PEST: A TOOL FOR IMPLEMENTING PSEUDO-EXHAUSTIVE SELF-TEST

Eleanor Wu

Eleanor Wu *is a member of technical staff in the Test and Diagnostics Department of AT&T Bell Laboratories at Princeton, New Jersey. Currently, she is working on cost-effective built-in self-test schemes for digital control logic. Ms. Wu received a B.S. from the University of Oregon, Eugene, and an M.S. from the University of Illinois, Champaign-Urbana, both in mathematics. After receiving a Ph.D. in mathematical logic from the Massachusetts Institute of Technology, Cambridge, Ms. Wu joined AT&T in 1985.*

Multichip modules, wafer-scale integration, and three-dimensional connections are the latest solutions to building faster computers. These new technologies will shorten electron travel distance, but they will compound an existing problem: testing. Without a systematic approach, testing even a single very large-scale integrated (VLSI) chip could be impossible. This paper describes a superior built-in self-test (BIST) design methodology that solves the VLSI testing problem. Pseudo-exhaustive self-test (PEST) is a design methodology that provides effective, 100-percent fault coverage without the need for fault simulation or deterministic test generation. We used PEST, an existing software package based on the algorithms discussed here, to implement the PEST function in three VLSI designs at AT&T.

Introduction

VLSI technology has allowed us to integrate extremely complex functions onto a single chip. It has been predicted that in the next decade the number of transistors that can be put on a chip will increase from about 1 million to close to 100 million. This increase will give chip designers more freedom to implement complex functions, but it will also make the structural verification of these chips more difficult. This paper presents a test methodology for VLSI design that solves the manufacturing test problem and also simplifies the system's testing and diagnostic requirements.

During the manufacturing process, various types of defects can appear in a chip. The defective part of the chip behaves as though one or more of the logic gates are "stuck at" one value, either a one or a zero. Traditionally, designers attempt to generate a set of test vectors, or inputs to the chip, that will exercise all possible stuck-at faults. After the test vectors are generated, fault simulation is implemented to determine the actual coverage of all stuck-at faults using that particular set of vectors.

Panel 1. Acronyms and Terms

ATE	automatic test equipment
ASIC	application-specific integrated circuit
BIST	built-in self-test
BIT	built-in test
CAD	computer-aided design
CMOS	complementary metal-oxide semiconductor
DMA	direct memory access
FIFO	first-in, first-out
IC	integrated circuit
ISCAS	10 benchmark circuits published in the International Symposium on Circuits and Systems in 1985
LFSR	linear-feedback shift register
MISR	multiple-input shift register
NIC	network-interface controller
NP	nondeterministic polynomial
PEST	pseudo-exhaustive self-test
RAM	random access memory
SWAP logic	swaps data
VLSI	very large-scale integration

Vector generation and fault simulation have become the bottleneck in manufacturing VLSI chips. Many months are added to a product cycle to obtain good fault coverage. Because of the complexity of the chip, in many cases we cannot obtain 100-percent fault coverage without special "design for test" considerations. With this in mind, designers must strive to guarantee the testability of their designs.

One solution to the digital VLSI testing problem is Built-In Self Test (BIST), a design philosophy that advocates the addition of a customized test function into each chip. (See Panel 1 for definitions of terms.) This paper discusses one such customized test function, PEST, which automatically generates the test vectors needed to guarantee 100-percent fault detection for each chip. With this test function in place, a functionally verified chip design can be manufactured without the delays caused by test-vector generation and fault-coverage verification.

The PEST concept was first described in 1981 by McCluskey,[1] who stressed the need for a test strategy that could be used in manufacturing, system, and field tests. Today, within AT&T, we have successfully incorporated the PEST function into three application-specific integrated circuits (ASICs). To automate the implementation of PEST, computer-aided test software was developed at the AT&T Engineering Research Center in Princeton, New Jersey.

The remainder of this paper presents:

- An overview of the new PEST scheme
- A circuit-segmentation algorithm
- Test generation using canonical vectors
- Results of implementing the PEST scheme in the network-interface controller (NIC) chip.

Overview of the PEST Scheme

The ideal test for a chip would incorporate an exhaustive set of test vectors, that is, the test would include vectors that would exercise every possible path in the chip. Therefore, if a VLSI circuit passes an exhaustive test, we would know it works correctly. An exhaustive test for a VLSI chip with 100 inputs and 500 internal memory elements would have on the order of 2^{600} or 10^{18} vectors. It is impossible to conduct an exhaustive test on a VLSI chip that includes such an enormous number of vectors within a realistic time frame. PEST, however, can conduct the *equivalent*[2] of an exhaustive test in an affordable testing time frame with fewer than 1 million test vectors.

From a logic designer's point of view, a VLSI chip is composed of combinational logic and flip-flops. Combinational logic is made up of primitive elements, such as AND gates, OR gates, and inverters. Flip-flops are the primitive single-bit memory elements. A collection of the primitive combinational elements that combine to produce one output is known as a combinational logic cone. The combinational logic of a chip is a collection of combinational logic cones.

PEST tests the combinational logic and flip-flops of a circuit independently. PEST conducts an exhaustive test on each logic cone, thereby guaranteeing 100-percent stuck-at fault detection. The flip-flops are tested with a combination of an initialization test and PEST test vectors.

Logic cones that occur naturally on the chip may be too large to test exhaustively. For example, for an n input logic cone, 2^n test vectors are needed to conduct

