deals with characterizing the quality of this test process. Note that though the scan path acts as both a random-pattern generator and as a data-compression circuit, there is no way of predicting *a priori* information related to the randomness of the test patterns generated or the degree of masking that may occur. Note that if an error is generated at the output of C (see Figure 11.32), then an error will occur in the scan path, say in cell Q_j. On the one hand, during the next clock cycle this error may propagate through C and produce several errors in the scan path. On the other hand, possibly no sensitized paths exist through C and this initial error just propagates to the next cell Q_{j+1}

in the scan path. However, if the data input D_{j+1} to this cell also is in error during this clock cycle, then the two errors *cancel* and masking occurs. Though errors frequently are canceled in the scan path, usually not all errors are canceled. For more information on simultaneous self-test see [DasGupta *et al.* 1982] and [Bardell and McAnney 1985].

11.4.10 Cyclic Analysis Testing System (CATS)

Figure 11.33 shows a BIST architecture referred to as *cyclic analysis testing systems* (CATS) [Burkness 1987]. The test aspects of this architecture are based on the fact that sequential circuits, by their very nature, are nonlinear binary-sequence generators. Figure 11.33(a) shows a sequential circuit that is the CUT. The CATS architecture is shown in Figure 11.33(b). Here during test mode, the outputs of S drive the inputs. If there are more outputs than inputs, then extra outputs can be combined using XOR gates. If the number of outputs is less than the number of inputs, then some outputs can drive more than one input. Thus all outputs and inputs can be exercised during testing. The analysis of this type of BIST architecture is complex. Different permutations of interconnections of outputs to inputs clearly lead to different test effectiveness. Also asynchronous feedback loops can be created, including ones having critical races. Hence care must be exercised in forming the feedback.

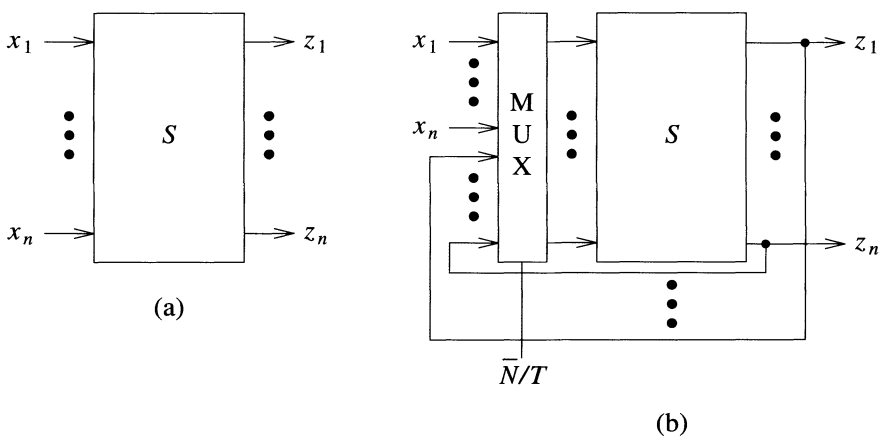


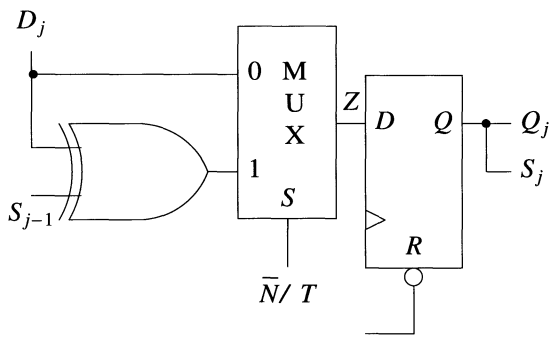
Figure 11.33 The CATS BIST architecture

To test the CUT, the outputs are connected to the inputs and the CUT clocked a predetermined number of times. For large circuits this can be tens of thousands of steps. The actual number is a function of the desired fault coverage and must be determined by fault simulation. The area overhead for this technique is low. Its effectiveness is clearly circuit dependent.

11.4.11 Circular Self-Test Path (CSTP)

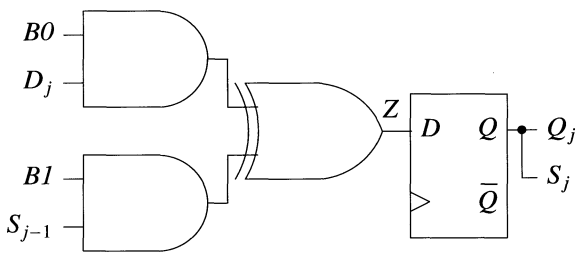
The *circular self-test path* (CSTP) BIST architecture is similar to the SST design. It differs primarily in three aspects. First, it employs a self-test cell design shown in Figure 11.34(a), rather than the one shown in Figure 11.31. Second, it is a register-based architecture; i.e., self-test cells are grouped into registers. Third, it employs partial

self-test; i.e., not all registers must consist of self-test cells. Some necessary features of this architecture are (1) all inputs and outputs must be associated with boundary scan cells and (2) all storage cells must be initializable to a known state before testing.



$\bar{N}/T$	Z	Mode
0	D_j	System
1	$D_j \oplus S_{j-1}$	Test

(a)



B0	B1	Z	Mode
0	0	0	Reset
0	1	S_j	Scan
1	0	D_j	System
1	1	$D_j \oplus S_{j-1}$	Test

(b)

Figure 11.34 Storage cell designs for use in circular self-test path BIST architectures

Consider the circuit shown in Figure 11.35. The registers R_1, R_2, R_3, R_7, and R_8 are part of the circular self-test path. Note that the self-test cells form a circular path. If this circular path contains m cells, then it corresponds to a MISR having the characteristic polynomial $(1 + x^m)$. Registers R_4, R_5, and R_6 need not be in the self-test path, nor do they require reset or set lines, since they can be initialized based on the state of the rest of the circuit. That is, once R_1, R_2, and R_3 are initialized, if R_4, R_5, and R_6 are issued two system clocks, then they too will be initialized to a known state.

The same cannot be said of R_3 because of the feedback loop formed by the path $R_3 - C_3 - R_6 - C_6 - R_3$. However, if R_3 has a reset line, then it need not be in the self-test path. Increasing the number of cells in the self-test path increases both the BIST hardware overhead and the fault coverage for a fixed test length.

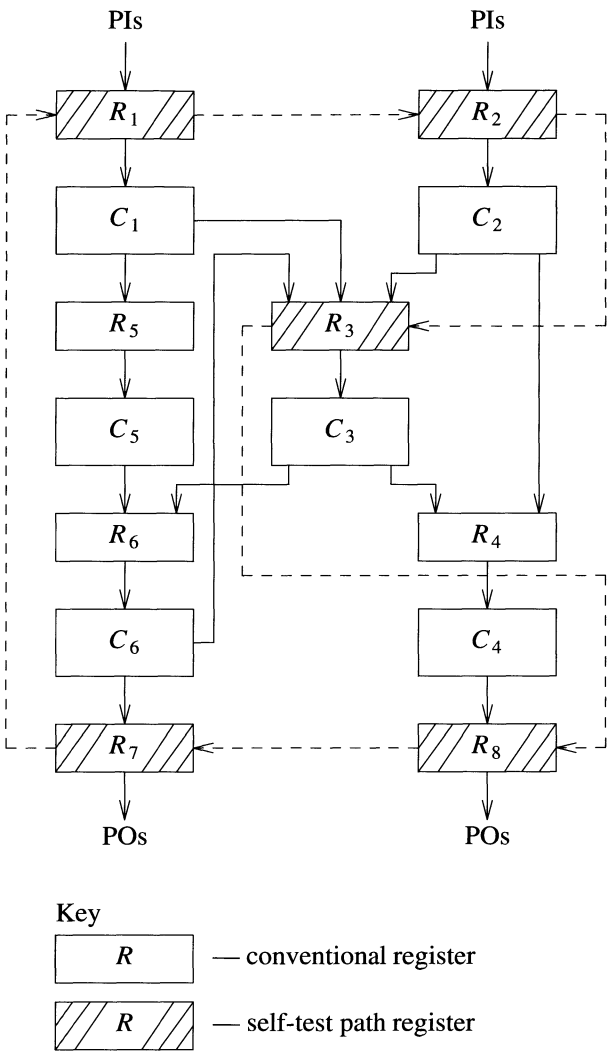


Figure 11.35 A design employing the circular self-test path architecture

The test process requires three phases.

1. *Initialization:* All registers are placed into a known state.
2. *Testing of CUT:* The circuit is run in the test mode; registers that are not in the self-test path operate in their normal mode.
3. *Response evaluation.*

During phase 2 the self-test path operates as both a random-pattern generator and response compactor. During phase 3 the circuit is again run in the test mode. But now

the sequences of outputs from one or more self-test cells are compared with precomputed fault-free values. This comparison can be done either on-chip or off-chip.

More information on this approach can be found in [Krasniewski and Pilarski 1989].

A similar BIST approach has been reported by Stroud [1988]. Here a signature analyzer is incorporated in the BIST circuitry to simplify identification of faulty circuit operation. Also a self-test cell having a scan mode is used, as shown in Figure 11.34(b).

Performance Analysis

Figure 11.36 shows the general form of a circular self-test path design. The circular path corresponds to an LFSR having the primitive polynomial $p(x) = 1+x^m$. In this section we will briefly present some theoretical and experimental results about certain performance aspects of this class of design. These results are applicable to many designs where a MISR is used as a PRPG.

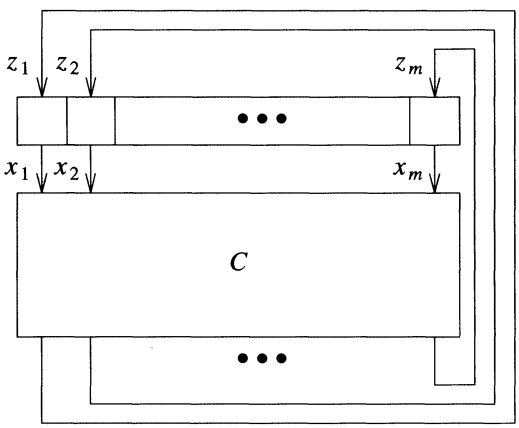


Figure 11.36 General form of a circular self-test path design

Let $z_i(t)$ and $x_i(t)$ be the input and output, respectively, to the ith cell in a circular self-test path. Then assume that the sequences of bits applied to each cell in the path are independent and that each sequence is characterized by a constant (in time) probability of a 1; e.g., for input z_i, $p_i = Prob\{z_i(t) = 1\}$, $t = 1,2,\ldots$.

Theorem 11.4: [Krasniewski and Pilarski 1989]. If there exists an input to the circular path, z_i, such that $0<p_i<1$, then, independent of the initial state of the path, $\lim_{t\to\infty} Prob\{x_j(t) = 1\} = 0.5$ for $j = 1, 2, \ldots, m$. □

Thus if the response of the circuit to the initial state of the circular path is neither the all-0 nor the all-1 pattern, then some time after initiatization the probability of a 1 at any bit position of the circular path is close to 0.5.

By carrying out extensive simulations, Krasniewski and Pilarski have made the following observations.

Observation 1: The number of clock cycles required for $x_j(t)$ to converge to 0.5 (the equiprobable steady state) is a function of the length of the circular path, and is usually small compared to the number of test patterns normally applied to the circuit.

The *pattern coverage* (also known as the *state coverage*) is denoted by $C_{n,r}$ and is defined as the fraction of all 2^n binary patterns occurring during r clock cycles of the self-testing process at n arbitrary selected outputs of the circular path. These outputs can be the n inputs to a block of logic C.

Observation 2: As the length of the circular path increases, the impact of the value of p_i on the pattern coverage decreases.

Observation 3: The circular path provides a block C with an almost exhaustive test for test lengths a few (four to eight) times longer than an exhaustive test.

Observation 4: When the number of clock cycles associated with a test exceeds the length of the circular path, the impact of the location of the n cells feeding the block C on the pattern coverage is negligible. For long test times, the pattern coverage associated with n cells is almost independent of the length of the path; it is close to the pattern coverage of an n-bit path.

More recently, Sastry and Majumdar [1989] have obtained the following theoretical results dealing with the distribution of pattern coverage in r cycles, as well as the distribution of r, the number of clock cycles required to obtain a pattern coverage of j (out of $N = 2^n$ of the possible inputs to an n-input circuit). In the results presented here we assume that all cells in the circular path are self-test scan cells.

$E[C_{n,r}]$ is the expected value of the random variable $C_{n,r}$. Let Y be a random variable representing the number of distinct patterns applied at time r, where we assume that at time 0 no patterns have yet been applied. Let $P_{j,r}$ denote the probability that $Y = j$, where $j \leq r$ and $j \leq 2^n$.

Definition 11.1: $\left\{ {z \atop k} \right\}$ are Stirling's numbers of the second kind, where $x^z = \sum_{k=0}^{z} \left\{ {z \atop k} \right\} \left[{x \atop k} \right] k!$ and $\left[{x \atop y} \right] = x!/y!(x-y)!$.

