PEST - A Tool for Implementing Pseudo-Exhaustive Self Test

Eleanor Wu
AT&T Bell Laboratories
Engineering Research Center
Princeton, New Jersey 08540

Paul W. Rutkowski
AT&T Bell Laboratories
Whippany, New Jersey 07981

Abstract
PEST is a CAD tool for implementing pseudo-exhaustive self test in integrated circuits. PEST's unique features are: 1. Parallel testing of all cones, 2. Global test point selection, and 3. A new cost effective scheme for test vector generation. AT&T's Network Interface Controller chip was designed using PEST. Effective 100% fault coverage without fault simulation or test generation was obtained with less than 24% transistor overhead. This paper begins with an overview of PEST's approach. The test generation algorithm is then described. Finally, the implementation of PEST in the Network Interface Controller is presented.

1.0 Pseudo-Exhaustive Self Test in PEST
In PEST the combinational portions of a circuit are partitioned into structural parts, each of which is then exhaustively tested. Flip-flops in the circuit are arranged in a scan chain and are tested by an initialization test. Because this scheme detects all stuck-at faults we can guarantee 100% fault coverage without fault simulation.

The structural parts mentioned above are based on combinational logic cones. In PEST every input to a flip-flop is considered a pseudo-output and every output of a flip-flop a pseudo-input. The combinational logic cones are traced from the primary outputs and/or pseudo-outputs to either the primary inputs or pseudo-inputs (see Figure 1). The number of primary/pseudo inputs driving a primary/pseudo output is called the cone size of the combinational logic cone. If a cone has a cone size x then it will take 2^x test vectors to exhaustively test that cone. The largest cone size of the circuit will determine the minimum length of the test. In order to limit the total test length it may be necessary to further partition the larger cones into smaller cones. This can be done by observing appropriate locations (test points) and controlling them using multiplexers. One of the two outputs of the software tool PEST is a file containing the list of test point locations determined by the PEST partition algorithm.

The second of the output files of PEST contains the routing information between the test generator and the primary/ pseudo inputs. Exhaustive test vectors are routed to the inputs of each cone from a dedicated test generator. The test generator is a modified Linear Feedback Shift Register (LFSR). The assignment between the output of the test generator and the inputs of the cones is complicated when the cones overlap. In PEST a new vector assignment algorithm based on linear algebra provides the solution for a minimum length exhaustive test.

To complete the picture we need to describe how the test results are evaluated. The flip-flops in the circuit are organized into a multiple input signature analyzer (MISR). The outputs of the cones go directly into the signature analyzer to be compacted. At the end of the testing session the contents of the flip-flops are scanned out and compared to a good signature to determine the condition of the chip under test.

PEST is the only pseudo-exhaustive self test scheme capable of testing all cones simultaneously. This eliminates test scheduling, simplifies self-test control, and shortens test time. PEST is able to accomplish this because 1. The dedicated test generator is not a part of the original circuit and 2. All of the flip-flops act as MISRs during self test.
2.0 Pseudo-Exhaustive Test Generation from Canonical Vectors

Generating the shortest pseudo-exhaustive test is a NP-complete problem [6]. In our generic combinational circuit the size of the largest cone is \( w \). Since any pseudo-exhaustive test must exhaustively test the largest cone, the absolute lower bound on the length of a pseudo-exhaustive test is \( 2^w \), and the longest test is the exhaustive test \( 2^n \). The shortest test for a given circuit is somewhere between \( 2^w \) and \( 2^n \).

Most pseudo-exhaustive test generation solutions [refs. 2,3,5,6,8,10,11], accept the worst case assumption that all of the cones in the circuit have size \( w \). Thus, their test generation methods generate a test such that each and every subset of inputs of size \( w \) receives an exhaustive test. Except for the case where \( w = n - 1 \) [6], the worst case assumption results in a test of length strictly greater than \( 2^w \). Figure 2 shows the distribution of cone sizes in an actual circuit. Clearly, only a small number of the cones in the circuit have cone size \( w \).

![Figure 2. An Example Cone Size Distribution](image)

2.1 The New Method

We explore the potential for finding a test with a length between the absolute minimum \( 2^w \) and the length for the worst case assumption. Since the worst case assumption prevents the existence of a test of the length equal to the absolute lower bound, we need to make use of the actual input-output dependency information of a circuit.

At the heart of our test generation method is a mapping between the \( n \) input pins of a given circuit and a set \( A \) of canonical column vectors of height \( 2^w \). The canonical vectors in set \( A \) and the algorithm for the mapping are thus crucial for the success of our method. In section 2.2 the selection of the canonical vectors for \( A \) will be explained. Section 2.4 contains the properties of the set \( A \) that are used in the mapping algorithm. The hardware necessary to generate the canonical vectors is presented in section 2.5.

