[16] J. P. Hayes, "A NAND model for fault diagnosis in combinational networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1496–1506, Dec. 1971.
[17] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unrealiable components," in *John von Neumann Collected Works*, vol. V., A. H. Taub, Ed. New York: Pergamon, 1963, pp. 329–378.

Prathima Agrawal (S'74) was born in Mysore, India. She received the B.E. and the M.E. degrees, (with distinction), both in electrical communication engineering, from the Indian Institute of Science, Bangalore, India, in 1964 and 1967, respectively, and the M.S. degree in electrical engineering from the University of Rochester, Rochester, N. Y., in 1968. At present, she is working for her Ph.D. at the University of Southern California, Los Angeles.

During 1964–65, she was a technical assistant in the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, India. During 1967–68, she was a teaching assistant in electrical engineering at the University of Rochester, Rochester, N. Y. From 1968 to 1970 she worked on digital simulation and design diagnostics while she was employed on the Illiac IV Project, as research assistant and later as research programmer at the Department of Computer Science, University of Illinois, Urbana. She continued this work in 1970–71 when she was with the Automation Technology, Inc., Champaign, Ill. During 1973–74, she worked at the Computer Center of the Indian Institute of Technology, New Delhi, India, on design automation and diagnosis of digital systems. Since 1974 she has been pursuing research in the same area at the University of Southern California, Los Angeles.

Vishwani D. Agrawal (S'68–M'70) was born in Allahabad, India, on February 7, 1943. He received the B.Sc. degree from the University of Allahabad, Allahabad, India, in 1960, the B.E. degree in telecommunication engineering with honours from the University of Roorkee, Roorkee, India, in 1964, the M.E. degree in electrical communication engineering with distinction from the Indian Institute of Science, Bangalore, India, in 1966, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, in 1971.

During 1966–67 he taught in the Department of Electrical Engineering at the Indian Institute of Technology, New Delhi, India. From 1967 to 1970 he was with the University of Illinois, Urbana, where he worked on phased arrays at the Antenna Laboratory and taught at the Department of Electrical Engineering. During 1970–71 he was with the Automation Technology, Inc., Champaign, Ill., working on the Illiac IV Project. During 1971–72, as a Senior Scientist at E. G. & G., Inc., Albuquerque, N. Mex., he worked on the design and evaluation of wide-band antennas. Since 1972, he has been an Assistant Professor at the School of Radar Studies, Indian Institute of Technology, New Delhi, India, where he teaches and conducts research on phased array radars.

The Weighted Random Test-Pattern Generator

H. DANIEL SCHNURMANN, MEMBER, IEEE, ERIC LINDBLOOM, AND ROBERT G. CARPENTER

***Abstract*—A heuristic method for generating large-scale integration (LSI) test patterns is described. In particular, this paper presents a technique for generating statistically random sequences to test complex logic circuits. The algorithms used to obtain a set of tests by means of weighted logic signal variations are included. Several techniques for assigning these weights and for varying them are discussed on the basis of the primary algorithm. Also described is a means of obtaining a minimal number of test patterns. This approach has proved successful in obtaining fault-detecting patterns.**

***Index Terms*—Fault-detecting patterns, heuristic algorithm, large-scale integration, testing, testing algorithms, test-pattern generator, weighted random patterns.**

Manuscript received December 3, 1973; revised January 2, 1975.
The authors are with the System Products Division, IBM Corporation, East Fishkill, N.Y. 12533.

INTRODUCTION

THE FIELD of test-pattern generation has grown in importance with the advent of large-scale integration (LSI). Numerous algorithms have been developed over the years with the object of arriving at a set of logic patterns that detect stuck faults in a circuit.