Theorem 11.5: Once the circular path has reached the steady (equiprobable) state, then

$$P_{j,r} = \frac{j! \left[{N \atop j} \right] \left\{ {r \atop j} \right\}}{N^r} \tag{11.1}$$

The computation of $P_{j,r}$ is complex. When r and N are both large such that $0 < \lambda = r/N < \infty$, we can approximate $P_{j,r}$ by

$$P_{j,r} = \left[{N \atop j} \right] (1-e^{-\lambda})^j e^{-\lambda(N-j)} \tag{11.2}$$

For the case where $j = r$, from (11.1) we obtain

$$P_{r,r} = \frac{N!}{(N-r)!N^r} = \prod_{k=0}^{r-1} \left[1 - \frac{k}{N} \right]$$

Let $\mu_{k,N,r}$ be the k-th moment of Y conditioned on N and r; i.e.,

$$\mu_{k,N,r} = E[Y^k|N,r].$$

Then

$$\mu_{k,N,r} = N\left[\mu_{k-1,N,r+1} - \left(1-\frac{1}{N}\right)^r \sum_{m=0}^{k-1} \binom{k-1}{m} \mu_{m,N-1,r}\right].$$

From this result we can determine the expected value and variance of $C_{n,r}$

$$E[Y|N,r] = \mu_{1,N,r} = N(1-(1-\frac{1}{N})^r)$$

hence

$$E[C_{n,r}] = (1-(1-\frac{1}{2^n})^r).$$

This same result was obtained by Kim *et al.* [1988].

The variance on the random variable Y can be expressed as $Var[Y] = E[Y^2] - E^2[Y]$. For Y conditioned on N and r we have

$$Var[Y|N,r] = N\left[\left(\frac{N-1}{N}\right)^r + (N-1)\left(\frac{N-2}{N}\right)^r - N\left(\frac{N-1}{N}\right)^{2r}\right]$$

Thus the variance of Y approaches zero as r approaches infinity, and therefore the accuracy of $E[Y]$ as an estimate of the actual pattern coverage increases as r increases.

Let R be the random variable representing the number of clock cycles required to obtain a pattern coverage of j (out of N). Then the probability of requiring r clock cycles to achieve the pattern coverage of j ($r \geq j$) is given by

$$P\{R=r|N,j\} = \frac{(N-j+1)}{N} P_{j-i,r-1}$$

where $P_{j,r}$ is given in equation (11.1). Then

$$E[R|N,j] = j + \sum_{k=1}^{j-1} \frac{k/N}{(1-k/N)}$$

11.4.12 Built-In Logic-Block Observation (BILBO)

One major problem with several of the designs presented previously is that they deal with an unpartitioned version of a CUT; i.e., all primary inputs are grouped together into one set, all primary outputs into a second set, and all storage cells into a third set. These sets are then associated with PRPGs and MISRs. Since the number of cells in these registers is usually large, it is not feasible to consider exhaustive or pseudoexhaustive test techniques. For example, a chip can easily have over 100 inputs and several hundred storage cells. To circumvent this problem one can attempt to cluster storage cells into groups, commonly called registers. In general these groups correspond to the functional registers found in many designs, such as the program counter, the instruction register, and the accumulator found in a microprocessor design. Some BIST architectures take

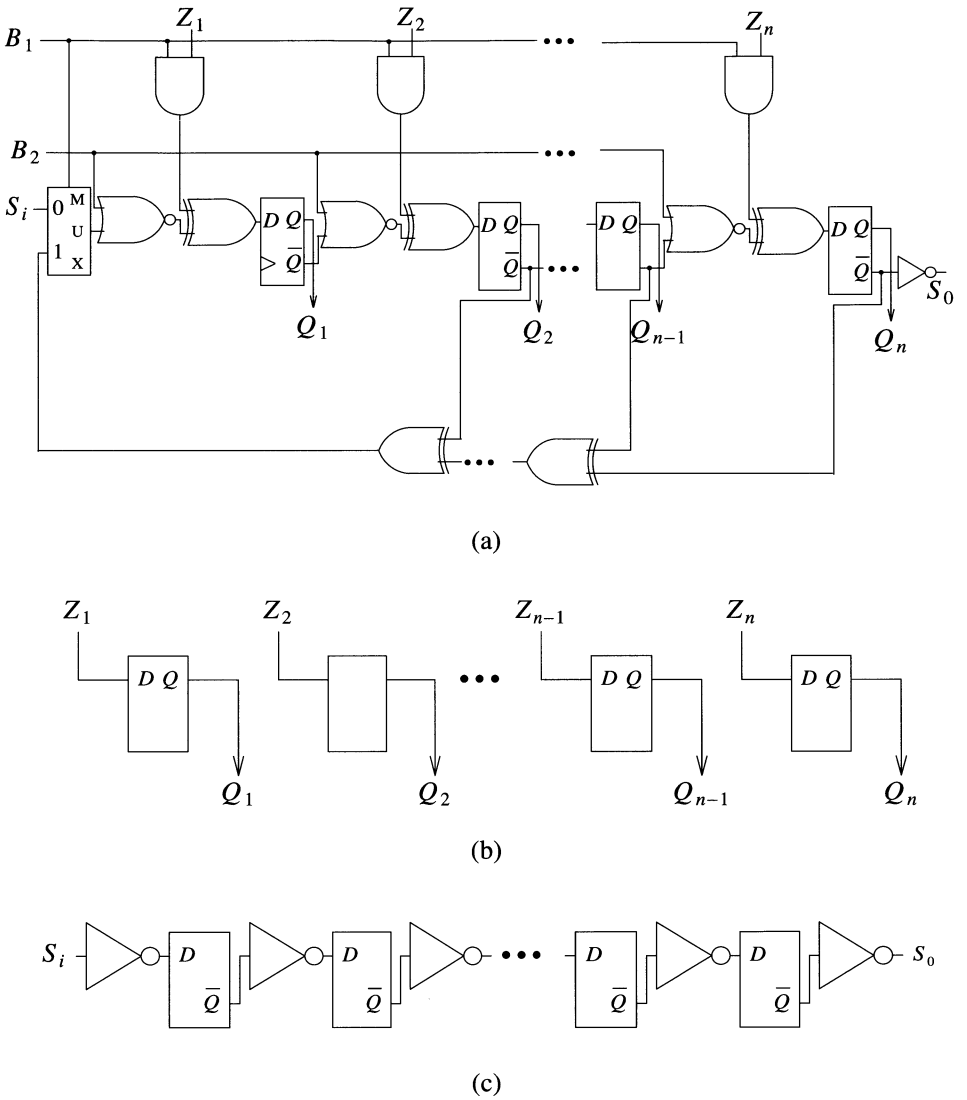
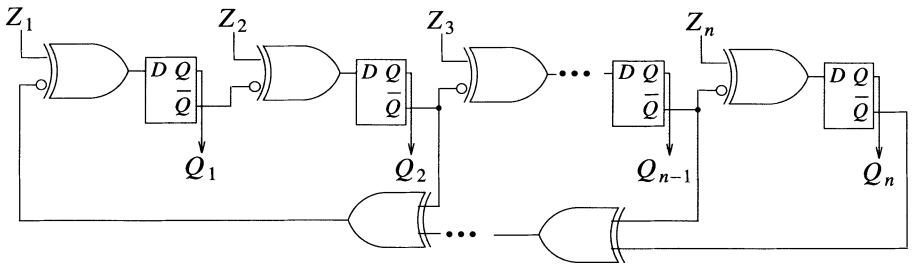


Figure 11.37 (a) n-bit BILBO register (b) normal mode ($B_1 = B_2 = 1$) (c) shift register mode ($B_1 = B_2 = 0$) (d) LFSR (test) mode ($B_1 = 1, B_2 = 0$)

advantage of the register aspects of many designs to achieve a more effective test methodology.

One such architecture employs *built-in logic-block observation* (BILBO) registers [Koenemann *et al.* 1979, 1980], shown in Figure 11.37(a). In this register design the inverted output $\overline{Q}$ of a storage cell is connected via a NOR and an XOR gate to the data input of the next cell. A BILBO register operates in one of four modes, as specified by



(d)

Figure 11.37 (Continued)

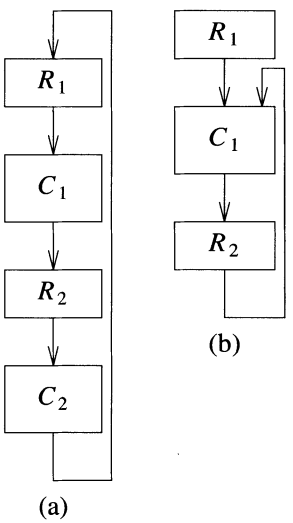
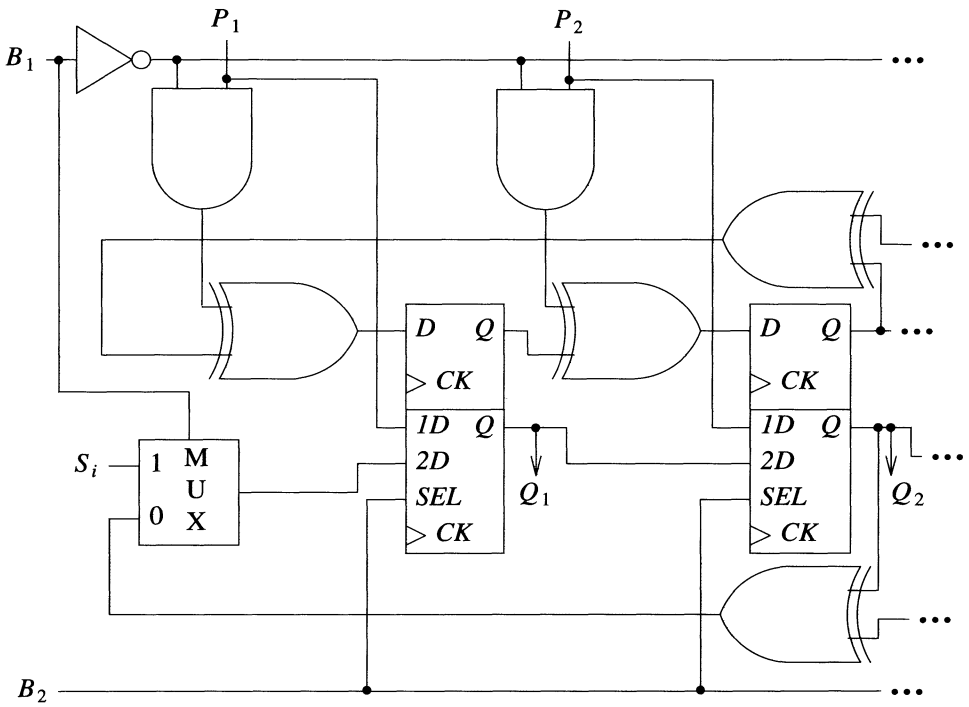


Figure 11.38 BIST designs with BILBO registers

the control inputs B_1 and B_2. When $B_1 = B_2 = 1$, the BILBO register operates in its normal parallel load mode (see Figure 11.37(b)). When $B_1 = B_2 = 0$, it operates as a shift register with scan input S_i (see Figure 11.37(c)). Note that data is complemented as it enters the scan register. When $B_1 = 0$ and $B_2 = 1$, all storage cells are reset. When $B_1 = 1$ and $B_2 = 0$, the BILBO register is configured as an LSFR (see Figure 11.37(d)), or more accurately the register operates as a MISR. If the Z_is are the outputs of a CUT, then the register compresses the response to form a signature. If the inputs $Z_1, Z_2, ..., Z_n$ are held at a constant value of 0, and the initial value of the register is not all 0s, then the LFSR operates as a pseudorandom-pattern generator.

A simple form of a BILBO BIST architecture consists of partitioning a circuit into a set of registers and blocks of combinational logic, where the normal registers are replaced by BILBO registers. In addition, the inputs to a block of logic C are driven by a BILBO register R_i, and the outputs of C drive another BILBO register R_j.



B_1	B_2	Operation mode
—	0	Normal
1	1	Scan
0	1	PRPG/MISR

Figure 11.39 Concurrent BILBO register

Consider the circuit shown in Figure 11.38(a), where the registers are all BILBOs. To test C_1, first R_1 and R_2 are seeded, and then R_1 is put into the PRPG mode and R_2 into the MISR mode. Assume the inputs of R_1 are held at the value 0. The circuit is then run in this mode for N clock cycles. If the number of inputs of C_1 is not too large, C_1 can even be tested exhaustively, except for the all-zero pattern. At the end of this test process, called a *test session*, the contents of R_2 can be scanned out and the signature checked. Similarly C_2 can be tested by configuring R_1 to be a MISR and R_2 to be a PRPG. Thus the circuit is tested in two test sessions.

Figure 11.38(b) shows a different type of circuit configuration, one having a self-loop around the R_2 BILBO register. This design does not conform to a normal BILBO architecture. To test C_1, R_1 must be in the PRPG mode. Also, R_2 should be in both the MISR mode and the PRPG mode. This is not possible for the design shown in Figure 11.37(a). What can be done is to place R_2 in the MISR mode. Now its outputs are essentially random vectors that can be used as test data to C_1. One feature of this scheme is that errors in the MISR produce "erroneous" test patterns that are applied to C_1, which in turn tend to produce more errors in R_2. The bad aspect of this approach is that there may exist faults that are never detected. This could occur, for example, if the input data to C_1 never propagate the effect of a fault to the output of C_1.