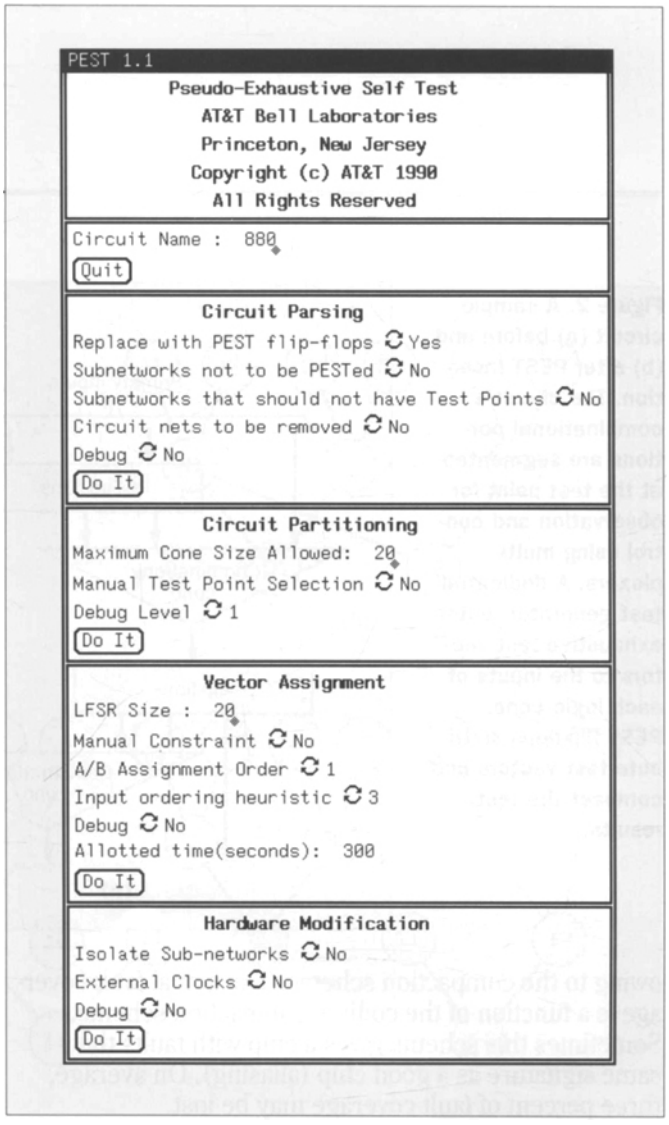


Figure 1. The PEST user interface menu. During circuit parsing, the logic design of the input circuit is read. The segmentation algorithm then partitions the circuit and generates additional observable and controllable test points (to reduce the size of logic cones), if needed. During vector assignment, the assignment algorithm designates test vectors to inputs of the logic cones. PEST then modifies the logic design to insert the PEST function generated during circuit partitioning and vector assignment. The output of the PEST software is a modified chip.

an exhaustive test. If n is 30, then 1 billion test vectors will be needed. To control the length of the test, large logic cones must be divided into smaller ones. This is accomplished by adding observable and controllable test points at strategic locations in the circuits. In PEST, a segmentation algorithm, discussed later, is used to select the test points.

The PEST software is a computer-aided design (CAD) tool that analyzes the circuitry of a chip and incorporates custom testing hardware into it. Figure 1 shows the PEST user interface menu.

The output of the PEST software is a modified chip. The modifications appear in four ways. First, PEST adds a test-vector generator to provide the exhaustive test vectors to the inputs of each logic cone. Second, the observable and controllable test points are implemented using multiplexers. Third, all flip-flops in the chip are modified to distribute the test vectors and to compact the test results. During test, the test responses are coded (compacted) into a binary sequence of a fixed length, called a signature. The coding is accomplished in hardware using the flip-flops of the chip.

Finally, a self-test controller is added. Some BIST schemes test parts of the chip in sequence. Our PEST scheme tests all logic cones simultaneously. This eliminates test scheduling, simplifies self-test control, and shortens test time. Figure 2 shows a chip before and after PEST insertion. The self-test controller is not depicted here.

The PEST function is activated when the control lines to the PEST flip-flop are set to test-result-compaction mode and the control line to the test generator is set to go. At the end of the testing session, the contents of the flip-flops, which give the final signature of the chip, are scanned out and compared with the signature of a good chip to determine the condition of the chip under test.

In reality, testing always involves tradeoffs. Ideally, we balance the various tradeoffs in hardware overhead, testing time, and test data compaction schemes to maximize the overall fault coverage on the chip. Although the PEST implementation assures 100-percent fault detection, the actual fault coverage may not be that high,

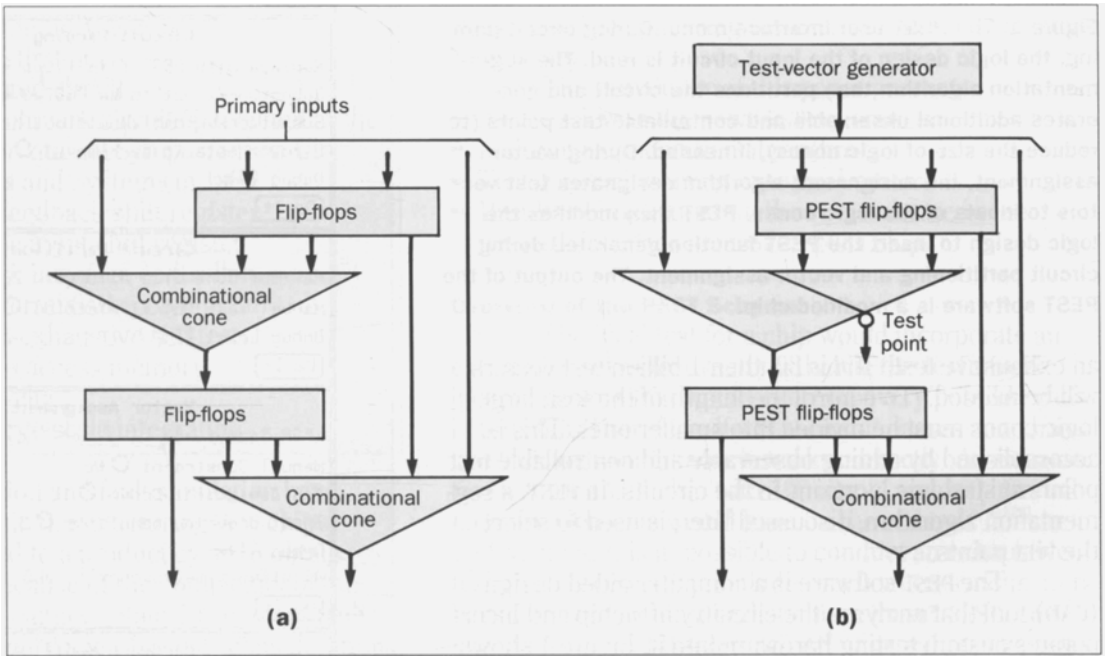


Figure 2. A sample circuit (a) before and (b) after PEST insertion. The circuit's combinational portions are segmented at the test point for observation and control using multiplexers. A dedicated test generator routes exhaustive test vectors to the inputs of each logic cone. PEST flip-flops distribute test vectors and compact the test results.

