A method for generating weighted random test patterns

by J. A. Waicukauski
E. Lindbloom
E. B. Eichelberger
O. P. Forlenza

A new method for generating weighted random patterns for testing LSSD logic chips and modules is described. Advantages in using weighted random versus either deterministic or random test patterns are discussed. An algorithm for calculating an initial set of input-weighting factors and a procedure for obtaining complete stuck-fault coverage are presented.

1. Introduction

An interest in the use of random patterns for testing logic devices has existed at least since 1965, when Seshu [1] described a test-generation method that included both random selection of input values and a deterministic generator. In 1971, Nagamine [2] outlined a method that used random patterns in its initial phase and a deterministic test-pattern algorithm, the D-Algorithm [3], as a finishing procedure. Agrawal and Agrawal [4] reported in 1972 on experiments in testing Illiac IV logic boards. The finding of this work was that the computer time required for test generation could be reduced by combining random patterns with the D-Algorithm.

The LSI era brought additional motivations for the development of random-pattern test methods. Benowitz et al. [5] argued that including linear feedback shift register (LFSR) random-pattern generators in logic circuits could reduce the costs of both tester equipment and diagnostic software. A 1975 paper by Schnurmann et al. [6] pointed to ways of using a random-pattern generator to achieve a test of better quality than that obtainable with existing deterministic test-generation programs.

The acceptance of design-for-testability rules, like LSSD [7] over a decade ago, began to transform logic test into an essentially combinational problem. One effect of this was that existing deterministic pattern systems received extended life; another was that associated research received new impetus. Furthermore, development of random-pattern strategies was deferred for a time. Soon, however, it was observed that even with testability design constraints such as LSSD, deterministic test data were likely to grow in a nonlinear manner in the VLSI era and potentially cause test application time problems [8–10].

In 1977 Williams and Eichelberger [11] noted that LSSD chips were often highly testable with 1000 random patterns. Two years later, Koenemann et al. [12] proposed a built-in random-pattern technique, called BILBO, which could be used with scan-path-designed logic. It was conjectured that with modest circuit additions, scan-path-constrained logic readily adapted to random-pattern test [13–15]. But the question remained: Can a random-pattern strategy provide the quality of test required in manufacturing? At IBM, for instance, technology groups were faced with the problem of supplying product for use with the Thermal Conduction Module (TCM) [16], which would contain approximately 100 chips and permitted limited rework cycles. Near 100% stuck-fault coverage on chips was viewed as a prerequisite for a successful TCM program.

Proposals have been made regarding methods for estimating the number of random patterns required to achieve a given coverage target for any particular device [17–20]. Experiments conducted with a large number of different logic designs for actual product, reported by Motika et al. [21], confirmed the belief that some LSI devices would not receive a full stuck-fault test with any tolerable number of random patterns. There are nonredundant logic networks with faults that are not easily exposed with random patterns [15]—such faults have become known as *random-pattern-resistant* faults. Something must be added to random-pattern testing to make it a complete strategy, one competitive in today's quality arena. It might be possible to supplement random patterns with deterministic ones to cover the random-pattern-resistant faults. However, this would entail stored test patterns and the erosion of the purported data-reduction advantage of random patterns. A solution to the problem would seem to imply that the device to be tested must be modified to remove the random-pattern obstacles, or the patterns themselves must be modified to provide full coverage with an economical number of patterns.

Modifying circuit designs to make them random-pattern-testable was discussed by Eichelberger and Lindbloom in 1983 [15]. More recently Briers and Totton [22] have developed a general procedure for doing such circuit modification.

The alternative—modifying the random test patterns themselves—was suggested by Carpenter et al. [23] in 1973. In 1975, two papers, one by Agrawal and Agrawal [24], the other by Schnurmann et al. [6], described modifications of random test patterns to improve their efficiency. The first method, based on logic depth and fan-in, was a means for obtaining a single weight parameter for all inputs of symmetric, nonreconvergent networks. The second method, aimed at unconstrained sequential logic, used good machine simulation to observe circuit switching resulting from primary input changes when stimulated with random patterns. In the final test-pattern set, individual inputs were given a weighted probability of switching in proportion to this observed internal activity count. In 1976 Parker described a means for adapting random patterns that involved a hill-climbing technique guided by fault simulation [25]; he also pointed to the utility of dividing patterns into subtests governed by different weights [26]. After a few years' hiatus in reported new developments, Timoc et al. [27] reported on an approach to testing a microprocessor that involved weighting the likelihood of the active state of selected functional pins; the choices were guided by repetitive use of a special-purpose hardware fault simulator. Lieberherr [28] discussed the analysis of test patterns generated with weighted input probabilities. In 1985 Wunderlich [29] proposed a general hill-climbing technique ("PROTEST") for obtaining a set of weighted random patterns; and in 1987 he published [30] a procedure for optimizing input probabilities. In 1986 Lisanke et al. [31] described a method which uses a testability-analysis tool, a gradient-calculation algorithm, and a fault simulator to achieve patterns with nonuniform input probabilities.

In the body of this paper a new method for generating weighted random patterns for LSSD logic is outlined [32]. Designated as the "weighted random-pattern" (WRP) method, it rests on a linear algorithm for calculating an initial set of input-weighting factors [33]. This initial weight set is based on the connections between logic gates in the device to be tested. A fast combinational fault simulator is then used to determine whether additional weight sets are required. If any are required, a conventional test-generation algorithm becomes the means for devising the finishing set or sets of weights.

Our production experience with bipolar logic chips shows that the WRP method is an efficient approach to test-pattern preparation. In addition, when it is integrated into a manufacturing, tester-based random-pattern test system, of the kind described in 1983 [21], it can improve the quality of logic final test.

The remaining sections of this paper describe the motivation for a weighted random-pattern system, the LSSD WRP application sequences, the initial weight-set algorithm, the procedure for generating additional weight sets, and some results obtained with the WRP method.

Throughout the paper three kinds of test patterns are distinguished: random, weighted random, and deterministic. *Random* as used here refers to patterns obtained directly from a maximum-length LFSR [34]. While bit sequences generated in this manner have some of the attributes of randomness—the next bit has a nearly equal and independent likelihood of being either a '1' or a '0'—they are, strictly speaking, "pseudorandom" in that they are predictable. *Weighted random* refers to patterns obtained from a pseudorandom source, together with a means for changing the 1/0 probability of any input bit according to a predetermined value. *Deterministic* refers to explicit, manually or automatically precalculated, usually dense, stored patterns.