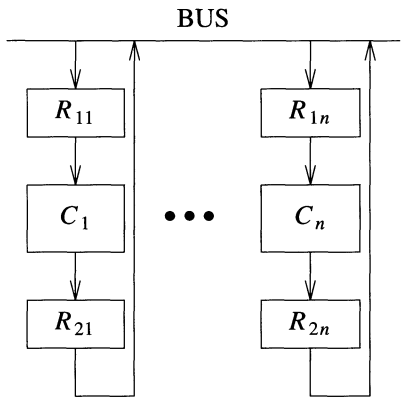


Figure 11.40 Bus-oriented BILBO architecture

This situation can be rectified by using a *concurrent built-in logic-block observation* (CBILBO) register [Wang and McCluskey 1986c]. This register, shown in Figure 11.39, operates simultaneously as a MISR and a PRPG. The top row of D flip-flops and associated logic form a MISR. The bottom row of dual-port flip-flops and associated logic form a register that can operate either in the normal parallel-load mode or the scan mode or as a PRPG.

Recall that when a BILBO register is in the PRPG mode, its inputs need to be held at some constant value. This can be achieved in several ways. Often the BILBO test methodology is applied to a modular and bus-oriented system in which functional modules, such as ALUs, RAMs, and ROMs, are connected via a register to a bus (see Figure 11.40). By disabling all bus drivers and using pull-up or pull-down circuitry, the register inputs can be held in a constant state.

However, some architectures have a pipeline structure as in Figure 11.41. To deactivate the inputs to a BILBO register during its PRPG mode, a modified BILBO register design having three control signals (B_1,B_2,B_3) can be used, as shown in Figure 11.42. There are now eight possible control states, and one can be used to specify the MISR mode and another the PRPG mode.

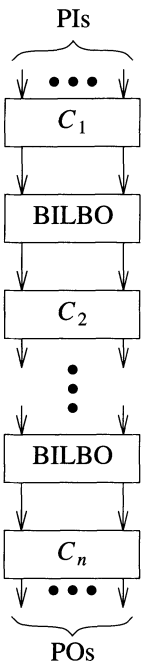


Figure 11.41 Pipeline-oriented BILBO architecture

One aspect that differentiates the BILBO architecture from the previously discussed BIST architectures is the partitioning of storage cells to form registers and the partitioning of the combinational logic into blocks of logic. Once such a partitioning is achieved, then many of the techniques discussed previously dealing with pseudoexhaustive testing can be employed. The BILBO registers can be replaced by other types of registers, such as constant-weight counters or more complex forms of LFSRs.

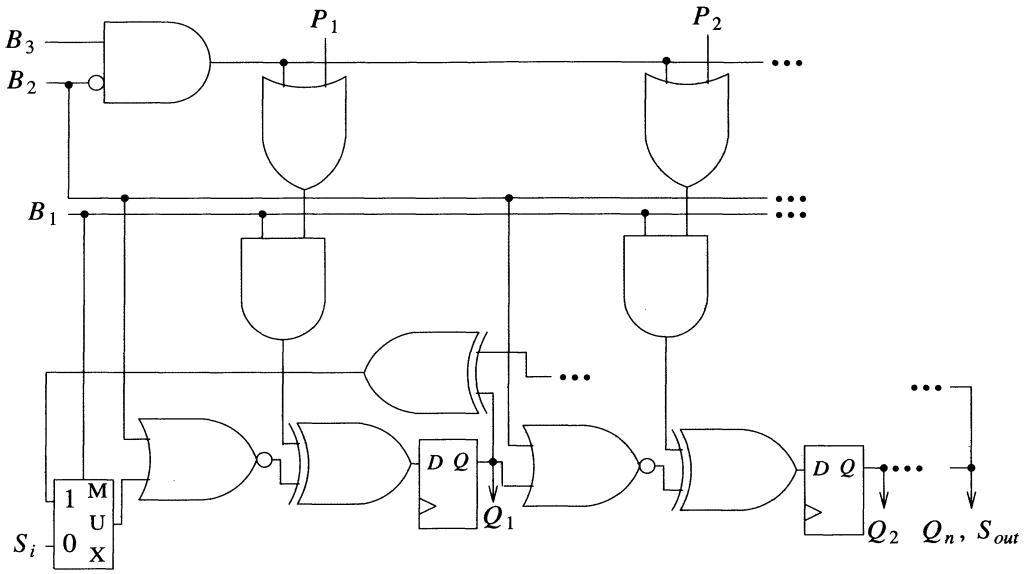
The *built-in digital-circuit observer* (BIDCO) is an extension of the BILBO design approach to board-level testing [Fasang 1980]. Related work can be found in [Beucler and Manner 1984].

Several techniques, such as RTD, CSTP, and BILBO, use a MISR as a PRPG. Previously we have presented some theoretical results pertaining to the effectiveness of using a circular self-test path as a pattern generator. We next present some additional results related to the use of a MISR for test-pattern generation.

Analysis of Patterns Generated by a MISR

Background

In this section we will analyze some of the properties of patterns generated by a MISR. The results of this analysis are useful when a MISR is used as a TPG. Consider an n-bit MISR whose characteristic polynomial is primitive. Then if the input is held constant, the MISR behaves as a maximum-length LFSR and generates $2^n - 1$ patterns. However,



B_1	B_2	B_3	Operation mode
1	1	0	Normal
0	1	0	Reset
1	0	0	Signature analysis (MISR)
1	0	1	Pattern generation (PRPG)
0	0	0	Scan

Figure 11.42 BILBO register with input disable control

even if a primitive polynomial is not employed, all states can still be generated by changing the inputs to the MISR.

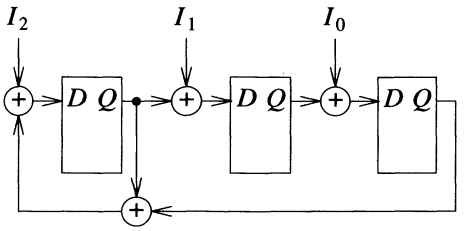
Figure 11.43 shows a 3-bit MISR and the state transitions that occur under two different inputs.

The predecessor state of state 101 under input pattern 010 is 110; its successor state is 000. Because of the linear nature of LFSRs, the following results can be shown [Golomb 1982].

Property 1: The number of successor states of a given state of a MISR is equal to the number of unique input patterns of the MISR.

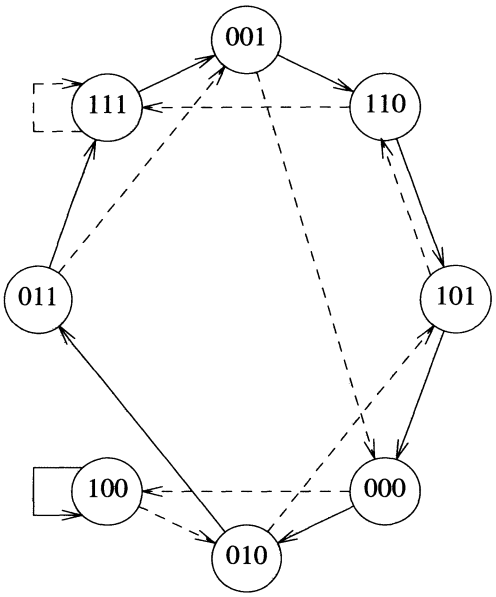
Property 2: The number of predecessor states of a given state of a MISR is equal to the number of unique input patterns of the MISR.

Property 3: If the number of input patterns of an n-bit MISR is greater than 1, the MISR can possibly generate all 2^n patterns.



$P(x) = x^3 + x^2 + 1$

(a)



Input pattern

$\longrightarrow$ $I_2I_1I_0 = 010$

$- \rightarrow$ $I_2I_1I_0 = 100$

(b)

Figure 11.43 (a) A 3-bit MISR (b) State transition diagram for two different inputs

Completeness of State Space

The behavior of a MISR can be modeled by a *Markov chain*. Let p_{ij} be the probability of a state transition from state s_i to state s_j. Then the state-transition diagram of an n-bit MISR can be described by a $2^n \times 2^n$ matrix, called the *transition-probability matrix*, and denoted by $P = [p_{ij}]$, $i, j = 1, 2, ..., 2^n$.

For the example shown in Figure 11.43, let the probability of occurrence of input pattern 010 be 1/3 and that of 100 be 2/3. The transition-probability matrix P is shown in Figure 11.44.

Let π_j^k be the probability that a MISR is in state s_j after the application of k input patterns. The state-probability vector is denoted by $\pi(k) = (\pi_1^k, \pi_2^k, ..., \pi_N^k)$ where $N = 2^n$. The $\pi(k)$ is related to the initial-probability vector $\pi(0)$ and P by the equation $\pi(k) = \pi(0)P^k$.

$$P = \begin{bmatrix} 0 & 0 & 1/3 & 0 & 2/3 & 0 & 0 & 0 \\ 2/3 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 2/3 & 0 & 0 \\ 0 & 2/3 & 0 & 0 & 0 & 0 & 0 & 1/3 \\ 0 & 0 & 2/3 & 0 & 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 2/3 & 0 & 1/3 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 2/3 \end{bmatrix}$$

Figure 11.44

For example, several of the state-probability vectors for the circuit of Figure 11.43, given the initial state (111) and the P matrix given above are listed below:

$$\begin{aligned} \pi(0) &= (00000001) \\ \pi(1) &= \pi(0)\,P = (0\ 1/3\ 00000\ 2/3) \\ \pi(2) &= \pi(1)\,P = \pi(0)\,P^2 = (0.22\ 0.22\ 0\ 0\ 0\ 0\ 0.11\ 0.44) \\ &\vdots \\ \pi(7) &= (0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1358) \\ \pi(8) &= (0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1235\ 0.1317) \end{aligned}$$

Note that as k increases, each element of $\pi(k)$ approaches 1/8. We will show that in general as k increases each state becomes equally probable.

A Markov chain is said to be *regular* if every state can be reached from every other state. A transition-probability matrix P corresponding to a Markov chain is *doubly stochastic* if each row and column sum is 1.

A Markov process having these properties satisfies the following theorem [Taylor and Karlin 1984]:

Theorem 11.6: If a Markov chain is regular and the transition-probability matrix P is doubly stochastic, then $\lim_{k\to\infty} \pi(k) = (1/N, 1/N, ..., 1/N)$, where N is the number of states.

□

From properties 1, 2, and 3, the state diagram of a MISR is regular if the number of input patterns is greater than 1. Also, the transition-probability matrix P associated with a MISR is doubly stochastic. Based on the law of large numbers [Taylor and Kalin 1984] and Theorem 11.6, we have the following result:

Theorem 11.7: For a given n-bit MISR the probability of appearance of each pattern becomes $1/2^n$ after a sufficient number of clock cycles, provided that the number of different input patterns to the MISR is greater than 1. □

Let x_i be the output of one the i-th bit of the MISR, and let $p_i = Prob\{x_i = 1\}$. Then from Theorem 11.7 it follows that $p_i = 0.5$.

Effectiveness as a PRPG

In this section we will show that a MISR can be effectively used as a PRPG. Let $\{v_1, v_2, \ldots, v_N\}$ be the set of all binary n-tuples. Each v_i represents a state of an n-bit MISR. Assume a MISR operates for m clock cycles and generates a sequence of patterns $S = S_1, S_2, \ldots, S_m$. Let D_m be the number of distinct patterns in this sequence. Let

$$x_{i,m} = \begin{cases} 1 & \text{if } v_i \text{ occurs in the sequence } S \\ 0 & \text{otherwise.} \end{cases}$$

Then $D_m = x_{1,m} + x_{2,m} + \ldots + x_{N,m}$, and the expected value of D_m is

$$E[D_m] = E\left[\sum_{i=1}^{N} x_{i,m}\right] = \sum_{i=1}^{N} E(x_{i,m}).$$

Since $Prob(x_{i,m}) = 1-(1-\frac{1}{2^n})^m$, then $E[D_m] = N(1-(1-\frac{1}{2^n})^m)$.

When $m << 2^n$, then

$$E[D_m] = 2^n(1-(1-\frac{1}{2^n})^m) \cong 2^n(1-(1-\frac{m}{2^n})) = m.$$

Thus for a small number of patterns, the chances of generating repeated patterns is small.

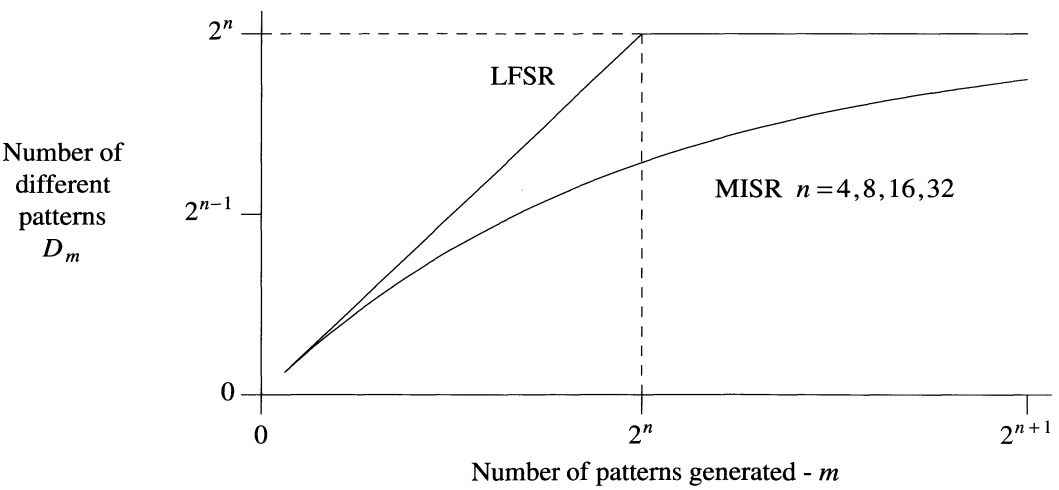


Figure 11.45

Some results relating the value of D_m to m for various values of n are shown in Figure 11.45. Since the results for $n = 4$, 8, 16, and 32 are similar, only one curve is shown. It is seen that for $m << 2^n$, the MISR acts as a PRPG; but for larger values of m it

acts more as a random-pattern generator. Kim *et al.* [1988] have also shown that the properties pertaining to the randomness of the patterns generated by a MISR are true even if the inputs are nonequiprobable.