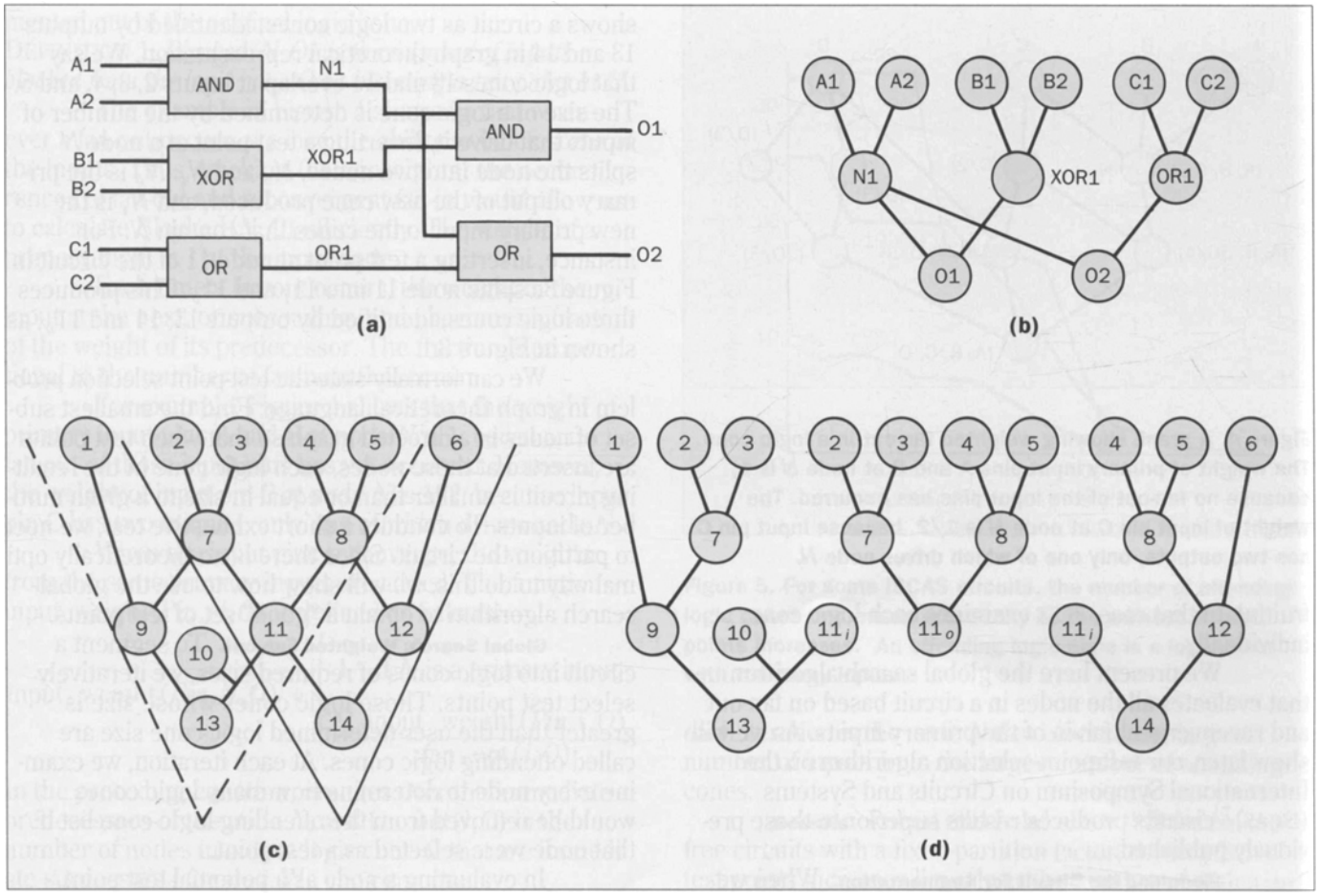
owing to the compaction scheme. The actual fault coverage is a function of the coding/compaction scheme. Sometimes this scheme gives a chip with faults the same signature as a good chip (aliasing). On average, three percent of fault coverage may be lost.

With nearly 100-percent fault coverage, PEST does impose a hardware-overhead penalty, but it eliminates the need for complex automatic test equipment (ATE), exterior test generation, and fault simulation, while it dramatically improves system-level testing.

Circuit Segmentation—Test Point Selection

To make PEST practical for large combinational circuits, we must limit the number of inputs driving each logic cone by partitioning the circuit,[3] i.e., by adding observable and controllable test points. A complication involved in partitioning logic cones occurs when certain inputs affect more than one output—the circuit fans out. Several logic cones may therefore share inputs and overlap in complex ways. For fan-out-free circuits, it is possible to give a simple formula for calculating the smallest number of test points needed for each circuit.[4] But, for circuits with fan-outs, determining the smallest number of test points for the circuit becomes NP-complete,[5] that is, there is no computationally practical way to obtain the smallest set of test points.

Since an optimal solution is not possible, a number of approximation algorithms or solutions have been proposed. Such algorithms are categorized as either partition or segmentation algorithms. Partition algorithms produce a circuit that is a collection of disjoint subcircuits. Segmentation algorithms produce a circuit consisting of overlapping subcircuits. Although partitioning a circuit results in a shorter test, segmentation is less expensive to implement and provides higher fault coverage.[6] In the interest of minimizing implementation cost,



our new algorithm is a segmentation algorithm.

Our segmentation algorithm analyzes both the global and local characteristics of the circuit. A global evaluation of all the logic cones allows us to pinpoint overlapping cones, which then can be used in selecting test points. Because a global search is very computationally intensive, no other segmentation algorithms have attempted to use global information. On some circuits, a global search for overlapping logic cones may not be

Figure 3. (a) A sample combinational circuit represented as (b) a graph. Another graph represents a slightly more complex circuit with (c) two logical cones identified by outputs 13 and 14. Inserting a test point at node 11 of the circuit creates (d) three logical cones identified by outputs 13, 14 and 11_o.

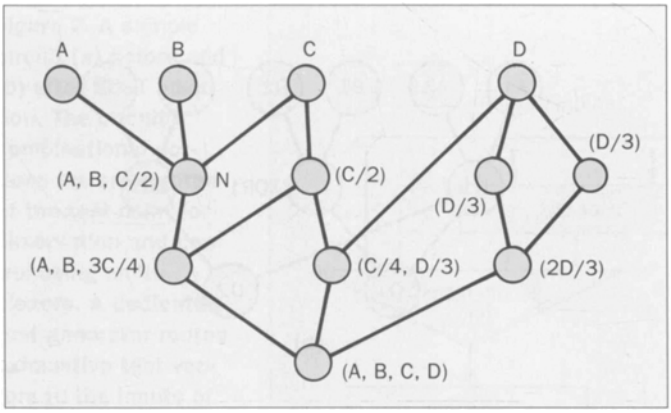


Figure 4. A graph showing weighted fan-out in a logic cone. The weight of primary input pins A and B at node *N* is 1, because no fan-out of the input pins has occurred. The weight of input pin C at node *N* is 1/2, because input pin C has two outputs, only one of which drives node *N*.

fruitful. In that case, PEST examines each logic cone individually.

