Cellular Automata for Weighted Random Pattern Generation

Danial J. Neebel, *Member*, *IEEE*, and Charles R. Kime, *Fellow*, *IEEE*

Abstract—Fault testing random-pattern-resistant circuits requires that BIST (built-in self-test) techniques generate large numbers of pseudorandom patterns. To shorten these long test lengths, this study describes a cellular automata-based method that efficiently generates weighted pseudorandom BIST patterns. This structure, called a *weighted cellular automaton* (WCA), uses no external weighting logic. The design algorithm MWCARGO combines generation of the necessary weight sets and design of the WCA. In this study, WCA pattern generators designed by MWCARGO achieved 100 percent coverage of testable stuck-at faults for benchmark circuits with random-pattern-resistant faults. The WCA applies complete tests much faster than existing test-per-scan techniques. At the same time, the hardware overhead of WCA proves to be competitive with that of current test-per-clock schemes.

Index Terms—Built-in self-test, weighted random patterns, multiple weight sets, cellular automata, hybrid cellular automata, weighted cellular automata, test-per-clock pattern generation.

1 Introduction

The complex digital systems of today mandate the use of design-for-testability (DFT) techniques. One such technique is built-in self-test (BIST). When applications require short test times and high fault coverage, BIST techniques that test with pseudorandom patterns are a cost-effective solution. Stored deterministic testing involves large overhead and low unmodeled fault coverage [1]; pseudorandom testing, however, provides low overhead and high fault coverage for most circuits. Some circuits, however, are inherently random-pattern test resistant and, therefore, require prohibitively large numbers of equiprobable patterns.

In efforts to solve this problem, researchers [2], [3], [4], [5] have found that weighted pseudorandom pattern testing provides significantly shorter test lengths than pseudorandom pattern testing. Analysis by Ströle and Wunderlich [1] and others [6] show that weighted pseudorandom pattern testing needs multiple weight sets in order to achieve extremely high fault coverage, i.e., 100 percent of all detectable single stuck-at faults (the fault class considered here).

Research literature [2], [7], [8] discusses scan-based methods of delivering weighted patterns with multiple weight sets. The length of time required to load long scan chains with large numbers of vectors has led some researchers to investigate methods of generating and applying vectors directly to the inputs of the circuit under test (CUT) [9], [5], [10], [7]. Two problems have arisen in these techniques. Bou-Ghazale and Marinos [5] address one problem—added overhead from extra flip-flops. Another problem is added delay during normal operation and testing caused by placing the weighting logic between the storage elements and the normal logic. This structure and its relation to delay are shown in Fig. 1. The multiplexer between the normal storage element, *d*, and the circuit adds delay to the critical path in normal operation. Also, the weighting logic is in series with the normal logic under test and, thus, adds further delay during testing. Delay behaviors of three test-per-clock weighted pseudorandom approaches are discussed later in the paper.

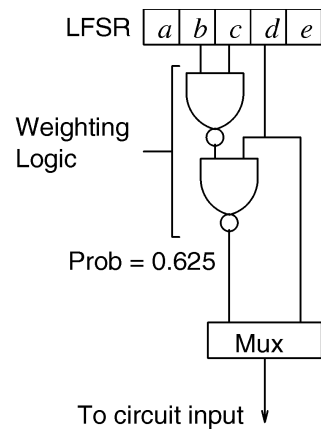


Fig. 1. The problem of added delay for some weighted random pattern generation circuits. Storage elements b and c are extra storage elements required to generate the bit with probability of 0.625.

This paper presents a different structure that can generate patterns with multiple weight sets at a test-per-clock rate. This structure, the *Weighted Cellular Automaton* (WCA), was first presented in [11]. The weight-logic outputs are joined to the normal logic paths using the same multiplexers at flip-flop inputs as those for serial scan, thus adding little delay to normal operation. Unlike the weight logic of the circuit in Fig. 1, the weight logic of the WCA evaluates in parallel with the logic under test. This is because the

- *D.J. Neebel is with the College of Integrated Science and Technology, James Madison University, Harrisonburg, VA 22807. E-mail: neebeldj@jmu.edu.*
- *C.R. Kime is with the Department of Electrical and Computer Engineering, University of Wisconsin, 1415 Engineering Drive, Madison, WI 53706. E-mail: kime@engr.wisc.edu.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105245.

0018-9340/97/$10.00 © 1997 IEEE

weight logic is built into the pattern generation structure. The computation of the next weighted pattern and the application of the current weighted pattern occur simultaneously. Consequently, during testing, this structure adds no weight-logic delay to normal-operation delay.

Using a *hybrid cellular automata* (HCA) for the WCA allows building the weight logic into the structure. Fig. 2 features an example HCA register that has *null boundary conditions*, meaning that the cell inputs at the ends of the register are set to zero. The HCA contains rule 90 and 150 cells and is a maximal-length sequence generator called a 90/150 Linear Hybrid CA or LHCA [12], [13].[1] Hortensius [14] discusses the possibility of using nonlinear uniform CA to generate weighted patterns and presents the probabilities of each site for small uniform CA, but does not detail nonlinear HCA.

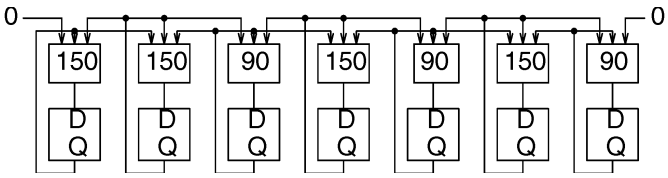


Fig. 2. An example one-dimensional three-cell neighborhood cellular automaton.

The succeeding two sections further describe WCA and a design algorithm (MWCARGO) for use with WCA to generate multiple weight sets. In the fourth section, the testing lengths of the WCA used with the MWCARGO algorithm are compared to those of the best test-per-scan technique. Also in the fourth section, the delay and hardware overhead of WCA and those of two other test-per-clock methods are compared. Based on these comparisons, the final section concludes that the combination of WCA and MWCARGO can generate weighted pseudorandom test patterns accurately, quickly, and at moderate cost.

2 The Weighted Cellular Automaton, WCA

The term "weighted" refers to the patterns that the WCA generates, not its structure. Each site in a WCA may have a different rule, so the different WCA sites generate the value one with different probabilities. Combining several different rules into a single CA register allows WCA to generate patterns with a given distribution of probabilities by using a single register and no external weighting logic. Fig. 3 shows a small WCA and the probability of a 1 at each of the outputs.