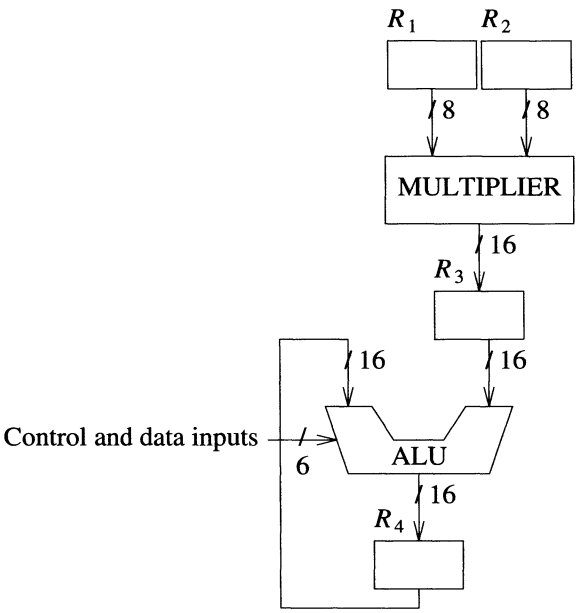


Figure 11.46 Portion of the TMS32010 signal-processing chip

11.4.12.1 Case Study

Figure 11.46 shows a portion of the TMS32010 signal-processing chip. Various BIST architectures were applied to this chip, and the results derived by Kim *et al.* [1988] are presented here. The 8×8-bit multiplier and 16-bit ALU were modeled using 880 and 354 gates respectively. The numbers of single stuck-fault classes in these circuits are 1255 and 574 respectively.

Four BIST designs are shown in Figure 11.47.

Design 1: Figure 11.47(a) shows a BILBO-type BIST architecture that requires two test sessions, one to test the multiplier, at which time the BILBO register operates as a MISR; the second to test the ALU, at which time the BILBO register operates as a PRPG. This design requires the addition of a 16-bit and a 6-bit PRPG to drive the leftmost 16-bits of the ALU, and the five control inputs and carry-in of the ALU.

Design 2: Figure 11.47(b) shows a BIST design where R_3 is unmodified and again a 22-bit PRPG is added to the circuit. The response data produced by the multiplier are used as test data to the ALU. Now only one test session is required.

Design 3: Figure 11.47(c) shows a BIST design where R_3 is extended to be a 38-bit MISR. The inputs to the first 22-bits of this MISR are held constant at 0. Again only one test session is required.

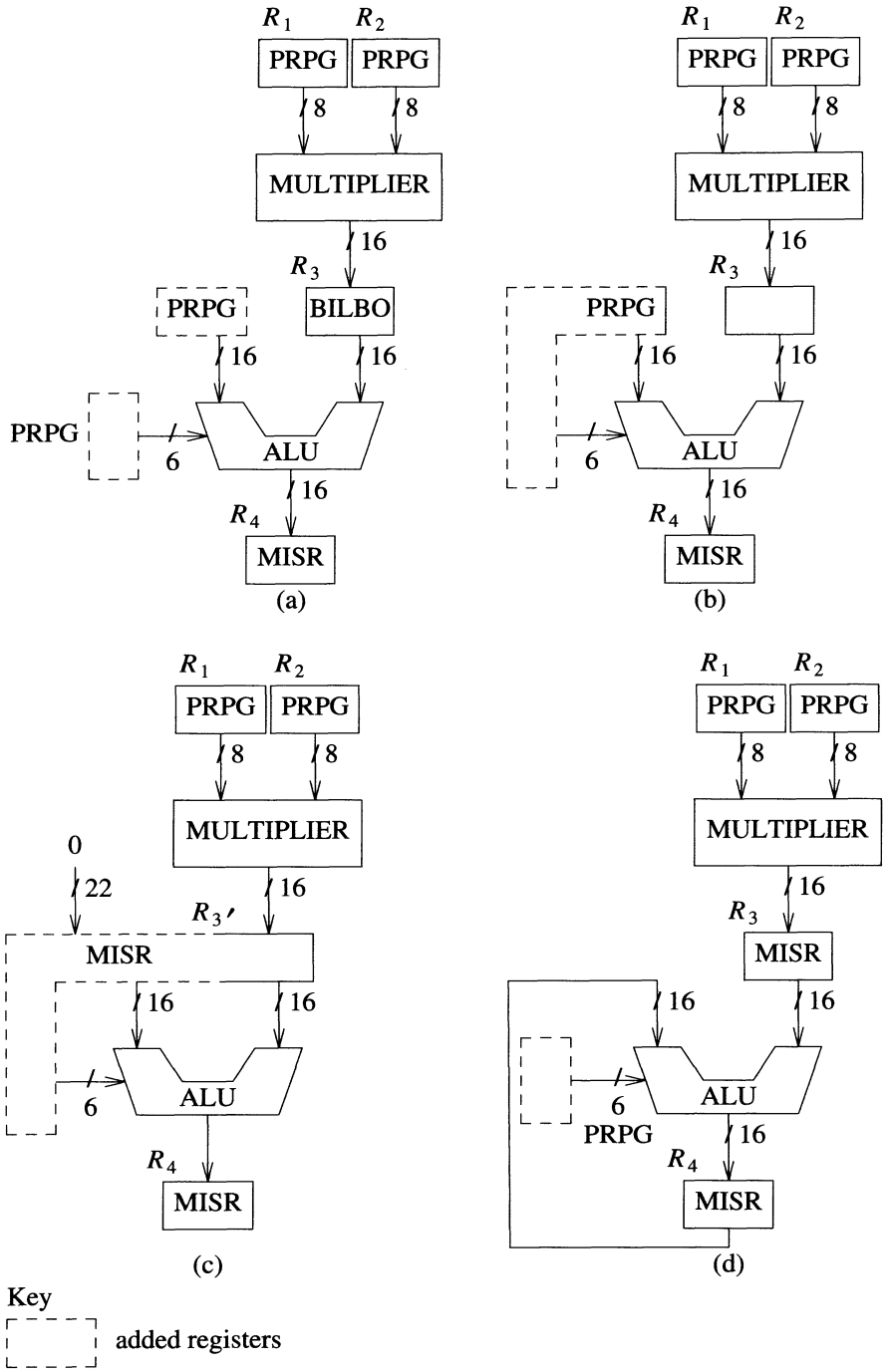


Figure 11.47 Four testable designs for the TMS32010
(a) BILBO solution (b) Solution where R_3 remains unmodified (c) Solution where ALU is tested by separate MISRs (d) Solution where ALU is tested by sharing a MISR

Design 4: Figure 11.47(d) shows still another BIST design. Here R_3 and R_4 are MISRs and are used to drive the data inputs to the ALU. Of the four designs, this one uses the least amount of test hardware. Again only one test session is required.

Simulation results based on 20 different runs are shown in Figure 11.48, where each run corresponds to a different selection of seed values in the LFSRs.

Number of test patterns	Design 1	2	3	4
Average	2,177	≥ 3,000	1,457	1378
Minimum	830	—	634	721
Maximum	3,619	—	2,531	2,121
Fault coverage (%)	100	64.5	100	100

Figure 11.48 Fault-simulation results for various BIST designs

For designs 1, 3, and 4, simulation terminated as soon as 100 percent fault coverage was achieved. For design 2, fault coverage saturated at 64.5 percent. The number of test patterns for BILBO design 1 is the sum of the test patterns for each of the two test sessions. Note that designs 3 and 4 lead to a substantial reduction in test length. For design 4, the input patterns to the ALU are not necessarily uniformly distributed since they are dependent on the current state of the MISR R_3; i.e., the probability-transition matrix P is not doubly stochastic.

In summary, it appears that a MISR can be used effectively as a source of test patterns. Also, it is usually not necessary to test circuits exhaustively or even to apply a verification test.

11.4.13 Summary

BIST represents an active research area, and new BIST architectures are continually being developed. In this section we have shown how some of these architectures have evolved. Initially, applying techniques such as CSBL and BEST, sequential circuits were tested using pseudorandom test patterns, and results compressed using signature analyzers. To obtain better access to internal circuitry, scan paths can be used so that the kernel being tested is combinational. The next step was to embed the PRPG and MISR into the CUT and employ boundary scan to test isolated logic. To reduce test time, approaches such as SST and CSTP were developed where a scan path does not have to be serially loaded before the application of each test pattern. Here a more complex pseudo-scan-path design is used. Finally, a register-based BIST architecture (BILBO) was presented. Note that for some BIST approaches, such as CSBL, BEST, and CSSP, the circuit under test operates as a sequential circuit during the test process; hence some aspects of dynamic testing are present. For those approaches that use a scan operation between test patterns, such as RTS, LOCST, and STUMPS, testing is more of a static operation.

One problem associated with several BIST designs such as RTD, SST, and CSTP, is that during self-test, the D inputs to the storage cells are not tested. Hence some additional tests need to be run while in the normal mode. This can be done by loading the scan path with a test pattern, applying a normal clock, and then checking the contents of the scan path. Usually a few patterns are enough to test the logic not tested by the self-test process.

Figure 11.49 shows some of the main attributes of the architectures presented.

Architecture (section)	Centralized (C) or distributed (D)	Separated (S) or embedded (E)	Combinational (C) or sequential (S) kernels	Boundary scan	Chip (C) or board (B) level	
CSBL (11.4.1)	C	S	C or S	N	B	
BEST (11.4.2)	C	S or E	C or S	Y	C	
RTS (11.4.3)	D	S	C	N	C	(1)
LOCST (11.4.4)	C	S	C	Y	C	
STUMPS (11.4.5)	C	S	C	N	B or C	
CBIST (11.4.6)	C or D	S	C	Optional	C	(2)
CEBS (11.4.7)	C	E	C	Y	C	
RTD (11.4.8)	D	E	C	Y	C	
SST (11.4.9)	D	E	C	N	C	
CATS (11.4.10)	NA	NA	S	N	C	(3)
CSTP (11.4.11)	D	E	C or S	Y	C	
BILBO (11.4.12)	D	E	C	N	C	(4)

(1) A non-BIST architecture
(2) A concurrent BIST architecture
(3) In a minimal configuration there is no BIST hardware except for MUXs
(4) Can be extended to sequential kernels

Figure 11.49 Summary of BIST architecture

11.5 Some Advanced BIST Concepts

In this section we will consider three advanced concepts related to some BIST architectures: (1) scheduling of test sessions, (2) control of multiple test sessions when a distributed and embedded BIST architecture is employed, and (3) partial BIST designs. For simplicity we will assume a BILBO architecture is employed. First, some basic concepts will be presented.

A register R_i is said to be a *driver* of a block of logic C if some outputs of R_i are inputs to C. A register R_j is said to be a *receiver* of C if some outputs of C are inputs to R_j. R_i is said to be *adjacent* to R_j if there exists a block of logic C such that R_i is a driver of C, and R_j is a receiver of C. If R_i is both a receiver and a driver of C, then it is *self-adjacent.*

Example 11.5: Consider the portion of the circuit shown in Figure 11.50(a). R_1, R_3, and R_4 are drivers of C, R_2 and R_3 are receivers of C, and R_3 is a self-adjacent register. Also R_1 and R_2 are adjacent, but R_1 and R_4 are not adjacent.

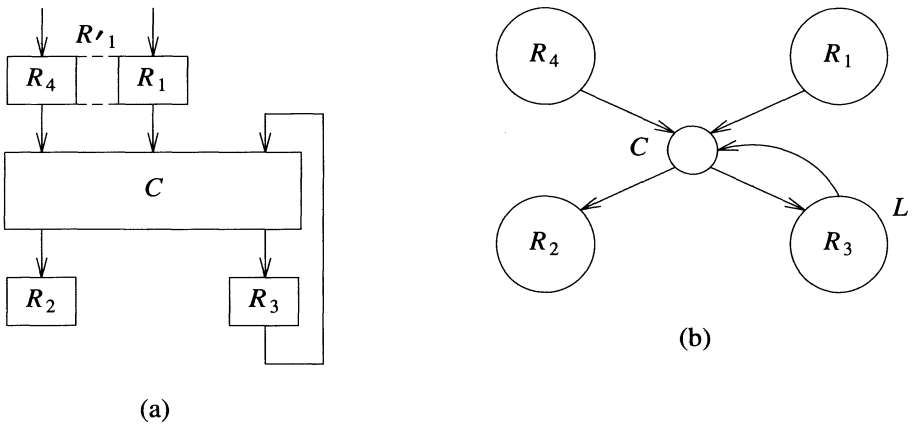


Figure 11.50 (a) Part of a circuit (b) Its register adjacency graph

To test C, R_1 and R_4 should be placed in the PRPG mode, and R_2 in the MISR mode. R_3 should be in both the PRPG mode and the MISR mode. This can be accomplished by using a CBILBO register, such as the one shown in Figure 11.39. To reduce hardware overhead, we will operate self-adjacent registers in the MISR mode. Note that storage cells and functional registers can be clustered into new registers in order to form a BILBO register. For example, it may be feasible to cluster R_1 and R_4 into a single LFSR R'_1 during the test mode. However, it would not be beneficial to cluster R_2 and R_4 together since they operate in different ways during the test mode. □

11.5.1 Test Schedules

A *test session* is defined as an assignment of test modes to BILBO registers to test one or more blocks of logic. A block of logic is considered to be tested if its driver registers are in the PRPG mode, and its receivers and self-adjacent registers are in the MISR mode.

