Multiple Design Error Diagnosis and Correction in Digital VLSI Circuits *

Andreas Veneris
University of Illinois
CS Department
Urbana, IL 61801

Srikanth Venkataraman
Intel Corporation
Santa Clara, CA 95052

Ibrahim N. Hajj
University of Illinois
ECE Department and CSL
Urbana, IL 61801

W. Kent Fuchs
Purdue University
ECE Department
W. Lafayette, IN 47907

Abstract

With the increase in the complexity of VLSI circuit design, logic design errors can occur during synthesis. In this work, we present a method for multiple design error diagnosis and correction. Our approach uses the results of test vector simulation for both error detection and error correction. This makes it applicable to circuits with no global BDD representation. In addition, diagnosis is performed through an implicit enumeration of potentially erroneous lines in an effort to avoid the exponential explosion of the error space. Experimental results on ISCAS'85 benchmark circuits show that our approach can typically detect and correct 1, 2 and 3 errors within seconds of CPU time.

1 Introduction

Design errors may occur at many stages during the design of VLSI digital circuits. In this paper, we are interested in the problem of **multiple Design Error Diagnosis and Correction (DEDC)** in an erroneous netlist. In our treatment of this problem, design errors are functional mismatches between the specification and the erroneous gate–level implementation (design). Sources of these mismatches can be the manual interference of the designer with the synthesis process, errors at a higher level of the design flow, and occasionally, software bugs in CAD tools [1]. In addition, we assume that the only netlist available as an input to the problem is the erroneous design. The specification, possibly available in some high level description language, can only provide the primary output responses in terms of a primary input stimulus.

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 that involves the functional misbehavior of gate element(s) and wire interconnection error(s). Experimental data shows that this model covers 97.8% of the errors that occur in a real life synthesis environment [1]. For this reason and because of its simplicity, the model of [2] has been used by the majority of the existing literature including the work presented here.

Aas et al. [1] show that the average number of actual design errors is usually less than or equal to 2. However, since there may be more than one way to synthesize a particular function, there may be equivalent corrections, along with the actual one, that can rectify the design [6]. In this paper, we refer to the actual and any equivalent set of corrections as **valid** corrections.

It can be seen, that the problem of multiple DEDC is indeed a challenging one as the space of potential erroneous lines grows exponentially to the number of errors [11]:

$$(\#\ of\ circuit\ lines)^{(\#\ of\ erroneous\ lines)} \tag{1}$$

A number of approaches have been developed for solving the DEDC problem. These approaches are divided into two categories with respect to the underlying technique used for error location and error correction. Those based on **Boolean function manipulation** techniques [6] [8] and those based on **test vector simulation** [7] [9] [10] [11] [12].

Both techniques are run-time efficient and exhibit good error isolation for single errors. In addition, symbolic techniques can guarantee the correctness of the design. However, heuristics are usually employed for multiple errors due to the increase in the error space as indicated by Eq. 1. This is because most techniques perform an explicit enumeration of the error space, *i.e.* they start with the complete error space and delete candidates that do not meet certain criteria. In addition, little work has been performed in the area of error correction with test vector simulation for the complete model of Abadir et al. [11] [12].

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, a set of stuck-at and random vectors $\mathcal{V}_{test}$, a set of vectors $\mathcal{V}_{act} \subseteq \mathcal{V}_{test}$, such that each of them produces at least one primary output response in F_C

*This research was supported in part by the Joint Services Electronics Program under Contract #N00014-96-1-0129.

The e-mail address of the contact author is aveneris@cs.uiuc.edu that is different from G_C, and an initial guess for the number of required modifications N. We compile $\mathcal{V}_{act}$ from an initial test vector simulation based verification step with vectors from $\mathcal{V}_{test}$. $\mathcal{V}_{test}$ is also used during correction, while $\mathcal{V}_{act}$ gives information for diagnosis. In our experiments, $\mathcal{V}_{test}$ is usually less than 15,000 vectors and the average size of $\mathcal{V}_{act}$ fluctuates between 40 and 300 vectors.

