A Hybrid Approach to Design Error Detection and Correction *

Andreas Veneris
University of Illinois
CS Department
Urbana, IL 61801
aveneris@cs.uiuc.edu

Ibrahim N. Hajj
University of Illinois
ECE Department and CSL
Urbana, IL 61801
i-hajj@uiuc.edu

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 diagnosis and employs BDDs during correction so that it remains both computational efficient and accurate. Experimental results on ISCAS'85 benchmark circuits show that our approach can typically detect and correct 2 and 3 errors within seconds of CPU time.

1 Problem Description

Design errors may occur during the design cycle 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 an erroneous gate-level description. 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.

Usually, design errors belong to a small predetermined set of possible error types, known as the **design error (correction) model**. Abadir et al. [2] presented such a model. This design error model [2] contains two major types of errors; errors that involve the functional misbehavior of *gate element(s)* (gate-replacement, extra inverter, missing gate, etc) and wire interconnection error(s) (missing wire, extra wire, etc). Experiments by Aas et al. [1] show that the design error model proposed by Abadir et al. [2] is realistic and that the average number of design errors is equal to 2. Because of its simplicity, the model of Abadir et al. [2] has been used by the majority of existing literature including the work presented here.

Throughout our presentation we assume that the only netlist available to the problem is that of the erroneous design. No information on the internal structure of the specification, which can be only simulated. For example, the specification may be available in some high-level description language. It can be now 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 [9]:

$$error\ space = (\#\ of\ circuit\ lines)^{(\#\ of\ errors)} \qquad (1)$$

Previous approaches for DEDC use either **Boolean function manipulation (symbolic)** techniques [5] [7] or **test vector simulation** [6] [8] [9] [10]. The main advantage of simulation over symbolic techniques lies in its performance and resolution during diagnosis. Nevertheless, unlike symbolic approaches, test vector simulation techniques cannot give guarantees on the quality of the corrections proposed [2][10]. On the other hand, symbolic techniques are accurate during correction but their performance degrades for diagnosis as the number of error increases.

In this paper, we describe a *hybrid* method for DEDC. Test vector simulation is used during diagnosis and BDDs [4] are employed for correction. This combination allow us to obtain both a run-time efficient and accurate algorithm. The novelty of our method is that diagnosis is performed through an *implicit enumeration* of the error space in an effort to overcome the exponential explosion with the increasing number of errors according to Eq. 1. During correction, an *error equation* with multiple unknowns is formed. A solution to this equation suggests for appropriate corrections.

The rest of this paper is organized as follows. The next two sections contain the diagnosis and correction algorithms. Section 4 presents the experimental results and Section 5 contains the conclusions.

2 Error Diagnosis

During diagnosis, the algorithm "samples" the error space to identify a set of suspicious lines without explicitly computing the complete error space. It consists of two steps, **implicit error enumeration** and **error simulation**. The procedure starts with an initial guess for the number of design errors N that it is equal to 1. If it fails to return a solution for 3 times and the same value of N, it increases the value of N by 1 and repeats. Both steps of diagnosis are based on the results of test vector simulation. In the discussion that follows, we use the symbol $\mathcal{V}_{act}$ to denote a set of input test vectors with erroneous primary output responses.

*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

During implicit enumeration, the algorithm quickly obtains an estimate of the error space. Potential erroneous signal lines are initially marked with the use of the path-trace procedure, developed by Venkataraman et al. [11]. These lines are inserted in a graph and certain graph operations direct the search for erroneous lines in the circuit. Graph processing and maintenance is discussed in detail in the following paragraphs.

Let vector $v \in \mathcal{V}_{act}$. **Path–trace** [10] starts from a failing 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.

Each set of lines, marked by a distinct run of the path-trace procedure, becomes a vertex in a graph, the **Intersection Graph (IG)**. Adjacencies in this graph are computed so that two vertices share an edge if and only if their set of lines they contain intersect.

