Design Error Diagnosis and Correction Via Test Vector Simulation

Andreas Veneris, *Student Member, IEEE*, and Ibrahim N. Hajj, *Fellow, IEEE*

***Abstract*—With the increase in the complexity of digital VLSI circuit design, logic design errors can occur during synthesis. In this paper, we present a test vector simulation-based approach for multiple design error diagnosis and correction. Diagnosis is performed through an implicit enumeration of the erroneous lines in an effort to avoid the exponential explosion of the error space as the number of errors increases. Resynthesis during correction is as little as possible so that most of the engineering effort invested in the design is preserved. Since both steps are based on test vector simulation, the proposed approach is applicable to circuits with no global binary decision diagram representation. Experiments on ISCAS'85 benchmark circuits exhibit the robustness and error resolution of the proposed methodology. Experiments also indicate that test vector simulation is indeed an attractive technique for multiple design error diagnosis and correction in digital VLSI circuits.**

***Index Terms*—Correction, design error, diagnosis, simulation.**

I. Introduction

DURING the design cycle of a VLSI digital circuit, functional mismatches between the specification and the gate-level implementation (design) can occur. These functional mismatches, also known as *design errors*, usually involve the functional misbehavior of some gate elements and/or some wire interconnection errors. A common source of these errors is the manual interference of the designer with the design during the synthesis process in order to achieve specific optimization goals [1]. Errors at a higher level of the design flow and software bugs in automated design tools can also translate to design errors in a netlist [33].

Once a verification tool finds that a design does not agree with its specification, *design error diagnosis and correction* (DEDC) is performed. Diagnosis attempts to identify lines in the design that may have a design error. The quality of diagnosis is determined by its ability to narrow the space of potential erroneous lines, that is, its error resolution. Once a set of potential erroneous lines has been identified, correction is performed. The goal of correction is to suggest appropriate modifications to the netlist that make it functionally equivalent to the specification. The quality of correction depends on the nature and number of the proposed modifications. Resynthesis should be minimal so that most of the engineering effort invested in the design is preserved.

Usually, design errors are assumed to belong to a small predetermined set of possible error types, known as the *design error* or *correction model*. Abadir *et al.* [2] presented such a model, shown in Fig. 2, that consists of ten distinct types of errors. In the same work [2], it is proven that a complete set of stuck-at fault vectors for the erroneous design guarantees to detect the majority of these errors (Types A–G) and has a good chance to detect the remaining ones (Types H–J). A theorem is also presented in [2], proving that a complete single-stuck-line fault test set can always detect the substitution of a gate for a unate function. An automatic test-pattern generation (ATPG) simulation-based design error verification method for the errors in [2] is developed by Asaad and Hayes [4]. The fault coverages of the test sets derived by them validate the theoretical results of Abadir *et al.* [2] and show that sometimes design errors are "hard to detect."

The experiments carried by Aas *et al.* [1] confirm that the design error model of [2] is indeed a realistic one, as it covers 97.8% of the errors that usually occur during a manual resynthesis procedure. These experiments [1] also show that the average number of design errors is usually less than or equal to two. For these reasons, and because of its simplicity, the model of [2] has been used by the majority of the existing literature for DEDC. In this paper, we define and use a design error model that is a simple extension of the one presented in [2].

In our treatment of the DEDC problem, diagnosis and correction is formulated around the number and nature of the modifications required to correct the design and not around the number of the actual design errors. Since there may be more than one way to synthesize a particular function, there may be equivalent corrections other than the actual correction that rectify an erroneous design [10]. Throughout this paper, we will refer to the actual and any equivalent set of corrections, as *valid* corrections. The terms *modification*, *correction*, and *design error* will also be used interchangeably.

In addition, we assume that the only netlist available as an input to the problem is the incorrect gate-level implementation. The specification can only provide the (correct) primary output responses for a primary input stimulus. For example, the specification might be available in a register-transfer-level format, coded in some hardware-description language and no line naming equivalence with the netlist is available.

Manuscript received March 6, 1999; revised July 13, 1999. This work was supported by the Joint Services Electronics Program under Contract N00014-96-1-0129. This paper was recommended by Associate Editor L. Stok.

A. Veneris was with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. He is now with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont. M5S 3G4 Canada.

I. N. Hajj is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA.

Publisher Item Identifier S 0278-0070(99)09917-0.

It should be noted that the space of potential erroneous lines grows exponentially in the number of errors [26]

$$\text{error line space} = (\#\text{of circuit lines})^{(\#\text{of errors})}. \quad (1)$$

It can also be seen that the correction space for a design where modifications are selected from the model of Abadir *et al.* [2] is lower bounded by the error line space.

In this paper, we describe a test vector simulation based methodology for combinational circuit DEDC. The contribution of this work is twofold. First, we present a test vector simulation based methodology for multiple DEDC. The novelty of the proposed work compared to previous approaches lies in the fact that, in practice, it avoids the exponential explosion of the error space according to (1) and remains computationally efficient as the number of errors increases. In addition, unlike previous methods, correction is also based on the results of test vector simulation. Therefore, since both steps of diagnosis and correction are based on information provided by test vector simulation, the method is applicable to circuits that have no global Binary Decision Diagram (BDD) representation [6], [7], [23].

Next, we examine the quality of test vector simulation for the DEDC problem. Since exhaustive test vector simulation is prohibitive for most circuits, it is of interest to know the quality of a DEDC method that bases its results on a small subset of the complete input test vector space. Our experiments suggest that test vector simulation is an attractive approach for this problem, but they also mandate the use of a logic verifier to guarantee the correct functionality of the final design.

This paper is organized as follows. In the remainder of this section, we discuss previous work and outline the steps of our approach. Section II contains relevant definitions and a description of the design error model that we use. Diagnosis and correction are presented in Sections III and IV, respectively. Experimental results and a discussion on the quality of test vector simulation for the DEDC problem can be found in Section V. Section VI reviews related research problems, and Section VII concludes this paper.

A. Previous Work

A number of approaches have been developed for the DEDC problem. These approaches can be divided into two categories with respect to the underlying technique used for error location and error correction: those based on *Boolean function manipulation (symbolic)* techniques [8]–[11], [16]–[18], [24], [27], [32] and those based on *test vector simulation* [13]–[15], [21], [22], [25], [26], [28], [29], [33], [34].

Techniques based on Boolean function manipulation have the advantage that they can return valid corrections, if such corrections exist in the design error model they use. They also have good error resolution, and they are computationally efficient for single errors. Nevertheless, their performance degrades as the number of design errors increases. For this reason, heuristics are usually employed during diagnosis, certain subclasses of errors from the model of Abadir *et al.* [2] are only considered, or the amount of resynthesis may not be minimal. More important, symbolic techniques that use a global BDD [6] representation of the circuit are not applicable to circuits that have no such efficient representation [6], [7], [23].

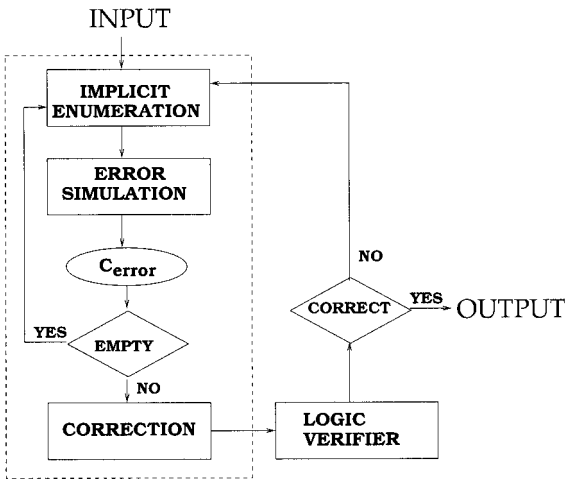


Fig. 1. Overview of DEDC methodology.

On the other hand, test vector simulation-based methods for DEDC are applicable to all circuits. In addition, existing work shows that test vector simulation can be a computationally efficient route to DEDC for designs corrupted with one and two errors. However, just like the symbolic methods, their performance decreases as the number of errors increases, and little work [26], [28], [29] has been performed on design error correction for the complete error model of Abadir *et al.* [2].

B. Work Overview

In this section, we outline the steps of our DEDC methodology. For diagnosis, the method performs an *implicit enumeration* of error lines in an effort to avoid the exponential explosion of the error space (1) and remain computationally efficient. In detail, unlike most previous test vector simulation-based approaches [13]–[15], [22], [25], [26], [28], [33], [34], it does not attempt to explicitly compute the complete error space and eliminate areas that cannot contain an error(s). Instead, it samples and searches a small area of the error space for error candidates. At the same time, correction uses an extension of the design error model of Abadir *et al.* [2], and the amount of resynthesis is as little as possible. Although theoretically the proposed approach may compute the error space according to (1), our experiments show that this never happens in practice, and it remains computationally efficient as the number of errors increases.

Our DEDC method is shown in Fig. 1. The *input* to the algorithm is the functional specification F_C, the erroneous gate-level description G_C, an initial guess for the number of required modifications N, a set of stuck-at [19] and random input test vectors $\mathcal{V}_{\text{test}}$, and a set of vectors $\mathcal{V}_{\text{act}} \subseteq \mathcal{V}_{\text{test}}$, such that each of them activates the inconsistencies, i.e., it produces at least one primary output response in F_C that is different from G_C. In our experiments, we start the method with $N = 1$.

