On Random Pattern Generation with the Selfish Gene Algorithm for Testing Digital Sequential Circuits

Junwu Zhang   Michael L. Bushnell
Dept. of Elec. and Computer Eng., Rutgers University
96 Frelinghuysen Rd., Piscataway, NJ 08854-8088
zhjunwu@caip.rutgers.edu   bushnell@caip.rutgers.edu

Vishwani D. Agrawal
Dept of ECE, Auburn University
200 Broun Hall, Auburn, AL 36849-5201
vagrawal@eng.auburn.edu

Abstract

A selfish gene *(SG) algorithm differs from the* genetic algorithm *(GA) because it evolves genes (characteristics) that provide higher fitness rather than evolving individuals with higher fitness. We enhance the spectral method of sequential circuit test generation by using a SG algorithm. The objects of evolution are the Hadamard spectral matrix, non-linear* digital signal processing *(DSP) filtering cutoff values, vector holding time, and relative input phase shifts, which are all modeled as genes. These characteristics, extracted from compacted test vectors, are used to create new vector sequences to be further compacted with higher fault coverage. Alternatively, new vectors were generated by holding randomly selected vectors and then randomly perturbing some bits in 8-bit chunks of bit streams. Both the SG algorithm and holding with bit-perturbation can outperform the previously-published spectral method in either fault coverage, or shorter vector length, or both. The SG algorithm is often superior to random bit-perturbation but it requires more CPU time.*

1 Introduction

We propose sequential test generation, using random bit-perturbation. Initially, random patterns are generated, fault simulated, and compacted, using *linear reverse order restoration* (LROR) vector compaction [9, 10, 12, 22]. A *selfish gene* (SG) algorithm [5, 6] extends the test sequence, randomly perturbs bits within 8-bit chunks of the *primary input* (PI) patterns, and evolves probabilities of flipping bits. Winning mutations cause the bit flipping probabilities to change to favor the bits being set as in that mutation. Results of this method matched Giani *et al.*'s spectral test generator [7], but it may be faster, because random bit flipping uses less computation than matrix algebra.

We also improved on Giani *et al.*'s spectral *automatic test-pattern generation* (ATPG) work [7]. They applied non-linear *digital signal processing* (DSP) techniques to sequential ATPG. After generating random patterns and fault simulating and compacting them, the correlation of the serial input bit streams in the circuit test patterns was calculated with respect to the *basis vectors* of the Hadamard matrix. These vectors, called *spectra*, are orthogonal, and from the correlation coefficients for the bit stream, the original test vectors can be reconstructed. Giani *et al.* used non-linear signal filtering to remove basis vector correlations that were less than a correlation coefficient cutoff (or in the noise floor). They also used random vector holding, since Guo *et al.* [11, 12] showed that this vastly improved sequential circuit fault coverages.

The SG genetic algorithm [5, 6] has the *gene* as the basic element of evolution, rather than the *individual*, as in the genetic algorithm. A *gene* is a specific genetic trait of the individual, and is inherited from parents. An *allele* is the specific value assigned to a gene. The *genotype* is the collection of genes that completely determines the individual's traits. The evolution optimization is preserved by the genotype, and directly passed to the next generation, rather than being preserved by individuals and passed to the next generation via the crossover mechanism. In the Hadamard matrix, genes are assigned for each spectra, for the cutoff coefficient for non-linear filtering, for the vector holding time, for the number of bits flipped in the matrix to introduce entropy into the system, and for the phase shift for each PI, relative to PI_1. The SG algorithm independently evolves these genes, to produce better test patterns. Its evolutionary spectral techniques are often better than Giani *et al.*'s method for a particular circuit. On average, Giani's method tests almost as many faults, but has 80.4% longer test sequences.

It was observed that a combination of all bit-perturbation and enhanced spectral methods is most effective, and we implemented this in the SG algorithm framework. On the ISCAS '89 circuits, this method outperformed Hitec [21] (a time frame expansion type of deterministic sequential test generator) by detecting 3.64% more faults, using 74.2% fewer test vectors. We outperformed Giani *et al.*'s method, by detecting the same average number of faults with 4.7% fewer vectors.

In this paper, Section 2 describes prior work, Section 3 describes new simulation-based ATPG methods, Section 4 presents results, and Section 5 draws conclusions.

0-7803-8580-2/04 $20.00 Copyright 2004 IEEE

2 Prior Work

ATPG techniques for sequential circuits are either deterministic or simulation-based. Deterministic techniques use either branch-and-bound search [2, 3, 21], or observed test sequence properties, to derive tests. The simulation-based techniques fault-simulate sequences that are generated randomly, or otherwise, and keep only the patterns useful for detecting faults. They have the advantage of being adaptable to any simulatable fault model. Simulation-based ATPG efficiency is controlled by the *concurrent fault simulation* algorithm [25] and the sequence generation method. *Random patterns, weighted random patterns,* cost-function directed patterns [4], and *genetic algorithms* (GAs) [19, 23, 24] were used with simulation-based ATPG.

Sequential test compaction techniques have reduced the test length drastically by simulating the test sequences and discarding useless vectors for detecting faults [22]. Several methods generate new sequences using the compacted sequence for guidance [7, 8, 11, 12]. *Vector restoration*-based compaction [9, 10, 22] is the fastest method.

Guo *et al.* [11, 12, 13] generated new sequences by perturbing and randomly holding (repeating) some vectors in the compacted test set to extend it, probabilistically selecting some vectors (with probability 1/6) or subsequences of length 5 (with probability 5/6) from the compacted test set to extend it, and generating weighted random patterns using the probabilities of 1s and 0s in the compacted test set.

Hsiao *et al.* [16] used a hybrid ATPG method that combined simulation-based and deterministic algorithms. A simulation-based GA algorithm first generated tests for easy faults. Then, a deterministic algorithm found tests for random-pattern resistant faults. Finally, faults not detected in the first two stages were targeted individually.

Giani *et al.* [7] used DSP to analyze the digital spectra of the compacted sequence and generate new tests. They used the state and fault-detection data from the compacted vectors to generate new sequences [8]. All of these methods were superior in terms of fault coverage, test length, and CPU time. In the spectral algorithm, a random test set is first created and compacted. First, 0 bits in the compacted test set are replaced with -1s. Then the algorithm finds the correlation coefficients of the PI bit stream with the basis vectors, and filters out coefficients that fall below the *cutoff*. Then, all -1 bits are replaced with 0s.