2. Motivation

The testing of VLSI chips requires an extremely high-quality test that can be applied economically. A common way to accomplish this is with a deterministic test whose objective is to detect a very high percentage (>99%) of the nonredundant single stuck-faults (SSF). This will be considered a minimum test requirement, but with the increasing need to produce chips of higher quality, even this may not be sufficient. Nonmodeled defects (any defect which does not behave as an SSF) such as transition faults [35] and shorts are becoming the dominant contributor to quality problems. Generating deterministic tests specifically for these faults is almost certain to be prohibitively expensive. Even

the generation of a test for all SSFs has severe problems associated with high test-preparation costs and large test-data volumes.

The SSF model continues to play an important role in the testing of nonmodeled faults. Associated with many nonmodeled faults there is an SSF whose test is a necessary (though not sufficient) condition for detecting the nonmodeled fault. Nonmodeled faults which satisfy this condition are referred to as *dependent nonmodeled faults* (DNMF). For example, the detection of a dot-or short between two gates requires a test which will detect the stuck-at-'1' fault on one of the gates and place a '1' state on the other gate. Similarly, the detection of a transition fault requires a test which will detect the corresponding SSF and cause a transition to occur at the point of the fault. While the test for the SSF does not guarantee a test for the DNMF, it does satisfy a necessary condition. It follows, then, that the more frequently the SSF is detected, the more opportunities there will be to detect the DNMF, resulting in a higher probability of detection. Note that unless the corresponding SSF is detected at least once, there is no chance of detecting the DNMF. The DNMF coverage is always limited by the SSF test coverage. Since it is impractical to test explicitly for all the DNMFs, the desired objective of a test is to detect every SSF at least once and as many times as practical.

Random-pattern testing combined with collecting output responses in signature registers can solve some test problems. It eliminates test generation (although fault simulation may still be required) and requires only a small amount of data to define the entire test. Furthermore, the resulting increase in pattern count can improve the ability to detect nonmodeled defects but is limited by the SSF test coverage. However, the increased pattern count aggravates other problems, such as the cost of determining expected responses and evaluating the test coverage. But the fatal flaw of random-pattern testing is its inability to consistently achieve the required SSF test coverage with a reasonable number of test patterns. This deficiency is due to the lack of control of the random patterns to be directed toward untested faults. Pattern control allows a deterministic test to advance steadily toward the desired test-coverage goal, while a random test remains subject to the inherent random-pattern resistance of the design.

The use of weighted random-pattern testing is a means of controlling a random test to achieve maximum test coverage in a relatively small number of patterns. The benefits of a random test, such as minimal test generation and test data volumes, are retained. Given maximum SSF test coverage and many times more patterns (10–50×) relative to a deterministic test, the WRP test will almost certainly have an improved test coverage of nonmodeled faults.

To meet the test-coverage objectives for all structures in a production environment, the WRP method performs the following tasks:

- Determines the appropriate random-pattern application sequences that provide opportunities to test all faults. (These sequences define the events necessary to apply a single random pattern.)
- Calculates the desired weighting factors for each structure input that provide an adequate test probability for each fault.
- Determines the number of times the pattern application sequences must be repeated for a given set of weighting factors to achieve the desired SSF test coverage.
- Determines the expected signature register value for the resulting test. (The signature is a compression of output responses collected during the test. A comparison of the actual signature with the expected value determines whether the device has passed the WRP test.)
- Precisely measures the SSF test coverage.

At this point it may be useful to notice that in several ways WRP test is not a random procedure. The test patterns are perfectly *repeatable*; given the definition of the linear feedback shift registers (the source of the weighted random stimuli), the weighting factors, and the pattern application sequences, the WRP test is completely predetermined. Given this predictability, a WRP test is *measurable*; by using fault simulation, the expected signature and the exact test coverage are calculated. Finally, WRP testing is highly *structured*; device clock and scan pins are identified, and LSSD load and unload sequences—described in the next section—are applied at the required repetition.

3. WRP pattern application sequences

Objectives of pattern sequences

The first task of the WRP test generator is to create pattern sequences that identify how to apply random patterns in an LSSD environment. A sequence defines how input stimuli are applied to the device under test and how responses that result from these stimuli are collected. Sufficient sequences are created to allow all single stuck-faults to be detectable in at least one sequence. Each sequence is structured such that it can be represented by a single combinational pattern to allow for maximum simulation performance. For a combinational design, this consists of a single sequence that contains only two events: application of Weighted Random Values (WRV) to each Primary Input (PI) followed by compression of all Primary Output (PO) responses into a Multiple Input Signature Register (MISR). This sequence can then be repeated as often as desired with the appropriate weighting factors until the desired test coverage is achieved. LSSD structures, however, contain clocks and latches which require several, more complex sequences to provide the necessary opportunity to test all faults. It is assumed that all designs are double-latch LSSD, as indicated in **Figure 1**.

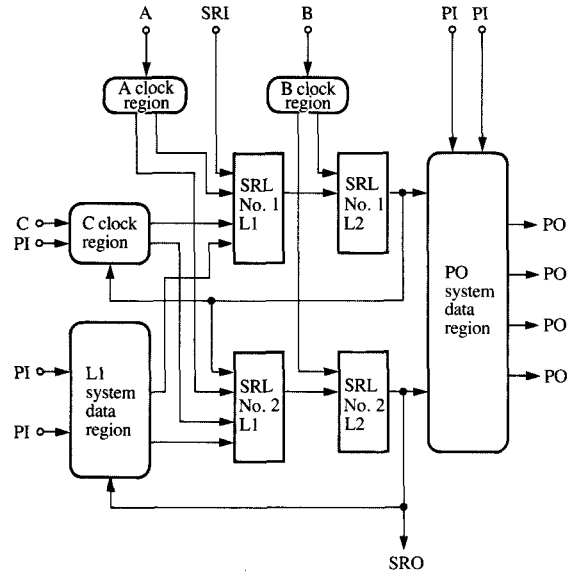

Figure 1

Fault regions in double-latch LSSD structure.

Table 1 Clock and observation points of fault regions.

Region	*Clock*	*Observe*
A clock	A, $\overline{A}$	L1
B clock	B, $\overline{B}$	L2
C clock	C, $\overline{C}$	L1
L1 system data	C	L1
PO system data	—	PO

• *Description of LSSD*

LSSD places all storage elements in scannable shift registers [7]. Each Shift Register Latch (SRL) of the Shift Register (SR) is a pair of latches, which are designated as L1 and L2. Individual SRLs may be treated as pseudo-PIs and pseudo-POs, since they are both controllable and observable points. Desired values can be placed in SRLs by serially shifting in those values from the SR Input (SRI) using the A and B test-mode clocks. System data can be captured into an L1 by application of a system clock (C), and can then be observed by serially shifting out values through the SR Output (SRO). During shifting out, either the L1 or L2 values may be selected to be observed. The observation of the L2 values is accomplished by repeating the shift sequence (measure SRO, pulse A scan clock, pulse B scan clock) for each SRL in the SR. To observe L1 values, the B scan clock is first pulsed, placing the L1 values into L2 latches, followed by the L2 observation routine. The clocks of an L1 latch (the A scan clock and the system clocks) are referred to collectively as "L1 clocks." Similarly, the clocks of an L2 latch (the B scan clock) are designated as "L2 clocks."

