14. PLA TESTING

About This Chapter

Since programmable logic arrays (PLAs) can implement any Boolean function, they have become a popular device in the realization of both combinational and sequential logic circuits and are used extensively in VLSI designs and as LSI devices on printed circuit boards. Testing of PLAs to detect hardware faults has been studied by numerous researchers. A partial taxonomy of PLA test methods is shown in Figure 14.1. In this chapter we survey many of these methods. First we discuss fault models and traditional methods of test generation. Then we briefly describe test generation algorithms designed specially for PLAs. In Section 14.4 we present several semi-built-in and fully built-in test methodologies for PLAs. In Section 14.5 we compare these various techniques in terms of such measures as area overhead, test time, and fault coverage.

14.1 Introduction

A PLA consists of three parts: an input decoder, an AND array, and an OR array. Figure 14.2 shows the basic structure of a PLA. The input decoder partitions the n primary inputs into $2n$ bit lines. There are two common forms of input decoders: 1-bit decoders that transform each input x_i into x_i and its complement, and 2-bit decoders that transform a pair of inputs into the four minterms of the two variables.

Depending on the technology, a PLA implements two-level logic of various forms, such as NAND-NAND or NOR-NOR. For simplicity we assume the logic is of the form AND-OR, with the AND array forming the product (conjunctive) terms and the OR array the disjunctive terms. The intersection of a bit or output line with a product line is called a *crosspoint*. The proper logical terms are created by either making or not making a connection at each crosspoint. Usually a transistor exists at each crosspoint, and if the connection is not wanted, the associated transistor is disabled by one of several techniques, such as cutting a line.

A PLA can be described by a matrix P = (A, O), where A is an $m \times n$ matrix representing the AND array, O is an $m \times k$ matrix representing the OR array, and n, m, and k are the number of inputs, product terms, and outputs, respectively. This matrix is called the PLA's *personality*. The function realized by a PLA can also be represented by a set of *cubes*, each cube corresponding to a row in the personality matrix. A simple example is given in Figure 14.3. Here a 1(0) in the AND array corresponds to a connection to x_i ($\overline{x_i}$); an x denotes no connection to x_i or $\overline{x}_i$. A 1(0) in the OR array represents a connection (no connection). The PLA shown in Figure 14.3(a) implements the functions

$$f_1 = \overline{x}_1 + x_3 x_4 + \overline{x}_2 \overline{x}_3$$

$$f_2 = x_3 x_4 + \overline{x}_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 + x_2 x_3 + x_1 x_2$$

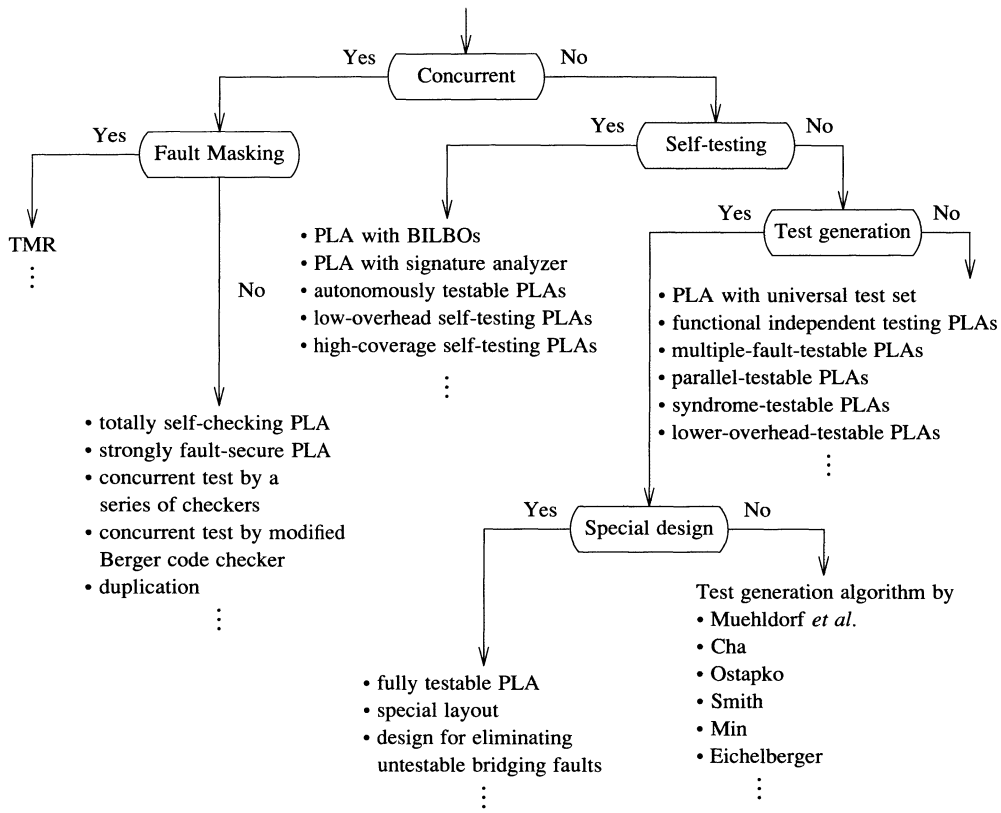


Figure 14.1 A taxonomy of PLA test methodologies

14.2 PLA Testing Problems

The widespread application of PLAs makes PLA testing an important issue. Though PLAs offer many advantages, they also present new testing problems.

14.2.1 Fault Models

In testing digital circuits, the most commonly considered fault model is the stuck-at fault. A *s-a-*1 or *s-a-*0 fault may occur on any wire within a PLA, including input/output lines, bit lines, and product lines. Stuck-at faults alone cannot adequately model all physical defects in PLAs. Because of the PLA's array structure, a new class of faults, known as *crosspoint faults*, often occur. A crosspoint fault is either an extra or a missing connection at a crosspoint in the AND or OR array of a PLA. There are four kinds of crosspoint faults:

1. **Shrinkage fault** — an extra connection between a bit line and a product line in the AND array that causes the implicant to shrink because it includes one additional input variable;

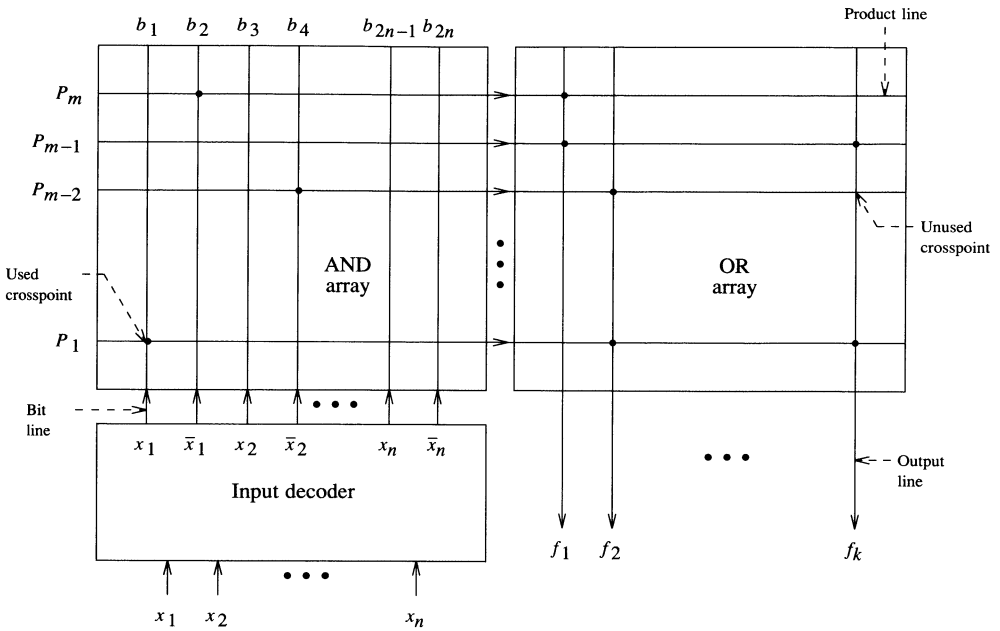
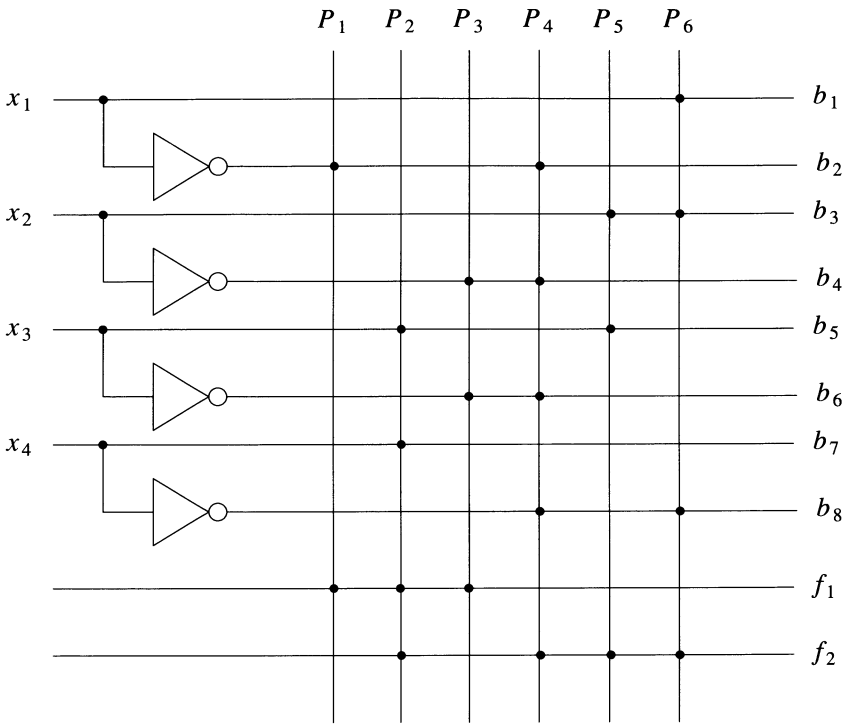


Figure 14.2 A general PLA structure with 1-bit input decoders

2. **Growth fault** — a missing connection between a bit line and a product line in the AND array that causes the implicant to grow because it becomes independent of an additional input variable;
3. **Appearance fault** — an extra connection between a product line and an output line in the OR array that causes the corresponding implicant to appear in the output function;
4. **Disappearance fault** — a missing connection between a product line and an output line in the OR array that causes an implicant to disappear from the corresponding output function.

Note that missing crosspoints are equivalent to some stuck-at faults, but extra connections cannot be modeled by stuck faults. To illustrate these faults, refer to Figure 14.3(a). If an extra connection (shrinkage fault) exists between P_1 and b_5, then f_1 contains the term $\bar{x}_1 x_3$ rather than just $\bar{x}_1$, and the function f_1 loses the two minterms covered by $\bar{x}_1 x_2 \bar{x}_3$. If the connection between P_4 and b_2 is missing (growth fault), then the product term on line P_4 becomes $\bar{x}_2 \bar{x}_3 \bar{x}_4$, and f_2 inherits the extra minterm $x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$. Note that there are $(2n + k)m$ possible single crosspoint faults and $2^{(2n+k)m} - 1$ different single and multiple crosspoint faults. For large PLAs, explicit consideration of single crosspoint faults is difficult; for multiple faults it is totally impractical.



(a)

$$P = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & f_1 & f_2 \end{matrix} \\ \left[\begin{array}{cccc|cc} 0 & x & x & x & 1 & 0 \\ x & x & 1 & 1 & 1 & 1 \\ x & 0 & 0 & x & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ x & 1 & 1 & x & 0 & 1 \\ 1 & 1 & x & 0 & 0 & 1 \end{array}\right] \end{array} = [A,O]$$

(b)

Figure 14.3 A PLA example (a) A PLA implementation (b) A personality matrix

Because of the way PLAs are made, multiple faults are common. Multiple stuck faults, multiple crosspoint faults, or combinations of these faults may occur in newly manufactured circuits. The compact array structure of PLAs makes shorts between two

adjacent lines more likely to occur. The effect of shorts between a pair of lines is determined by the technology used to manufacture the PLA. Generally, either a "high" or "low" voltage will dominate, so these shorts can be modeled as OR or AND bridging faults.

The various fault modes that may occur in a PLA can make test generation a complex process. However, analysis of the relation between different kinds of faults reduces the complexity of the problem. It has been shown that a complete test set for single crosspoint faults also covers most single stuck faults in input decoders and output lines, as well as many shorts and a large portion of multiple faults [Cha 1978, Ostapko and Hong 1979, Smith 1979]. Min [1983a] presented a unified fault model and proved that any stuck-at fault or bridging fault (of AND type) is equivalent to a multiple crosspoint fault. It has been verified that 98 percent of all multiple crosspoint faults of size 8 and less are inherently covered by every complete single crosspoint fault test set in a PLA [Agarwal 1980]. These results suggest that single crosspoint faults should be of primary concern in testing. If other classes of faults are considered significant, special effort must be made to ensure for their high fault coverage.

14.2.2 Problems with Traditional Test Generation Methods

A PLA corresponds to a two-level sum-of-product circuit with input inverters, although in nMOS technology it is often implemented by two levels of NOR-NOR gates with output inverters. One way to generate tests for a PLA is first to convert the PLA into a two-level gate circuit and then to find tests for stuck faults in the "equivalent" gate circuit. Many algorithms exist for generating tests for such circuits (see Chapter 6). However, these methods share two serious problems. First, although the two-level circuit is logically equivalent to the PLA, as far as fault behavior is concerned, they are not equivalent. Some faults in the PLA, such as an extra crosspoint fault in the AND array, cannot be modeled as a stuck fault in the gate circuit. Therefore a high fault coverage is not guaranteed. Second, traditional test generation algorithms are not always effective for PLAs because PLAs have high fanin, reconvergent fanout, and redundancy. Although exhaustive testing is not affected by these factors, it becomes less applicable as the size of the PLA increases.

