E. B. Eichelberger
E. Lindbloom

Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test

Embedded linear feedback shift registers can be used for logic component self-test. The issue of test coverage is addressed by circuit modification, where necessary, of random-pattern-resistant fault nodes. Also given is a procedure that supports net-level diagnosis for structured logic in the presence of random test-pattern generation and signature analysis.

Introduction

Logic self-test, the facility to test logic with built-in pattern sources and response evaluators, is currently being widely investigated in the literature. The *self-test* approach is offered as an alternative to conventional component final test, which uses software-precalculated stimuli and response data stored in external test equipment. A number of recent self-test proposals suggest partitioning logic designs into sub-networks, each with a small number of inputs, such that *exhaustive* pattern sets can be applied to ensure complete coverage [1–3]. As attractive as exhaustive testing is, these proposals encounter problems. Actual logic designs contain networks that only allow partitioning to the required limits with unacceptable circuit delays or pin costs. Further, the partitions, where achieved, are sensitive to engineering design changes. A different approach to self-test, also discussed in current literature [4–8], argues that *random* patterns can be applied to an *entire* logic design in sufficient number and speed to ensure adequate coverage. These latter proposals have in common scan-path structured design and the use of *linear feedback shift registers* (LFSRs) to generate pseudo-random patterns and to collect response signatures. The body of this paper is concerned with the second approach, i.e., random-pattern self-test.

Present and projected testing costs supply the motivation for the investigation of self-test. If logic self-test can be put into practice, there are a number of potential benefits. Those most frequently cited are 1) reduction of test-pattern generation and data management costs, 2) use of less costly equipment for chip and field-replaceable-unit test, and 3) migration of component tests to field service use.

But if logic self-test is going to involve hardware generation of pseudo-random patterns (RPs), then there are questions that need additional study. Is the *fault-coverage* achievable with RP self-test equal to or superior to that of conventional test methods? Will the required percentage of faults be exposed by the number of RPs that can be generated in some acceptable length of time? Further, if long runs of RPs are used for fault detection, is *diagnosis* of failures possible with self-test? Are there diagnostics techniques that can be used in the self-test environment that are comparable in efficiency and result to those now used with conventional final test?

One part of RP self-test, compressing test responses into signatures, is already actual testing practice [9]. If there are ways of ensuring the efficacy of the other major part, *random-pattern generation* (RPG), then, given the disciplined state of design that the acceptance of constraints for testability has brought about, it may be possible to combine RPG and signature generation to realize the benefits of self-test.

A proposal that addresses the questions of coverage and diagnostics is described in the following sections of this

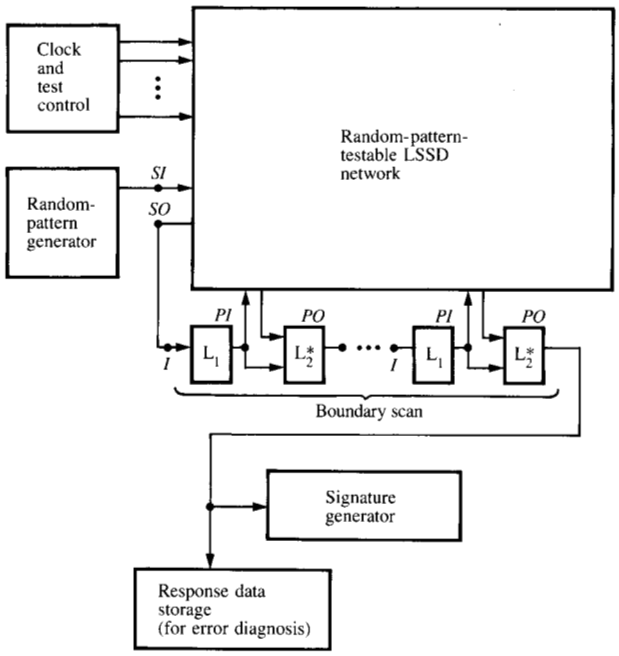

Figure 1 General structure for random-pattern self-test.

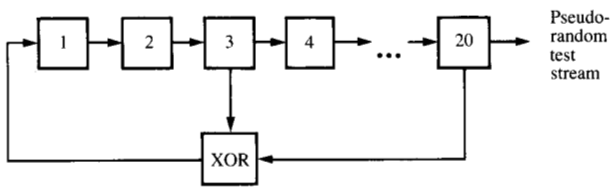

Figure 2 Random-pattern generator: A 20-bit, maximal-length-sequence linear feedback shift register with taps at bits 3 and 20, and with a test-bit-stream output port.

