# Choice of Tests for Logic Verification and Equivalence Checking and the Use of Fault Simulation

Vishwani D. Agrawal
Bell Labs, Lucent Technologies
700 Mountain Avenue, Room 2C-476
Murray Hill, NJ 07974 USA
*va@research.bell-labs.com*

## Abstract

*A new method is proposed for checking the equivalence of two irredundant logic implementations of a combinational Boolean function. The procedure consists of generation of complete checkpoint fault test sets for both circuits. The two test sets are concatenated and both circuits are simulated to obtain the response to the combined test set. If the responses of the two circuits match for all vectors, then they are declared to be equivalent. We examine a case where this heuristic fails. In such cases, the use of fault simulation is shown to discover non-equivalence even when the two circuits produce the same output. We prove that if the two circuits were different, then some faults on the primary inputs of a composite equivalence checking circuit must be detectable. Using the simulation of single stuck-at faults at the primary inputs of that circuit, the new heuristic recommends the use of a vector set in which the Hamming distance between any two vectors does not exceed 3.*

## 1. Introduction

The problem of establishing equivalence of two logic circuits frequently occurs in digital design. In a typical scenario, a circuit may undergo changes due to technology mapping or optimization, and must retain equivalence to some previously verified version. Theoretically, this problem can posed as a Boolean satisfiability problem, which is known to be NP-complete. Effective solutions using binary decision diagrams (BDD) and other mathematical formulations often work but cannot always guarantee results. Many heuristics have been suggested in the literature some of which are quite efficient. Still the search for alternative solutions continues. An interested reader will find useful reviews of the current methods in books by Huang and Cheng [6] and Kunz and Stoffel [7].

Alternative procedures, known as *formal verification*, rely on mathematical models of the system and prove that the model has the required attributes. The general application of formal verification is in checking the correctness of an implementation against the specification. Although it is a difficult problem and the procedures are often complex, significant progress has occurred in formal verification methods. Some commercial tools have also become available. The reader is referred to the recent book by Kurshan [8]. The focus of the present contribution is the traditional practice of industry where simulation or other forms of verification provides no guarantee like the formal verification. Our attempt is toward deriving some formal conclusions from simulation. However, as the reader will note in the end that our success, at this time at least, is only partial.

The origin of this work is in the author's experience in designing circuits, as described in this paragraph. For some time, I have used a heuristic to verify the equivalence of combinational circuits. Typical situations are where circuits are synthesized by different procedures, or a circuit is modified to remove redundant faults or untestable paths or to speed up paths. I am interested in determining that no error was committed to change the function of the circuit. As a "quick" check, I separately derive tests for all stuck-at faults for the two circuits and if both tests produce the same response from the two circuits, I presume that they are probably equivalent. When the responses differ on some inputs, the faults detected by those inputs usually help in finding the error. Initially, I started using this heuristics only as a rough check. I also verified several small cases by exhaustive simulation, which provided some confidence. Yet, attempts to prove sufficiency did not succeed.

In this article, we give examples to show that the simulation strategy is not sufficient for establishing equivalence. We then propose, perhaps for the first time, the use of fault simulation. It is shown that fault simulation can uncover differences in two circuits even when all applied vectors produce identical outputs.

0-7695-0487-6/00 $10.00 ©2000 IEEE

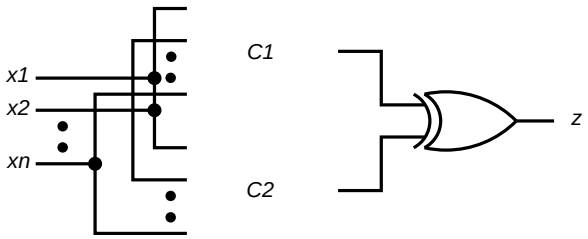



Figure 1: Equivalence checking circuit.

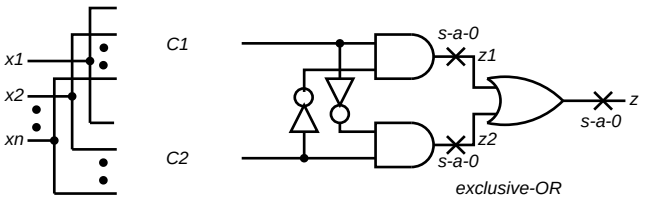



Figure 2: Dominance fault collapsing.