• *Stuck-fault regions*

Single stuck-faults are classified according to how the fault can become detectable. The classification is based on where the fault is observable (L1, L2, or PO) and what, if any, clocks must be activated for the fault to be observable. The five basic regions in a double-latch LSSD structure are illustrated in Figure 1. Faults in the scan path itself are assumed tested in a separate functional test of the SR and hence are not considered here. Faults in the clock region may cause clock inputs of latches to be stuck in either their "on" or "off" states. Detection of stuck "off" faults requires the application of the clock, while detection of stuck "on" faults requires the absence of the clock. It should be noted that a clock "on" fault can disturb the scan process and thus affect the accuracy of the simulation. However, it is unlikely that these faults will escape detection if the simulation predicts that the faults will be detected.

Table 1 identifies the clock necessary to expose each fault group and the observation point where the fault is detected. The bar above a clock indicates that the absence of that clock is necessary to detect certain faults in the region.

Pattern sequences are then generated to allow the detection of faults in each region. The appropriate sequences are repeated as many times as necessary to detect all faults. If a fault resides in more than one region, it need only be detected once. There are two basic types of pattern sequences depending on whether L1 or L2 latches are selected to be clocked and observed. Faults detected at POs can be observed with either sequence type.

• *Pattern sequence for faults that propagate to L1 latch or PO*

Most faults in a double-latch LSSD device belong to this class. These faults reside in the A clock region, C clock region, L1 system data region, and PO system data region, as shown in Figure 1. Detection of these faults requires placing WRVs on SRLs and nonclock PIs, followed by collecting PO responses and pulsing the appropriate L1 clock, and finally shifting out the L1 values. A separate sequence is defined for each L1 clock; the pattern application sequence for this class is as follows:

- Load shift register with WRVs.
- Apply WRVs to nonclock PIs.
- Collect PO responses in MISR (measure POs).
- Pulse L1 clock.
- Unload L1 values into MISR.

Since there are faults in L1 clock circuitry that require the L1 clock to be off to become detectable, one variation of the sequence is created by omitting the pulse of the L1 clock. For the example shown in Figure 1, three sequences of this type are created—an A clock sequence, a C clock sequence, and a sequence with no clocks.

• *Pattern sequence for faults that propagate to L2 latch or PO*

Faults in this class include faults in the B clock region in addition to the PO system data region faults already detected in the prior sequence. To detect the clock line faults, the value on the data line must be different from the latch value. For the B scan clock, the data line is the L1 latch which is paired with the L2. A normal SR load results in an L1 value being identical to its associated L2, which prevents the detection of these B scan clock faults. To allow L1 and L2 for the same SRL to be different, a skewed load of the SR is performed which adds an extra A scan clock pulse to the normal SR load. This disturbance of the L1 values does not affect the ability to detect other faults because the L1 latch only propagates to its paired L2 latch in a double-latch design. The class 2 sequence consists of the following events:

- Load shift register with WRVs (with extra A clock).
- Apply WRVs to nonclock PIs.
- Collect PO responses in MISR.
- Pulse L2 clock.
- Unload L2 values into MISR.

Once again, to detect faults in L2 clock lines which require the L2 to be in the off state, one variation of the sequence is created by omitting the pulse of the L2 clock. For the example shown in Figure 1, there are two sequences of this type—a B clock sequence and a sequence with no clocks. This results in a total of five sequences to test all the fault regions shown in Figure 1.

• *Clock grouping*

Although an LSSD chip might have as few as three clock inputs (C, A, and B), in general it might contain a number of different C clocks as well as multiple A and B clocks. This could result in large numbers of both types of pattern sequences. To minimize these sequences, an attempt is usually made to group the clocks together so that the clocks in a single group may be pulsed simultaneously. This is desirable because the extra clocks pulsed in a sequence allow additional faults to be detectable for a given number of patterns, which reduces the total test length.

Care must be taken in selecting the L1 or L2 clocks which can be safely grouped together. Incorrect grouping can result in race conditions or overlaying of desired values in latches. Clocks may be grouped together if they are the same type (either L1 or L2) and do not propagate directly to a common latch.

Table 2 Optimal weights for isolated gates.

No. of inputs (n)	*Probability of NCV*
1	0.50
2	0.60
3	0.69
4	0.75
5	0.80
.	.
.	.
.	.
n	.

4. Initial weight generation

After it is determined how the random patterns are to be applied, it is necessary to provide sufficient control (via weights) to the random patterns to ensure that all SSFs are detected.

• *Optimal weighted probabilities for isolated gates*

Even a single logic gate can require an enormous number of random patterns in order to test all of the stuck faults associated with the gate. For example, the average random-pattern test length for a 20-input AND gate is approximately three million patterns; this contrasts with a test length of only 21 patterns for a deterministic test. An examination of the patterns that detect the faults reveals that there is one fault (the output stuck-at-'0') that requires all 20 inputs at a '1' state, while the other 20 faults (the input stuck-at-'1' faults) need 19 inputs at a '1' state and the other input at a '0'. This strongly suggests that increasing the probability of a '1' state on the inputs would decrease the expected number of random test vectors required to detect every fault. If the probability of a '1' state being applied to any given input line is increased to 0.95 (approximately the distribution of '1' states in the set of deterministic patterns), then the average random test length is reduced to 190 patterns—a reduction of more than four orders of magnitude!

This shifting of the probabilities of a '0' and a '1' away from a "purely random" even distribution is what is meant by "weighting." By applying properly generated weighted random values to a gate, it is possible to dramatically reduce the number of vectors needed to fully test the faults associated with it. To accomplish this, it is desirable to increase the probability of the noncontrolling value (NCV) of the inputs to the logic gate ('1' for the AND-type blocks and '0' for the OR-type blocks). The optimal weights for minimizing the average test length for any isolated gate with n inputs, as derived from Monte Carlo experiments, are given in **Table 2**. When $n = 1$, the optimal probability is given by $n/(n + 1)$ or 1/2. As n becomes large, the optimal probability becomes progressively less than $n/(n + 1)$ and approaches, but never exactly equals, $(n - 1)/n$.