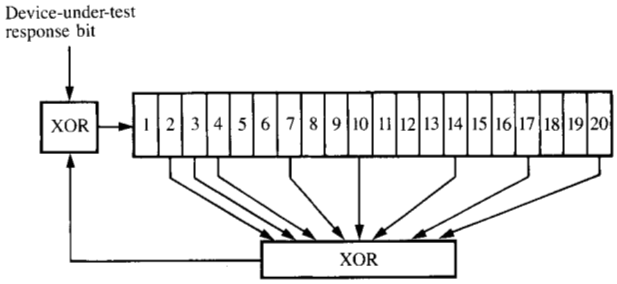

Figure 3 Signature generator: A 20-bit maximal-length-sequence LFSR with eight taps and with a device-test-response-bit input port.

paper. A general structure for self-test is first discussed, and then circuit *modifications* that render logic designs random-pattern testable are described. Next, modifications for some particular random-pattern-resistant networks—cascaded gate-array ANDs and large programmable logic arrays (PLAs)—are discussed in some detail. Finally, a method for practicing net-level diagnosis with *level-sensitive scan design* (LSSD) [10] random-pattern self-test is presented.

Random test-pattern coverage

Assume a general self-test structure as shown in Fig. 1. To an LSSD network, with all primary inputs (PIs) and primary outputs (POs) latched in a scan string, is added an LFSR, for instance that in Fig. 2, as a source for random patterns [11]. Also added is another LFSR, for instance that in Fig. 3, to compress the serial scan-out data stream into a signature which can be compared to a precalculated good result. This structure can be viewed as a particular means for performing LSSD testing: Test patterns are scanned into shift-register latch (SRL) strings, system clocks are pulsed, and test results are scanned out. But how adequate is the test coverage when patterns are generated with an LFSR as opposed to being algorithmically generated with software and stored at a tester? Given that an n-bit LFSR can be constructed to generate

$$2^n - 1/\textit{scan length} + \textit{PIs}$$

pseudo-random patterns on LSSD logic, how many RPs are required for any particular network? It has been shown that RPs are often effective in detecting faults in combinational networks [12, 13]; this result has also been shown to extend to LSSD networks [14]. "Effective" here means obtaining the desired fault coverage with a set of patterns which, while much larger than a set of deterministic patterns with comparable coverage, is still much smaller than a set of exhaustive patterns. The fact that the required set of RPs is large is not an obstacle to a self-test technique where patterns are generated by hardware.

But is it not possible that there are logic networks which are the product of intelligent design choices and comply with LSSD design-for-testability rules that are *not* fully tested with any practical number of RPs?

If the number of RPs needed for complete testing is related to the size of the highest fan-in networks within the device to be tested, then it is likely that VLSI/LSSD packages will contain some hard-to-expose faults. To say that a fault is not easily testable with RPs means either that there exists a node in the network where the probability of randomly arriving at "1" (or "0") is very low, or that having achieved the required value, the probability of also setting up a sensitized path to an output is very low. We show that in either case it is possible to *modify* the original logic, without changing the system function so that the fault is more easily detected with RPs.

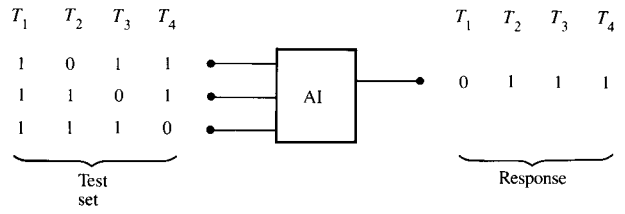

Figure 4 Required tests for an AND-INVERT block.

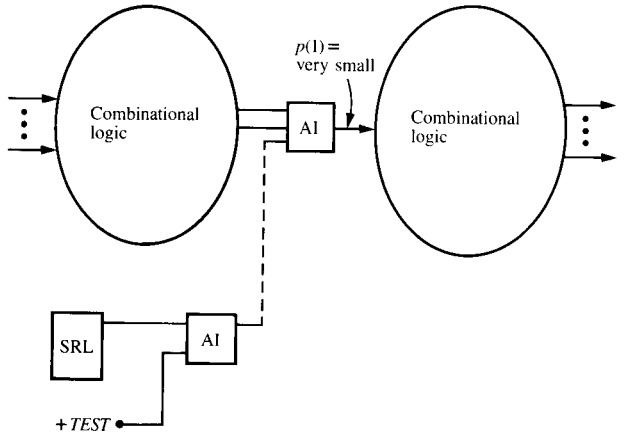

Figure 5 Addition of random test variable to modify RP testability.