At present, most test-pattern generators (TPG's) are deterministic in nature and revolve around the concept of sensitizing a path between inputs and outputs. These TPG's can be divided into two categories: fault-oriented and path-oriented. Their mode of operation has been described in the literature [1]–[4]. However, deterministic TPG's are often unable to cope with the increasing com-

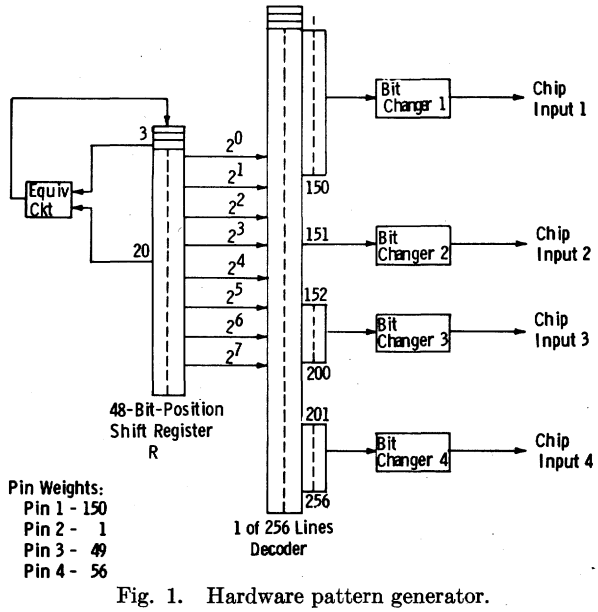


Fig. 1. Hardware pattern generator.

plexity of LSI packages and their inherently low pin-to-circuit ratios.

The method discussed in this paper represents a significant departure from these techniques. In this method, a set of randomly generated patterns is applied to the primary inputs (PI's) of the device under test. Since not all the PI's have the same functional importance, some being more critical than others, a method was devised to exercise certain PI's more often than others.

This approach has undergone a number of evolutions throughout its development. In its initial phase, the efficacy of purely random patterns was measured. At later stages, some intelligence was introduced by assigning *weights* to the PI's in proportion to their relative importance; hence its name, weighted random test-pattern generator. Finally, a more sophisticated approach, the dynamic adaptive technique, was implemented by analyzing the rate of change of *switching activity* inside the logic as the result of exercising a PI.

This last method has achieved high testability coverage where deterministic TPG's have failed. However, because any random approach usually creates a large number of test patterns, a reduction technique was required. Such a method was added, and it succeeded in compressing the patterns generated to a more manageable size without loss of test coverage.

THE TEST-PATTERN GENERATOR

Basically, a software pattern generator was implemented by modeling a hardware tester configuration. The method for generating test patterns is best seen via a description of the hardware.

The random pattern generator is built around a 48-bit shift register R (Fig. 1) [5]. Two bit positions of R are tapped to provide the inputs of a Boolean equivalence function circuit E. The output of E is returned to bit position 1 of R to provide a feedback path. A known combination of 0's and 1's is initially fed to R. Whenever the bits advance from one position of R to the next, bit 1 will be either 0 or 1, depending on the values taken by, say, bits 3 and 20 of R. N bit positions are selected from R and are fed into a 2^n-bit binary-to-decimal decoder. For every shift of R, one of the 2^n decode bits is selected as specified by the n-bit number (8 in the example). Over a large number of shifts, each of the decoder positions will be exercised approximately the same number of times [6].

A bit changer, such as a binary trigger, is connected to each PI of the device under test. Each time a new signal is applied to the bit changer, the logic level of the corresponding PI also changes. Thus, the patterns generated in this fashion contain only single input changes. A *weight* is then assigned to each PI in proportion to its importance. This factor was initially determined by measuring the amount of switching activity produced inside the logic as the result of exercising a PI. Thus, if the activity was high, the weight of that pin should also have been high. Conversely, if a pin did not produce a measurable effect inside the logic, a low weight was assigned to it. Clearly, since activity cannot be measured inside an LSI device, a simulation of the pattern generator and of the logic was made to obtain a set of appropriate weights for each PI.

