February 1993

UILU-ENG-93-2207
DAC-35

Analog and Digital Circuits

LOGIC DESIGN ERROR DIAGNOSIS AND CORRECTIONS

Pi-yu Chung, Yi-Min Wang, and Ibrahim N. Hajj

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2207 (DAC-35)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A
7a. NAME OF MONITORING ORGANIZATION Office of Naval Research	
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-90-J-1270	
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Logic Design Error Diagnosis and Correction

12. PERSONAL AUTHOR(S)

Chung, Pi-yu; Wang, Yi-Min; and Hajj, Ibrahim N.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 1991 TO 1992	93 Feb 18	31

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Design verification; design correction; design diagnosis; BDD

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

(attached)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Logic Design Error Diagnosis and Correction

Abstract

Logic verification tools are often used to verify a gate-level implementation of a digital system in terms of its functional specification. If the implementation is found not to be functionally equivalent to the specification, it is important to correct the implementation automatically. This paper describes a formal method for the diagnosis and correction of logic design errors in an incorrect gate-level implementation. We use boolean equation techniques to search for potential error locations. An efficient search and pruning algorithm is developed by introducing the notion of immediate dominator set. Two correction procedures are proposed. Gate correction corrects errors such as wrong gate type, missing inverters, etc.; line correction corrects errors such as missing wires and wrong connection. Our method is robust and covers all simple design errors described by Abadir et al. [1]. Experimental results for a set of ISCAS and MCNC benchmark circuits demonstrate the effectiveness of the proposed techniques. Circuits with thousands of gates can be corrected in minutes.

1 Introduction

The design of digital circuits usually starts from a behavioral description. At some stage in the design process, a gate-level circuit implementation is synthesized either automatically or manually according to the behavioral description. Although logic synthesis tools have been increasingly used by designers, design changes are still mostly being made manually to improve timing performance, to obtain more compact structure, or to carry out small specification changes. With the increase in circuit size and complexity, logic design errors can easily occur. It is important then to correct these errors early in the design cycle.

Logic verification has been studied intensively for finding out whether a gate-level circuit implementation is functionally equivalent to its functional specification. When the implementation is proven to be incorrect, it is necessary to diagnose and correct the design errors. This paper provides an efficient tool for accomplishing this task.

In practice, there usually exist very few errors in a design and the types of commonly encountered errors are limited. In this paper, we assume that exactly one *simple design error* (defined in the next section) exists. This design error could be an incorrect gate type, a missing or extra inverter, a missing or extra gate, a missing or extra gate input or an incorrectly placed gate input.

An overview of our work is shown in Fig. 1. First, given a functional specification and a gate-level implementation, a logic verification tool is used to examine their functional equivalence. If the two files agree, the gate-level implementation is correct; otherwise, a diagnosis procedure is performed to search for a potential error location. When a potential error location is found, two correction procedures are available for correcting the error. *Gate correction* changes the function of the gate driving the error location, and *line correction* uses another line in the circuit to drive the error location. However, a potential error location may not be an actual error location. In such cases, the attempt for correction may fail and the diagnosis and correction procedures are repeated until an actual error location is found. It is possible that the errors in the circuit do not satisfy our single error assumption. This can be concluded if none of the potential error locations can be successfully corrected.

Several previous papers have discussed this problem. Tomita et al. [2] suggest a method for correcting a circuit containing a single design error. *Input test patterns for locating logic design errors (IPLDEs)* are generated. Each time an IPLDE is applied, a set of error candidates is

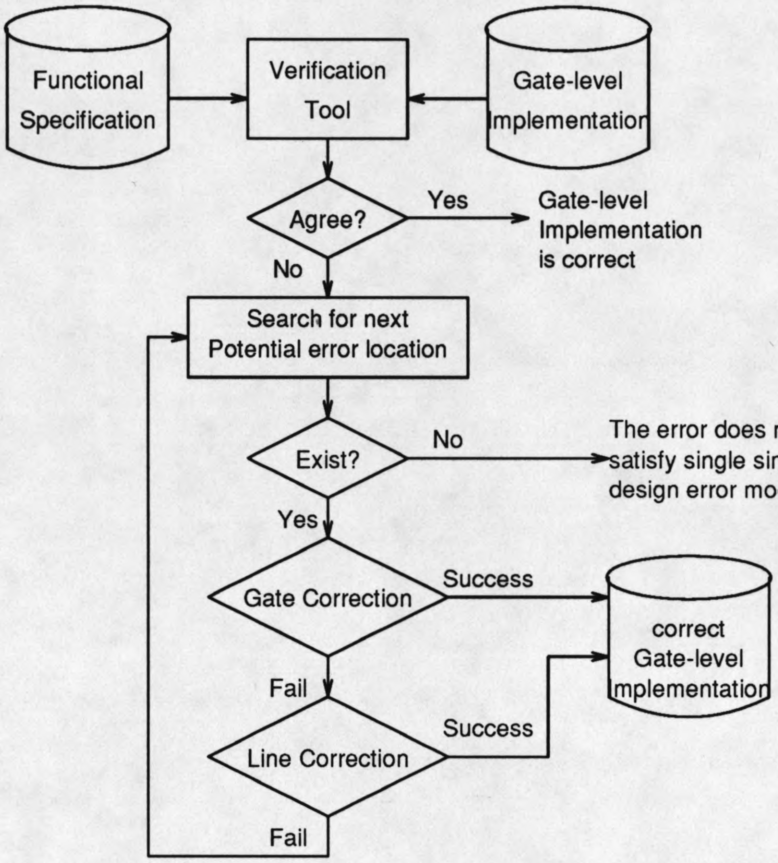

Functional Specification
Verification Tool
Gate-level Implementation
Agree?
Yes
Gate-level Implementation is correct
No
Search for next Potential error location
Exist?
No
The error does not satisfy single simple design error model
Yes
Gate Correction
Success
correct Gate-level Implementation
Fail
Line Correction
Success
Fail

Figure 1: Overview of logic design verification and correction procedures

generated. After all IPLDEs are applied, the error candidates in the intersection of all sets are considered as potential error locations. Any candidate in the intersection can be chosen to correct the circuit. However, there are two problems with this scheme. The first one is that the IPLDEs may not exist for some incorrect circuits; for example, an inverter is missing at some primary output. The second problem is that it has not yet been proven that the modification at any potential error location determined by the IPLDEs can always correct the circuit. In the work by Fujita et al. [3], the *error compensation procedure* used in the transduction method [4] is adopted. The algorithm is however not robust. In some cases, their method does not guarantee a solution. In [5, 6], formal methods for locating and correcting design errors are developed. The problem of locating errors is transformed into that of solving boolean equations. For each gate in the circuit, an equation is derived and the existence of solutions to this equation can determine as to whether the gate is a potential error location or not. However, only those errors which can be modeled as a single gate with an incorrect function are considered. Their correction techniques cannot rectify connection errors, and are in general very slow. In [7], the work by [5, 6] is extended to cover missing gate input errors and 2-input primitive gate missing errors. However, the diagnosis and gate correction procedures are still not efficient enough for large circuits.

There are two major parts in this paper, diagnosis and correction. We adopt the formal method proposed in [5, 6] for diagnosis but use a different formulation of boolean equations which allows incremental calculation of the boolean equations and provides a much stronger pruning condition for significantly reducing the search time. The gate correction procedure is based on the one suggested in [6], equipped with the following two major improvements : (1) for calculating the new gate function, implicit enumeration instead of explicit enumeration is used, so that the complexity is changed from exponential to linear (in the number of gate inputs) for most primitive gate types; (2) incremental calculation is used in the enumeration to further reduce the execution time. The line correction procedure is able to correct more types of connection errors than is previous work [7]. The procedure derives a boolean interval for the correct function at a potential error location and then search for an existing line in the circuit with function within that interval to replace the incorrect line.