The design of a WCA involves choosing the proper rule for each cell. Because there are many dependencies between locations, this process is not straightforward. The following sections provide more background on this process and other characteristics of WCA.

1. The notation used for specifying CA rules can be found in any of [11], [12], [13].

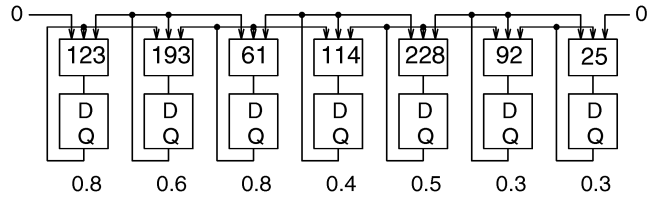


Fig. 3. A small WCA and the probabilities associated with each site.

2.1 WCA Evolutions

An often discussed and studied characteristic of CA is the patterns generated by the CA outputs at consecutive time steps, called *evolutions*. A 66-site WCA generated the pattern of 90 evolutions on the left in Fig. 4. A black spot indicates the value 1 and white indicates 0. The sites run horizontally and the evolutions progress from top to bottom. Note how, after a few evolutions, the WCA values at some sites appear random, while values at other sites are fixed. The WCA behaves this way because some of its sites are designed to produce probabilities that are 0.5 and other sites are designed to produce probabilities closer to 0 or 1. Those sites that remain at 0 after only a few evolutions are designed for probabilities less than 0.3.

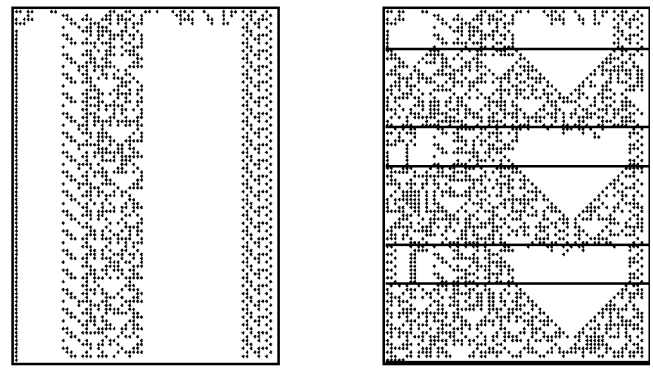


Fig. 4. Evolutions for a 66 site WCA without (left) and with (right) reinitialization.

The fixed and short-cycle sections of the WCA need frequent reinitialization or else they produce few unique patterns. Reinitialization is performed by changing the CA function, or set of rules, from the WCA to an LHCA repeatedly for a random number of clock cycles. To maximize randomization in a short number of cycles, the LHCA is constructed of maximal length sequence generators. The pattern at the right in Fig. 4 shows the evolutions for a WCA reinitialized by an LHCA. During reinitialization, the WCA runs for 10 clocks, followed by the LHCA running for 20 clocks. The values for each *run interval*, or number of clocks before switching the CA function, were chosen for illustrative purposes only. The horizontal lines mark the change from one CA function to the other. In this pattern, the WCA runs first and the sequence repeats three times.

The authors' short study [15] of WCA run intervals showed that a run interval of between 15 and 31 worked best in most cases. Although the WCA runs for a fixed interval, the LHCA does not. The run interval for the LHCA is determined by five bits of a 15-bit LFSR, so that

the average run interval for the LHCA matches the run interval of the WCA. This structure provides more effective randomization without significantly increasing test length.

2.2 Achievable Accuracy of WCA

Analysis by Ströle and Wunderlich [1] indicates that the effect of lack of accuracy in weights is to increase the test length. They showed that a resolution of 0.125 (1/8th) for an accuracy of ±0.0625 does not significantly increase the test length. The following describes an experiment to estimate the accuracy of the WCA.

Due to the nonlinear, hybrid nature of WCA, analytically determining its achievable accuracy on probabilities is very difficult. Even determining the output probabilities for a given WCA is #P-complete. Since a direct solution was not feasible, an experiment was performed to estimate the lower bound on the local achievable accuracy for a general WCA at a given site *i*. Accurate simulation of all rules for all cells of a WCA would show exactly the achievable probabilities and accuracy at all sites, but would require years of simulation time. Therefore, a small section of a WCA was analyzed with the 168 different rules used for WCA design (shown in Table 1).[2] Fig. 5 shows the model WCA section for the experiment. If sites $i - k$ to $i + k$ make up the section of WCA in the experiment, then k must be as large as possible but small enough to keep the experiment tractable. The set of 168 rules yield 168^{2k+1} distinct WCA to simulate. The number of rules is less than 256 because some rules have undesirable properties with respect to ability to generate both a 0 and a 1 and with respect to dependency upon neighboring sites. Setting $k = 2$ implies over 100 billion distinct WCA. Thus, $k = 2$ is too high. For $k = 1$, there are 4,741,632 distinct WCA sections, a large but tractable number.

Values 0.1, 0.2, 0.3, …, 0.9 are the range of probabilities for the boundary cells c_{i-2} and c_{i+2}. To determine the accuracy, the desired probabilities for all three modeled cells were set to the multiples of 0.1 between 0.1 and 0.9 (the same range as the boundary conditions). Then, the experiment sought a triplet of cell rules that gives probabilities for c_{i-1} and c_{i+1} within ±0.05 of the desired ones and gives the smallest error possible for the probability of c_i. For example, consider the first entry in Table 2, in which the desired probabilities for sites c_{i-1} and c_{i+1} are 0.6 and 0.1, respectively (falling in the intervals 0.55 to 0.65 and 0.0 to 0.15, respectively). In this entry, the desired probability for c_i is 0.3 and the boundary conditions for c_{i-2} and c_{i+2} are 0.4 and 0.5, respectively. To find this entry's minimum achieved probability error, the closest actual probability of a 1 to 0.3 had to be found, with the probability of cell c_{i-1} being 1 between 0.55 and 0.65 and that of cell c_{i+1} being 1 between 0.0 and 0.15. Rather than simulate a WCA section, the actual probabilities for such a small WCA were found exactly using an eight-state discrete-time Markov model.