We present here the global search algorithm that evaluates all the nodes in a circuit based on fan-out and reconvergent fan-in of the primary inputs. As we will show later, our test-point-selection algorithm on the International Symposium on Circuits and Systems (ISCAS)[7] circuits produces results superior to those previously published.

Modelling the Circuit for Segmentation. When we segment a circuit, we focus on the structure rather than the functionality of the chip. We are not interested in the type of each logic element (AND, OR, or inverter gate). Each logic element is simply a node in a graph. The connections between the logic elements are important; they are arcs connecting the nodes in the graph. Graph theoretical modeling simplifies the presentation of the algorithm by emphasizing the structual aspect of the circuit. Figure 3a shows a combinational circuit that has been translated into a graph (Figure 3b). Figure 3c shows a circuit as two logic cones, identified by outputs 13 and 14 in graph theoretical representation. We say that logic cones 13 and 14 overlap at inputs 2, 3, 4, and 5. The size of a logic cone is determined by the number of inputs that drive it. Inserting a test point at a node N splits the node into two nodes, N_o and N_i. N_o is the primary output of the new cone produced, and N_i is the new primary input to the cones that contain N. For instance, inserting a test point at node 11 of the circuit in Figure 3c splits node 11 into 11_i and 11_o. This produces three logic cones, identified by outputs 13, 14 and 11_o, as shown in Figure 3d.

We can formally state the test-point selection problem in graph theoretical language: Find the smallest subset of nodes in a directed graph so that when test points are inserted at these nodes, each logic cone in the resulting circuit is smaller than or equal in size to a given number of inputs. To conduct a short exhaustive test, we need to partition the circuit. Since there is no theoretically optimal way to do this, we will show how to use the global-search algorithm to obtain a "good" set of test points.

Global Search: Weighted Fan-out. To segment a circuit into logic cones of required sizes, we iteratively select test points. Those logic cones whose size is greater than the user-determined logic-cone size are called offending logic cones. At each iteration, we examine every node to determine how many logic cones would be removed from the offending-logic-cone list if that node were selected as a test point.

In evaluating a node as a potential test point, we want to know the number of inputs "blocked" by this node for each logic cone. If no input is blocked by a node, then the node does not segment any logic cone. For example, the node N shown in Figure 4 completely blocks both inputs A and B from the primary output. It does not block input C, however, since a path exists from input C to the primary output without going through N. C is said to have a fan-out. When N is used as a test point, the original output will have three inputs: N, C, and D. Because N does not block input C, C is not seg-

mented out of the original logic cone.

DEFINITION 1: *Blocked(N,O) is the number of inputs blocked from the logic cone O by inserting a test point at N.*

The example in Figure 4 shows us that whenever a fan-out occurs, we lose the ability to block some of the inputs. Thus, tracking (using weights) the occurrences of fan-out and reconvergent fan-in would allow us to calculate Blocked(*N*,*O*) efficiently. The weight of a primary input is 1 at the input node.

Each time a fan-out occurs, the weight of the input at the next, or succeeding, node becomes a fraction of the weight of its predecessor. The fraction is proportional to the number of fan-outs that occur.

For example, Figure 4 shows that the weight of primary input pins A and B at node *N* is 1, because no fan-out of those input pins has yet occurred. However, the weight of input pin C at node *N* is 1/2, because input pin C has two outputs, only one of which drives node *N*.

When the input pins reconverge, the weights from the same primary inputs are added. The function input_weight(*Pin*,*N*,*O*) is defined as follows:

$$\text{input\_weight}(Pin,N,O) = \begin{cases} 1 & \text{if } N \text{ is a primary input} \\ \sum_i \dfrac{\text{input\_weight}(Pin,i,O)}{\text{fan\_out}(i,O)} \end{cases}$$

In the preceding equation, *i* ranges over the immediate predecessors of the node *N*, and fan_out(*i*,*O*) is the number of nodes in output logic cone *O* that are immediate successors of node *i*. We can compute input_weight(*Pin*,*N*,*O*) by propagating every primary input of logic cone *O* to its successors, along with the value of input_weight.

THEOREM 1: *A primary input pin is blocked by a node* N *relative to logic cone* O *if and only if input_weight(*Pin,N,O*) = 1. If input_weight(*Pin,N,O*) is a fractional value, there is a path from the* Pin *to* O *that does not pass through* N.

Theorem 1 enables us to calculate Blocked(*N*,*O*) from the input-weight function. The test point selected at each iteration is the node *N* that blocked the largest number of inputs from the largest number of offending cones.

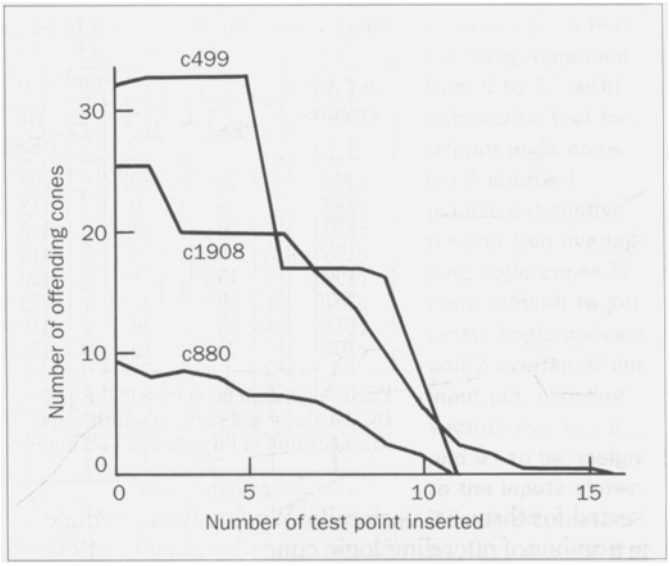


Figure 5. For some ISCAS circuits, the number of offending logic cones decreases drastically as the number of test points increases. An offending logic cone is a logic cone with too many inputs.