In the next section, we formally describe the problem and introduce the simple design error model. Section 3 describes the potential error locations and derives the diagnosis procedure; error equations are defined and an efficient branch-and-bound search algorithm is given. Gate correction and line correction are described in section 4. Section 5 presents the experimental results for a set of ISCAS and MCNC benchmark circuits which demonstrate the efficiency of our method.

2 Problem Description

The existence of design errors is detected by verification tools [8, 1, 9]. We assume that a *functional specification* is given which defines the input and output relationship for a certain design. Suppose the design has n primary inputs, $\{x_i : 1 \leq i \leq n\}$, and m primary outputs, $\{y_i : 1 \leq i \leq m\}$. Let S denote the functional specification. S can be represented by a vector consisting of m boolean functions, i.e., $S(X) = (s_1(X), s_2(X), ..., s_m(X))$, where $X = (x_1, x_2, ..., x_n)$ and each $s_i(X)$ is the boolean function defining output y_i.

Suppose that a gate-level implementation N with n inputs and m outputs has been designed to realize S and the correspondence between the inputs (outputs) of S and the inputs (outputs) of N is given. The function F of N can be derived from the circuit structure and represented by $F(X) = (f_1(X), f_2(X), ..., f_m(X))$, where each $f_i(X)$ is the boolean function for its corresponding output y_i.

The gate-level implementation is said to be error-free if and only if $F(X) = S(X)$, or more precisely $s_i(X) = f_i(X)$ for $1 \leq i \leq n$. If there exists at least one j such that $s_j(X) \neq f_j(X)$, the circuit is considered as having logic design errors. In practice, either boolean comparisons [8], probabilistic verification [9], or test vector method [1] is used to verify the design.

The following assumptions are used in this paper:

1. Both the functional specification and the gate-level implementation are combinational circuits; or they are both synchronous sequential circuits with the same state variables and the same state assignments. By considering the inputs of flip-flops as pseudo primary outputs and the outputs of flip-flops as pseudo primary inputs, a synchronized sequential circuit can be treated as a combinational circuit [1].

2. Only one *simple design error* [1] exists in the gate-level implementation.

2.1 Simple Design Error Model

The simple logic design error model is adopted from [1]. The model includes eight commonly encountered design errors as listed in Fig. 2. *Simple gate replacement* means that a gate G is implemented by a wrong type of gate, for example, an OR gate is used for an AND operation. *Extra (missing) inverter* means that an inverter is accidentally inserted (omitted) on some line. *Simple extra gate* means that two wires are accidentally gated together before feeding to another gate. *Simple missing gate* refers to the reversed situation of *simple extra gate*. *Extra gate input* means that a wire is accidentally added from the output of a gate G_1 to the input of another gate G_2. *Missing gate input* refers to the reversed situation of *extra gate input*. *Incorrectly placed gate input* means that an input to some gate G_3 should have come from gate G_1, but is mistakenly drawn from another gate G_2. Gate G_1 in *simple extra gate* and G_2 in *missing gate input* are restricted to primitive gates (AND, OR, NAND, NOR or XOR) for simplicity. Other gates can assume any complex function.

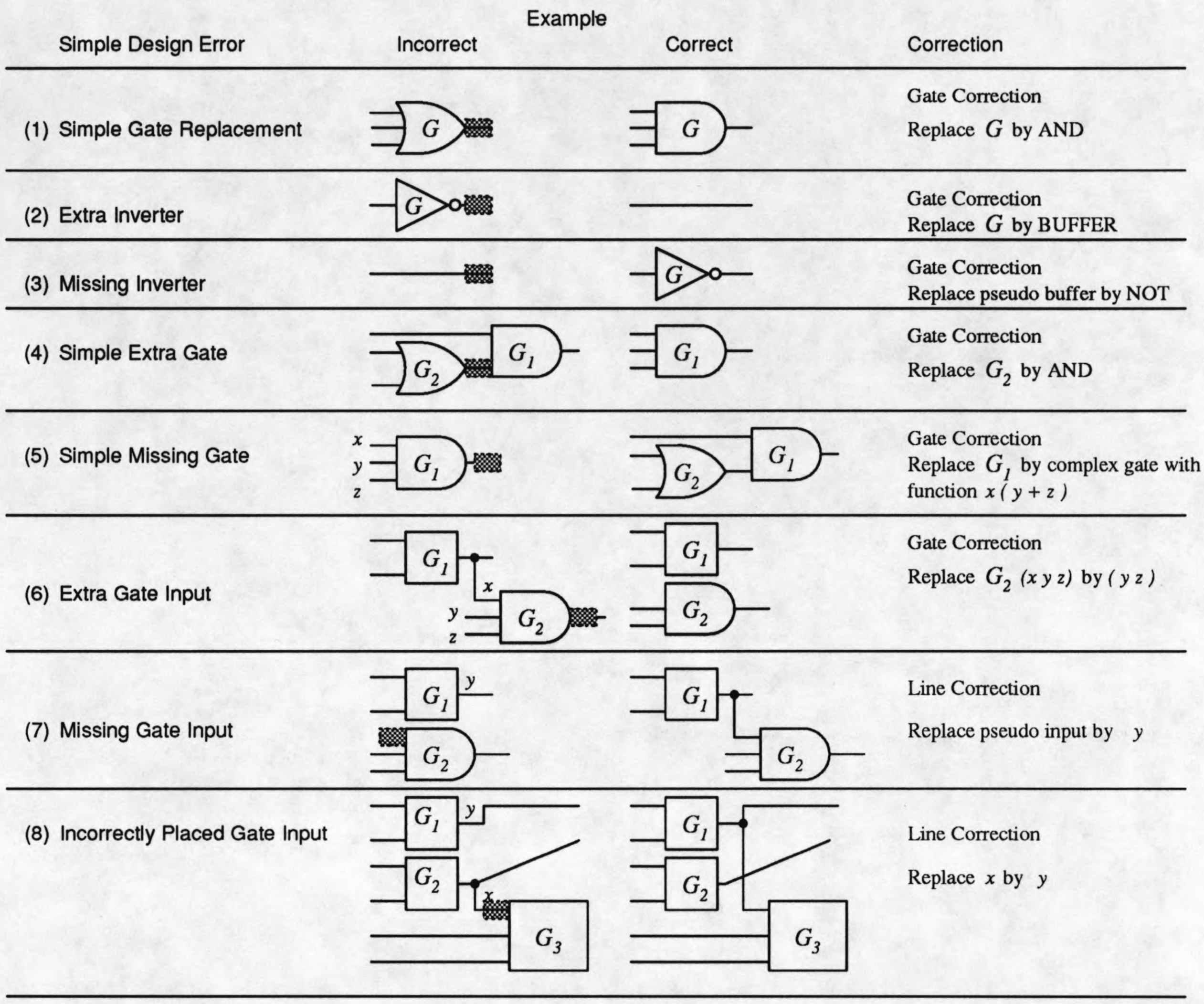


Figure 2: Simple Design Error Models

2.2 Correction Methods

We will demonstrate that every simple design error shown in Fig. 2 can be corrected by either *gate correction* or *line correction.* The shadowed area in the incorrect scenario for each design error indicates the error location.