The pattern generator is weighted by the manner in which the decoder outputs are assigned to the bit changers. The PI weights are assigned for each device prior to test. If 100 decoder outputs are connected via an OR function to drive bit changer 1, PI_1 will receive 100 times more changes than a PI driven from a bit changer connected to only one decoder output. The particular configuration in Fig. 1 shows a pin weight distribution of 150 for PI_1, 1 for PI_2, 49 for PI_3, and 56 for PI_4. This could be a typical pin weight distribution where PI_1 was a clock input, PI_2 a reset, and PI_3 and PI_4 data inputs. 256 bits, handled by

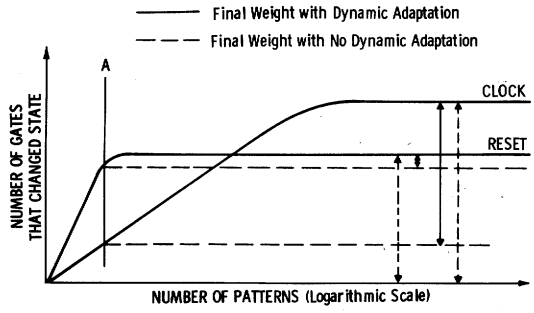


Fig. 2. Clock/reset activity.

8 shift register positions, provided an adequate weight distribution.

RANDOM PATTERNS

A purely random technique assigns the same numerical weight to every PI of the chip to be tested. This is achieved by distributing the outputs of the decoder among the PI's in such a manner that each *bunch* contains the same quantity of lines. Thus, each PI is exercised approximately the same number of times, and the resulting patterns have the characteristic of randomness. This approach is most suited for chips that contain only combinatorial logic, because they do not require sequences of patterns. Often, a set of equally weighted or random patterns is all that is required. However, when sequential logic is present—particularly when functional packages such as counters and shift registers are included in the chip—the equally-weighted-patterns method has proved inadequate.

WEIGHTED ADAPTIVE PATTERNS

Because of the inability of equally weighted patterns to test sequential logic, regardless of the number of patterns generated, the next step was to assign weights to the PI's in proportion to their relative importance. Off-line good-machine simulation is performed by software using a set of random patterns as input. Recalling that each pattern from the generator activates only one PI at a time, it is possible to count how many gates inside the chip change for the first time from a logic 1 to 0, and vice versa, as the result of switching one of the PI's. The switching *activity count* is then accumulated over the complete set of patterns. By comparing the activity created by all PI's, one can determine not only the relative importance of each PI, but also the most effective way to bunch the decoder outputs for the next pass. Through the use of this weight vector to generate a new pattern set of equal length, iteration continues until the activity counter no longer increases. No significant activity has been recorded for logic configurations of up to 500 circuits after the application of 1000 patterns.

Several methods of measuring activity were evaluated, e.g., observing the state of circuit inputs by themselves or in combination with their output state. Results indicated that there was no appreciable difference in fault coverage between the methods measured, although there was a definite improvement for all of them over the equally weighted patterns. However, the weighted adaptive patterns did not achieve satisfactory results. This was due to two problems: namely, the improper handling of reset lines and the difficulty created by not having a set of weights *in step* with the simulation procedure.

The problem with the reset lines is that they characteristically directly switch a large number of gates. After applying a set of patterns, the reset lines consistently show a high activity count. In terms of the final fault-detection patterns, overactive resets are undesirable. The second problem, that of having the adaptive weights *one step behind* the simulation, is that the weights obtained at the end of the simulation are really a history of what happened in that past. It is a weight that would probably be more efficient in changing the state of the same gates that were already switched. It is not, however, a predictive vector capable of assigning the best possible weights to the remaining gates not yet affected.

These two drawbacks were instrumental in the development of a new approach to the computation of adaptive weights: the dynamic adaptive weight generator.

DYNAMIC ADAPTIVE PATTERNS

The dynamic adaptive technique [7] still measures activity in terms of gate switching states. This technique specifically searches for *rates of change* of activity. This can be illustrated by considering two inputs: a clock and a reset line. When the random pattern simulation is performed, it is the reset that initially shows the steepest increment in activity. The activity count of the clock, on the other hand, rises at a slower rate, and its final value is reached at a much later stage than that of the reset. This is shown in Fig. 2.