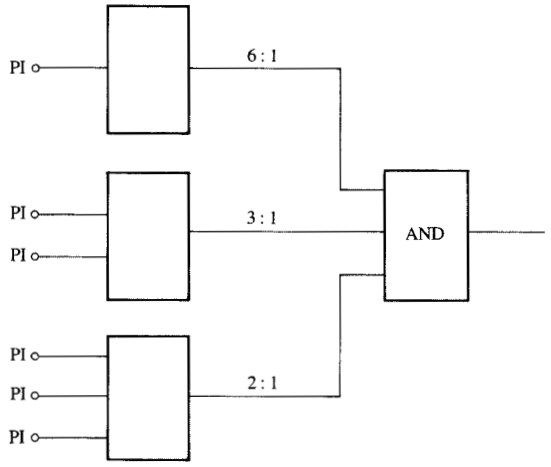

Figure 2

Calculated weights for gate imbedded in logic structure; '1' : '0' ratios are indicated.

Table 3 Formulas for calculating *W0* and *W1*.

Logic function of g	$W0_i$	$W1_i$
AND	$W0_g$	$Ri \cdot W1_g$
NAND	$W1_g$	$Ri \cdot W0_g$
OR	$Ri \cdot W0_g$	$W1_g$
NOR	$Ri \cdot W1_g$	$W0_g$

• *Calculation of weights for a logic structure*

When a gate is imbedded in a logic structure, the calculation of the optimal weighting factors for each input becomes more complex. The weighting factors must provide a test not only for all faults on a gate but also for the faults which are tested through it. The probability of applying the NCV to each gate input is still enhanced, but the degree of enhancement depends on the number of faults that must be tested through each of the gate's inputs. In general, the more faults that must be tested through a gate input, the more the other inputs should be weighted to the NCV.

Define the number of *device inputs* (*NDI*) for each gate to be the number of PIs and SRLs that propagate directly to the gate. *NDI* will be used as a relative measure of the number of faults that are to be detected through a given gate. The desired ratio (*Ri*) of the NCV to the controlling value for each gate input is approximated by dividing *NDI* of the gate (NDI_g) by *NDI* of the inputs (NDI_i),

$$Ri = NDI_g/NDI_i . \quad (1)$$

Consider the three-input AND gate shown in **Figure 2**, which has six device inputs feeding it. As indicated, its three inputs are assumed to contain one, two, and three device inputs. More faults must be detected through the third input than through the others; this should result in the other inputs being weighted more heavily toward the NCV. Using Equation (1) to calculate the desired ratios of a '1' state to a '0' state for each input gives the following results:

$$R1 = NDI_g/NDI_i = 6/1 = 6,$$

$$R2 = NDI_g/NDI_i = 6/2 = 3,$$

$$R3 = NDI_g/NDI_i = 6/3 = 2.$$

This indicates that the first input is to receive six times more '1' values than '0' values, the second input three times more, and the third input only two times more.

Equation (1) can be used on each gate of an entire circuit to determine how to weight the PIs of the circuit. The algorithm which performs this calculation consists of the following steps:

1. Determine the *NDI* for all logic gates in the circuit.
2. Assign to each logic gate two numbers, called the "0 weight" (*W0*) and the "1 weight" (*W1*), and initialize both to '1'. The ratio of the final values of *W0* and *W1* for device inputs gives the desired odds of having a '0' placed on the input.
3. Perform a backtrace from each device output (POs and SRLs). As the backtrace goes from a gate *g* to a gate *i* driving one of its inputs, *W0* and *W1* of gate *i* ($W0_i$ and $W1_i$) are adjusted depending on the logical function of gate *g*. The weights for gate *i* resulting from the path from gate *g* are given in **Table 3** for the primitive logic functions of gate *g*, where *Ri* is the value calculated from Equation (1). The new value of $W0_i$ is the larger of *W0* calculated from Table 1 and the previous value of $W0_i$. Similarly, the new value of $W1_i$ is the larger of *W1* from the table and its previous value. The previous value may be either the initial value or the value calculated from another path.
4. Finally, for each device input, determine the following:
   - Weighted value (*WV*)
     WV represents the logical value to which the input is to be biased. If $W0 > W1$, $WV = 0$, else $WV = 1$.
   - Weighting factor (*WF*)
     WF indicates the amount of biasing toward the weighted value. It is calculated by dividing the larger of *W0* and *W1* by the smaller. *WV* and *WF* for a single input will define the weight for that input, and *WV* and *WF* for all inputs constitute a set of weights. The set of weights calculated by this algorithm is designated as the global set of weights.

• *Weight calculation example*

The circuit shown in **Figure 3** is used to illustrate the algorithm that generates a set of weights. The circuit contains five primitive logic gates, 12 device inputs, and two device outputs.

Step 1

The value of *NDI* is calculated for each logic gate and is shown in parentheses inside each gate. The value of *NDI* for a PI is 1 by definition.

Step 2

The *W0* and *W1* values for all gates and PIs are initialized to 1.

Step 3

A backtrace from both POs is performed to calculate *W0* and *W1* for all gates and PIs. PO1 is traced first, followed by PO2. The order has no effect on the final result and, in fact, the backtrace may even be performed in parallel as long as all calculations are carried to completion. The values at each step of the backtrace are given in **Table 4** as the trace goes from a gate *g* to a gate on an input *i*.

Step 4

The weight set is determined from the final values of *W0* and *W1* for all 12 PIs. The first four PIs are to be weighted so that the '1' state is favored by a factor of 8 to 1 (the probability of the '1' state is 8/9). PIs 5 through 8 receive a probability of a '0' state of 4/5, and PIs 9 through 12 receive a probability of a '1' state of 8/9. **Table 5** compares the test detection probability [36] of all nonequivalent single stuck-faults in the circuit for both a random test and the weighted random test resulting from this calculation. Note that the most random-pattern-resistant faults (PIs 1–4 stuck at '1', Input 1 of Gate 5 stuck at '1', etc.) all improved their test probability by at least a factor of 8. Further notice that some highly random testable faults (Output PO1 stuck at '0', Output PO1 stuck at '1', etc.) declined slightly in test probability. Since the probability of testing these faults was still high, this had no effect on the WRP test length. The test detection probabilities can be used to calculate an average test length (number of patterns required to achieve a 50% chance of detecting all faults). The WRP test reduces this average pattern length to 66 from the 600 pattern needed for a random test. This compares to 10 deterministic patterns which are required to test all of the faults in the circuit in Figure 3.

• *Hardware generation of weighted random values*

The weight calculation allows for the generation of a continuum of weighting factors; i.e., a particular device input can acquire any ratio of '1's to '0's. In practice, this proves to be prohibitively expensive to allow in hardware, so

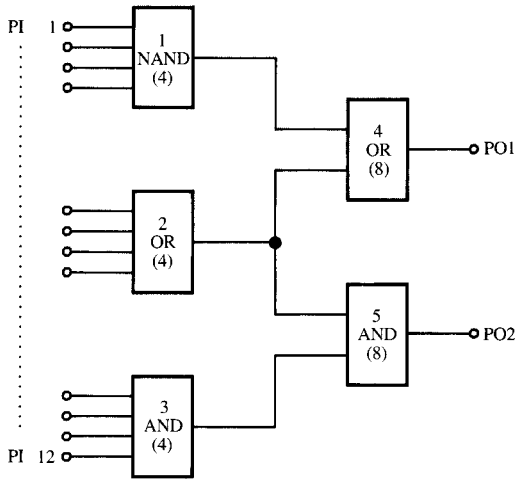

Figure 3

Weight-calculation circuit.