2.2 Selecting Canonical Vectors

**Definition 1:** We say a set of \( x \) vectors from the set \( A \) forms an exhaustive set if, when viewed as a \( 2^w \times x \) matrix, the rows within the matrix form an exhaustive test.

In selecting the set \( A \) of canonical vectors the property we need is that a large number of the \( w \) element sets of \( A \) form exhaustive tests.

**Reference Vectors** The binary counting sequence from 0 to \( 2^w \) is an exhaustive test for a cone of size \( w \). All other exhaustive tests for 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 the set \( A \) are the \( w \) columns of the binary counting sequence. Call these \( w \) vectors the reference canonical vectors, \( a_1, a_2, ... , a_w \). Figure 3 shows the reference vectors for the case \( w = 3 \).

![Figure 3. Binary Counting Sequence used as Reference Vectors](image)

The remaining canonical vectors are generated from the reference canonical vectors. Let \( b_i \) be the vector generated by taking the linear sum (modulo 2) of all but the \( i \)th reference canonical vectors, \( 1 \leq i \leq w \) or

\[
b_i = [ \oplus_{j=1}^{i-1} a_j ] \oplus a_i
\]

**Definition 2:** The set of canonical vectors \( A \) is \( \{ a_1, a_2, ..., a_w, b_1, b_2, ..., b_w \} \).
2.3 Exhaustiveness & Linear Independence

Each canonical vector can be thought of as a linear function where the values at the coordinates are the coefficients. A well known theorem in linear algebra (adopted to our mod 2 field) says:

A set of distinct linear functions is independent if and only if no subset sums to zero (mod 2).

The vectors from $A$ have length $2^w$. If $w$ distinct vectors of $A$ are linearly independent then we can show that this set of $w$ vectors must form an exhaustive set (see definition 1). Thus, to find out which subset of $A$ forms an exhaustive set we need only to determine whether the given subset is linearly independent.

**Theorem:** $w$ distinct vectors of $A$ are linearly independent iff the set of $w$ vectors form an exhaustive set.

2.4 Properties of the set $A$

**Definition 3:** A subset of $A$ contains an i-pair if for some $i$, both $a_i$ and $b_i$ are in it.

For example: $(a_1, a_2, b_2, a_3)$ is a subset of $A$ that contains a 2-pair.

The characterization of subsets of $A$ that form exhaustive sets is presented in four parts in the Theorem. The parts indicate a grouping of the subsets of $A$. The first group contains subsets of $A$ with no i-pairs. The second contains subsets with one i-pair and the third contains subsets with more than one i-pair.

**Theorem:** Let $A$ be the set of canonical vectors. Let $X$ be a subset of $A$ ,

- a. If $X$ has less than $w$ elements and contains no i-pairs then it forms an exhaustive set.
- b. If $X$ has $w$ elements and contains no i-pairs then it forms an exhaustive set iff the number of $b_i$'s is even;
- c. If $X$ contains only one i-pair then it forms an exhaustive set.
- d. If $X$ contains more than one i-pair then it does not form an exhaustive set.

The Theorem provides easy rules for verifying that a mapping between the set $A$ and the input pins will give the circuit the pseudo-exhaustive test that the circuit requires. Since the canonical vectors are of height $2^w$ the length of the test is $2^w$.

**Corollary:** A vector assignment to the input pins of a combinational circuit produces a pseudo-exhaustive test iff for every cone one of the following is true:

- a. It contains exactly one i-pair or
- b. It contains no i-pair and the number of B's is even or
- c. If the cone size is less than $w$ then it contains no more than one i-pair.

It may happen that the set $A$ does not accommodate the constraints required by the cones. In that case the canonical set with length $2^{w+1}$ should be used.

2.5 Test Generator Implementation

Figure 4 shows a scheme used to generate the set $A$ for the case $w = 3$. Because the LFSR has a primitive feedback polynomial, all $2^w - 1$ patterns (excluding the all zero pattern which can be taken care of by a "reset" operation) are generated. Because the sequence of test vectors is of no importance in exhaustive testing of combinational circuits, an LFSR is used. The LFSR outputs are the reference vectors $a_i$'s. The column vectors $b_i$'s are derived from the $a_i$'s as indicated by equation 1.

