A Tutorial on Built-In Self-Test

Part 1: Principles

DURING ITS LIFETIME, a digital system is tested and diagnosed on numerous occasions. For the system to perform its intended mission with high availability, testing and diagnosis must be quick and effective. A sensible way to ensure this is to specify test as one of the system functions—in other words, self-test. Digital systems involve a hierarchy of parts: chips, boards, cabinets, and so on. At the highest level, which may include the entire system, the operation is controlled by software. Self-test is often implemented in software. While a purely software approach to self-test may suffice at the system level, it has several disadvantages. Such testing may have poor diagnostic resolution because it must test parts designed without specific testability considerations. In addition, a good software test can be very long, slow, and expensive to develop.

An increasingly attractive alternative is built-in self-test—that is, self-test implemented in the hardware itself:

BIST is a design-for-testability (DFT) technique in which testing (test generation and test application) is accomplished through built-in hardware features.

VISHWANI D. AGRAWAL

AT&T Bell Laboratories

CHARLES R. KIME

KEWAL K. SALUJA

University of Wisconsin,

Madison

This tutorial provides an overview of built-in self-test (BIST) principles and practices. In Part 1, the authors present the issues and economics underlying BIST and introduce the related hierarchical test structures. They explain the fundamental BIST concepts of pattern generation and response analysis. A discussion of LFSR theory supplements Part 1. Part 2, in our June issue, will cover BIST hardware implementations, applications, and tools.

The simplicity of this definition belies the complexities involved in implementing BIST. This article addresses the pertinent issues and describes the advantages and limitations of BIST.

In the several years since this magazine's publication of a pair of tutorial articles on BIST,[1,2] both BIST research and its application have grown rapidly. Although our original goal was to write a detailed tutorial, we found that adequate coverage of the myriad of techniques available was not feasible within our space limitations. Hence, on some aspects of BIST we present limited detail, supported by pointers to the literature. Also, to limit the number of sources the interested reader needs to consult, we often refer to books rather than original papers. In no way do we intend to diminish the contributions of the original researchers or developers.

Motivations for BIST

When testing is built into the hardware, it has the potential of being not only fast and efficient but also hierarchical. In other words, in a well-designed testing strategy, the same hardware can test chips, boards, and system. The cost benefits, which may not seem significant at the chip level, are enormous at the system level. Alternative strategies are chip-wise and system-foolish. Moreover, BIST offers solutions to several major testing problems.

Table 1. BIST costs.

	Design, test development	Fabrication	Testing	Maintenance test	Diagnosis, repair	Service interruption
Chips	+/−	+	−			
Boards	+/−	+	−		−	
Systems	+/−	+	−	−	−	−

+ cost increase; − cost reduction (saving); +/− cost increase ≈ saving

The complexity issue. As the complexity of VLSI systems increases, we ask if the testing problem can be partitioned. The answer, unfortunately, is no. For example, consider two devices connected in a cascade. There is often no simple way to derive tests for the cascade from the given tests for its individual parts. Another possibility is the use of a hierarchical approach. The complex design automation problems of synthesis and physical design are often solved through hierarchical procedures. The testing problem, however, is not easy to solve with traditional hierarchical techniques. For example, no simple method exists for deriving a board test from tests for chips on the board.

BIST, however, does offer a hierarchical solution to the testing problem. Consider the testing of a chip embedded in a board that is a part of a system. The top-down hierarchy consists of system, boards, and chips. Suppose all levels of the hierarchy use BIST. To test the chip, the system sends a control signal to the board, which in turn activates self-test on the chip and passes the result back to the system. Thus, BIST provides efficient testing of the embedded components and interconnections, reducing the burden on system-level test, which need only verify the components' functional synergy.

The quality issue. A product's quality depends on the tenacity of its tests. Test tenacity or ability is most frequently measured as coverage of single stuck-at faults. Thus, we calibrate tests according to their ability to detect single lines that appear as if shorted to ground (stuck-at-0) or to the power supply (stuck-at-1). Since the kind and number of faults that occur depends on the type of device (chip, board, and so on) and the technology (CMOS, bipolar, GaAs), evaluating test quality can be a complicated task.[3] In general, quality requirements such as 95% fault coverage for complex VLSI chips or 100% coverage of all interconnect faults on a printed circuit board (PCB) are based on practical considerations. The test engineer tries to achieve a low *reject ratio* (percentage of faulty parts in the number passing the test)—for example, 1 in 10,000—while controlling the cost of test generation and application. For very large systems, such requirements are achievable only through DFT. Our discussion will show that BIST is the preferred form of DFT.

Test generation problem. As pointed out earlier, the problem of generating tests is difficult to solve by using hierarchy. The difficulty lies in carrying the test stimulus through many layers of circuitry to the element under test and then conveying the result again through many layers of circuitry to an observable point. BIST simplifies this problem by localizing testing.

Test application problem. For almost a decade, in-circuit testing (ICT) has dominated the PCB testing scene.[4] In this method, a bed-of-nails fixture cus-tomized for the board under test enables the tester to access the pins of the chips mounted on the board. ICT effectively applies chip tests for diagnosis and also effectively tests board wiring. The method, however, presents several problems. First, ICT is effective only after a board is removed from the system; therefore, it is no help in system-level diagnosis. Second, in surface-mount technology (SMT), components are often mounted densely on both sides of the board. Bed-of-nails fixtures for such boards are either too expensive or impossible to build.