Unlike previous diagnosis approaches, our method performs an implicit enumeration of suspicious lines in an effort to avoid the exponential explosion of the error space. This search of the error space is performed in two steps (Fig. 1). Initially, information on candidate error lines is collected through the path-trace procedure [15] to form a graph. Subsequent graph reductions on the vertices of this graph direct the search for potential error lines without necessarily computing the complete error line space. Next, 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 parameter values. Otherwise, it proceeds to correction and returns results from a design error model that is an *extension* of the model of Abadir et al. [2]. Finally, a logic verifier is used to *output* valid corrections. If the set of output valid corrections is empty, then the procedure is repeated until it returns with success or a maximum number of iterations is reached. In such a case, the algorithm is repeated for a larger value of N. Although, theoretically, the proposed approach can result in a procedure that computes the error set according to Eq. 1, our experiments show that this does not happen in practice and it remains computationally efficient for 1, 2 and 3 errors. This makes it applicable to circuits that are corrupted by multiple errors and circuits that have no global BDD [5] representation. In addition, our diagnosis methodology is *independent* of the design error model used.

This paper is organized as follows. The next section contains relevant definitions and describes the design error model we use. Diagnosis and correction are presented in sections 3 and 4, respectively. Experimental results are given and discussed in section 5 and section 6 concludes this paper.

2 Preliminaries

In this work, we examine incorrect combinational netlists G_C with `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 sets [2]. If such gates are included, one might prefer to use a test set derived specifically for design errors [4] to compile $\mathcal{V}_{test}$. During the execution of our algorithm, we introduce one buffer for every fan-out line of a branch. We say that a line l, fan–in to an `AND` or `NAND` (`OR` or `NOR`) gate, has *controlling value* for input vector v if the value of l is 0 (1). If l drives a `NOT` or a `BUFFER` it always has controlling value. Our algorithm can handle incompletely specified (X) input test sets as long as there are vectors in $\mathcal{V}_{act}$ that cause sensitized path(s) of specified values (0 and 1) from all error locations to primary outputs. Therefore, without loss of generality, we consider input test vectors with fully specified values [2] [13].

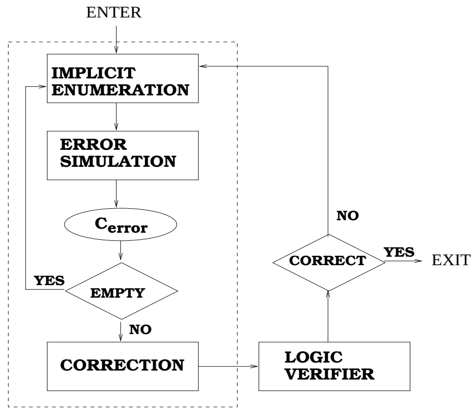

Figure 1: Overview of DEDC Methodology

In the design error model presented by Abadir et al. [2], errors are classified as **wrong gate** and **wrong wire** related errors. Possible wrong gate related errors are gate replacement, missing inverter, extra inverter, missing gate and extra gate errors. Possible wire related errors are extra wire, missing wire and incorrectly placed wire errors. Abadir et al. [2] proved that a complete set of test vectors for stuck-at faults for the erroneous circuit is guaranteed to detect the majority of the design errors of their model and has a very good chance of detecting the remaining ones. This result is experimentally confirmed by Asaad and Hayes [4] who also developed a test vector generator for design errors. 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 of [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.

In our discussion, an *N-error line tuple* $L = \{l_1, l_2, \ldots, l_N\}$, $N \geq 1$, is a set of N distinct circuit lines in 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.

3 Diagnosis

3.1 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. [14] [15] that is based on the critical path tracing algorithm [3]. In our work, it is used during implicit error space diagnosis.

Let vector $v \in \mathcal{V}_{act}$. Path–trace [14] [15] starts from an *erroneous* primary output for v and it traces backwards 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 non–controlling 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. 2 with primary outputs O_1 and O_2. If O_1 has been marked by path–trace, it can proceed by marking lines $S_1 = \{O_1, G_4, G_3, B_1, I_2\}$. If path–trace begins from O_2, lines $S_2 = \{O_2, B_4, G_2, B_2, I_2\}$ are marked.