Results. Robert and Lala[4] show that for fan-out-free circuits with a fixed-partition factor, the number of test points increases linearly as the number of nodes in the circuit increases. In fan-out-free circuits, each node resides in and, therefore, affects only one logic cone. But in circuits with fan-outs, such as the ISCAS circuits, a node that resides in one logic cone may affect many logic cones. Our global approach is to select (from among all the nodes in the circuit) the test point (at each iteration) that will eliminate the largest number of offending cones at once.

The graph in Figure 5 shows the number of offending logic cones versus the number of test points

Table I. Test Points Needed to Segment a Circuit

ISCAS circuits	Number of test points (cone size = 20)						
	PEST	DC	% DC/PEST	GL	% GL/PEST	RL	% RL/PEST
c432	19	20	105	21	105	44	231
c499	8	9	112	9	112	32	400
c880	10	14	140	14	140	48	480
c1355	8	9	112	9	112	32	400
c1908	15	18	120	17	113	53	353
c2670	30	37	123	29	97	77	256
c5315	30	42	140	46	153	82	273
c7552	75	79	105	85	113	75	100

PEST: algorithm presented in this paper.
DC, GL: local and global algorithms published by Hellebrand and Wunderlich.[5]
RL: algorithm of Roberts and Lala implemented by Reference 5.

inserted for three ISCAS circuits. We drastically reduce the number of offending logic cones for circuits c499 and c1908 as we increase the number of test points. However, we see no such drastic reduction for circuits such as c880, which is dense with fan-outs and reconvergent fan-ins. For such circuits, we need to work on each logic cones individually.

Table I presents the number of test points needed to segment or partition each of the ISCAS circuits according to four different algorithms. To compare the PEST results with the other algorithms, we include a percentage that represents the published results divided by the PEST results. The results of the algorithms DC, GL, and RL are published by Hellebrand and Wunderlich.[5]

The DC algorithm performs 5 to 40 percent below that of the PEST algorithm. With one exception, the GL algorithm performed 5 to 53 percent worse than the PEST algorithm.

Pseudo-Exhaustive Test Generation

In PEST we seek the most inexpensive way of supplying the smallest set of vectors to exhaustively test all the logic cones at once. For a single n-input cone, the shortest exhaustive test has length 2^n. We can generate an exhaustive test easily by running a binary counter of length n from 0 to 2^n, as shown in Figures 6a and b.

Generating the shortest exhaustive test for two overlapping logic cones is a little more difficult (Figure 6c). Logic cones 1 and 2 overlap at one input pin. To provide an exhaustive test set for logic cone 1, we assign the vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, and $\mathbf{a}_4$ of Figure 6c to the inputs, as shown. The reader can verify that the column vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, and $\mathbf{a}_4$—reordered, for example, as $\mathbf{a}_2$, $\mathbf{a}_4$, $\mathbf{a}_3$, and $\mathbf{a}_1$—will still present exhaustive row vectors. Thus, the inputs of logic cone 2 can have assignments as shown in Figure 6c.

To summarize, there are two parts to the test-generation problem. First, we need a generator that produces vectors which, together, contain an exhaustive test (vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, and $\mathbf{a}_4$, for example). Second, we need a method of assigning the outputs of the test generator to the inputs of the logic cones so that each logic cone will receive an exhaustive set of test patterns.

In circuits that we have examined, the overlapping among logic cones can be very complicated. An input pin may belong to as many as 60 different logic cones. In general, it is not possible to generate a pseudo-exhaustive test using the binary-counting sequence