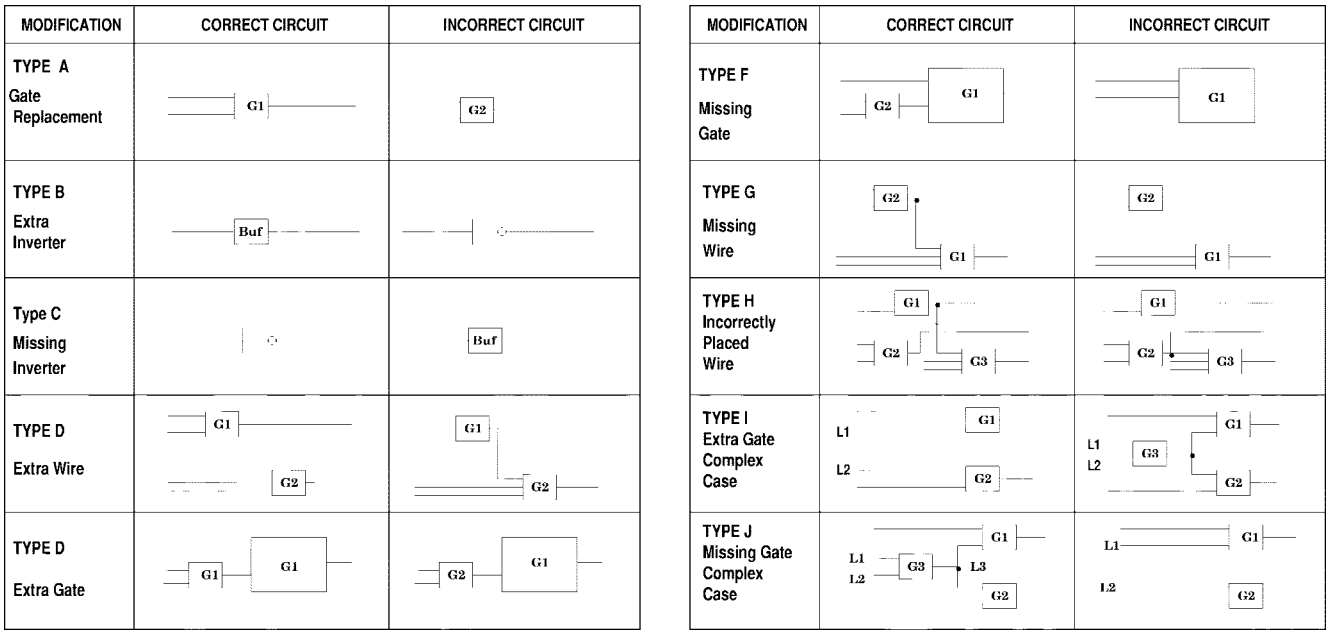


Fig. 2. Design error model of Abadir *et al.* [2].

$\mathcal{V}_{\text{act}}$ is compiled during a test vector simulation-based verification step that precedes our DEDC method. During that step we simulate F_C and G_C with all vectors from $\mathcal{V}_{\text{test}}$. $\mathcal{V}_{\text{test}}$ is also used during correction, while $\mathcal{V}_{\text{act}}$ gives information for diagnosis. In our experiments, $\mathcal{V}_{\text{test}}$ is usually less than 15 000 vectors for the ISCAS'85 benchmark circuits, and the average size of $\mathcal{V}_{\text{act}}$ fluctuates between 40 and 300 vectors.

As mentioned earlier, the method performs a search of the error space to identify potential erroneous signals. This search is performed in two steps (Fig. 1). First, information on candidate error lines is collected through the path-trace procedure [30], [31] to form a graph. A novel graph processing procedure directs the search for potential error lines without necessarily computing the complete error space. Second, error simulation is performed to improve the error resolution and to output a set of candidate error lines C_{error}. If C_{error} is the empty set, then diagnosis is repeated for different internal parameter values, as explained in Section II-E.

When diagnosis returns with a nonempty C_{error}, the method proceeds with correction, and a logic verifier [6] is used to *output* valid corrections. If this set of corrections is empty, then the procedure is repeated until it returns with success or a maximum number of iterations is reached, and the algorithm is repeated for a larger value of N.

II. Preliminaries

In this work, we examine incorrect combinational netlists G_C with *simple* logic NOT, BUFFER, AND, NAND, OR and NOR gates. Although our algorithm can handle XOR and XNOR gates, we do not run experiments on such circuits because these errors are difficult to detect with stuck-at fault test vector sets [2]. Our experiments also suggest that random test vector simulation may not detect all errors on XOR and XNOR gates. If such gates are present in the circuit, one may want to include in $\mathcal{V}_{\text{test}}$ a design error-specific test set such as the one developed by Asaad and Hayes [4].

During the execution of our algorithm, we introduce one buffer for every fan-out line of a branch. We also assume that both F_C and G_C are completely simulatable, that is, $\mathcal{V}_{\text{test}}$ contains test vectors with specified logic values zero and one only. This assumption is relaxed in Section III-F.

A line l, fan-in to an AND or NAND (OR or NOR) gate, is said to have *controlling value* for input vector v if the value of l is zero (one). If l drives a NOT or a BUFFER, it always has controlling value. A line whose value changes during simulation under the presence of some fault(s) is called a *sensitized line*, and a path of sensitized lines is called a *sensitized path*.

A. Design Correctability

The design error model proposed by Abadir *et al.* [2], shown in Fig. 2, contains ten different cases of possible design error types. These errors can be classified as *wrong gate* errors (Types A, B, C, E, F, I, and J) and *wrong wire* errors (Types D, G, and H). In our design error model, which is a simple *extension* of [2], a *modification* can be either a wrong gate, a wrong wire, or a *wrong gate/wrong wire*, that is, an occurrence of both types of errors [2] on the gate driving a single line. For example, a gate-replacement/missing wire error can be an error where a four fan-in OR gate is replaced with a NAND gate with only three of the four original fan-ins, etc.

Throughout our presentation, we use the following terminology. An *N-error line tuple* $L = \{l_1, l_2, \ldots, l_N\}, N \geq 1$, is a set of N distinct lines in the netlist G_C. If, for every $l_i \in L$, there exists a correction c_i from *some* design error model that can rectify the design, then we say that L is a *valid* N-error line tuple and G_C is *N-source correctable*. If every such c_i belongs to our design error model, then we say that $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ is a valid *N-correction tuple* for L.

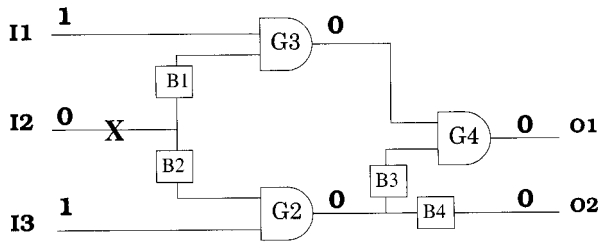


Fig. 3. A one-source correctable design.

The diagnosis approach, found in Section III, returns results for N-source correctable designs as it tries to identify valid N-error line tuples. Therefore, diagnosis is independent of any design error model, and it can also be used for macro-based circuits, where a macro is defined as a logic block with multiple inputs and one output that implements some boolean function [22].

III. Error Diagnosis

In this section, we describe our diagnosis procedure. Diagnosis, as shown in Fig. 1, consists of two steps: *implicit error space enumeration* and *error simulation*.

During implicit error space enumeration, an initial estimate C_{error} of the error space in terms of N-error lines tuples is obtained. Implicit enumeration relies on the path-trace procedure [30], [31], presented next, that marks suspicious circuit lines. Path-trace alone can diagnose designs with single errors. For multiple errors, it is used to construct a graph that contains information about the error space. A graph processing procedure, presented in Section III-C1, obtains the estimate C_{error} that is further screened by error simulation.

A. The Path-Trace Procedure

In this section we review the *path-trace procedure*, a line marking algorithm developed for fault diagnosis by Venkataraman *et al.* [30], [31] that is based on the critical path-tracing algorithm [3].

Let vector $v \in \mathcal{V}_{\text{act}}$. Path trace [30], [31] starts from an *erroneous* primary output for v and traces backward toward the primary inputs marking lines as follows: if the output of a gate G has been marked and G has one or more fan-in(s) with controlling values, then the procedure randomly marks any one controlling fan-in; if G has all fan-ins with noncontrolling inputs, then all fan-ins are marked; if a branch is marked, then the algorithm automatically marks the stem of the branch.

For example, consider the circuit of Fig. 3 with primary outputs O_1 and O_2. If O_1 has been marked by path trace, it can proceed by marking set of lines $S_1 = \{O_1, G_4, G_3, B_1, I_2\}$. If path trace begins from O_2, a set of lines $S_2 = \{O_2, B_4, G_2, B_2, I_2\}$ are marked.

Define V_j^i to be the set of lines marked by path trace when tracing from erroneous primary output PO_i and vector $v_j \in \mathcal{V}_{\text{act}}$ [31]. The following theorem, together with Theorem 2 presented later, is crucial for the correctness of our diagnosis algorithm.

Theorem 1: Let G_C be an N-source correctable design and $L = \{l_1, l_2, \ldots, l_N\}$ be *any* valid N-error line tuple. If v_j is a vector that activates the inconsistencies, and PO_i is an erroneous primary output for v_j, then V_j^i contains at least one element from L.

Proof: Let L' be any minimal subset of lines of L so that when their values get complemented for v_j and this difference is propagated at the primary outputs, PO_i has a correct response. Clearly, by definition of L', there is one or more sensitized paths from each member of L' to PO_i. We claim that V_j^i contains some element $l \in L'$. Proving this claim completes the proof.

Let $V_{j,n}^i \subseteq V_j^i$ be the set of lines marked by path trace during the nth iteration of the algorithm. For example, for the circuit of Fig. 3, $V_{v,1}^{O_1} = \{O_1\}$ and $V_{v,3}^{O_1} = \{G_3\}$. We use induction to show that for *every* n, $V_{j,n}^i$ contains some line on a sensitized path from $l \in L'$ to PO_i. This is sufficient to prove the claim, as the algorithm will eventually mark l in $V_{j,n}^i$ for some $n \leq$ *max circuit level*.

For the base case $n = 1$, the argument holds as path trace correctly marks the erroneous primary output PO_i. Assume that it holds for n steps, that is, $V_{j,n}^i$ contains line l' on a sensitized path from l to PO_i. To prove that it holds for the next iteration, observe that if $l' \in V_{j,n}^i$ is a branch, then the stem, which has to be on a sensitized path, will automatically be included in $V_{j,n+1}^i$ by the algorithm. If l' is a fan-out of a gate with multiple controlling fan-ins, then all such fan-ins (also marked by path trace) need to get complemented so that the value of l' changes. Thus, every such fan-in will be on some sensitized path(s) from elements of L' and induction holds for $V_{j,n+1}^i$.