2.2.1 Results of Accuracy Experiment

Two experiments used the limited set of rules from Table 1, with run intervals of 15 and 31. With this restricted set of rules, the experiment results show that almost all desired probabilities can be met locally within ±0.05. Of 32,805 (45 · 9 · 9 · 9) different combinations of boundary conditions and probabilities, 136, or 0.41 percent, failed to meet the desired probability for c_i within ±0.05 with run interval 15 and 192, or 0.59 percent, failed with run interval 31. Table 3 is a histogram showing how many combinations fell within seven error ranges.

The increase in error for increase in run interval is due to lock-up of cell values, as shown in the left side of Fig. 4. A more detailed analysis of these transient effects appears in [15]. As was shown in Fig. 4, some sites of the CA tend to lock up after even a few evolutions. This lockup drives the probabilities of 1 toward 0 or 1, making it more difficult to achieve probabilities close to 0.5 in adjacent cells. Closer analysis of the experimental results revealed that only when the desired probability for cell *i* is 0.5 and the desired for cell $i - 1$ or $i + 1$ is an extreme (0.1 or 0.9), did the error from the desired probability for cell *i* exceed 0.05, as in the last entry of Table 2.

TABLE 1
RULES USED FOR ALL EXPERIMENTS IN THIS PAPER

Rules								
1	33	65	94	116	142	164	191	225
2	36	66	95	118	144	165	193	226
5	37	67	96	120	145	166	194	227
6	38	70	97	121	146	167	197	228
9	39	71	98	122	147	169	198	229
10	40	73	99	123	148	172	199	230
16	41	74	100	124	149	173	201	231
18	43	75	101	125	150	175	202	232
20	44	77	103	126	151	177	203	233
22	45	78	104	127	152	178	209	235
23	46	80	105	129	154	180	210	245
24	52	82	106	130	155	181	211	246
25	53	83	107	131	156	182	212	247
26	54	86	108	133	157	183	214	249
27	56	88	109	134	158	184	215	250
28	57	89	110	135	159	185	216	251
29	58	90	111	137	160	188	217	
30	61	91	113	139	161	189	218	
32	62	92	114	141	163	190	219	

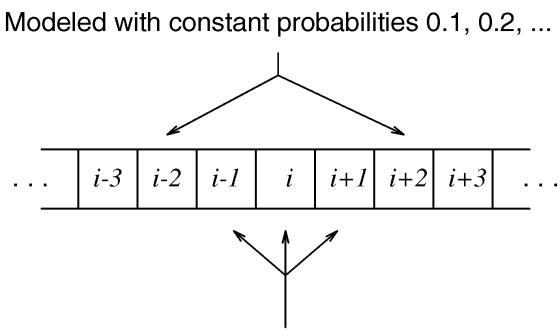


Fig. 5. WCA segment model for the experiment.

2. A discussion of these rules can be found in [15].

TABLE 2
EXAMPLES OF DESIRED PROBABILITIES, BOUNDARY CONDITIONS, AND CORRESPONDING RULES AND ERRORS

Desired Probability			Boundary	Absolute Error	Rules		
c_{i-1}	c_i	c_{i+1}	Conditions	for Cell c_i	c_{i-1}	c_i	c_{i+1}
0.6	0.3	0.1	0.4 0.5	0.0000612	2	5	139
0.8	0.2	0.4	0.4 0.4	0.0000045	229	75	154
0.3	0.9	0.1	0.6 0.1	0.0000017	2	247	129
0.8	0.1	0.1	0.3 0.2	0.0000182	214	46	20
0.6	0.5	0.1	0.4 0.5	0.0552083	2	172	209

TABLE 3
EXPERIMENT RESULTS FOR DETERMINING ACHIEVABLE PROBABILITIES

	Combinations In Range	
Error Range	$RI = 15$	$RI = 31$
$x < 0.05$	32669	32613
$0.05 \leq x < 0.06$	100	116
$0.06 \leq x < 0.07$	24	48
$0.07 \leq x < 0.08$	8	0
$0.08 \leq x < 0.09$	0	8
$0.09 \leq x < 0.10$	4	0
$x \geq 0.10$	0	20
Worst Error	0.095	0.216

In short, the results show that, locally, a WCA can generate weighted sequences with accurate arbitrary probabilities. This local experiment indicates that a general WCA is likely to generate patterns with a given set of probabilities with a few exceptions. To accommodate these exceptions, the design algorithm (MWCARGO) includes a feature discussed in Section 3.3.3 that achieves the accuracy of probabilities within ±0.05.

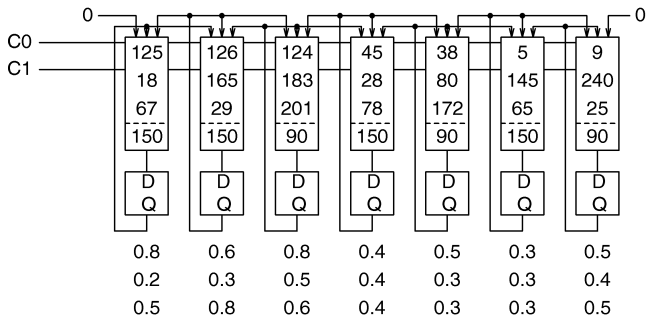


Fig. 6. A small MWCA.

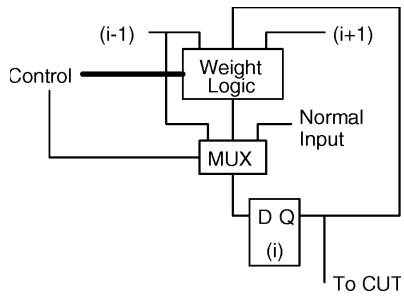


Fig. 7. MWCA cell block diagram.

2.3 Multiple WCA, MWCA

MWCA contain several sets of function rules—one set of functions for each weight set plus an LHCA function set. Fig. 6 shows a seven-site MWCA with three WCA function sets and an LHCA function set. The sets of probabilities for each function set appear in rows below the MWCA. Fig. 7 features a block diagram for an individual cell.

Section 2.1 noted that a WCA is paired with an LHCA in order to reinitialize the WCA. For an MWCA with multiple weight sets, many other operation modes are possible. Fig. 8 displays the mode used in this study because of its consistency with the steps taken in the MWCA design algorithm, which is discussed in the next section. Time flows from the top of the diagram to the bottom. Each section of the figure indicates the CA rules set by which the MWCA is configured during the corresponding phase of MWCA operation.