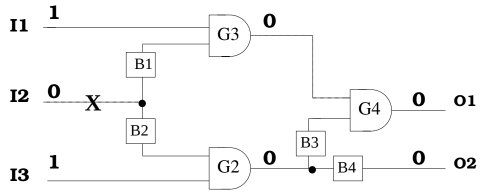

Figure 2: A 1–Source Correctable Design

Define V_j^i to be the set of lines selected by path–trace when tracing from erroneous primary output PO_i and vector v_j [15]. The following theorem [13], along with Theorem 2 presented in the next subsection, guarantee the correctness of our diagnosis algorithm.

<u>**Theorem**</u> **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.*

3.2 Pruning the Error Space

In this subsection, we define the intersection graph, introduced by Venkataraman and Fuchs [15], and present the concept of a N–graph reduction that allows efficient search for valid N–error line tuples.

The **Intersection Graph(IG)** [15] $G = (V, E)$ 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 $V_i \cap V_j \neq \emptyset$. In this paper, 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. 3(a) has 6 vertices, $V_1, V_2, \ldots, V_6$, and the IG in Fig. 3(b) has 3 vertices and 2 cliques.

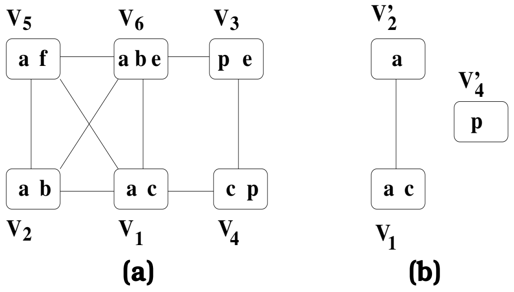

Figure 3: Examples of Intersection Graphs

<u>**Definition**</u> **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 $V_i^{adj} = \{V_{i_1}, V_{i_2}, \ldots, V_{i_k}\}$ 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 $\forall i, 1 \le i \le N$, 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**.

For example, the IG of Fig. 3(b) is the resulting graph when a 2–graph reduction, with respect to $V_R = \{V_2, V_4\}$, is carried on Fig. 3(a). Observe that $\{V_1\} \notin V_2^{adj} = \{V_5, V_6\}$ because it is adjacent to $V_4 \in V_R - V_2$. Notice that the reduced graph of Fig. 3(b) is still 2-reducible. It will be seen shortly, that N–graph reductions provide the means for acquiring an accurate estimate of the error space C_{error}.

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 we perform an N-reduction(s) whenever possible. This procedure terminates when path-trace is run for all vectors in $\mathcal{V}_{act}$. Theorem 2 that follows guarantees that this graph processing does not jeopardize the error resolution. A formal proof of this theorem can be found in [13].

<u>**Theorem**</u> **2** *Let G_C be an N–source correctable design and G be an IG for the design. At every stage of the graph processing, G contains at least one line from every valid N–error line tuple and it can have* at most *N vertices pairwise non–adjacent to each other.*

Observe that due to Theorem 2, if the *IG G* for erroneous design G_C has N pairwise non–adjacent vertices, then the design is guaranteed not to be K–source correctable for $K < N$, but it can be M–source correctable for some $M > N$. It can also be shown that if we allow K–graph reductions on the IG of an N–source correctable design, $K < N$, Theorem 2 no longer holds and error resolution may be jeopardized [13].