Definition 1 *Suppose an error location L is driven by a gate G with p inputs. Gate correction replaces G with a different gate which can assume any function of the same p input variables.*

It is clear that *simple gate replacement* error can be corrected by gate correction. In the case of *extra inverter* , the inverter can be replaced by a buffer and since the existence of a buffer would not affect the functionality of the whole circuit, the buffer can then be deleted. On the other hand, we have to pretend that there is a buffer driving the error location in the case of *missing inverter*, the buffer can then be replaced by an inverter. In the case of *simple extra gate*, shown in in Fig. 2(4), if G_1 is AND or NAND, replacing G_2 by an AND gate would correct the error; if G_1 is OR or NOR, replacing G_2 by an OR gate would correct the error; if G_1 is XOR, then G_2 must be replaced by a XOR gate. In the case of *simple missing gate*, G_1 cannot be simply replaced by another primitive gate; instead, a complex gate combining G_1 and G_2 should be used. Since the complex gate use the same set of input variables of G_1, the replacement is considered as gate correction. In the case of *extra gate input*, as shown in Fig. 2(7), the correct function of G_2 depends on a subset of the original inputs, which also satisfies the definition of gate correction.

Definition 2 *Given an error location L, line correction disconnects the line to L and connects another line existing in the circuit to L.*

In the case of *missing gate input*, we pretend that every gate has an extra input, connected to ZERO if the gate type is OR, NOR or XOR, or connected to ONE if the gate type is AND or NAND. The extra input is the error location and is corrected by replacing it with y as shown in Fig. 2(7). The case of *incorrectly placed gate input* can obviously be corrected by line correction.

2.3 Error Equivalence

Since there is more than one way to synthesize a given function, it is possible that there is more than one way to model the error in an incorrect implementation, i.e., the correction can be made at different error locations. For example, in Fig. 3, the incorrect scenario of ab can be modeled as incorrectly placed gate input at either of the two locations; the incorrect scenario of $\overline{abcd}$ can be modeled as missing gate input at either of the two locations; the incorrect scenario for $a \oplus b$ can be modeled as *missing inverter* or as *simple gate replacement.* These errors are considered functionally equivalent. Our algorithm diagnoses an error to within a functional equivalence class, which means that if a designer makes a simple design error α, the error diagnosed by our scheme is functionally equivalent to α.

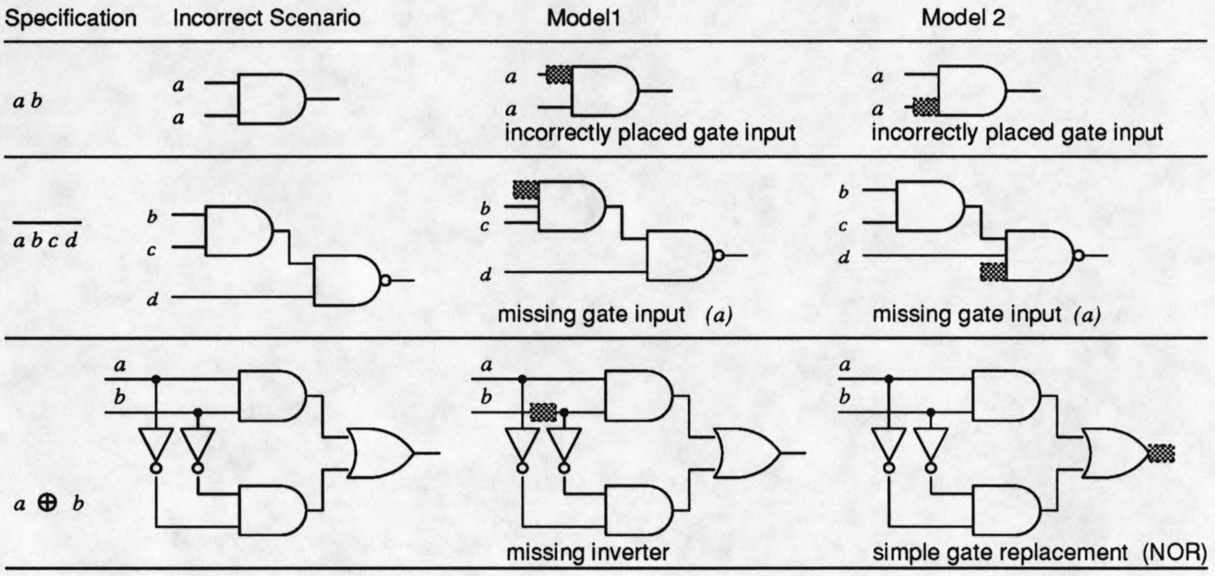


Figure 3: Equivalent errors

3 Search for Error Location

When a gate-level implementation N has been shown to be incorrect, the first step is to search for an error location. Under the single error assumption, an error location must exist within the intersection of the backtrace cones of all erroneous primary outputs, called the *suspicious area.*

Definition 3 *Given two lines p, q in N, the boolean function $f_q^p(X,z)$ is constructed as follows: (1) disconnect p from its fanouts; (2) introduce z as a new primary input and connect z to p's fanouts; (3) $f_q^p(X,z)$ is the function evaluated at q in terms of X and z. Note that f_q^p is independent of z if q is not a successor of p.*

Definition 4 *Given a line l in N, define $E^l(X,z) = \sum_{y_i \in RPO(l)}(f_{y_i}^l(X,z) \oplus s_i(X))$, where $RPO(l)$ is the set of primary outputs reachable from l. $E^l(X,z) = 0$ is called the* error equation *at line l. $E^l(X,z) = 0$ is said to be* consistent *if $E^l(X,z) = 0$ has at least one solution for z.*

For example, in Fig. 4, $f_{y_1}^b(X,z) = x_1x_3z$, $f_{y_2}^b(X,z) = \overline{x_3} + \bar{z}$, $f_{y_1}^h(X,z) = x_2x_3z$, and $f_h^c(X,z) = x_1x_2$. $E^b(X,z) = x_1x_3z \oplus s_1(X) + (\overline{x_3} + \bar{z}) \oplus s_2(X)$ and $E^h(X,z) = x_2x_3z \oplus s_1(X)$.

Lemma 1 *If l is an error location, then $E^l(X,z) = 0$ is consistent.*

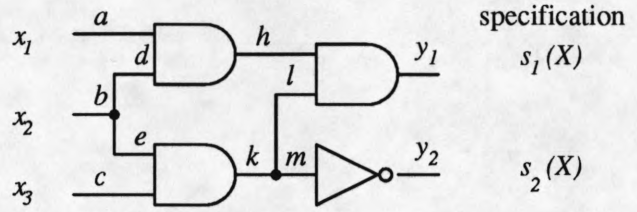


Figure 4: Example circuit

Proof. If l is an error location, let $f_l(X)$ denote the new global function at l after the correction is made. Because the function at the primary outputs after correction must be equal to the specification, we have $f^l_{y_i}(X, f_l(X)) = s_i(X)$, or $f^l_{y_i}(X, f_l(X)) \oplus s_i(X) = 0$, for all $1 \leq i \leq m$. Thus, $E^l(X, f_l(X)) = 0$ and $f_l(X)$ is a solution of z to the error equation $E^l(X, z) = 0$. □

The basic procedure for searching for the error equation can then be described as follows. An error equation is formed for each line in the suspicious area. If the equation is consistent, the line is declared as a *potential error location.*

3.1 Dominator Set

The basic search process as described above is very expensive because the lines in the suspicious area have to be examined one by one until an error location is found, and the equations are calculated by symbolic boolean function manipulation. In this section, we introduce the notion of *dominator set* which not only reduces the cost for equation calculation but also provides a strong pruning condition for reducing the search space.

Definition 5 *A dominator set of a line l is a set of lines $\{e_1, e_2, ..., e_k\}$ such that*

(1) $e_j \neq l$, for $1 \leq j \leq k$;

(2) $RPO(e_1), RPO(e_2), ..., RPO(e_k)$ form a partition of $RPO(l)$;

(3) for every primary output $y_i \in RPO(e_j)$, every path from l to y_i must pass through e_j.

For example, in Fig 4, the dominator sets of c are $\{k\}$, $\{l, m\}$, $\{y_1, m\}$, $\{l, y_2\}$, and $\{y_1, y_2\}$ and the dominator sets of b are $\{y_1, m\}$ and $\{y_1, y_2\}$. Note that h is not in any dominator set of b because there is a path from b to y_1 (b-e-k-l-y_1) not passing through h, which violates (3) in the above definition.