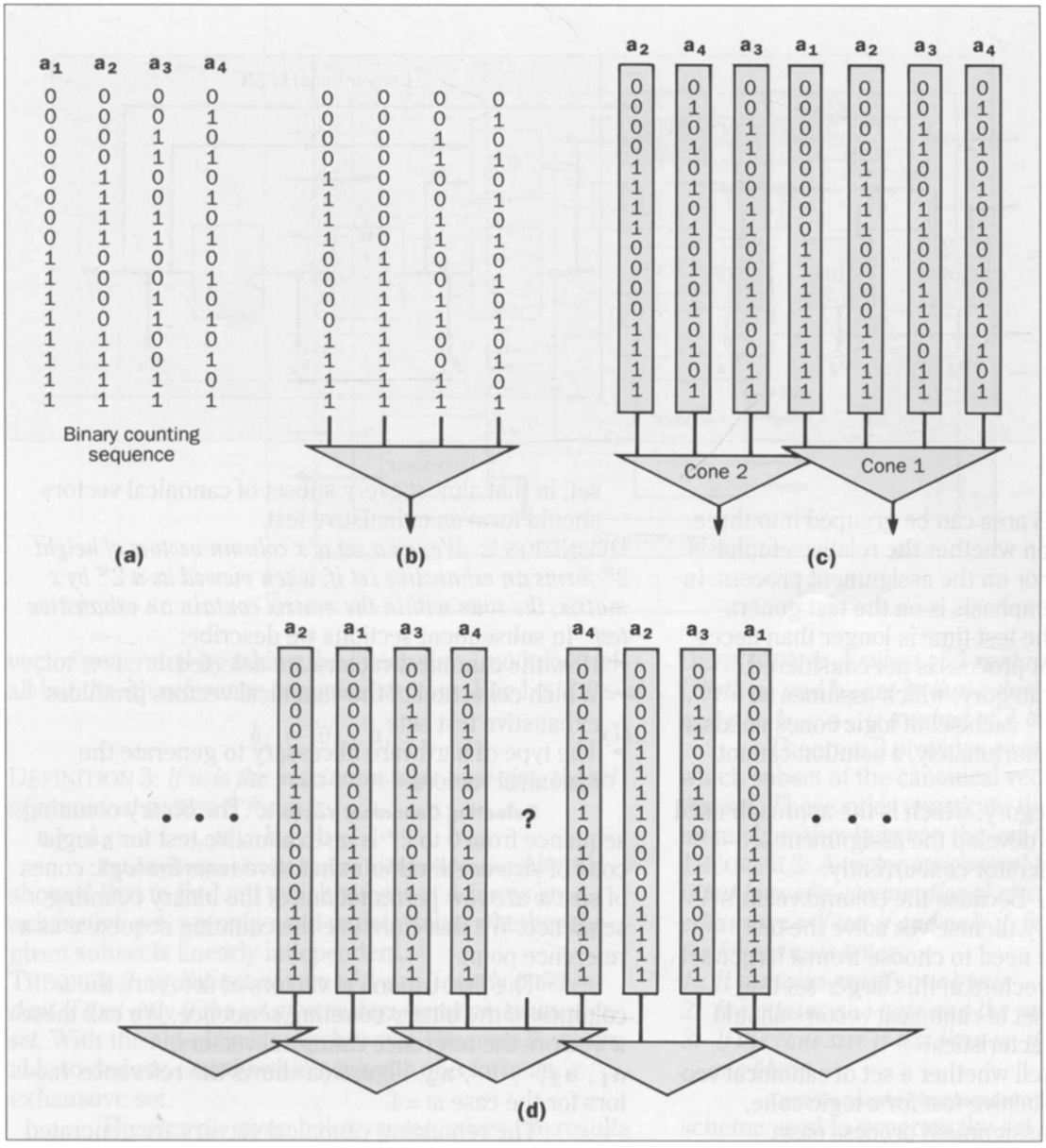


Figure 6. (a) A binary counting sequence from 0 to 2^n is (b) an exhaustive test for an *n*-input logic cone. (c) A shortest pseudo-exhaustive test for two overlapping logic cones is more difficult to generate. Logic cones 1 and 2 overlap at one input pin, allowing vectors a_1, a_2, a_3, and a_4 to be assigned to the inputs shown. Reordered, vectors a_2, a_4, a_3, and a_1 of logic cone 2 still present exhaustive row vectors. (d) Overlapping of logic cones, in general, eliminates the possibility of using a binary counting sequence exclusively to generate a pseudo-exhaustive test.

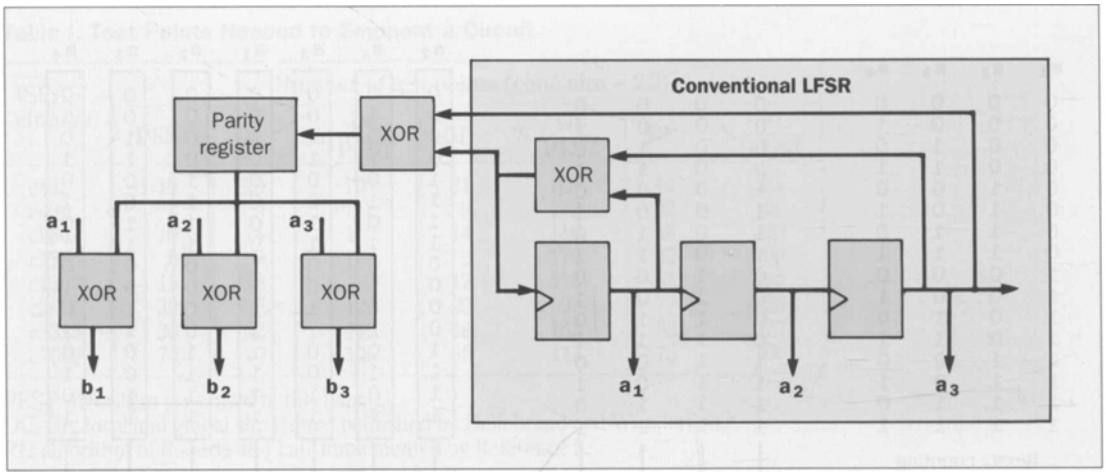


Figure 7. A scheme used to generate the set A for the case $w = 3$.

exclusively (Figure 6d).

Research in this area can be grouped into three categories, depending on whether the relative emphasis is on the test generator or on the assignment process. In the first category, the emphasis is on the test generator.[8,9] In these cases, the test time is longer than necessary and the assignment process is not considered.

In the second category, which assumes an assignment procedure,[10] each set of logic cones needs a customized solution. Unfortunately, a solution cannot always be found.

In the third category, which is the approach used in our PEST scheme, we develop the assignment procedure and the test generator concurrently.

The New Method. Because the column vectors from a counter cannot by themselves solve the test-generation problem, we need to choose from a larger set of vectors. We call the vectors in this larger set the canonical vectors. The set of canonical vectors should have the following characteristics:

- It should be easy to tell whether a set of canonical vectors provides an exhaustive test for a logic cone, thereby making the assignment process easy.
- The canonical vectors selected should be the "best" set, in that almost every subset of canonical vectors should form an exhaustive test.

DEFINITION 2: *We say a set of x column vectors of height 2^w forms an exhaustive set if, when viewed as a 2^w by x matrix, the rows within the matrix contain an exhaustive test.* In subsequent sections we describe:

- How the canonical vectors are selected
- Which collection of the canonical vectors produces exhaustive test sets
- The type of hardware necessary to generate the canonical vectors.

Selecting Canonical Vectors. The binary counting sequence from 0 to 2^w is an exhaustive test for a logic cone of size w. All other exhaustive tests for logic cones of size w are row permutations of the binary counting sequence. We therefore use the counting sequence as a reference point.

The first canonical vectors of set A are the w columns of the binary counting sequence. We call these w vectors the reference canonical vectors, $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_w$. Figure 6a shows the reference vectors for the case $w = 4$.

The remaining canonical vectors are generated from the reference canonical vectors. Let $\mathbf{b}_i$ be the

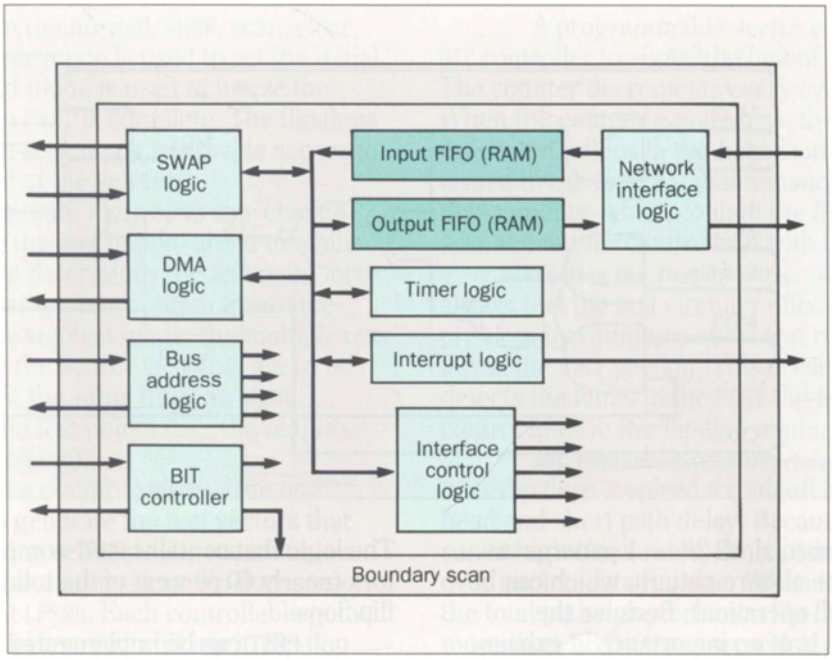


Figure 8. This diagram shows the three major BIST schemes used in the NIC. PEST tests the random sequential logic (see lightly shaded portions), regular structure BIST tests the RAM (see darkly shaded portions), and boundary scan tests the interconnections between NIC and other chips on the circuit board.