BIST offers a superior solution to the test application problem. First, built-in test circuitry can test chips, boards, and the entire system without expensive, external automatic test equipment. Second, for off-line testing of boards and chips and for production testing, we can use the same tests and test circuitry that we use at the system level.

Economics of BIST. In deciding whether to use BIST, system planners and designers must weigh costs against benefits. At the chip level, BIST offers small savings in testing costs. But in product life-cycle costs, the savings are overwhelmingly in favor of BIST.

Table 1 shows the impact of BIST on testing costs for chips, boards, and systems. We find that the additional expense of designing BIST hardware is somewhat balanced by the savings from test generation. Fabrication cost increases at all levels due to the extra hardware

BIST requires. Testing cost decreases due to more-efficient tests, less-expensive test equipment, and improved trouble-shooting during assembly and integration. Maintenance test is a system-level function involving "sanity checks" and diagnosis. Thus, BIST's impact on maintenance cost is greatest at the system operation level. BIST also reduces diagnosis and repair costs at the board and system levels. In alternative strategies, lengthy or improper diagnosis is often responsible for great loss of revenue due to service interruption; BIST decreases such interruption.

The main point of Table 1 is the significant benefit that BIST provides at the system level. Thus, even with considerably lower benefits at chip and board levels, we believe BIST is still the best DFT alternative. On this point the reader should consult pertinent works on BIST economics.[5,6]

BIST concepts

In considering the concepts underlying BIST, we must look at the basic BIST architecture and its hierarchical application. Then we will focus on two specific BIST components: pattern generation and response analysis.

BIST architecture. The basic BIST architecture requires the addition of three hardware blocks to a digital circuit: a pattern generator, a response analyzer, and a test controller. Examples of pattern generators are a ROM with stored patterns, a counter, and a linear feedback shift register (LFSR). A typical response analyzer is a comparator with stored responses or an LFSR used as a signature analyzer. A control block is necessary to activate the test and analyze the responses. However, in general, several test-related functions can be executed through a test manager (or test controller) circuit.

Consider a hierarchical application of the BIST concept. The system consists of several circuit boards. Each board may contain several VLSI chips. Figure 1 shows such a system. The test manager at the system level can simultaneously activate self-test on all boards. The test manager on each board, in turn, activates self-test on each chip on that board. A chip test manager is responsible for executing self-test on the chip and then transmitting the result (fault-free or faulty) to the test manager of the board containing the chip. The board test manager accumulates test results from all its chips and transmits them to the system test manager. Using these results, the system test manager can isolate faulty chips and boards.

The effectiveness of this diagnosis procedure depends on the thoroughness of the self-test implemented on chips. Thus, fault coverage is a major issue in BIST designs. Other important issues are area overhead and its impact on chip yield, additional pins required for test, and performance penalty.

At the chip level, BIST involves the application of test patterns to the logic to be tested and observation of the corresponding responses. Often, the test engineer modifies on-chip logic, using some DFT technique such as scan, so that latches and flip-flops can be controlled independently of the circuit's combinational logic. Thus, in most but not all cases, the circuit under test (CUT) consists of combinational logic. However, logic may intervene between the pattern generator and the CUT and between the CUT and the response analyzer, as indicated by the shaded area in Figure 1.

Pattern generation. We now discuss BIST test pattern types, the means of obtaining them, and related fault coverage issues. Distinct BIST methodologies are associated with each type of test pattern.

Stored patterns. Stored-pattern BIST may use programs or microprograms, typically stored in ROM, to perform func-
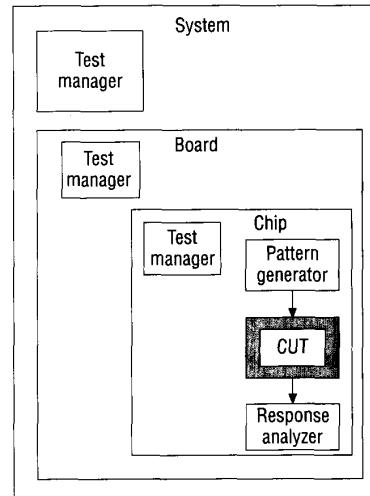
Figure 1. BIST hierarchy.

tional tests of the hardware. Successful applications of such techniques exist,[7] but they are not our focus here. In alternative techniques, we use traditional automatic test pattern generation (ATPG) and fault simulation to generate the test patterns. We store the patterns on the chip or board, apply them to the CUT when BIST is activated, and compare the CUT responses with the corresponding stored responses. Because of the stored data's magnitude, this method is attractive only in limited cases. These include testing structured logic and detecting a small number of faults not handled by other BIST techniques. Overall, although stored-pattern BIST can provide excellent fault coverage, it has limited applicability due to its high area overhead.

Exhaustive or pseudoexhaustive patterns. Exhaustive-pattern BIST eliminates the test generation process and has very high fault coverage. To test an n-input block of combinational logic, we apply all possible 2^n-input patterns to the block. Even with high clock speeds, the time required to apply the patterns may make exhaustive-pattern BIST impracti-

Table 2. Hardware structures used for BIST.