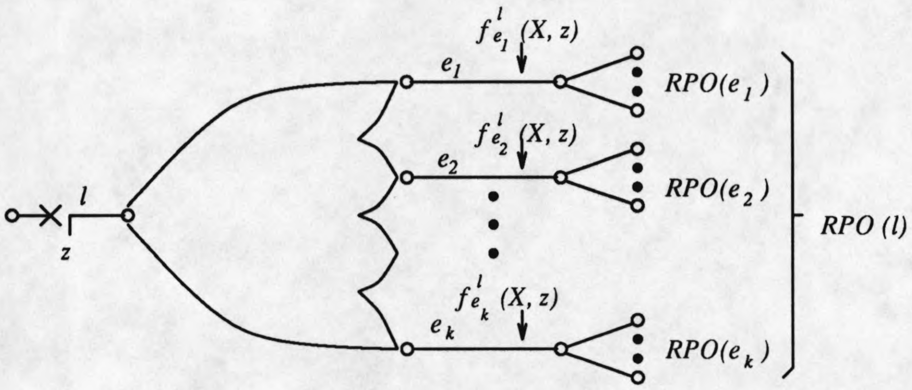


Figure 5: A dominator set of l

Lemma 2 *Let $\{e_1, e_2, ..., e_k\}$ be a dominator set of line l.*

(a) For every $y_i \in RPO(e_j)$, $f^l_{y_i}(X,z) = f^{e_j}_{y_i}(X, f^l_{e_j}(X,z))$.

(b) $E^l(X,z) = \sum_{j=1}^{k} E^{e_j}(X, f^l_{e_j}(X,z))$.

Proof. See Fig. 5. (a) Recall that $f^{e_j}_{y_i}(X,z)$ is the function at y_i with the function at e_j set to z and $f^l_{e_j}(X,z)$ is the function at e_j with the function at l set to z. Because every path from l to y_i goes through e_j, we can replace the z in $f^{e_j}_{y_i}(X,z)$ by $f^l_{e_j}(X,z)$ and obtain $f^l_{y_i}(X,z) = f^{e_j}_{y_i}(X, f^l_{e_j}(X,z))$.

$$
\begin{aligned}
\text{(b) } E^l(X,z) &= \sum_{y_i \in RPO(l)} f^l_{y_i}(X,z) \oplus s_i(X) \\
&= \sum_{j=1}^{k} \sum_{y_i \in RPO(e_j)} f^l_{y_i}(X,z) \oplus s_i(X) \quad \text{by (2) in Definition 5} \\
&= \sum_{j=1}^{k} \sum_{y_i \in RPO(e_j)} f^{e_j}_{y_i}(X, f^l_{e_j}(X,z)) \oplus s_i(X) \quad \text{by part (a)} \\
&= \sum_{j=1}^{k} E^{e_j}(X, f^l_{e_j}(X,z)). \qquad \square
\end{aligned}
$$

Note that Lemma 2(a) is not true if e_j does not belong to any dominator set of l. For example, in Fig. 4, h does not belong to any dominator set of b, so $f^b_{y_1}(X,z) = x_1x_3z$ does not equal $f^h_{y_1}(X, f^b_h(X,z))$, since $f^b_h(X,z) = x_1z$ and $f^h_{y_1}(X, x_1z) = x_1x_2x_3z$. This is because when we evaluate $f^h_{y_1}$, the function at l is x_2x_3 but it should be x_3z when we evaluate $f^b_{y_1}$.

Theorem 1 *Let $\{e_1, e_2, ..., e_k\}$ be a dominator set of line l. If $E^l(X,z) = 0$ is consistent, then $E^{e_j}(X,z) = 0$ is consistent for $1 \le j \le k$.*

Proof. If $E^l(X,z)=0$ is consistent, there exists a solution $r(X)$ such that $E^l(X,r(X))=0$. By Lemma 2(b), $\sum_{j=1}^{k} E^{e_j}(X, f_{e_j}^l(X,r(X)))=0$. Thus, $E^{e_j}(X, f_{e_j}^l(X,r(X)))=0$ for $1 \le j \le k$ and every $E^{e_j}(X,z)=0$ is consistent because $f_{e_j}^l(X,r(X))$ is a solution. □

In the following discussions, we say line l is consistent if $E^l(X,z)=0$ is consistent. By Theorem 1, a line is consistent only if all the lines in its dominator sets are consistent. This property provides a pruning strategy in the search for potential error locations. More specifically, a line cannot be a potential error location if any line in its dominator sets is shown to be not consistent.

By applying Shannon expansion to Lemma 2(b), we get

$$E^l(X,z) = \sum_{j=1}^{k} (\overline{f_{e_j}^l(X,z)} E^{e_j}(X,0) + f_{e_j}^l(X,z) E^{e_j}(X,1)) \tag{1}$$

$E^l(X,0)$ and $E^l(X,1)$ can then be obtained by substituting z with 0 and 1, respectively. Equation (1) allows an incremental calculation of $E^l(X,0)$ and $E^l(X,1)$ from $E^{e_j}(X,0)$ and $E^{e_j}(X,1)$. The calculation is carried out backwards from the primary output side to the primary input side. Once $E^{e_j}(X,0)$ and $E^{e_j}(X,1)$ are calculated, only $f_{e_j}^l(X,z)$ needs to be computed to obtain $E^l(X,z)$. At every primary output y_i, the error equation $E^{y_i}(X,z) = z \oplus s_i(X) = 0$ is clearly consistent. We can then obtain $E^{y_i}(X,0) = s_i(X)$ and $E^{y_i}(X,1) = \overline{s_i(X)}$ as the bases for the incremental calculation.

The following alternative forms of the error equation will be used throughout the paper.

Lemma 3 *$E^l(X,z)=0$ is consistent if and only if $E^l(X,0)E^l(X,1)=0$, and any function in the interval $[E^l(X,0), \overline{E^l(X,1)}]$ is a solution* [10, 11].

3.2 Immediate Dominator Set

For any line l in the circuit N, there may exist more than one dominator set. For our application, we choose to use the *immediate dominator set* for both the pruning and the equation calculation. The *immediate dominator set* of l is the dominator set which is closest to l. For any line that has no fanout branches, its immediate dominator set has only one element, i.e., the output line of its successor gate. For example, $\{i\}$ is a's immediate dominator set in Fig. 6(a). For any line that has nonreconvergent branches, its immediate dominator set consists of all of its branches. For example, $\{o_1, o_2\}$ is o's immediate dominator set in Fig. 6(a). For those lines that have reconvergent branches, their immediate dominators set can be found by using the same algorithm for the supergates[1] [14]. For example, in Fig. 6(a), the immediate dominator sets of b is $\{i_1, r, s\}$.

The relationship based on the immediate dominator set can be represented by a directed acyclic graph called the *I-DAG* and denoted by $\mathcal{I}$. Each node in $\mathcal{I}$ represents a line in N and an edge (u, v) exists if v is in the immediate dominator set of u. Fig. 6(b) shows the *I-DAG* for the circuit in (a).