Table 4 Calculations of *W0* and *W1* from backtrace.

Calculation no.	*From (g)*	*To (i)*	$W0_i$	$W1_i$
1	PO1	Gate 4	1	1
2	Gate 4	Gate 1	2	1
3	Gate 1	PIs 1–4	1	8
4	Gate 4	Gate 2	2	1
5	Gate 2	PIs 5–8	8	1
6	PO2	Gate 5	1	1
7	Gate 5	Gate 2	2	2
8	Gate 2	PIs 5–8	8	2
9	Gate 5	Gate 3	1	2
10	Gate 3	PIs 9–12	1	8

Table 5 Fault-detection probabilities.

Fault	*RP test*	*WRP test*
PIs 1–4*	0.0039	0.032
PIs 5–8*	0.0076	0.078
PIs 9–12*	0.059	0.046
Gate 1 output†	0.059	0.154
Gate 2 output*	0.0076	0.314
Gate 2 output†	0.113	0.451
Gate 3 output*	0.819	0.221
Gate 4 input 2†	0.059	0.369
Gate 5 input 1*	0.0039	0.256
Gate 4 output*	0.0039	0.256
Gate 4 output†	0.996	0.744
Gate 5 output*	0.941	0.632
Gate 5 output†	0.059	0.368

* Stuck at 1
† Stuck at 0

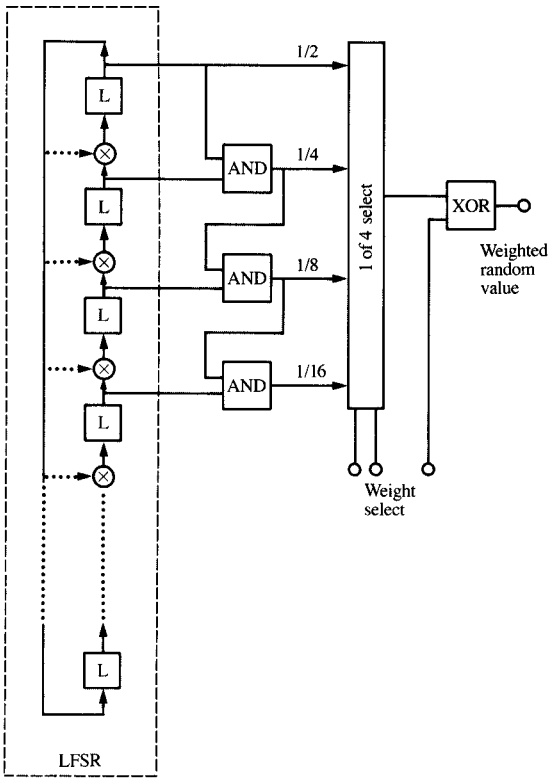

Figure 4

Hardware implementation of weight generator.

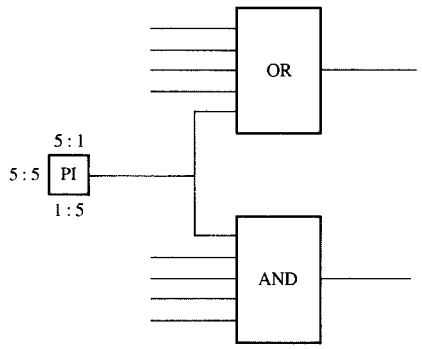

Figure 5

Conflicting weighting factors.

only a small number of weight factors that can easily be generated at a tester are used. A 32-bit linear feedback shift register (LFSR) is used as the source of the weighted bits. Instead of simply taking a random bit from the LFSR and applying it to the device under test, a group of bits is used to generate the single weighted bit, as shown in **Figure 4**. Latches are designated by Ls; XOR designates an exclusive-OR circuit. By taking, for instance, the last three bits in the LFSR and performing an AND function upon them, a bit with a weight factor of 7 is produced (7/8 probability of '0' or 7/8 probability of '1' with inversion). Similarly, by taking four bits at a time and performing the AND function on them, a bit with a weight factor of 15 is produced. This bit is then passed through an exclusive-OR gate whose other input is the desired weighted value, resulting in bits weighted toward the '1' or the '0' state. This particular design of a weighting generator produces weight factors of 1 (random), 3, 7, and 15 toward both the '0' and '1' states. The desired weight factors calculated by the algorithm are the closest allowable value. In addition to the standard weights, the zero and infinite weights which are equivalent to deterministic values are also available. Note that in order to eliminate any direct dependency between consecutively produced weighted bits, all of the bits used in producing a single weighted bit are shifted out of the LFSR before succeeding weighted bits are created.

- *Problems with a single set of weights*

In the attempt to find a weight that satisfies all random-pattern resistant faults from different paths to a single gate, there often occurs a competition of conflicting weights. If the weights are of opposite value and equal magnitude, this results in an unbiased weight which does nothing to improve testability.

An example of weight competition is shown in **Figure 5**. In this example the PI fans out to two gates, both of which desire large, but opposite, weights on the PI. The problem is how to weight the PI to minimize the number of random patterns necessary to test all the faults through both the OR gate and the AND gate. The OR gate heavily favors the '0' state on the PI, while the AND gate equally heavily favors the '1' state. The algorithm would calculate $W0 = 5$ and $W1 = 1$ for the PI from the OR gate path and $W0 = 1$ and $W1 = 5$ from the AND gate path. Since only the larger values are selected for the final values, the result is $W0 = 5$ and $W1 = 5$ for the PI. The effects of both gates have averaged out at the PI, resulting in a balanced weight being assigned to the PI. Whenever random-pattern-resistant faults result in conflicting weights, a single set of weights is not sufficient to test all faults.

In resolving conflicts, it should be noted that extreme care must be used in selecting weighting factors. The maximum benefit that can be gained by weighting a single input to the proper value (probability changed from 1/2 to 15/16) is an

improvement in test probabilities of about 2. However, the damage created by weighting this input to the incorrect value (probability changed from 1/2 to 1/16) can be as high as a factor of 8. Therefore, unless the calculation of the weights is done properly, the weights may do more harm than good.

5. Strategy for creating a complete WRP test

Since a single set of weights cannot be expected to always satisfy all the random-pattern-resistant faults in the device, the solution is to allow additional sets of weights. In addition to the previously described global set of weights, which attempts to make the entire device testable, additional sets of weights are created that focus on faults left untested. The untested faults are determined by fault simulation using the recently developed Parallel Pattern Single Fault Propagate (PPSFP) simulator [37].