vector generated by taking the linear sum (modulo 2) of all but the ith reference canonical vectors. For $1 \leq i,j \leq w$

$$b_i = \oplus_{j \neq i} a_j \quad (1)$$

DEFINITION 3: *If w is the maximum logic cone size, the set of canonical vectors A for w is*

$\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_w, \mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_w$.

Exhaustiveness and Linear Independence. Akers[11] showed that to find out which subset of A forms an exhaustive set, we only need to determine whether the given subset is linearly independent.

THEOREM 2: w *distinct vectors of* A *are linearly independent if and only if the set of* w *vectors forms an exhaustive set.* With the aid of linear algebra and Theorem 2, we are able to deduce exactly which subsets of A form an exhaustive set.

Theorem 3, given below, summarizes the results with the aid of Definition 4.

DEFINITION 4: *A subset of A contains an i-pair if, for some i, both* $\mathbf{a}_i$ *and* $\mathbf{b}_i$ *are in that subset of A. For example,* $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2, \mathbf{a}_3$ *is a subset of A that contains a 2-pair.*

Theorem 3 provides easy rules for recognizing which subset of the canonical vectors forms an exhaustive set. These rules constitute the basis for the assignment algorithm between the set A and the input pins.

THEOREM 3: *A vector assignment from the set* A *to the input pins of a combinational circuit produces a pseudo-exhaustive self-test if and only if, for every logic cone, one of the following is true:*

1. *It contains exactly one* i*-pair*
2. *It contains no* i*-pair and the number of* bs *is even*
3. *If the cone size is less than* w*, it contains no more than one* i*-pair.*

Test-Generator Implementation. Figure 7 shows a scheme used to generate the set A for the case $w = 3$. Because the linear-feedback shift register (LFSR) has a

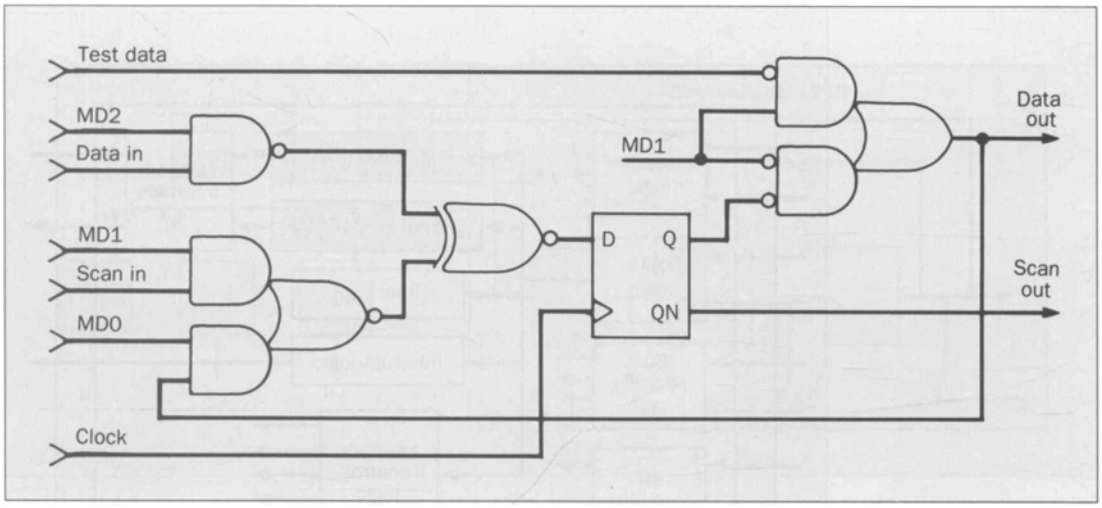


Figure 9. PEST isolates the combinational logic from the sequential circuit by observing every flip-flop input and controlling every flip-flop output. The PEST flip-flop shown can be reconfigured using mode lines MD0 through MD2 to perform the normal, MISR, scan, clear, and hold modes.

primitive feedback polynomial, all $2^w - 1$ patterns are generated (excluding the all-zero pattern, which can be produced using a "reset" operation). Because the sequence of test vectors is of no importance in exhaustive testing of combinational circuits, we use an LFSR.

The LFSR outputs are the reference vectors $\mathbf{a}_i$s. The column vectors $\mathbf{b}_i$s are derived from the $\mathbf{a}_i$s, as indicated earlier in Equation 1.

PEST in the NIC Chip

The NIC chip is an ASIC developed by AT&T Bell Laboratories. It provides an asynchronous, microprocessor-like interface to the data-flow network in AT&T's digital signal processor. The NIC is fabricated using AT&T's 1.25-μm (micrometer) CMOS process. It contains about 200,000 transistors on a die that measures 502 millimeters per side. The device is packaged in a 256-pin ceramic-chip carrier. Figure 8 shows the three major BIST schemes used in the NIC:

- PEST for the random sequential logic
- Special RAM BIST
- Boundary scan, which is used to test interconnections between chips on a circuit board.

The logic that contains PEST comprises 112,000 transistors (nearly 60 percent of the total chip), including 1400 flip-flops.

PEST can be implemented in a circuit by following these steps:

1. Replace all flip-flops in the circuit with PEST flip-flops.
2. Add test points as needed.
3. Add a test generator and distribute the test vectors.
4. Add BIST control logic to conduct concurrent testing.

These steps are described here in detail.

Step 1. Our PEST scheme isolates the combinational logic from the sequential circuit by observing every flip-flop input and controlling every flip-flop output. We reconfigure the registers (i.e., groups of flip-flops) into MISRs so we can compact test results in the flip-flops. To control the flip-flop output, the outputs of the registers are multiplexed with the test data generated by the source LFSR.