A circuit can be modeled by a bipartite graph $G = (N_A, N_B, E)$, referred to as a *register adjacency graph* (RAG), where N_A is a set of type A nodes, each of which represents a register, N_B is a set of type B nodes, each of which represents a block of combinational logic, and E is a set of edges between type A and B nodes. A directed edge exists from a type A node (register R_i) to a type B node (block of logic C_k) if R_i is a driver of C_k, and a directed edge exists from a type B node C_k to a type A node R_j if R_j is a receiver of C_k. In addition, a type A node corresponding to a self-adjacent register is flagged with an L. Figure 11.50(b) shows the RAG associated with Figure 11.50(a).

The *test-scheduling* problem is defined as follows: determine the minimal number of test sessions required to test all blocks of combinational logic. This problem is related to determining the *chromatic number* of a graph G^*, which is the minimal number of colors

that can be assigned to the nodes of G^* such that no edge connects two nodes of the same color. Methods for determining the minimal number of test sessions have been presented in [Craig *et al.* 1988].

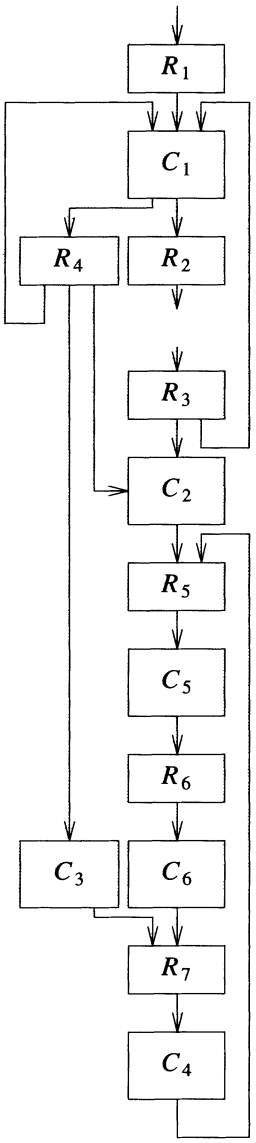


Figure 11.51 Example circuit

Example 11.6: Figure 11.51 shows a circuit to be made testable using the BILBO methodology. Figure 11.52 shows its corresponding RAG. The circuit can be completely tested with just three test sessions, as shown in Figure 11.53. Figure 11.53(a) shows a test session where C_1 and C_5 are tested simultaneously.

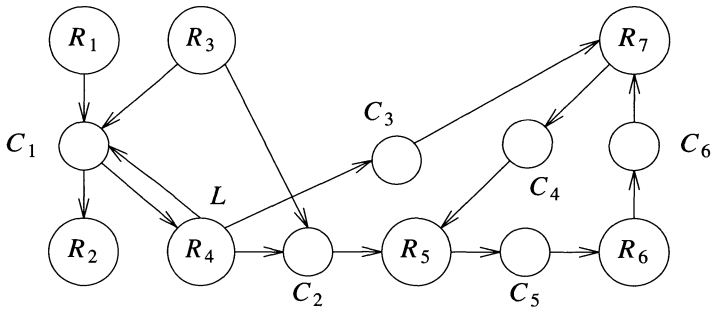


Figure 11.52 Register adjacency graph for the circuit in Figure 11.51

The test-scheduling problem becomes more complex when the test time for each block of logic is considered. Now the objective is to minimize the test time for the entire chip. A test session can now be more complex; for example, while C_i is being tested, C_j is also tested, and when the testing of C_j is completed, C_k is then tested. The analysis of these time-dependent test schedules is considered in [Jone *et al.* 1989].

11.5.2 Control of BILBO Registers

When multiple test sessions exist, the efficient control of the BILBO registers becomes an important issue. Each BILBO register requires three bits of control information to specify in which of five possible modes to operate, namely, normal, scan, reset, PRPG, or MISR. It would be desirable if these control bits were common to all registers and could then be broadcast using three common control signals. However, since in each test session the role of a register can be either a MISR or a PRPG, it is only possible to broadcast two signals. These two signals can be used to specify one of four modes, namely, normal, scan, reset, or test. Each register can then have one unique control signal to indicate if it should operate as a PRPG or a MISR while in the test mode. The distribution and control of these individual signals could require a significant area overhead. An alternative approach is to distribute the control. This can be done by associating with each BILBO register R_i a test storage cell T_i. If $T_i = 1$, then R_i operates in the MISR mode when in the test mode; otherwise it operates in the PRPG mode. The test storage cells can be organized into a separate scan path or made part of the same scan path that incorporates the BILBO registers. Before executing a test session the test cells can be initialized so that each BILBO register operates in its appropriate mode. Figure 11.54 shows a BILBO register design that incorporates a test mode storage cell [Hudson and Peterson 1987]. A basic cell in the BILBO register is shown in Figure 11.54(a). The first cell of the BILBO register and the control lines to all the cells are driven by the logic shown in Figure 11.54(b). Figure 11.54(c) shows the truth table for the control circuitry C_1. The test mode storage cell T_i is in the same scan path as the BILBO registers. The output of T_i is $T2$, which is an input to C_1. The signal $S*$ drives the input S_0 to the first cell in the BILBO register corresponding to $j = 1$. S_i is the normal scan-in input used to scan data into the BILBO register; FB is the feedback signal from the linear feedback network. Setting $T0 = 1$ disables the clock to T_i, allowing T_i to hold its state during the test mode.

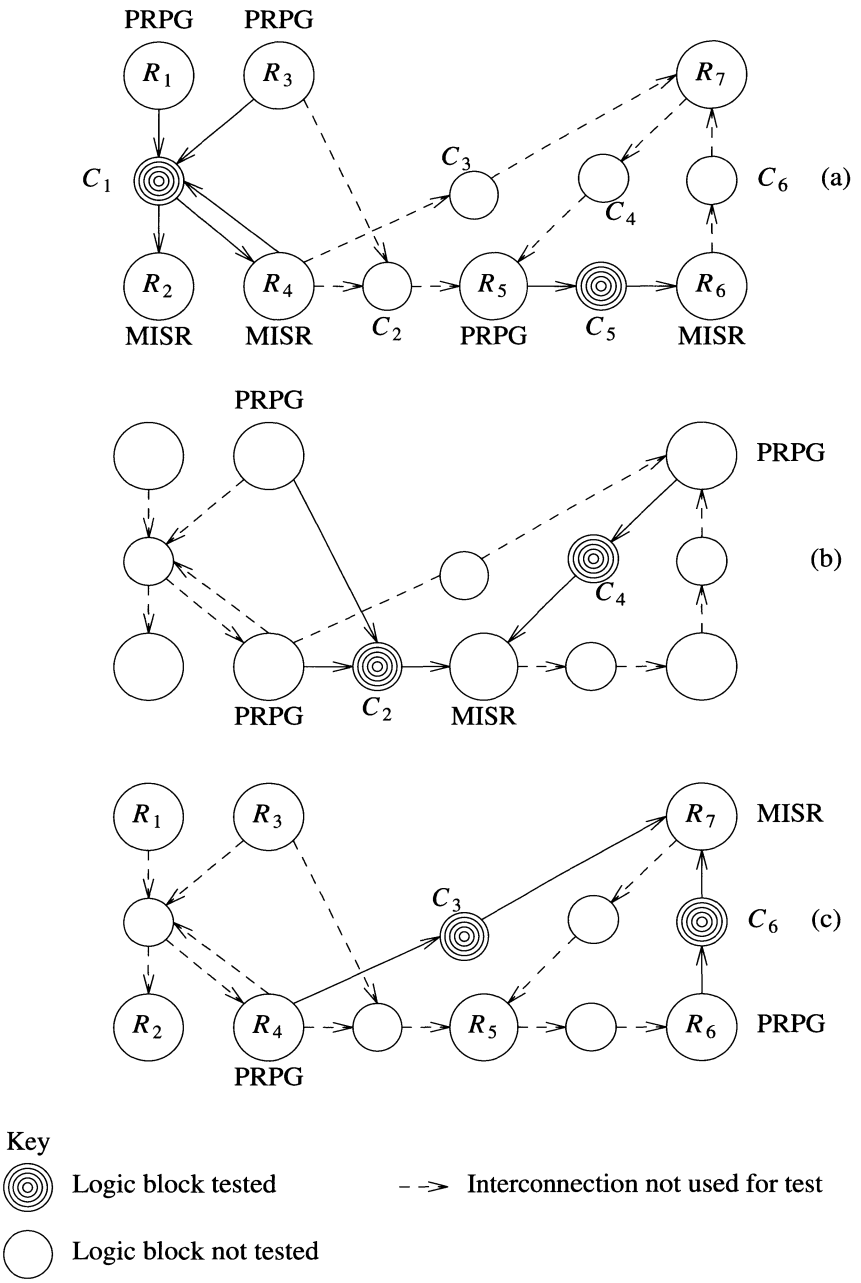
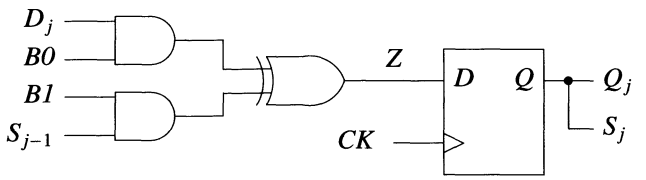
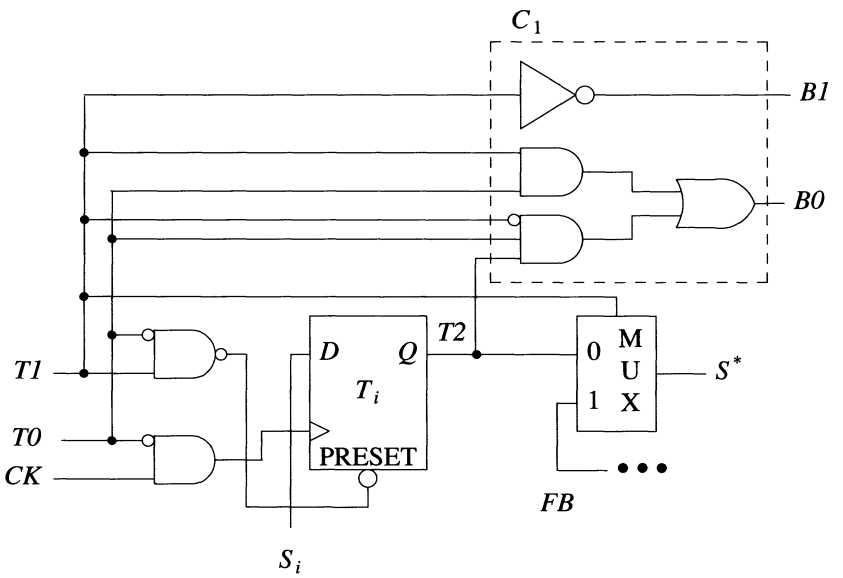


Figure 11.53 Three test sessions for the circuit in Figure 11.51

To control a test session completely, a controller is required to keep count of the number of test-patterns processed during each test session and the number of shift operations required to load and read data from the various scan paths. The design of these test



(a)



(b)

$T0$	$T1$	$T2$	Mode	$B0$	$B1$	S^*
1	0	0	PRPG	0	1	FB
1	0	1	MISR	1	1	FB
0	0	Q	SHIFT	0	1	Q
1	1	x	LATCH	1	0	x
0	1	1	RESET	0	0	x

(c)

Figure 11.54 Control of a BILBO register

controllers is discussed in [Kalinowski *et al.* 1986] and [Breuer *et al.* 1988]. Most control sequences for the various BIST architectures are similar and consist of the following major steps:

1. Inhibit system clocks and enter the test mode.

2. Initialize control registers with data specific to a test session, such as seed data, scan-path lengths, identification of desired scan paths, and number of test patterns to be processed.
3. Seed the LFSRs and scan paths.
4. Initiate the test process.
5. Process the final signature to determine if an error has been detected.

11.5.3 Partial-Intrusion BIST

Partial-intrusion BIST refers to the concept where only a subset of the registers and/or storage cells in a circuit are made part of an LFSR or shift-register path. The motivation for partial-intrusion BIST is to reduce circuit complexity and enhance performance. Partial-intrusion BIST can be implemented in several ways. We will focus on those techniques where the kernel is combinational logic.

A partial-intrusion BIST design can be achieved using *I*-paths [Abadir and Breuer 1985], which were discussed in Chapter 9.