In general PLAs are not efficiently tested by random test vectors due primarily to the large number of *crosspoints used* in the AND array. Suppose a product line p is connected to j bit lines ($j \leq n$). To test for a missing crosspoint fault at A(p, b_i), one needs to place a 0 on the i-th bit line and 1's at all the other (j–1) connected bit lines. Since there are 2^j possible patterns on these j bit lines and only one pattern can test this fault, the probability of detecting such a missing connection with a random pattern is approximately $1/\ 2^j$. Since j is frequently 10 or larger, many random patterns may be needed to achieve a high fault coverage.

14.3 Test Generation Algorithms for PLAs

Since conventional test generation methods are not suitable for PLAs, several ad hoc test generation approaches have been developed [Bose and Abraham 1982, Cha 1978, Eichelberger and Lindbloom 1980, Min 1983b, Muehldorf and Williams 1977, Ostapko and Hong 1979, Smith 1979, Wei and Sangiovanni-Vincentelli 1985]. It has been shown that a PLA's regular structure leads to more efficient test generation and fault

simulation algorithms than for random logic. We will only briefly review these concepts.

14.3.1 Deterministic Test Generation

The basic idea behind most PLA test generation algorithms is path sensitization, namely, to select a product line and then sensitize it through one of the output lines. We will show that a test must match or almost match (only one bit differs from) a product term in the AND array. If a PLA's personality is known, tests of this nature can easily be found.

The *sharp* operation, denoted by "#"[1], is especially useful for finding such a test. For example, let ci be the cube representation of product line P_i. Suppose the connection between P_i and the j-th input variable is missing, thus creating a growth fault. The cube ci' representing the faulty P_i would be the same as ci except the j-th bit changes to x. To detect this missing crosspoint fault, a test t must be covered by ci' # ci. Let $z(i,k)$ be a set of cubes representing product lines that are connected to output line k, except ci. To propagate the error through output k, the test t must also cause all product lines represented by $z(i,k)$ to be 0. That is to say, $t \in (ci'$ # $ci)$ # $z(i, k)$.

If the result is empty, another k should be tried until a test t is found. If no test can be computed, the fault is undetectable. Formulas for generating tests for other crosspoint faults can be similarly defined [Ostapko and Hong 1979, Smith 1979].

Example 14.1: Consider the PLA shown in Figure 14.3. Assume the crosspoint between lines P_2 and b_7 is missing; hence the product term x_3x_4 becomes x_3. Then $(ci'$ # $ci)$ is equivalent to $x_3(\overline{x_3x_4}) = x_3\bar{x}_4$. Choosing $k = 1$, $z(i,k)$ is equivalent to $\bar{x}_1 + \bar{x}_2\bar{x}_3$, and $(ci'$ # $ci)$ # $z(i, k)$ is equivalent to $x_3\bar{x}_4(\bar{x}_1 + \bar{x}_2\bar{x}_3)' = x_1x_3\bar{x}_4$. Hence the set of tests is represented by $1x10$. □

To determine fault coverage, after a test is generated, test analysis is carried out instead of conventional fault simulation. A PLA's regularity makes it possible to determine directly what faults are detected by a given test. For example, if a test t results in only product line P_j being set to 1, and there is an output line q that does not connect to P_j, then an appearance fault at position (j,q) in the OR array can be detected by t. Such rules of fault analysis can be easily formulated and work much faster than fault simulation for random logic. Details of test analysis for PLAs are given in [Ostapko and Hong 1979, Smith 1979].

Under the single-fault assumption, most test generation algorithms can achieve a high fault coverage for detectable stuck faults and crosspoint faults. AND-type bridging faults can also be covered by some algorithms [Cha 1978, Ostapko and Hong 1979]. Min [1983b] proposed a test generation algorithm for irredundant PLAs that covers multiple crosspoint faults. The tests produced by this algorithm also detect stuck-at faults and bridging faults. In general, the size of the test set produced by these algorithms is bounded by the number of crosspoints in a PLA, namely $m(2n + k)$.

---

1. By definition, a # $b = a\bar{b}$.

14.3.2 Semirandom Test Generation

Deterministic test generation for PLAs is feasible but laborious. Random patterns can be easily generated but are often ineffective. A PLA's regular structure allows deterministic and random test generation methods to be combined into an effective and inexpensive *semirandom test generation* technique [Eichelberger and Lindbloom 1980]. A PLA has a direct logic correspondence with a two-level AND-OR combinational circuit with all input variables and their complements supplied. It is well-known that the circuit shown in Figure 14.4(a) can be tested, under the single stuck-fault assumption, by applying a set of critical cubes to each AND gate (see Figure 14.4(b)) and sensitizing its output through the OR gate. The critical cubes are hard to generate randomly. However, it is probable that a random pattern on G_2 and G_3 will result in a sensitized path for G_1, since for an AND gate with i inputs, $2^i - 1$ out of 2^i possible input combinations lead to an output of 0.

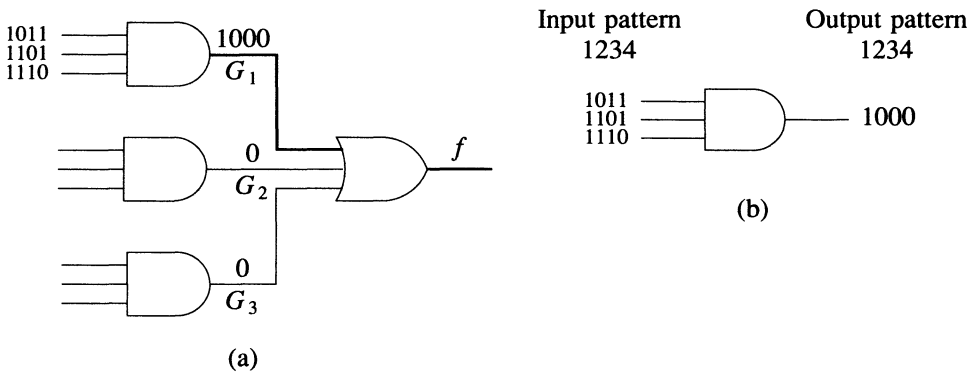


Figure 14.4 (a) Critical patterns for an AND gate together with a sensitized test path (b) Critical cubes for an AND gate

Based on these observations, a semirandom test pattern generation procedure can be developed that assigns critical cubes deterministically and assigns unspecified bits in the corresponding input vector arbitrarily. Such fully specified patterns are candidate tests. They are subject to *pattern evaluation* for determining if the random assignments provide for a sensitized path.

This heuristic test generation method is fast, but it has a limitation in that it only deals with missing crosspoint faults. For 31 PLAs studied, an average fault coverage of 98.44 percent of the modeled faults, which did not include extra connections, was achieved. All undetected crosspoint faults were found to be redundant. Thus semirandom tests can be used for PLAs. Such tests can be easily and quickly generated. However, the resulting test sets are usually large, and hence fault coverage is difficult to determine. To circumvent this problem, a strategy that mixes biased random patterns and deterministic tests can be used. The biased random-pattern method is first used to generate tests for most of the faults. Then the uncovered faults are processed using a deterministic method. The resulting test set can be substantially reduced using fault simulation. An example of such a system is PLATYPUS [Wei and

Sangiovanni-Vincentelli 1985], which employs several heuristics and algorithms used in the PLA logic optimizer ESPRESSO-II [Brayton *et al.* 1984].

The test generation methods discussed provide a software solution to the PLA testing problem. They do not require any hardware changes to the PLA. Since test generation and the application of large test sets are expensive, several alternative methods have been developed for testing PLAs.

14.4 Testable PLA Designs

As PLAs increase in size, more test patterns have to be generated and stored. Sophisticated automatic test equipment is needed to execute the test process. Hence stored-pattern testing becomes a time-consuming and expensive task. To alleviate this problem, several hardware-oriented approaches have been developed that add extra built-in test (BIT) circuitry to the original PLA such that the modified PLA can be easily tested. Most techniques reported to date fall into one of four categories, namely, special coding, parity checking, divide and conquer, and signature analysis. In the following, we will summarize basic principles of testable PLA design methodologies and give examples for each category.

14.4.1 Concurrent Testable PLAs with Special Coding

A PLA's regular memory-like structure suggests the application of special coding for either concurrent or off-line fault detection. The most popular code for PLA testing is the parity code; these techniques will be discussed in a separate section.

To test a PLA concurrently, i.e., during normal operation, requires that during fault-free operation only one product line can be activated by any input vector. Not every PLA has this property. However, simultaneous activation of product lines can be detected and removed, which, unfortunately, increases PLA size [Wang and Avizienis 1979]. In this section we assume that at most one product line is activated by any input vector.

14.4.1.1 PLA with Concurrent Error Detection by a Series of Checkers

A technique proposed by Khakbaz and McCluskey [1981] makes use of the following facts about a PLA:

- The bit lines in the AND array naturally form a two-rail code, i.e., x_i and $\bar{x}_i$.
- During normal operation the signals on the m product lines form a 1-out-of-m code. (This condition is sometimes referred to as a *nonconcurrent* PLA.)
- The fault-free output patterns are determined by the PLA's personality matrix. They can be coded into some error-detection code by adding extra output lines to the OR array.

The proposed testable PLA, shown in Figure 14.5, has three checkers. C_1 is a totally self-checking (TSC) 1-out-of-m checker on all product lines and detects any fault that destroys the nonconcurrent property, such as a product line stuck at 1(0), or any missing and/or extra crosspoint in the AND array. C_2 is a TSC two-rail checker that tests all single stuck-at faults on the bit lines and input decoders. C_3 is an output-code checker. Its complexity depends on how the outputs are coded. The coding is

implemented in that portion of the OR array denoted by D. The simplest code makes all output patterns have even (odd) parity. Here, only one extra output line needs to be added, and C_3 would be a parity checker. In general, C_3 is not a TSC checker and may not be fully tested during normal operation, since the inputs to C_3 are basically the PLA's outputs that are not directly controllable.

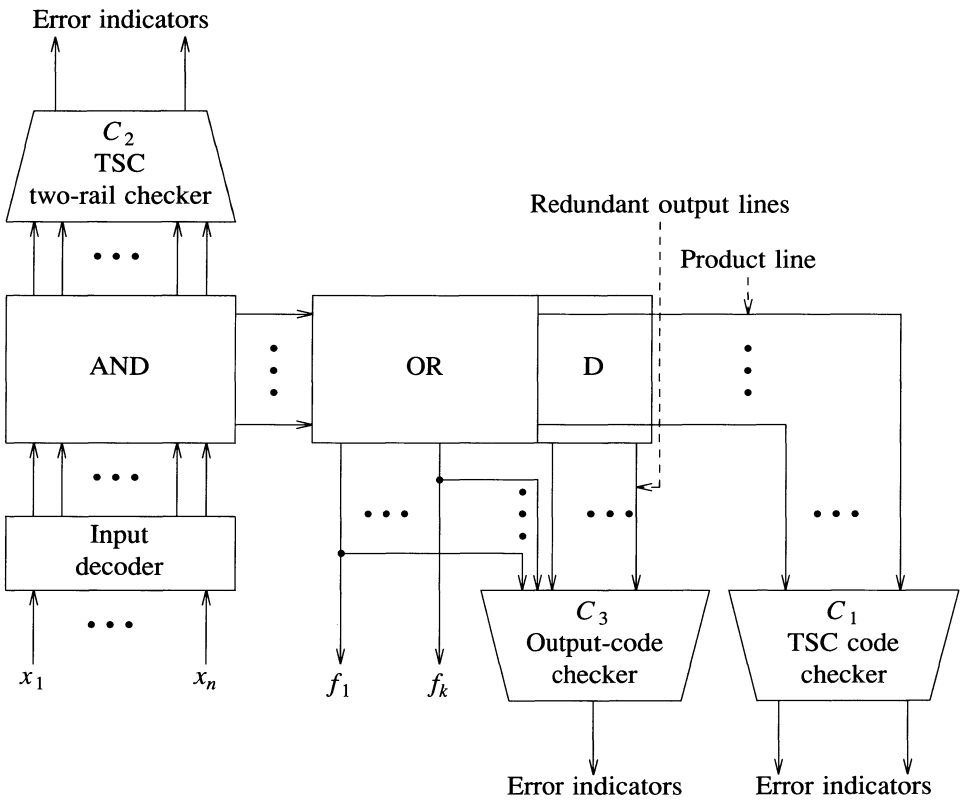


Figure 14.5 A concurrent testable PLA design (CONC1)

Testing occurs concurrently with normal operation. Most errors are caught by one of the three checkers. However, the circuit is not totally self-checking. Therefore off-line testing is still needed to ensure a high fault coverage. This technique combines concurrent error detection with off-line testing by using the same added circuits for both modes of testing. A simple test generation procedure that produces a complete test set has been developed, which is partially function-dependent, and covers faults in both the original PLA and the additional hardware [Khakbaz and McCluskey 1981].

14.4.1.2 Concurrent Testable PLAs Using Modified Berger Code

Output errors of a non-concurrent PLA caused by single stuck-at and crosspoint faults and some multiple faults inside a PLA, except input stuck-at faults, are unidirectional

[Dong and McCluskey 1981]. Consequently, any codes that detect unidirectional errors, such as n-out-of-m codes [Wakerly 1978] and Berger codes [Berger 1961], may be used for the concurrent testing of PLAs. One design that uses these concepts is shown in Figure 14.6. The inputs are associated with a parity bit x_{n+1}. A duplicate parity checker (parity tree) and a totally self-checking two rail checker C_1 detect faults on input lines, bit lines, and in the input decoder. An on-line generator C_2 produces check symbols C^* for each output pattern, which the checker C_1 compares with check symbol C which is generated by the OR array. These check symbols are used to detect output errors.