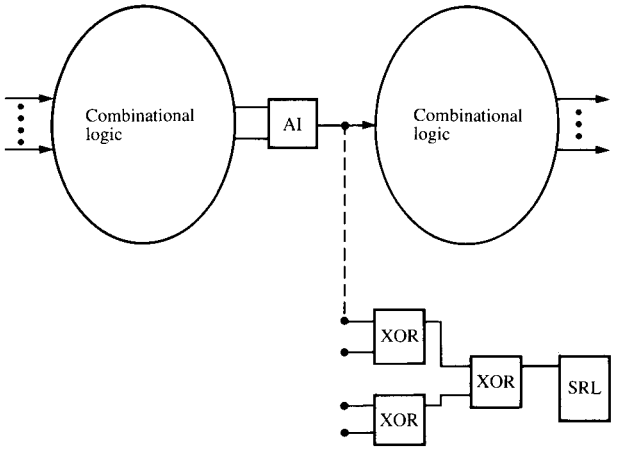

Figure 6 Addition of observation points to modify RP testability.

Table 1 Fault coverage with random patterns and with a deterministic test-pattern generator for two sample chips.

	Inputs	*Logic gates*	*Fault coverage (%) with random patterns*			*Fault coverage (%) with a deterministic generator*
			100	1,000	10,000	
Chip 1	63	926	86.1	94.1	96.3	96.6
Chip 2	54	1103	75.2	92.3	95.9	97.1

Modifying circuits for RP testability

• *A general technique*

The following is a technique for modifying a combinational network (an LSSD sub-network) to make it RP-testable. Consider the three-input NAND circuit shown in Fig. 4. All stuck faults associated with this logic circuit will be tested if the four tests shown are applied to the circuit and its output observed either directly or through other logic circuits. Consequently, if this circuit is part of an LSSD network and not fully tested by a set of RPs, then one of two possibilities exist: Either the RPs did not result in all four primitive tests being applied, or, when applied, the output of the circuit was not observable. If the problem is due to one or more of the tests not being generated by the RPs, this is usually caused by one or more of the circuit inputs having a *very low probability* of being at "1" (or "0"). This can be corrected by modifying the logic circuit feeding the untested circuit, as shown in Fig. 5. When the "+ *TEST*" signal is held positive during test, the probability of the circuit output being "1" is changed from "very small" to approximately 0.5. (Where the probability of the circuit output being "0" is very low, the logic modification is the same as indicated in Fig. 5 except that in this case the "new" variable is dotted with the output of the circuit.) If, however, the RPs result in the tests T_1, T_2, T_3, T_4 in Fig. 4 being applied but the output of the circuit is not *observable*, this can be addressed by an additional fan-out from the circuit to an SRL. If more than one new internal observation point is required, they can share one SRL through an EXCLUSIVE-OR tree, as shown in Fig. 6.

These two design modification techniques are a form of test-point addition, but notice that in an LSSD environment the overhead need not be large. No additional pins are needed. In general, an existing SRL can be used to generate the random test variable in Fig. 5, and the same variable can fan-in to other circuits if needed. The observation points in Fig. 6 require one additional SRL and $n - 1$ EXCLUSIVE-ORs for n added points. Further perspective on overhead can be gained by recalling that previous experience with RPs in an LSSD environment indicates that many networks require no modification to be testable [14]. Recent results with current product samples are encouraging (see Table 1). The two logic modification techniques are proposed as insurance

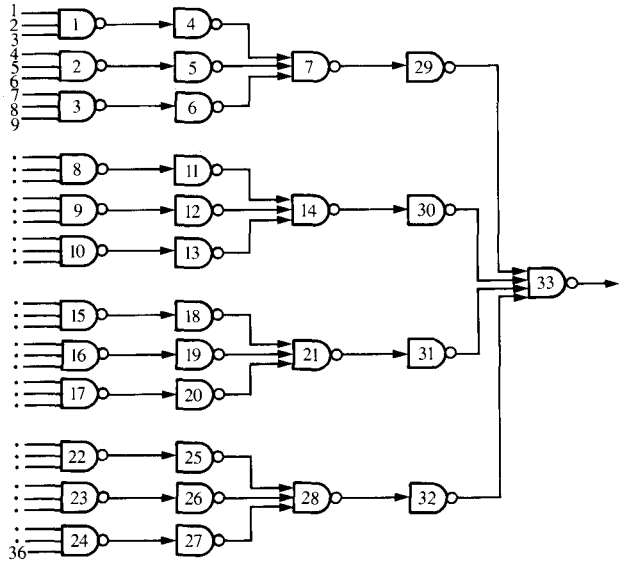

Figure 7 Circuit A, a cascaded AND network.