During PEST sessions, the built-in test (BIT) controller sets the multiplexer to apply the test data to the subsequent combinational logic. Figure 9 shows a PEST flip-flop design that performs these functions. This flip-flop can be reconfigured using mode control lines MD0

through MD2 to perform the normal, MISR, scan, clear, and hold modes. The clear mode is used to set the initial MISR values, and the hold mode is used to freeze the resulting signature when PEST is complete. The flip-flops are connected in a single-scan chain to provide access to the signatures at the end of the self-test.

Step 2. The maximum logic cone size chosen was 20, thereby limiting the test time to about one million cycles. PEST software determines the test-point locations but permits the user to restrict them from time-critical paths. During the self-test mode, the multiplexers select the test data from the source LFSR that are to be presented to the logic. At the same time, an MISR observes the inputs of the test points (i.e., the outputs from the previous logic cones).

Step 3. We used a circuit similar to the one described in Figure 7 to generate the test vectors that stimulate the circuit during self-test. The NIC stimulus comes from the existing boundary-scan flip-flops, reconfigured as multiple 20-bit LFSRs. Each controllable point in the circuit (i.e., every primary circuit input, flip-flop output, and test point) is assigned by the PEST tool to one of the LFSR outputs in such a way as to present each logic cone with an exhaustive vector set. To simplify vector routing and reduce routing overhead, PEST permits the user to restrict vector assignment.

Step 4. The NIC was designed to perform concurrent self-test. Using a test-bus interface, a maintenance processor located external to NIC sends self-test instructions to the NIC chip. The BIT–controller logic interprets these instructions and then configures the chip to perform the desired pseudo-exhaustive, FIFO RAM, and boundary-scan tests.

The NIC BIT controller also contains logic to control the clocks. The NIC normally operates using several asynchronous clocks. But, to assure stable and predictable BIST operation, the multiple clock inputs are replaced with a single test clock. A clock-switching circuit was developed to perform this clock multiplexing in a stable fashion.

A programmable vector counter was added to the BIT controller to signal the end of the self-test sessions. The counter decrements every cycle during self-test. When the counter equals zero, the BIT controller places the PEST flip-flops in the hold mode, and an interrupt is issued to the external maintenance processor. The flip-flop contents, which contain the final signatures, are then scanned out for comparison with their expected values.

Testing the Test Circuitry. BIST schemes cannot always test the test circuitry effectively. PEST solves this problem by running a short test routine before it performs the BIST session. This prescribed initialization test detects the faults in the PEST flip-flops, scan chain, and control lines in the flip-flop mode.

NIC Implementation Overhead. The design of the PEST flip-flops involved a tradeoff between low area overhead and short path delay. Because the NIC chip must run at high clock rates, short path delay took precedence over implementation size. When PEST flip-flops are used, the total path delay between flip-flops increases by no more than 1.3 ns (nanoseconds).

The PEST overhead consists of 26,712 transistors, which is 24 percent of the random sequential logic, but only 14 percent of the total chip. This overhead consists of:

- Logic needed to convert original flip-flops to PEST flip-flops
- Test points (29 were added)
- MISR logic
- Control-line buffers
- Boundary-scan cell modifications for LFSR and MISR operation while PEST is being run.

Fault-Simulation Results. The bottom line of a BIST methodology is the total fault coverage. The fault coverage is 98 percent (after only 10,000 BIST vectors plus the initialization test).

Conclusions

PEST is a design concept that embodies an ideal test strategy. It provides effective, 100-percent stuck-at fault coverage for the integrated circuit without requiring

expensive ATE, labor-intensive test generation, or fault simulation. It also gives the system easy access to this self-test capability on the chip.

We have discussed one way of incorporating this ideal into real circuit designs. Our experience shows that PEST lives up to the highest design expectations.

Acknowledgment

I would like to acknowledge the support from my department head, Dick Campbell, and supervisors Rod Tullous, Scot Davidson, and Shianling Wu. This work would not have been possible without the technical contributions of James Medlock, John Malleo-Roach, Paul Rutkowski, and Subrata Roy, and the encouragement of Gregg Fackler.

References

1. E.J. McCluskey and S. Bozorgui-Nesbat, "Design for Autonomous Test," *IEEE Transactions on Computers*, Vol. C-30, No. 11, November 1981, pp. 866-875.
2. E.C. Archambeau and E.J. McCluskey, "Fault coverage of pseudo-exhaustive testing," *Proc. 14th International Symposium on Fault-Tolerant Computing*, Kissimmee, Florida, July 1984, pp. 141-145.
3. E. J. McCluskey, "Verification testing - A pseudo-exhaustive testing technique," *IEEE Transactions on Computers*, Vol. C-33, No. 6, June 1984, pp. 541-46.
4. M. W. Roberts and P. K. Lala, "An algorithm for the partitioning of logic circuits," *IEE Proc.*, Vol. 131, Pt. E, No. 4, July 1984, pp. 113-118.
5. H-J Wunderlich and S. Hellebrand, "Tools and Devices Supporting the Pseudo-Exhaustive Test," *Proc. European Design Automation Conference*, Glasgow, Scotland, March 1990, pp. 13-17.
6. J. F. Udell, Jr. and E. J. McCluskey, "Pseudo-Exhaustive Test and Segmentation: Formal Definitions and Extended Fault Coverage Results," *CRC Tech. Rep.*, No. 88-12, Stanford University, 1988.
7. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *Proc. IEEE International Symposium of Circuits & Systems (ISCAS)*, Kyoto, Japan, June 1985, pp. 663-698.
8. D.T. Tang and L.S. Woo, "Exhaustive Test Pattern Generation with Constant Weight Vectors," *IEEE Transactions on Computers*, Vol. C-32, No. 12, December 1983, pp. 1145-1150.
9. D.T. Tang and C.C. Chen, "Logic Test Pattern Generation Using Linear Codes," *IEEE Transactions on Computers*, Vol. C-33, No. 9, September 1984, pp. 845-849.
10. E. Barzilai, D. Coppersmith, and A. L. Rosenberg, "Exhaustive generation of bit patterns with applications to VLSI self-testing," *IEEE Transactions on Computers*, Vol. C-32, No. 2, February 1983, pp. 190-194.
11. S.B. Akers, "On the use of linear sums in exhaustive testing," *Proc. 15th International Symposium on Fault-Tolerant Computing*, University of Michigan, June 1985, pp. 148-153.

(Manuscript received November 2, 1990)