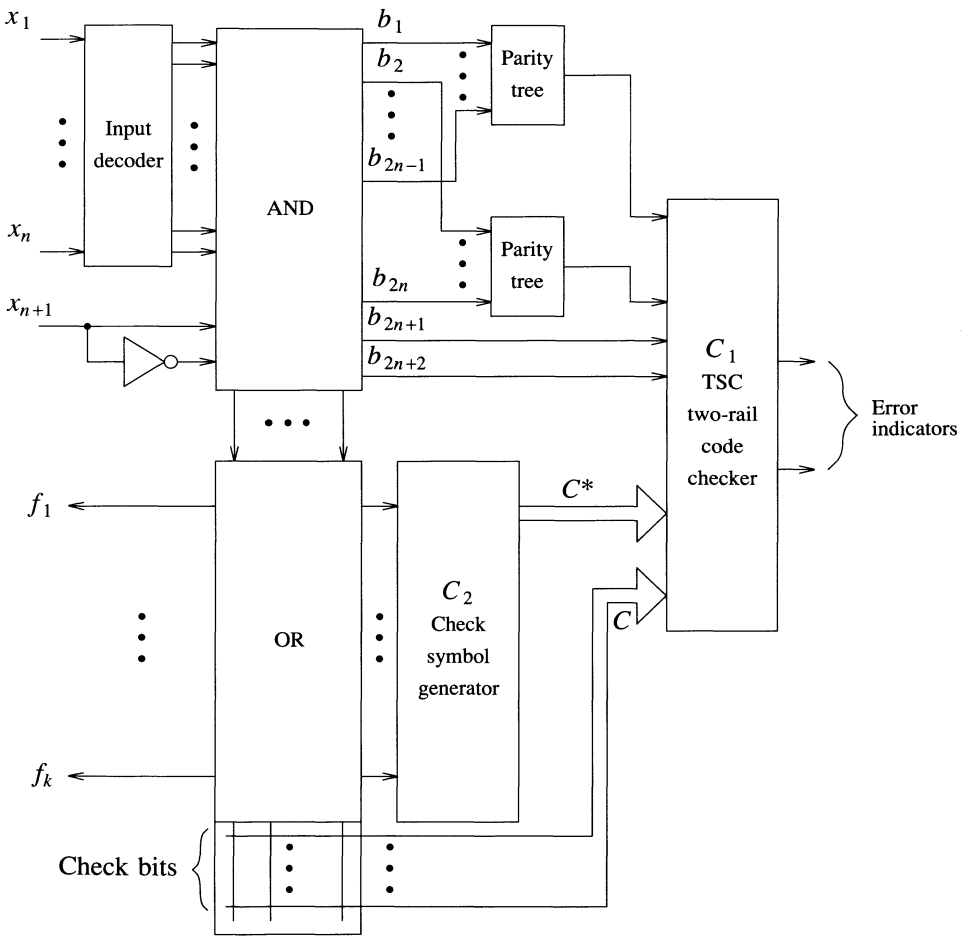


Figure 14.6 A PLA design for concurrent testing (CONC2)

Modified Berger (MB) codes [Dong 1982] are separable codes in which each code word consists of a data part D and a check symbol C. Let $O(D)$ be the number of 1s in D. Then a check symbol C for a MB codeword $[D\ C]$ can be obtained by first

setting $C' = O(D) \ mod \ M$, and then encoding C' into a codeword C that detects unidirectional errors. Here M is a predetermined integer and all unidirectional errors of size not equal to multiples of M can be detected by MB codes. By proper selection of M, all single faults in the AND and OR array can be detected. Suppose the number of inputs is odd, and the inputs are associated with a parity bit x_{n+1}. Faults on input lines, on bit lines, and in the input decoders are detected by a pair of parity checkers (trees) in which one tree generates the parity of all x_i's and the other generates the parity of all $\bar{x}_i$'s. During normal operation, the outputs of the two trees are always complement values, which together with the parity bit, are monitored by a TSC two-rail checker.

Since all other faults cause unidirectional errors on the outputs, the PLA's outputs are checked using a MB code. A check symbol is attached to every output pattern. During normal operation, an on-line generator produces a check symbol C^* for each output pattern, which is compared with the attached check symbol C using a TSC two-rail checker. Any disagreement between these two signals indicates some fault in either the PLA or the checking circuits.

This scheme is suitable for PLAs with a large number of outputs and product lines and in situations where each product line is shared by a small number of outputs, since the number of check bits required only depends on M. If $O(D)$ is small, a small M can be chosen and hence area overhead is not too large. Also with a small number of code bits, the likelihood of every codeword being applied to the PLA increases, so that the totally self-checking property of the two-rail checker can be ensured.

14.4.2 Parity Testable PLAs

Since PLAs have a regular array structure, it is possible to design a PLA so that it can be tested by a small set of deterministic tests that are function-independent, i.e., independent of the personality matrix. This is possible because of two important concepts.

1. Let N_i be the number of used crosspoints on bit line b_i. One can add an extra product line and make connections to it such that every bit line has an even (odd) number of connections with product lines. Then any single crosspoint fault on b_i changes the parity of N_i. The same is true for output lines. Therefore single crosspoint faults can be detected by parity checking on these lines.
2. To test a PLA easily, it must be possible to control individually each bit and product line, and sensitize each product line through the OR array.

14.4.2.1 PLA with Universal Test Set

One of the first easily testable PLA designs employing a universal test set that is thus independent of the PLA's personality is shown in Figure 14.7 [Fujiwara and Kinoshita 1981]. We refer to this design as UTS.

One column and one row are appended to the AND array and the OR array, respectively, so that each bit line in the AND array can have an odd number of connections and the portion of each product line in the OR array can have an even number of connections. Two parity checkers consisting of cascades of XOR gates are used to examine the parities of the two arrays during testing.

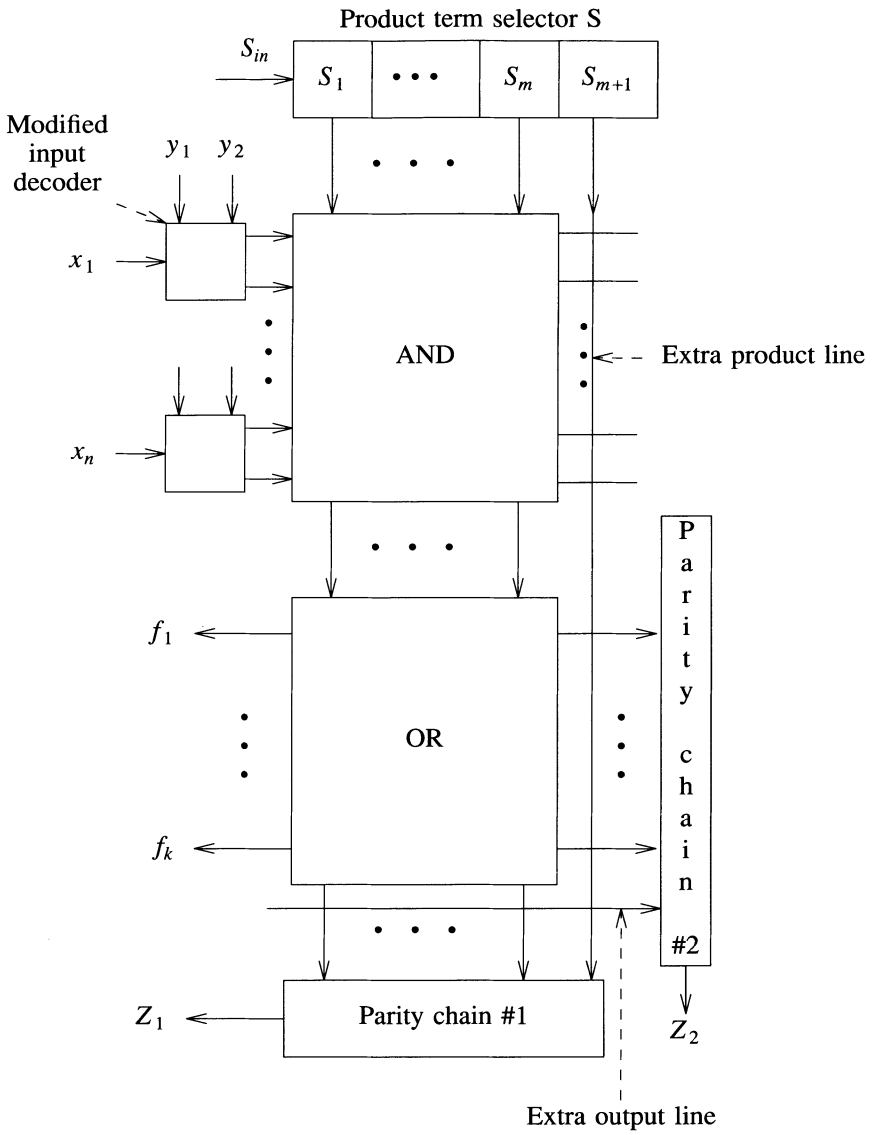
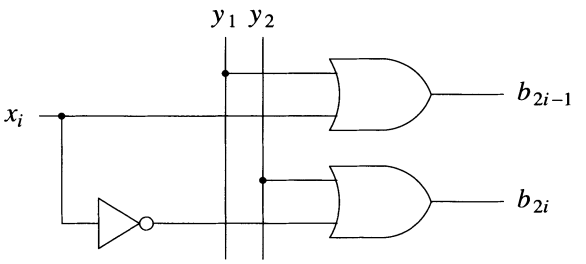


Figure 14.7 PLA with universal test set (UTS)

A selection circuit is used to activate the columns and rows in the PLA so that every crosspoint can be individually selected and tested. In UTS, a product line is selected by adding a shift register S as shown in Figure 14.7. Each bit S_i of S is connected to a product line P_i to create a new product line $P_i' = P_i S_i$. Thus if S_i=1 and S_j=0 for all $j \neq i$, product line P_i' is selected and sensitized to the output, since all other product lines are 0.

Bit lines are selected by modifying the input decoder as shown in Figure 14.8. By properly assigning values to y_1, y_2, and the x_is, all but one input line can be set to 0. Thus that line is selected and tested. During normal operation $y_1 = y_2 = 0$.



x_i	y_1	y_2	b_{2i-1}	b_{2i}
a	0	0	a	$\bar{a}$
a	0	1	a	1
a	1	0	1	$\bar{a}$
—	1	1	1	1

$a \in \{0,1\}$

Figure 14.8 Augmented input decoder and its truth table

Based on this augmented PLA design, a universal test set can be derived (see Figure 14.9), which is applicable to all PLAs regardless of their function. This test set attempts to sensitize each product line and bit line, one at a time. For example, when the test I_{j0} or I_{j1} is applied, all bit lines and the j-th product line are set to 1, and all other product lines are set to 0. Then the crosspoints along the j-th product line in the OR array are tested by the parity chain connected to the output lines, and the result can be observed from Z_2. Similarly, the AND array is tested by applying the J_{i0}s and J_{i1}s and observing the output Z_1. All single stuck-at and crosspoint faults are detected by the parity checkers. This test set can also detect stuck-at faults in the input decoder and the added hardware. The size of the universal test set is linear with n and m. The test I_1 is used to check stuck-at-1 faults in the product term selector.

Hong and Ostapko [1980] independently proposed another design for function-independent testable PLAs (FIT), which employs the same philosophy as UTS. The test set for this design is similar to that of UTS.

14.4.2.2 Autonomously Testable PLAs

For UTS- or FIT-type PLAs, the test vectors must be either stored or generated externally, and the test results have to be monitored during the test process. Yajima and Aramaki [1981] have augmented these schemes so that the resulting PLA circuitry can generate a universal test using a built-in feedback-value generator, a product-term selector, and an input shift register (see Figure 14.10). The feedback-value generator produces test data based on its current state and the outputs of various parity-tree generators. A feedback shift register C_1 is used as both a product-term selector and simultaneously as a signature analyzer. Test results are applied to the product lines using the product-term selector, and evaluated when the test operation is completed by the signature analyzer and detector circuit. Any single fault in the original PLA, and in most of the additional circuits, changes the final signature and therefore is detected. However, some faults in the feedback-value generator and the signature detector are not covered; hence these circuits need to be duplicated to achieve a high fault coverage.

	x_1 • • • x_i • • • x_n	y_1 y_2	S_1 • • • S_j • • • S_m	Z_1 Z_2
I_1	— • • • — • • • —	— —	0 • • • 0 • • • 0	0 0
For j = 1, ..., m				
I_{j0}	0 • • • 0 0 0 • • • 0	1 0	0 • • • 0 1 0 • • • 0	1 1
I_{j1}	1 • • • 1 1 1 • • • 1	0 1	0 • • • 0 1 0 • • • 0	1 1
For i = 1, ..., n				
J_{i0}	1 • • • 1 0 1 • • • 1	0 1	1 • • • 1 1 1 • • • 1	e_m —
J_{i1}	0 • • • 0 1 0 • • • 0	1 0	1 • • • 1 1 1 • • • 1	e_m —

$$e_m = \begin{cases} 0, & \text{if } m \text{ is odd} \\ 1, & \text{if } m \text{ is even} \end{cases}$$

"—" represents a don't-care condition

Figure 14.9 Universal test set for PLAs

The advantages of autonomously testable PLAs are that test patterns need not be generated *a priori* nor stored, hence field testing becomes easy. Since the tests are function-independent, any PLA can be modified in a uniform way.