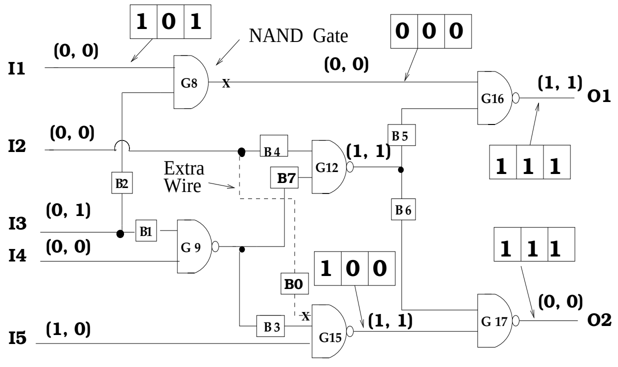

Figure 4: A 2–Source Correctable Design

Example 1: Consider the erroneous circuit in Fig. 4 with a gate replacement error on G_8 (`NAND` gate) and an extra wire B_0 (dotted line) simulated for input vectors $(0,0,0,0,1)$ and $(0,0,1,0,0)$. Line values during simulation are shown in parentheses. The first vector produces erroneous results at both primary outputs and the second vector activates the inconsistencies at primary output G_{16}. For the first input vector, the path–trace procedure marks set of lines $V_1 = \{G_{16}, G_8, I_1, B_2, I_3\}$ when it originates from G_{16} and it marks 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. 5(a). Fig. 5(b) shows the intersection graph and its adjacencies when vertex V_3 is added from path-trace for the second input test vector and erroneous output G_{16}. This allows a 2–graph reduction with respect to $V_{R_1} = \{V_1, V_2\}$. The resulting graph is shown in Fig. 5(c) where vertices V_1 and V_3 have been replaced by V_4 with line set $V_4 = V_1 \cap V_3$.

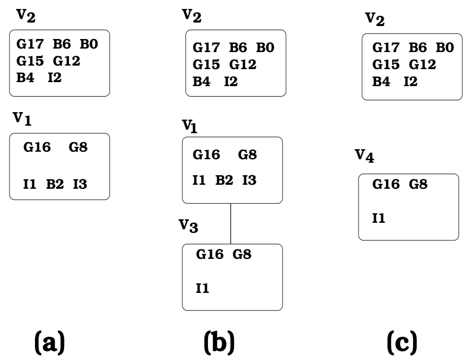

Figure 5: IG Processing for 2-Source Correctable Design

3.3 Implicit Enumeration

Given a processed IG $G = (V, E)$, the next step of our algorithm is to *implicitly enumerate* N–error line tuples from G. In this subsection, we describe such a 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 $= \{\cup_{j=1}^{n} V_j\}$. During implicit enumeration, given G, the first step is to identify the maximum number $K \geq 1$ of pairwise non–adjacent 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. N should be larger than or equal to K and, in our experiments, its value is usually found by running the algorithm for increasing values of N as long as it fails to return a valid correction(s).

Once the algorithm computes K, it selects a set $\{V_1, V_2, \ldots V_K\} \subseteq V$ of pairwise non-adjacent vertices and an n-sample for a small value of n, usually less than 5. Subsequently, it exhaustively compiles N-error line tuples placing them in C_{error}, where the j–th entry of each tuple is picked from V_j, when $1 \leq j \leq K$, and from the n-sample, when (and if) $K < j \leq N$. Finally, for every other distinct set of K pairwise non–adjacent vertices $\{V_1', V_2', \ldots, V_K'\}$ 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 a distinct V_l'.

Example 2: Implicit enumeration for the IG of Fig. 5(c) yields 21 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), $(G_8, I2)$, (I_1, G_{17}), (I_1, B_6), (I_1, B_0), (I_1, G_{15}), (I_1, G_{12}), (I_1, B_4) and $(I_1, I2)$.

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 n-sample chosen.

3.4 Error Simulation

Error Simulation (ES) is a diagnostic procedure, similar to the one presented in [7] [12], that follows implicit enumeration. It returns the *maximum* subset of C_{error} that can rectify the design for all vectors of $\mathcal{V}_{act}$. Therefore, if C_{error} does not contain valid N-error line tuples because of a poor n-sample choice, at the end of ES, C_{error} will become empty.

During ES, for every N–error line tuple $L = \{l_1, l_2, \ldots, l_N\} \in C_{error}$ and for every $v \in \mathcal{V}_{act}$, we do $2^N - 1$ 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 i-th bit of m is 1, and has complemented value, if the i-th bit of m is 0. The intuition behind ES is that a line l with complemented simulation value for v indicates a line with a design error that is excited.

If there is a vector $v \in \mathcal{V}_{act}$ such that no error excitation scenario for L yields correct primary output responses, then L gets deleted from the error list. Otherwise, L qualifies and its 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 $v \in \mathcal{V}_{act}$. This additional information will prove helpful during correction.