3.3 The Search Algorithm

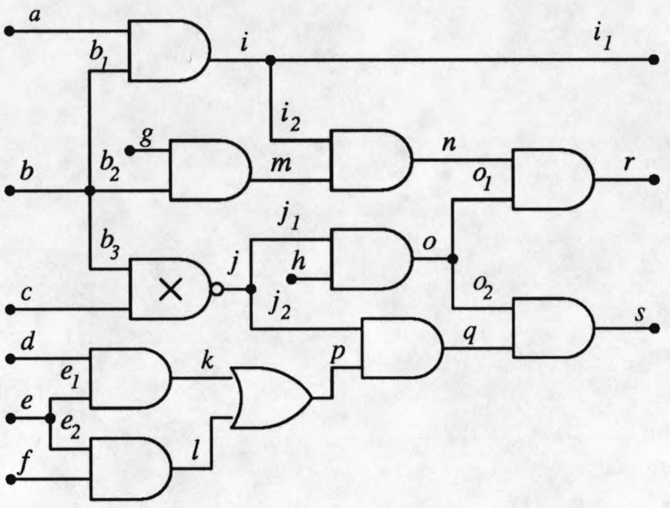
To search for potential error locations, we first find the subgraph of the *I-DAG* consisting of all the lines in the suspicious area, denoted by $\mathcal{I}_s$. A reversed depth-first-search is performed by starting from a sink node in $\mathcal{I}_s$. While visiting a node l, we first determine whether it is consistent or not; if not, the search on l's predecessors in $\mathcal{I}_s$ is pruned. The algorithm is shown in Fig. 7. The benefit of this branch-and-bound approach is that the pruning condition can reduce the search space size quickly. The search algorithm is very efficient as shown by the experimental results.

The consistency determination algorithm follows the previous discussion. If l is a primary output, then l is consistent. If any e_j in l's immediate dominator set is not consistent, then it is not. It is possible that the consistency of e_j is not checked yet at this point because e_j is outside the suspicious area or l is searched before e_j in the depth-first order. In such a case, the subroutine will be called recursively to check e_j's consistency. If all e_j's are consistent, $E^l(X, 0)$ and $E^l(X, 1)$ can then be calculated incrementally to determine l's consistency. The algorithm is shown in Fig. 8.

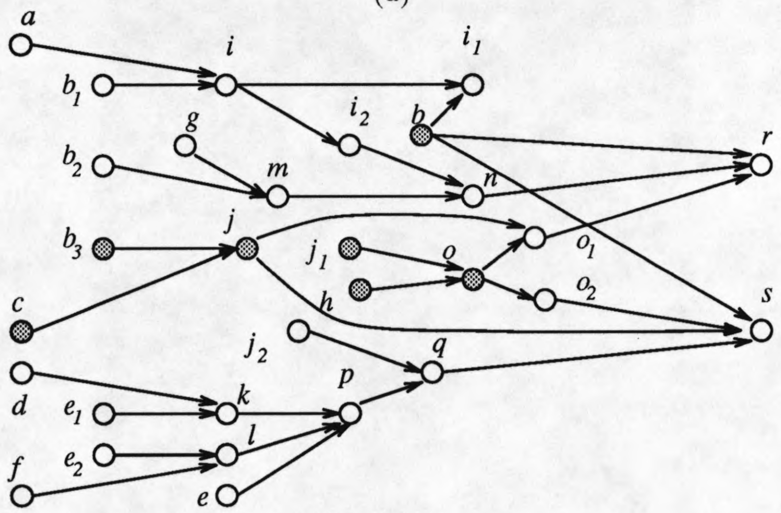
Example 1. In Fig. 6(a), let the primary input vector $X = (a, b, c, d, e, f, g, h)$. The specified functions for the three primary outputs i_1, r and s are $(ab, abcgh, bceh(d + f))$. A *simple gate replacement* error is inserted by changing the AND gate driving j to a NAND gate. The primary output functions derived from the incorrect circuit structure is $(ab, ab\bar{c}gh, (\bar{b} + \bar{c})eh(d + f))$. r and s are the erroneous outputs, the intersection of their backtrace cones consists of $\{o, j_1, h, j, b_3, c, b\}$, indicated by the grey nodes in Fig. 6(b). $\mathcal{I}_s$ is the subgraph induced by these grey nodes, in which $\{o, j, b\}$ are the three sink nodes. Fig. 9 shows the steps of finding potential error locations.

We start with checking o. Since $\{o_1, o_2\}$ is o's immediate dominator set, we first check if o_1 and o_2 are consistent. Because o_1's immediate dominator set is $\{r\}$, and r is a primary output,

[1]The difference is that all the edges must be reversed in the graph and the input nodes of a supergate of a node X in the modified graph is then the immediate dominator set of X.



(a)



(b)

Figure 6: (a)Example 1 (b) *I-DAG* and $\mathcal{I}_s$

```
**Search_Potential_Error_Locations()**
**begin**
**foreach** sink node $u \in \mathcal{I}_s$, **Depth_First_Search**($u$);
**end**

**Depth_First_Search**($u$)
**begin**
**Consistent**(u);
**if** $u$ is consistent, **then**
    /* $u$ is a potential error location */
    **if gate_correction**($u$) succeeds, **then STOP**;
    **if line_correction**($u$) succeeds, **then STOP**;
    **foreach** unvisited predecessor $v$ of $u$, **Depth_First_Search**($v$);
**else** mark predecessors of $u$ as visited;
**end**
```

Figure 7: The algorithm for searching for potential error locations

```
**Consistent**($l$)
mark $l$;
**if** $l$ is primary output $y_i$, **then**
    $l$ is consistent;
    $E^l(X,0) = s_i(X)$, $E^l(X,1) = \overline{s_i(X)}$;
    **return**;
**foreach** $e_j$ in $l$'s immediate dominator set
    **if** $e_j$ is unmarked, **then Consistent**($e_j$);
    **if** $e_j$ is not consistent, **then**
        $l$ is not consistent;
        **return**;
    /* every node in $l$'s immediate dominator set is consistent */
**foreach** $e_j$, calculate $f^l_{e_j}(X,z)$;
calculate $E^l(X,z) = \sum_{j=1}^{k} \overline{f^l_{e_j}(X,z)} E^{e_j}(X,0) + f^l_{e_j}(X,z) E^{e_j}(X,1)$;
calculate $E^l(X,0)$ and $E^l(X,1)$;
**if** $E^l(X,0)E^l(X,1) = 0$, **then** $l$ is consistent;
**else** $l$ is not consistent;
**return**;
```

Figure 8: The algorithm for determining consistency

Depth_First_Search(o)	
Consistent(o)	
Consistent(o_1)	Consistent(o_2)
Consistent(r)	Consistent(s)
r is PO r is consistent $E^r(X,0) = abcgh$ $E^r(X,1) = \overline{abcgh}$	s is PO s is consistent $E^s(X,0) = bceh(d+f)$ $E^s(X,1) = \overline{bceh(d+f)}$
$f_r^{o_1}(X,z) = abgz$ $E^{o_1}(X,z) = \overline{abgz}abcgh + abgz\overline{abcgh}$ by Equation (1) $E^{o_1}(X,0) = abcgh$ $E^{o_1}(X,1) = (\bar{c}+\bar{h})abg$ $E^{o_1}(X,0)E^{o_1}(X,1) = 0$ o_1 is consistent	$f_s^{o_2}(X,z) = (\bar{b}+\bar{c})e(d+f)z$ $E^{o_2}(X,z) = \overline{(\bar{b}+\bar{c})e(d+f)z}bceh(d+f) +$ $(\bar{b}+\bar{c})e(d+f)z\overline{bceh(d+f)}$ $E^{o_2}(X,0) = bceh(d+f)$ $E^{o_2}(X,1) = bceh(d+f) + (\bar{b}+\bar{c})e(d+f)$ $E^{o_2}(X,0)E^{o_2}(X,1) \neq 0$ o_2 is not consistent
o is not consistent	
mark j_1, h as visited (j_1, h are not consistent)	
return	

Depth_First_Search(j)	
Consistent(j)	
o_1 is consistent	s is consistent
$f_{o_1}^j(X,z) = hz$ $f_s^j(X,z) = he(d+f)z$ $E^j(X,z) = \overline{hz}abcgh + hz(\bar{c}+\bar{h})abg + \overline{he(d+f)z}bceh(d+f) + he(d+f)z\overline{bceh(d+f)}$ $E^j(X,0) = abcgh + bceh(d+f)$ $E^j(X,1) = \bar{c}abgh + (\bar{b}+\bar{c})he(d+f)$ $E^j(X,0)E^j(X,1) = 0$ j is consistent, j is a potential error location.	