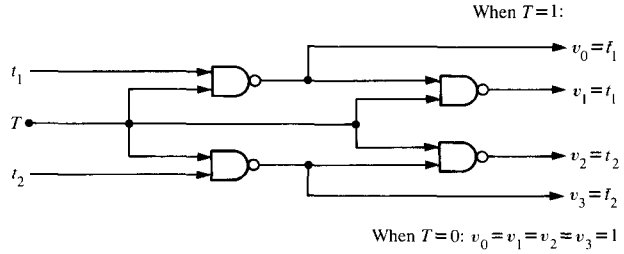

Figure 8 Circuit used for 1-of-4 select.

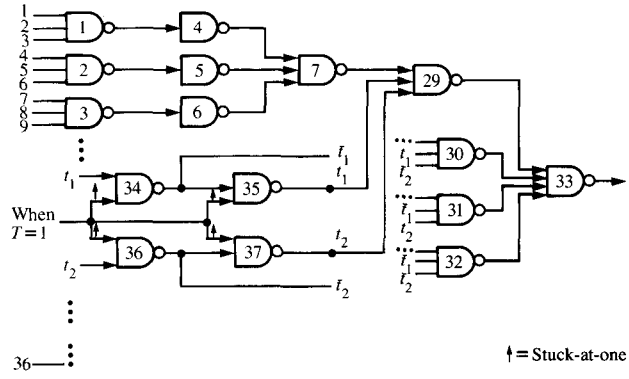

Figure 9 Circuit B, AND network modified by addition of 1-of-4 select.

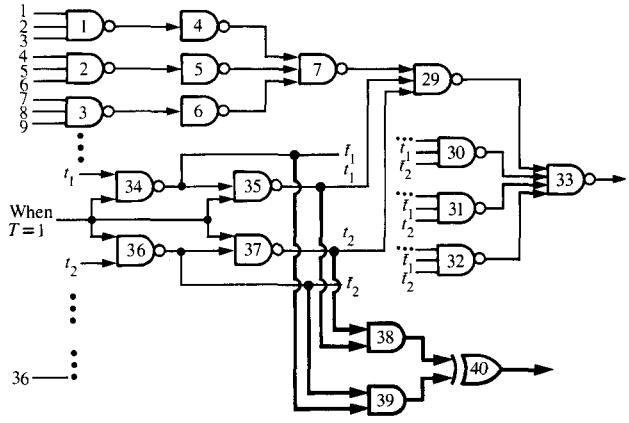

Figure 10 Circuit C, AND network with 1-of-4 select and additional circuits for observation.

against exposure with using RPs on some particular design. Notice that the techniques both effectively shorten the length of a path in the logic, from an input edge where all test values are approximately equally likely, to an observation point where the test is captured; and for masterslice (gate-array) logic, where the allowable fan-in to a circuit is usually low, that is often all that is needed to render the logic testable with a practical number of RPs.

• *Modifying a large, cascaded AND network*

If, as seems likely, RP-resistant faults are likely to cluster on certain characteristic sub-networks, then a more global modification is often effective. Consider the large, cascaded AND, designated as Circuit A, in Fig. 7. Here, even if the circuit is directly accessible to test, each of the prime faults has a detection probability of only $1/2^{36}$. The only fault detected in Circuit A after simulation with 20 000 RPs is the output of block 33 stuck-at-zero. To enhance the RP testability of Circuit A we can add the *1-of-4 select* circuit shown in Fig. 8. The resulting composite circuit, labeled Circuit B, is shown in Fig. 9. T is a test mode pin, t_1 and t_2 are random variables, which could be existing, independent SRL bits. The 1-of-4 select circuit is such that when test signal $T = 0$, all four derived signals v_0, v_1, v_2, and $v_3 = 1$, and the system function is unaltered. When $T = 1$, one and only one of the sets of derived signals, taken pairwise, $v_1\, v_3$, $v_1\, v_2$, $v_0\, v_3$, and $v_0\, v_2$, are set equal to 1, and the prime faults in Circuit B have a random-pattern detection probability of $1/2^9$. The effect of the added circuit is that in test mode any random pattern sets exactly 3/4 of the original circuit to a noncontrolling state, substantially increasing the probability of achieving a test on the remaining quadrant.

Fault simulation of Circuit B revealed that after 20 000 RPs all faults were detected, except the four indicated in Fig. 9. The added circuit introduces four new faults nearly as difficult for RP testing as the original circuit. Further, these new faults, if present, would affect system function. To cover these faults, we can add the observation circuit shown in bold lines in Fig. 10, arriving at a new composite, designated as

Circuit C. The XOR output can be observed in an SRL or added to an observation XOR tree, as in the preceding discussion of general modifications. With this addition, Circuit C is 100% testable with 20 000 RPs. See Table 2 for the fault-simulation statistics.