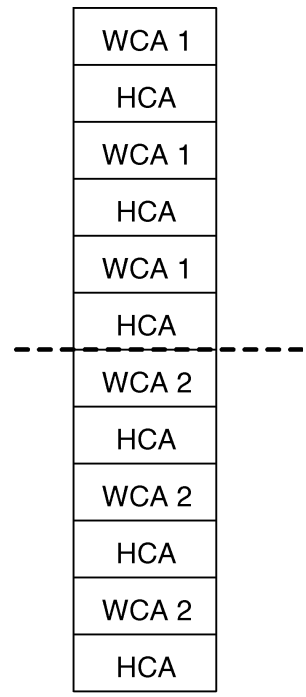


Fig. 8. MWCA operation mode.

3 MULTIPLE WCA DESIGN ALGORITHM

The previous section described the operation and various characteristics of the MWCA. A necessary part of the process for learning about MWCA was the development of a design algorithm first presented in [16]. That design algorithm is the focus of this section.

The MWCA design algorithm works to achieve 100 percent coverage of all detectable faults while limiting the number of weight sets (hardware overhead) and the test length within a specified maximum. Fig. 9 gives the top-level description MWCARGO (Multiple Weighted Cellular Automata Register Generation and Optimization). The specific tools designed for each step appear in parentheses at the end of each line.

```
BEGIN MWCARGO
  Determine_Hard_To_Detect_Faults;   (CASIM)
  Runs = TL_user/(2 × RI × WS_user)
  WHILE (Fault_List_Not_Empty
         AND Fault_List_Changed) DO
     Generate_Test_Patterns;          (ATPGX)
     Partition_Pattern_Set;           (PPP)
     Assign_Weights;                  (PPP)
     Generate_Single_WCA;             (WCARGO)
     Fault_Simulate_all_WCA;          (CASIM)
     Extract_Undetected_Faults;       (CASIM)
  END WHILE
END MWCARGO
```

Fig. 9. Top level of the Multiple WCA design algorithm, MWCARGO.

In the first step, CASIM fault simulates patterns from a 90/150 LHCA to determine if all faults can be covered by random patterns within the user set maximum test length. A 90/150 LHCA maximal length sequence generator was chosen because that is the structure used within the WCA for reinitialization. MWCARGO assumes that any uncovered faults are hard-to-detect. CASIM is the combination of a simple CA simulator and ppcpt [17], a parallel pattern critical path tracing fault simulator.

The equation in the second step of MWCARGO calculates the variable, *Runs*, from the user set parameters of *User Set Test Length* (TL_{user}), *User Set # of Weight Sets* (WS_{user}), and *Run Interval* (*RI*). *Runs* is defined as the number of WCA re-initializations using the LHCA. In the equation, the factor of two accounts for the WCA and LHCA run intervals that are, on average, equal. Each time a new WCA structure is added to the MWCA, the test length increases by $2 \times RI \times Runs$ because the MWCA structure is built one WCA (with the LHCA for reinitialization) at a time. This process guarantees that the simulated patterns are the same as those generated by the physical hardware.

Once the hard-to-detect faults (HTD faults) and the test length are known, ATPGX finds a set of test vectors for the HTD faults. P^3 then partitions the vectors into multiple sets and determines weights for each set. Then, WCARGO generates a new WCA for one weight set (first in the list). Finally, CASIM determines which faults remain undetected. If any faults remain, the process then continues with the ATPGX step. The next three sections describe ATPGX, P^3, and WCARGO in more detail.

3.1 Generating Patterns, ATPGX

The ATPGX program performs test generation and fault simulation with fault dropping. After test generation (using the PODEM algorithm [18]) is complete, the new vector is simulated with each bit set to X in the order that PODEM set the bits to 0 or 1 during the search. If the bit can be set to X and the new vector still detects the target fault, the bit remains X. If not, the bit is changed back to its original value. The program performs this process to increase the number of unspecified (X) entries in the test vectors. Having more unspecified entries permits P^3 to better tune the weights to the remaining specified entries, which are critical to producing tests for the faults using weighted random pattern generation. Once all bits have been set to X (and reset to 0 or 1, as necessary), ATPGX fault simulates the vector using three-valued single-fault propagation to find and drop all covered faults remaining in the fault list. ATPGX continues until all faults are detected, marked as undetectable, or aborted.

3.2 Probabilities from Partitioned Patterns, P^3

P^3 takes as input a set of patterns (test vectors from ATPGX), a desired test length (TL_{user}), and a level of confidence that all patterns will be generated within the user-set test length. P^3 starts with a single set of patterns, then creates more sets with patterns from the existing sets–one new set at a time, as shown in Fig. 10—until the test length is less than the user-set maximum. Fig. 11 shows the pseudocode for the partitioning algorithm.

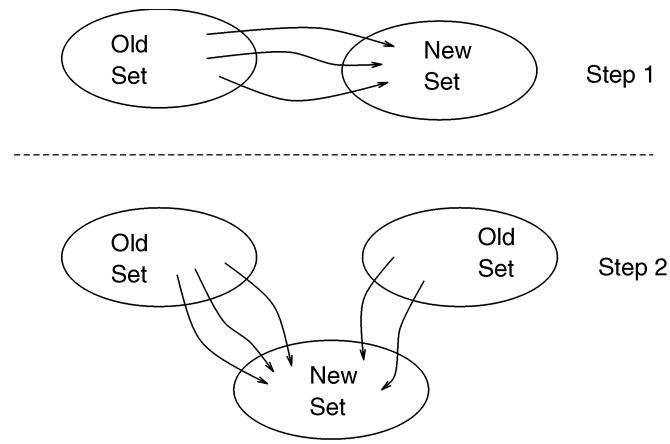


Fig. 10. Actions in the first two steps of P^3.

```
BEGIN
  Place_All_Patterns_Into_Single_Set

  WHILE Test_Length > Maximum DO
    WHILE (Change_In_Cost < 0) DO
       FOR ALL Patterns J IN ALL Sets S
          Evaluate_Partition(J,S);
          Save_Best_Pattern(J);
       END FOR
       New_Set = New_Set + Best_Pattern;
    END WHILE
    New_Partition = Current_Partition + New_Set;
    Test_Length = Find_Test_Length(New_Partition);
    Current_Partition = New_Partition;
  END WHILE
END
```

Fig. 11. Pattern partitioning algorithm for P^3.

The function Evaluate_Partition finds a pattern to move to a new set using a "cost" function that estimates the probability of a pattern occurring. Assuming all vectors are in one set at the beginning, the cost of placing a pattern in a set, *S*, is the sum of the costs for each bit in the pattern. The cost for bit v_{ij} in pattern V_j is the number of bits in position *i* of all patterns in *S* that disagree with v_{ij}. The cost of bits with value X is defined as 0. P^3 places the pattern that causes the sharpest decrease in total cost in the new set.