The faster initial slope of the reset is due to the switching of a large number of gates that are directly driven by that line. By the time the first few dozen random patterns have been exercised, most of those gates will have already changed state. For the next few hundred patterns, few additional new gates will be switched by it. Consequently, the activity of the reset tapers off. In previous techniques, the final weight associated with the reset was the high value of the activity count achieved during its initial per-

TABLE I
ILLUSTRATION OF THE SET OF TEST PATTERNS BEFORE AND AFTER APPLYING THE REDUCTION ALGORITHM

	Pattern Number	Original First Detect	Final Detect
2	1	*F*1, *F*2, *F*3, *F*4	
	2	*F*5, *F*6	
	3		
	4	*F*7	*F*7
	⋮		
	12	*F*8	
	⋮		
5	145		*F*3, *F*4, *F*8
	⋮		
	150	*F*9	*F*9
	⋮		
13	987		*F*1, *F*2, *F*5, *F*6
	⋮		
	1000	*F*10	*F*10

formance. If a counter with a required count of, say, 256 is found in the logic, it is obvious that, in the weighted simulation, the maximum of 256 would never be reached because of the high activity count of the reset. Every few patterns after the clock is stepped, in all likelihood, a reset signal will reset the counter back to 0.

The dynamic adaptive scheme locates the placement of line A of Fig. 2 where the reset line is about to leave the *knee* of the curve. The search for the knee is achieved in the following manner. A suitable increment Δ is selected. The reset is monitored by comparing its activity at regular pattern intervals (i.e., every ten patterns). Initially, the increment of the activity count of the reset exceeds the value Δ. At some point, however, the increment of the activity of the reset tapers off, and the increments become less than Δ. The knee of the curve is that point where the transition from less than Δ to greater than Δ occurs. At the pattern where this happens, the simulation is interrupted, the activity count (AC) of all the PI's is reset back to 0, and the run is allowed to continue from the pattern where the interruption occurred. At the end, the clock weight, which was only slightly affected by the resetting to 0 of AC, will have shown a considerable increase in its activity, whereas the reset can only reach a steady-state value slightly more than its value at the time of the intersection with line A. The final weight of the clock will therefore be high, and that of the reset, low. The handling of clocks and resets does not presuppose prior identification of their functional characteristics. Any PI displaying the behavior shown in Fig. 2 will be classified as such.

PATTERN REDUCTION TECHNIQUE

Testers that include the TPG hardware of the type described earlier are the most efficient source for adaptive patterns, because patterns are created and applied at machine cycle-time speed as opposed to off-line, software generation. The number of patterns required for a particular chip by this method is no obstacle since the only data that need to be supplied to the tester for each new LSI package are a known start pattern and the pin weights.

If, however, we are limited to conventional testers, which require external preparation of dc patterns, the number of patterns can become the throughput gating factor. But even here the expanded coverage of the adaptive generator can be utilized. A software version of the generator is run with a reduction algorithm that extracts only those patterns or pattern sequences that detect new faults.

The reduction algorithm works as follows. It performs full fault simulation and records the pattern number where each fault is first detected. Then it takes the *most difficult* fault (last fault detected) and determines the sequence required to detect this fault. All other *easier* faults also detected in this sequence may now be eliminated from further consideration. The next most difficult fault not detected in the first sequence is now processed in the same way. This procedure continues until all the faults which are detectable in the initial randomly generated pattern set have sequences which make them detectable as either a difficult fault or an easy fault.

The following example illustrates the mechanism of this algorithm. Assume that 1000 patterns detect all 10 stuck faults of a given logic network in the manner shown in the first two columns of Table I.

The reduction technique calls for detecting the most difficult fault first, namely, *F*10. Pattern 1000 is first simulated by itself. If the fault is not detected, then patterns 999 through 1000 are simulated. This process of incorporating an additional new pattern to the previous set is continued until the detecting sequence is found. The entire circuit is assumed to be in an unknown state whenever a new sequence is tried out. Suppose that, upon resimulating with these 13 patterns, faults *F*1, *F*2, *F*5,