![Figure 4. Test Vector Generator](image-url)
3.0 PEST in the Network Interface Controller chip

3.1 The Network Interface Controller chip

The Network Interface Controller (NIC) chip is an application-specific integrated circuit developed by AT&T Bell Laboratories to provide an asynchronous microprocessor-like interface to AT&T's signal processor data flow network. The NIC is fabricated using AT&T's 1.25 um twin-tub CMOS process and consists of approximately 200,000 transistors on a die 502 mils per side. The device is packaged in a 256-pin ceramic leaded fine-pitch chip carrier. Figure 5 shows the three major built-in self-test (bist) schemes used in the NIC: Pseudo-exhaustive self-test for the random sequential logic (lightly shaded portions), special RAM bist (darkly shaded portions) and Boundary Scan. The PEST'ed logic consists of 112,000 transistors (nearly 60% of the total chip) and contains 1400 flip-flops.

![Figure 5. NIC Block Diagram](image)

### 3.2 Procedure for implementing PEST

**Step 1: Replace all Flip-flops with PEST Flip-flops**

Our pseudo-exhaustive self-test scheme isolates the combinational logic from the sequential circuit by observing every flip-flop input and controlling every flip-flop output. The observability is accomplished by reconfiguring the registers (i.e., groups of flip-flops) into multiple-input shift registers (MISR's). The controllability is accomplished by multiplexing the outputs of the registers with the test data generated by the source LFSR. During pseudo-exhaustive self-test sessions the multiplexer is selected to apply the test data to the subsequent combinational logic.

A PEST flip-flop design that performs these functions is shown in Figure 6. This flip-flop can be reconfigured using the mode lines MD[2:0] 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 pseudo-exhaustive self-test is complete. The flip-flops are connected together in a single scan chain to provide access to the signatures at the end of self-test.

![Figure 6. PEST Flip-flop](image)

**Step 2: Add Test Points as Needed**

For the NIC chip the maximum logic cone size was chosen to be 20, thereby limiting the exhaustive test time to approximately one million cycles. During self-test mode the multiplexers select test data from the source LFSR to be presented to the subsequent logic. At the same time, the inputs of the test points (the output of the previous logic cones) are observed by a MISR. The test point locations are determined by the PEST software. PEST permits the user to make restrictions on test point locations to prevent time-critical paths from getting test points.

**Step 3: Add Test Generator and Distribute Test Vectors**

A circuit similar to the one described in section 2.5 was used 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 LFSR's. Each controllable point in the circuit (i.e., every primary circuit input, every flip-flop output, and every test point) is assigned to one of the LFSR outputs in such a way as to present each logic cone with an exhaustive vector set. This vector assignment is done by the PEST tool. To simplify vector routing and reduce routing overhead, PEST permits the user to make vector assignment restrictions.

**Step 4: Add Bist Control Logic for Concurrent Testing**

The NIC was designed to perform in-system concurrent self-test. Self-test instructions are sent to the NIC chip from an external maintenance processor via a modified...
ETM interface. The BIT Controller logic interprets these instructions and configures the chip to perform the desired pseudo-exhaustive, FIFO ram, and boundary scan tests. The various test operations are done by changing the control lines of the PEST flip-flops and the other bist elements.

The NIC BIT Controller also contains logic to control the clocks. The NIC normally operates using several asynchronous clocks. Therefore, to assure stable and predictable bist operation, all bist is performed by replacing the multiple clock inputs 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 signal the end of self-test sessions. When self-test is performed, the counter decrements every cycle. When the counter equals zero the PEST flip-flops are placed in the HOLD mode and an interrupt is issued to the external maintenance processor. The flip-flop contents (containing the final signatures) are then scanned out for comparison with their expected values.

3.3 Testing the Test Circuity

One drawback of some bist scheme is the inability to effectively test the test circuitry. PEST solves this problem by running a short test routine before the actual bist session. This prescribed "Initialization Test" detects the faults in the PEST flip-flops, scan chain, and flip-flop mode control lines. The Initialization Test consists of a prescribed sequence of instructions that configure the flip-flop mode lines to different states that exercise the bist logic.

3.4 NIC Implementation Overhead

Timing Overhead The design of the PEST bist flip-flops involved a trade-off between low area overhead and short path delay. Because the NIC chip is required to run at high clock rates, short path delay took precedence over implementation size. Using the PEST bist flip-flops, the total flip-flop to flip-flop path delay increases by no more than 1.3 ns (under nominal conditions of 5.0 volts, 25 deg C junction temperature, and nominal processing conditions for the 1.25 um CMOS standard cell technology).

Hardware Overhead The PEST overhead is 26,712 transistors, which is 24% of the random sequential logic but only 14% 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 during PEST

3.5 Fault Simulation experimental results

The bottom line of a BIST methodology is the total fault coverage. The fault coverage after only 10,000 BIST vectors plus the initialization test is 98%. Result compaction was not accounted for in this simulation. Preliminary experimental results showed that 1% - 3% of fault coverage may be lost due to compaction.

References