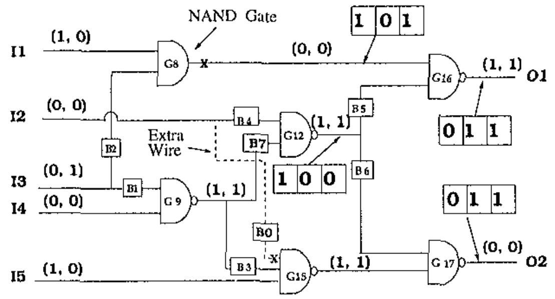

Figure 1: A Design With 2 Errors

Example 1: Consider the erroneous circuit in Fig. 1 with a gate replacement error on G_8 (NAND gate) and an extra wire B_0 (dotted line) simulated for input vectors $(1, 0, 0, 0, 1)$ 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 $V_1 = \{G_{16}, G_8, B_2, I_3\}$ when it starts 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 starts from G_{17}. The resulting intersection graph is shown in Fig. 2(a). Fig. 2(b) contains the new graph when vertex V_3 (path-trace for the second input vector from G_{16}) is introduced.

A N**-graph reduction** is the operation that replaces N pairwise non–adjacent vertices with N new vertices with smaller line sets. Graph reductions are desirable operations because they improve the error resolution. In detail, a N-graph reduction replaces a set of N non-adjacent vertices V_1, V_2, …,V_N and respective sets of adjacencies $S_{V_i} = \{V : V$

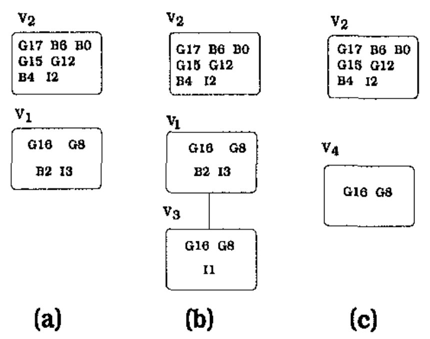

Figure 2: IG Processing for a Design With 2 Errors

adjacent to V_i and not adjacent to $V_j, i \neq j,\ 1 \leq i, j \leq N\}$, with N non-adjacent vertices V_1', V_2', …, V_N', where $line\_set(V_i') = line\_set(V_i) \cap (\cup_{V \in S_{V_i}} line\_set(V))$. For example, with respect to Fig. 2(b), a 2-graph reduction on V_1, where $V_3 = S_{V_1}$, and V_2, gives the new IG of Fig. 2(c).

The overall graph processing proceeds as follows. First, we collect a number of distinct path-trace runs, the *initial* vertices, for vectors of $\mathcal{V}_{act}$ and different erroneous primary outputs. Then we insert the initial vertices into the graph *performing as many* N*-graph reductions as (and whenever)* possible.

Given a processed graph, the last step of implicit error enumeration is to compile tuples (*i.e.* pairs, triples, etc) of suspicious lines from the graph. Each such tuple is a set of lines that are likely to contain an error. To do this, it first picks a *sample* S of lines from the graph, that is, the union of the set of lines of some randomly chosen vertices. Then, it identifies k, $1 \leq k \leq N$, non-adjacent vertices $\{V_1, V_2, \ldots, V_k\}$ in the IG. Finally, it exhaustively compiles error tuples of the form $L = \{l_1, l_2, \ldots, l_N\}$ where line l_i is from V_i, when $i \leq k$, and the sample, when $k < i \leq N$. For example, error line enumeration for the IG of Fig. 2(c) yields 14 error pairs for $k = N = 2$, namely, the cartesian product of the lines contained in $V_2 \times V_4$.

The second stage of diagnosis, **Error Simulation (ES)**, improves on error resolution. During ES, for every set of candidate erroneous lines $L = \{l_1, l_2, \ldots, l_N\}$ that is an output from implicit enumeration and for every $v \in \mathcal{V}_{act}$, we perform $m = 1 \ldots 2^N - 1$ simulations. During each such simulation, represented by the *error excitation scenario* m, l_i maintains the golden 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 potential 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, it qualifies.

Example 2: The boxes in Fig. 1 contain the values of the lines during each of the three steps of error simulation for error line pair (G_8, G_{12}), input vector $(1, 0, 0, 0, 1)$ and $N = 2$. 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*). With respect to Fig. 1, when the error simulation value is complementary to that of the golden simulation one then it is shown in a shaded box.

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}) and it can be performed efficiently in parallel if we save the golden simulation values on each of the lines of the circuit for all vectors in $\mathcal{V}_{act}$ [10].