The Hadamard matrix H is computed as follows:

$$H(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad H(k) = \begin{bmatrix} H(k-1) & H(k-1) \\ H(k-1) & -H(k-1) \end{bmatrix}$$

where $k = 2, \ldots, n$. They showed how the following 8-bit stream is extended: $[1,0,1,1,1,0,1,0]^T$. They replaced each 0 with a -1 and got: $[1,-1,1,1,1,-1,1,-1]^T$. Next, the bit stream was left multiplied by an 8×8 Hadamard matrix, $H(3)$, to obtain the spectral coefficients:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ -2 \\ 2 \\ 2 \\ -2 \\ -2 \\ 2 \end{bmatrix}$$

Every coefficient in the coefficient vector whose absolute value was less than 4 (the *cutoff*) was set to 0. After filtering, they got: $[0,1,0,0,0,0,0,0]^T$. Left multiplying the new vector by $H(3)$ yielded: $[1,0,1,0,1,0,1,0]^T$, so the 4th bit in the original vector changed from 1 to 0.

Corno *et al.* [5] implemented Dawkins's new *selfish gene* algorithm [6], where the *gene*, rather than the *individual*, is the element of evolution. A conventional genetic algorithm uses a sizable number of individuals as the element of evolution, and creates new individuals through *crossover* and *mutation* operations. Survival of individuals depends on their ranking by a fitness function. However, the SG algorithm maintains a generic genotype, consisting of all genes, and the probabilities of various *alleles* occupying the *locus* (location) of each gene. The SG algorithm uses a *virtual population*, where specific individuals are not of interest and not stored. Instead, it models the gene pool, and stores, as a vector, the marginal probabilities that the various alleles for a particular gene will actually occupy that locus. The SG algorithm does not use a crossover operator, but instead uses a *tournament* mechanism, where new individuals are generated solely for competing in a tournament, and statistics are recorded only about the winner, which is still determined by a *fitness function*. The winner can reproduce, but the loser may not. All alleles belonging to the winner increase their selection probability, at the expense of those of the loser. The *steady state* condition occurs when, for each locus L_i, there exists an allele, a_{ij}, whose probability exceeds a threshold p_t (usually 0.95). The SG algorithm always found a better solution than the genetic algorithm for a 0/1 multiple knapsack problem.

Hamzaoglu and Patel [15] created a deterministic sequential test generator based on *atom*, a combinational ATPG program. This test generator uses improved unique sensitization, improved backtracing with conflict checks, multiple PI and flip-flop assignments, lookback search techniques, support regions (the logic that affects state justification for the faulty line), and state space reduction. Ichihara and Inoue [17] used the sequential holding theorem [20] to improve sequential ATPG by generating sequential circuit tests using combinational ATPG.

2.1 Sequential Holding Theorem

Guo *et al.* [12] showed that holding some vectors of the compacted test set improved fault coverage for sequen-

tial ATPG. *Acyclic* sequential circuits, which contain no cyclic paths involving flip-flops, have an ATPG complexity similar to combinational circuits [1, 14, 17, 18, 20]. We show why holding is effective for acyclic and cyclic sequential circuits, using the converse of a theorem by Min and Rogers [20].

Consider an acyclic sequential circuit C, called a *Type-S* circuit, and its kernel combinational circuit C^B, which is created by replacing all flip-flops in C with wires, and is called a *Type-C* circuit [20]. Let f^B be a stuck-at fault at arc $v_i \rightarrow v_j$ in C^B. Consider two cases:

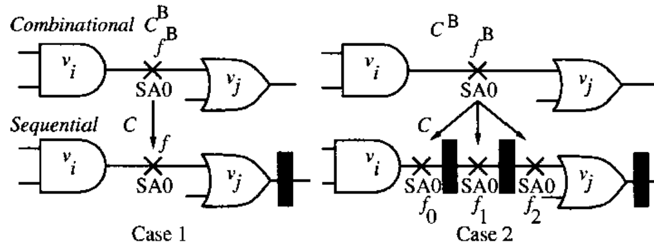

Figure 1: Corresponding Combinational and Sequential Faults

1. *In circuit C, arc $v_i \rightarrow v_j$ has no flip-flops. Thus f^B corresponds to one fault f of the same stuck-at value at arc $v_i \rightarrow v_j$ in C.*
2. *In circuit C, arc $v_i \rightarrow v_j$ has p flip-flops. So, f^B corresponds to $p+1$ faults $f_0, f_1, \ldots, f_p$, all of the same stuck-at value at arc $v_i \rightarrow v_j$ in C (see Figure 1).*

We restate the converse of Min and Rogers's theorem [20] with a more succinct proof using Figure 1, where black boxes represent flip-flops.

Theorem 2.1 Holding a test vector V^B for a testable stuck-at fault f^B in combinational circuit C^B $d+1$ times, where d is the sequential depth of the corresponding acyclic sequential circuit C created by adding flip-flops at any wire of C^B, gives a test sequence for all sequential faults in C corresponding to f^B [20].

Proof: *Adding flip-flops at any wire is viewed as increasing the wire length. The equivalent length increases when adding one flip-flop equals the signal propagation distance in one clock period. Sequential circuit C can be viewed as combinational circuit C^B with its wires replaced by longer wires. Since f^B can be detected by V^B, then this vector sensitizes and propagates fault f^B. If V^B is held at inputs of C long enough for all logic transitions to finish, then the circuit eventually reaches steady state, and the correct logic values required to sensitize and propagate fault f^B will be set at all wires. Circuit C with long wire delays will eventually have the same logic values as circuit C^B. For both cases 1 and 2 above, the corresponding sequential faults will be detected. Since the sequential depth of C is d, when any vector is held for $d+1$ clock cycles, at the $(d+1)$st clock cycle, there will be no more signal transitions at any wire and the circuit reaches steady state. Thus, we need to hold V^B at most $d+1$ clock cycles to test all sequential faults corresponding to f^B.*

For case 2, the vector set resulting by repeating V^B $d+1$ times will test all $p+1$ sequential faults $f_0, f_1, \ldots, f_p$. The combinational vector, upon holding, detects the corresponding faults in all acyclic sequential circuits formed by any arbitrary distribution of flip-flops in the combinational circuit. If f^B is untestable, we do not know whether its corresponding sequential fault is testable.