• *A random-pattern-testable design for PLAs*

There is a popular design choice for VLSI custom logic that is characterized by very high fan-in circuits—*programmable logic arrays* (PLAs). The problem that PLAs present for random-pattern generation has been observed [13] and consists simply of the fact that the AND array may contain single circuits with high fan-in, say 24. In this case, the probability of randomly establishing any one of the primitive tests for the circuit is $1/2^{24}$, or approximately one out of 16 million, given probabilities of 0.5 for "1" and "0" on all of the circuit inputs. (Note that high fan-in to the OR array does not present the same problem because here the random probability that any circuit input is in the noncontrolling state is very high. For a number of PLAs selected from a microprocessor and characterized in Table 3, PLA 1 contained a 37-way OR-array circuit. However, with no AND-array circuit in PLA 1 larger than a 12-way, almost all testable faults were covered with 10^4 patterns.) Maximum fan-in to any AND-array circuit can be used as an index to the random testability of a PLA. If this becomes large, then there are two choices: 1) break the single PLA into smaller ones with "narrower" AND arrays, or 2) add circuitry to the large PLA to make it random-pattern-testable.

Figure 11 illustrates a circuit modification that makes a PLA testable with RPs. This additional circuitry consists of two sections; the first is called *segment select*. Here, four signals u_0, u_1, u_2, and u_3 are generated from two random test variables t_1 and t_0 under control of the test signal T such that they have the following properties:

1. When $T = 0$, all four signals u_0, u_1, u_2, and u_3 are in their "off" state and the PLA bit-partitioning logic works in the normal way. (The normal PLA inputs are shown in Fig. 11.)
2. When $T = 1$, exactly one of the four signals u_0, u_1, u_2, and u_3 is "off" while the other three are "on," forcing *all* AND-array inputs in their quadrants to the "1" value. The AND-array inputs to the fourth quadrant are controlled by the normal PLA input.

The test mode, when $T = 1$, changes the probability of generating a primitive test for an evenly partitioned 24-input AND gate from $1/2^{24}$ to $1/2^8$, if random patterns are applied to the PLA inputs.

Although the segment select improves the chances of getting the required primitive test values, it hinders observability. To ensure that tests propagate through the OR array,

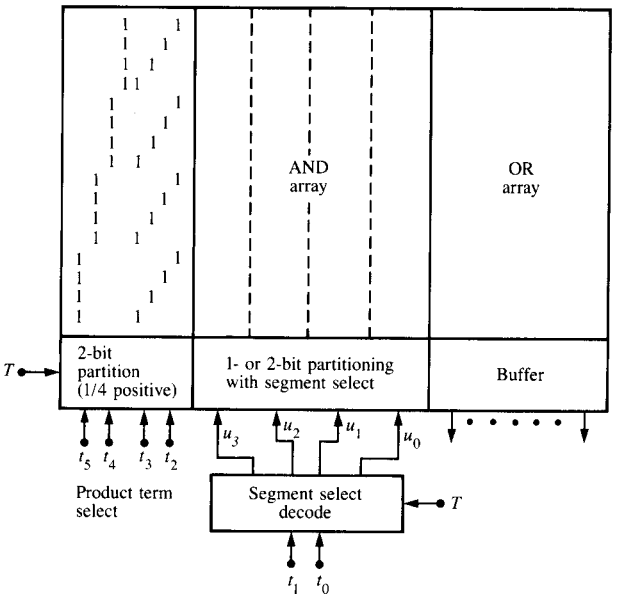

Figure 11 Random-pattern-testable PLA.

Table 2 Fault-simulation statistics for Circuits A, B, and C.

LFSR pattern count	*Fault coverage (%)*		
	Circuit A	*Circuit B*	*Circuit C*
100	1.0	7.9	23.3
1,000	1.0	46.0	52.8
10,000	1.0	95.5	97.5
20,000	1.0	98.2	100.0

Table 3 Fault coverage with random patterns for four sample PLAs.