Similar reasoning proves that induction holds for the case where l' is a fan-out of a gate with all noncontrolling fan-ins. This proves the claim and completes the proof.

B. Single Error Case

Theorem 1 directly translates into a diagnosis algorithm for designs corrupted by a single error because every valid error line l is guaranteed to be marked by every run of the path-trace procedure for different erroneous primary outputs and vectors of the set $\mathcal{V}_{\text{act}}$. In other words, every such line l is guaranteed to be in the intersection of the lines marked by distinct path-trace runs.

The following corollary, immediate from Theorem 1, formalizes the above observation.

Corollary 1: If G_C is a one-source correctable design, then for every valid one-error line tuple l

$$l \in \bigcap_{PO_i \text{ erroneous for } v_j} V_j^i, \qquad j = 1, \ldots, |\mathcal{V}_{\text{act}}|. \tag{2}$$

Again, consider the circuit of Fig. 3 and assume that there is an inverter missing on line I_2. In such case, input vector (1, 0, 1) produces erroneous responses at both primary outputs O_1 and O_2. As explained earlier, two potential runs of path trace from O_1 and O_2 mark sets of lines S_1 and S_2, respectively. Corollary 1 asserts that every single valid error line belongs in their intersection, that is, $S_1 \cap S_2 = \{I_2\}$.

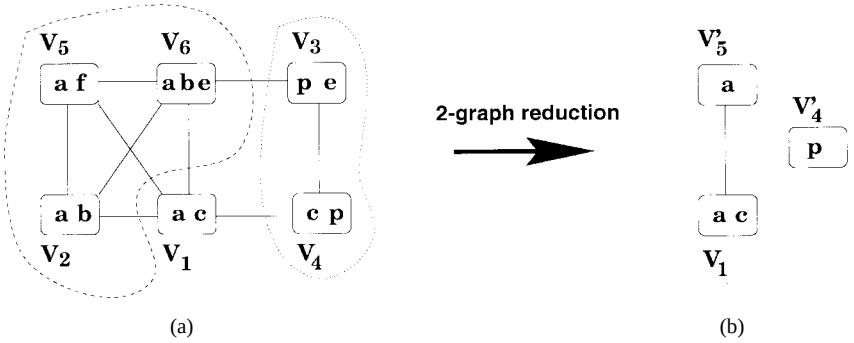


Fig. 4. Examples of intersection graphs.

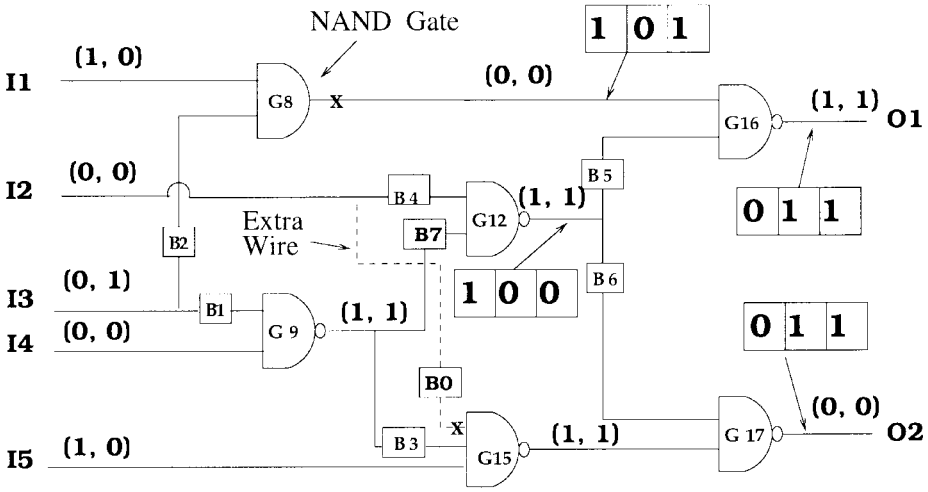


Fig. 5. A two-source correctable design.

C. Multiple Error Case

If the design G_C is corrupted by multiple errors, that is, $N>1$, different V_i^j sets will contain different lines from valid error line tuples, according to Theorem 1, and Corollary 1 no longer holds. In this subsection, we present the concept of the intersection graph, originally developed by Venkataraman and Fuchs [31] for bridging fault diagnosis. We introduce the operation of an N-graph reduction that allows the processing of the graph in order to prune the error space. We also describe a novel enumerating procedure that computes N-error line tuples from a processed graph.

1) Pruning the Error Space: The *intersection graph* (IG) [31] $G=(V,E)$ of a G_C is an undirected graph where each vertex $V_i \in V$ contains a set of lines from G_C. Edge $(V_i,V_j) \in E$ if and only if $V_i \cap V_j \neq \emptyset$. Throughout our presentation, we use the symbol V_i to denote either the vertex V_i or the set of lines vertex V_i contains, depending on the context in which the symbol is used.

For example, the IG in Fig. 4(a) has six vertices, $V_1, V_2, \ldots, V_6$, and the IG in Fig. 4(b) has three vertices and two cliques.

Definition 1: Let IG $G=(V,E)$ and let $V_R=\{V_1,V_2,\ldots,V_N\}$, $N>1$, be N distinct vertices of G with pairwise disjoint line sets. For every $V_i \in V_R$, let $V_i^{\text{adj}}=\{V_{i_1},V_{i_2},\ldots,V_{i_k}\}$ denote a maximum set of vertices that form a clique that contains V_i, and no vertex of this clique is adjacent to any vertex of $V_R-\{V_i\}$. We define an *N-graph reduction* with respect to V_R on G to be the operation that gives a new IG $G'=(V',E')$ where $\forall i, 1 \leq i \leq N$, set V_i^{adj} is replaced by a new vertex V_i', $V_i'=V_{i_1} \cap V_{i_2} \cap \ldots \cap V_{i_k}$ and recompute edge adjacencies. If an N-graph reduction exists, then we say that G is *N-reducible*.

It will be seen shortly that N-graph reductions provide the means for pruning the error space. An example of a graph reduction operation is shown in Fig. 4. The IG of Fig. 4(b) is the resulting graph when a two-graph reduction, with respect to $V_R=\{V_5,V_4\}$, is carried on Fig. 4(a). The two sets of vertex adjacencies involved in the reduction are shown in dotted lines. Observe that $\{V_1\} \notin V_5^{\text{adj}}=\{V_2,V_5,V_6\}$ because it is adjacent to $V_4 \in V_R-\{V_5\}$. Notice that the reduced graph of Fig. 4(b) is still two-reducible.

The *graph processing* proceeds as follows. Initially, the IG has no vertices. We begin adding vertices to the IG from distinct runs of the path-trace procedure and perform an N-graph reduction(s) whenever possible. This procedure terminates when path trace is called for all vectors in $\mathcal{V}_{\text{act}}$ and all respective erroneous primary outputs.

Example 1 illustrates the graph processing procedure for a two-source correctable design.

Example 1: Consider the erroneous circuit in Fig. 5 with a gate replacement error on G_8 (NAND gate) and an extra wire B_0 (dotted line) simulated for input vectors $v_1=(1,0,0,0,1)$

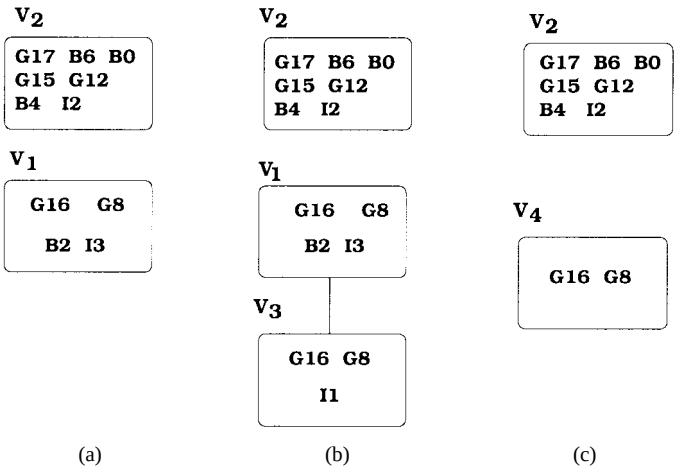


Fig. 6. IG processing.

and $v_2 = (0,0,1,0,0)$. Line values during simulation are shown in parentheses. v_1 produces erroneous results at both primary outputs and v_2 activates the inconsistencies at output driven by G_{16}.

For the first input vector, the path-trace procedure marks set of lines $V_1 = \{G_{16}, G_8, B_2, I_3\}$ when it originates from G_{16} and it marks a set of lines $V_2 = \{G_{17}, B_6, B_0, G_{15}, G_{12}, B_4, I_2\}$ when it originates from G_{17}. The resulting intersection graph is shown in Fig. 6(a). Fig. 6(b) shows the intersection graph, and its adjacencies when vertex V_3 is added from path trace for v_2 starting at G_{16}. This allows a two-graph reduction with respect to $V_{R_1} = \{V_1, V_2\}$. The resulting graph is shown in Fig. 6(c) where vertices V_1 and V_3 have been replaced by V_4 with line set $V_4 = V_1 \cap V_3$.

Theorem 2 below guarantees that the graph processing procedure described above will not jeopardize the error resolution as long as N-graph reductions are applied on the IG of an N-source correctable design. Before we state and prove the theorem, we prove the following lemma.

Lemma 1: Let G_C be an N-source correctable design and $G = (V, E)$ be an IG at some stage of the graph processing procedure. If every vertex of G contains at least one line from each valid N-error line tuple, this property is maintained after an arbitrary number of vertex additions from path trace. If G has at most N pairwise nonadjacent vertices, this property is maintained after an arbitrary number of vertex additions from path trace.

Proof: The first claim of the lemma, that is, that every vertex of G contains a line from every valid N-error line tuple after a number of vertex additions from path trace, is a straightforward application of an inductive argument and Theorem 1 on the number of vertices of G.