### 1.1. Statement of the Problem

Consider two combinational logic circuits, $C1$ and $C2$, with an identical set of input variables. For simplicity, only single output functions are considered, though the results can be easily generalized for multiple outputs. Figure 1 shows an equivalence checking setup which is usually analyzed by a logic simulator. If the two circuits are identical, then the output $z$ of the exclusive-OR gate should be 0 for all input vectors. On the other hand, the existence of an input vector that satisfies the Boolean variable $z$ (i.e., sets it to 1) immediately proves the non-equivalence of $C1$ and $C2$.

Other approaches involve the use of a test generation algorithm to find a test for the stuck-at-0 fault on $z$, or use of a redundancy identification algorithm to prove that the fault is untestable. In either case, a completely reliable procedure will have an exponential complexity.

The present approach relies on test generation but tests are generated for $C1$ and $C2$ separately, and never together. Tests are derived for all faults in each circuit. In general, it is required that the implementations be irredundant. The necessity of this requirement stems from the fact that the effectiveness of the derived tests may become questionable in the presence of redundant faults [1].

## 2. A Heuristic Examined

Suppose $T(C1)$ is a set of vectors that detects single stuck-at faults on all checkpoints in $C1$. Checkpoints are the primary inputs (PI) and all fanout branches. The following is an important result in digital testing [1]:

**Theorem 2.1** *In a combinational circuit any test set that detects all single stuck-at faults at checkpoints also detects all single stuck-at faults in that circuit.*

Thus, $T(C1)$ will detect all single stuck-at faults in $C1$. There are many efficient automatic test pattern generation (ATPG) programs available for obtaining such a test set.

Next, we obtain a similar test set $T(C2)$ for the circuit $C2$. The equivalence checking set up of Figure 1 is simulated with the combined (concatenated) vector set, $T(C1)+T(C2)$. If no output from the two circuits differs, i.e., $z = 0$ for all vectors, then we heuristically conclude that the circuits are probably equivalent.

The above conclusion is based on a conjecture, i.e., if the tests for all checkpoint faults in both circuits cannot produce a *different* output from the two circuits, then no other vector will. Because of the internal fanout structure within the exclusive-OR gate, we cannot directly conclude the redundancy of the "$z$ stuck-at-0" fault. However, the following discussion builds up arguments to support the heuristic and points to its limitations.

Suppose we apply the combined test set $T(C1)+T(C2)$ to the circuit of Figure 1 and observe that the simulated output $z$ is 0 for all vectors in the set. We would like to conclude that the stuck-at-0 fault on $z$ is untestable for all possible inputs. Notice that only true-value simulation is done here. However, our conclusion will be derived from the known fault detection characteristics of the test set.

In Figure 2, the exclusive-OR function (enclosed in the dotted line box) has been expanded in terms of Boolean primitives. Notice that the target fault $z$ s-a-0 *dominates*[1] the two stuck-at-0 faults on $z1$ and $z2$. More specifically, $z1$ s-a-0 can be detected only if a 10 pattern is applied to the exclusive-OR gate. Similarly, $z2$ s-a-0 is detectable only by a 01 pattern applied to the exclusive-OR gate. Together, the tests for these two faults represent all patterns that would detect our target fault, $z$ s-a-0.

Our objective is to use simulation-based verification and we will not try to prove the fault $z$ s-a-0 as redundant either via test generation or by some redundancy identification technique. If we can show that the tests $T(C1) + T(C2)$ detect all checkpoint faults of the entire circuit in Figure 2, then $z = 0$ for the entire test will prove the redundancy of $z$ s-a-0. The status of those checkpoint faults is discussed below:

1. Checkpoints of $C1$. All single stuck-at faults on

[1] A fault $f1$ is said to dominate the fault $f2$ when all tests of $f2$ also detect $f1$ [1].

these are detected when $T(C1)$ is applied. A single fault in $C1$ means $C2$ must be fault-free and any fault effect appearing at the output of $C1$ is always passed on to $z$. Notice that the output $z = 0$ is constantly expected during simulation. Because any deviation from that output immediately proves the non-equivalence of the circuits.