Table 4 includes an example cost calculation for a small set of vectors. The cost for bit 4 of vector V4 is the number of 1s in column 4, since bit 4 of vector V4 is a 0. The cost for vector V4, C(V4), is the sum of the cost of its bits, which is 6. In this example, V4 has the highest cost, so it will be moved to the new empty pattern set. If P^3 cannot find a vector with lower cost in the new set, then P^3 considers the new set complete and checks the test length of the new partition using Find_Test_Length.

TABLE 4
EXAMPLE COST CALCULATION FOR P3

Vec	v_{1j}	v_{2j}	v_{3j}	v_{4j}	v_{5j}	v_{6j}	$\mathcal{C}(V_j)$
V1	X	0	X	1	X	1	1
V2	1	X	X	X	0	1	3
V3	0	X	X	1	0	1	4
V4	0	0	X	0	1	1	6
V5	1	X	X	X	X	1	2
1s	2	0	0	2	1	5	–
0s	2	2	0	1	2	0	–
$\mathcal{C}$(V4)	2	0	0	2	2	0	6
Weights	0.50	0.10	0.50	0.67	0.33	0.90	

Find_Test_Length first assigns the weights using the method in [9] by counting 0s and 1s in a column. The weight for a given primary input is the number of 1s divided by the sum of the number of 0s and number of 1s. Xs are ignored for weight assignment purposes because they need not be set to any particular value to detect faults. For a column with no 0s or 1s, the weight value is 0.5. The user can set minimum and maximum values for the weights; their default settings are 0.1 and 0.9. The weights are given in the last row of Table 4.

Find_Test_Length then uses the weight sets to calculate the necessary test length, which is the number of weighted random patterns to be generated to insure that 0.9999 of all patterns in the test set appear in the sequence generated by the WCA. Because no closed form solution for the test length, *N*, exists, the authors adapted the equation from [19] for an upper bound on test length given detection probabilities. The adapted equation appears below. P_H is the probability of occurrence of the least likely or hardest-to-generate pattern. The number of hard-to-generate vectors, *k*, is calculated as the number of vectors with probability within one order of magnitude of the least likely vector. This is an overly pessimistic choice of *k*; however, the value of *k* has limited effect on *N* in most cases.

$$N = \frac{\log\left((1-M)/k\right)}{\log\left(1-P_H\right)}. \quad (1)$$

When the estimated test length becomes less than the user-set maximum or the number of weight sets reaches the user-set maximum, P^3 stops and outputs a list of current weight sets. WCARGO [11] then generates a WCA for the first weight set on the list.

3.3 WCARGO

The design algorithm for a single weight WCA, WCARGO, is a greedy method, using a heuristic search through a portion of the complete design space. The strategy of the algorithm is to generate a design, simulate it to determine the sites where the error in probability is too high, and choose new rules for those sites. Pseudocode for the WCARGO algorithm is given in Fig. 12.

```
BEGIN WCARGO
  Initialization
  SET Working Threshold = Original Threshold
                          + Rate × Steps
  LOOP UNTIL Original Threshold MET
    LOOP UNTIL Working Threshold MET
      Simulate Current Design
      IF worst_error > Working Threshold THEN
        Design_New_CA
    END LOOP
    DECREMENT Working Threshold BY Rate
  END LOOP
END WCARGO
```

Fig. 12. The top level algorithm for WCARGO.

For each site, WCARGO extracts the average probabilities from simulation data and calculates the error from desired probability. The error is the absolute value of the difference between the desired and average probabilities. Using this error term, WCARGO finds the sites with an error larger than the working threshold, then redesigns those sites. The redesign process is explained further in Section 3.3.1.

Two parameters, Rate and Steps, control how fast the design process attempts to approach the user-set parameter Original Threshold. During initialization, the Working Threshold is set to the Original Threshold + (Rate × Steps). Each time WCARGO finds a design with worst-case error less than the Working Threshold, it decrements Steps and recalculates the Working Threshold. This ensures that the design process does not try to make too many changes in one iteration. Steps is decremented until it is equal to zero.

3.3.1 Designing a New CA

Design_New_CA redesigns, or chooses new rules for, those cells with actual probabilities that differ from the desired probability by more than the working threshold. To choose a new rule for a cell, Design_New_CA estimates the probability of a cell with each of the available cell rules and chooses the type with the estimated probability closest to the design goal. Design_New_CA tries to redesign the same sites *k* times; $k = 5$ is the default setting. If those attempts fail, Design_New_CA redesigns the neighboring cells. Design_New_CA stores the path through the search space such that no WCA design is tried more than once.

3.3.2 Splitting WCA

WCARGO, as first presented in [11], could not generate designs for circuits with more than about 300 inputs, due to the computation time required to search such a large design space. To keep the design time within reason, WCARGO was modified to split the WCA into smaller pieces. By using null boundary conditions on the smaller WCA, WCARGO can design each piece separately and then combine the pieces to build a single large WCA. WCARGO divides the WCA into equal pieces as close to the user-set average value as possible. This study used a value of 100 for all circuits in order to cover small-to-medium circuits with one WCA and, at the same time, allow larger circuits to use reasonably sized WCA. Due to WCA's nonlinear nature, correlation between the pattern sequences of separate pieces of WCA is unlikely.

3.3.3 Increasing Desired Probability Accuracy

In early experiments, the WCA design algorithm did not always converge to within the user-set threshold of ±0.10 of the desired probability for all sites. These shortcomings were expected because of the results of the local experiments on the WCA. In fact, the causes of WCARGO's failures to produce the correct probability were the same as those of the local experiments. In the case of such errors, MWCARGO accepts the less-than-ideal design and continues the design process. Some designers, however, may require tighter tolerances on the error. Such designers need an alternative design function that will allow WCARGO to complete a design for a given set of probabilities within a given threshold.

Because most local probabilities are achievable, a local alternative—adding dummy cells and increasing the design iteration limit—is used to achieve probabilities within the error threshold. Consider any site *i* where the probability is not within the error threshold. If WCARGO cannot find a rule for site *i* that will provide the desired probability, then WCARGO places a "dummy" cell to the right or left of site *i*. The dummy cell does not have an enforced goal probability. The complete set of rules for adding dummy cells appears in [15].