TABLE II
COMPARISON OF THE TESTABILITY COVERAGE ACHIEVED BY THREE DIFFERENT TPG'S

Circuit	Number of Blocks	Number of I/O's	Number of Logic Levels between PI's and PO's	Number of Feedback Cuts	Deterministic TPG's		Dynamic Weighted Adaptive TPG
					Path-Oriented	Fault-Oriented	
A	60	21	17	16	93.6[a] 96[b] 0:11[c]	92.7 98 0:08	94.8 134 0:37
B	155	21	16	90	76.7 493 1:12	13.1 21 1:47	82.6 1116 2:37
C	236	35	25	62	38.8 185 5:37	25.8 45 2:28	86.5 1131 4:11
D	317	46	17	124	66.7 1087 4:38	81.0 834 4:19	88.4 3092 7:25
E	519	38	37	45	81.5 463 10:02	91.6 433 6:32	92.0 796 7:20

[a] Testability coverage in percent.
[b] Number of patterns after applying pattern reduction.
[c] CPU time on System 360/Model 85 in minutes.

and *F*6 are also detected. These are now removed from the fault list, and the process is repeated for the next most difficult fault, namely, *F*9. This fault is detected by the sequence 145 through 150. Once again, additional faults, *F*3, *F*4, and *F*8, are also detected. Finally, *F*7 is detected by applying patterns 3 and 4. Thus, 20 out of the original 1000 patterns suffice to detect all failures. These 20 patterns constitute the reduced set to be applied to the logic network.

Techniques employing a weighted search are used to reduce the amount of resimulation to find a detection sequence.

The initial number of weighted random patterns generated ranged from 1000 to 5000 for logic configurations of up to 100 circuits and from 5000 to 15 000 for those of up to 500 circuits. With the reduction technique, these pattern sets were reduced by an average factor of 18. More significantly, the number of patterns after reduction is competitive with well-known deterministic pattern-generation methods. This is true even after the patterns created by deterministic TPG's are compressed by use of the same technique.

RESULTS

The weighted random TPG has been applied to a variety of LSI development technologies.

Generally, the original random generator achieved satisfactory failure coverage for simple LSI packages with circuit-to-pin ratios not exceeding three to one, and with a limited number of logic levels between inputs and outputs. In more complex technologies, namely those with embedded sequential functional elements connected to one another, the weighted random TPG performed well.

Table II shows five packages of varying degrees of complexity. The topological description is given by the number of blocks and the combined total of PI's and primary outputs (PO's). In addition, the complexity of each package is shown by the maximum number of logic levels that separate the PI's from the PO's, and by the number of feedback cuts necessary to convert all the sequential elements into combinatorial ones.

The results of three TPG's are presented, the first two in the deterministic, path-sensitizing category, and the last, the weighted random TPG. The number of test patterns in the dynamic adaptive weight generator is the total arrived at after applying the reduction technique to the original set. Since its initial pattern count was high (5000 to 12 000 patterns), considerable time was required to arrive at the final low count. This reduction technique was also applied to the other deterministic TPG's.

Although no clear-cut conclusions are to be drawn from such a small sample, this table is useful to show some of the results one may achieve by using alternative methods. The encouraging aspect of this table lies in the high testability figures of the weighted random TPG, even in the presence of embedded counters and shift registers of considerable complexity. Quite often, usually in simpler configurations, the only untested faults were those created by redundancies in the logic or by those unexhibitable by present simulation techniques. Examples of the latter are the output of a single shot or clock chopper and the inability of the fault simulator to set certain gates to a known binary state.

CONCLUSION

The weighted random TPG represents a significant departure from classical methods of generating test sequences for complex LSI packages. The virtue of this technique is its simplicity and the fact that test-generation time is virtually independent of the number of PI's or gates in the logic package to be tested. This technique can be used both in a conventional tester and in a tester where the weighted random TPG is implemented in hardware. In

the first case, an elimination technique permits the reduction of the original set of patterns to a more manageable size. In the second, since patterns are created at machine cycle-time speed, their large number is no deterrent.

Considerable success has been achieved with the weighted-random-generator approach in arriving at high testability figures for complex LSI devices.

ACKNOWLEDGMENT

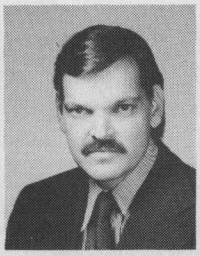
The authors would like to thank M. T. McMahon, Jr., for his contribution throughout the development of these techniques, and Dr. G. R. Putzolu for his valuable suggestions. Highly efficient versions of both the weighted generator and the reduction algorithm have been programmed by T. M. Storey.