2. Checkpoints of $C2$. By an argument similar to the above, faults on these are detected at $z$ by $T(C2)$.

3. Checkpoints $x1$, $x2$, . . . $xn$ at primary inputs of the equivalence checking circuit. $T(C1)$ and $T(C2)$, contain vectors that activate faults on $xi$ to the outputs of $C1$ and $C2$, respectively. If $C1$ and $C2$ were equivalent, then these faults will not be detected. Because, any vector that activates a fault on $xi$ through $C1$ will also activate the same fault through $C2$. Thus, the fault effects will simultaneously arrive at the two inputs of the exclusive-OR gate and cancel each other. If $C1$ and $C2$ are not equivalent, then some faults on $xi$'s may be detected, but not all are guaranteed to be detected. In fact, any vector that activates a fault on $xi$ through one circuit without activating it through the other will prove that the two circuits are not equivalent.

4. Four checkpoints (fanout branches) in the exclusive-OR function. Since s-a-0 faults around an AND gate can be collapsed together, the relevant set contains six faults: 4 s-a-1 faults at the inputs of AND gates and 2 s-a-0 faults on $z1$ and $z2$. When the two circuits are equivalent, only 00 and 11 inputs will be applied to the exclusive-OR. These will detect the four s-a-1 faults and leave two s-a-0 faults (shown in Figure 2) undetected. When $C1$ and $C2$ are non-equivalent and the vectors $T(C1) + T(C2)$ produce differentiating outputs, 01 and 10, applied to the exclusive-OR function, only then the two s-a-0 faults will be detected.

Because of the uncovered checkpoint faults at primary inputs and the four s-a-0 faults in the exclusive-OR function, we cannot guarantee a redundant status for the s-a-0 fault on $z$. We make following observations:

- *Observation A:* The uncovered primary input (PI) checkpoint stuck-at faults in Figure 1 or 2 are responsible for the incompleteness of our equivalence heuristic.

- *Observation B:* Only those PI checkpoint faults that produce different outputs from $C1$ and $C2$ can be detected in the circuit of Figure 2.

We will return to these observations in subsequent sections.

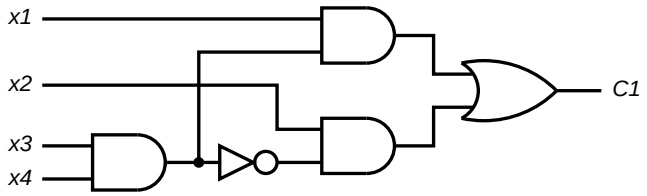



Figure 3: A multi-level implementation $C1$ for $x1x3x4 + x2\overline{x3} + x2\overline{x4}$

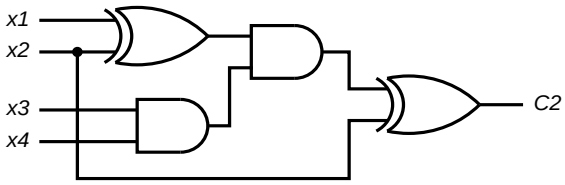



Figure 4: An exclusive-OR transform implementation $C2$ for $x1x3x4 + x2\overline{x3} + x2\overline{x4}$

We have successfully used the checkpoint test simulation for debugging implementations of 8 and 16 bit adders and many other combinational circuits of varying complexity. As is well known, simulation with properly selected inputs can effectively detect errors, but it is not sufficient for proving equivalence. The following examples show some pitfalls of the method.

## 3. Examples

We consider two circuits that implement the same Boolean function of four variables:

$$C1 = C2 = x1x3x4 + x2\overline{x3} + x2\overline{x4} \quad (1)$$

These are shown in Figures 3 and 4. The circuit C1 is a minimal multi-level implementation and $C2$ was obtained by a specialized exclusive-OR transform technique [3].

Complete checkpoint fault tests were generated for the two circuits by the gate-level test generation program, Gentest [2]. The two test sets, expressed in Figure 5 by the shaded minterms, were quite different. When the two test sets were concatenated for simulating the setup of Figure 2, the output $z$ was always 0 as expected. Many modifications of $C2$ were attempted in which the function was changed. All of those, except one, changed the output $z$ to 1 for one or more vectors. The exceptional circuit $C2'$ is shown in Figure 6. The circuit $C2'$ was obtained by replacing the first exclusive-OR gate by an OR gate. Its checkpoint tests are shown in Figure 7. When the circuit of Figure 2 is simulated for $C1$ and $C2'$, we find a failure of our heuristic. The output $z$ remains 0 for all vectors. although the two circuits functionally differ

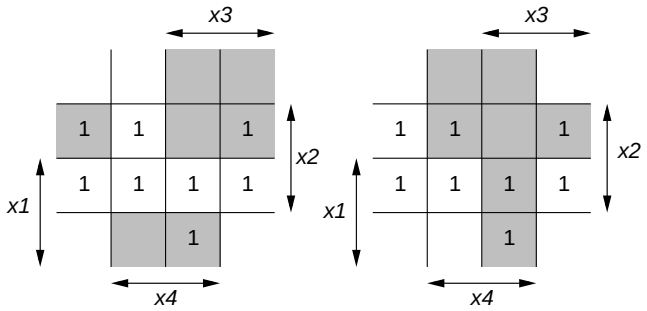



Figure 5: Checkpoint test sets for $C1$ and $C2$. Shaded minterms are tests.