The following procedure tests some of the cyclic sequential circuit faults using Theorem 2.1 [20]. Cut the minimum number of wires in sequential circuit C to make it an acyclic circuit C_a. Each cut adds one *S-input* and one *S-output* to the original circuit. Replace all flip-flops in C_a by wires to get the kernel combinational circuit ${C_a}^B$. The subset of the testable stuck-at fault set of C_a^B that can be tested without using the *S-inputs* and *S-outputs* corresponds to a sequentially testable fault set F in C. Each fault in F is tested by repeating the test vectors for C_a^B at most $d+1$ times, where d is the sequential depth of C_a.

3 Selfish Gene Algorithm Formulation

We now formulate simulation-based sequential ATPG with compaction using the SG algorithm. Algorithm I below is used for combining any simulation-based ATPG with the SG Algorithm [5, 6]. We customize this algorithm into five different algorithms, and ultimately, we combine all of the features of those five into a single algorithm.

- **Step 1:** On the first iteration, generate random vectors and compact them, but on subsequent iterations, use the compacted test sequence of the last iteration.
- **Step 2:** In all algorithms except Spectral + Selfish Gene 2 (Section 3.2.1), for each vector in the compacted sequence, randomly generate a vector holding time between 0 and 64 and hold the vector accordingly to extend the sequence. If the holding time is 0, discard the vector. Holding a vector for some clock cycles is very important for high fault coverage. Further expand the test sequence (explained below for each algorithm).
- **Step 3:** Evolve the genotype using the expanded sequence to get better fault coverage (explained below).
- **Step 4:** Fault simulate and compact the best sequence. If the fault coverage is satisfactory or 125 iterations are finished, stop. Otherwise, iterate.

3.1 Bit-Perturbation + SG Method

Randomly flipping some bits and holding the compacted vectors for some clock cycles produces good test

sequences. We use random bit-perturbation, in periodic 8-bit chunks of PI bit streams, instead of the matrix filtering operation, to generate new test sequences. There is no spectrum concept in this bit-perturbation method.

Genes and Alleles. There is a gene for each bit in the original sequence. The probabilities p_i of flipping each bit i are also stored, and initially set to 0.5.

Selfish Gene Algorithm (Algorithm I): We use the SG algorithm for random bit perturbation.

- **Step 2:** *Hold vectors to form a new sequence S_{new}. After that, the following procedure is performed 50 times:*
  - *Generate a random perturbation ϵ in the range $(-0.05, +0.05)$ for each bit i in the original sequence. Flip each bit with $p_i + \epsilon > 0.5$. Do this twice, to generate two new sequences, S_1 and S_2, which are fault-simulated. The winning sequence has the highest fault coverage. Change the bit-flipping probabilities in the gene. If the bit was flipped (not flipped) in both the winner and the loser, there is no change. If it flipped in the winner but not in the loser, $p_i = p_i + 0.01$, otherwise $p_i = p_i - 0.01$.*

  Finally, only the bit with highest p_i in each 8-bit chunk of each PI bit stream is flipped, provided that its $p_i > 0.5$ (only one flip/chunk). This sequence becomes the extended sequence S_{new}.
- **Step 3:** *The final probabilities p_i as evolved in* **Step 2** *are discarded and reset to 0.5 for the next round of holding and perturbation.*

This bit-perturbation method differs from generating random test patterns [25]. We are not perturbing bits in a random test set, but in periodic chunks of the compacted test set. Vectors in the compacted test set are good for detecting faults, so by perturbing this set, we reuse its good properties and detect new faults.

3.2 Enhanced Spectral Test Generation

We use the SG algorithm [5, 6] to improve the spectral test method by evolving better parameters for spectral matrix filtering. We then use the SG algorithm to randomly introduce noise into the Hadamard matrix, and to phase shift PI bit streams to detect more faults.

3.2.1 Enhanced Hadamard Matrix Methods

We evolve the best digital spectrum, the best vector holding time, and the DSP filtering cutoff to generate good sequences using the compacted test set.

Principles. Row vectors in the Hadamard matrix that are dominant show up in the testing bit stream with a larger contribution. Non-dominant spectral vectors show up with a smaller contribution. By fault-simulating two bit streams, we find which detects more faults. We reward the spectra of the better bit stream with higher probability of survival. We use the spectra of the compacted test set as a seed and search for better spectra.

Genes and Alleles. The Spectral + Selfish Gene 1 Algorithm decomposes any given 16-bit/32-bit stream (depending on the dimension of the Hadamard matrix used) into a linear combination of the basis vectors in the Hadamard matrix. We assign a gene whose allele is the correlation coefficient of each column in the 16×16 Hadamard matrix, to get 16 genes. Any 16×1 vector can be linearly composed from these 16 genes. We evolve the best spectra that compose the best vector. Initial values of these genes are the spectral coefficients from compacting the initial, randomly-generated test sequence.

The spectral coefficients range from -32 to 32 for a 32×32 Hadamard matrix. The single cutoff value used in matrix filtering of the absolute values of coefficients is an integer randomly picked in this range, $\{0, 32\}$.