14.4.2.3 A Built-In Self-Testable PLA Design with Cumulative Parity Comparison

To obtain more efficient testable PLA designs, some researchers have focused on reducing area overhead and/or increasing fault coverage. This can be done using the idea of parity compression [Fujiwara 1984, Treuer *et al.* 1985]. In the testable designs discussed previously [Fujiwara and Kinoshita 1981, Hong and Ostapko 1980], two parity checkers are used to monitor the parity of the two arrays. These checkers can be replaced by a *cumulative parity comparison* method, i.e., by accumulating parity signals in a flip-flop and comparing its value with expected values only at specific times. This scheme is illustrated in Figure 14.11. Two control lines C_1 and C_2 are added to the AND array to disable all x_i's and $\bar{x}_i$'s, respectively. C_1, C_2, and the primary inputs can be used together to select each bit line. As before, a shift register is added to select product lines. One or two product lines are appended to the AND array so that every bit line has (1) an odd number of used crosspoints, and (2) an odd number of unused crosspoints. The same is done for the OR array. Area is saved by eliminating the parity-checking circuit for the product lines. Only one parity chain is employed at the PLA's outputs, and cumulative parity comparison is used to detect errors.

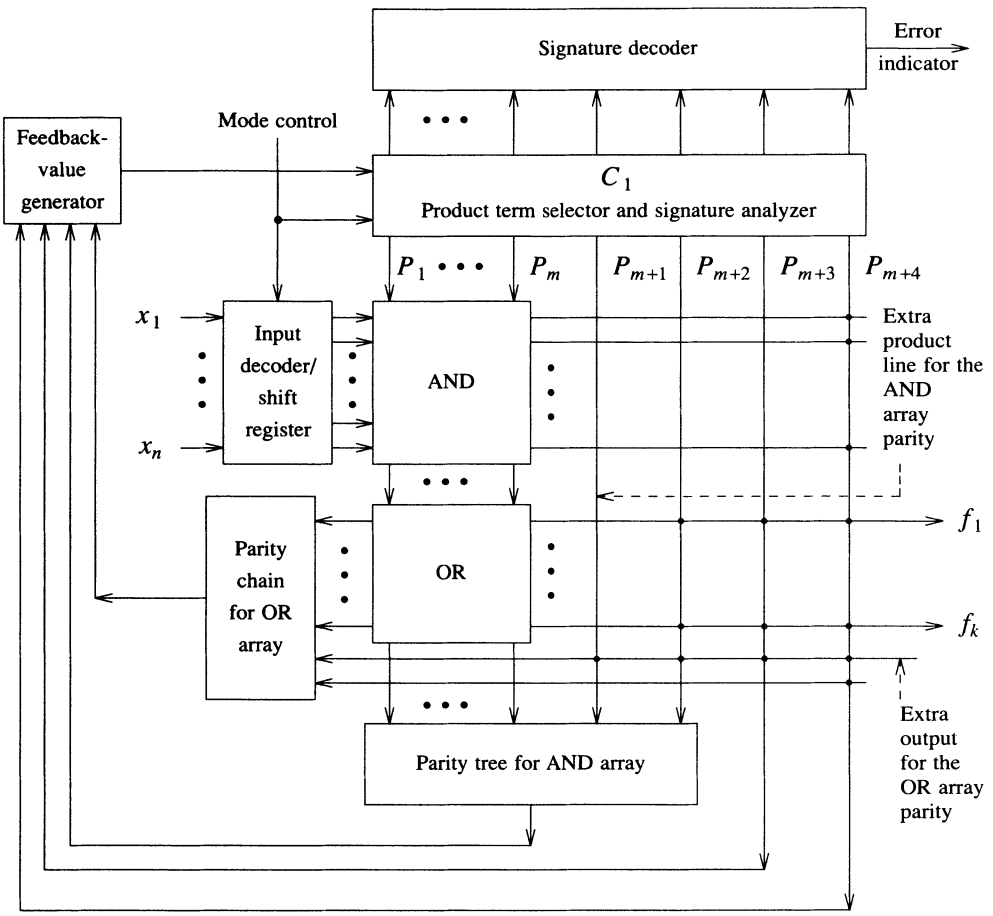


Figure 14.10 An autonomously testable PLA (AUTO)

In the test mode, a universal test set of length $2m(1 + n) + 1$ is applied to the inputs of the PLA. Faults in the OR array can be detected by the parity checker on the outputs. The AND array is tested as follows. Suppose we select one bit line b_i and activate product lines one at a time. When P_j is activated, if there is a device connecting b_i and P_j, P_j is forced to 0 and the output pattern should be all 0s; thus the parity value of the output is 0. If there is no device connecting b_i and P_j, P_j is 1 and the output should have an odd parity. Since there are an odd number of used and unused crosspoints on each bit line, an odd number of 1s and 0s are produced from the output of the parity tree. This information is cumulated in a parity counter that produces the parity of the parity sequence. An interesting property of this scheme is that the sequence of cumulated parity bits at $2n+2m+1$ selected check points is simply a sequence of alternating 0s and 1s. Hence it is easy to generate the expected value on-line. The comparator is shown at the bottom of Figure 14.11. The proven fault

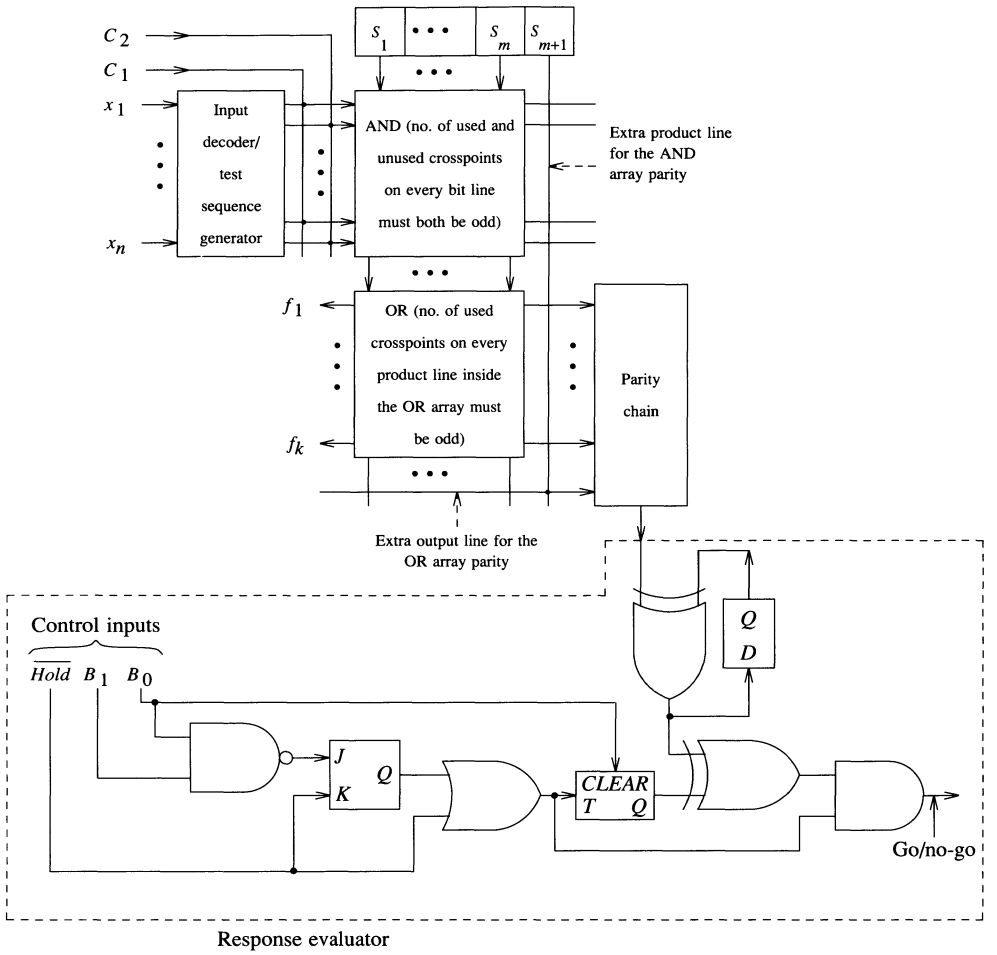


Figure 14.11 A testable PLA with cumulative parity comparison (CPC)

coverage of this testable design is high; all single and $(1 - 2^{-(m+2n)}) \times 100$ percent of all multiple crosspoint faults and all stuck-at and bridging faults are covered.

14.4.3 Signature-Testable PLAs

Signature analysis is a simple and effective way for testing digital systems, and several self-testing PLA designs using this concept have been proposed. In these approaches, a set of input patterns is applied and the results are compressed to generate a signature, which, when compared with a known correct value, determines whether the PLA is faulty.

14.4.3.1 PLA with Multiple Signature Analyzers

One approach proposed by Hassan and McCluskey [1983] uses at least four linear feedback shift registers, *L1*, *L2*, *G*, and *LS*, the first three having the same length and characteristic polynomial (see Figure 14.12). We refer to this design as SIG. Here *G* is used as a maximum-length sequence generator for exhaustive input-pattern generation and the others for signature analysis. The complemented bit lines ($\bar{b}_i$) are fed into *L1* and the true bit lines (b_i) into *L2*. These two signature analyzers are used to detect all single and multiple stuck-at faults on the input and the bit lines. In the absence of faults at the input and bit lines, the signature analyzer *LS* can detect faults in the AND and OR array.

In the test mode, all possible input patterns are generated by *G*. Responses from the PLA are compacted in the three signature analyzers. The final signatures are shifted out for inspection. Using this scheme, all multiple bit-line stuck-at faults, all output faults, most product-line stuck-at faults, and most crosspoint faults are detected. This method is only practical for PLAs with a small number of inputs/outputs and a large number of product lines. Because of the lack of parity checkers, the delay per test pattern is very small.

14.4.3.2 Self-Testable PLAs with Single Signature Analyzer

Grassl and Pfleiderer [1983] proposed a simple function-independent self-testable (FIST) design for PLAs that attempts to reduce the number of test patterns and additional hardware and also takes into account the important aspect of PLA layout. It appears to be applicable to large PLAs. Figure 14.13 shows the block diagram for this design. The input decoder is modified to be a test pattern generator during test mode. It shifts a single "1" across the bit lines. Product lines are selected by the shift register SHR and the selector SEL. The SHR selects a pair of adjacent product lines at a time, so its length is only half of the number of product terms. The SEL will in turn connect product line 1, 3, 5, ... or 2, 4, 6, ... to ground, thus resulting in a unique selection of each product line. During testing, every crosspoint in the AND array is addressed using the two shift registers. The results are analyzed in the multiple-input signature register (MISR) that acts as a conventional latch register during normal operation.

Splitting the product line selector into the SHR and SEL allows the added circuit to fit physically into the narrow pitch of a PLA array, substantially reducing area overhead. A detailed implementation of such a two-stage *m*-bit shift register has been given in [Hua *et al.* 1984]. It consists of a $\lceil m/2 \rceil$-bit shift register and $\lceil m/2 \rceil$ 2-to-2 switch boxes, shown in Figure 14.13(b). A compact parity-checker design is also given. The layout for these BIT structures can fit into the PLA's normal pitch. The connection cost between the BIT hardware and the original PLA is almost zero. This design and layout technique has been adopted by many designers.

The SHR and SEL circuitry operate as follows. Consider the switch box B_i driven by S_i. If $S_i = 0$, then $P_{2i} = P_{2i-1} = 0$; and if $S_i = 1$, then $P_{2i} = 1$ if $Q = 1$, and $P_{2i-1} = 1$ if $Q = 0$. During the test mode a single "1" is shifted through the register consisting of $S_1, S_2, \ldots, S_{m/2}$, which is clocked at one half the normal clock rate. When $Q = 0$, the odd-numbered product terms are selected in turn, while when $Q = 1$ the even-numbered product terms are selected. When $T_2 = 0$ the circuit is in the non-test

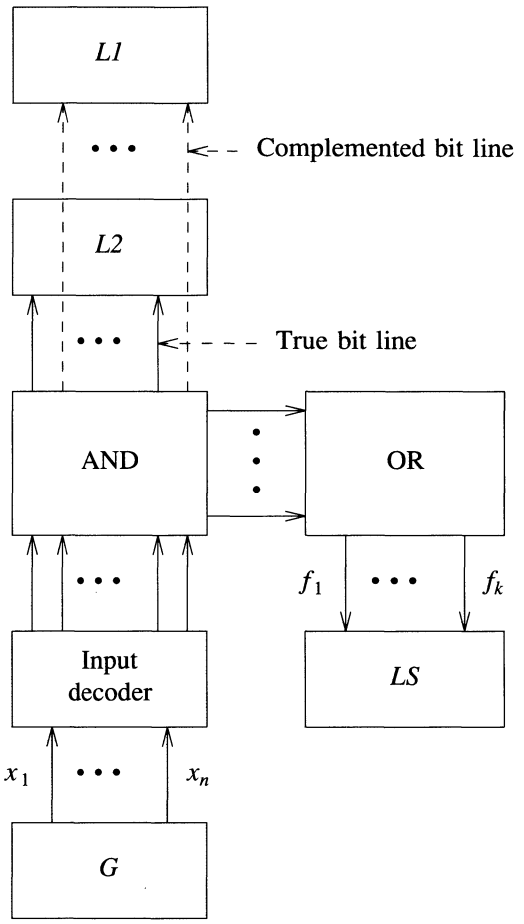


Figure 14.12 Self-testable PLA using signature analysis (SIG)

mode and the connections to the *P*s are cut, leaving all the *P*s in their high impedance mode. In the test mode, $T_2 = 1$ and the values of the *P*s are determined by the state of the two-stage shift register. Now flip-flop *A* toggles every clock time. Combining this operation with the "1" propagating up the shift register produces the desired selection of the P_j lines.