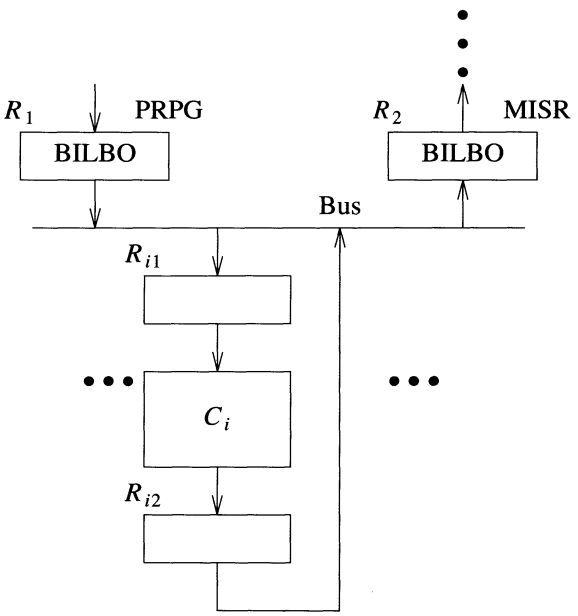


Figure 11.55 A design employing partial BIST

Example 11.7: A portion of a complex design is shown in Figure 11.55. Using the classical BILBO methodology to test C_i, R_{i1} and R_{i2} must be BILBO registers; R_{i1} must be put into the PRPG mode and R_{i2} into the MISR mode. Using a partial-intrusion BIST/BILBO methodology, it is possible to keep R_{i1} and R_{i2} as normal registers and only convert R_1 and R_2 into BILBO registers. To test C_i, R_1 is configured as a PRPG

and R_2 as a MISR. Figure 11.56 illustrates a *test plan* which specifies how C_i is tested by a single test pattern.

Time	Operation	
t	OP_1:	R_1 generates a new test pattern.
$t+1$	OP_2:	R_1 drives bus and R_{i1} loads from bus.
$t+2$	OP_3:	Test data propagate through C_i and response is loaded into R_{i2}.
$t+3$	OP_4:	R_{i2} drives bus and R_2 loads from bus (while in the MISR mode).

Figure 11.56 Initial test plan for C_i

This plan can be repeated until C_i is completely tested. The plan can be optimized so that C_i is tested by a new test pattern every other clock cycle rather than every four clock cycles. To achieve this new plan the design will be considered to be a pipeline. Since both operations OP_2 and OP_4 use the bus, these operations cannot occur simultaneously. To process a new test pattern every other clock cycle, the original test plan must be modified and a hold operation inserted between OP_3 and OP_4. The new test plan is shown in Figure 11.57.

Time	Operation	
t	OP_1:	R_1 generates a new test pattern.
$t+1$	OP_2:	R_1 drives bus and R_{i1} loads from bus.
$t+2$	OP_3:	Test data propagate through C_i and response is loaded into R_{i2}.
$t+3$	OP_4:	R_{i2} holds its state.
$t+4$	OP_5:	R_{i2} drives bus and R_2 loads from bus.

Figure 11.57 Modified test plan for C_i

The resulting pipelined test plan, shown in Figure 11.58, consists of a sequence of parallel operations, where now OP_2 and OP_5 cannot occur simultaneously.

It is seen that two clock cycles after the initiation of test pattern $(j-1)$ occurs, the j test pattern is initiated. The test process consists of two phases that are repeatedly executed. During phase 1 operations OP_4 and OP_2 occur simultaneously, while during phase 2

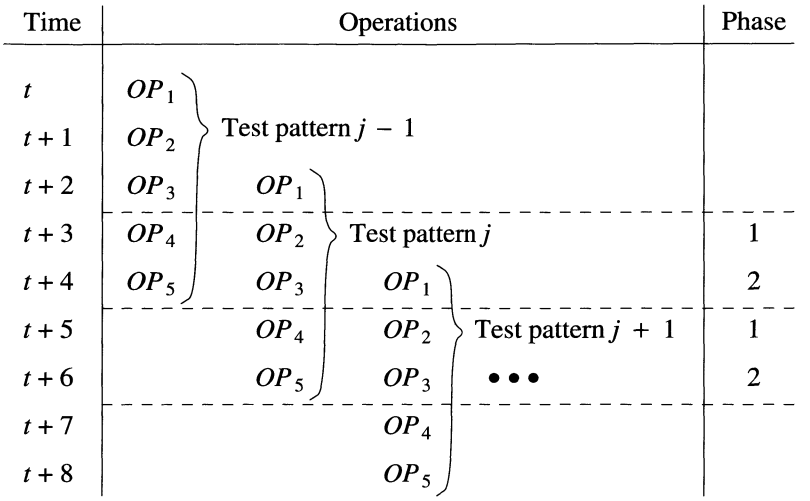


Figure 11.58 Pipelined test plan

operations OP_5, OP_3, and OP_1 are executed. It is assumed that when R_1 and R_2 are not executed in the MISR or PRPG mode, they remain in the hold mode. This is an optimal plan in that it is not possible to process a new test pattern any faster than every other clock cycle using the given hardware configuration. This is because both the driver and receiver must use the bus.

This design approach leads to a reduction in test hardware overhead. Also the delay between R_{i1} and R_{i2} is not increased, which it would be if R_{i1} and R_{i2} were replaced by BILBO registers. This is important if a critical timing path exists between registers R_{i1} and R_{i2}. In addition, R_1 and R_2 can be used to test kernels other than C_i. □

Algorithms for constructing these forms of optimal test plans that are executed with a minimal delay between consecutive test patterns are discussed in [Abadir and Breuer 1985, 1986]. Using this partial-intrusion BIST approach, the blocks of logic in a circuit can be tested sequentially. This is an example of a centralized and embedded version of the original distributed BILBO concept.

Another partial-intrusion BIST approach has been suggested by Krasniewski and Albicki [1985a, 1985b]; an example is shown in Figure 11.59. Here no *I*-paths are used, and R_2 is a normal latch register. Assume that a BILBO-type test method is to be used, that all registers are of width 16, that R_1 is configured as a PRPG and R_3 as a MISR, and that C_1 is to be tested exhaustively. Then C_2 is said to be tested *functionally exhaustively*. That is, even though all 2^{16} patterns are probably not generated at the output of R_2, all possible patterns that can occur in this circuit under normal operation are generated. Thus some input patterns corresponding to don't-care minterms associated with C_2 may not be applied to C_2. This approach can be extended to more complex architectures having both feedforward and feedback interconnections.

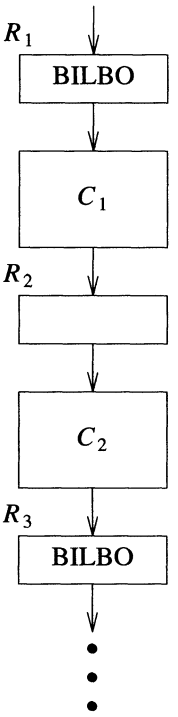


Figure 11.59 A partial BILBO pipeline architecture

11.6 Design for Self-Test at Board Level

Most PCBs consist of commercial chips that do not support IEEE 1149.1. Figure 11.60 shows part of a typical microprocessor-based design. Here the normal buffers, latch registers, and transceivers have been replaced by functionally equivalent ICs that have added capabilities including compatibility with IEEE 1149.1, a pseudorandom-pattern-generation mode, and a multiple-input signature-register mode. This design can now be configured, under control of the test bus, to be put in a BIST mode, where the microprocessor, glue logic, and memory are tested. For example, to test the glue logic, the input buffers can be configured to be in the PRPG mode, while the output buffer and latch register are placed in the MISR mode. Once the test has been run, the test registers can be placed into their scan mode and the resulting signature verified to see if it is correct. A family of circuitry is emerging that supports IEEE 1149.1 and includes test-related functions and can be easily incorporated into board designs to enhance their testability [TI 1989].

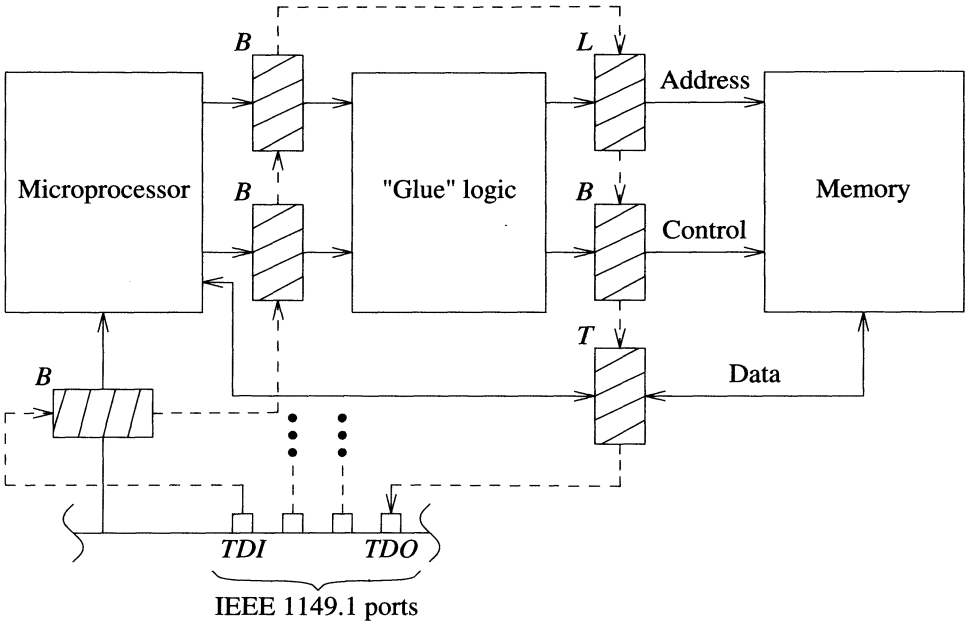


Figure 11.60 PCB with IEEE 1149.1 test chips

REFERENCES

[Abadir and Breuer 1985] M. Abadir and M. A. Breuer, "A Knowledge Based System for Designing Testable VLSI Chips," *IEEE Design & Test of Computers*, Vol. 2, No. 4, pp. 55-68, August, 1985.

[Abadir and Breuer 1986] M. Abadir and M. A. Breuer, "Test Schedules for VLSI Circuits," *IEEE Trans. on Computers*, Vol. C-35, No. 4, pp. 361-367, April, 1986.

[Akers 1985] S. B. Akers, "On the Use of Linear Sums in Exhaustive Testing," *Digest of Papers 15th Annual Intn'l. Fault-Tolerant Computing Symp.*, pp. 148-153, June, 1985.

[Archambeau 1985] E. C. Archambeau, "Network Segmentation for Pseudoexhaustive Testing," Center for Reliable Computing Technical Report No. 85-10, Stanford University, 1985.

[Archambeau and McCluskey 1984] E. C. Archambeau and E. J. McCluskey, "Fault Coverage of Pseudoexhaustive Testing," *Digest of Papers 14th Annual Intn'l. Fault-Tolerant Computing Symp.*, pp. 141-145, June, 1984.

[Bardell *et al.* 1987] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, John Wiley and Sons, New York, 1987.

[Bardell and McAnney 1982] P. H. Bardell and W. H. McAnney, "Self-Testing of Multichip Logic Modules," *Digest of Papers 1982 Intn'l. Test Conf.*, pp. 200-204, November, 1982.

[Bardell and McAnney 1984] P. H. Bardell and W. H. McAnney, "Parallel Pseudorandom Sequences for Built-In Test," *Proc. Intn'l. Test Conf.*, pp. 302-308, October, 1984.

[Bardell and McAnney 1985] P. H. Bardell and W. H. McAnney, "Simultaneous Self-Testing System," U.S. Patent No. 4,513,418, April 23, 1985.

[Bardell and McAnney 1986] P. H. Bardell and W. H. McAnney, "Pseudorandom Arrays for Built-In Tests," *IEEE Trans. on Computers*, Vol. C-35, No. 7, pp. 653-658, July, 1986.

[Bardell and Spencer 1984] P. H. Bardell and T. H. Spencer, "A Class of Shift-Register Sequence Generators: Hurd Generators Applied to Built-In Test," IBM Corp. Technical Report No. TR00.3300, IBM, Poughkeepsie, NY, September, 1984.

[Barzilai *et al.* 1981] Z. Barzilai, J. Savir, G. Markowsky, and M. G. Smith, "The Weighted Syndrome Sums Approach to VLSI Testing," *IEEE Trans. on Computers*, Vol. C-30, No. 12, pp. 996-1000, December, 1981.

[Barzilai *et al.* 1983] Z. Barzilai, D. Coppersmith, and A. L. Rosenberg, "Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing," *IEEE Trans. on Computers*, Vol. C-32, No. 2, pp. 190-194, February, 1983.

[Beenker 1985] F. P. M. Beenker, "Systematic and Structured Methods for Digital Board Testing," *Proc. Intn'l. Test Conf.*, pp. 380-385, November, 1985.

[Benowitz *et al.* 1975] N. Benowitz, D. F. Calhoun, G. E. Alderson, J. E. Bauer, and C. T. Joeckel, "An Advanced Fault Isolation System for Digital Logic," *IEEE Trans. on Computers*, Vol. C-24, No. 5, pp. 489-497, May, 1975.

[Beucler and Manner 1984] F. P. Beucler and M. J. Manner, "HILDO: The Highly Integrated Logic Device Observer," *VLSI Design*, Vol. 5, No. 6, pp. 88-96, June, 1984.

[Boney and Rupp 1979] J. Boney and E. Rupp, "Let Your Next Microcomputer Check Itself and Cut Down Your Testing Overhead," *Electronic Design*, pp. 101-106, 1979.