Spectral + Selfish Gene 1 (Updated Algorithm I):

- **Step 2:** *Extend the test set with two vector sequences, which use the best spectra genes with some random perturbation, simulate them, compare fault coverages, and use the best sequence. For non-linear filtering, randomly select a cutoff value.*
- **Step 3:** *Favor the winner's spectral coefficients in the genotype by adding the winner's spectral coefficients to and subtracting the loser's from the genes.*

Spectral + Selfish Gene 1 Pseudo-Code [5, 6]

```
spectrum SG (spectrum initial)
{
  spectrum Best, Spectrum1, Spectrum2 ;
  initialize all coef[i] to initial[i] ;
  Best = initial ; /* use initial as the first best */
  do {
     Spectrum1 = select_individual () ;
     Spectrum2 = select_individual () ;
     /* tournament */
     if (flt_cvrg (Spectrum1) > flt_cvrg (Spectrum2)) {
        evolve_spectrum (Spectrum1, Spectrum2) ;
        if (flt_cvrg (Spectrum1) > flt_cvrg (Best))
           Best = Spectrum1 ; /* update best */
     } else {
        evolve_spectrum (Spectrum2, Spectrum1) ;
        if (flt_cvrg (Spectrum2) > flt_cvrg (Best))
           Best = Spectrum2 ; }
```

```
    } while (steady_state ()==FALSE or
        max_iteration==FALSE) ;
  return Best ;
}
```

Individual Selection

```
spectrum select_individual ()
{
  spectrum Individual ;
  for (each locus i = 1...H_size )
      Individual[i] = coef[i] + random_numb(-ε, ε) ;
  return Individual ;
}
```

Evolution Mechanism

```
evolve_spectrum (spectrum winner, spectrum loser)
{
    for (each locus i = 1...H_size)
     coef[i] = coef[i] +
          (winner[i]-loser[i])× evolve_step ;
}
```

Variable *max_iteration* is 125 and *evolve_step* is 1. H_{size} is the Hadamard matrix size being used. Variable ϵ, the amplitude of random perturbation, is set to $H_{size}/8$.

The Spectral + Selfish Gene 2 algorithm holds vectors for some clock cycles to extend the test sequences, and evolves genes for the best cutoff value and holding time.

Genes and Alleles. Add one gene R_{best} for the holding time. Add one gene C_{best} for the filtering cutoff value between 2-14 for $H(4)$ (16×16 matrix) or 2-30 for $H(5)$ (32×32 matrix). R_{best} and C_{best} are initialized as described below in **Step 2, 1 (a)**.

Spectral + Selfish Gene 2 (Updated Algorithm I):

- **Step 2:**
  1. *Use modified random holding to expand the given test sequence:*
     (a) *If iteration = 1, randomly hold vectors for R = 2 to 50 clock cycles, compare fault coverages of the 49 different holding times, and select the best holding time as R_{best}. Use the sequence corresponding to R_{best}. In this process, try all matrix filtering cutoff values in the range 2 to 14 for $H(4)$ or 2 to 30 for $H(5)$ and save the best cutoff value in C_{best}.*
     (b) *Otherwise, try random holding for i clock cycles, i= $R_{best}-4$, ..., $R_{best}+4$, compare the fault coverages, and use the best sequence of the 9 generated. The cutoff values are tried in the range $j = C_{best}-4$, ..., $C_{best}+4$.*
  2. *Use* **Step 2** *of the Spectral + Selfish Gene 1 algorithm to evolve the spectra using C_{best} and extend the test sequence.*
- **Step 3:** *Update R_{best} and C_{best} with the best values from* **Step 2**. *Evolve the spectral coefficients as in* **Step 3** *of the Spectral + Selfish Gene 1 algorithm.*

3.2.2 Perturbed Hadamard Matrix Method

Principles. Instead of using a pure Hadamard matrix for spectral filtering, we use a Hadamard matrix with *noise*. Some elements in the Hadamard matrix are flipped, from 0 to 1 or from 1 to 0. We started with the pure $H(4)$ (16×16) matrix. In each subsequent round, we incrementally flipped 16 more elements, randomly selected in the matrix. For the 17th round, 256 matrix entries were flipped, the same as the number of elements in $H(4)$. For each round, each element flipped was selected randomly, so it can be one flipped before or one never flipped. These 16 rounds of flipping make the matrix random. The matrix filtering operation using either $H(4)$ or a perturbed $H(4)$ is periodic with a period of 16 clock cycles.

Genes and Alleles. The gene g_{best} added for the perturbed 16×16 Hadamard matrix method is the best number of rounds of random flips (0 to 16) done on the matrix. In each round, 16 matrix elements are randomly selected and flipped. The matrix positions flipped are not remembered. Initialization is described below in **Step 2**.

Updated Algorithm I:

- **Step 2:** Generate the perturbed Hadamard matrix:
  1. For the first iteration, g_{flip} values from 0 to 16 are applied to generate 17 matrices, with the first matrix being the pure Hadamard matrix, and the last one being almost random.
  2. Otherwise, g_{flip} takes the three values $g_{best}-1$, g_{best}, and $g_{best}+1$ to generate three matrices.

  Using each of the perturbed matrices above, apply the matrix filtering operation on the sequence by holding vectors in the compacted sequence from the last iteration to generate 17 (3) expanded sequences in the first (subsequent) iteration. Select the sequence with the highest fault coverage to extend the test sequence.
- **Step 3:** Replace g_{best} with the g_{flip} value for the sequence with highest fault coverage in **Step 2**.

Filtering using a perturbed Hadamard matrix differs from random bit-perturbation, because it decomposes each bit stream into a linear combination of its row vectors. The combination coefficients are the spectra. Perturbing one basis vector may change the entire bit pattern in a range equal to the basis vector size, so this filtering is periodic.

3.2.3 Spectral + Phase Shifting Method

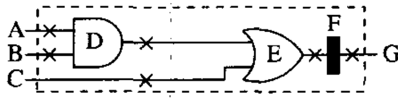

Figure 2: Example Sequential Circuit C_0

Principles. Consider the circuit C_0 in Figure 2. The black boxes represent D flip-flops. The complete test set and the faults tested by each vector are given in Table 1.

Table 1: Test Set for Circuit C_0

A	B	C		Faults Detected
1	1	0	→	sa0 – AD, BD, DE, EF, FG
1	0	0	→	sa1 – BD, DE, CE, EF, FG
0	1	0	→	sa1 – AD, DE, CE, EF, FG
0	0	1	→	sa0 – CE, EF, FG

Denote the circuit in Figure 3 as C. Sub-circuit part I is exactly the same as C_0. The alphabetic gate numbers of C_0 from Figure 2 are shown in parentheses in Figure 3. Denote sub-circuit part II as C_1. Notice that the output of sub-circuit C_1, i.e., the output of gate 14, is 0 iff the input sequence of (1000) is given to input $I1$ in Figure 3.