	Inputs	*Product terms*	*Fault coverage (%) with random patterns*			*Untested faults*
			100	*1,000*	*10,000*	
PLA 1	14	54	41.5	79.2	99.3	4
PLA 2	15	30	39.8	93.6	100.0	0
PLA 3	19	32	92.4	99.4	99.6	2
PLA 4	21	27	94.2	100.0	100.0	0

the second section of circuitry shown in Fig. 11, *product term select*, is added. The operation of this logic is as follows:

1. When $T = 0$, all product-term-select inputs to the AND array are forced to "1," allowing the PLA to function normally.
2. When $T = 1$, the random variable inputs t_2, t_3, t_4, and t_5 are pair-wire-decoded such that exactly two out of eight inputs to the AND array are "1." This ensures that one

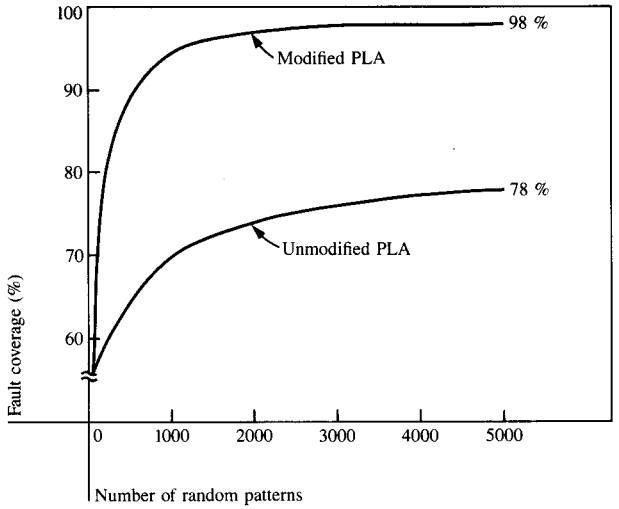

Figure 12 Random-pattern test coverage curves with unmodified and modified PLA.

product term has two "1" inputs and all other product terms have at least one "0" input.

Thus, at most, one product term can be selected at a time and the results of any associated applied test can be observed at the output of the PLA. The number of additional random inputs required can be reduced by partitioning the product terms of a PLA into n groups, such that each member of a group has at least one path to an observation point not shared by another member, and selecting all members of one group together.

An experiment was performed in which a large PLA, one with 38 primary inputs, 117 product terms, and a maximum AND-array fan-in of 20, was modified by the inclusion of segment-select and product-term-select circuitry. The unmodified PLA had a 94.3% fault coverage with a state-of-the-art deterministic generator (the remaining faults were logically redundant) and an 82.3% coverage with 10^4 RPs. See Fig. 12 for coverage curves. After modification, 98.0% was achieved with 4300 RPs. The increase in testability over the deterministic generator was due primarily to the fact that the previously redundant crosspoints were made testable with the use of the product-term-select technique.

A number of recent papers have proposed designs for fully testable PLAs [15–17]. These designs physically or logically partition the PLA AND array and OR array into separate circuits divided by added SRL strings. The separate circuits are fully testable by function-independent, universal, deterministic pattern sets. These designs, despite their generous addition of SRLs, do not aid—nor were they intended to—in the random-pattern testability of high fan-in circuits characteristic of PLAs.

Determining RP coverage

Apart from the special case of PLAs in custom logic just discussed, how do we know if a masterslice design is RP-testable? For a given design, are *any* of the internal test points discussed earlier needed? If so, *where* should they be added? In support of self-test, a tool is needed to determine RP testability and potential "trouble points."

Fault simulation is a possibility: Large numbers of LFSR-generated patterns could be fault-simulated against an LSSD network and the residual untested circuits studied. But this is a not entirely attractive alternative since it eliminates one of the intended benefits of self-test—reduction of computer costs involved in test generation.

A software fault simulator tailored to LSSD networks and long runs of patterns against a small residue of untested faults may be feasible. Recent developments with special-purpose hardware offer the prospect of a fault-simulation machine [18]. Another alternative is an *analytic* tool. It may be possible to write an efficient program to calculate the joint probability of controlling and observing a test bit on each net in a combinational network [19]. Such a tool must contain a solution to the perennially troublesome "reconvergent-fan-out" problem. Given these probabilities, it will be clear whether logic modification is necessary and where added internal random variables or observation points are needed. SCOAP [20] numbers, which are integer values indicating the minimum number of primary inputs needed to control and observe a net state, do not appear helpful unless a solution to the reconvergent-fan-out problem is found [21–22].

Diagnostics with LSSD random-pattern self-test

Before we implement self-test as a manufacturing procedure, it will be advantageous to have shown that it can support net-level diagnosis. One way of accomplishing this might be the following. Referring to Fig. 1, the scan-out signal line which drives the signature generator is also brought to a pin, such that it can (but only when needed) drive a small, external data-storage facility. This pin, together with *intermediate* signatures recorded when calculating the final signature, supports the following diagnostic procedure:

1. Run self-test to completion in normal mode.
2. If the signature at completion of full test does not agree with the precalculated one, re-initiate the pattern generator and run for 100 cycles, collecting all scan data in the storage facility.