REFERENCES

[1] W. G. Bouricius *et al.*, "Algorithms for detection of faults in logic circuits," *IEEE Trans. Comput.*, vol. C-20, pp. 1258–1264, Nov. 1971.
[2] J. P. Roth, "Diagnosis of automata failure: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, no. 4, pp. 278–291, 1966.
[3] G. R. Putzolu and J. P. Roth, "An algorithm and a program for generation of test patterns for sequential circuits," in *Proc. 3rd Hawaii Int. Conf. Syst. Sci.*, 1970, pp. 64–67.
[4] D. W. Bray, "The ATVG program: A test vector generator for sequential networks," in *Conf. Rec., 2nd Workshop Fault Detection and Diagnosis*, Lehigh Univ., Bethlehem, Pa., 1971, pp. 42–54.
[5] R. G. Carpenter, E. Lindbloom, and T. M. McMahon, Jr., "Statistical logic test system having a weighted random test pattern generator," U. S. Patent 3 719 885, Mar. 6, 1973.
[6] S. W. Golomb, *Shift Register Sequences*. San Francisco: Holden-Day, 1967.
[7] H. D. Schnurmann, "A computer program for weighted test pattern generation in Monte Carlo testing of integrated circuits," *IBM Tech. Disclosure Bull.*, vol. 16, pp. 417–423, July 1973.

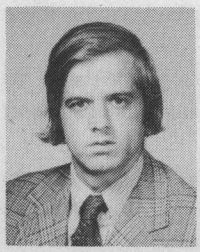
H. Daniel Schnurmann (S'61–M'62) was born in Strasbourg, France, on March 18, 1939. He received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1961, and the M.S.E.E. degree from the Polytechnic Institute of Brooklyn, Brooklyn, N.Y., in 1964.

From 1961 to 1963 he was with the Western Union Telegraph Co., New York, N.Y. Since 1963 he has been with the IBM Corporation, first with the Components Division in Poughkeepsie and East Fishkill, N.Y., where he worked in developing some of the earlier large-scale integration (LSI) devices. In 1965 he was transferred to the Thomas J. Watson Research Center, Yorktown Heights, N.Y., where he was involved in the development of a computer-controlled system to measure the performance of integrated circuits. In 1970 he returned to East Fishkill, where he has been engaged in a number of tasks related to testing of LSI circuits, mainly in the development of a variety of software systems related to test-pattern generation, simulation, and the release of test data to testers.



Eric Lindbloom was born in Detroit, Mich., on December 5, 1934. He received the B.A. degree in philosophy from the University of Michigan, Ann Arbor, in 1957, and did graduate study in philosophy at the University of California, Berkeley.

He returned to the University of Michigan in 1959, serving as a Teaching Fellow and then as an Instructor in the Department of Philosophy. In 1963 he joined the Logic Automation Group of the IBM Systems Development Division, Poughkeepsie, N.Y., to work on automated design. Since 1966 he has been with the IBM Components/System Products Division, East Fishkill, N.Y., working in the areas of three-valued logic simulation and stuck-fault test-pattern generation. His current interest is fault diagnosis of large-scale integrated (LSI) circuits. Since 1973 he has been a manager in Test System Development.



Robert G. Carpenter was born in Pittsfield, Mass., on February 23, 1942. He received the B.S. and M.S. degrees in electrical engineering from the University of Massachusetts, Amherst, in 1964 and 1966, respectively.

Since 1966 he has been with the Components/System Products Division of IBM, East Fishkill, N.Y. From 1966 to 1968 he worked in the area of placement and wiring of large-scale integration (LSI) circuits and has since been involved in various phases of LSI logic-circuit testing. From 1969 to 1970 he developed a system for semiautomated test generation using the IBM 1130/2250 Display System. Since then he has worked on automated-test-generation systems, with emphasis on parametric and other technology-dependent testing problems. He is currently involved in the development of systems to provide test data to manufacturing automatically.