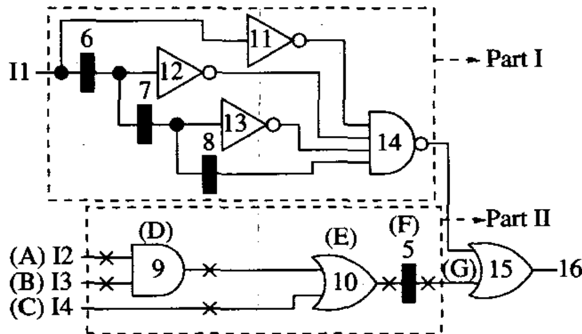

Figure 3: Sequential Circuit C

In circuit C, if gate 14 can be set to 0, then we can use the test set in Table 1 to test faults in sub-circuit C_0. We use (1000) at input $I1$, and the input values for $I2$, $I3$, and $I4$ in Table 1, to form the test sequence:

cycle#	$I1$	$I2$	$I3$	$I4$
*cycle*1 →	1	1	1	0
*cycle*2 →	0	1	0	0
*cycle*3 →	0	0	1	0
*cycle*4 →	0	0	0	1

Since gate 14 is set to 0 only at clock cycle 4, only fault effects that approach the input of gate 15 at clock cycle 4 can propagate to a *primary output* (PO). Fault effects that approach gate 15 at other times are blocked. At clock cycle 4 the tested faults are sa1 faults at AD, DE, CE, EF, and FG. We repeat the test set once from clock cycle 5 to 8. Since gate 14 is set to 0 only at clock cycle 8 this time, only fault effects that approach the input of gate 15 at clock cycle 8 can propagate to a PO. Thus, only the same set of faults can be tested and no new faults are tested.

The test set for C_0 is a *sensitization sequence* X, though this sequence also partially propagates the fault effects. The sequence to set gate 14 to 0, (1000), is the fault *propagation sequence* Y. X and Y have the same length. When we repeat them to extend the test sequence, they are phase-locked with each other and no new faults are tested. In order to test new faults, we have to rotate either the sensitization sequence X or the propagation sequence Y to break the phase-lock. From clock cycles 5 to 8, we rotate the sequence X by 1 and keep sequence Y as it is.

cycle#	$I1$	$I2$	$I3$	$I4$
*cycle*5 →	1	1	0	0
*cycle*6 →	0	0	1	0
*cycle*7 →	0	0	0	1
*cycle*8 →	0	1	1	0

New faults sa0 at CE, EF, and FG are tested. We rotate the sequence X by 2 and form another set:

cycle#	$I1$	$I2$	$I3$	$I4$
*cycle*9 →	1	0	1	0
*cycle*10 →	0	0	0	1
*cycle*11 →	0	1	1	0
*cycle*12 →	0	1	0	0

New faults sa0 at AD, BD, and DE are tested. Rotating X by 3 clock cycles leads to the following set.

cycle#	$I1$	$I2$	$I3$	$I4$
*cycle*13 →	1	0	0	1
*cycle*14 →	0	1	1	0
*cycle*15 →	0	1	0	0
*cycle*16 →	0	0	1	0

A new fault (sa1 at BD) is detected, so all faults in sub-circuit C_0 are detected. By doing such digital phase shifting 3 times, together with the original non-shifted sequence, we detected all faults in sub-circuit C_0.

Useful Cases for Phase Shifting. A sequential circuit contains three sub-circuits A, B, and C, each driven by different PIs. B controls the sensitization of faults in A, and C controls the propagation of faults in A. Sometimes B and C are the same circuit. Both B and C cannot stay in their desired states for more than one clock cycle in a row, due to their structure. So, B (or C) must leave the state and then return to it. This loop corresponds to an input sequence of B (or C). To test faults in A, we synchronize this input sequence with the input sequence of A, but with different shift offsets. Other methods may also

Table 2: Results of the Spectral Method with SG Algorithms 1 and 2, Perturbed Matrix, and Phase Shifting

Ckt.	Pure Hadamard [7]			Spectral + Selfish Gene 1			Spectral + Selfish Gene 2			Spectral + Perturbed Hadamard Matrix				Spectral + Phase Shifting			
	Flt. Det.	# Vec.	Time (min.)	Flt. Det.	# Vec.	Time (min.)	Flt. Det.	# Vec.	Time (min.)	Flt. Det.	# Vec.	Time (min.)	f	Flt. Det.	# Vec.	Time (min.)	p
s382	364	567	1	364	594	12.5	364	574	4.7	**364**	**538**	**4.9**	9	364	544	102.4	2
s400	382	588	1.5	382	519	4.4	382	509	4.8	382	492	3.1	6	**382**	**489**	**34.0**	4
s713	476	89	0.4	476	84	1.1	476	84	1.1	**476**	**78**	**0.4**	7	476	80	4.7	16
s1196	1239	244	1.2	1238	232	35.6	1239	232	23.1	**1239**	**227**	**290.0**	9	1238	235	630.8	16
s1238	1283	255	1.1	1282	245	4.2	**1283**	**240**	**26.1**	1283	240	146.4	7	1281	241	26.4	14
s1423	1416	927	16.3	1416	1051	516.0	**1417**	**4359**	**732.2**	1416	1224	146.9	12	1416	1884	403.8	6
s1488	1444	384	3.2	**1444**	**342**	**31.0**	1444	342	67.2	1444	427	51.0	8	1444	402	162.6	6
s1494	1453	388	2.9	**1453**	**328**	**33.4**	1453	328	58.0	1453	379	25.2	2	1453	489	123.7	14
s5378	**3643**	**734**	**43.5**	3510	1409	49.1	3631	931	1316.2	3317	1923	712.0	10	3535	2591	140.2	4
Avg.	1300	464	7.9	1285	534	76.4	1299	844	248.2	1264	614	153.3	8	1288	773	181.0	9

test faults detected by phase-shifting. This sequence also tests C_0, and phase-shifting cannot generate it:

cycle#	*I1*	*I2*	*I3*	*I4*	*cycle#*	*I1*	*I2*	*I3*	*I4*
*cycle*1 →	1	0	0	0	*cycle*9 →	1	0	0	0
*cycle*2 →	0	0	0	0	*cycle*10 →	0	0	0	0
*cycle*3 →	0	0	1	0	*cycle*11 →	0	1	1	0
*cycle*4 →	0	0	0	0	*cycle*12 →	0	0	0	0
*cycle*5 →	1	0	0	0	*cycle*13 →	1	0	0	0
*cycle*6 →	0	0	0	0	*cycle*14 →	0	0	0	0
*cycle*7 →	0	0	0	1	*cycle*15 →	0	1	0	0
*cycle*8 →	0	0	0	0	*cycle*16 →	0	0	0	0

Genes and Alleles. The genes are the best phase shifts d_{best_k}, ($k = 2, 3, \ldots, \#PI$), for each PI, relative to PI_1. PI_1 is never shifted. The alleles are the amounts of phase shift (0 to 16 clocks) for each PI.