After the patterns for the global set of weights are applied, the remaining faults are either random-pattern-resistant faults or redundant. A deterministic generator becomes the source of the additional sets of weights that will test the remaining faults. The advantages of using deterministic test generation are the following:

- Any fault that can be tested with deterministic patterns can be tested with WRP.
- The appropriate sequence necessary to detect the fault can be identified. Given where the deterministic pattern observed the fault and what clocks must be on to detect the fault, the corresponding sequence can be directly determined.
- In most cases, the deterministic test generator can identify redundant faults, which are then eliminated from further consideration.

A deterministic test may be translated to a set of weights by assigning a weight of 15/16 to the desired values for PIs and SRLs that are set by the test. The choice of 15/16 is desirable because it gives a high probability of testing the focal fault while still allowing a reasonable probability of detecting the remaining SSFs that require a similar test. In this respect, a single set of weights created by a deterministic test is more productive at testing faults than the deterministic test itself, which can only detect additional faults that require the identical test. This ability, plus the fact that most faults have already been detected by the global set of weights, minimizes the deterministic test-generation effort required to generate a complete WRP test. This sometimes results in improved single-stuck-fault coverage over a deterministic test, because of faults that resist deterministic test generation but are detected with WRPs.

To achieve the goal of testing all possible single stuck-faults in a minimum number of weighted random patterns and sets of weights, the following algorithm, depicted in **Figure 6**, was developed:

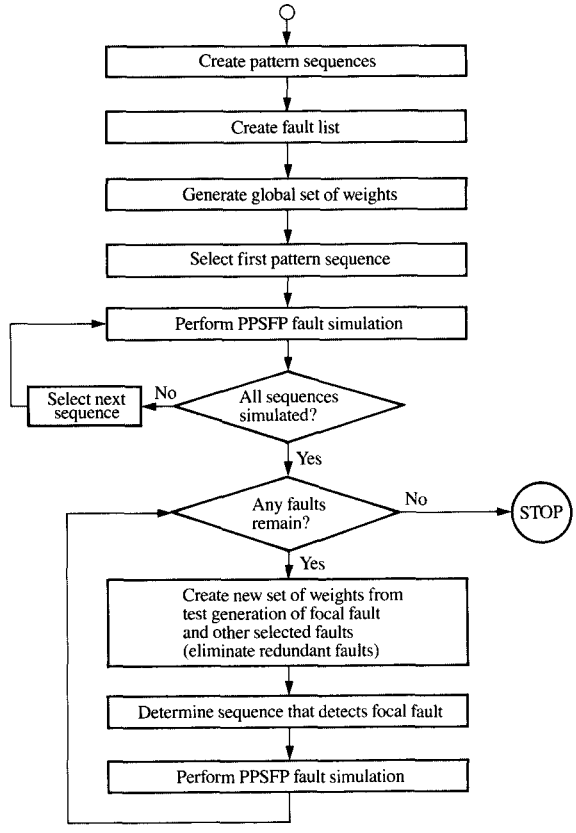

Figure 6

WRP test-generation algorithm.

1. Create the set of pattern sequences that can test all faults.
2. Create the set of single stuck-faults which are to be tested.
3. Calculate the global set of weights (described in Section 4).
4. For each sequence, use the PPSFP fault simulator to fault-simulate the precise WRPs that result from the selected set of weights for the selected sequence, eliminating faults which become detectable. Patterns are simulated 256 at a time, and when a new set of 256 patterns no longer detects at least one fault, the simulation is terminated. The number of patterns simulated up to this point is selected as the number of patterns to be applied for this set of weights for this sequence. As the simulation is performed, the expected signatures are calculated.
5. Select one fault (focal fault) still in the fault list and generate a deterministic test that can detect the fault. Assign a weight of 15/16 toward the desired value of all

PIs and SRLs set by the test. From the observation point of the fault and the clock (if any) which must be on for detection, the appropriate sequence that can detect the focal fault is determined. This becomes a new set of weights which has a high probability of detecting the focal fault. To maximize the number of faults detected by this set of weights, additional test generation is performed on faults selected from the fault list which reside in logic not set by previous test generations. If the new test is consistent with the selected sequence and results in five or fewer conflicting values on PIs/SRLs when compared to this set of weights, the inputs set by this test are also given a 15/16 probability to the desired state, except for conflicts which receive a 1/2 probability. If test generation identifies a fault as redundant, it is removed from the fault list. If the test generator fails to create a test (or prove redundancy) for a fault in the maximum allowed CPU time (1 s, IBM 3081), a weight set cannot be created for that fault. The fault is excluded from further test generation but continues to be simulated. If the fault is still undetected at the end of the WRP test, it is considered an untested fault, even though it may be redundant.

6. For the sequence that can detect the focal fault, repeat the simulation described in Step 4 to determine the faults that are detected with the new set of weights. In this case the simulation is continued until at least the focal fault is detected, and the simulation is then terminated by a new set of 256 patterns that fail to detect a new fault.
7. Repeat Steps 5 and 6 until no faults remain.

This algorithm can test any fault that can be detected with a deterministic test by controlling the degree to which the individual PIs and SRLs are weighted. Clearly, there will be more WRPs than deterministic patterns, but the ratio is now reasonably bounded. Empirical results on a large number of designs of varying size gave a range of about 10 to 50 times more WRPs than deterministic patterns.

With up to 50 times more patterns, one might question the efficiency of the calculation of the test coverage and signature. Great care was taken in constraining the design to be LSSD and constructing pattern sequences that could be represented by a combinational pattern. This environment can now be very profitably exploited using the PPSFP simulator to perform these calculations. An implementation of the WRP test-generation algorithm which has been run on more than 1600 chip designs of about 1000 logic gates required an average of only 2.1 CPU seconds (on an IBM 3081) to create and evaluate the full WRP test.

In our experience, the factor of 50 in *test patterns* is represented in less than 10% of the *test data* needed for a deterministic test. This is due to the WRP compact coding of the patterns. The complete WRP pattern set is repesented by initial seeds, sequence types and length, weighting factors, and response signatures—as opposed to the explicit coding of all the test stimuli and responses in the deterministic pattern set.

The other concern with a 50× factor in pattern count is *test application time.* In our environment, where an LT tester [38] has been modified to apply weighted random patterns, it has proved possible to apply the 50× WRP patterns in less time than a deterministic set. With a tester that had high-speed pattern buffers per pin, the test time advantage might be with the deterministic test for some product. However, the WRP method may still be the preferred method for the following reasons:

- The absolute time of the test, though longer than a deterministic one, is still small, since it can be run at product maximum scan speed, and hence attractive, particularly in view of the improved fault coverage which is achieved through the use of the WRP method.
- The WRP strategy and equipment would reduce test data requirements, particularly for high-density chips and modules.
- The tester implementing the WRP method could be less costly than the high-speed deterministic tester.