Pattern generators	Response analyzers
ROM	ROM and comparison logic
LFSR	LFSR
	Multiple-input signature register (MISR)
Cellular automaton	Cellular automaton
Binary counter	Level counter
	Transition counter
XOR trees	XOR trees

Note: Each pattern generator in the left column can be used with any response analyzer in the right column.

cal for a circuit with n greater than about 25. Thus, we must partition or segment the logic into smaller, possibly overlapping blocks with fewer than n inputs. Then we exhaustively test each block. This approach is called pseudoexhaustive-pattern BIST.[8,9, p. 461]

Fault coverage for the exhaustive or the pseudoexhaustive method is nearly 100% and, with proper design, can be achieved without fault simulation. Exhaustive testing detects all detectable faults that do not induce sequential behavior within each block. Extensive circuit partitioning and segmentation may require significant effort, and the added hardware to achieve such partitioning and segmentation can be expensive. The added hardware may also adversely affect performance if avoiding critical timing paths becomes impossible. We can keep test application time reasonable by choosing suitably small values of n for blocks that can be tested in parallel.

Pseudorandom patterns. In contrast with other methods, pseudorandom-pattern BIST may require a long test time and necessitate evaluation of fault coverage by fault simulation. This pattern type, however, has the potential for lower hardware and performance overheads and less design effort than the preceding methods. In pseudorandom test patterns, each bit has an approxi-

mately equal probability of being a 0 or a 1 (as well as other statistical properties not detailed here[10]). The number of patterns applied is typically of the order of 10^3 to 10^7 and is related to the circuit's testability and the fault coverage required.

Among the pattern types discussed so far, the exhaustive and the pseudorandom are the most frequently used. Applying exhaustive patterns for a portion of a VLSI circuit is comparatively straightforward, but applying pseudorandom patterns for a portion or all of the circuit is considerably more complex. These are the principal related issues: 1) How do we determine the number of pseudorandom test patterns to apply? 2) How do we evaluate fault coverage? 3) How do we deal with residual uncovered faults, often referred to as *hard-to-detect* or *random-pattern-resistant* faults?

Research has demonstrated that one can estimate the number of pseudorandom patterns required for a circuit from information based on the desired fault coverage and on either the set of hard-to-detect faults[11] or circuit testability.[12] The number of patterns can be fairly large; 1,000,000 is not uncommon. To perform exact fault simulation for such a large pattern set, a fast fault simulation technique is essential. For combinational circuits, parallel-pattern single-fault propagation (PPSFP) is such a technique.[13, p. 112]

If the test length is too long to be practical (more than a few million vectors), one can deal with some of the hard-to-detect faults by other means. One approach is to use deterministic patterns generated by ATPG for detection of these faults. Another is to modify the combinational logic to improve testability.[13, p. 93]

Weighted pseudorandom patterns. A hybrid between pseudorandom and stored-pattern BIST, weighted pseudorandom-pattern BIST is effective for dealing with hard-to-detect faults. In a pseudorandom test, each input bit has a probability of 1/2 of being either a 0 or a 1. In a weighted pseudorandom test, the probabilities, or input weights, can differ. The essence of weighted pseudorandom testing is to bias the probabilities of the input bits so that the tests needed for hard-to-detect faults are more likely to occur. One approach uses software that determines a single- or multiple-weight set based on a probabilistic analysis of the hard-to-detect faults.[13, p. 142] Another approach uses a heuristic-based initial weight set followed by additional weight sets produced with the help of an ATPG system.[13, p. 150] The weights are either realized by logic or stored in on-chip ROM. With these techniques, researchers obtained fault coverage over 98% for 10 designs, which is the same as the coverage of deterministic test vectors.[13, p. 159]

BIST test patterns are generated by a variety of hardware structures, as shown in the left half of Table 2. The most prevalent approach for exhaustive, pseudoexhaustive, and pseudorandom patterns is the use of an LFSR. We discuss the theory of LFSRs and their application to both pattern generation and response analysis in the box on page 79. An alternative pattern generator is the cellular automaton,[14] in which each cell, consisting of a flip-flop and a few gates, is connected only to its neighboring cells. Advantages claimed for this pattern generator are that it has only local connections between cells and that

it produces patterns more like true random patterns than those from an LFSR.

Researchers are focusing on new techniques for producing shorter BIST test pattern sequences than are achievable by the usual pseudorandom techniques. Their approach is to produce specific designs for BIST pattern generators using LFSRs or cellular automata. These generators produce sequences that include a set of deterministically generated test patterns for the CUT.[15-17]

Response analysis. The right half of Table 2 shows the hardware structures that we can employ to determine the validity of a CUT's outputs. Clearly, when we apply test patterns to test the CUT, we must know its fault-free response(s). For a given set of test vectors applied in a particular order, we can obtain the expected responses and their order from a "gold" (known-good) CUT or by simulating the CUT. Similar to stored-pattern BIST, we can also store responses in on-chip ROM, but such a scheme can require too much silicon area to be of practical value. Alternatively, methods that compress the test patterns and the corresponding responses of a fault-free CUT and regenerate them during self-test are also of limited value for general VLSI circuits.