[Breuer *et al.* 1988] M. A. Breuer, R. Gupta, and J. C. Lien, "Concurrent Control of Multiple BIT Structures," *Proc. Intn'l. Test Conf.*, pp. 431-442, September, 1988.

[Buehler and Sievers 1982] M. G. Buehler and M. W. Sievers, "Off-Line, Built-In Test Techniques for VLSI Circuits," *Computer*, Vol. 18 pp. 69-82, 1982.

[Burkness 1987] D. C. Burkness, "Self Diagnostic Cyclic Analysis Testing System (CATS) for LSI/VLSI," U.S. Patent No. 4,680,761, July 14, 1987.

[Butt and El-Ziq 1984] H. H. Butt and Y. M. El-Ziq, "Impact on Mixed-Mode Self-Test of Life Cycle Cost of VLSI Based Designs," *Proc. Intn'l. Test Conf.*, pp. 338-347, October, 1984

[Chandra *et al.* 1983] A. K. Chandra, L. T. Kou, G. Markowsky, and S. Zaks, "On Sets of Boolean n-Vectors with All k-projections Surjective," *Acta Inform.*, Vol. 19, pp. 103-111, October, 1983.

[Chen 1986] C. L. Chen, "Linear Dependencies in Linear Feedback Shift Register," *IEEE Trans. on Computers*, Vol. C-35, No. 12, pp. 1086-1088, December, 1986.

[Chen 1987] C. L. Chen, "Exhaustive Test Pattern Generation Using Cyclic Codes," *IEEE Trans. on Computers*, Vol. 37, No. 3, pp. 329-338, March, 1987.

[Chin and McCluskey 1984] C. Chin and E. J. McCluskey, "Weighted Pattern Generation for Built-In Self-Test," Center for Reliable Computing Technical Report No. 84-7, Stanford University, 1984.

[Chin and McCluskey 1987] C. K. Chin and E. J. McCluskey, "Test Length for Pseudorandom Testing," *IEEE Trans. on Computers*, Vol. C-36, No. 2, pp. 252-256, February, 1987.

[Craig *et al.* 1988] G. L. Craig, C. R. Kime, and K. K. Saluja, "Test Scheduling and Control for VLSI Built-In Self-Test," *IEEE Trans. on Computers*, Vol. 37, No. 9, pp. 1099-1109, September, 1988.

[DasGupta *et al.* 1982] S. DasGupta, P. Goel, R. F. Walther, and T. W. Williams, "A Variation of LSSD and Its Implications on Design and Test Pattern Generation in VLSI," *Digest of Papers 1982 Intn'l. Test Conf.*, pp. 63-66, November, 1982.

[Eichelberger and Lindbloom 1983] E. B. Eichelberger and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM Journal of Research & Development,* Vol. 27, No. 3, pp. 265-272, May, 1983.

[El-Ziq and Butt 1983] Y. M. El-Ziq and H. H. Butt, "A Mixed-Mode Built-In Self-Test Technique Using Scan Path and Signature Analysis," *Proc. Intn'l. Test Conf.*, pp. 269-274, October, 1983.

[Fasang 1980] P. P. Fasang, "BIDCO, Built-In Digital Circuit Observer," *Digest of Papers 1980 Test Conf.*, pp. 261-266, November, 1980.

[Fasang 1982] P. P. Fasang, "A Fault Detection and Isolation Technique for Microprocessors," *Digest of Papers 1982 Intn'l. Test Conf.*, pp. 214-219, November, 1982.

[Gelsinger 1987] P. P. Gelsinger, "Design and Test of the 80386," *IEEE Design & Test of Computers*, Vol. 4, No. 3, pp. 42-50, June, 1987.

[Gloster and Brglez 1989] C. Gloster, Jr., and F. Brglez, "Boundary Scan with Built-In Self Test," *IEEE Design & Test of Computers*, Vol. 6, No. 1, pp. 36-44, February, 1989.

[Golomb 1982] S. W. Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, California, 1982.

[Ha and Reddy 1986] D. S. Ha and S. M. Reddy, "On the Design of Random Pattern Testable PLAs," *Proc. Intn'l Test Conf.,* pp. 688-695, September, 1986.

[Hassan 1986] S. Z. Hassan, "An Efficient Self-Test Structure for Sequential Machines," *Proc. Intn'l. Test Conf.*, pp. 12-17, September, 1986.

[Hortensius *et al.* 1989] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, and H. C. Card, "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test," *IEEE Trans. on Computer-Aided Design*, Vol. 8, No. 8, pp. 842-859, August, 1989.

[Hudson and Peterson 1987] C. L. Hudson, Jr., and G. D. Peterson, "Parallel Self-Test with Pseudorandom Test Patterns," *Proc. Intn'l. Test Conf.*, pp. 954-963, September, 1987.

[Ichikawa 1982] M. Ichikawa, "Constant Weight Code Generators," Center for Reliable Computing Technical Report No. 82-7, Stanford University, June, 1982.

[Jone and Papachristou 1989] W. B. Jone and C. A. Papachristou, "A Coordinated Approach to Partitioning and Test Pattern Generation for Pseudoexhaustive Testing," *Proc. 26th Design Automation Conf.*, pp. 525-530, June, 1989.

[Jone *et al.* 1989] W. B. Jone, C. A. Papachristou, and M. Pereina, "A Scheme for Overlaying Concurrent Testing of VLSI Circuits," *Proc. 26th Design Automation Conf.*, pp. 531-536, June, 1989.

[Kalinowski *et al.* 1986] J. Kalinowski, A. Albicki, and J. Beausang, "Test Control Line Distribution in Self-Testable VLSI Circuits," *Proc. Intn'l. Conf. on Computer-Aided Design*, pp. 60-63, November, 1986.

[Karpovsky and Nagvajara 1989] M. G. Karpovsky and P. Nagvajara, "Design of Self-Diagnostic Boards by Signature Analysis," *IEEE Trans. on Industrial Electronics*, Vol. 36, No. 2, pp. 241-245, May, 1989.

[Kim *et al.* 1988] K. Kim, D. S. Ha, and J. G. Tront," On Using Signature Registers as Pseudorandom Pattern Generators in Built-in Self-testing," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 8, pp. 919-928, August, 1988.

[Komonytsky 1982] D. Komonytsky, "LSI Self-Test Using Level-Sensitive Scan Design and Signature Analysis," *Digest of Papers 1982 Intn'l. Test Conf.*, pp. 414-424, November, 1982.

[Komonystky 1983] D. Komonytsky, "Synthesis of Techniques Creates Complete System Self-Test," *Electronics*, pp. 110-115, March, 1983.

[Konemann *et al.* 1979] B. Konemann, J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Technique," *Digest of Papers 1979 Test Conf.*, pp. 37-41, October, 1979.

[Konemann *et al.* 1980] B. Konemann, J. Mucha, and G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits," *IEEE Journal Solid State Circuits*, Vol. SC-15, No. 3, pp. 315-318, June, 1980.

[Krasniewski and Albicki 1985a] A. Krasniewski and A. Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules," *Proc. Intn'l. Test Conf.*, pp. 362-371, November, 1985.

[Krasniewski and Albicki 1985b] A. Krasniewski and A. Albicki, "Self-Testing Pipelines," *Proc. Intn'l. Conf. on Computer Design*, pp. 702-706, October, 1985.

[Krasniewski and Pilarski 1989] A. Krasniewski and S. Pilarski, "Circular Self-Test Path: A Low-Cost BIST Technique for VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 8, No. 1, pp. 46-55, January, 1989.

[Kuban and Bruce 1984] J. R. Kuban and W. C. Bruce, "Self-Testing of the Motorola MC6804P2," *IEEE Design & Test of Computers*, Vol. 1, No. 2, pp. 33-41, May, 1984.

[Kumar 1980] S. K. Kumar, "Theoretical Aspects of the Behavior of Digital Circuits Under Random Inputs," Ph.D. thesis, University of Southern California, June, 1980.

[Lake 1986] R. Lake, "A Fast 20K Gate Array with On-Chip Test System," *VLSI System Design*, Vol. 7, No. 6, pp. 46-66, June, 1986.

[LeBlanc 1984] J. LeBlanc, "LOCST: A Built-In Self-Test Technique," *IEEE Design & Test of Computers*, Vol. 1, No. 4, pp. 42-52, November, 1984.

[Lin and Costello 1983] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[Lisanke *et al.* 1987] R. Lisanke, F. Brglez, A. J. deGeus, and D. Gregory, "Testability-Driven Random Test-Pattern Generation," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 6, pp. 1082-1087, November, 1987.

[McCluskey 1984] E. J. McCluskey, "Verification Testing — A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, Vol. C-33, No. 6, pp. 541-546, June, 1984.

[McCluskey and Bozorgui-Nesbat 1981] E. L. McCluskey and S. Bozorgui-Nesbat, "Design for Autonomous Test," *IEEE Trans. on Computers*, Vol. C-30, No. 11, pp. 860-875, November, 1981.

[Parker 1976] K. P. Parker, "Adaptive Random Test Generation," *Journal of Design Automation and Fault-Tolerant Computing*, Vol. 1, No. 1, pp. 52-83, October, 1976.

[Patashnik 1983] O. Patashnik, "Circuit Segmentation for Pseudoexhaustive Testing," Center for Reliable Computing Technical Report No. 83-14, Stanford University, 1983.

[Peterson and Weldon 1972] W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*, 2nd ed., M.I.T. Press, Cambridge, Massachusetts, 1972.

[Resnick 1983] D. R. Resnick, "Testability and Maintainability with a New 6K Gate Array," *VLSI Design*, Vol. 4, No. 2, pp. 34-38, March/April, 1983.

[Saluja *et al.* 1988] K. K. Saluja, R. Sharma, and C. R. Kime, "A Concurrent Testing Technique for Digital Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 12, pp. 1250-1259, December, 1988.

[Sastry and Marjumdar 1989] S. Sastry and A. Marjumdar, private communication, September, 1989.

[Savir and Bardell 1984] J. Savir and P. H. Bardell, "On Random Pattern Test Length," *IEEE Trans. on Computers*, Vol C-33, No. 6, pp. 467-474, June, 1984.

[Schnurmann *et al.* 1975] H. D. Schnurmann, E. Lindbloom, and R. F. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Trans. on Computers*, Vol. C-24, No. 7, pp. 695-700, July, 1975.

[Sedmak 1979] R. M. Sedmak, "Design for Self-Verification: An Approach for Dealing with Testability Problems in VLSI-Based Designs," *Digest of Papers 1979 Test Conf.*, pp. 112-124, October, 1979.

[Shperling and McCluskey 1987] I. Shperling and E. J. McCluskey, "Circuit Segmentation for Pseudoexhaustive Testing via Simulated Annealing," *Proc. Intn'l. Test Conf.*, pp. 58-66, September, 1987.

[Stroud 1988] C. S. Stroud, "Automated BIST for Sequential Logic Synthesis," *IEEE Design & Test of Computers*, Vol. 5, No. 6, pp. 22-32, December, 1988.

[Tang and Woo 1983] D. T. Tang and L. S. Woo, "Exhaustive Test Pattern Generation with Constant Weight Vectors," *IEEE Trans. on Computers*, Vol. C-32, No. 12, pp. 1145-1150, December, 1983.

[Tang and Chen 1984a] D. T. Tang and C. L. Chen, "Logic Test Pattern Generation Using Linear Codes," *IEEE Trans. on Computers*, Vol. C-33, No. 9, pp. 845-850, September, 1984.

[Tang and Chen 1984b] D. T. Tang and C. L. Chen, "Iterative Exhaustive Pattern Generation for Logic Testing," *IBM Journal of Research & Development*, Vol. 28, pp. 212-219, March, 1984.

[Taylor and Karlin 1984] H. M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, Academic Press, New York, 1984.

[TI 1989] Texas Instruments, "SCOPE Testability Octals," Preview Bulletin SCBT098, 1989.

[Timoc *et al.* 1983] C. Timoc, F. Stott, K. Wickman, and L. Hess, "Adaptive Self-Test for a Microprocessor," *Proc. Intn'l. Test Conf.*, pp. 701-703, October, 1983.

[Udell 1986] J. G. Udell, Jr., "Test Set Generation for Pseudoexhaustive BIST," *Proc. Intn'l. Conf. on Computer-Aided Design*, pp. 52-55, November, 1986.

[Udell and McCluskey 1989] J. G. Udell, Jr., and E. J. McCluskey, "Pseudoexhaustive Test and Segmentation: Formal Definitions and Extended Fault Coverage Results," *Digest of Papers 19th Intn'l. Symp. on Fault-Tolerant Computing*, pp. 292-298, June, 1989.

[Vasanthavada and Marinos 1985] N. Vasanthavada and P. N. Marinos, "An Operationally Efficient Scheme for Exhaustive Test-Pattern Generation Using Linear Codes," *Proc. Intn'l. Test Conf.*, pp. 476-482, November, 1985.

[Wagner *et al.* 1987] K. D. Wagner, C. K. Chin, and E. J. McCluskey, "Pseudorandom Testing," *IEEE Trans. on Computers*, Vol. C-36, No. 3, pp. 332-343, March, 1987.

[Wang 1982] L. T. Wang, "Autonomous Linear Feedback Shift Register with On-Line Fault-Detection Capability," *Digest of Papers 12th Annual Intn'l. Symp. Fault-Tolerant Computing*, pp. 311-314, June, 1982.