Figure 9: Searching for potential error locations in example 1

and thus consistent, we conclude that o_1 is consistent after calculating $E^{o_1}(X,0)$ and $E^{o_1}(X,1)$, as shown in Fig. 9. A similar procedure shows that o_2 is not consistent, which means that o is not consistent. j_1 and h, the predecessors of o, can then be pruned.

Next, another sink node j is checked. o_1 and s are in j's immediate dominator set and they have been shown to be consistent. After calculation, we find that j is consistent and is therefore a potential error location. Because j can be corrected by the gate correction procedure (described later), the program stops. □

4 Error Correction

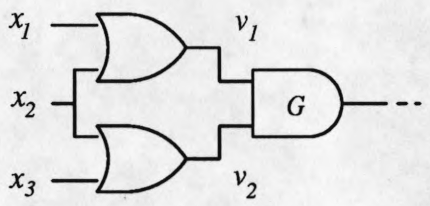
4.1 Gate Correction

If line l is a potential error location, we first try to use gate correction to correct it. Suppose l is driven by a gate[2] G and G has p fanins, $v_1, v_2, ..., v_p$. Let $f_{v_i}(X)$ denote the global function at v_i. Gate correction is successful if a new p-input function g_{new} can be obtained such that $g_{new}(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X))$ is in the interval $[E^l(X,0), \overline{E^l(X,1)}]$ as shown in lemma 3.

We illustrate the problem by the example shown in Fig. 10(a). Let $X = (x_1, x_2, x_3)$. Gate G has two fanins, v_1 and v_2, with $f_{v_1}(X) = x_1 + x_2$ and $f_{v_2}(X) = x_2 + x_3$. Suppose we want to find a function g_{new} for G such that $g_{new}(f_{v_1}(X), f_{v_2}(X))$ is in interval $[\overline{x_1}\ \overline{x_2}\ \overline{x_3},\ \overline{x_1}\ \overline{x_3}\ ]$. Fig. 10 (b) lists the truth tables of the above functions. The truth table of g_{new}, which is a two-input function, can be constructed in the following way. For (v_1, v_2) = (0, 0), the first row in (b) shows that the interval is [1, 1], so the first entry in g_{new}'s truth table is 1, as shown in (c). Similarly, the 2nd and the 3rd entries in (c) can be determined from the 2nd row and the 5th row in (b), respectively. For (v_1, v_2) = (1, 1), the interval is the intersection of the intervals of the remaining rows in (b), which is [0, 0], so the 4th entry is 0. Since the entries are (1, 0, 0, 0), g_{new} can be implemented by a NOR gate and the gate correction is successful.

In contrast, suppose we want to find a function $g'_{new}(v_1, v_2)$, such that $g'_{new}(f_{v_1}(X), f_{v_2}(X))$ is in interval $[x_1x_2x_3,\ x_1x_3]$. We can construct the truth table of g'_{new} by the same method. However, when (v_1, v_2) = (1, 1), the intersection of the interval [0, 0] for the 3rd, 4th and 7th rows and the

[2]If l is a fanout branch, we assume that a pseudo buffer drives l.



(a)

	x_1	x_2	x_3	$f_{v_1}(X) = x_1 + x_2$	$f_{v_2}(X) = x_2 + x_3$	$\overline{x_1}\ \overline{x_2}\ \overline{x_3}$	$\overline{x_1}\ \overline{x_3}$	$x_1x_2x_3$	x_1x_3
1	0	0	0	0	0	1	1	0	0
2	0	0	1	0	1	0	0	0	0
3	0	1	0	1	1	0	1	0	0
4	0	1	1	1	1	0	0	0	0
5	1	0	0	1	0	0	0	0	0
6	1	0	1	1	1	0	0	0	1
7	1	1	0	1	1	0	0	0	0
8	1	1	1	1	1	0	0	1	1

(b)

v_1	v_2	g_{new}	g'_{new}
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	?

(c)

Figure 10: Example for Gate Correction

interval [1, 1] for the 8th row is empty. Thus, we conclude that there is no way to implement g'_{new}, so the gate correction fails.

The basic concept of the gate correction is to synthesize an incompletely specified function with on-set $E^l(X,0)$, off-set $E^l(X,1)$, and dc-set $\overline{E^l(X,0)}\ \overline{E^l(X,1)}$ by all the fanins' global functions $f_{v_i}(X)$. The problem can be formally stated as follows. The function $g_{new}(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X))$ is in the interval $[E^l(X,0), \overline{E^l(X,1)}]$ if and only if [6, 11]

$$E^l(X,0)\overline{g_{new}}(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X)) + E^l(X,1)g_{new}(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X)) = 0.$$

By Shannon's expansion, we get

$$(E^l(X,0)\overline{g_{new}}(0,0,...,0) + E^l(X,1)g_{new}(0,0,...,0))\overline{f_{v_1}(X)}\ \overline{f_{v_2}(X)}...\overline{f_{v_p}(X)} \quad +$$

$$
\begin{aligned}
(E^l(X,0)\overline{g_{new}}(0,0,...,1) + E^l(X,1)g_{new}(0,0,...,1))\overline{f_{v_1}(X)}\ \overline{f_{v_2}(X)}...f_{v_p}(X) &\ + \ ... \\
(E^l(X,0)\overline{g_{new}}(1,1,...,1) + E^l(X,1)g_{new}(0,0,...,1))f_{v_1}(X)\ f_{v_2}(X)...f_{v_p}(X) &\ = \ 0
\end{aligned}
$$

The expansion can be summarized as follows.

$$\sum_{i=0}^{2^p-1}(E^l(X,0)\overline{g_{new}[i]} + E^l(X,1)g_{new}[i])P_i(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X)) = 0,$$

where $P_i(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X))$ is the *elementary product* [12], i.e., the global function of the ith minterm of g_{new}, and $g_{new}[i]$ is the ith entry in the truth table of g_{new}. $g_{new}[i]$ can be set to 0 only if the product $E^l(X,0)P_i(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X)) = 0$, which means that this minterm does not intersect the on-set. Similarly $g_{new}[i]$ can be set to 1 only if the product $E^l(X,1)P_i(f_{v_1}(X), f_{v_2}(X), ..., f_{v_p}(X)) = 0$, which means that this minterm does not intersect the off-set. If both products are nonzero, $g_{new}[i]$ cannot be assigned 0 or 1 because this minterm intersects both the on-set and the off-set, which means that no correct gate function exists and l cannot be corrected by gate correction. If both products are zero, $g_{new}[i]$ can be assigned 0 or 1 and the entry is a don't care.

By the above method, we have to calculate 2^p minterm products [6]. Instead of explicitly enumerating all 2^p minterms, implicit enumeration can be used to save computation cost. Similar to the minterms, a cube can be assigned by the same argument. Hence, instead of generating 2^p minterms directly, we generate them in depth-first order in a binary tree form as shown in Fig. 11. The gate correction algorithm is described as follows. At each node in the binary tree, two products are calculated, which are the intersections with the on-set and with the off-set, respectively, for the corresponding cube. If a cube does not intersect the on-set (or the off-set), we can assign 0 (or 1) to all the truth table entries corresponding to the cube. Therefore, a cube needs to be further divided only if it intersects both the on-set and the off-set. When such a cube is a minterm, it indicates that the correction fails. The algorithm is shown in Fig. 12.