An alternative to response compression is compaction of responses into a relatively short binary sequence(s) called a signature(s). Let us explain the difference between compression and compaction: Compression is lossless in the sense that the original sequence can be fully regenerated from the compressed sequence. In the case of compaction, regenerating the original sequence from the compacted sequence may not be possible. For the more mathematically-minded reader, it suffices to say that the compression function is invertible, whereas the compaction function is not. In the following paragraphs, we explain in abstract terms the basic concept of compaction as
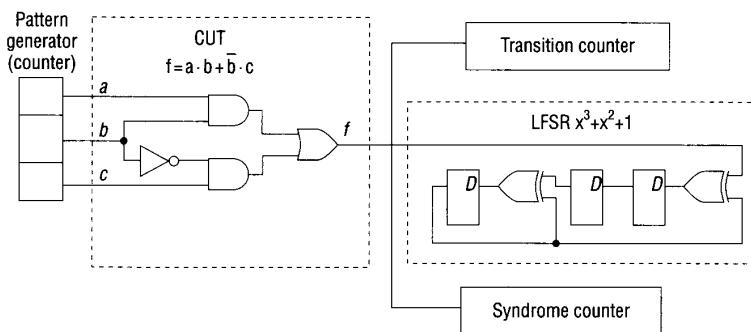
used in self-test and then describe proposed compaction methods and their relative strengths and weaknesses.

After obtaining a response sequence R for a given order of test vectors from a gold CUT or a simulator, we use a compaction function C to produce a vector or a set of vectors $C(R)$. We expect the number of bits in $C(R)$ to be far fewer than the number in R. We store the compacted vectors on chip or off chip, and, during BIST, we use the compaction function C to compact the CUT's actual responses R^* to provide $C(R^*)$. Finally, to determine the CUT's status (fault-free or faulty), we compare $C(R)$ and $C(R^*)$. We declare the CUT fault-free if these two values are identical.

For compaction to be of practical value, the function C should be simple to implement on chip, the compacted responses should be sufficiently small, and, above all, a faulty CUT should not be declared fault-free. If a faulty circuit and the fault-free circuit provide different response sequences but the compacted response sequences are identical, *aliasing* has occurred.

The BIST literature has described three compaction functions in detail. To explain these methods, we use a simple CUT consisting of a three-variable combinational function. Figure 2 shows a realization of the function. In describing these methods, we assume the pattern generator is a counter that generates all

eight input vectors for testing the CUT.

One of the first compaction functions proposed in the context of testing was the *transition count*.[11, p. 94] This function counts the total number of 0-to-1 and 1-to-0 transitions in the response stream. For the example function in Figure 2, the transition count value for the fault-free circuit is three, as shown in Table 3 (next page). The table also shows transition count values for three different faults in the circuit.

The *signature analysis* function was first used by Hewlett-Packard as a compaction function and was described by Frohwerk in 1977.[11, p. 100] In this method, the response sequence is fed to an LFSR. The compacted sequence, whose length is the same as that of the LFSR, is called the signature of the CUT for the applied test vector sequence. Table 3 (next page) summarizes the signatures of the example circuit and three faulty circuits. A special case of this method, parity check compaction, uses an LFSR of length one.[11, p. 97]

The *syndrome* or *1's counting* function was proposed in 1980.[11, p. 102] This compaction function counts the total number of 1's in the response sequence (the total may be normalized with respect to the length of the response sequence). The total is called the syndrome of the CUT. Table 3 lists the syndromes of the example function and the three faulty functions.

Figure 2. *BIST of a three-variable function.*

Table 3. Analysis of circuit in Figure 2.

abc	Fault-free	a stuck-at-1	f stuck-at-1	b stuck-at-1
000	0	0	1	0
001	1	1	1	0
010	0	1	1	0
011	0	1	1	0
100	0	0	1	1
101	1	1	1	1
110	1	1	1	1
111	1	1	1	1
Function				
Transition count	3	3	0	1
Signature analysis	001	101	001	010
Syndrome	4	6	8	4

It is evident from Table 3 that all three compaction functions are prone to aliasing. Detection of aliasing is a far more expensive process than fault simulation because it entails computing $C(R)$ for every fault in the fault list. That typically requires simulating every fault for every test vector without fault dropping. Hence, researchers have proposed models to compute aliasing probability analytically.[13, p. 69] These models often make assumptions about the occurrence of errors in the CUT's output sequence. No reasonable models, however, relate errors to faults in a circuit. Generally speaking, the three compaction methods have identical aliasing probabilities, which decrease exponentially with an increase in the test sequence length or the number of bits in $C(R)$.

Our description of the three methods seems to imply that compaction is applicable only to single-output CUTs. This is not so. We can extend the transition count and syndrome functions to multiple-output CUTs by assigning different weights to the outputs and thus obtaining a weighted compaction of the output sequence. Saxena and Robinson present a generalization of the transition count and syndrome testing methods for multiple-output CUTs.[18]

We can also use a *space compactor*, typically a linear circuit, to reduce the number of outputs to be compacted.[19] For the signature method, we integrate the linear circuit with the LFSR to obtain a *multiple-input linear feedback shift register* (MISR), which compacts the output sequences from a multiple-output CUT. An MISR can be viewed as performing space compaction while compacting the output sequences from a CUT.