6. Benchmark results

The WPR test generator has been run on the 10 ISCAS benchmark circuits [39]. These circuits, which were selected as a representative mixture of actual designs, are combinational and range in size from 239 to 3827 logic gates. The primary inputs and outputs are considered gates and are included here in the logic-gate count. Two designs (C432 and C499) contain exclusive-OR gates that were remodeled into more basic logic gates. This results in additional logic gates and faults; 18 of these extra faults were found to be redundant for the C432 design. When more than one of the inputs to a gate come from a common gate, the faults on these inputs are identified as redundant without requiring test generation. The C3540 design contains six of these faults, while C1908 and C2670 have two each.

The WRP test-generation results are shown in **Table 6**. A description of each column follows:

1. Design	The assigned name for each circuit.
2. No. of gates	The total number of logic gates in the circuit, including PIs and POs.
3. No. of faults	The total number of fault-equivalence classes generated from the circuit model.
4. No. of wt. sets	The total number of sets of weights that were necessary to test all faults in the circuit.

Table 6 Benchmark results.

Design	*No. of gates*	*No. of faults*	*No. of wt. sets*	*No. of WRP patterns*	*CPU time* (3081-seconds)	*NUF*	*NRF*	*NTGEN*
C432	239	560	1	1024	0.6	0	22	22
C499	483	1158	2	1792	1.1	0	8	10
C880	469	942	2	1280	0.5	0	0	12
C1355	619	1574	3	2098	1.7	0	8	14
C1908	938	1879	6	5376	2.2	0	9	21
C2670	1566	2747	8	5888	16.3	11*	106	129
C3540	1741	2428	4	3840	3.5	0	137	137
C5315	2608	5350	2	2048	2.2	0	59	60
C6288	2480	7744	1	512	9.2	0	34	34
C7522	3827	7550	10	9728	13.9	0	131	175

* These 11 untested faults were redundant but were not identified by the WRP method as redundant.

5. No. of WRP patterns — The total number of WRP patterns necessary to test all faults in the circuit.
6. CPU time — The total CPU time, in IBM 3081-seconds, required to perform the entire WRP test generation. This includes the weight generation, fault simulation, and signature calculation.
7. NUF — The number of untested faults (not counting identified redundant faults) remaining at the end of test.
8. NRF — The number of faults identified as redundant by the deterministic generator.
9. NTGEN — The number of faults for which the WRP deterministic generator was invoked to create a test.

Examination of the results shows that all nonredundant faults were tested for each design. The C2670 design did have 11 faults which the WRP deterministic test generator could not identify as redundant but which are known to be redundant [40]. Eleven seconds of the WRP test generation time were spent in an unsuccessful effort to generate a test for these 11 faults. The number of weight sets varied from 1 to 10, and no design required more than 10000 WRP patterns to test all faults. The IBM 3081 CPU time required to perform the complete WRP procedure for the largest design was only 13.9 seconds. Compared to a deterministic approach, little effort was spent on test generation and, in fact, most of the test generation was performed on redundant faults.

Concluding remarks

A new method has been presented in this paper for generating weighted random patterns for testing LSSD logic devices. The advantages observed in using the WRP procedure are the following:

- It provides full stuck-fault coverage, unlike a random-pattern test, and can provide improved coverage of nonmodeled faults compared to a deterministic pattern approach.
- It has the ability to apply at least an order of magnitude more patterns to a chip or module than a deterministic test in comparable time in our tester environment. This is accomplished with an order of magnitude less precalculated test data.
- The computer time spent to generate the weighting data and determine the number of pattern sequences to obtain full stuck-fault coverage is small compared to that used in generation of a deterministic test.
- These advantages are realized with no circuit overhead other than that required for the inclusion of LSSD.

Using a method such as the WRP method, it is now possible to begin to realize the data and application advantages promised for random-pattern testing, while also obtaining an improvement in fault coverage.

Acknowledgments

The authors gratefully recognize the contributions of F. Motika, whose mastery of tester hardware/software was a constant guide, and D. Giramma, who worked on the early development of the WRP algorithm. We are indebted to R. Langmaid for his calculation of optimal weights for isolated logic gates. The management support of T. Y. Tao and E. Nielsen was essential throughout the project. Special thanks are due B. Koenemann for his careful reading of the original manuscript.

References

1. S. Seshu, "Improved Diagnosis Program," *IEEE Trans. Electron. Computers* **EC-14**, 76–79 (1965).
2. M. Nagamine, "An Automated Method for Designing Logic Circuit Diagnostic Programs," *Proceedings*, SHARE/ACM/IEEE Design Automation Workshop, 1971, pp. 236–241.