The second claim of the lemma is proved by contradiction. Suppose that there is a set V_{big} of $N + k, k > 0$, pairwise nonadjacent vertices after a vertex addition from path trace on G. By Theorem 1 and the first claim of this lemma, every vertex of the resulting graph contains at least one element from every valid N-error line tuple. Therefore, there should be at least two vertices of V_{big} that are adjacent, which is a contradiction.

Theorem 2: Let G_C be an N-source correctable design and $G = (V, E)$ be the initial IG for the design, that is, a graph with no vertices. At every stage of the graph processing, G contains at least one line from every valid N-error line tuple. In addition, G can have *at most* N vertices pairwise nonadjacent to each other.

Proof: We prove the theorem with induction on the number of reductions. In our proof, we let $G^i = (V^i, E^i)$ be the processed IG *before* the ith reduction.

For G^1, that is, the initial graph with some vertices from path trace and vertex adjacencies computed, the theorem holds as every vertex in V^1 contains at least one element of every valid N-error line tuple (Theorem 1). In addition, G^1 cannot contain a set V_{big} of $N + k, k > 0$, pairwise nonadjacent vertices due to an argument similar to the one presented in the proof of Lemma 1.

Assuming that the theorem holds for the IG G^n, we prove that it holds for G^{n+1}, the IG obtained after applying the nth graph reduction with respect to some set V_R. Observe that Lemma 1 guarantees the induction step is true for *any* arbitrary number of path-trace vertex additions on G^{n+1}.

To prove the first claim of the theorem, observe that due to the inductive hypothesis and because G_C is N-source correctable, each set $V_i^{\text{adj}}, V_i \in V_R$, contains the same line from each valid N-error line tuple, otherwise there would be two vertices $V_i, V_j \in V_R, i \neq j$, that are adjacent. Since every vertex of V_i^{adj} contains the same line from every valid N-error line tuple, this line should also appear in their intersection.

To prove the second part of the theorem, G^{n+1} cannot contain a set V_{big} of $N + k, k > 0$, pairwise nonadjacent vertices as it leads to a contradiction with a similar argument to the one presented in the proof of Lemma 1.

Observe that Theorem 2 also gives a *lower bound* on the number of modifications needed to rectify an erroneous G_C. If G, at some stage of graph processing, has N pairwise nonadjacent vertices, then the design is guaranteed not to be K-source correctable for $K < N$, due to Theorem 2. However, it can be M-source correctable for some $M > N$.

As noted earlier, Theorem 2 holds only when N-graph reductions are performed on an N-source correctable design. The following example shows that K-graph reductions on an N-source correctable design when $K < N$ can jeopardize error resolution.

Example 2: The IG of Fig. 7(a) is a processed IG for some three-source correctable design G_C. Assume that (A, B, C) and (X, Y, Z) are the only two valid error line triples for G_C. If we perform a two-graph reduction with respect to $V_R = \{V_2, V_4\}$, we obtain the graph of Fig. 7(b), where vertices $\{V_1, V_2\}$ and $\{V_4, V_5\}$ are replaced by V_6 and V_7, respectively. It can be seen that this new graph violates Theorem 2, as there are vertices that do not contain a line from every valid error line triple.

Corollary 2: If G is the processed graph for some N-source correctable design and we perform a K-graph reduction for $K < N$, then Theorem 2 no longer holds.

2) Implicit Error Enumeration: Given a processed IG $G = (V, E)$, the next step of the algorithm is to *implicitly enumerate*

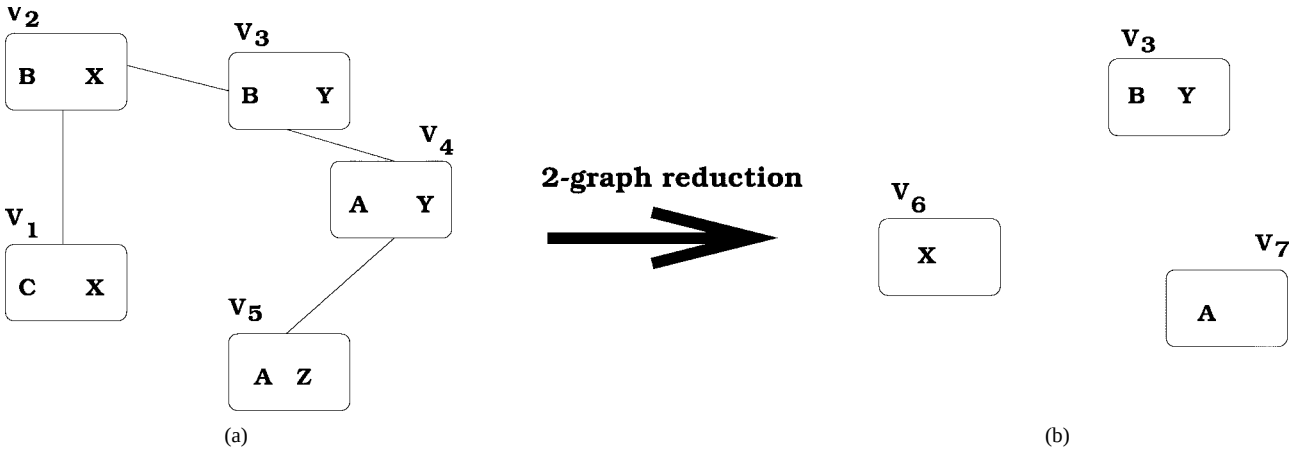


Fig. 7. A two-graph reduction on a three-source correctable design.

N-error line tuples from G. In this subsection, we describe such an enumeration procedure.

Let an *n-sample* of G, $0 \leq n \leq |V|$, be the union of lines of n randomly chosen vertices $\{V_1, V_2, \ldots, V_n\} \in V$, that is, n-sample $= \cup_{j=1}^{n} V_j$. For example, a two-sample for the graph of Fig. 7(a) may consist of vertices $V_4 \cup V_5 = \{A, Y, Z\}$ or $V_1 \cup V_5 = \{A, C, X, Z\}$.

During implicit enumeration, given G, the first step is to identify the maximum number $K, 1 \leq K \leq N$ of pairwise nonadjacent vertices. However, as explained earlier, G_C can be N-source correctable for any $N > K$. Therefore, an initial guess N of the number of modifications is required. In our experiments, its value is found by running the algorithm for increasing values of N. We start with $N = 1$ and increase its value as long as implicit enumeration fails to return a nonempty C_{error} (Fig. 1).

Once the algorithm computes K, it selects a set $\{V_1, V_2, \ldots V_K\} \subseteq V$ of pairwise nonadjacent vertices and an n-sample for a small value of n, usually less than five. Subsequently, it exhaustively compiles N-error line tuples $L = \{l_1, \ldots, l_K, l_{K+1}, \ldots, l_N\}$, placing them in C_{error}. The jth entry of each tuple is picked from V_j when $1 \leq j \leq K$, and from the n-sample when (and if) $K < j \leq N$. Last, for every other distinct set of K pairwise nonadjacent vertices $V' = \{V'_1, V'_2, \ldots, V'_K\}$ from G, the algorithm deletes the N-error line tuples from C_{error} that do not have a subset of K lines, each of them in the line set of some distinct $V'_i \in V'$.

Example 3: Implicit enumeration for the IG of Fig. 6(c) yields 14 error pairs as $K = N = 2$. These pairs are (G_{16}, G_{17}), (G_{16}, B_6), (G_{16}, B_0), (G_{16}, G_{15}), (G_{16}, G_{12}), (G_{16}, B_4), (G_{16}, I_2), (G_8, G_{17}), (G_8, B_6), (G_8, B_0), (G_8, G_{15}), (G_8, G_{12}), (G_8, B_4), and $(G_8, I2)$.

It can be seen that the ability of implicit enumeration to give valid N-error line tuples depends on the choice of the n-sample. Error simulation, described next, is a procedure that improves on error resolution. It also quantifies the quality of the chosen n-sample.

D. Error Simulation

Error Simulation is a diagnostic procedure that extends the ideas presented in [13] and [27]–[29] and also provides information useful during correction. It returns the *maximum* subset of C_{error} that can rectify the design for all vectors of $\mathcal{V}_{\text{act}}$. Therefore, if C_{error} does not contain valid N-error line tuples because of either a poor n-sample choice or an incorrect guess for N, at the end of error simulation it usually becomes empty.

```
ERROR_SIMULATION( F_C, G_C, l_1, l_2, ..., l_N)
1.   for every vector v_k ∈ V_act, 1 ≤ k ≤ |V_act|, do
2.       for i = 1 to 2^N − 1 do
3.           for j = 1 to N do
4.               if (2^(j−1) AND i) = 1 then
5.                   complement the k-th entry in l_j's Elist
6.           simulate at fan-out cones of l_1, l_2, ..., l_N
7.           if simulation gives correct primary output response then
8.               add (l_1, l_2, ..., l_N, i, v_k) in C_error
9.           restore Elist values at l_1, l_2, ..., l_N and fan-out cones
10.      if no simulation gives correct responses for v_k then
11.          delete all (l_1, l_2, ..., l_N) entries from C_error
12.          exit
```

Fig. 8. Error simulation.

During error simulation, shown in Fig. 8, for every N-error line tuple $L = \{l_1, l_2, \ldots, l_N\} \in C_{\text{error}}$ and for every $v \in V_{\text{act}}$ (line 1), we perform $2^N - 1$ simulations. During each such simulation for vector v, represented by a unique *error excitation scenario number* $m = 1, \ldots, 2^N - 1$, l_i maintains the original simulator value in G_C for v if the ith bit of m is zero, and has the complemented value if the ith bit of m is one (lines 3–6). The motivation is that a line l with a complemented simulation value for v indicates a line with a potential design error that is excited.

If there is a vector $v \in \mathcal{V}_{\text{act}}$ such that no error excitation scenario for L yields correct primary output responses, then L gets deleted from the error list as it cannot be a valid N-error line tuple. Otherwise, L qualifies and its N-error line tuple entry in C_{error} is updated with all excitation scenarios that give correct primary output results $\{m_1, m_2, \ldots, m_k\}$ for every $v \in \mathcal{V}_{\text{act}}$.