3. Compare the signature with the expected one after 100 cycles; if it is good, run and record response data for the next 100 cycles—overlaying the previous data.
4. If an intermediate signature does *not* agree with the expected one, compare the data in the storage facility with the good machine responses and determine which bits failed. (Note that the response data can be generated on demand by software that ripples the pattern generator LFSR up to the pattern of interest and then performs good machine simulation on the segment containing the fail(s). Or experience may indicate that it is preferable to do a one-time generation of the first few thousand patterns' worth of good machine responses.)

With the identification of the individual failing bits, we have the information required for current diagnostic practice with LSSD chips and multi-chip modules [23, 24]. From the failing responses and the self-test cycle counter, we can exactly determine the primary input pattern exposing each failing response; now it is possible to do the post-test full-fault diagnosis described in [23, 24]. The strategy of this diagnostic practice is that since each LSSD pattern completely reinitializes the device under test, fault simulation of the entire pattern set is not required to help locate the cause of an observed failure. In place of the traditional precalculated fault dictionaries, LSSD diagnostics requires fault simulation of only those load, clock, and unload sequences that contain actual failures observed under test. Provided that the self-test pattern generator LFSR is physically distinct from the LFSR used for signature collection (as in Fig. 1), so that no input pattern is corrupted by an upstream failure bit, and a snapshot of any response data is available in diagnostic mode (data storage buffer, Fig. 1), failure location is as good as that with conventional LSSD testing. What the diagnostic simulator requires is identification of 1) failing responses and 2) the input patterns that exposed these fails. We hope to have shown that both can be efficiently obtained in the presence of RPs and signature registers.

Conclusion

We have described a self-test proposal that faces the questions of test coverage and diagnosis. Significant work remains to be done in the area of tools to determine if and where circuit modification is required for RP testability. The same tool used to calculate the difficulty of exposing each fault in a net could be rerun, after any necessary modification, and used to estimate the number of patterns required and the fault coverage.

Another significant challenge to LSSD RP self-test methodology is the additional degree of design discipline required to ensure that a network is RP-testable. A design-for-testability rules checker must ensure not only that the logic is scannable and level-sensitive but also that no indeterminate state can exist in test mode, such as an uninitialized RAM cell.

A final reflection: The self-test approach as described here rests on random-pattern testing; but if RP testing is effective, why is it not already widely practiced? The answer is that initially the test community was faced with large, unconstrained, sequential packages and RPs were not effective with them. With design constraints, RP testing was still not economical if it meant generation of enormous software pattern sets and transmitting them to be applied by external test equipment. Self-test with LFSRs, however, provides a way of generating and compressing these enormous data sets via simple hardware. But persistent concerns remained: There are networks in some well-designed products that could force us, with RP self-test, to completely unreasonable test lengths. And, having compacted all the test responses into a signature, we could be faced with an intractable diagnostic problem. The design modifications and diagnostic procedure we have described, which are simple in concept and implementation, should alleviate these concerns.

Acknowledgments

Our thanks go to Jim Fiore and Elizabeth Bouldin for conducting the experiment of altering a large PLA to make it RP-testable. We are indebted to Gordon Smith and Phil Noto for their analysis of characteristic RP-resistant faults in high-performance masterslice designs. Finally, we are grateful to the reviewers who pointed out difficulties in the original manuscript and acquainted us with additional literature in this rapidly developing field.

References

1. R. M. Sedmack, "Design for Self-Verification: An Approach for Dealing with Testability Problems in VLSI-Based Designs," *Digest of Papers, 1979 IEEE Test Conference,* Cherry Hill, NJ, pp. 112–120.
2. E. J. McCluskey and S. Bozorgui-Nesbat, "Design for Autonomous Test," *Digest of Papers, 1980 IEEE Test Conference*, Cherry Hill, NJ, pp. 15–21.
3. J. Savir, "Syndrome-Testable Design of Combinational Circuits," *IEEE Trans. Computers* **C-29,** 442–451 (1980).
4. B. Koenemann, J. Mucha, and G. Zwehoff, "Built-In Logic Block Observation Techniques," *Digest of Papers, 1979 IEEE Test Conference*, Cherry Hill, NJ, pp. 37–41.
5. H. Eiki, K. Inagaki, and S. Yajima, "Antonomous Testing and its Application to Testable Design of Logic Circuits," *Digest FTCS-10, 10th Annual International Conference on Fault-Tolerant Computing*, Kyoto, Japan, 1980, pp. 173–179.
6. E. R. Holland and J. L. Robertson, "GUEST—A Signature-Analysis-Based Test System for ECL Logic," *Hewlett-Packard J.* **33,** 26–29 (1982).
7. P. Bardell and W. H. McAnney, "Self-Testing of Multichip Logic Modules," *Digest of Papers, 1982 International Test Conference*, Philadelphia, PA, pp. 200–204.
8. D. Komonytsky, "LSI Self-Test Using Level-Sensitive Scan Design and Signature Analysis," *Digest of Papers, 1982 International Test Conference*, Philadelphia, PA, pp. 414–424.