The first error simulation corrects O_1 for the input vector but maintains the erroneous response at O_2. The other two error simulations yield erroneous responses for O_1. Since none of the three error simulations yields correct primary output responses, (G_8, G_{12}) is deleted from the error list.

3 Error Correction

Given a set of suspicious lines $L = \{l_1, l_2, \ldots, l_N\}$, the goal of correction is to propose a set of modifications from a design error model that rectifies the design. In this section, we present a symbolic technique, an extension of the results in [5], that returns corrections from the design error model of Abadir et al. [2]. Given a functional specification F_C, an erroneous circuit G_C and a set of suspicious lines L, the algorithm forms and solves an error equation with N-unknowns for L.

The **modified network** [5] G_C^X for L is obtained when every line $l_i \in L$ is disconnected from its fan-in and it is set to implement some *new* function X_i. Therefore, in the modified network, the primary outputs become functions of the primary input variables and the variables of vector $\underline{X} = \{X_1, X_2, \ldots, X_N\}$. The **error equation** of the modified network G_C^x for L is formed as follows [3] [5]:

$$\sum_{PO_i \text{ erroneous}} PO_i^{G_C^x} \oplus PO_i^{F_C} = 0 \qquad (2)$$

where $PO_i^{G_C^x}$ ($PO_i^{F_C}$) denotes the function implemented at primary outopput PO_i of the erroneous design (specification).

Example 3: The modified circuit of Fig. 3, shown in dotted lines, is erroneous. The errors are a gate replacement on line G_3 and an extra inverter on line PI_2. The function implemented at the output of the specification F_C is $PI_1 PI_2$ but the output of the erroneous design is $PI_1 PI_3 + \overline{PI_2}$. The circuitry implementing the error equation is also shown in Fig. 3. This circuit is a simple XOR function of the primary output of F_C and G_C^x. The error equation is $EQ^x(PI, \underline{X}) = (X_1 + PI_3)X_2 \oplus PI_1 PI2 = 0$.

We say that an assignment of primary input function values on $X_1, X_2, \ldots, X_N$ *satisfies* an error equation with N unknows when such an assignment evaluates the error equation to true. Intuitively, a satisfying assignment for an error equation is an assignment of values to the elements of $\underline{X}$, in terms of primary input values $PI_1, PI_2, \ldots, PI_m$, that rectify the design. In other words, a satisfying assignment for the elements of $\underline{X}$ is such an assignment of function values that makes corresponding primary outputs in G_C^x and F_C agree.

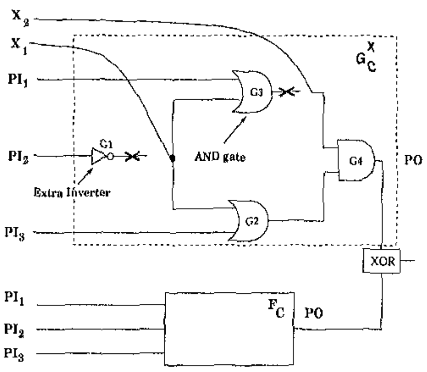

Figure 3: Error Equation

A satisfying assignment to an error equation is also called a *solution* to the error equation. The following theorem [3] provides an iterative algorithm to find such a solution, if it exists for the particular tuple L. For the rest of this section, if f and g are boolean functions on set of variables $\underline{X}$, then we say that $f(\underline{X}) \leq g(\underline{X})$ when $f(\underline{X})\overline{g(\underline{X})} = \underline{0}$ [3].

Theorem 1 *Let $EQ^x(PI, \underline{X}) = 0$ be an error equation and let the function f_N be defined as follows:*

- $f_N(\underline{X}) = EQ^x(PI, \underline{X})$
- $f_{i-1}(X_1, X_2, \ldots, X_{i-1}) = f_i(X_1, X_2, \ldots, X_{i-1}, 0) f_i(X_1, X_2, \ldots, X_{i-1}, 1)$, $1 \leq i \leq N$

The error equation has a **solution** *if and only if $f_0 = 0$ and the* **solution interval** *for the i-th variable, $1 \leq i \leq N$, is:*

$$f_i(X_1, X_2, \ldots, X_{i-1}, 0) \leq X_i \leq \overline{f_i(X_1, X_2, \ldots, X_{i-1}, 1)} \qquad (3)$$

The above method is known as the method of **successive elimination of variables** [3] since the solution interval for variable X_i, $1 < i \leq N$, depends on the solutions chosen independently for all variables X_j, where $j < i$. The example that follows illustrates the technique.