In our implementation, we maintain two bit lists at every line $l \in G_C$, the *Elist* and *Clist*. These bit lists are created during the initial test vector simulation verification step that precedes our DEDC method. The ith bit of the *Elist* for l contains the value of l when we simulate the ith vector in $\mathcal{V}_{\text{act}}$. For example, the *Elist* for the lines of the circuit in Fig. 5 is the

values in parenthesis. The bits of the *Clist* are defined similarly for the vectors of $\mathcal{V}_{\text{test}} - \mathcal{V}_{\text{act}}$. These bit lists help perform both error simulation and correction procedures efficiently.

Example 4: The boxes in Fig. 5 contain the values of the lines during error simulation for error line pair (G_8, G_{12}) and input vector $v_1 = (1,0,0,0,1)$. Recall that this vector gives erroneous responses at both primary outputs. The first entry contains the line values for error excitation scenario (G_8 = *excited*, G_{12} = *not excited*), the second entry contains the line values when (G_8 = *not excited*, G_{12} = *excited*), and the third one when (G_8 = *excited*, G_{12} = *excited*). Observe that this procedure needs to be carried out *only* in the set of six lines that is the union of the fan-out cones of (G_8, G_{12}). All three simulations can be performed in parallel with the use of the *Elist* bit lists. The first error simulation corrects O_1 for v_1 but maintains the erroneous response at O_2. The other two error simulations yield erroneous responses at O_1. Since none of the three error simulations yields correct primary output responses, (G_8, G_{12}) is deleted from C_{error}.

On the other hand, error pair (G_8, G_{15}) qualifies error simulation since error excitation scenarios (G_8 = *excited*, G_{15} = *excited*) for v_1 and (G_8 = *excited*, G_{15} = *not excited*) for v_2 yield correct primary output results and tuples $(G_8, G_{15}, 3, v_1)$ and $(G_8, G_{15}, 1, v_2)$ are recorded.

The reason we need all the excitation scenario numbers that give correct output responses for every vector is because we need to record conditions where the potential error(s) on the line(s) does not influence the primary output responses during simulation of the test vector. This will prove helpful during correction so we do not accidentally miss valid N-error corrections. The following example illustrates the basic idea.

Example 5: Consider the circuit in Fig. 5 during simulation of vector $v_2 = (0,0,1,0,0)$. Recall that error pair (G_8, B_0) qualified implicit error enumeration, as shown in Example 3. It also qualifies error simulation, and for v_2 tuples, $(G_8, B_0, 1, v_2)$ and $(G_8, B_0, 3, v_2)$ are recorded. Error excitation scenario numbers 1 and 3 together imply that the logic value of B_0 for v_2 can also be a "don't care" (X). This is expected since B_0 has no sensitized paths to any primary output for v_2 and *any* type of correction can be applied on the line for v_2.

E. Overall Diagnosis Approach

In this section, we describe the overall error diagnosis approach. For multiple error diagnosis, we present the concept of a checkpoint, an observation on the *structural* properties of a combinational circuit that allows us to speed the procedure.

We define a *checkpoint* $B \in G_C$ to be either a primary output or a fan-out stem of G_C. We define the *clan* B_{clan} of checkpoint B to be the set of all lines l, including B, such that every path from l to some primary output goes through B and B is the first such checkpoint. Computing the set of checkpoints for a circuit G_C takes time linear in the number of lines of G_C. It should be noted that our definition and use of checkpoints is different from the one presented in [35].

For example, the circuit in Fig. 5 has checkpoints $I_2, I_3, G_9, G_{12}, O_1$, and O_2. We also have $G_{12\text{clan}} = \{B_7, B_4, G_{12}\}$.

```
IMPLICIT_MULTIPLE_ERROR_DIAGNOSIS( F_C, G_C, N, n, iters, V_act, V_all)
1.   V'_all = V_all
2.   while V'_all ≠ ∅ do
3.        Randomly choose V_j ∈ V'_all and add it in IG G
4.        Delete V_j from V'_all
5.        while there are N-graph reductions on G do
6.             REDUCE_IG(G, N)
7.   C_error=IMPLICIT_ERROR_ENUMERATION(N, n)
8.   for every error tuple (l_1, l_2, .... l_N) in C_error do
9.        ERROR_SIMULATION(F_c, G_c, l_1, l_2, ..., l_N, V_act)
10.  if C_error = ∅ then
11.       if iters iterations performed for same value of N then
12.            N = N + 1
13.       go to line 1
14.  return(C_error)
```

Fig. 9. Multiple error diagnosis algorithm.

The following theorem, which follows immediately from the work in [10] and [16], is essential for the correctness of our multiple design error diagnosis algorithm.

Theorem 3: Let line $l \in G_C$ in the clan of checkpoint B, that is, $l \in B_{\text{clan}}$. If l belongs to some valid N-error line tuple, then B also belongs to some valid N-error line tuple.

Intuitively, the theorem holds because every sensitized path from l to the erroneous primary output(s) must necessarily pass through checkpoint B.

Single error diagnosis ($N = 1$) is a straightforward extension of the ideas presented in Sections III-B and III-D. First, we compile a set of distinct vertices V_{all} from consecutive path-trace runs for all vectors of $\mathcal{V}_{\text{act}}$. Then, we quickly reduce the error space by intersecting the lines of the members of V_{all}, and we follow with error simulation. If C_{error} is empty after error simulation. then we increase the value of N by one and enter multiple error diagnosis.

Referring to Fig. 9, multiple design error diagnosis ($N \geq 2$) proceeds as follows. The IG graph is built and processed, according to the algorithm presented in Section III-C, in lines 1–6. Vertex insertions (line 3) are followed by N-graph reductions (lines 5–6) until V'_{all} is empty and no more reductions are possible. The error set C_{error} is created in line 7 with the implicit error enumeration procedure described in Section III-C2 and a value for the n-sample specified by the user. If C_{error} becomes empty during error simulation (lines 8–9), then we repeat the process, as shown in lines 10–13. If the maximum number of iterations *iters* has been reached, we also increase the value of N by one.

In our implementation, the procedure of Fig. 9 is first applied on the set of the checkpoints of the circuit G_C, G_C^{cp}. Due to Theorem 3, the theory developed in this section also holds for the checkpoints of the circuit. Once the algorithm of Fig. 9 terminates on the checkpoints of G_C and outputs a set of N-error checkpoint tuples, $C_{\text{error}}^{\text{cp}}$, we exhaustively create C_{error} from the clans of the checkpoints contained in $C_{\text{error}}^{\text{cp}}$. Then we run error simulation to obtain the final set C_{error} of N-error line tuples. This set is the input to the correction algorithm, presented in the next section.

F. Handling Unknown Values

In our presentation, the assumption is that the design is completely simulatable; that is, we are able to perform simulation

with specified values zero and one, and we exclude unknown value X. This assumption can be partially relaxed in two steps, so that we allow simulation using three-valued logic values.

If the specification is incomplete and the output of the circuit is not specified for some input combinations, then as long as none of these combinations is applied, the method works with no change in the algorithm [2].

However, if it is not possible to drop all such unspecified input combinations from the test set, observe that in our presentation of path trace, the procedure starts from an erroneous primary output where the good circuit has a fully specified value (zero or one) and the faulty one has the opposite response. Consequently, this erroneous primary output cannot have the unknown value X. Now, observe that a gate whose output corresponds to the line under consideration either has one or more controlling inputs or has all noncontrolling values at its inputs. Therefore, unknown values present no problem for implicit error enumeration, as long as there are input test vectors that create sensitized path(s) from every error location to some primary output(s).

The same argument holds for error simulation, as we only simulate specified values.

IV. Error Correction

Correction follows after diagnosis terminates with a nonempty C_{error}. During correction, every set of N lines $L = \{l_1, l_2, \ldots, l_N\} \in C_{\text{error}}$ is considered separately for every vector $v \in \mathcal{V}_{\text{act}}$. The following theorem guarantees that the correction procedure, described later in this section, will include all valid N-correction tuples from our correction model.

Theorem 4: Let G_C be an N-source correctable design, and let $L = \{l_1, l_2, \ldots, l_N\} \in C_{\text{error}}$ with excitation numbers $m_1, m_2, \ldots, m_k$ for $v \in \mathcal{V}_{\text{act}}$. Define set $\mathcal{S}_i$

$$\mathcal{S}_i = \{2^{i-1} \text{ and } m_1, 2^{i-1} \text{ and } m_2, \ldots, 2^{i-1} \text{ and } m_k\}, \quad 1 \le i \le N.$$

If $C = \{c_1, c_1, \ldots c_N\}$ is a valid N-correction tuple for the lines of L and f is the logic value of l_i under simulation of v (i.e., if v is the jth vector in $\mathcal{V}_{\text{act}}$, then the jth bit of *Elist* is f), then the value of l_i, $\forall i, 1 \le i \le N$, for v when l_i implements function c_i should be:

1) f, if $\mathcal{S}_i$ contains all zeroes;
2) $\overline{f}$, if $\mathcal{S}_i$ contains all ones;
3) it can be any value (zero or one), if $\mathcal{S}_i$ contains both zeroes and ones.

Proof: We only prove the first case, as the other two cases can be proven in a similar manner.

If $\mathcal{S}_i$ contains only zeroes, then it means that error simulation gives correct primary output responses for simulations where the potential error on l_i is not excited. In other words, a valid correction c_i maintains on line l_i the logic value f for vector v.

During correction, for every N-error line tuple $L = \{l_1, l_2, \ldots, l_N, m_1, m_2, \ldots, m_k, v\} \in C_{error}$ and for every $v \in \mathcal{V}_{act}$, we *exhaustively* compile a list of corrections from our design error model. From these corrections, we keep the ones that when applied to the lines of L produce for each l_i a new logic value for v that satisfies the requirements of Theorem 4. For wire-related corrections, we consider adding wires that do not create loops in the combinational circuitry.