Example 2. In example 1, we have found that j is a potential error location. Now we show how j can be corrected by gate correction. The NAND gate driving j has two fanins with global functions b and c, respectively. In Fig. 9, we have calculated that $E^j(X,0)$ and $E^j(X,1)$. The binary tree expansion and the value assignment are shown in Fig. 13. Since the truth table entries are 0, 0, 0 and 1, the correct gate type is AND. □

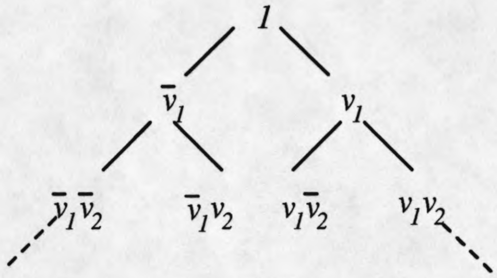


Figure 11: Binary tree for implicit enumeration

```
Gate_Correction(l)
begin
    return Assign($E^l(X,0), E^l(X,1), 0, 0$)
end

Assign($P_{on}(X), P_{off}(X), i, base$)
begin
if ($P_{on}(X) \neq 0$ and $P_{off}(X) \neq 0$), then /* intersect both on-set and off-set */
    if $i = p$, return FAIL /* the cube is a minterm */
    else begin /* divide the cube into two smaller cubes */
        $i = i + 1$
        $P'_{on}(X) = P_{on}(X)\overline{f_{v_i}(X)}$
        $P'_{off}(X) = P_{off}(X)\overline{f_{v_i}(X)}$
        if Assign($P'_{on}(X), P'_{off}(X), i, base$) fails, return FAIL
        $P'_{on}(X) = P_{on}(X)f_{v_i}(X)$
        $P'_{off}(X) = P_{off}(X)f_{v_i}(X)$
        if Assign($P'_{on}(X), P'_{off}(X), i, base + 2^{(p-i)}$) fails, return FAIL
    end
else if ($P_{on}(X) = 0$ and $P_{off}(X) = 0$), then $v = DC$ /* does not intersect on-set and off-set */
else if $P_{on}(X) = 0$, then $v = 0$ /* does not intersect on-set */
else if $P_{off}(X) = 0$, then $v = 1$ /* does not intersect off-set */
assign $v$ to truth table entries from $base$ to $base + 2^{(p-i)} - 1$
return SUCCESS
end
```

Figure 12: The Gate Correction Algorithm

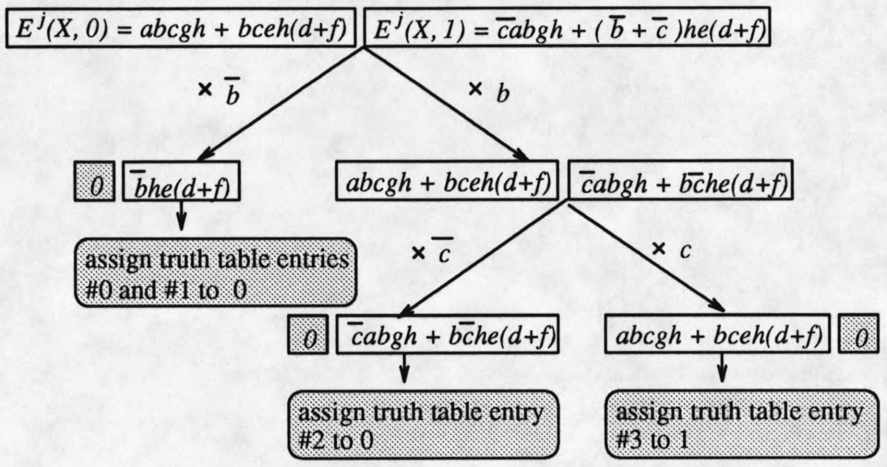

$E^j(X,0) = abcgh + bceh(d+f)$
$E^j(X,1) = \overline{c}abgh + (\overline{b}+\overline{c})he(d+f)$
$\times\ \overline{b}$
$\times\ b$
0
$\overline{b}he(d+f)$
assign truth table entries #0 and #1 to 0
$abcgh + bceh(d+f)$
$\overline{c}abgh + b\overline{c}he(d+f)$
$\times\ \overline{c}$
$\times\ c$
0
$\overline{c}abgh + b\overline{c}he(d+f)$
$abcgh + bceh(d+f)$
0
assign truth table entry #2 to 0
assign truth table entry #3 to 1

Figure 13: Gate correction for j in example 1

Example 2 shows the following advantages of our approach.

1. By using implicit instead of explicit enumeration, the complexity is reduced from exponential to linear (in the number of gate inputs) if the correct gate function is BUFFER, NOT, AND, OR, NAND or NOR.

2. The two products at each level can be used for incrementally calculating the products at the next level.

4.2 Line Correction

Line correction is used for correcting *missing gate input* and *incorrectly placed gate input*. Line correction for a line l follows immediately after gate correction for l fails (see Fig. 1). Based on the calculated solution interval for l, line correction for line l searches for any existing line in the circuit with a function falling into that interval. Because there should not be any feedback loop in a correct implementation, the only candidates are those lines not reachable from l. A successful candidate with function $h(X)$ must satisfy the following two boundary tests: (1) $E^l(X,0) \leq h(X)$ and (2) $h(X) \leq \overline{E^l(X,1)}$. Line correction checks the candidates one by one until a solution is found.

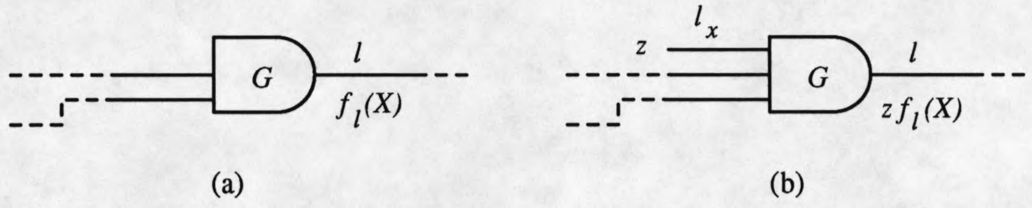


Figure 14: Test for missing gate input. (a) gate G. (b) gate G with pseudo input l_x.

4.2.1 Screening Tests for Missing Gate Input

Note that for missing gate input, we have to introduce a pseudo input as the error location. The following lemma shows that if gate G has a missing gate input, the output of G must be a potential error location. So we suspect a gate has a missing gate input only if its output is proven to be a potential error location.

Lemma 4 *If gate G has a missing gate input, then the output of G is a potential error location.*

Proof. We pretend that G has an extra input l_x, as shown in Fig. 14. Let the output of G be l. The immediate dominator set of l_x is $\{\ l\ \}$. Because l_x is the error location, l_x is consistent and l_x reaches all erroneous outputs. By Theorem 1, l is consistent. Because every path from l_x to a primary output goes through l, l reaches all erroneous outputs too, i.e., l is in the suspicious area. Hence, l is a potential error location. $\square$

The consistency of the pseudo input can be determined by the same method stated in Section 3, but the test can be further simplified.

Lemma 5 *Suppose G is an AND gate with output line l, and l is consistent. The pseudo input l_x of G is consistent if and only if $E^l(X,0) \leq f_l(X)$, where $f_l(X)$ is the function evaluated at l. The solution interval for l_x is $[E^l(X,0),\ \overline{f_l(X)} + \overline{E^l(X,1)}]$.*

Proof. G is an AND gate, so $f_l^{l_x}(X,z) = zf_l(X)$, as shown in Fig. 14.

$$\begin{aligned}
E^{l_x}(X,z) &= \overline{zf_l(X)}E^l(X,0) + zf_l(X)E^l(X,1) \quad \text{from Eq. (1)}\\
E^{l_x}(X,0) &= E^l(X,0)\\
E^{l_x}(X,1) &= \overline{f_l(X)}E^l(X,0) + f_l(X)E^l(X,1)\\
E^{l_x}(X,0)E^{l_x}(X,1) &= \overline{f_l(X)}E^l(X,0) \quad \text{because } l \text{ is consistent, } E^l(X,0)E^l(X,1) = 0.
\end{aligned}$$

Table 1: Screening Test and Solution Interval for Missing Gate Input l_x to gate G. (l is G's output line and l is consistent.)