In our implementation, we maintain two bit lists at every line $l \in G_C$, the *Elist* and *Clist*. The i-th bit of the *Elist* for l contains the value of l when we simulate the i-th vector in $\mathcal{V}_{act}$. The bits of the *Clist* are defined similarly for the vectors of $\mathcal{V}_{test} - \mathcal{V}_{act}$. An example of the *Elist* bit list for two test vectors under simulation, v_1 and v_2, is shown in Fig. 6. These bit lists are useful during ES and correction.

Example 3: The boxes in Fig. 4 contain the values of the lines during ES for error line pair $(I_1,\ G_{15})$ and input vector $(0,0,0,0,1)$. The first entry contains the lines values for error excitation scenario $(I_1 = excited,\ G_{15} = not\ excited)$, the second entry contains the lines values when $(I_1 = not\ excited,\ G_{15} = excited)$, and the third one when $(I_1 = excited,\ G_{15} = excited)$. Observe that this procedure needs to be carried out *only* in the union of the fan-out cones of $(I_1,\ G_{15})$ and it is performed efficiently in parallel with the use of the *Elist* bit lists. Since none of these three simulations yields correct primary output responses, $(I_1,\ G_{15})$ is deleted from C_{error}.

3.5 Overall Diagnosis Algorithm

We define a **checkpoint** $B \in G_C$ to be either a primary output or a fan-out stem of G_C, and its **clan** B_{clan} 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. For example, for the circuit in Fig. 4, lines $I_3, I_2, G_9, G_{12}, O_1$ and O_2, are its checkpoints and $G_{12\,clan} = \{B_7, B_4, G_{12}\}$. It can be proven [6] that if line l, where $l \in B_{clan}$ for some checkpoint B, belongs to some valid N-error line tuple then B also belongs to some valid N-error line tuple.

With respect to Fig. 1, the overall diagnosis strategy proceeds as follows. The input to the diagnosis algorithm is F_C, G_C, the guess or the number of modifications N, a number n for the n-sample, and a maximum number of iterations $iters$ the procedure is repeated if C_{error} becomes empty after ES. First, we compile a set of different runs of the path-trace procedure, as described in subsection 3.1, with the only difference that the line sets of these vertices contain *only* lines that are checkpoints. Subsequently, we build and process the IG graph as explained in subsection 3.2 and use this information to implicitly enumerate N-error line tuples and create C_{error} according to the procedure of section 3.3. Then, ES processes the elements of C_{error}, and if C_{error} becomes empty, the whole process is repeated for a different n-sample, unless the maximum number of iterations $iters$ is reached. If such the case, diagnosis is run for a higher value of N. Finally, the N-error line tuples of checkpoints in C_{error} are replaced with multiple tuples with lines from their respective clans and ES is run on this set.

4 Correction

Once diagnosis terminates with a non–empty C_{error}, we enter the correction phase (Fig. 1).

During correction, for every N-error line tuple $L = \{l_1, l_2, \ldots, l_N\} \in C_{error}$ with error excitation scenarios $\{m_1, m_2, \ldots, m_k\}$ for vector $v \in \mathcal{V}_{act}$, 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 on the lines of L, produce for each l_i an *Elist* value for v in the set $\{2^i\ AND\ m_1, 2^i\ AND\ m_2, \ldots, 2^i\ AND\ m_k\}$, for *every* line $l_i \in L$ and *every* $v \in V_{act}$ [13]. For wire related corrections, we consider wires that do not create loops in the combinational circuit.