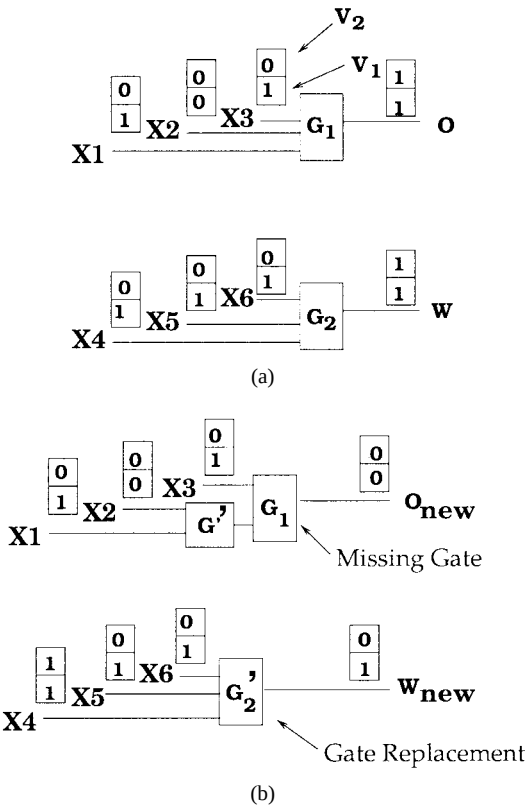


Fig. 10. Wrong gate correction for error pair. (a) Erroneous circuit and (b) pair of corrections that qualify.

The following examples illustrate the above correction strategy.

Example 6: Recall from Example 4 that error pair $(l_1, l_2) = (G_8, G_{15})$ qualified error simulation and tuples $(G_8, G_{15}, 3, v_1)$ and $(G_8, G_{15}, 1, v_2)$ were recorded. For vector v_1, we have $\mathcal{S}_1 = 1$ and $\mathcal{S}_2 = 1$. According to Theorem 4, a pair of valid corrections complements the existing logic values on lines (G_8, G_{15}) for v_1. Indeed, when G_8 is replaced by an NAND gate and extra wire B_0 is removed, the new logic values on G_8 and G_{15} are one and zero, respectively. Similar reasoning shows that this pair of corrections satisfies Theorem 4 for tuple $(G_8, G_{15}, 1, v_2)$ and qualifies the correction procedure.

Example 7: Consider the circuitry in Fig. 10(a) and assume that it is a pair of suspicious lines in some erroneous macro-based circuit G_C simulated for vectors $v_1, v_2 \in \mathcal{V}_{\text{act}}$. The *Elist* bit list for the vectors is shown in boxes above each line.

Correction is applied on error pair $\{G_1, G_2\}$ with the following entries obtained from error simulation, $\mathcal{T}_1 = (G_1, G_2, 1, v_1), \mathcal{T}_2 = (G_1, G_2, 1, v_2)$, and $\mathcal{T}_3 = (G_1, G_2, 3, v_2)$. As explained earlier, excitation configuration 1 for locations $\{O, W\}$ and vector v states that v produces correct primary output results for error simulation if the potential error on O is excited and the error on W is not. Equivalently, excitation configuration 3 implies that both potential errors on

TABLE I
CIRCUIT CHARACTERISTICS

ckt name	# of primary inputs/outputs	# of lines	# of checkpoints	average clan size	Error Space 1 error	Error Space 2 errors	Error Space 3 errors
C432	36/7	545	89	**6.1**	545	$2.9 \cdot 10^5$	$1.6 \cdot 10^8$
C880	60/26	880	126	**7.0**	880	$7.7 \cdot 10^5$	$6.8 \cdot 10^8$
C499	41/32	1224	155	**7.8**	1224	$1.4 \cdot 10^6$	$1.8 \cdot 10^9$
C1355	41/32	1355	259	**5.1**	1355	$1.8 \cdot 10^6$	$2.4 \cdot 10^9$
C1908	33/25	1908	384	**4.9**	1908	$3.6 \cdot 10^6$	$6.9 \cdot 10^9$
C2670	157/63	2670	454	**5.8**	2670	$7.1 \cdot 10^6$	$1.9 \cdot 10^{10}$
C3540	50/22	3540	601	**5.8**	3540	$1.2 \cdot 10^7$	$4.4 \cdot 10^{10}$
C5315	178/123	5315	806	**6.5**	5315	$2.8 \cdot 10^7$	$1.5 \cdot 10^{11}$
C6288	32/32	6319	1487	**4.2**	6319	$3.9 \cdot 10^7$	$2.5 \cdot 10^{11}$
C7522	207/108	7552	1300	**5.8**	7552	$5.7 \cdot 10^7$	$4.3 \cdot 10^{11}$

O and W need to be excited. Observe that tuples $\mathcal{T}_2$ and $\mathcal{T}_3$ are similar to the situation presented in Example 5.

Assume that the correction pair under consideration is a missing gate $G' \neq G_1$ for O and a gate replacement error $G_2' \neq G_2$ for W. Also, assume that when we apply these corrections, as shown in Fig. 10(b), and perform one *local* simulation step with the use of the *Elist* bit lists on $X_1, \ldots, X_6$, we get the new values for O_{new} and W_{new} shown in the shaded boxes. It can be seen that these new values respect Theorem 4 for v_1 because O_{new} has a complemented bit entry zero (2^0 *and* $1 = 1$) and W_{new} maintains its one *Elist* value (2^1 *and* $1 = 0$). The same holds for v_2, and this correction pair qualifies. Note that if N-error line tuple $\mathcal{T}_3$ was not in the output of error simulation, then the above two-correction tuple would not qualify.

Once we exhaustively compile a list of corrections as described above, a last test vector simulation-based verification step is performed for the vectors of the *Clist* bit list at the fan-out cones of the lines with corrections. Corrections that give correct primary output responses are the *output* of our DEDC algorithm and input to the logic verifier.

V. EXPERIMENTS

We implemented the DEDC algorithm in C language and ran it on a Sparc 10 workstation for the ISCAS'85 circuits corrupted with one, two, and three design errors from our design error model. The types and locations of the errors injected were selected randomly. We ran 20 experiments for each of the three different scenarios for a total of 60 experiments per circuit. For the three-error case of circuit $C6288$, we ran only five experiments due to the increased complexity of the design [7], [13]. The average values of the results of our experiments are reported in the next pages. All run times are in seconds.

The nonoptimized ISCAS'85 benchmark circuit characteristics can be found in Table I. The initial size of the error space, that is, the total number of N-error line tuples for each of these designs, can be found in the last three columns of that table. These numbers are computed according to (1). The run-time savings for diagnosis due to the observations in Section II-E can be computed if we estimate the *average clan size*, shown in column 5. This number also gives a lower bound for the average speed up of the algorithm on one-source correctable designs versus the naive approach that considers all circuit lines. For two-source correctable designs, this lower bound is the number in column 5 raised to the power of two, etc.

A. Results on Diagnosis and Correction

Table II contains results on the performance and the output of our DEDC methodology. The three rows for each circuit correspond to the one-, two-, and three-source correctability cases, respectively.

The second column of Table II contains the size of $\mathcal{V}_{\text{test}}$, that is, the total number of stuck-at [19] and random vectors we use during the initial simulation based verification step. The next column contains the hit ratio of these vectors to activate the inconsistencies of the erroneous design. We use a subset of these vectors to compile $\mathcal{V}_{\text{act}}$. The average size of $\mathcal{V}_{\text{act}}$ is shown in column 4. Unless the design error(s) is redundant [9], $\mathcal{V}_{\text{act}}$ is never empty throughout our experiments.

The next three columns contain results on implicit error enumeration. Column 5 shows the most frequent IG type obtained. The clique case is the most computationally expensive case to handle, while an N-disconnected component IG provides faster and more accurate error resolution. We can see that implicit error enumeration is a quite efficient procedure considering the size of the initial error space shown in the last three columns of Table I. As explained in Section II-E, the algorithm begins with $N = 1$ and repeats *iters* times if C_{error} becomes empty at the end of error simulation. In our implementation, we set the value of *iters* to three. This number of iterations proves to be sufficient to obtain a good error sample. The average number of iterations is less than 1.8 for both the two- and three-source correctability cases, and the average size of the n-sample is 3.3 vertices for the two-source correctable experiments and 7.2 vertices for the three-source correctable experiments. If the algorithm reaches the maximum value of *iters*, it automatically increases the value of N by one.

Results for error simulation on both checkpoints and their clans can be found in columns 8 and 9. Usually, the majority of the error tuples entering from implicit error enumeration are deleted during the first iterations of error simulation for the vectors of $\mathcal{V}_{\text{act}}$. There are also cases that particular input vectors reduce the size of C_{error} dramatically. These experimental observations suggest that if $\mathcal{V}_{\text{test}}$ contains a design error-specific test set [4], then we can possibly improve on run-time performance.