9. R. A. Frohwerk, "Signature Analysis: A New Digital Field Service Method," *Hewlett-Packard J.* **28,** 2–8 (1977).
10. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Annual Design Automation Conference*, New Orleans, LA, June 1977, pp. 462–468.
11. S. W. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, 1967.
12. P. Agrawal and V. D. Agrawal, "Probabilistic Analysis of Random Test Generation Method for Irredundant Combinatorial Logic Networks," *IEEE Trans. Computers* **C-24,** 691–695 (1975).
13. H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Trans. Computers* **C-24,** 695–700 (1975).
14. T. W. Williams and E. B. Eichelberger, "Random Patterns Within a Structured Sequential Logic Design," *Digest of Papers, 1977 IEEE Semiconductor Test Symposium*, Cherry Hill, NJ, pp. 19–27.
15. S. J. Hong and D. L. Ostapko, "FITPLA: A Programmable Logic Array for Function Independent Testing," *Digest FCTS-10, 10th Annual International Conference on Fault-Tolerant Computing*, Kyoto, Japan, 1980, pp. 131–136.
16. H. Fujiwara, K. Kinoshita, and H. Ozaki, "Universal Test Sets for Programmable Logic Arrays," *Digest FCTS-10, 10th Annual International Conference on Fault-Tolerant Computing*, Kyoto, Japan, 1980, pp. 137–142.
17. W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," *IEEE Trans. Computers* **C-30,** 829–833 (1981).
18. G. F. Pfister, "The Yorktown Simulation Engine: Introduction," *Nineteenth Design Automation Conference Proceedings*, Las Vegas, NV, June 1982, pp. 51–54.
19. J. Savir, G. Ditlow, and P. H. Bardell, "Random Pattern Testability," *Research Report RC-9643*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.
20. L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. Circuits Systems* **CAS-26,** 685–693 (1979).
21. V. D. Agrawal and M. R. Mercer, "Testability Measures—What Do They Tell Us?" *Digest of Papers, 1982 International Test Conference*, Philadelphia, PA, pp. 362–370.
22. J. Savir, "Good Controllability and Observability Do Not Guarantee Good Testability," *Research Report RC-9432*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.
23. Y. Arzoumanian and J. Waicukauski, "Fault Diagnosis in an LSSD Environment," *Digest of Papers, 1981 International Test Conference*, Philadelphia, PA, pp. 362–370.
24. J. J. Curtin and J. Waicukauski, "Multi-Chip Module Test and Diagnostic Methodology," *IBM J. Res Develop.* **27,** 27–34 (1983).

Received August 30, 1982; revised December 20, 1982

Edward Eichelberger *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Dr. Eichelberger is an IBM Fellow and manager of advanced VLSI technology and testing at Kingston. He joined IBM in Endicott, New York, in 1956 and has worked in areas of VLSI chip design, circuit design, design automation, and design for testability. In 1973 he received an IBM Outstanding Contribution Award for the level-sensitive scan design (LSSD) technique. Since 1977, he has managed various custom design VLSI projects in both FET and bipolar technologies. He has reached the fifth IBM invention plateau and holds 15 U.S. patents. Dr. Eichelberger received his B.S. in electrical engineering in 1956 from Lehigh University, Bethlehem, Pennsylvania, and his Ph.D. in electrical engineering in 1973 from Princeton University, New Jersey.

Eric Lindbloom *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Mr. Lindbloom is a senior programmer in the Advanced VLSI Technology and Testing Department in Kingston. He joined IBM at Poughkeepsie, New York, in 1963 to work on automated logic design. Since 1966 he has pursued technical interests in test-pattern generation, simulation, and diagnosis for logic circuit final test. He received a B.A. in philosophy from the University of Michigan, Ann Arbor, in 1957. Mr. Lindbloom is a member of the Institute of Electrical and Electronics Engineers Computer Society. In 1972 he received an IBM Invention Achievement Award for contributions to semiconductor test.