Once again aliasing raises its ugly head, posing a problem for multiple-output CUTs. Researchers have attempted to analyze the aliasing probability for compacted multiple-output responses[20,21] and to reduce that probability. Some of the methods proposed to reduce aliasing are compaction-testable designs,[9, p. 431] multiple signatures,[22] output data modification,[23] and rearranging test vectors.[24]

By far, the most popular compaction function is signature analysis, realized by means of an LFSR or an MISR. These structures are easy to implement, and because they are serially scannable, they can be read out easily by an external tester at the completion of self-test.

Finally, there has been study of the use of the cellular automaton for response analysis as well as for pattern generation.[11] This structure's effectiveness in BIST environments is yet to be fully established.

WE HAVE INTRODUCED BIST in the context of its application not only to chips but to systems by use of a hierarchical BIST architecture. This approach appears to be expanding rapidly in commercial products. For example, extensive use of BIST in workstation products has recently been reported. The development of hierarchical BIST applications is likely to accelerate as the use of surface-mount technology further limits conventional board testing methods. The BIST solution's positive economic impact at the system level motivates exploration of this hierarchical approach.

Both conventional and hierarchical BIST employ the same fundamental concepts: pattern generation, response analysis, and test management. The pattern generation and response analysis techniques we detailed in Part 1 differ little in conventional and hierarchical approaches. Most contemporary pattern generation and response analysis implementations are based on LFSRs. Thus, the basic understanding of LFSR theory presented here is useful to the BIST designer.

In Part 2, we will examine hardware implementations of BIST structures based on the concepts introduced here, and we will discuss several real-world BIST applications. Finally, we will describe CAD tools critical to the production of correct, efficient, and effective BIST designs. ◄D&T►

Acknowledgment

LFSR theory

Linear feedback shift registers are widely used in BIST because they are simple and fairly regular in structure, their shift property integrates easily with serial scan, and they can generate exhaustive and/or pseudorandom patterns. The typical components of an LFSR are D flip-flops and XOR gates. Despite their simple appearance, LFSRs are based on a rather complex mathematical theory.[1] Here we present only the aspects of the theory that help explain their behavior as pattern generators and response analyzers.

Figure A shows two example LFSRs. Both use D flip-flops and linear logic elements (XOR gates). Their basic difference is that the Figure A1 circuit uses XORs between flip-flops, whereas the Figure A2 circuit does not; instead the XORs appear only in the feedback path. For this reason we call the Figure A1 realization an *internal-XOR* LFSR and the Figure A2 realization an *external-XOR* LFSR. The two types are equivalent in the sense that, knowing the properties of the first structure, we can deduce the properties of the second.[2] We will concentrate on the behavior of the first type of structures.

In test pattern generation mode, a pattern generated by an LFSR is the state of all the D flip-flops in the LFSR. Obviously, we can deduce consecutive patterns generated by an LFSR by simulating it. But by associating polynomials with LFSRs and bit streams or vectors, we can use polynomial algebra to *predict* LFSR behavior. Throughout this discussion we discuss polynomials with binary coefficients, but almost all the results can be stated in more general terms.
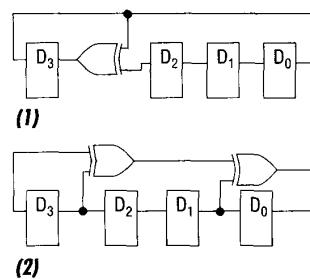
We can express a binary vector $R = r_m r_{m-1} \dots r_0$ as a polynomial $r_m x^m + r_{m-1} x^{m-1} + \dots + r_0$. For example, a vector 10111 can be written $x^4 + x^2 + x + 1$. The superscript of the highest nonzero term in a polynomial is called the *degree of the polynomial*. We can perform arithmetic on polynomials, just as we can on integers. For example, consider two polynomials, $q(x) = x^3 + x^2 + 1$ of degree 3, and $p(x) = x^2 + x + 1$ of degree 2. Then:

$$p(x) + q(x) = x^3 + 2x^2 + x + 2 = x^3 + x$$

since coefficients are added modulo 2. Similarly,

$$p(x) \cdot q(x) = x^5 + 2x^4 + 2x^3 + 2x^2 + x + 1 = x^5 + x + 1$$

We can also express polynomials *mod* a polynomial. That is, two polynomials $r(x)$ and $s(x)$ are congruent modulo $n(x)$, written as $r(x) \equiv s(x) \bmod n(x)$ if there is a polynomial $q(x)$ such that $r(x) = n(x) \cdot q(x) + s(x)$. As with integers, we find the least positive residue in polynomials by dividing $r(x)$ by $n(x)$ and taking the remainder. Thus, we can perform modulus arithmetic on polynomials as we do with integers. For example:

$$(x^2 + x + 1) \cdot (x^3 + 1) \equiv (x^5 + x^4 + x^3 + x^2 + x + 1) \bmod (x^4 + x)$$