Updated Algorithm I:

- **Step 2:** *Hold vectors to form a new sequence* S_{new}. *For the first iteration, select* d_k *from the range* $(0, 16)$ *for all PIs. Subsequently, select* d_k *from the range* $(d_{best_k} - 2, \ldots, d_{best_k} + 2)$. *Generate three random numbers* d_{1_k}, d_{2_k}, d_{3_k} *in the range for each* PI_k *(except for* PI_1*). The input bit stream* S_k*, corresponding to* PI_k*, is delayed by* d_{i_k} *clock cycles and put back into the initial sequence. Perform this phase shifting three times to generate three expansions for each PI. Matrix filtering is not used. Use the sequence with highest fault coverage.*
- **Step 3:** *Save the best of* d_{1_k}, d_{2_k}, *and* d_{3_k} *(with the highest fault coverage in* **Step 2***) for each* PI_k *in* d_{best_k}.

4 Results

Using the combined SG Spectral method, we generated tests for the ISCAS '89 benchmarks in Table 2. The second through sixth major blocks of the table represent the results for the Pure Hadamard [7], the Spectral + Selfish Gene 1, the Spectral + Selfish Gene 2, the Spectral + Perturbed Hadamard Matrix, and the Spectral + Phase Shifting methods. In this table, as also in subsequent tables, the best result for each circuit is in bold. A result is best if it has the highest fault coverage, or equal fault coverage but shorter vector length, or equal fault coverage and vector length but shorter CPU time (measured on a SunBlade 2000 workstation). Each algorithm extended the test set 125 times, but may have achieved the best results before the last iteration. For the Perturbed Hadamard Matrix method, f is the best number of matrix element perturbations. For Phase Shifting, p is the maximum number of bit-stream shifts.

Spectral + Selfish Gene 2 always does better than Spectral + Selfish Gene 1 by achieving the same or higher fault coverage with shorter test sets. So, evolving the holding time and the DSP cutoff is productive. No one method in the table is clearly superior, as each was best for at least one circuit. For some circuits, the Perturbed Hadamard Matrix method detects fewer faults than the Pure one, but the Perturbed method is better for 6 out of 9 circuits. The spectral matrix filtering method adds periodic properties to the compacted tests. Periodicity is the reason why both the Pure and Perturbed matrix methods succeed. For Spectral + Phase Shifting, different circuits need different phase shifts. We experimented with up to 256 shifts, but the results are poor. Shifting more clock cycles causes the test patterns to be less like the compacted patterns, so we lose the good features of the compacted test set. Phase Shifting never reached a higher fault coverage than the Pure Hadamard method, so it is useful but not required. The Avg. row of Table 2 is a reasonable way to compare these algorithms, and we see that Giani *et al.*'s method [7] is still the best, but Spectral + Selfish Gene 2 attains almost the same fault coverage, though with significantly longer tests. It detected one more fault than Giani *et al.*'s

method for s1423, but 12 fewer for s5378.

The results for the Bit-Perturbation method are compared with Giani *et al.*'s results [7] and the Best of Enhanced Spectral Methods (i.e., the bold entries from Table 2) in Table 3. We see that random Bit-Perturbation gave the same fault coverage as the Pure Hadamard method for all circuits except s1423, where it detected 1 more fault, and s5378, where it detected 4 fewer. Perturbing no bits or all bits in an 8-bit chunk was unsuccessful. If one flips all of the bits in compacted vectors that detect faults, one discards all of the good patterns for fault detection. Repeating the exact same compacted vectors detects very few new faults. While repeating the vectors and then flipping some bits, one reuses the good patterns of the compacted vectors and detects new faults.

In order to generate good test sequences using simulation-based ATPG, holding is important. Using holding with the Bit-Perturbation + SG method and a vector compactor, the test sets are essentially just as good as those from the Pure Hadamard method of Giani *et al.*, and the test lengths are similar. The Bit-Perturbation method detected one more fault than Giani *et al.*'s method on s1423, with only slightly more test patterns. The Best of Enhanced Spectral Methods had the best results for 7 out of the 9 circuits, and the fault coverage was almost as good as Giani *et al.*'s, but the test length was far longer, because of the huge number of patterns generated to detect one more fault in s1423. The CPU time for all new methods in the tables includes compaction time, and is much higher than for the Pure Hadamard method, because our algorithms used omission-based compaction in their last pass (which is very time-consuming), whereas the Pure Hadamard method only used LROR compaction.

In Table 4, we use all random Bit-Perturbation and Enhanced Spectral Methods (evolving spectra, holding time, and cutoff value; perturbed matrix filtering; and phase shifting). We combine all of these techniques into one SG framework, in which vector compaction is the core and the above techniques are used to extend sequences. We compared our method with SEST [3], Gentest [2], and Hitec [21] for ISCAS '89 benchmarks in Table 4. For the average results row, the comparison universe is the set of circuits with results for all four test generators. We do not include Giani *et al.*'s results from the last major column in the average, because of missing results. The SG Spectral ATPG system has the highest average fault coverage, and the lowest average test vector length. In Table 4, the SG Spectral algorithm has lower fault coverage than Gentest on circuit s38584; than Pure Hadamard on s5378; and than Hitec on s208, s400, s420, s820, and s838. The SG Spectral test generator had the best results for 22 out of 30 circuits. In three circuits marked with *, the SG Spectral algorithm results were far worse than existing algorithms, because it could not initialize all flip-flops. Our results were also better than other recent work [13, 15, 17], as well.