14.4.4 Partitioning and Testing of PLAs

Testing becomes more complex as the size of the PLA increases. A common strategy for dealing with large problems is that of divide and conquer. This principle has also been applied to the design of testable PLAs.

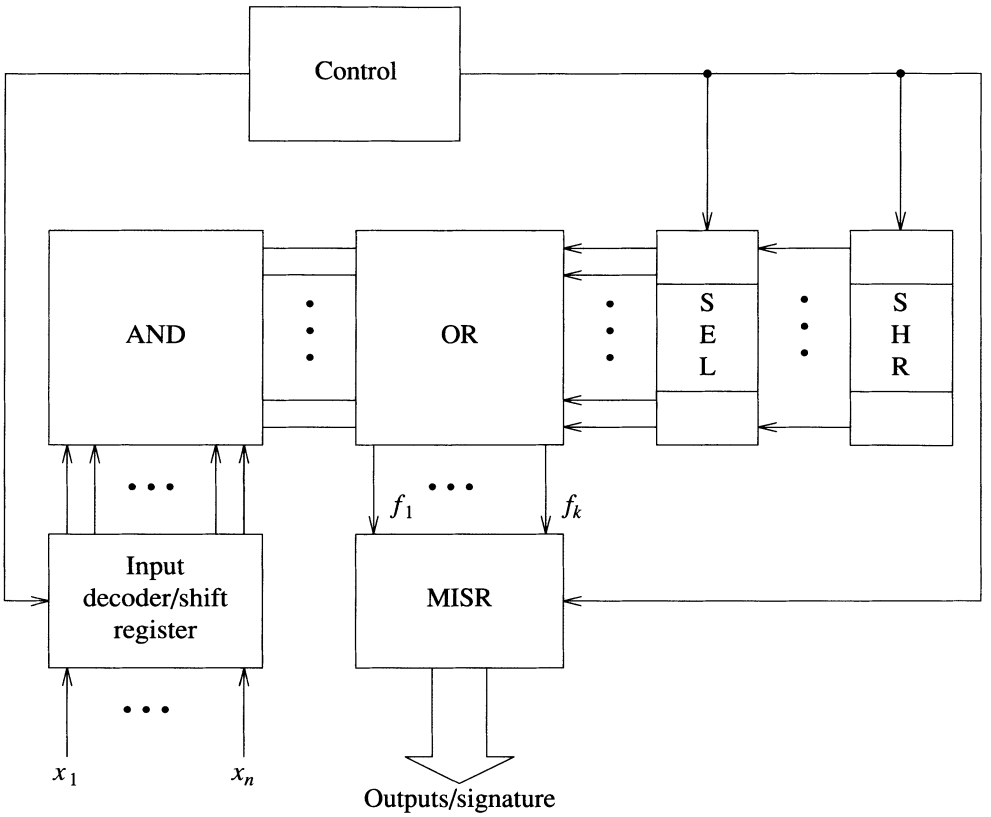
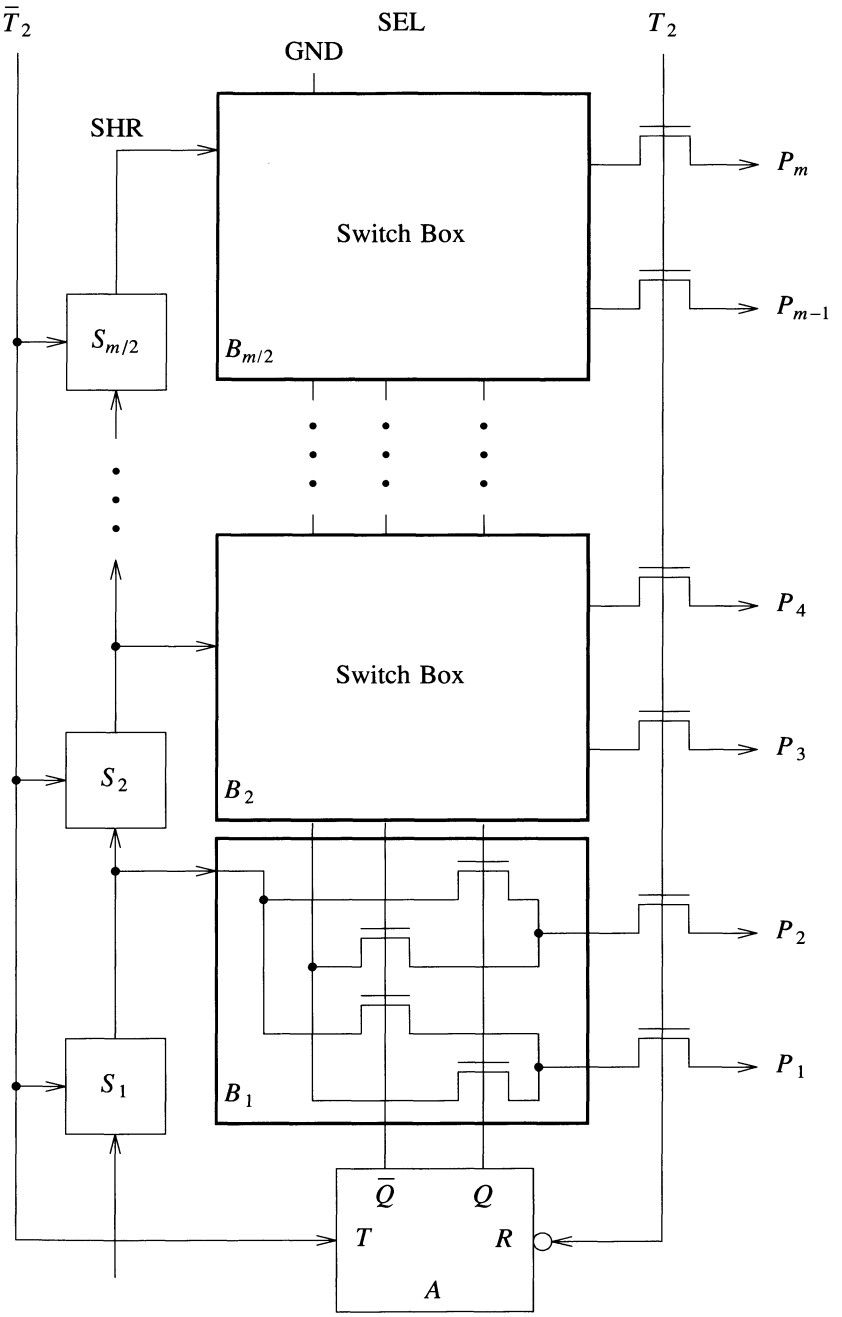


Figure 14.13 Block diagram of a self-testing PLA (FIST)

14.4.4.1 PLA with BILBOs

Daehn and Mucha [1981] suggested a partitioning approach for self-testable PLAs that partitions a PLA into three parts — an input decoder, an AND array, and an OR array — and then inserts three BILBOs between these parts as shown in Figure 14.14. Since BILBOs can be used for both test pattern generation and response evaluation, in the test mode the partitioned blocks can be tested separately by properly controlling the BILBOs. Each block is tested by letting the BILBO at its input operate as a test generator and the BILBO at its output operate as a signature analyzer. The final signature is shifted out for inspection. The three parts of the PLA are tested one by one.

After partitioning, the AND array and the OR array are just arrays of NOR gates. All inputs are controllable and outputs are observable. Testing now becomes a simple



(b)

Figure 14.13 (Continued)

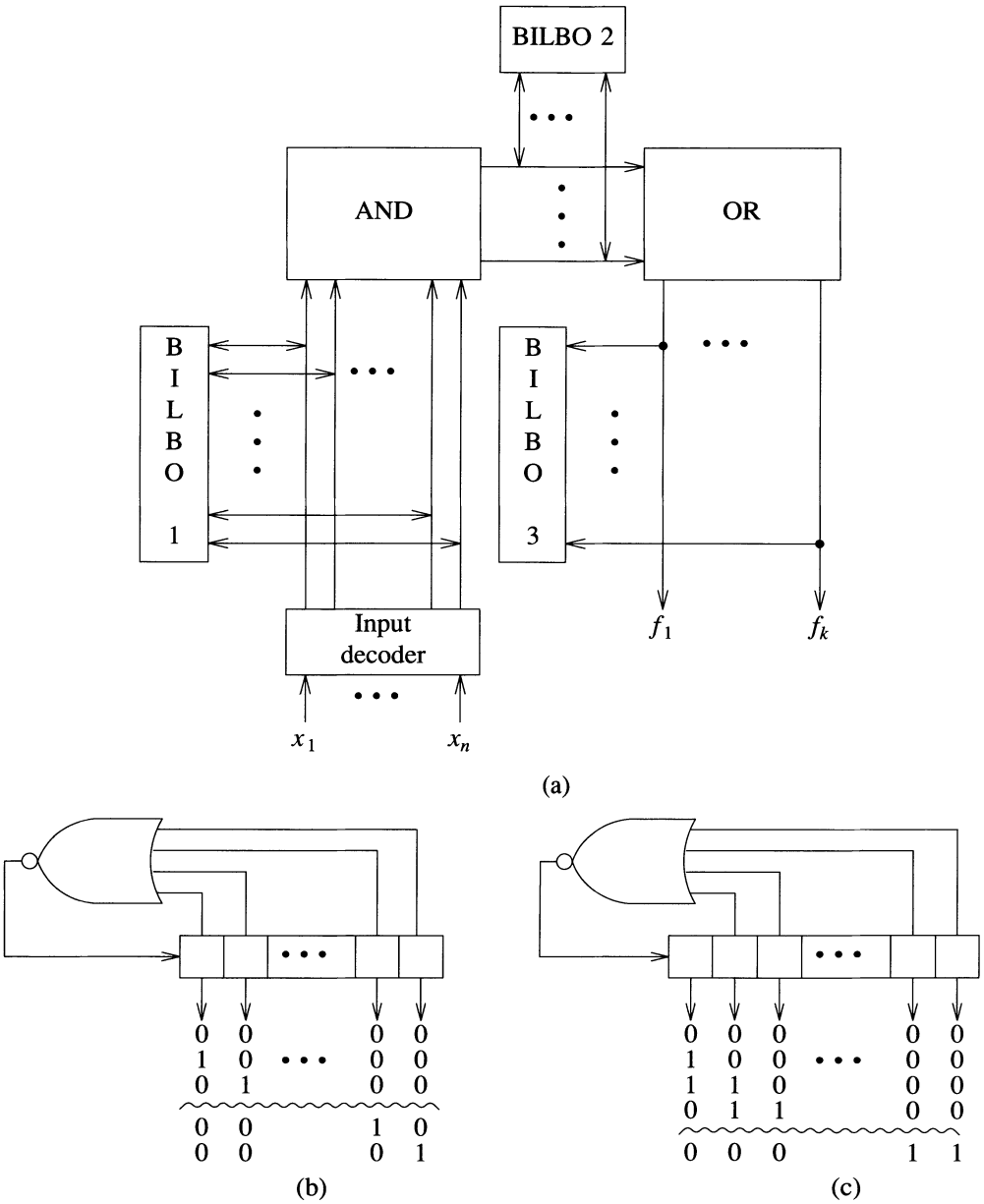


Figure 14.14 Testing a PLA using BILBOs

task. It is well known that a NOR gate can be fully tested by the following simple sequence:

0 0 ... 0
1 0 ... 0
0 1 ... 0
...
0 0 ... 1

A NOR gate array can be tested by the same patterns. Hence the test generator need not be a pseudorandom pattern generator producing all input combinations, but can rather be a nonlinear feedback shift register, as shown in Figure 14.14(b), producing the patterns given above. In this way, the number of test patterns is greatly reduced. This sequence can detect all single stuck faults, crosspoint faults, and bridge faults in the AND or OR array. The faults in the input decoder are tested by a similar sequence that can be produced by the nonlinear feedback shift register shown in Figure 14.14(c).

14.4.4.2 Parallel-Testable PLAs

In most testable designs discussed so far, the product lines are tested individually one at a time. This simplifies testing and leads to a high fault coverage. However, an m-bit shift register is required and a long test sequence of the order of $O(nm)$ may be necessary. By exploiting possible parallelism in testing product lines, high fault coverage can be maintained, while area overhead and the number of test vectors are reduced [Boswell *et al.* 1985].

The conditions for an entire PLA to be testable in parallel (*parallel testable*) are too stringent to be satisfied by general PLAs. It is possible, however, to partition the product lines into groups such that each group is tested in parallel. The procedure for forming these partitions is complex and will not be presented here. A shift register R_1 is added to the circuit, each of its cells controlling one partitioned block of product lines. An output Z_1 is added that connects to all bit lines. Finally a $2n$-bit shift register and an input-control circuit R are inserted for controlling each bit line individually. An example of a parallel-testable PLA (PTPLA) is shown in Figure 14.15.

In the test mode, some patterns are first applied to the entire PLA; then the groups of product lines are tested in sequence. Within each group, product lines are tested in parallel. The lengths of the shift register used for selecting product lines is reduced from m to the number of groups, which is usually less than $m/2$. The number of test patterns is reduced as well because the tests applied to each group are basically the same as those applied previously to each individual line. The test sequence is simple and universal, but the response is function-dependent. It has been shown that all single and multiple stuck faults and crosspoint faults are detected using this technique.