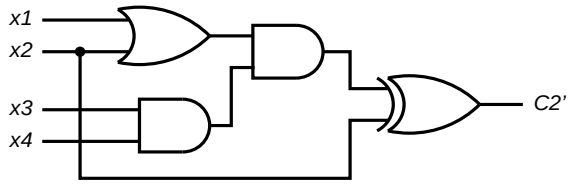



Figure 6: Circuit $C2'$ with first exclusive-OR in $C2$ replaced by OR

in the minterm $x1x2x3x4$. This minterm, marked with a cross in Figure 7, was included neither in the test set for $C1$ nor in that for $C2'$.

## 4. Use of Fault Simulation

Fault simulation is normally not used in logic verification or equivalence checking. Invoking the Observation B of Section 2, we find that fault simulation can be useful. The observation is formally stated as follows:

**Theorem 4.1** *In the equivalence checking circuit of Figure 1, if circuits $C1$ and $C2$ are equivalent, then no single or multiple stuck-at fault on primary input lines is testable.*

**Proof:** Suppose $C1$ and $C2$ are equivalent. Then their truth tables must be identical. Any single or multiple stuck-at fault on PI lines converts the input vector $V$ into $V'$, where $V'$ is obtained by changing some bits of $V$ according to the fault. Since the fault is assumed to occur before PI's fanout to $C1$ and $C2$, the same vector $V'$ is applied to both circuits. Having the same truth table, both circuits will produce an identical output, which can be either same as or different from that for $V$. The output of the exclusive-OR gate will therefore remain 0, as will be the case if no fault were present. Thus, the fault cannot be detected. ■

This result was observed in fault simulation of several circuits including those of examples in Section 3. Fault

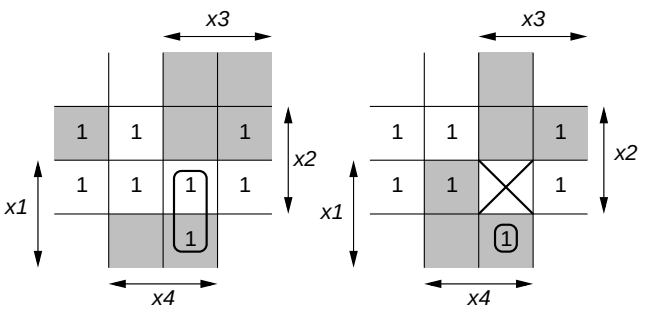



Figure 7: Checkpoint tests for $C1$ and $C2'$

simulations were run for the collapsed set of single stuck-at faults in the circuit of Figure 2. For all cases where $C1$ and $C2$ were equivalent circuits, 11 faults were not detected: 8 faults on four PI's, s-a-0 faults on $z1$ and $z2$, and s-a-0 on $z$.

When the circuits of Figures 3 and 6 were simulated for comparison, even though a constant output of $z = 0$ was observed and s-a-0 faults at $z1$, $z2$ and $z$ were not detected, three faults, s-a-1 at $x1$, $x2$ and $x3$ were detected. This shows how Theorem 4.1 allows us to decide the non-equivalence of the two circuits.

The last example illustrates the strength of the fault simulation method. Notice that logic verification based on true-value simulation can only prove the two circuits to be non-equivalent if at least one vector in the input set produces different outputs. Search for such vectors, when the two circuits are almost identical, can be very difficult. Fault simulation can establish non-equivalence even when the vector that produces different outputs is not available. In fact, it is only necessary to simulate faults on primary inputs of the circuit in Figure 1. The effect of fault simulation is that besides checking the equivalence for the simulated vectors, we also check equivalence for all vectors that are at unit Hamming distance from the simulated vectors. Unit distance is used because we assume single stuck-at faults. Multiple faults will correspond to larger Hamming distance. In checking for equivalence between the circuits of Figures 3 and 6, the PI s-a-1 faults on $x1$, $x2$ and $x3$ were detected by three test vectors (grey shaded up, down and left neighbors of the error minterm marked with cross) in Figure 7.