TABLE II
DEDC RESULTS

crt	$\|\mathcal{V}_{test}\|$	vector	$\|\mathcal{V}_{act}\|$	IG Processing			Error Simulation		Correction	
name		hit–ratio		IG type	$\|C_{error}\|$	time	$\|C_{error}\|$	time	# tuples	time
C432	10000	10.8 %	40		7.3	0.8	1.5	0.4	**1.9**	2.6
		24.2 %	150	clique	72.0	1.3	8.3	1.6	**8.2**	22.5
		39.7 %	150	2	8136.5	8.3	11.7	23.1	**11.5**	47.8
C880	10000	28.7 %	50		1.4	0.7	1.4	0.3	**1.2**	5.3
		61.8%	150	2	194.8	1.6	3.1	2.9	**2.3**	10.6
		77.0 %	150	3	1088.0	8.1	3.1	2.9	**4.4**	29.0
C499	10000	11.7 %	50		13.1	1.1	2.1	0.9	**2.5**	6.1
		52.8 %	220	2	1722.3	1.8	16.1	4.9	**9.1**	43.9
		67.1 %	220	2	24344.5	10.3	18.0	97.1	**11.1**	89.4
C1355	12000	20.0 %	80		7.6	1.0	0.2	0.9	**2.8**	4.3
		49.9 %	100	2	4018.6	3.7	9.3	4.9	**12.3**	94.1
		57.3%	120	clique	175300.0	10.9	21.3	139.8	**9.4**	174.5
C1908	12000	10.4 %	80	—	2.9	1.5	2.9	3.1	**2.1**	8.7
		38.0 %	220	clique	3709.5	6.9	7.3	9.4	**4.2**	107.1
		41.9 %	220	2	68544.9	14.4	28.1	180.6	**6.7**	212.7
C2670	12000	17.3 %	50	—	14.2	2.3	2.9	0.8	**2.5**	11.1
		61.7 %	220	2	3252.0	8.8	16.3	14.1	**13.4**	129.99
		79.1 %	300	2	74355.6	19.1	29.5	108.1	**22.9**	211.8
C3540	15000	12.6 %	80	—	3.8	2.0	3.0	1.1	**2.8**	11.8
		58.3 %	300	clique	7195.8	4.9	14.1	5.8	**8.9**	138.4
		74.2 %	300	clique	212776.0	21.0	22.3	245.1	**11.4**	204.3
C5315	15000	14.8 %	80	—	18.0	2.1	3.2	1.1	**2.2**	11.8
		53.2 %	300	2	77001.2	19.2	18.0	142.9	**4.7**	118.6
		71.6 %	300	2	77001.2	19.2	18.1	142.9	**4.9**	241.0
C6288	8000	20.7 %	100	—	3.2	3.9	2.1	1.1	**1.9**	5.8
		41.8 %	220	clique	122022.3	102.1	6.1	1233.0	**9.7**	149.3
		69.7 %	220	clique	2388039.5	230.7	28.9	8042.8	**24.1**	1092.6
C7552	15000	19.3 %	100		18.0	2.1	2.9	1.3	**1.4**	14.3
		46.2 %	300	2	1311.7	4.4	13.4	7.1	**18.2**	169.9
		62.4 %	300	2	871104.3	21.3	31.4	495.1	**21.1**	301.7

Column 10 contains the number of the N-correction tuples returned by the correction algorithm, and the next column contains the run times for returning *all* possible N-correction tuples. The run times of column 11 also contain the time needed for correction verification with the use of the *Clist* bit lists. The algorithm can be modified to exit when one correction is found for a fraction of the run times shown of column 11. Column 12 of Table II contains the overall time for DEDC and exhibits the robustness and good error/correction resolution of the proposed methodology.

We also performed experiments that allow two-graph reductions on IG's where the maximum number of pairwise nonadjacent vertices is two but the design is three-source correctable. These experiments validate the results in Corollary 2, and error resolution deteriorates in most (50%–85%) cases. However, whenever we are able to obtain valid three-error line tuples, the size of C_{error} is orders of magnitude smaller than the one shown in column 6, thus improving performance.

In addition, we ran our diagnosis algorithm on N-source correctable designs where $N>3$. In these cases, the clique was the most frequent IG structure, but the algorithm avoided the explosion of the error space according to (1). One can possibly further improve performance by rectifying small sets of erroneous primary outputs, one at a time, as described in Lin *et al.* [17]. Considering the complexity of the problem for $N>3$, an approach can also possibly allow K-graph reductions, $K<N$, on the design, whenever possible, and repeat the procedure if C_{error} becomes empty.

Last, we tested our approach on the combinational versions of the ISCAS'89 sequential circuits. The run times we obtained for some of the largest circuits of this family of benchmarks is comparable to the ones for midsized ISCAS'85 circuits. This is because the combinational logic depth of the transformed ISCAS'89 circuits decreases significantly when the inputs and outputs of the state elements become pseudoprimary outputs and inputs, respectively. Moreover, the large number of inputs and outputs versus the amount of combinational logic for the modified circuits causes distinct runs of path trace to mark nonoverlapping sets of circuit lines and the intersection graph to return with pairwise nonadjacent components most of the times. This, as we discussed, improves on error resolution and reduces the overall run time. It should be noted that experimental data on the performance of the implicit enumeration procedure alone for one- and two-source correctable ISCAS'89 designs can be found in [30] and [31], respectively.

B. On the Quality of Test Vector Simulation for DEDC

From our discussion in Section I, it is clear that the *quality* of any test vector simulation algorithm for multiple DEDC depends on the input test vector size $\mathcal{V}_{\text{test}}$ on which the algorithm bases its diagnosis and correction decisions. The set of input test vectors is crucial for diagnosis because it has to activate all inconsistencies in G_C. According to the theoretical

TABLE III
CORRECTION HIT RATIO FOR SMALLER $|\mathcal{V}_{\text{test}}|$

ckt name	1 source Corr.		2–source Corr.	
	$\mathcal{V}_{test}$	hit ratio	$\mathcal{V}_{test}$	hit ratio
C432	2000	97.8 %	2500	100 %
C499	3500	96.6 %	3500	90.1 %
C880	1200	100 %	2500	100 %
C1355	3000	99.1 %	3500	91.8 %
C1908	4000	99.0 %	4000	96.5%
C2670	4200	97.2 %	4500	89.8 %
C3540	4200	93.2 %	5000	94.1 %
C5315	4500	93.5 %	6000	98.9 %
C7552	5000	94.9 %	7000	91.3 %

and experimental results of [2] and [4], deriving a complete (i.e., 100% error coverage) design error test set for the model of Abadir *et al.* [2] through ATPG techniques, requires an expensive and practically unrealistic procedure.

The set of input test vectors is also important for correction. Since exhaustive test vector simulation is prohibitive for most designs, a correction method that bases its results on a subset $\mathcal{V}_{\text{test}}$ of the *complete* input test vector space can produce corrections that correct G_C for the vectors of $\mathcal{V}_{\text{test}}$ only. Therefore, it is of interest to know the quality of these corrections for the complete input test vector space.

We use the term *correction hit ratio* to denote the percentage of valid N-correction tuples in the set of corrections returned by our algorithm, or, equivalently, the number of corrections that exit the logic verifier [6] over the ones that enter it in Fig. 1. In our experiments, the correction hit ratio is 100% for all the ISCAS'85 circuits and the test vector set $\mathcal{V}_{\text{test}}$ sizes shown in Table II. In other words, no global looping of the DEDC algorithm occurred following the logic verifier. Table III contains the correction hit ratios for a smaller number of input test vectors. These numbers complement the results in [2] and [4], as they indicate that there are design errors not only "hard to detect" but "hard to correct" as well.

Overall, our experimental results suggest that test vector simulation is indeed an attractive route to DEDC. It is computationally efficient for diagnosis, as it can narrow down the error space rapidly and it scales well with increasing number of errors. It is also effective for correction, since the amount of resynthesis performed is indeed minimal and the vast majority of the corrections returned are valid corrections, although a logic verifier is a requirement at the back end of such a method. Last, it is applicable to circuits that have no global BDD representation.

C. Design Error Masking

The circuit in Fig. 11(a) is corrupted by two errors since G_1 should be an OR gate and G_2 an AND gate. Nevertheless, the error on G_1 is not observable since there is no sensitized path from G_1 to a primary output for any input test vector. Note that the error on G_1 is observable when the error on G_2 is corrected, as shown in Fig. 11(b). This situation is referred in the literature as *error masking* [21].

Unfortunately, design error masking can mislead existing DEDC techniques, including the work presented here. For

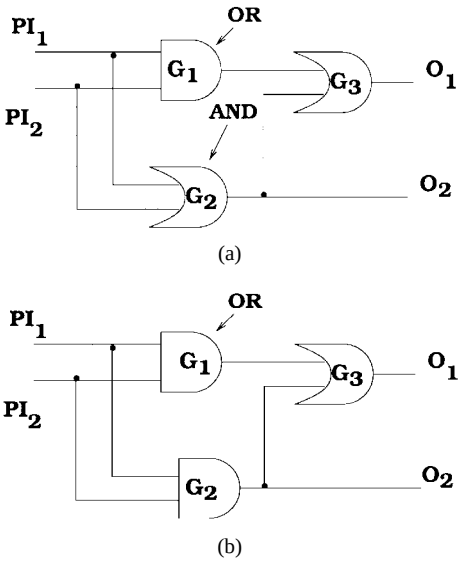


Fig. 11. Error masking. (a) Error masked and (b) not masked

example, the circuit of Fig. 11(a) is two-source correctable but existing DEDC techniques will attempt to correct it with a single correction. However, a single correction is not sufficient because the circuit of Fig. 11(b) is still erroneous.

If a design fails to be N-source correctable, it may be caused by error masking. In our experiments, error masking did not occur for $N \leq 3$. We ran experiments for higher values of N, $6 \leq N \leq 8$, and counted sensitized paths from the error locations to the primary outputs. Error masking was rare, as it occurred two times in a total of 60 experiments on circuits $C432$, $C880$, $C499$, and $C1908$. It may be concluded that design error masking is rare, but when it happens, it is a difficult problem to solve.

VI. DISCUSSION

A. Sequential Circuit DEDC

Our DEDC method is applicable to combinational circuits and sequential designs where a one-to-one correspondence of the state elements between the specification and the design is available because one can extract the combinational circuitry and apply the proposed algorithm [14].

If such correspondence is not available, combinational DEDC techniques, although applicable, may no longer be efficient for *sequential circuit DEDC*. For sequential circuit DEDC, the *iterative array representation* of the circuit for consecutive time frames seems to be necessary [14]. However, this representation increases the problem complexity even for designs corrupted with a single error because the combinational part of the circuit increases dramatically with every time frame, as does the error space (1). This is also experimentally reported by Huang *et al.* [14]. In addition, the generation of input test vector sequences that activate the inconsistencies is also a significant research challenge for sequential circuit DEDC [14].

Due to the inherent difficulty of the problem, little work has been performed [8], [14], [20], [33]. Considering the run-time

efficiency of the implicit error enumeration procedure, it will be interesting to know its application and performance on sequential DEDC. In addition, the quality of test vector simulation for sequential design error verification, diagnosis, and correction will need to be examined.

B. Engineering Change

In a typical VLSI synthesis process, specifications may change even at a late stage of the design cycle when the designer has already invested a significant amount of effort on the design. Since automated tools for synthesis and optimization tend to find a minimal representation of the requested function, engineering changes to the original specification may require large changes in the existing gate-level implementation if a conventional resynthesis procedure is applied. This is undesirable, as it can jeopardize some of the engineering effort already invested in the design. In the problem of *engineering change*, one is interested in the least amount of resynthesis on the existing design to obtain one that satisfies the new specification [5], [11], [12], [17].