14.4.4.3 Divide-and-Conquer Strategy for Testable PLA Design

The optimum partitioning of a PLA into parallel-testable blocks in a divide-and-conquer (DAC) strategy is a difficult problem. Partitioning a PLA structurally into three parts and inserting BILBOs is easy but requires a substantial amount of area overhead. Figure 14.16(a) illustrates a partitioning technique that has low area overhead and can be easily applied to any PLA [Saluja and Upadhyaya 1985]. The product lines are partitioned into J groups, and a J-bit shift register R_1 is used to control each group. Within each group, there are at most 2^I product lines, where $I = \lceil log_2(m/J) \rceil$; an individual line is selected by a decoder in the decoder-parity AND

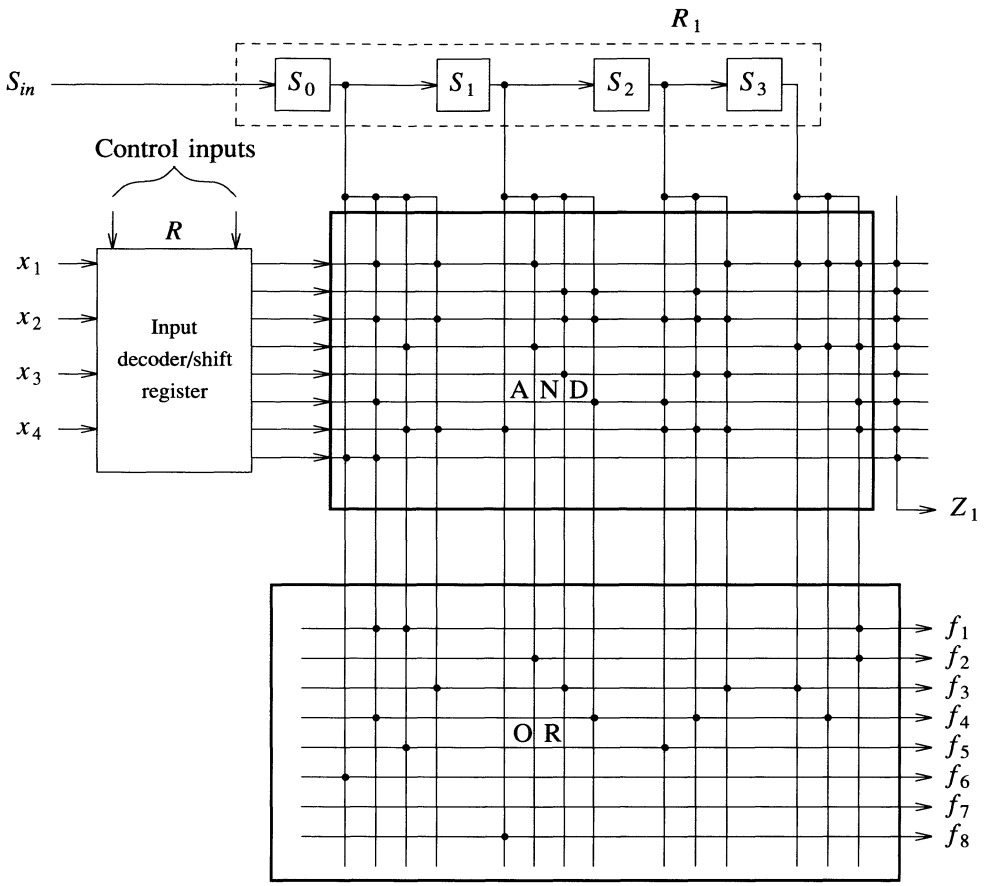
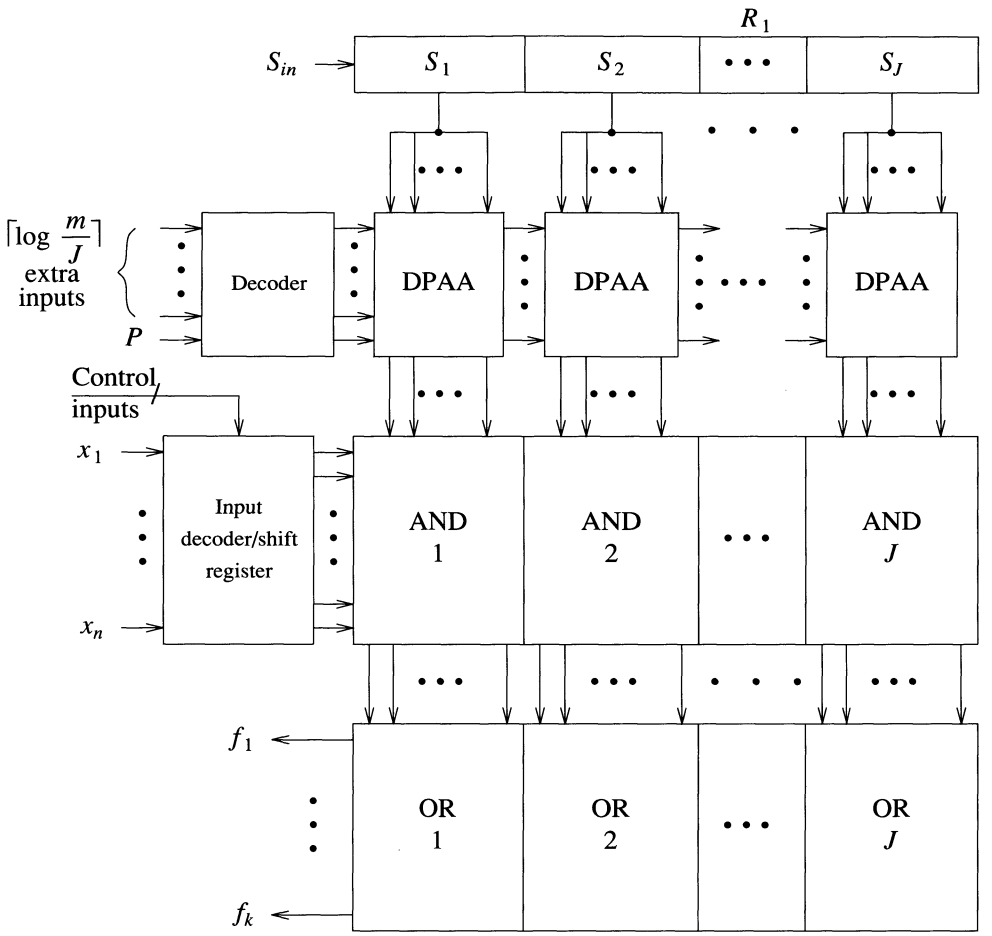


Figure 14.15 A parallel-testable PLA structure (PTPLA)

array (DPAA) [Upadhyaya and Saluja 1988], shown in Figure 14.16(b). During testing, groups of product lines are tested one by one. Within a group, product lines are also tested sequentially. This design does not reduce the number of test patterns, but it reduces the number of shift register cells for selecting product lines. This leads to a simpler layout. It has been proven that for this scheme, a PLA can be tested for all faults by a universal test set of length $m(3+2n+\lceil log(m/J)\rceil)+c$, where c is a constant.

14.4.5 Fully-Testable PLA Designs

Because of redundancy and concurrency in PLAs and the diversity of fault models, it is difficult to test a PLA and achieve a high fault coverage without changing the PLA design. The testable design methods discussed so far solve this problem by inserting a considerable amount of built-in test hardware. To design inherently testable PLAs, several methods have been proposed that simplify the test generation process or



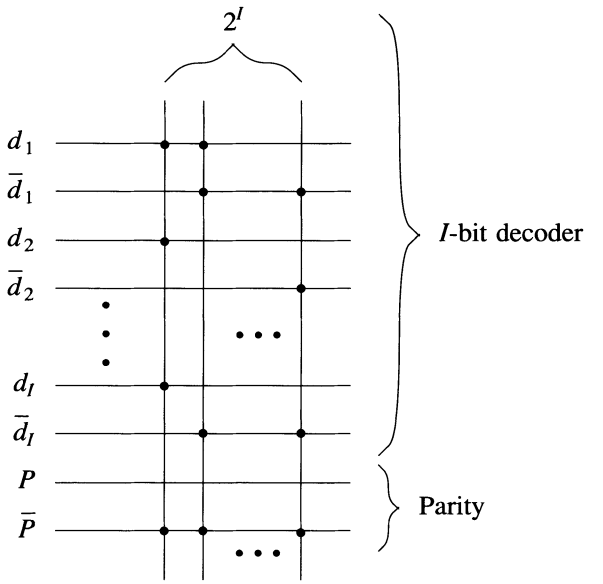
(a)

Figure 14.16 A testable PLA design with DAC partitioning (a) Design-for-test architecture (b) A decoder-parity AND array (DPAA)

improve the fault coverage of existing test generation algorithms by employing little or no BIT circuitry.

A Design of Testable PLAs by Specialized Layout

An example of this type has been proposed by Son and Pradhan [1980]. The PLA is modified to be nonconcurrent in the sense that the AND array only consists of mutually disjoint implicates of the function being implemented. The product lines of the PLA are arranged such that the crosspoint connection patterns on adjacent output lines are distinct. If this cannot be done, an extra output (test point) should be added



(b)

Figure 14.16 (Continued)

to make each pair of neighboring output lines differ from the others. It has been shown that PLAs designed in such a way have the property that the test set for crosspoint faults also covers stuck-at faults and bridging faults. A simple algebraic test generation algorithm exists that calculates the tests for all crosspoint faults.

A PLA Design for Testing Single Bridging Faults

Pradhan and Son [1980] have also shown that there are some undetectable bridging faults in PLAs that can *invalidate* a test set for crosspoint faults. Thus removal of undetectable faults should be considered to ensure a high fault coverage. Such undetectable bridging faults only occur between adjacent bit lines and adjacent product lines. The authors have developed a simple testable design for eliminating untestable bridging faults. For example, if the bridging faults produce AND functions, all bridging faults can be made testable by adding one product line and one output line.

A Design for Complete Testability of PLAs

To achieve a completely testable design, a PLA must be converted into a *crosspoint-irredundant PLA*, namely, a PLA in which all the crosspoint faults are detectable. Any PLA can be made crosspoint-irredundant by adding control inputs. This design also has the advantage that a test set for single crosspoint faults is sufficient to test all single stuck-at faults and bridging faults as well [Ramanatha and

Biswas 1982, 1983]. The complexity of test generation to achieve a high fault coverage is thus reduced.

Low-Overhead Design of Testable PLAs

Most testable designs of PLAs use a shift register to select individual product lines. An m-bit shift register takes up a significant amount of silicon area. A shift register is used to avoid concurrency that is inherent in most PLAs, i.e., an input pattern may activate two or more product lines. Each product term can be uniquely selected by increasing the Hamming distance among product terms [Bozorgui-Nesbat and McCluskey 1984]. Clearly if the Hamming distance between any pair of product terms is at least 1, any input patterns that activate product term P_j will not activate any other P_i for $i \neq j$. For this technique, extra inputs are added such that the Hamming distance between any two product terms is at least 2. Then a main test pattern and n auxiliary test patterns are applied to each product line. For P_i, the main pattern is a completely specified input that activates P_i, and the n auxiliary patterns are those inputs that have exactly one bit different from the main pattern. All main test patterns are found directly from the PLA's personality. Auxiliary patterns can be easily generated. Since an auxiliary pattern has only one bit different from a main pattern, the Hamming distance between any auxiliary pattern for P_i and a main pattern for other P_j is at least 1. Therefore, when testing P_i, no fault-free P_j ($j \neq i$) can be activated; i.e., each product line can be individually selected and tested in the test mode. All single stuck-at faults, missing crosspoint faults, or extra crosspoint faults can be detected using this technique. Since no extra circuitry except for control input lines is added, area overhead is low. However, the problem of finding the minimal number of extra inputs and assigning connections to product lines is *NP*-complete. A heuristic procedure for this problem is described in [Bozorgui-Nesbat and McCluskey 1984], but it also requires extensive computation.

Fully testable (FT) PLAs are alternatives to BIT PLAs. They offer a high degree of testability by changing the logic design of the PLA and adding check points. Since little or no change is made to the PLA's physical structure, they do not create any layout problem. In general, converting a given PLA into a FT PLA is a complex task, and software tools for logic modification and test generation are required. The amount of overhead involved in constructing a fully testable PLA is function-dependent and may be high.

14.5 Evaluation of PLA Test Methodologies

14.5.1 Measures of TDMs

PLAs have many testing and testable design methods. Each method has its advantages and disadvantages. There are several common measures that help in characterizing and evaluating DFT techniques. These measures can be classified into four categories, namely (1) testability characteristics, (2) resulting effect on the original design, (3) requirements for test environment, and (4) design costs.

Testability characteristics specify the degree of testability a test method can achieve, for example, the type of faults detected and the fault coverage; if it supports fault masking, concurrent testing, or self-testing; and whether the tests are function-dependent. A circuit is usually made testable by changing the design. This

may result in additional delay under normal operation, area overhead, and extra I/O pins. Most methods cannot operate without additional support and control logic. Requirements for the application environment specify the hardware needed to support a method in a real-time test process. Finally, design costs relate to the difficulty of implementing a test method, e.g., how complicated it is to modify the logic and the layout. In the following, we will briefly discuss some of these measures and then present a table indicating the values of these measures for numerous test methods.