### 4.1. Target Faults

The four-point analysis of Section 2 indicates that when we simulate the equivalence checking circuit of Figure 1 or 2, all internal faults of $C1$ and $C2$ must be detectable, irrespective of whether the two circuits are equivalent or different. Thus, fault simulation of internal faults pro-

vides no information about equivalence. Nevertheless, the use of vectors that can detect all internal faults is a good, though incomplete, heuristic.

Simulation of faults on primary inputs of the equivalence checking circuit provides additional information about equivalence. This is because the simulation of a fault requires an implicit simulation of two vectors. If we restrict to the simulation of single stuck-at faults, since that is easily done by the available fault simulators, non-equivalence can be effectively uncovered in many cases.

## 4.2. Fault Simulation Vectors

Consider fault simulation of the equivalence checking circuit (Figure 1 or 2). The circuit has $n$ primary inputs and only $2n$ single stuck-at faults on these are simulated. For a given vector $V1$, we effectively evaluate the output for $V1$ and $n$ other vectors that are at unit Hamming distance from $V1$. An input fault is found detectable (showing non-equivalence) only if the outputs of $C1$ and $C2$ were to differ. Having evaluated the circuit for these $n+1$ vectors, we should then select the next input vector that is at a Hamming distance 3 from $V1$. Similarly, the next vector $V3$ should be at Hamming distance 3 from both $V1$ and $V2$.

In a different context, covering the $n$ dimensional [0,1] space with vectors that are a minimum Hamming distance 3 apart is similar to finding an $n$ bit code with single bit error correction (or double bit error detection) capability. The number of code words in such a code is given by [4]:

$$Number\ of\ code\ words \leq \frac{2^n}{n+1} \qquad (2)$$

where $2^n$ is the total number of points in the $n$ dimensional binary code space. It is also the number of vectors in our vector space. A code word (vector) and its $n$ unit-distance neighbors form a sphere of volume $n+1$. In order to satisfy the minimum Hamming distance requirement, actual codes leave out some points when the space is not fully covered by non-overlapping spheres of radius 1. In our case, the requirement is on *maximum* Hamming distance. Therefore, the number of vectors will be generally larger than that given by the above relation. Thus,

$$Number\ of\ vectors \geq \frac{2^n}{n+1} \qquad (3)$$

**Example:** For $n = 4$, we obtain a set of four vectors: 0000, 0111, 1011, 1100. These are shown in Figure 8 as shaded minterms. Notice that some distances between these vectors are less than 3. This is because $n = 4$ does not permit a *perfect* code with distance 3. In a perfect code all distance-3 neighbors of a code word will also be acceptable code words [5] and the relation 2, known as the *Hamming bound* becomes an equality. Perfect codes exist for very few combinations of length $n$ and distance (3 here). In those cases where perfect codes exist, relation 3 will also be an equality. For the message coding problem, the Hamming distance between codes must not be less than 3 and one would use fewer codes, strictly following the relation 2. We have more vectors because we must not allow a Hamming distance greater than 3, but smaller distances are acceptable. Notice that every vector in the entire space is within the distance 1 from some selected vector. Thus, simulation of single stuck-at faults will actually examine the entire space. Several such sets are possible. ■

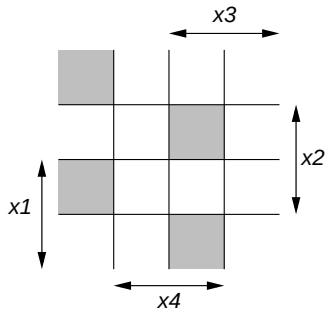



Figure 8: Four-bit vectors with Hamming distance $\leq 3$.