Performing the required division, we get

$$
\begin{array}{r}
x + 1 \\
x^4 + x \overline{\smash{\big)}\, x^5 + x^4 + x^3 + x^2 + x + 1} \\
\underline{x^5 \qquad\qquad x^2} \\
x^4 + x^3 \qquad + x \\
\underline{x^4 \qquad\qquad + x} \\
x^3 \qquad\qquad + 1
\end{array}
$$

Therefore, $(x^2 + x + 1) \cdot (x^3 + 1) \equiv x^3 + 1 \bmod (x^4 + x)$. Similar to prime numbers (numbers that cannot be factored) for integers, we can define polynomials that cannot be factored. Such polynomials are called *irreducible*. For example, the polynomials $a(x) = x^4 + x^3 + x^2 + x + 1$ and $b(x) = x^4 + x^3 + 1$ are two irreducible polynomials of degree 4. Irreducible polynomials help define an algebraic structure called a *field*. Although the general study of LFSRs requires an understanding of fields, we will forgo the details for lack of space. Let $p(x)$ be a polynomial of degree n, and let us compute $x, x^2, x^3, \dots \bmod p(x)$. Clearly all these polynomials will be of degree less than that of $p(x)$.

For a special type of polynomial $p(x)$, while computing increasing powers of $x \bmod p(x)$, we obtain all possible nonzero polynomials of degree less than that of $p(x)$—that is, $2^n - 1$ distinct nonzero polynomials. Such a polynomial $p(x)$ is called *primitive*. Let us clarify this through an example by computing $x, x^2, x^3, \dots \bmod a(x)$. The sequence we obtain is $x, x^2, x^3, x^4 = x^3 + x^2 + x + 1, x^5 = 1, x^6 = x$. We can now conclude that the succeeding powers of x will generate the same remainders over and over. Thus, in this case, succeeding powers of x generate only five distinct polynomials. On the other hand, if we repeat the same process for the polynomial $b(x)$, we obtain all 15 nonzero polynomials of degree less than 4.

Figure A. Example LFSRs: internal-XOR (1); external-XOR (2).

(1)

(2)

LFSR theory (continued)

Thus, the polynomial $b(x)$ is primitive, and the polynomial $a(x)$ is not primitive, although both are irreducible.

What is the relation between polynomial algebra and LFSRs? Like a binary vector, an LFSR can also be expressed as a polynomial. Figure B is a general representation of an internal-XOR LFSR. This LFSR is represented by a polynomial $g(x) = x^n + g_{n-1}x^{n-1} + \dots + g_0$. We call $g(x)$ the LFSR's *characteristic polynomial*. Thus, the characteristic polynomial of the LFSR in Figure A1 is $x^4 + x^3 + 1$. Furthermore, the contents of an LFSR, being a binary vector, can also be expressed by a polynomial of degree less than the degree of its characteristic polynomial.

We also know that a left shift of a binary vector in the polynomial representation of vectors is equivalent to multiplication by x. In the case of LFSRs, a left shift of an LFSR is equivalent to multiplying its contents by x, then computing its value mod the characteristic polynomial of the LFSR. To explain this through an example, let us assume that the content of the LFSR of Figure A1 is $1010 = x^3 + x$. Multiplying it by x gives $x^4 + x^2 \equiv x^3 + x^2 + 1 \bmod (x^4 + x^3 + 1)$. The reader can verify this result by manually simulating Figure A1's LFSR beginning in initial state 1010.

From the preceding discussion and example, we generalize that if an LFSR of characteristic polynomial $p(x)$ of degree n is initialized to x—in other words, the initial state of the LFSR is 00 … 010—then on consecutive shifts the contents of the LFSR will be $x^2 \bmod p(x)$, $x^3 \bmod p(x)$, …, and so on. If $p(x)$ is a primitive polynomial, the vectors generated by the LFSR will be all possible 2^n-1 nonzero vectors. Hence, such an LFSR can serve as an exhaustive (almost) test pattern generator for a CUT of n inputs. In fact, we can make a stronger statement: *An LFSR with a char-*
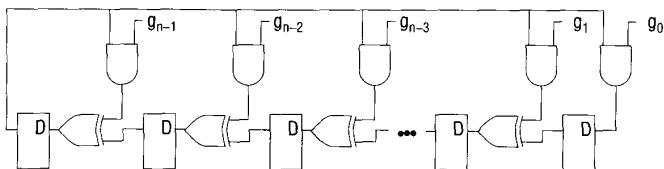
Figure B. *A general representation of an internal-XOR LFSR.*
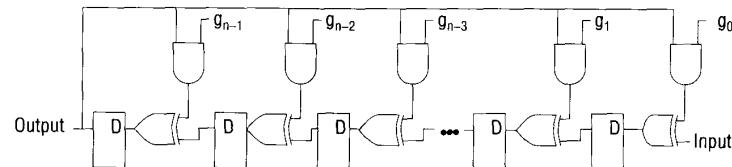
Figure C. *An LFSR for dividing a polynomial.*
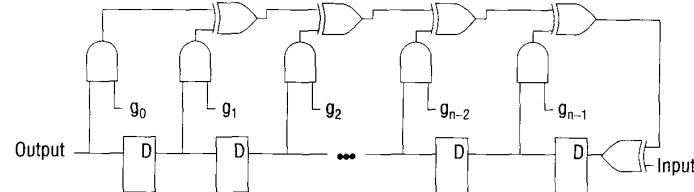
Figure D. *An external-XOR LFSR for dividing a polynomial.*

acteristic polynomial $g(x)$ of degree n will generate all possible 2^n-1 nonzero vectors if and only if $g(x)$ is a primitive polynomial.