5 Conclusions

It was shown that random holding of the combinational test vectors [7, 12] can test corresponding sequential faults not only in balanced sequential circuits, but also any general acyclic sequential circuit, based on the work of Kim *et al.* [18]. If partial-scan design makes the sequential circuit acyclic in test mode, one can use holding.

We developed a new sequential ATPG method in which we perturb bits in 8-bit chunks of the input test vector streams. We also investigated enhanced spectral sequential ATPG methods, where we used a selfish gene algorithm to evolve the spectra used for non-linear signal filtering of the test vectors, the cutoff for non-linear signal processing coefficients, the vector holding time, the amount of entropy inserted in the Hadamard DSP matrix, and the phase offset between different PI bit streams. Table 4 gives averages for all circuits with results for each test generator, and for common circuits among all methods. In both cases, the SG spectral test generator performed the best. These improvements were observed in Tables 2, 3, and 4:

1. Perturbing some bits after randomly holding vectors in the compacted test set gave equally good results to using a Hadamard matrix to expand the sequence. With the same fault coverage, it shortened the test sequence for s400, s713, s1196, s1238, s1423, and s1494, but lengthened it for s382 and s1488. It was more than twice as fast as the enhanced spectral techniques, but slower than the Hadamard spectral method, due to the SG algorithm.

2. The SG algorithm improved performance, because of its ability to incorporate multiple evolution methods for test patterns. It was just as important to evolve holding times and cutoff values as to evolve spectra.

3. The specific signal processing matrix for spectral ATPG was less important than generating entropy in the system by perturbing matrix entries (see Table 2). Adding noise to the transformation matrix improved results for s382, s400, s713, s1196, s1238, and s1494, but worsened them for s1423 and s1488.

4. Spectral + Phase Shifting (see Table 2) generated a better test sequence for s400 and worse sequences for 8 other circuits. Finding the optimal phase shift in a circuit or test set was difficult.

5. Fault coverage of the Best Enhanced Spectral Methods (see Table 2) was nearly as good as for Bit-Perturbation or a Pure Hadamard matrix, but vector lengths were longer.

These methods use longer CPU time than the Pure Hadamard method by Giani *et al.* [7], for two reasons. We used the omission-based vector compaction method in the last iteration of each experiment, and LROR in earlier

Table 3: Spectral Method vs. Random Bit-Perturbation + the SG Algorithm vs. Enhanced Spectral Methods

Ckt.	Pure Hadamard [7]			Bit-Perturbation + SG			Best of Enhanced Spectral Methods		
	Det.	# Vec.	Time (min.)	Det.	# Vec.	Time (min.)	Det.	# Vec.	Time (min.)
s382	364	567	1	364	584	1.9	**364**	**538**	**4.9**
s400	382	588	1.5	382	561	25.0	**382**	**489**	**34.0**
s713	476	89	0.4	476	82	3.8	**476**	**78**	**0.4**
s1196	1239	244	1.2	1239	239	7.6	**1239**	**227**	**290.0**
s1238	1283	255	1.1	1283	247	26.4	**1283**	**240**	**26.1**
s1423	1416	927	16.3	**1417**	**991**	**403.8**	1417	4359	732.2
s1488	1444	384	3.2	1444	426	162.6	**1444**	**342**	**31.0**
s1494	1453	388	2.9	1453	369	123.7	**1453**	**328**	**33.4**
s5378	**3643**	**734**	**43.5**	3639	741	431.7	3631	931	1316.2
Avg.	1300	464	7.9	1300	471	131.8	1299	837	274.2

iterations, whereas they only used LROR. Also, the SG algorithm is slower than Giani *et al.*'s algorithm.

For a large subset of the ISCAS '89 benchmarks, the SG Spectral algorithm detected 3.64% more faults than the best of the existing deterministic algorithms, using 74.2% fewer test vectors. For 22 of the 30 ISCAS '89 circuits, the SG Spectral algorithm gave the best result.

The limitation of this work is that for extremely large circuits, the fault simulation time of the compactor becomes excessive. Future work must find efficient ways to obtain the spectral coefficients of a sequential circuit, with many fewer fault simulation and compaction iterations.

References

[1] A. Balakrishnan and S. T. Chakradhar, "Sequential Circuits with Combinational Test Generation Complexity," in *Proc. of the Int'l. Conf. on VLSI Design*, Jan. 1996, pp. 5–35.

[2] R. Bencivenga, T. J. Chakraborty, and S. Davidson, "Gentest: the Architecture of Sequential Circuit Test Generator," in *Proc. of the Custom Integrated Circuits Conf.*, May 1991, pp. 17.1.1–17.1.4.

[3] X. Chen and M. L. Bushnell, *Efficient Branch-and-Bound Search with Application to Computer-Aided Design.* Boston: Klumer Academic Pub., 1996.

[4] K.-T. Cheng and V. D. Agrawal, *Unified Methods for Fault Simulation and Test Generation.* Boston: Kluwer Academic Pub., 1988.

[5] F. Corno, M. S. Reorda, and G. Squillero, "The Selfish Gene Algorithm: A New Evolutionary Optimization Strategy," in *Proc. of the Annual ACM Symp. on Applied Computing*, (Atlanta, Georgia), Feb. 1998, pp. 349–355.

[6] R. Dawkins, *Climbing Mount Improbable.* New York and London: W. W. Norton and Viking Penguin, 1996.

[7] A. Giani, S. Sheng, M. S. Hsiao, and V. Agrawal, "Efficient Spectral Techniques for Sequential ATPG," in *Proc. of the Design Auto. and Test in Europe Conf.*, Mar. 2001, pp. 204–208.

[8] A. Giani, S. Sheng, M. S. Hsiao, and V. Agrawal, "State and Fault Information for Compaction-based Test Generation," *J. of Electronic Testing: Theory and Applications*, vol. 18, no. 1, pp. 63–72, Nov. 2002.

[9] R. Guo, I. Pomeranz, and S. M. Reddy, "On Speeding-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits," in *Proc. of the Asian Test Symp.*, Dec. 1998, pp. 467–471.

[10] R. Guo, I. Pomeranz, and S. M. Reddy, "Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration," in *Proc. of the Design, Auto. and Test in Europe Conf.*, Feb. 1998, pp. 577–582.