3. J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electron. Computers* **EC-16,** 567–580 (1967).
4. V. D. Agrawal and P. Agrawal, "An Automatic Test Generation System for Illiac IV Logic Boards," *IEEE Trans. Computers* **C-21,** 1015–1017 (1972).
5. N. Benowitz, D. F. Calhoun, G. E. Alderson, J. E. Bauer, and C. T. Joeckel, "An Advanced Fault Isolation System for Digital Logic," *IEEE Trans. Computers* **C-24,** 489–497 (1975).
6. H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Trans. Computers* **C-24,** 695–700 (1975).
7. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings,* 14th Design Automation Conference, 1977, pp. 462–468.
8. P. Goel, "Test Generation Costs Analysis and Projections," *Proceedings,* 17th Design Automation Conference, 1980, pp. 77–84.
9. P. H. Bardell and W. H. McAnney, "A View from the Trenches: Production Testing of a Family of VLSI Multichip Modules," *Proceedings,* International Symposium on Fault-Tolerant Computing, 1981, pp. 281–283.
10. E. B. Eichelberger and E. Lindbloom, "Trends in VLSI Testing," *Proceedings,* IFIP TC 10/WG 10.5 International Conference on Very Large Scale Integration (VLSI 83, Trondheim, Norway), 1983, pp. 339–348.
11. T. W. Williams and E. B. Eichelberger, "Random Patterns Within a Structured Sequential Logic Design," *Digest of Papers,* 1977 IEEE Semiconductor Test Symposium, pp. 19–26.
12. B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Techniques," *Digest of Papers,* 1979 IEEE Test Conference, pp. 37–41.
13. P. H. Bardell and W. H. McAnney, "Self-Testing of Multichip Logic Modules," *Digest of Papers,* 1982 International Test Conference, pp. 200–204.
14. D. Komonytsky, "LSI Self-Test Using Level-Sensitive Scan Design and Signature Analysis," *Digest of Papers,* 1982 International Test Conference, pp. 414–424.
15. E. B. Eichelberger and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM J. Res. Develop.* **27,** 265–272 (1983).
16. A. J. Blodgett and D. R. Barbour, "Thermal Conduction Module: A High-Performance Multilayer Ceramic Package," *IBM J. Res. Develop.* **26,** 30–36 (1982).
17. J. J. Shedletsky and E. J. McCluskey, "The Error Latency of a Fault in a Combinational Digital Circuit," *Proceedings,* International Symposium on Fault-Tolerant Computing, 1975, pp. 210–214.
18. J. Savir and P. H. Bardell, "On Random Pattern Test Length," *IEEE Trans. Computers* **C-33,** 467–474 (1984).
19. T. W. Williams, "Test Length in a Self-Testing Environment," *IEEE Design Test* **2,** 59–63 (April 1985).
20. C. K. Chin and E. J. McCluskey, "Test Length for Pseudo-Random Testing," *IEEE Trans. Computers* **C-36,** 252–256 (1987).
21. F. Motika, J. A. Waicukauski, E. Lindbloom, and E. B. Eichelberger, "An LSSD Pseudo Random Pattern Test System," *Proceedings,* International Test Conference, 1983, pp. 283–288.
22. A. J. Briers and K. A. E. Totton, "Random Pattern Testability by Fast Fault Simulation," *Proceedings,* International Test Conference, 1986, p. 274–281.
23. R. G. Carpenter, E. Lindbloom, and T. M. McMahon, "Statistical Logic Test System Having a Weighted Random Test Pattern Generator," U.S. Patent 3,719,885, March 1973.
24. P. Agrawal and V. D. Agrawal, "On Improving the Efficiency of Monte Carlo Test Generation," *Digest of Papers,* 5th International Symposium on Fault Tolerant Computing, 1975, pp. 392–398.
25. K. P. Parker, "Adaptive Random Test Generation," *J. Design Aut. Fault-Tol. Comput.* **1,** 62–68 (1976).
26. K. P. Parker, "Compact Testing: Testing with Compressed Data," *Digest of Papers,* 6th International Symposium on Fault-Tolerant Computing, 1976, pp. 93–98.
27. C. Timoc, F. Stott, K. Wickman, and L. Hess, "Adaptive Self-Test for a Microprocessor," *Proceedings,* International Test Conference, 1983, pp. 701–703.
28. K. J. Lieberherr, "Parameterized Random Testing," *Proceedings,* 21st Design Automation Conference, 1984, pp. 510–516.
29. H. Wunderlich, "PROTEST: A Tool for Probabilistic Testability Analysis," *Proceedings,* 22nd Design Automation Conference, 1985, pp. 204–211.
30. H. Wunderlich, "On Computing Optimized Input Probabilities for Random Tests," *Proceedings,* 24th Design Automation Conference, 1987, pp. 392–398.
31. R. Lisanke, F. Brglez, A. de Geus, and D. Gregory, "Testability-Driven Random Pattern Generation," *Proceedings,* International Conference on Computer-Aided Design, 1986, pp. 144–147.
32. E. B. Eichelberger, R. Langmaid, E. Lindbloom, F. Motika, J. L. Sinchak, and J. A. Waicukauski, "Weighted Random Pattern Testing Apparatus and Method," U.S. Patent No. 4,687,988, August 1987.
33. F. Motika and J. A. Waicukauski, "Weighted Random Pattern Testing Apparatus and Method," U.S. Patent No. 4,688,223, August 1987.
34. E. J. McCluskey, *Logic Design Principles,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986, pp. 457–461.
35. J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iygenar, "Transition Fault Simulation by Parallel Pattern Single Fault Propagation," *Proceedings,* International Test Conference, 1986, pp. 542–549.
36. K. P. Parker and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks," *IEEE Trans. Computers* **C-24,** 668–670 (1975).
37. J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy, "A Statistical Calculation of Fault Detection Probabilities by Fast Fault Simulation," *Proceedings,* International Test Conference, 1985, pp. 779–784.
38. R. N. Powell, "IBM's VLSI Logic Test System," *Proceedings,* International Test Conference, 1981, pp. 388–392.
39. F. Brglez, P. Pownall, and R. Hum, "Accelerated ATPG and Fault Grading via Testability Analysis," *Proceedings,* IEEE International Symposium on Circuits and Systems, 1985, pp. 695–698.
40. M. Schulz and E. Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," *Proceedings,* 18th Symposium on Fault-Tolerant Computing, 1988, pp. 30–35.

Received January 20, 1988; accepted for publication September 21, 1988

John A. Waicukauski *IBM General Technology Division, East Fishkill facility, Route 52, Hopewell Junction, New York 12533.* Mr. Waicukauski is a senior engineer at the East Fishkill facility, where he is currently working on the development of software tools for creating, evaluating, and diagnosing random-pattern tests of VLSI devices. He earned a B.S. degree in chemistry from Southern Illinois University in 1971 and an M.S. degree in physical chemistry from the University of Illinois in 1973. Mr. Waicukauski received an IBM Outstanding Technical Achievement Award for the development of a chip and multi-chip module diagnostic system, and an IBM Outstanding Innovation Award and Corporate Award for the development of the weighted random-pattern testing system. He has reached his second IBM Invention Plateau and holds four U.S. patents.

Eric Lindbloom *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Mr. Lindbloom retired from IBM in 1987 as a senior engineering manager; he has returned to work as a consultant on VLSI testing at the Kingston facility. He received a B.A. in philosophy from the University of Michigan in 1957. He joined IBM in 1963 and during most of his career pursued interests in test-pattern generation, simulation, and diagnosis for logic-circuit final testing. In 1986, Mr. Lindbloom received an IBM Outstanding Innovation Award, and in 1988 an IBM Corporate Award, for his work on the development of the weighted random-pattern testing system. He has reached his second IBM Invention Plateau and holds five U.S. patents. Mr. Lindbloom is a senior member of the Institute of Electrical and Electronics Engineers.

Edward B. Eichelberger *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Dr. Eichelberger is an IBM Fellow and the manager of the Advanced VLSI Technology and Testing Department at the Kingston facility. He earned his B.S. degree from Lehigh University and his Ph.D. in electrical engineering from Princeton University. 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. He has received three IBM Division Awards and two IBM Corporate Awards. He has reached his seventh IBM Invention Plateau, and holds 18 U.S. patents. Dr. Eichelberger is a Fellow of the Institute of Electrical and Electronics Engineers.

Orazio P. Forlenza *IBM General Technology Division, East Fishkill facility, Route 52, Hopewell Junction, New York 12533.* Mr. Forlenza is a senior associate engineer in the New Products and Systems Department at the East Fishkill facility. Since joining IBM in 1985, he has worked in the areas of chip/module test validation and weighted random-pattern test generation. He received a B.S. degree in electrical engineering from the Polytechnic Institute of New York in 1985, and is currently pursuing an M.S. in electrical engineering from Syracuse University.