14.5.1.1 Resulting Effect on the Original Design

Area overhead: BIT circuitry requires extra area, referred to as BIT area. The area overhead for BIT is

$$\text{area overhead} = \frac{BIT\ area}{original\ area}.$$

Different CAD systems use different design rules and generate different PLA layouts. To estimate area overhead, the floor plan of a PLA given in [Treuer *et al.* 1985] will be used, from which it follows that

$$\text{area of original PLA} = 130nm + 65mk + 900n + 300m + 550k + 2200\ (\lambda^2)$$

where n, m, and k are the number of inputs, product lines, and outputs of the PLA, respectively, and λ is a scale factor that is a measure of the resolution of the manufacturing process. For simplicity, one can take the value of λ to be 1 micron. Since BIT circuits consist mainly of standard cells, such as shift register cells and parity checker cells, we will measure the additional area as follows:

$$\text{BIT area} = [\ \Sigma_i\ \#\ \text{of}\ type_i\ \text{cells} \times \text{area of a}\ type_i\ \text{cell}] \times \text{connection cost}.$$

Here connection cost is 1 if a BIT cell can be connected to the original PLA without requiring extra area for wiring. Otherwise, the connection cost is greater than 1. The dominant factor in area overhead is the part of BIT area that grows with PLA size. As no accurate figure for this area can be obtained without carrying out the chip layout, the estimates to be presented emphasize the major part of area overhead, although constant components are also considered. The estimates tend to give a lower bound on area overhead.

Extra delay: Adding BIT hardware may have some side effect on system performance, such as additional delay encountered during normal circuit operation. This delay may affect the system's normal operation or change the system's critical path. Extra delay is measured in terms of gate delays.

Delay per test: Delay per test refers to the maximal number of gate delays in each test cycle, i.e., the maximal time for the effect of one test pattern to propagate through the PLA. It is assumed that the original PLA has three gate delays that are attributed to the input decoder, the AND array, and the OR array. Delay per test partially determines the test application rate and may affect the system clock cycle.

14.5.1.2 Requirements for Test Environment

Test application time: Test application time specifies a lower bound for the time required to complete the entire test process. Suppose (1) there is no overlapping of tests between successive test cycles, (2) each test cycle finishes in one system clock cycle, and (3) all clock cycles are of the same length. Then

test application time = delay per test × length of the shortest test sequence that contains the entire test set in the right order

Note that a test sequence may be different from a sequence of patterns in a test set, because some transition sequences may be necessary in order to apply a given set of tests.

Means for test generation: The way in which tests are generated partially determines the applicability of a test method. To produce the required test inputs and responses before testing, some software tools may be necessary. There are several cases.

Case 1: No tools are required. For example, for the self-testable design shown in Figure 14.11, the tests [Treuer *et al.* 1985] are generated by the BIT hardware and responses are evaluated by the BIT circuit. The logic modification procedure is function-independent, and there is no need for *a priori* test generation.

Case 2: The test patterns are function-independent, but the responses are function-dependent, so simulation is required in order to determine the correct response. For example, in the method using signature analysis [Hassan 1982], the tests are pseudo-random sequences and are generated on-line. However, the expected signature is determined by the function realized by the PLA and has to be determined before testing.

Case 3: The test patterns are function-dependent but can be generated by a simple program.

Case 4: The test patterns are function-dependent and can only be generated by a complex program.

14.5.2 Evaluation of PLA Test Techniques

Various criteria can be used to compare different PLA test techniques. Based on this information, an evaluation matrix has been constructed that contains attribute values for most known PLA test techniques. This matrix is given in Figure 14.18. The abbreviations for the test techniques used in the evaluation matrix are listed in Figure 14.17. None of these techniques masks faults, and only CONC1 and CONC2 support concurrent testing.

Several of the characteristics of these test techniques vary radically with a PLA's parameters. A more complete study of these measures can be found in [Zhu and Breuer 1986a]. In general, area overhead tends to decrease as PLA size increases, where the size of a PLA is defined to be $m(2n + k)$. However, different test techniques are more sensitive to one parameter than another. For example, the area overhead for SIG varies much differently with large values of m than do UTS, BIST2, and FIST. Because of the complexity in evaluating and selecting a test technique for a particular PLA, an expert system has been constructed to aid designers in this task [Zhu and Breuer 1986b]. The techniques presented are applicable to normal PLAs. It is possible to reduce the area of a PLA by carrying out a process referred to as *PLA folding* [Hachtel *et al.* 1982]. Here more than one input and/or output variable can share the same column. Folding introduces a new fault mode, called a *cut point* fault. Ways of detecting this fault as well as how to extend the DFT and BIST techniques presented here to folded PLAs have been developed by Breuer and Saheban [1987].

EXH	Exhaustive test
CONC1	PLA with concurrent error detection by a series of checkers [Khakbaz and McCluskey 1981] — see Figure 14.5
CONC2	Concurrent testable PLA using error-detection code [Dong and McCluskey 1982] — see Figure 14.6
UTS	PLA with universal test set [Fujiwara and Kinoshita 1981] — see Figure 14.7
FIT	Function-independent-testable PLA [Hong and Ostapko 1980]
AUTO	Autonomously testable PLA [Yajima and Aramaki 1981] — see Figure 14.10
BILBO	PLA with BILBOs [Daehn and Mucha 1981] — see Figure 14.14
SIG	PLA with multiple signature analyzers [Hassan and McCluskey 1983] — see Figure 14.12
FIST	Function-independent self-testable PLA [Grassl and Pfleiderer 1983] — see Figure 14.13
TLO	Testable PLA with low overhead and high fault coverage [Khakbaz 1984]
BIT	Built-in tests for VLSI finite-state machines [Hua *et al.* 1984]
CPC	PLA design with cumulative parity comparison [Fujiwara 1984] — see Figure 14.11
LOD	Low-overhead design of testable PLA [Bozorgui-Nesbat and McCluskey 1984]
PTPLA	Parallel-testable PLA [Boswell *et al.* 1985] — see Figure 14.15
BIST2	Built-in self-testable PLA [Treuer *et al.* 1985]
HFC	High-fault-coverage built-in self-testable PLA [Upadhyaya and Saluja 1988]
DAC	Divide-and-conquer strategy for testable PLA [Saluja and Upadhyaya 1985] — see Figure 14.16
TSBF	PLA design for testing single bridging faults [Pradhan and Son 1980]
TBSL	Design of testable PLA by specialized layout [Son and Pradhan 1980]
DCT	Design for complete testability of PLA [Ramanatha and Biswas 1982]
EL	Test generation algorithm by Eichelberger and Lindbloom [1980]
SMITH	Test generation algorithm by Smith [1979]

Figure 14.17 Abbreviations of test techniques used in Figure 14.18

TDM	FAULT COVERAGE (%)						
	Single stuck	Single crosspoint	Single bridge	Multiple stuck-at	Multiple crosspoint		Multiple bridge
					Unidirectional	Others	
EXH	100	100	100	100	100	100	100
CONC1	99*	99*	99*				
CONC2	100	100			100		
UTS	100	100					
FIT	100	100					
AUTO	99*	100					
BILBO	99*	99*	99*				
SIG	98*	98*		98*			
FIST	99*	99*	99*	99*	99*	99*	99*
TLO	100	100		100	100	100	
BIT	100	100	100	100	100	100	
CPC	100	100	100	100	100	100	
LOD	100	100	100	100	100	100	
PTPLA	100	100		100	100		
BIST2	100	100	100	$100(1-2^{-(m+2n)})$	$100(1-2^{(m+2n)})$		$100(1-2^{-m})$
HFC	100	100			100	99.9*	
DAC	100	100		100	100	100	
TSBF			100				
TBSL	100	100	100				
DCT	100	100	100				
EL		~ 97 (This figure is for missing crosspoint faults only.)					
SMITH	100	100			100		

Note: Blank entries represent unknown data. * = estimated value.

Figure 14.18 The evaluation matrix of PLA test methodologies

TDM	Self testing	Function dependency		Test generation	Extra IO pins	Extra delay	Delay per test	Number of tests
		Tests	Responses					
EXH	no	no	yes	no	0	0	3	2^n
CONC1	no	yes	no	easy	5	0	12	$2m+D+8$
CONC2	no	no	no	no	2	0	3	0
UTS	no	no	no	no	5	1	$3+2logm$	$2(n+m)+1$
FIT	no	no	no	no	$6+log(n+1)$	1	$4+2logm$	$5+2.5n+2m$
AUTO	yes	no	no	no	2	1	$6+2(k+1)$	$2(n+m)+9$
BILBO	yes	no	yes	no	3 ~ 9	0	4	$2n+m+k+3$
SIG	yes	no	yes	no	4	0	6	2^n
FIST	yes	no	yes	no	1	3	7	$2nm$
TLO	no	yes	yes	easy	3	0	3	$m(7+n)+2$
BIT	yes	no	no	no	5	1	$m+4$	$2n+m+2$
CPC	no	yes	no	no	4	0	3	$2n(m+1)+m+3$
LOD	no	yes	yes	easy	F.D.	0	3	$(n+1)m$
PTPLA	no	no	yes	no	4	1	4	$2nJ+4n+4$
BIST2	yes	no	no	no	4	1	$1.25k+3$	$2m(n+1)+1$
HFC	no	no	yes	no	3	1	6	$m(2n+k+3)+k+2$
DAC	no	no	yes	no	4	1	4	$m(3+2n+logm/J)+4$
TSBF	no	yes	no	med.	2	0	3	$m+1$
TBSL	no	yes	yes	hard	1	0	3	F.D.
DCT	no	yes	yes	hard	F.D.	0	3	F.D.
EL	no	yes	yes	med.	0	0	3	random
SMITH	no	yes	yes	hard	0	0	3	F.D.

Note: F. D. stands for function dependent. J is the number of product line groups.

Figure 14.18 (Continued)

TDM	Test application time	Test storage	Area overhead
EXH	$(3)2^n$	0	0
CONC1	$12(2m+D+8)$	$(2m+D+8)(3+n)$	$15680(n/2-1)+APC(k+1)+AMC$
CONC2	0	0	F.D.
UTS	$(3+2logm)(2n+3m)$	$5(5+n)$	$(m+1)ASR+(m+k)APC+64(6n+m+k+15)$
FIT	$(4+2logm)(5+2.5n+2m)$	$8(5+2.5n+2m)$	$5397+977n+2113m+1021k$
AUTO	$(8+2k)(2n+2m+9)$	0	$2215m+1764n+1420k+11752$
BILBO	$4(2n+2m+max(n,k)+1)$	$2n+2m+max(n,k)$	$1.3(2n+m+max(n,k))AB$
SIG	$(6)2^n$	$n(2+[k/n])$	$1.3(3n+[k/n])ASG$
FIST	$14nm$	k	$(2n+m)ASR+1.1k(ASG)$
TLO	$3m(7+n)+6$	$(7m+nm+2)(n+k+2)$	$m(64+ASR)+400$
BIT	$(m+4)(2n+m+2)$	0	$(2n+m+1)ASR+(m+k+2)APC+2AF+128(m+2n+k)+896$
CPC	$6n(m+1)+3m+9$	$(k+1)(2n(m+1)+m+3)$	$(m+1)ASR+384n+64m+64k+944$
LOD	$3m(n+1)$	$m(n+nk+k)$	$C(128m+640)$
PTPLA	$8nJ+24n+12$	F.D.	$64m+128n+696+ASR(2n+J)$
BIST2	$(1.25k+3)(2nm+2m+1)$	0	$1360m+2160n+760k$
HFC	$6m(2n+k+1)+6k$	0	$(2n+m+k+2)ASR+(2AC+5)logm+128m+1040$
DAC	$4m(2+2n+logm)+16$	0	$(1+2n)ASR+(1+[log(m/J])AC$
TSBF	$3m+3$	$(m+1)(n+2)$	$64(2n+m+k)+816$
TBSL	F.D.	F.D.	F.D.
DCT	F.D.	F.D.	$16n+8k+256m+1989$
EL	random	random	0
SMITH	F.D.	F.D.	0

Note: C is an integer that is function-dependent and usually small. D is the number of used crosspoints. ASR=area of 1-bit shift register. APC=area of 1-bit parity checker. AMC=area of a 1-out-of-m checker. ASG=area of a signature analyzer cell. AB=area of a BILBO register cell. AC=area of 1-bit counter. AF=area of a flip-flop.