Yet another characteristic of the vectors generated by an LFSR is that they appear to be randomly ordered.[1] In fact, they satisfy most of the properties of random numbers even though we can predict them deterministically from the LFSR's present state and its characteristic polynomial. Therefore, we call these vectors *pseudorandom vectors*.

An LFSR modified to accept an external input, as shown in Figure C, acts as a polynomial divider. It divides the input sequence, represented by a polynomial, by the characteristic polynomial $g(x)$ of the LFSR. As this division proceeds bit by bit, the quotient sequence appears at the output of the LFSR and the remainder appears in the LFSR with every shift of the input sequence into the LFSR. Notice that the input

polynomial is shifted with the highest degree coefficient first (remember, division starts from the highest degree end). The polynomial divider structure of the LFSR often is used as a signature analyzer. The reader can verify that for the example in Figure 2 and Table 3, all the signatures are indeed correct. Error correction coding (ECC) applications make extensive use of the division property of LFSRs and the analogy between irreducible polynomials and prime numbers.[2]

Before concluding our discussion of LFSRs, we must comment on the relation between internal-XOR and external-XOR LFSRs. There is a one-to-one correspondence between the behaviors of the two types. The characteristic polynomial for the external-XOR LFSR shown in Figure D is $g(x) = x^n + g_{n-1}^{n-1} + \dots + g_0$. Notice that the coefficients g_i

LFSR theory (continued)

in this figure are labeled differently from the preceding two figures. An external-XOR LFSR also acts as a polynomial divider and produces the correct quotient bit sequence. However, the LFSR's contents is not the remainder as it is with the internal-XOR LFSR. Readers familiar with the theory and design of sequential circuits can draw an analogy that the two LFSRs are different realizations of the same state table using two different state assignments.

Both internal-XOR and external-XOR LFSRs have only one external input. They can be modified to obtain three different realizations of multiple-input LFSRs (or multiple-input signature registers, MISRs), as shown in Figure E. For theoretical analysis, we can reduce each realization to an equivalent sin-

gle-input LFSR by rearranging the inputs.[3] We achieve the rearrangement by means of the commutative property of the XOR operator and the fact that a flip-flop's input and output are related by a shift or delay operator.

References

1. S.W. Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, Calif., 1982.
2. W.W. Peterson and E.J. Weldon, Jr., *Error-Correcting Codes*, John Wiley & Sons, New York, 1972.
3. T. Sridhar et al., "Analysis and Simulation of Parallel Signature Analyzers," *Int'l J. Computers and Mathematics with Applications*, Vol. 13, No. 5/6, Feb. 1987, pp. 537-545.

Figure E. *Three multiple-input LFSRs, or multiple-input signature registers (MISRs).*

References

1. E.J. McCluskey, "Built-In Self-Test Techniques," *IEEE Design & Test of Computers*, Vol. 2, No. 2, Apr. 1985, pp. 21-28.
2. E.J. McCluskey, "Built-In Self-Test Structures," *IEEE Design & Test of Computers*, Vol. 2, No. 2, Apr. 1985, pp. 29-36.
3. S.C. Seth and V.D. Agrawal, "Characterizing the LSI Yield Equation from Wafer Test Data," *IEEE Trans. Computer-Aided Design*, Vol. CAD-3, No. 4, Apr. 1984, pp. 123-126.
4. I. Bateson, *In-Circuit Testing*, Van Nostrand Reinhold, New York, 1985.
5. A.P. Ambler et al., "Economically Viable Automatic Insertion of Self-Test Features for Custom VLSI," *Proc. Int'l Test Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., Sept. 1986, pp. 232-243.

6. I.D. Dear, "Economic Effects in Design and Test" *IEEE Design & Test of Computers*, Vol. 8, No. 4, Dec. 1991, pp. 64-77.
7. J. Kuban and W. Bruce, "Self-Testing the Motorola MC6804P2," *IEEE Design & Test of Computers*, Vol. 1, No. 2, May 1984, pp. 33-41.
8. E. Wu, "PEST: A Tool for Implementing Pseudo-Exhaustive Self-Test," *AT&T Technical J.*, Vol. 70, No. 1, Jan./Feb. 1991, pp. 87-100.
9. M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, 1991.
10. S.W. Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, Calif., 1982.
11. P.H. Bardell, W.H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, New York, 1987.
12. S.C. Seth, V.D. Agrawal, and H. Farhat, "A Statistical Theory of Digital Circuit Testability," *IEEE Trans. Computers*, Vol. C-39, No. 4, Apr. 1990, pp. 582-586.
13. E.B. Eichelberger et al., *Structured Logic Testing*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
14. P.D. Hortensius, R.D. McLeod, and B.W. Podaima, "Cellular Automata Circuits for Built-In Self-Test," *IBM J. Research and Development*, Vol. 34, No. 2/3, Mar./May 1990, pp. 389-405.
15. C. Dufaza and G. Cambon, "LFSR-Based Deterministic and Pseudo-random Test Pattern Generator Structures," *Proc. European Test Conf.*, IEEE CS Press, 1991, pp. 27-34.
16. M. Khare and A. Albicki, "Cellular Automata Used for Test Pattern Generation," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, 1987, pp. 56-59.
17. J. van Sas, F. Catthoor, and H. De Man, "Cellular Automata-Based Self-Test for Programmable Data Paths," *Proc. Int'l Test Conf.*, IEEE CS Press, 1990, pp. 769-778.
18. N.R. Saxena and J.P. Robinson, "Syndrome and Transition Count Are Uncorrelated," *IEEE Trans. Information Theory*,