Example 4: We computed the error equation for the circuit in Fig. 3 as $EQ^x(PI, \underline{X}) = (X_1 + PI_3)X_2 \oplus PI_1 PI2 = 0$. The method of Theorem 1 gives a solution interval for $X_1 \in [PI_1 PI_2 \overline{PI_3}, 1]$. If we assign to the unknown variable X_1 the value 1 from the solution interval, then the solution interval for variable X_2 becomes $[PI_1 PI_2, PI_1 PI_2]$, that is, X_2 can only take value $PI_1 PI_2$. When unknowns X_1 and X_2 in G_C^x take these new values then the design becomes correct.

Our correction methodology proceeds as follows. For every set of suspicious lines $L = \{l_1, l_2, \ldots, l_N\}$ that qualified diagnosis, we form an error equation with N unknowns on

ckt	# of	2 design errors					3 design errors				
name	lines	$\lvert\mathcal{V}_{act}\rvert$	IE time	ES time	corr. time	# corr. returned	$\lvert\mathcal{V}_{act}\rvert$	IE time	ES time	corr. time	# corr. returned
C432	545	20	1.3	1.6	38.1	5.2	60	9.3	23.1	110.2	12.8
C499	1224	40	1.8	4.9	173.1	9.1	70	12.8	50.8	312.0	14.1
C880	880	20	1.6	2.9	68.5	1.8	60	4.5	25.5	143.9	3.2
C1908	1908	50	6.9	8.1	101.2	5.8	80	17.1	91.0	276.4	7.4
C3540	3540	70	4.9	12.3	122.7	8.4	100	18.9	211.7	475.9	13.7
C7552	7552	100	4.4	7.1	204.5	17.5	150	35.6	386.2	525.6	19.8

Table 1: Multiple DEDC Results

the lines of L and we exhaustively examine different correction schemes from the model of Abadir et al. [2]. As explained earlier, a correction that satisfies the error equation is also a correction that rectifies the design.

4 Experimental Results

We implemented our DEDC algorithm in C language and ran it on a Sparc 10 workstation with 220Mb of memory, for ISCAS'85 benchmark circuits corrupted with 2 and 3 errors. For each error case and each circuit we ran 20 experiments. The average results of these experiments are shown in Table 1. All boolean functions are expressed by Binary Decision Diagrams (BDDs) [4]. All run-times are in seconds.

Column 2 of Table 1 contains the number of circuit lines. According to Eq. 1, this number raised to the power of design errors, it indicates the size of the initial error space. To compile the set $\mathcal{V}_{act}$, we use test–vector simulation of $10,000$-$15,000$ random and stuck-at fault vectors. Column 3 (8) of Table 1 shows the size of $\mathcal{V}_{act}$ used for diagnosis during implicit enumeration and error simulation when 2 (3) errors are present in the circuit.

The next three columns contain the run-time results for the diagnosis and the correction procedures so that they return *all* possible corrections. The average size of the sample S is 3.2 vertices. It should be noted that the procedure returns one correction within a fraction of the CPU time indicated in columns 6 and 11. It can be seen from the numbers in columns 6 and 11, that our DEDC approach is indeed accurate and computationally efficient.

5 Conclusion

We propose a *hybrid* approach to multiple design error detection and correction. We use test vector simulation for diagnosis so that we obtain a computationally efficient procedure. Corrections that guarantee to rectify the design are returned through a symbolic, BDD based, technique. Experiments are used to exhibit the effectiveness and robustness of our approach.

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] F.M.Brown, "Boolean Reasoning: The Logic of Boolean Equations," *Kluwer Academic Publishing,* 1990.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] A. Veneris, S. Venkataraman, I. N. Hajj and W. K. Fuchs, "Multiple Design Error Diagnosis and Correction in Digital VLSI Circuits," in *Proc. of IEEE VLSI Test Symp., pp.58–63,* 1999.

[11] 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.