Addition of dummy cells increases the hardware overhead, so the user can specify a maximum number of dummy cells. Applications that do not require such high accuracy need few or no dummy cells. Dummy cells were not needed to obtain any of the data in this study. Another possible, but unexplored, method to deal with convergence is to change the local ordering of cells in the CA.

4 Comparison of WCA to Existing Techniques

This section compares the WCA technique to other BIST techniques. Section 4.1 compares test lengths and weight-set sizes of WCA to those of the best-known test-per-scan method; Section 4.2 compares device counts, area overhead, and delay of WCA to those of other test-per-clock methods.

4.1 Test Length and Weight-Set Size Comparisons

To understand these comparisons, one must be familiar with the test circuits used in this study. The circuits are from the ISCAS 85 [20] and ISCAS 89 [21] benchmark sets. Circuit names beginning with "c" are from ISCAS 85 and those beginning with "s" are combinational versions from ISCAS 89 (each of the flip-flops is replaced with a primary output and a primary input). This study includes only circuits that contain random-pattern-resistant faults. Coverage data is given as a percentage of all detectable faults.

Table 5 shows the parameter settings for the runs reported in this section. The WCA run interval was set based on experiments reported in [11]. The test length and maximum weight sets are usually set by design guidelines for test time and overhead constraints. For this study, those parameters were set to conservative values.

TABLE 5
Setting of Parameters

Item	Setting
Test Length, TL_{user}	$10,000; 50,000; 100,000$
Weight Sets, WS_{user}	$3, 7$
Confidence Level	0.9999
User Set Threshold	0.10
Threshold Rate	0.01
Threshold Steps	25
Run Length for WCA	16
Run Length for LHCA	Average = 16
Max Depth	5
Iteration Limit	200
Max Added Sites	0
Average WCA size	100

A major motive for this work was to find a way for circuits to test themselves quickly with high fault coverage and low overhead. Because most previous weighted random test techniques are scan-based methods, the WCA results are compared to the test-application times required for the test-per-scan technique, Pattern Based Weight Calculation (PBWC) [22]. For PWBC methods, the test-application time is (*# scan cells* + 1) × (*# vectors*) + (*# scan cells*).

Tables 6 and 7 compare the PBWC method to the WMCARGO method. Two measurements are points of comparison—ratio of test-application times and difference in weight-set counts. The columns labeled Compare/TAT show the ratio of the PBWC test-application time to the MWCA test-application time, while the Compare/WS columns list the number MWCA sets minus the PBWC weight sets.

The test-application time ratios were not large for circuits with small numbers of inputs (less than 100); the ratios increased, however, for circuits with large numbers of inputs. MWCA, therefore, tested circuits with large numbers of inputs much faster than PBWC.

As for the weight-set results, the MWCA used fewer weight sets than the PBWC method in 14 out of 21 circuits studied. The PBWC method used fewer weight sets for only five circuits. Therefore, MWCA usually, but not always, used weight sets more economically than PBWC.

Note that in some cases—c2670 and s9234, for example—MWCARGO exceeded the user-set test length. This happened because MWCARGO continued the design process until it tested all faults or a WCA detected no more faults.

TABLE 6
COMPARISON OF RESULTS WITH SHORTEST TEST LENGTHS FOR MWCA AND PBWC

		PBWC			MWCA		Compare	
Circuit	PI	WS	V	TAT	WS	TAT	WS	TAT
s420	34	2	1825	63909	5	6621	3	9.65
s526	24	3	643	16099	3	3264	0	4.93
s641	54	3	593	32669	2	1504	-1	21.72
s713	54	3	576	31734	2	1504	-1	21.10
s820	23	5	1287	30911	5	5568	0	5.55
s832	23	4	1271	30527	4	4704	0	6.49
s838	66	5	893	59897	6	73568	1	0.81
s953	45	4	973	44803	4	4704	0	9.52
s1196	32	10	2002	66098	9	11776	-1	5.61
s1238	32	9	2460	81212	11	14976	2	5.42
s1423	91	5	620	57131	3	3072	-2	18.60
n1908	33	10	1253	42635	3	3679	-7	11.59
n2670	233	9	1360	318473	4	52288	-5	6.09
n3540	50	5	1298	66248	4	5024	-1	13.19
s5378	214	7	2270	488264	9	12480	2	39.12
n7552	207	12	2072	431183	10	13376	-2	32.24
s9234	247	11	3368	835511	8	27392	-3	30.50
s13207	700	7	3430	2405130	8	10240	1	234.88
s15850	611	6	3051	1867823	7	9344	1	199.90
s38417	1664	8	4972	8280044	7	45280	-1	182.86
s38584	1464	7	5381	7884629	6	88983	-1	88.61

TABLE 7
COMPARISON OF RESULTS WITH THE FEWEST WEIGHT SETS FOR MWCA AND PBWC

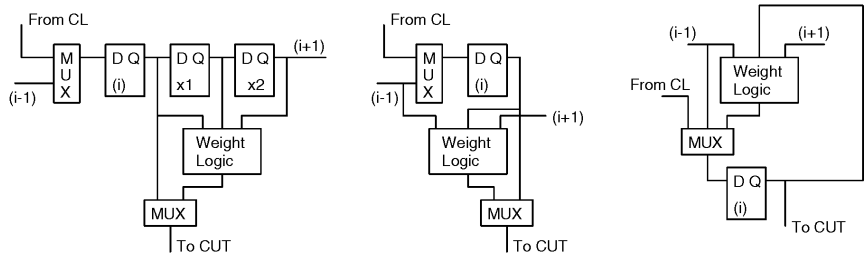
		PBWC			MWCA		Compare	
Circuit	PI	WS	V	TAT	WS	TAT	WS	TAT
s420	34	2	1825	63909	3	15221	1	4.20
s526	24	3	643	16099	2	2155	-1	7.47
s641	54	3	593	32669	1	2197	-2	14.87
s713	54	3	576	31734	1	2197	-2	14.44
s820	23	5	1287	30911	2	10257	-3	3.01
s832	23	4	1271	30527	2	10257	-2	2.98
s838	66	2	17484	1171494	5	58912	3	19.89
s953	45	2	3870	178065	2	7362	0	24.19
s1196	32	5	6424	212024	2	35569	-3	5.96
s1238	32	5	8559	282479	2	67592	-3	4.18
s1423	91	5	620	57131	1	14933	-4	3.83
n1908	33	10	1253	42635	2	6171	-8	6.91
n2670	233	3	12593	2946995	5	64098	2	45.98
n3540	50	5	1298	66248	3	6659	-2	9.95
s5378	214	4	11005	2366289	3	34766	-1	68.06
n7552	207	5	69495	14455167	3	76055	-2	190.06
s9234	247	4	160977	39922543	4	63577	0	627.94
s13207	700	4	15781	3961281	3	34766	-1	113.94
s15850	611	3	141181	35436681	2	38333	-1	924.44
s38417	1664	3	277109	69554609	5	136121	2	510.98
s38584	1464	2	107208	26909458	3	36649	1	734.25

In these cases, the user can increase the acceptable test length or settle for lower fault coverage.