Selected sources of BIST information

Abramovici, M., M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press (W. H. Freeman and Co.), New York, 1990. *The latest comprehensive text on testing.*

Agrawal, V.D. and S.C. Seth, *Test Generation for VLSI Chips*, IEEE Computer Society Press, Los Alamitos, Calif., 1988. *Easy-to-read tutorial on testing with comprehensive bibliography.*

Bardell, P., W. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo-random Techniques*, John Wiley & Sons, New York, 1987. *Provides mathematical treatment of BIST theory and practice.*

Eichelberger, E.B., E. Lindbloom, J.A. Waicukauski, and T.W. Williams, *Structured Logic Testing*, Prentice-Hall, Englewood Cliffs, N.J., 1991. *Provides detailed testing and self-test techniques used in level-sensitive scan design environment.*

IEEE Trans. Industrial Electronics, Special Issue on Testing, Vol. 36, No. 2, May 1989. *Contains five tutorial articles on testing, DFT, and BIST.*

J. Electronic Testing: Theory and Applications, Special Issue on Boundary Scan, Vol. 2, No. 1, 1991. *Includes a tutorial on boundary scan and articles on recent research results.*

Yarmolik, V.N. and S.N. Demidenko, *Generation and Application of Pseudorandom Sequences for Random Testing*, John Wiley & Sons, Chichester, UK, 1988. *Provides mathematical theory of feedback shift registers as applied in testing environment.*

Vol. 34, Jan. 1988, pp. 64-69.

19. S.M. Reddy, K.K. Saluja, and M.G. Karpovsky, "A Data Compression Technique for Built-In Self-Test," *IEEE Trans. Computers*, Vol. C-37, No. 9, Sept. 1988, pp. 1151-1156; correction, Vol. C-38, No. 2, Feb. 1989, p. 320.

20. K. Iwasaki and F. Arakawa, "An Analysis of the Aliasing Probability of Multiple-Input Signature Registers in the Case of a 2^m-ary Symmetric Channel," *IEEE Trans. Computer-Aided Design*, Vol. CAD-9, No. 4, Apr. 1990, pp. 427-438.

21. D.K. Pradhan, S.K. Gupta, and M.G. Karpovsky, "Aliasing Probability for Multiple-Input Signature Analyzer," *IEEE Trans. Computers*, Vol. C-39, No. 4, Apr. 1990, pp. 586-591.

22. S.Z. Hassan and E.J. McCluskey, "Increased Fault Coverage Through Multiple Signatures," *Proc. Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, 1984, pp. 354-359.

23. Y. Zorian and V.K. Agarwal, "Optimizing Error Masking in BIST by Output Data Modification," *J. Electronic Testing: Theory and Applications*, Vol. 1, Feb. 1990, pp. 59-72.

24. K. Akiyama and K.K. Saluja, "A Method of Reducing Aliasing in a Built-In Self-Test Environment," *IEEE Trans. Computer-Aided Design*, Vol. CAD-10, No. 4, Apr. 1991, pp. 548-553.

Vishwani D. Agrawal's biographical sketch and photo appear on page 28.

Charles R. Kime is a professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison, where he has developed and taught a broad range of computer engineering courses. His research interests include testing, design for testability, BIST, and fault-tolerant computing. He has served as general chairman of the 1979 International Symposium on Fault-Tolerant Computing, as associate editor of *IEEE Transactions on Computers* and *IEEE Transactions on Computer-Aided Design*, and on the program committees of IEEE conferences. Kime is a fellow of the IEEE and a member of the IEEE Computer Society.
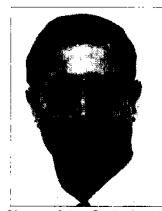
Kewal K. Saluja is a professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison, where he teaches logic design, computer architecture, microprocessor-based systems, and VLSI design and testing. Previously, he worked at the University of Newcastle, Australia. He has also held visiting and consulting positions at the University of Southern California, the University of Iowa, and Hiroshima University. His research interests include design for testability, fault-tolerant computing, VLSI design, and computer architecture. He is an associate editor of the *Journal of Electronic Testing: Theory and Applications*. Saluja received the BE from the University of Roorkee, India, and the MS and the PhD in electrical and computer engineering from the University of Iowa. He is a member of the IEEE Computer Society.

Send correspondence about this article to Vishwani D. Agrawal, AT&T Bell Laboratories, 600 Mountain Ave., Room 2C-476, Murray Hill, NJ 07974; e-mail: va@research. att.com.