Figure 14.18 (Continued)

TDM	EXTRA TEST SUPPORT MEANS REQUIRED		
	Test set generation	Test application	Response evaluation
EXH	response simulation	counter | LFSR	ATE
CONC1	simple TG program	ATE | ROM	monitor error indication
CONC2	none	none	none
UTS	none	ATE | on-line PG	monitor two extra outputs
FIT	none	ATE | on-line PG	ATE | ROM & comparator
AUTO	none	none	none
BILBO	obtain signature	control BILBO function	S.R. & comparator
SIG	obtain signature	none	ROM & comparator
FIST	obtain signature	none	comparator | signature decoder
TLO	simple TG program	ATE | ROM & on-line PG	ATE
BIT	none	none	monitor 3 extra outputs
CPC	response simulation	ATE | on-line PG	ATE
LOD	simple TG program	ATE | ROM & on-line PG	ATE
PTPLA	response simulation	ROM & simple control	ATE
BIST2	none	none	none
HFC	count crosspoints	ROM & simple control	none
DAC	response simulation	none	ATE
TSBF	special TG program	ATE | ROM	ATE | ROM & comparator
TBSL	special TG program	ATE	ATE
DCT	special TG program	ATE	ATE
EL	special TG program	ATE	ATE
SMITH	special TG program	ATE	ATE

Figure 14.18 (Continued)

TDM	Extra lines			Extra transistors	Assumptions & remarks
	Bit	Product	Output		
EXH	0	0	0	0	
CONC1	0	0	1	$10n+5k+logm(m+5)$	2-rail checker and 2 EOR trees
CONC2	2	0	C	F.D.	C = the length of check symbols
UTS	0	1	1	$4n+11m+5k+6$	column rank of OR array is k
FIT	0	3	1	$11m+5k+0.5nlogn+29$	
AUTO	0	4	2	$12n+12m+5k+50$	feedback generator is not duplicated
BILBO	0	0	0	$16(2n+m+k)$	
SIG	0	0	0	$6(3n+[k/n])$	
FIST	0	0	0	$12n+6m+8k$	
TLO	0	0	1	$6m$	
BIT	0	2	2	$12n+11m+5k$	PLA can be a sequential circuit
CPC	0	3	1	$6(m+1)$	
LOD	$2C$	0	0	0	C is an integer and needs to be calculated
PTPLA	$2log(m/J)$	0	0	$6J+2log(mJ)$	J is no. of product line groups
BIST2	0	3	1	$11n+6m+3.5k+30$	for n odd; otherwise 2 product lines are added
HFC	0	0	2	$12n+6m+6k+19logm$	
DAC	$2+2J$	0	0	$12n+6J+10log(m/J)$	J is no. of product line groups
TSBF	0	1	1	0	bridging faults are all AND and detectable
TBSL	0	0	1	0	AND array consists of disjoint product terms
DCT	2	2	2	0	PLA is crosspoint irredundant; bridging is AND
EL	0	0	0	0	only missing crosspoint faults are considered
SMITH	0	0	0	0	size of test set is bounded by $n(2m+k)$

Figure 14.18 (Continued)

REFERENCES

[Agarwal 1980] V. K. Agarwal, "Multiple Fault Detection in Programmable Logic Arrays," *IEEE Trans. on Computers*, Vol. C-29, No. 6, pp. 518-522, June, 1980.

[Berger 1961] J. M. Berger, "A Note on Error Detection Codes for Asymmetric Channels," *Inform. Control*, Vol. 4, No. 1, pp. 68-73, March, 1961.

[Bose and Abraham 1982] P. Bose and J. A. Abraham, "Test Generation for Programmable Logic Arrays," *Proc. 19th Design Automation Conf.*, pp. 574-580, June, 1982.

[Boswell *et al.* 1985] C. Boswell, K. Saluja and K. Kinoshita, "A Design of Programmable Logic Arrays for Parallel Testing," *J. Computer Systems Science and Engineering*, Vol. 1, pp. 5-16, October, 1985.

[Bozorgui-Nesbat and McCluskey 1984] S. Bozorgui-Nesbat and E. J. McCluskey, "Lower Overhead Design for Testability for PLAs," *Proc. Intn'l. Test Conf.*, pp. 856-865, October, 1984.

[Brayton *et al.* 1984] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Norwell, MA, 1984.

[Breuer and Saheban 1987] M. A. Breuer and F. Saheban, "Built-In Test for Folded Programmable Logic Arrays," *Microprocessors and Microsystems*, Vol. 11, No. 6, pp. 319-329, July/August, 1987.

[Cha 1978] C. W. Cha, "A Testing Strategy for PLAs," *Proc. 15th Design Automation Conf.*, pp. 326-331, June, 1978.

[Daehn and Mucha 1981] W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," *IEEE Trans. on Computers*, Vol. C-30, No. 11, pp. 829-833, November, 1981.

[Dong 1982] H. Dong, "Modified Berger Codes for Detection of Unidirectional Errors," *Digest of Papers 12th Annual Symp. on Fault-Tolerant Computing*, pp. 317-320, June, 1982.

[Dong and McCluskey 1981] H. Dong and E. J. McCluskey, "Matrix Representation of PLA's and an Application to Characterizing Errors," CRC Technical Report 81-11, Stanford University, September, 1981.

[Dong and McCluskey 1982] H. Dong and E. J. McCluskey, "Concurrent Testing of Programmable Logic Arrays," CRC Technical Report 82-11, Stanford University, June, 1982.

[Eichelberger and Lindbloom 1980] E. B. Eichelberger and E. Lindbloom, "A Heuristic Test-Pattern Generator for Programmable Logic Array," *IBM Journal of Research and Development*, Vol. 24, No. 1, pp. 15-22, January, 1980.

[Fujiwara and Kinoshita 1981] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Tests," *IEEE Trans. on Computers*, Vol. C-30, No. 11, pp. 823-828, November, 1981.

[Fujiwara 1984] H. Fujiwara, "A New PLA Design for Universal Testability," *IEEE Trans. on Computers*, Vol. C-33, No. 8, pp. 745-750, August, 1984.

[Grassl and Pfleiderer 1983] G. Grassl and H-J. Pfleiderer, "A Function-Independent Self-Test for Large Programmable Logic Arrays," *Integration, the VLSI Magazine*, Vol. 1, pp. 71-80, 1983.

[Hassan and McCluskey 1983] S. Z. Hassan and E. J. McCluskey, "Testing PLAs Using Multiple Parallel Signature Analyzers," *Digest of Papers 13th Annual Intn'l. Symp. on Fault-Tolerant Computing*, pp. 422-425, June, 1983.

[Hachtel *et al.* 1982] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "An Algorithm for Optimal PLA Folding," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-1, No. 2, pp. 63-76, April, 1982.

[Hong and Ostapko 1980] S. J. Hong and D. L. Ostapko, "FITPLA: A Programmable Logic Array for Functional Independent Testing," *Digest of Papers 10th Intn'l. Symp. on Fault-Tolerant Computing*, pp. 131-136, October, 1980.

[Hua *et al.* 1984] K. A. Hua, J. Y. Jou, and J. A. Abraham, "Built-In Tests for VLSI Finite-State Machines," *Digest of Papers 14th Intn'l. Symp. on Fault-Tolerant Computing*, pp. 292-297, June, 1984.

[Khakbaz and McCluskey 1981] J. Khakbaz and E. J. McCluskey, "Concurrent Error Detection and Testing for Large PLAs," *IEEE Trans. on Electron Devices*, Vol. ED-29, pp. 756-764, April, 1982.

[Khakbaz 1984] J. Khakbaz, "A Testable PLA Design with Low Overhead and High Fault Coverage," *IEEE Trans. on Computers*, Vol. C-33, No. 8, pp. 743-745, August, 1984.

[Min 1983a] Y. Min, "A Unified Fault Model for Programmable Logic Arrays," CRC Technical Report 83-5, Stanford University, May, 1983.

[Min 1983b] Y. Min, "Generating a Complete Test Set for Programmable Logic Arrays," CRC Technical Report 83-4, Stanford University, May, 1983.

[Muehldorf and Williams 1977] E. I. Muehldorf and T. W. Williams, "Optimized Stuck Fault Test Pattern Generation for PLA Macros," *Digest of Papers 1977 Semiconductor Test Symp.*, pp. 89-101, October, 1977.

[Ostapko and Hong 1979] D. L. Ostapko and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays (PLA's)," *IEEE Trans. on Computers*, Vol. C-28, No. 9, pp. 617-627, September, 1979.

[Pradhan and Son 1980] D. K. Pradhan and K. Son, "The Effect of Undetectable Faults in PLAs and a Design for Testability," *Digest of Papers 1980 Test Conf.*, pp. 359-367, November, 1980.

[Ramanatha and Biswas 1982] K. S. Ramanatha and N. N. Biswas, "A Design for Complete Testability of Programmable Logic Arrays," *Digest of Papers 1982 Intn'l. Test Conf.*, pp. 67-73, November, 1982.

[Ramanatha and Biswas 1983] K. S. Ramanatha and N. N. Biswas, "A Design for Testability of Undetectable Crosspoint Faults in Programmable Logic Arrays," *IEEE Trans. on Computers*, Vol. C-32, No. 6, pp. 551-557, June, 1983.

[Reddy and Ha 1987] S. M. Reddy and D. S. Ha, "A New Approach to the Design of Testable PLA's," *IEEE Trans. on Computers*, Vol. C-36, No. 2, pp. 201-211, February, 1987.

[Saluja and Upadhyaya 1985] K. K. Saluja and J. S. Upadhyaya, "Divide and Conquer Strategy for Testable Design of Programmable Logic Arrays," *Proc. 4th Australian Microelectronics Conf.*, May, 1985.

[Saluja *et al.* 1983] K. K. Saluja, K. Kinoshita and H. Fujiwara, "An Easily Testable Design of Programmable Logic Arrays for Multiple Faults," *IEEE Trans. on Computers*, Vol. C-32, No. 11, pp. 1038-1046, November, 1983.

[Saluja *et al.* 1985] K. K. Saluja, H. Fujiwara, and K. Kinoshita, "Testable Design of Programmable Logic Arrays with Universal Control and Minimal Overhead," *Proc. Intn'l. Test Conf.*, pp. 574-582, 1985. Also in *Intn'l. Journal of Computers and Mathematics with Applications*, Vol. 13, No. 5/6, pp. 503-517, February, 1987.

[Smith 1979] J. E. Smith, "Detection of Faults in Programmable Logic Arrays," *IEEE Trans. on Computers*, Vol. C-28, No. 11, pp. 848-853, November, 1979.

[Somenzi *et al.* 1984] F. Somenzi, S. Gai, M. Mezzalamo, and P. Prinetto, "PART: Programmable Array Testing Based on a Partitioning Algorithm," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-3, No. 2, pp. 142-149, April, 1984.

[Son and Pradhan 1980] K. Son and D. K. Pradhan, "Design of Programmable Logic Arrays for Testability," *Digest of Papers 1980 Test Conf.*, pp. 163-166, November, 1980.

[Treuer *et al.* 1985] R. Treuer, H. Fujiwara, and V. K. Agrawal, "Implementing a Built-In Self-Test PLA Design," *IEEE Design & Test of Computers*, Vol. 2, No. 2, pp. 37-48, April, 1985.

[Upadhyaya and Saluja 1988] J. S. Upadhyaya and K. K. Saluja, "A New Approach to the Design of Built-In Self-Testing PLAs for High Fault Coverage," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 1, pp. 60-67, January, 1988.

[Wakerly 1978] J. Wakerly, *Error Detecting Codes, Self Checking Circuits and Applications*, American-Elsevier, New York, 1978.

[Wang and Avizienis 1979] S. L. Wang and A. Avizienis, "The Design of Totally Self-Checking Circuits Using Programmable Logic Arrays," *Digest of Papers 9th Annual Intn'l. Symp. on Fault-Tolerant Computing*, pp. 173-180, June, 1979.

[Wei and Sangiovanni-Vincentelli 1985] R-S. Wei and A. Sangiovanni-Vincentelli, "PLATYPUS: A PLA Test Pattern Generation Tool," *Proc. 22nd Design Automation Conf.*, pp. 197-203, June, 1985.

[Yajima and Aramaki 1981] S. Yajima and T. Aramaki, "Autonomously Testable Programmable Logic Arrays," *Digest of Papers 11th Annual Intn'l. Symp. on Fault-Tolerant Computing*, pp. 41-43, June, 1981.

[Zhu and Breuer 1986a] X. Zhu and M. A. Breuer, "A Survey of Testable PLA Designs," *IEEE Design & Test of Computers*, Vol. 5, No. 4, pp. 14-28, August, 1988.

[Zhu and Breuer 1986b] X. Zhu and M. A. Breuer, "A Knowledge Based TDM Selection System," *Proc. 1986 Fall Joint Computer Conf.*, pp. 854-863, November, 1986.

PROBLEMS

14.1 Draw a PLA representation for the sum and carry functions over three variables.

14.2 For the PLA shown in Figure 14.3, indicate an extra crosspoint fault that cannot be modeled as a s-a fault in the sum of product representation of f_1 or f_2.

14.3 Prove that a missing crosspoint fault is equivalent to some stuck-at fault in the sum-of-product representation of a PLA.

14.4 Describe a test generation procedure, similar to the one presented in this chapter for missing crosspoints, for detecting shrinkage, appearance, and disappearance crosspoint faults.

14.5 Modify the PLA of Example 1 so that no two product lines are activated by any input vector.

14.6 Construct a testable PLA for the PLA of Example 1 using the concurrent error detection by a series of checkers technique discussed in Section 14.4.1.1. For every possible checkpoint fault that can occur to product line P_3, construct an input test that will detect the fault and indicate the values on the error-indicators. Show the output error-detection code used.

14.7 Complete the design of the PLA of Example 1 using the universal test set method described in Section 14.4.2.1. Construct a table of all the test vectors in the universal test set, and indicate next to each vector which faults are detected. Use the same format for the test vectors shown in the text.

14.8 For the UTS technique listed in Figure 14.17 compute the information shown in Figure 14.18. Assume $n = 32$, $m = 64$, and $k = 16$.

14.9 Consider the PLA design shown in Figure 14.3(a). Construct the Karnaugh graph for f_1 and f_2 for each of the following situations, and indicate the product terms implemented by the PLA.

(a) The original PLA;

(b) A shrinkage fault caused by a connection between bit line b_1 and product line P_1;

(c) A shrinkage fault caused by a connection between bit line b_6 and P_1;

(d) A growth fault caused by the missing connection between bit line b_1 and P_6;

(e) An appearance fault caused by an extra connection between P_4 and f_1;

(f) A disappearance fault caused by the missing connection between P_6 and f_2.

14.10 Explain how a shrinkage fault can be functionally equivalent to a disappearance fault.

14.11 Let $f_1 = x_1x_2$ and $f_2 = x_1x_2x_3$. If these functions were implemented using a PLA, the requirement for concurrent testability is violated because both product terms can be activated at the same time. Show how to reimplement f_1 or f_2 so that only one product term is activated at a time. Estimate the area penalty for this modification.