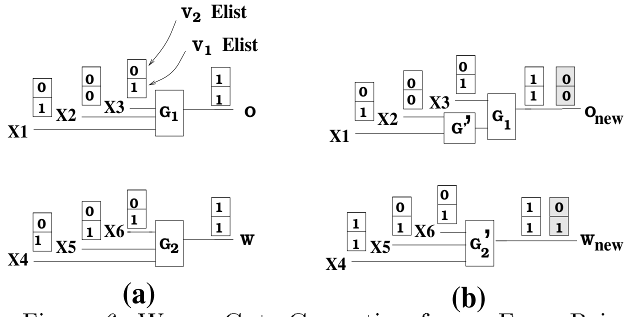

Figure 6: Wrong Gate Correction for an Error Pair

Example 4: Correction is applied on 2-error line pair $\{O, W\}$ with excitation scenarios $(1, v_1)$, $(1, v_2)$, and $(3, v_2)$, as shown in Fig. 6(a). The *Elist* bit list for the two vectors is in boxes above each line. As explained earlier, excitation configuration 1 for locations $\{O, W\}$ and vector v says that v produces correct primary output results for ES if the potential error on O is activated and the error on W is not. Equivalently, excitation configuration 3, implies that both potential errors on O and W during ES for v are activated and produce correct primary output values. 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, producing "new" gate structures driving O_{new} and W_{new}, as shown in Fig. 6(b). Assume, when we (locally) simulate these "new" gate configurations driving O_{new} and W_{new} with the use of the *Elist* bit lists on $X_1, \ldots, X_6$, we get the new simulation values in the shaded boxes. We observe that this new values respect excitation scenario 1 for v_1. The same holds for v_2 and this correction pair qualifies. Observe that if excitation scenario $(3, v_2)$ was not an output of ES, then the above correction pair would not qualify.

Once we compile exhaustively 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.

5 Experiments

We implemented the DEDC algorithm and ran it on a Sparc 10 workstation for ISCAS'85 *irredundant* circuits corrupted with 1, 2, and 3 design errors from our design error model. The types and locations where the errors were injected were selected randomly. We ran 20 experiments for each of the three different scenarios for a total of 60 experiments per circuit. The average values of the results of our experiments are reported in Table 5. All run–times are in seconds and the three rows for each circuit correspond to the 1–, 2–, and 3–source correctability case, respectively.

The second column in Table 5 contains the average size of $\mathcal{V}_{act}$ used for each different correctability case. As mentioned earlier, during the initial verification step we use a set $\mathcal{V}_{test}$ of 10,000-15,000 random and stuck-at vectors. Column 3 shows the most frequent IG type obtained. The clique case is the most computationally expensive case to handle, while a N–disconnected component IG provides the faster error resolution. We also ran our diagnosis algorithm on N-source correctable designs where $N > 3$. In these cases, the clique was the most frequent IG structure, thus, increasing the complexity for diagnosis. Although the algorithm avoided an explosion of the error space according to Eq. 1, one could possibly further improve performance by rectifying small sets of erroneous primary outputs, one at a time, as described in Lin et al. [8].

In our implementation, the algorithm begins with $N = 1$ and repeats three times if C_{error} becomes empty at the end of ES. If this is the case, it increases the value of N by 1. However, throughout our experiments, the algorithm always returned with valid corrections and no global looping of the DEDC method occured. The average number of iterations because of a bad n–sample choice is less than 1.8 and the average size of the n–sample is 3.3 vertices. The number of

Table 1: DEDC Results

crt	$\mid \mathcal{V}_{act} \mid$	IG Processing			Error Simulation		Correction		**total**	1-source		2-source	
name		IG type	# tuples	time	# tuples	time	**# tuples**	time	**time**	# vectors	hit-ratio	# vectors	hit-ratio
C499	50	—	13.1	1.1	2.1	0.9	**2.5**	6.1	**8.1**	3500	96.6%	3500	90.1%
	220	2	1722.3	1.8	16.1	4.9	**9.1**	43.9	**50.6**				
	220	2	24344.5	10.3	18.0	97.1	**11.1**	89.4	**196.8**				
C1908	80	—	2.9	1.5	2.9	3.1	**2.1**	8.7	**13.3**	4000	99.0%	4000	96.5%
	220	clique	3709.5	6.9	7.3	9.4	**4.2**	107.1	**123.4**				
	220	2	68544.9	14.4	28.1	180.6	**6.7**	212.7	**407.7**				
C5315	80	—	18.0	2.1	3.2	1.1	**2.2**	11.8	**15.0**	4500	93.5 %	6000	98.9%
	300	2	77001.2	19.2	18.0	142.9	**4.7**	138.6	**300.7**				
	300	2	77001.2	19.2	18.1	142.9	**4.9**	241.0	**403.1**				
C7552	100	—	18.0	2.1	2.9	1.3	**1.4**	14.3	**17.7**	5000	94.9%	7000	91.3%
	300	2	1311.7	4.4	13.4	7.1	**18.2**	169.9	**181.4**				
	300	2	871104.3	21.3	31.4	495.1	**21.1**	301.7	**818.1**				

N–error line tuples returned by implicit enumeration is contained in column 4 and the run–time of this procedure is in column 5. Results for ES can be found in the next two columns, confirming that our diagnosis methodology is indeed computationally efficient. Column 8 contains the number of N–correction tuples returned by the correction algorithm and the next column contains the run-times for returning *all* possible corrections. The algorithm can be modified to exit when one correction is found for a fraction of the run–times shown of column 10. Column 11 of Table 5 contains the overall time for DEDC, exhibiting the robustness of the proposed approach.