Fault simulation with these four vectors correctly checks the equivalence for the circuits discussed in Section 3. We can prove the following result:

**Theorem 4.2** *Consider the equivalence checking circuit of Figure 1 being simulated with a vector set such that every vector in the input space is within a unit Hamming distance from some vector in the set. If the output $z$ remains 0 and no single stuck-at fault on primary inputs is detected by the vector set, then the circuits $C1$ and $C2$ are equivalent.*

**Proof:** Simulation of primary input single stuck-at faults with a vector $V$ means that the output $z$ must be computed for $V$ and $n$ other vectors at a unit Hamming distance from $V$. It is given that $z = 0$ when $V$ is applied. That is, $C1$ and $C2$ agree on $V$. Each of the $n$ neighboring vectors represents the transformation of $V$ by a single stuck-at fault on a PI. Only when a neighboring vector produces identical response from $C1$ and $C2$, will the corresponding fault remain undetected. If $V$ does not detect any PI stuck-at fault, then the equivalence of $C1$ and $C2$ is checked for $V$ and its unit distance neighbors. Since the vectors in the set and their unit distance neighbors cover the entire vector space, after the

simulation of the vector set if always $z = 0$ and no input fault is detected, then the two circuits must have agreed on all vectors in the space. ■

An important contribution of Theorem 4.2 is that it requires the true-value simulation of the circuit. Additionally, only the faults on primary input lines of the circuit of Figure 1 need be simulated.

The use of maximum Hamming distance vectors for random testing and methods of generating such vectors have been proposed by Wu *et al.* [13]. In their application, the main interest was the overall fault coverage, which included the internal faults of the circuit. In the present application, we are interested in covering the entire vector space using the concurrent simulation capability of a fault simulator.

There are existing algorithms for finding codes with given minimum Hamming distance. For our application, however, the existing coding theory algorithms [10, 11] will require modification because we need a *set of vectors with maximum Hamming distance of 3 to cover the entire vector space.* The vectors in the preceding example were manually obtained.

Recent methods provide efficient simulation of multiple stuck-at faults [9, 12]. If multiple stuck-at faults on the PI's of the equivalence checking circuit are simulated, then the number of vectors to be simulated can be reduced. This is because multiple fault detection will cover a larger distance around the vector being simulated. Thus, vector complexity will be traded down with with the increased complexity of multiple fault simulation.

## 5. Summary of Contributions

This paper proposes the following procedures for logic verification:

- *Checkpoint tests.* Tests that cover the checkpoint faults in both circuits can uncover many differences in the circuits. Although not investigated here, these tests may allow diagnosis of observed differences. This is a good strategy but, as shown, can fail.

- *Fault simulation.* Fault simulation, especially for PI faults of the equivalence checking circuit can discover differences in circuits even when logic simulation does not give different outputs.

- *Vectors for fault simulation.* Simulation of PI faults of the equivalence checking circuit can prove the logic equivalence of the two circuits when a complete set of vectors with maximum Hamming distance of 3 is used.

## 6. Conclusion

A proper selection of vectors can improve the debugging capability of simulation-based verification process. The potential of checkpoint tests for diagnostics should be explored. Algorithms for finding the minimal vector sets with maximum Hamming distance 3 are needed. Finally, complexity trade-offs between reduced vector set for larger Hamming distance and multiple fault simulation may be examined.

## References


[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* Piscataway, New Jersey: IEEE Press, 1995. Revised printing.

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

[3] P. Chavda, J. Jacob, and V. D. Agrawal, "Optimizing Logic Design using Boolean Transforms," in *Proc. 11th International Conf. VLSI Design*, January 1998, pp. 218–221.

[4] R. W. Hamming, *Coding and Information Theory.* Englewood Cliffs, New Jersey: Prentice-Hall, 1980.

[5] D. G. Hoffman, D. A. Leonard, C. C. Linder, K. T. Phelps, C. A. Rodger, and J. R. Wall, *Coding Theory: The Essentials.* New York: Marcel Dekker, 1991.

[6] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging.* Boston: Kluwer Academic Publishers, 1998.

[7] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques.* Boston: Kluwer Academic Publishers, 1997.

[8] R. P. Kurshan, *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton, New Jersey: Princeton University Press, 1994.

[9] K. P. Lentz, *Multiple-Domain Comparative and Concurrent Simulation (MDCCS).* PhD thesis, Northeastern Univ., Boston, 1994.

[10] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes.* Amsterdam: North-Holland, 1977.

[11] S. M. Reddy, "On Block Codes with Specified Maximum Distance," *IEEE Trans. Information Theory*, vol. IT-18, pp. 823–824, 1972.

[12] E. G. Ulrich, V. D. Agrawal, and J. H. Arabian, *Concurrent and Comparative Discrete Event Simulation.* Boston: Kluwer Academic Publishers, 1994.

[13] S. H. Wu, Y. K. Malaiya, and A. P. Jayasumana, "Anti-random Testing: Beyond Random Testing," Computer Science Technical Report CS-98-103, Colorado State Univ., Fort Collins, Colorado, 1998.