Depending upon the information available, two versions for engineering change can arise. For each version, a fundamentally different solution is developed. In the first version, a naming equivalence (i.e., functional equivalence) between signals of the new and old specification and the existing design is available from the synthesis process. Existing work [5], [11], [12] uses this information to resynthesize the signals that are not functionally equivalent. In the second version [17], such a naming correspondence is not available as the old and new specification can only provide primary output responses in terms of the primary input stimuli. It is reported by Lin *et al.* [17] that DEDC can be also viewed as an instance of the second version of engineering change. Nevertheless, engineering change is inherently more difficult, as we cannot necessarily expect that a few modifications will always provide a solution.

In terms of the DEDC problem, the second version of engineering change can be expressed as a problem where the *minimum* sequence of the following two operations:

- add/delete any existing wire in the design as fan-in to any gate or as a branch line;
- add/delete any simple gate;

is required on the existing design to obtain one that implements the new specification. It is clear that this set of operations, which is a subset of the design error model of Abadir *et al.* [2], is sufficient to transform the design to the new one. It is our conjecture that this problem is NP-Complete, and an efficient algorithm will need to make use of heuristics. We believe that existing DEDC methodologies can lead to efficient solutions where engineering changes are performed while reusing a significant amount of the existing design.

VII. Conclusions

We described a test vector simulation-based approach for multiple design error diagnosis and correction. Diagnosis is independent of any design error model, and correction uses an extension of the model presented by Abadir *et al.* [2].

Diagnosis uses the results of the path-trace procedure to construct an intersection graph. A novel implicit enumeration procedure derives potential error lines from this graph and avoids the exponential growth of the error space with increasing numbers of errors. An error simulation procedure improves on error resolution. It also records information that is used during correction. Last, correction returns a set that contains N-correction tuples, and this set is the input to a logic verifier.

Experiments confirm theoretical results, as they exhibit the efficiency and accuracy of our approach. They also indicate that test vector simulation is an attractive alternative to symbolic techniques for multiple design error diagnosis and correction, since it is applicable to all circuits and its performance scales well as the number of errors increases. Nevertheless, a logic verifier is required at the back end of such a method to guarantee the quality of the proposed corrections since some design errors may not only be hard to diagnose but also can be hard to correct.

Acknowledgment

The authors thankfully acknowledge the valuable comments of the anonymous reviewers of this paper and previous conference versions of this work that helped improve the presentation of the material. They would also like to acknowledge the technical contribution of Dr. S. Venkataraman and Prof. W. K. Fuchs.

References

[1] E. J. Aas, K. Klingsheim, and T. Steen, "Quantifying design quality: A model and design experiments," in *Proc. EURO-ASIC*, 1992, pp. 172–177.

[2] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic verification via test generation," in *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 138–148, Jan. 1988.

[3] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical path tracing: An alternative to fault simulation," in *IEEE Design Test Comput. Mag.*, vol. 1, pp. 89–93, Feb. 1984.

[4] H. A. Asaad and J. Hayes, "Design verification via simulation and automatic test pattern generation," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1995, pp. 174–180.

[5] D. Brand, A. Drumm, S. Kundu, and P. Narain, "Incremental synthesis," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 14-18.

[6] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.

[7] ——, "On the complexity of VLSI implementation and graph representations of Boolean functions with application to integer multiplication," in *IEEE Trans. Comput.*, vol. 40, pp. 205–213, Feb. 1991.

[8] M. Fujita, Y. Tamiya, Y. Kukimoto, and K. C. Chen, "Application of Boolean unification to combinational synthesis," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1991, pp. 510–513.

[9] P. Y. Chung, Y. M. Wang, and I. N. Hajj, "Logic design error diagnosis and correction," in *IEEE Trans. VLSI Syst.*, vol. 2, pp. 320–332, Sept. 1994.

[10] P. Y. Chung and I. N. Hajj, "Diagnosis and correction of multiple design errors in digital circuits," in *IEEE Trans. VLSI Syst.*, vol. 5, pp. 233–237, June 1997.

[11] S. Y. Huang, K. C. Chen, and K. T. Cheng, "Error correction based on verification techniques," in *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 258–261.

[12] ——, "Incremental logic rectification," in *Proc. IEEE VLSI Test Symp.*, 1997, pp. 143–149.

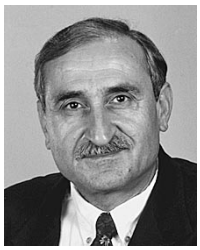
[13] ——, "ErrorTracer: A fault simulation-based approach to design error diagnosis," in *Proc. IEEE Int. Test Conf.*, 1997, pp. 974–981.

[14] S. Y. Huang, K. T. Cheng, K. C. Chen, and J. Y. J. Lu, "Fault-simulation based design error diagnosis for sequential circuits," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 667–691.

[15] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin, "Error diagnosis for transistor level verification," in *Proc. Design Automation Conf.*, 1994, pp. 218–224.
[16] H. T. Liaw, J. H. Tsaih, and C. S. Lin, "Efficient automatic diagnosis of digital circuits," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1990, pp. 464–467.
[17] C. C. Lin, K. C. Chen, S. C. Chang, M.-M. Sadowska, and K. T. Cheng, "Logic synthesis for engineering change," in *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 647–652.
[18] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of digital errors with uuPRIAM," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, June 1989, pp. 30–33.
[19] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. Eur. Automation Conf.*, 1991, pp. 214–218.
[20] I. Pomeranz and S. M. Reddy, "A method for diagnosing implementation errors in synchronous sequential circuits and its implications on synthesis," in *Proc. Eur. Design Automation Conf.*, 1993, pp. 252–258.
[21] ——, "On diagnosis and correction of design errors," in *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 255–264, Feb. 1995.
[22] ——, "On error correction in macro-based circuits," in *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1088–1100, 1997.
[23] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *IWLS Workshop Notes*, 1993, session 3a, pp. 1–12.
[24] K. A. Tamura, "Locating functional errors in logic circuits," in *Proc. Design Automation Conf.*, June 1989, pp. 185–191.
[25] M. Tomita, H. H. Jiang, T. Yamamoto, and Y. Hayashi, "An algorithm for locating design errors," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1990, pp. 468–471.
[26] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano, "Rectification of multiple logic design errors in multiple output circuits," in *Proc. Design Automation Conf.*, 1994, pp. 212–217.
[27] A. Veneris and I. N. Hajj, "A fast algorithm for locating and correcting simple design errors in VLSI digital circuits," in *Proc. 7th IEEE Great Lakes Symp. VLSI*, 1997, pp. 45–50.
[28] ——, "Correcting multiple design errors in digital VLSI circuits," in *Proc. Int. Symp. Circuits and Systems*, to be published.
[29] A. Veneris, S. Venkataraman, I. N. Hajj, and W. K. Fuchs, "Multiple design error diagnosis and correction in digital VLSI circuits," in *Proc. IEEE VLSI Test Symp.*, 1999, pp. 58–63.
[30] S. Venkataraman, I. Hartanto, and W. K. Fuchs, "Dynamic diagnosis of sequential circuits based on stuck-at faults," in *Proc. VLSI Test Symp.*, 1996, pp. 198–203.
[31] S. Venkataraman and W. K. Fuchs, "A deductive technique for diagnosis of bridging faults," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1997, pp. 562–567.
[32] Y. Watanabe and R. K. Brayton, "Incremental synthesis for engineering changes," in *Proc. IEEE/ACM Int. Conf. Computer Design*, 1991, pp. 40–43.
[33] A. M. Wahba and D. Borrione, "Design error diagnosis in sequential circuits," in *Proc. Correct Hardware Designs and Verification Methods, Lecture Notes in Computer Science*, 1995, no. 987, pp. 171–188.
[34] ——, "A method for automatic design error location and correction in combinational logic circuits," in *J. Electron. Testing, Theory, Applicat.*, vol. 8, no. 2, pp. 113–127, Apr. 1996.
[35] N. Yanagida, H. Takahashi, and Y. Takamatsu, "Multiple fault diagnosis in sequential circuits using sensitizing sequence pairs," in *Proc. Fault Tolerant Computing Systems*, 1996, pp. 86–95.

Andreas Veneris (S'96) was born in Athens, Greece. He received the diploma in computer engineering and informatics from the University of Patras, Greece, in 1991, the M.S. degree in computer science from the University of Southern California, Los Angeles, in 1992, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1998.

In 1998, he became a Visiting Faculty Member at the University of Illinois until 1999, when he joined the Department of Electrical and Computer Engineering at the University of Toronto, Canada, as an Assistant Professor. He has also carried out research in distributed algorithms and combinatorics. He is the coauthor of a book on Fortran programming language and he contributed to the first webcasting ever. His research interests include CAD for synthesis, testing, and diagnosis of digital VLSI circuits and theoretical computer science.



Ibrahim N. Hajj (S'64–M'70'SM'82–F'90) received the B.E. degree (with distinction) from the American University of Beirut, Beirut, Lebanon, the M.S. degree from the University of New Mexico, Albuquerque, and the Ph.D. degree from the University of California at Berkeley, all in electrical engineering.

He is currently a Professor of electrical and computer engineering and a Research Professor at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Prior to joining the University of Illinois, he was with the Department of Electrical Engineering at the University of Waterloo, Waterloo, Ont., Canada. His current research interests include computer-aided design of VLSI circuits, design for reliability and low-power synthesis, physical design, and testing. He has published more than 160 journal and conference papers and book chapters on these subjects. He is a coauthor of *Switch-Level Timing Simulation of MOS VLSI Circuits* (Norwell, MA: Kluwer, 1989).

Dr. Hajj is a member of Computer-Aided Network Design (CANDE), ACM, and Sigma Xi. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART II: ANALOG AND DIGITAL SIGNAL PROCESSING and of the IEEE CIRCUITS ANDSYSTEMS MAGAZINE. In 1992, he was a corecipient of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award.