Since the corrections in column 8 guarantee to correct G_C only for the vectors of $\mathcal{V}_{test}$, it is of our 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. The correction hit–ratio is 100% for all circuits and set $\mathcal{V}_{test}$. The last four columns of Table 5 contain the correction hit-ratios for a smaller number of input test vectors. These numbers confirm the results in [2] [4] that there are hard to detect design errors.

6 Conclusion

We described a test vector simulation based approach for multiple DEDC. The approach constructs a graph and uses it to get information for diagnosis, in an effort to avoid the exponential growth of the error space with increasing number of errors. Correction is also based on the results of simulation. Experiments exhibit the efficiency and accuracy of our approach. The high correction hit–ratio and run–time efficiency obtained, indicate that DEDC through test vector simulation is indeed an attractive alternative to one based on global BDDs.

References

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

[2] M. S. Abadir, J. Ferguson and T. E. Kirkland, "Logic Verification via Test Generation," in *IEEE Trans. on CAD, vol. 7, pp. 138–148,* January 1988.

[3] M. Abramovici, P. R. Menon and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," in *IEEE Design and Test of Computers, vol. 1, pp. 89–93*, February 1984.

[4] H. A. Asaad, and J. Hayes, "Design Verification via Simulation and Automatic Test Pattern Generation," in *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design, pp. 174–180,* 1995.

[5] R. E. Bryant, "Graph–Based Algorithms for Boolean Function Manipulation," in *IEEE Trans. on Computers, vol. C–35, no. 8, pp. 677–691,* 1986.

[6] P. Y. Chung, and I. N. Hajj, "Diagnosis and Correction of Multiple Design Errors in Digital Circuits," in *IEEE Trans. on VLSI Systems, vol. 5, no. 2, pp. 233-237,* June 1997.

[7] S. Y. Huang, K. T. Cheng, and K. C. Chen, "ErrorTracer: A Fault Simulation–Based Approach to Design Error Diagnosis, " in *Proc. of IEEE Int'l Test Conf., pp. 974–981,* 1997.

[8] C. C. Lin, K. C. Chen, S. C. Chang, and M.M-.Sadowska, "Logic Synthesis for Engineering Change," in *Proc. of ACM/IEEE Design Automation Conf., pp.647– 52,* 1995.

[9] I. Pomeranz and S. M. Reddy, "On diagnosis and correction of design errors," in *IEEE Trans. on CAD, vol. 14, pp. 255–264,* February 1995.

[10] I. Pomeranz and S. M. Reddy, "On error correction in macro-based circuits," in *IEEE Trans. on Computer–Aided Design, pp. 1088–1100,* 1997.

[11] M. Tomita, T. Yamamoto, F.Sumikawa, and K. Hirano, "Rectification of Multiple Logic Design Errors in Multiple Output Circuits," in *Proc. of the Design Automation Conf., pp. 212–217,* 1994.

[12] A. Veneris and I. N. Hajj, "Correcting Multiple Design Errors in Digital VLSI Circuits," *Proc. of Int'l Symp. on Circuits and Systems,* 1999 (to appear).

[13] A. Veneris and I. N. Hajj, "Diagnosis and Correction of Multiple Design Errors with Test Vector Simulation," *IEEE Trans. on Computer–Aided Design* (submitted).

[14] S.Venkataraman, , I. Hartanto, and W.K.Fuchs, "Dynamic Diagnosis of Sequential Circuits Based on Stuck–at Faults," in *Proc. of the VLSI Test Symposium, pp.198–203,* 1996.

[15] S. Venkataraman, and W. K. Fuchs, "A Deductive Technique for Diagnosis of Bridging Faults," in *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design, pp. 562–567,* 1997.