[11] R. Guo, I. Pomeranz, and S. M. Reddy, "A Fault Simulation-Based Test Pattern Generator for Synchronous Sequential Circuits," in *Proc. of the IEEE VLSI Test Symp.*, Apr. 1999, pp. 260–267.

[12] R. Guo, S. M. Reddy, and I. Pomeranz, "PROPTEST: A Property Based Test Pattern Generator for Sequential Circuits Using Test Compaction," in *Proc. of the Design Auto. Conf.*, 1999, pp. 653–659.

[13] R. Guo, S. M. Reddy, and I. Pomeranz, "On Improving a Fault Simulation Based Test Generator for Synchronous Sequential Circuits," in *Proc. of the Asian Test Symp.*, Nov. 2001, pp. 82–87.

[14] R. Gupta and M. A. Breuer, "The BALLAST Methodology for Structured Partial Scan Design," *IEEE Trans. on Computers*, vol. C-39, pp. 538–544, Apr. 1990.

[15] I. Hamzaoglu and J. H. Patel, "Deterministic Test Pattern Generation Techniques for Sequential Circuits," in *Proc. of the Int'l. Conf. on Computer-Aided Design*, 2000, pp. 538–543.

[16] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal," in *Proc. of the European Design and Test Conf.*, 1999, pp. 22–28.

[17] H. Ichihara and T. Inoue, "Test Generation for Acyclic Sequential Circuits with Single Stuck-at Fault Combinational ATPG," in *Proc. of the Design, Auto. and Test in Europe Conf.*, 2003, pp. 1180–1181.

Table 4: ATPG Result Comparison: Spectral Method (with SG, Bit-Perturbation, and Phase Shifting), SEST [3], Gentest [2], Hitec [21], and Giani *et al.*'s Spectral Method [7] (* See text for explanation.)

Ckt.	Total Faults	SG Spectral		SEST		Gentest		Hitec		Pure Hadamard	
		Det.	Vec.	Det.	Vec.	Det.	Vec.	Det.	Vec.	Det.	Vec.
s27	32	**32**	**10**	-	-	32	18	-	-	-	-
s208	217	18*	4	137	255	18	10	**137**	**184**	-	-
s298	308	**265**	**81**	264	270	265	220	264	306	-	-
s344	342	**329**	**56**	329	161	329	80	329	142	-	-
s349	350	**335**	**56**	335	164	335	105	335	137	-	-
s382	399	**364**	**538**	362	3778	364	3796	363	4931	364	567
s386	350	**314**	**109**	309	475	314	201	314	311	-	-
s400	424	382	489	368	1719	382	1517	**383**	**4309**	382	588
s420	455	28*	5	179	265	28	14	**179**	**218**	-	-
s444	474	**424**	**464**	410	1221	423	2441	424	2240	-	-
s510	564	0	0	0	0	0	0	0	0	-	-
s526	555	**454**	**1088**	384	766	440	2223	365	2232	-	-
s641	467	**404**	**101**	401	230	402	121	404	242	-	-
s713	581	**476**	**78**	434	180	476	127	476	253	476	89
s820	850	811	437	722	1801	646	392	**812**	**1254**	-	-
s832	870	**818**	**712**	707	1562	661	367	817	981	-	-
s838	931	41*	4	254	275	48	18	**254**	**166**	-	-
s953	1079	**90**	**9**	86	20	85	15	89	14	-	-
s1196	1242	**1239**	**227**	1239	540	1239	362	1239	453	1239	244
s1238	1355	**1283**	**240**	1282	543	1283	346	1283	478	1283	255
s1423	1515	**1417**	**991**	347	39	419	35	-	-	1416	927
s1488	1486	**1444**	**342**	1377	1390	1355	393	1444	1294	1444	384
s1494	1506	**1453**	**328**	1383	1357	1356	421	1453	1407	1453	388
s5378	4603	3639	741	-	-	3407	408	-	-	**3643**	**734**
s9234	3964	**547**	**15**	18	5	137	7	18	24	-	-
s13207	5918	**834**	**84**	621	371	599	93	-	-	-	-
s15850	3717	**1222**	**302**	76	13	85	7	86	32	-	-
s35932	38454	**35101**	**144**	34604	801	35095	589	34697	1000	-	-
s38417	31180	**1348**	**3471**	1077	613	-	-	-	-	-	-
s38584	36101	7273	3003	6646	2459	**7678**	**142**	-	-	-	-
Average	2539	**1993**	**243**	1902	741	1906	573	1923	942		
Ave. of Common Ckts.		**949**	**320**	921	1358	922	995	949	1875	949	359

[18] Y. C. Kim, V. D. Agrawal, and K. K. Saluja, "Combinational Test Generation for Various Classes of Acyclic Sequential Circuits," in *Proc. of the Int'l. Test Conf.*, Oct. 2001, pp. 1078–1087.

[19] P. Mazumder and E. M. Rudnick, *Genetic Algorithms for VLSI Design, Layout, and Test Automation.* Upper Saddle River, New Jersey: Prentice-Hall, 1999.

[20] H. B. Min and W. A. Rogers, "A Test Methodology for Finite State Machines using Partial Scan Design," *Journal of Electronic Testing: Theory and Applications*, vol. 3, no. 2, pp. 127–137, May 1992.

[21] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," in *Proc. of the European Design Auto. Conf.*, 1991, pp. 214–218.

[22] I. Pomeranz, S. M. Reddy, and R. Guo, "Static Test Compaction for Synchronous Sequential Circuits Based on Vector Restoration," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 7, pp. 1040–1049, Jul. 1999.

[23] P. Prinetto, M. Rebaudengo, and M. S. Reorda, "An Automatic Test Generator for Large Sequential Circuits Based on Genetic Algorithms," in *Proc. of the Int'l. Test Conf.*, Jun. 1994, pp. 240–249.

[24] E. M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of a Simple Genetic Algorithm to Sequential Circuit Test Generation," in *Proc. of the European Design and Test Conf.*, Mar. 1994, pp. 40–45.

[25] D. M. Schuler, E. G. Ulrich, T. E. Baker, and S. P. Bryant, "Random Test Generation Using Concurrent Logic Simulation," in *Proc. of the Design Auto. Conf.*, Jun. 1975, pp. 261–267.