[Wang 1984] L. T. Wang and E. J. McCluskey, "A New Condensed Linear Feedback Shift Register Design for VLSI/System Testing," *Digest of Papers 14th Intn'l Symp. on Fault-Tolerant Computing*, pp. 360-365, June, 1984.

[Wang and McCluskey 1986a] L. T. Wang and E. J. McCluskey, "A Hybrid Design of Maximum-Length Sequence Generators," *Proc. Intn'l. Test Conf.*, pp. 38-47, September, 1986.

[Wang and McCluskey 1986b] L. T. Wang and E. J. McCluskey, "Condensed Linear Feedback Shift Register (LFSR) Testing — A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, Vol. C-35, No. 4, pp. 367-370, April, 1986.

[Wang and McCluskey 1986c] L. T. Wang and E. J. McCluskey, "Concurrent Built-In Logic Block Observer (CBILBO)," *Intn'l. Symp. on Circuits and Systems*, Vol. 3, pp. 1054-1057, 1986.

[Wang and McCluskey 1986d] L. T. Wang and E. J. McCluskey, "Complete Feedback Shift Register Design for Built-In Self-Test," *Proc. Intn'l. Conf. on Computer-Aided Design*, pp. 56-59, November, 1986.

[Wang and McCluskey 1986e] L. T. Wang and E. J. McCluskey, "Feedback Shift Registers for Self-Testing Circuits," *VLSI Systems Design*, pp. 50-58, 1986.

[Wang and McCluskey 1986f] L. T. Wang and E. J. McCluskey, "Circuits for Pseudoexhaustive Test Pattern Generation," *Proc. Intn'l Test Conf.*, pp. 25-37, September, 1986.

[Wang and McCluskey 1986g] L. T. Wang and E. J. McCluskey, "Circuits for Pseudoexhaustive Test Pattern Generation Using Cyclic Codes," Center for

Reliable Computing Technical Report (CRC TR) No. 86-8, Stanford University, July, 1986.

[Wang and McCluskey 1987a] L. T. Wang and E. J. McCluskey, "Circuits for Pseudoexhaustive Test Pattern Generation," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 1, pp. 91-99, January, 1987.

[Wang and McCluskey 1987b] L. T. Wang and E. J. McCluskey, "Built-In Self-Test for Sequential Machines," *Proc. Intn'l. Test Conf.*, pp. 334-341, September, 1987.

[Wang and McCluskey 1987c] L. T. Wang and E. J. McCluskey, "Linear Feedback Shift Register Design Using Cyclic Codes," *IEEE Trans. on Computers*, Vol. 37, No. 10, pp. 1302-1306, October, 1987.

[Williams 1985] T. W. Williams, "Test Length in a Self-Testing Environment," *IEEE Design & Test of Computers*, Vol. 2, No. 2, pp. 59-63, April, 1985.

[Wunderlich 1987] H. J. Wunderlich, "Self Test Using Unequiprobable Random Patterns," *Digest of Papers 17th Intn'l. Symp. on Fault-Tolerant Computing*, pp. 258-263, July, 1987.

[Wunderlich 1988] H. J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns," *Proc. Intn'l. Test Conf.*, pp. 236-244, September, 1988.

PROBLEMS

11.1 Construct the test sequence generated by the LFSR shown in Figure 11.15.

11.2
a. Give what you believe is a good definition of an exhaustive test for a sequential circuit.

b. Can the test described in part a. be realized by using a complete LFSR on the primary inputs?

c. What if the inputs and feedback lines were driven by a complete LFSR?

11.3 Show that by ANDing together pairs of outputs from a 4-stage maximal-length autonomous LFSR, a pseudorandom weighted-patterns generator can be constructed where the probability of generating a 1 is 0.25.

11.4 Show that the modified LFSR shown in Figure 11.61 generates all 2^4 states. The added OR and NOR gates force the LFSR into the (0000) state after the (1000) state.

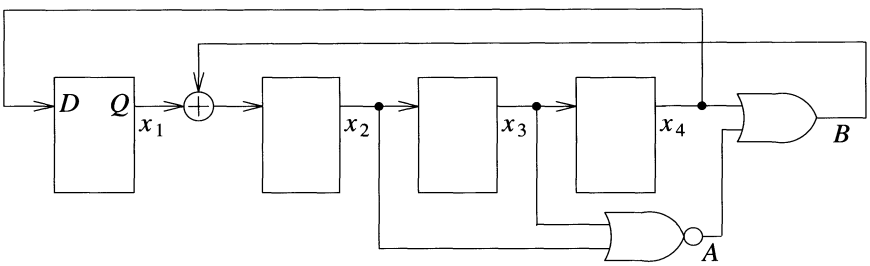


Figure 11.61 A complete LFSR

11.5 For the circuit of Figure 11.5

a. Find a fault that creates additional 1-entries in the dependency matrix but is detected by the given pseudoexhaustive test set.

b. Determine a pseudoexhaustive test set of four vectors that also detects the BF $(a.c)$ (hint: modify the given test set).

11.6 Determine two partitions of the dependency matrix given in Figure 11.62.

11.7 Show that complementing one column of a matrix representing a pseudoexhaustive test set creates a test set that is also pseudoexhaustive.

11.8 Consider a binary matrix where any set of n columns has all 2^n combinations. Show that any set of $k < n$ columns has all 2^k combinations.

11.9 Construct a pseudoexhaustive test set of minimum length for a circuit whose dependency matrix is given in Figure 11.63.

	a	*b*	*c*	*d*	*e*	*f*
x	1	0	1	1	0	0
y	0	1	0	1	0	0
z	0	1	0	0	1	1

Figure 11.62

	a	*b*	*c*
x	1	1	0
y	0	1	1
z	1	0	1

Figure 11.63

11.10 Based on Theorem 11.1, construct the matrices T_0 and T_2 for the case n=5 and $k = 3$. What relation can be observed between T_0 and T_2?

11.11 Determine if the circuit shown in Figure 11.64 is a maximal-test-concurrency circuit, and derive a minimal verification test for this circuit.

11.12 Apply Procedure 11.1 to the circuit of Figure 11.64.

11.13 Apply Procedure 11.1 to the circuit of Figure 11.65.

11.14 Prove that for the case where $p = w + 1$, applying all possible binary patterns of p bits with either odd or even parity will produce a test set where for every subset of w lines, all possible binary patterns will occur.

11.15 For the circuit shown in Figure 11.16, add a minimal number of bypass storage cells so that the resulting circuit can be tested by a verification test for the values of w specified below. Show the various segments produced by the addition of the bypass storage cells.

a. $w = 3$

b. $w = 5$

11.16 Consider a microprocessor chip that contains a PLA-based finite-state controller, a data path consisting of registers and logic, and a RAM and a ROM for memory. Describe the pros and cons of employing the following generic BIST architectures to this circuit.

a. central and separate;

b. distributed and separate;

c. distributed and embedded.

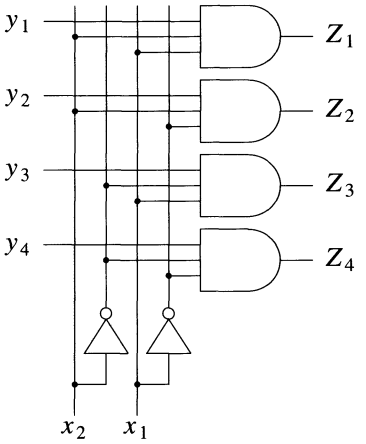


Figure 11.64 A data selector

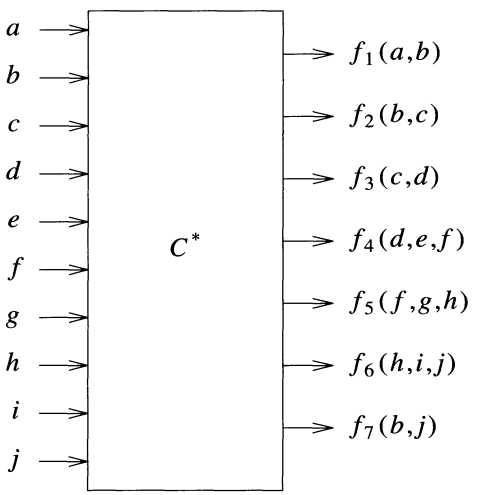


Figure 11.65

11.17 Give a logic design for the BIST hardware required to support the STUMPS methodology at the board level. Assume each chip is designed with a single scan path. Thus every LFSR must be placed into one or more "STUMPS" chips.

11.18 In the CATS BIST architecture why is it necessary that all storage cells be initialized to a known state before executing a test?

11.19 Show the complete circuitry required to test the circuit shown in Figure 11.66 using the following BIST architectures.

a. CSBL;

b. BEST.

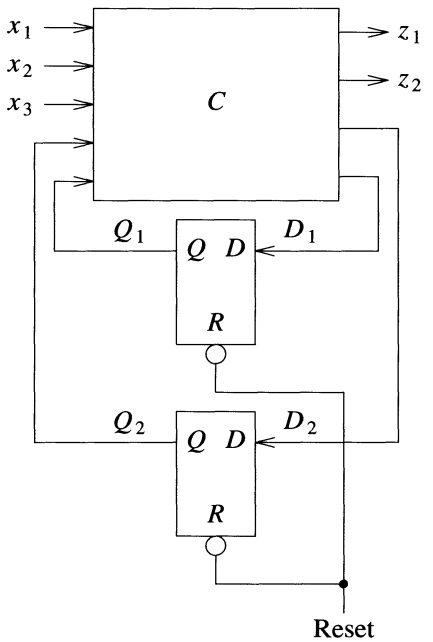


Figure 11.66

11.20 Replace the storage cells in the circuit shown in Figure 11.66 with a scan path. Show the complete circuitry required to test this circuit using the LOCST BIST architecture.

11.21 Consider the concurrent BIST architecture CBIST, where the normal data into the CUT correspond to the binary equivalent of the input sequence 7, 6, 5, 4, 3, 2, 1, 0, 7, 6, 5, ..., while the PRPG goes though the state sequence 0, 1, 2, ..., 7. Determine the number of clock cycles required to test the CUT.

11.22 Determine optimal test plans for the partial-intrusion BIST designs shown in Figures 11.67(a) and 11.67(b).

11.23 Consider the partial-intrusion BIST architecture shown in Figure 11.68, where R_1 generates all 2^{16} test patterns. Is C_2 tested functionally exhaustively? Justify your answer.

11.24 For the register adjacency graph shown in Figure 11.69, determine the minimal number of test sessions required so that each block of logic is tested. Assume that self-adjacent registers can operate as CBILBO registers.

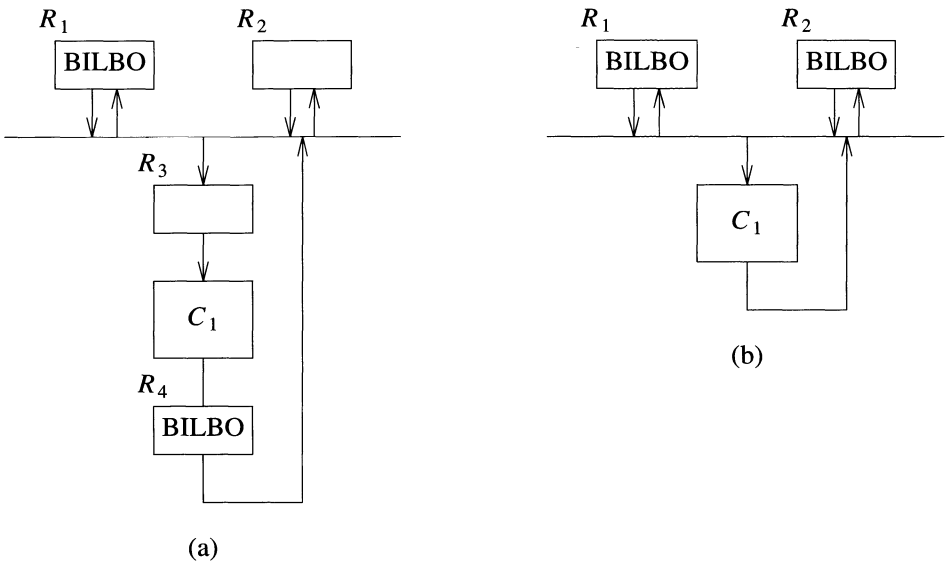


Figure 11.67

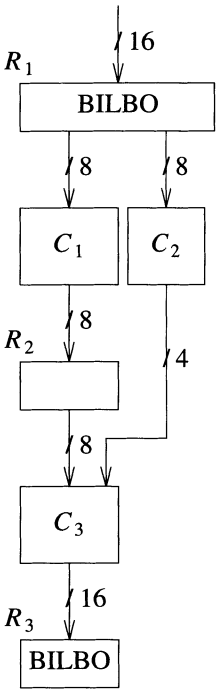


Figure 11.68

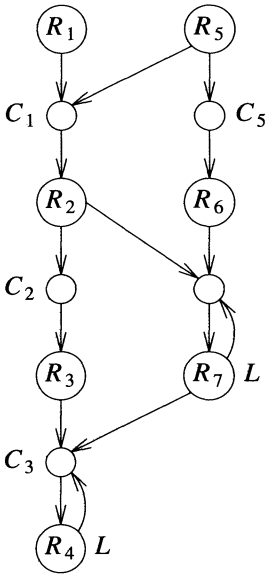


Figure 11.69 Determination of minimal number of test sessions