The CPU time required to generate a design varies greatly with the size of the WCA designed and the values in the desired weight set. Design times reported by MWCARGO ranged from less than 12 minutes for s526 to 17 hours for s38417 on a SPARCstation 10 for the entire MWCA design process, including fault simulation and TPG. A worst-case analysis shows that the algorithm will iterate and redesign a given circuit up to $(N-1) * (\text{Steps})$ times, where N is the maximum number of redesigns per incremental change in the threshold (threshold rate) and Steps is the number of times the threshold is decremented.

4.2 Hardware Overhead and Delay Comparisons

To determine whether WCA improves on traditional test-per-clock methods, this section examines the overhead of test-per-clock schemes. The three test-per-clock weighted random pattern schemes to be compared—uncorrelated weighting, correlated weighting, and WCA—are given in Fig. 13; all are designed to give weight resolution to 0.125. The traditional techniques in Figs. 13a and 13b use weighting logic between the normal storage elements and the logic being tested; thus, these methods introduce delay between



a) Uncorrelated Weighting b) Correlated Weighting c) WCA

Fig. 13. Three types of weighted random BIST. (x1 and x2 denote extra storage elements and CL stands for Combinational Logic).

TABLE 8
COMPARISON OF DEVICE COUNT OVERHEAD

Weight Sets	Uncorrelated	Correlated	WCA
Single	48	18	36
Triple	69	39	48
2^M	$9(2^M)+M+57$	$9(2^M)+M+27$	$9(2^M)+M+26$
Unlimited	162	132	142

the storage element and the normal logic. The uncorrelated weighting approach is a worst-case method with respect to hardware overhead, since the weights are derived from disjoint sets of flip-flops, requiring the addition of two flip-flops for each input to the circuit. The correlated weighting approach is a best case in terms of hardware overhead. Because this approach correlates bits from the pseudorandom pattern generator, correlated weighting does not guarantee adequate fault coverage.

The added device counts per weighting cell for the three schemes in Fig. 13 are given in Table 8. The device counts assume fully complementary CMOS logic with dynamic combinational logic used where advantageous. For all multiple weight cases but the unlimited case, the weight selection lines are fully encoded. The device counts given are worst-case assuming no minimization and using device site counts for ROMs. For all but the 2^M case, the LHCA configuration is included in the design in addition to the specified number of weights for the WCA.

Overall, the WCA device counts are midway between those for the correlated weighted case and the uncorrelated weighted cases. Thus, the WCA hardware overhead appears competitive with that of other test-per-clock weighted approaches.

A test-per-clock circuit not shown or tabulated here is the GURT [10], which, with an added device overhead of approximately seven devices per cell, is certainly a winner in terms of device count, but has potentially high routing overhead. In addition, it appears to apply easily only to the single weight case.

In addition to the device counts given, actual cell designs, including flip-flops and weighting logic, were created using a commercial layout tool and simulator for the single-weight and five-weight cases for the three approaches. These cases were chosen because they represent the bounds on the number of weights required in Table 7 for the WCA for the random-pattern resistant ISCAS benchmark cases. The cells were fit into a framework based on a three-micron standard cell technology. Because of a tight power and ground pitch, this technology limited the design of complex cells.

The comparative area results for these approaches using the three-micron process are given in μm^2 per weight cell, including the normal flip-flop, in Table 9. Note that the WCA implements the LHCA function, in addition to the weight sets. The designs use dynamic weighting logic to reduce area where necessary. In all cases, a general design that applies to any combination of weight probabilities was used. Consequently, these are worst cases for area overhead. The areas given include all logic and interconnects, including all added interconnects, in the local routing channels related to the weighted test pattern generation approach.

TABLE 9
COMPARISON OF AREA OVERHEAD FOR WEIGHTED RANDOM CELLS IN μ^2

	Uncorrelated Weighting	Correlated Weighting	Weighted Cellular Automata (WCA)
W = 1	166,916	59,993	119,437
W = 5	255,528	128,315	171,495

When compared, the results for the three schemes are in the same order as the device count results. The WCA is at somewhat of a disadvantage because it is a general design that can implement any set of W weights, plus an LHCA. In an actual automated design situation, each cell could be customized to its actual weight needs. This would often result in a reduction in area. A similar, but less significant, reduction would occur for the other two schemes.

In summary, local device counts and areas are fairly high. Thus, using these techniques is appropriate only when the overhead can be 1) amortized over a substantial amount of logic or 2) hidden in areas such as the chip boundaries. The former involves selectively applying the

TABLE 10
COMPARISON OF NORMAL DELAYS AND MINIMUM TESTING CLOCK PERIODS IN NS

		Uncorrelated Weighting	Correlated Weighting	Weighted Cellular Automata (WCA)
W = 1	Normal	2.8	2.4	1.1
	Test	22.6 + DCL	21.5 + DCL	21.4
W = 5	Normal	3.1	2.6	1.1
	Test	28.1 + DCL	27.6 + DCL	46.7

weighted techniques only to the subset of all combinational logic circuits in an IC that are random pattern resistant. The latter routinely occurs when hiding the area required by boundary scan.

In addition to determining areas, delays for the three test-per-clock approaches were also evaluated by circuit simulation. The results appear in Table 10. The increase in the delay of the circuits during normal operation are given for one weight set and five weight sets. The increases in delay for the competing techniques ranges between two and three times the delay increases for WCA.

The minimum clock period during testing is also of importance if the speed advantage of test-per-clock circuits is to be fully realized. Due to the structure of the circuits, the minimum clock period for the Uncorrelated Weighting and Correlated Weighting cases is determined by the delay of the test logic plus the delay of the normal combinational logic, represented by DCL in Table 10. For the WCA, the test logic and normal logic are in parallel. For the W = 1 case, the WCA is clearly faster regardless of the delay of the normal logic. For the W = 5 case, the WCA testing clock period is shorter if the delay of the combinational logic is greater than 18.6 ns; otherwise, it is longer. An 18.6 ns delay corresponds to the delay of five to six four-input NAND gates connected in series implemented in three-micron technology. Finally, at-speed testing is not considered in these designs, which were optimized for area, not speed.

5 SUMMARY AND CONCLUSIONS

Clearly, weighted cellular automata (WCA), i.e., nonlinear hybrid cellular automata, using a wide range of linear and nonlinear functions, can perform weighted test-per-clock pattern generation for handling random pattern resistant circuits. The MWCARGO algorithm covers WCA design for single and multiple weight-sets. WCA and the MWCARGO algorithm form an effective solution that achieves high fault coverage and relatively small test times compared to test-per-scan schemes. Moreover, the solution requires overhead comparable to other test-per-clock weighted-random-pattern-generation schemes. Together, WCA and MWCARGO fill the need for a generator of weighted pseudorandom test patterns that is accurate, fast, and comparable in cost to competing schemes.

ACKNOWLEDGMENTS

The contribution of Chung Kai Chow in the form of layouts and simulations for determining cell area and delay is gratefully acknowledged. The comments of the reviewers of the paper are also very much appreciated. Last, but not least, the comments and editing of Colleen Pettit were extremely helpful in preparing this paper for publication.

REFERENCES

[1] A.P. Ströle and H.-J. Wunderlich, "TESTCHIP: A Chip for Weighted Random Pattern Generation, Evaluation, and Test Control," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, pp. 1,056-1,063, July 1991.

[2] H.D. Schnurmann, E. Lindbloom, and R.G. Carpenter, "The Weighted Random Test-Generator," *IEEE Trans. Computers*, vol. 24, no. 7, pp. 695-700, July 1975.

[3] H.-J. Wunderlich, "PROTEST: A Tool for Probabilistic Testability Analysis," *Design Automation Conf. Proc.*, pp. 204-211. IEEE / ACM, 1985.

[4] F. Siavoshi, "WTPGA: A Novel Weighted Test-Pattern Generation Approach for VLSI Built-In Self Test," *Proc. Int'l Test Conf.*, pp. 256-262. IEEE, 1988.

[5] S. Bou-Ghazale and P.N. Marinos, "Testing with Correlated Test Vectors," *Digest Int'l Symp. Fault-Tolerant Computing*, pp. 254-262. IEEE, 1992.

[6] J.A. Waicukauski, E. Lindboom, E.B. Eichelberger, and O.P. Forlenza, "A Method for Generating Weighted Random Test Patterns," *IBM J. Research & Development*, vol. 33, no. 2, pp. 149-161, Mar. 1989.

[7] J. Hartmann and G. Kemnitz, "How to Do Weighted Random Testing for BIST?" *Digest Int'l Conf. Computer-Aided Design*, pp. 568-571. IEEE, 1993.

[8] F. Brglez, C.S. Gloster, and G. Kedem, "Hardware-Based Weighted Random Pattern Generation for Boundary Scan," *Proc. Int'l Test Conf.*, pp. 264-274. IEEE, 1989.

[9] F. Muradali, V.K. Agarwal, and B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test," *Proc. Int'l Test Conf.*, pp. 660-669. IEEE, 1990.

[10] H.-J. Wunderlich, "Self Test Using Unequiprobable Random Patterns," *Digest Int'l Symp. Fault-Tolerant Computing*, pp. 258-263. IEEE, 1987.

[11] D.J. Neebel and C.R. Kime, "Inhomogeneous Cellular Automata for Weighted Random Pattern Generation," *Proc. Int'l Test Conf.*, pp. 1,013-1,022. IEEE, 1993.

[12] W. Pries, A. Thanailakis, and H.C. Card, "Group Properties of Cellular Automata and VLSI Applications," *IEEE Trans. Computers*, vol. 35, no. 12, pp. 1,013-1,024, Dec. 1986.

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

[14] P.D. Hortensius, "Parallel Computation of Non-Deterministic Algorithms in VLSI," PhD dissertation, Univ. of Manitoba, Manitoba, Canada, 1987.

[15] D.J. Neebel, "Cellular Automata for Weighted Random Pattern Generation," PhD dissertation, Univ. of Wisconsin–Madison, Dept. of Electrical and Computer Eng., May 1994.

[16] D.J. Neebel and C.R. Kime, "Multiple Weighted Cellular Automata," *Proc. VLSI Test Symp.*, pp. 81-86. IEEE, 1994.

[17] B.S. So and C.R. Kime, "A Fault Simulation Method: Parallel Pattern Critical Path Tracing," *J. Electronic Testing: Theory and Applications*, vol. 4, pp. 255-265, 1993.

[18] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers*, vol. 30, no. 3, pp. 215-222, Mar. 1981.

[19] J. Savir and P.H. Bardell, "On Random Pattern Test Length," *IEEE Trans. Computers*, vol. 33, no. 6, pp. 467-474, June 1984.
[20] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," *Digest Int'l Symp. Circuits and Systems*, pp. 151-158. IEEE, 1985.
[21] F. Brglez, D. Bryan, and K. Kozminski, "Accelerated ATPG and Fault Grading via Testability Analysis," *Digest Int'l Symp. Circuits and Systems*, pp. 1,929-1,934. IEEE, 1989.
[22] M. Bershteyn, "Calculation of Multiple Sets of Weights for Weighted Random Testing," *Proc. Int'l Test Conf.*, pp. 1,031-1,040. IEEE, 1993.

Danial J. Neebel is an assistant professor in the Integrated Science and Technology Program (ISAT) at James Madison University, Harrsionburg, Virginia, where he has been developing and teaching new courses in instrumentation for the growing ISAT program. His interests range from curriculum development with the use of educational technologies to design for testability and built-in self-test of digital systems. He is a member of the ACM, the IEEE Computer Society, and the IEEE Instrumentation and Measurement Society.

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 courses in computer engineering. His current research interests include built-in self-test, synthesis of random pattern testable circuits, and VLSI design. He has served as the general chairman of the 1979 International Symposium on Fault-Tolerant Computing, as an associate editor of the *IEEE Transactions on Computers* and of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. He has been a member of a number of program committees for IEEE-sponsored conferences. Dr. Kime is a fellow of the IEEE and a Golden Core member of the IEEE Computer Society.