G	Consistency Test	Solution Interval
AND	$E^l(X,0) \leq f_l(X)$	$[E^l(X,0), \overline{E^l(X,1)} + \overline{f_l(X)}]$
OR	$f_l(X) \leq \overline{E^l(X,1)}$	$[E^l(X,0)\overline{f_l(X)}, \overline{E^l(X,1)}]$
NAND	$f_l(X) \leq \overline{E^l(X,1)}$	$[E^l(X,1), \overline{E^l(X,0)} + f_l(X)]$
NOR	$E^l(X,0) \leq f_l(X)$	$[E^l(X,1)f_l(X), \overline{E^l(X,0)}]$
XOR	None(l_x is consistent)	$[\overline{f_l(X)}E^l(X,0) + f_l(X)E^l(X,1), f_l(X)E^l(X,0) + \overline{f_l(X)}E^l(X,1)]$

By Lemma 3, l_x is consistent if and only if $\overline{f_l(X)}E^l(X,0) = 0$, which is equivalent to $E^l(X,0) \leq f_l(X)$. The solution interval of l_x is then $[E^{l_x}(X,0), \overline{E^{l_x}(X,1)}] = [E^l(X,0), \overline{f_l(X)} + \overline{E^l(X,1)}]$. □

The condition $E^l(X,0) \leq f_l(X)$ in Lemma 5 is called the *screening test* and can be viewed as follows. The goal is to make the function at the output of G fall in the solution interval $[E^l(X,0), \overline{E^l(X,1)}]$. For an AND gate, if one more fanin is added, the new output function is the conjunction of the old output function and the added input function. Thus the function at the gate's output is *reduced.* If the old output function does not include the lower bound, neither does the new function. Therefore, the consistency test for AND gates checks whether the gate's function includes $E^l(X,0)$ or not, and screens out those gates not possible of having *missing gate inputs.*

The screening tests and solution intervals for the other gate types are listed in Table. 1.

Example 3. Suppose $abc + \bar{b}\bar{c}$ is the specified function, and the corresponding circuit is implemented as shown in Fig. 15(a). There are errors in the implementation because the output function of the circuit is $ab + \bar{b}\bar{c}$.

The circuit has only one output, so everything is in the suspicious area. The *I-DAG* is shown in Fig. 15(b). Using the techniques described in Section 3, the problem is solved in the following steps.

1. f is a potential error location, but cannot be corrected by gate correction or line correction.
2. We check if G_3 has a missing gate input. G_3 is an OR gate and its output function is $ab + \bar{b}\bar{c}$. Applying screening test to G_3, we get $\overline{E^f(X,1)} = abc + \bar{b}\bar{c}$ and $ab + \bar{b}\bar{c} \not\leq abc + \bar{b}\bar{c}$, so G_3 can not have a missing gate input.

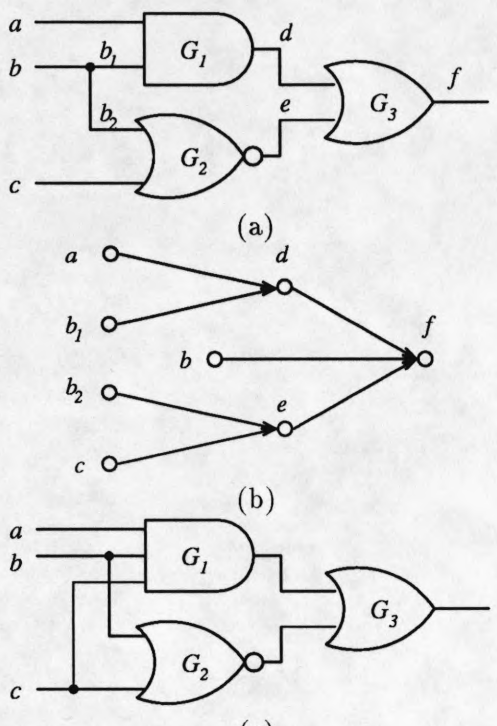


Figure 15: Example 3

3. d is a potential error location, but cannot be corrected by gate correction or line correction.

4. We check if G_1 has a missing gate input. G_1 is an AND gate and its output function is ab. $E^d(X,0) = abc$ and $abc \leq ab$, so G_1 passes the screening test. We add a pseudo input to G_1 and calculate the solution interval to be $[abc, c + \bar{a} + \bar{b}]$. The candidates for the pseudo input are $\{\ a, b, c, \bar{b}\bar{c}\ \}$. Since c is in $[\ abc,\ c + \bar{a} + \bar{b}\ ]$, the line correction succeeds. The correct implementation is shown in Fig. 15(c).

5 Experimental Results

We have implemented our diagnosis and correction algorithms by adopting *shared BDDs* [15] for symbolic boolean expression manipulation. The BDD package developed by Brace et al. [13] and the results for BDD ordering described in [16, 17] are used.

The following experiments on a set of ISCAS and MCNC benchmark circuits have been done. The test circuits are listed in Table 2. We generate the functional specifications from the given circuit descriptions. For each test circuit, we repeatedly insert a random simple design error into its gate-level implementation for 100 times. Both the error location and error type are chosen randomly. The error found and corrected is then compared to the inserted error. Based on the results, we divide the errors into three categories, namely, *exact*, *equivalent* and *redundant*. Exact error means the error found is exactly the inserted error; equivalent error means the error found is not the error inserted but is equivalent to it; redundant error means the inserted error does not change the function at any primary output and therefore does not need any diagnosis or correction. Table 3 lists the number of errors in each category. It is interesting to notice that a large percentage of errors are equivalent errors. This is because there exist many different realizations for the same boolean function. Table 3 also lists the total cpu times including diagnosis, correction and garbage collection for BDDs. All the errors are corrected in less than 8 minutes on a SPARC-II workstation. The high standard deviation is due to different BDD sizes and different search and correction costs for different error locations.

Table 4 lists the cpu times for searching for potential error locations including the calculation of error equations. The first column gives the average number of erroneous outputs. The calculation of the error equations is more expensive if more outputs are erroneous. Table 4 also lists the average number of lines visited until an error location is found in the depth-first search process. This reflects how many times the consistency checking has been performed. In general, the numbers of lines visited are very small compared to the total numbers of lines existing in the circuits. This demonstrates the effectiveness of our pruning technique.

Table 5 lists the average cpu times for correction procedures. The first two columns give the average and the maximum numbers of potential error locations found during the search. The numbers show how many times the correction procedures have been called. In most cases, the real error location is the first one or two potential error location found during the search, which means the potential error locations are very useful hints of error locations. In the worst case, the number of potential error locations found can be up to 23.

Fig. 16 shows the total time and search time for each error inserted in circuits C432, *rot* and C5315. The errors are sorted according to the total execution time. The time for gate correction and line correction contributes most to the difference between these two curves. In these three circuits, the total times for about 70 percent of the errors are less than the average time. For small

Table 2: Test circuits

Circuit	Inputs	Outputs	Gates	lines	SBDD size	Average Verification time
C432	36	7	160	432	4855	0.7
C499	41	32	202	499	42890	5.6
C880	60	26	383	880	19168	2.7
C1355	41	32	546	1355	134448	13.3
C1908	33	25	880	1908	25405	3.8
C5315	178	123	2307	5315	35088	3.4
b9	41	21	124	291	267	0.0
apex6	135	99	831	1840	1761	0.3
apex7	49	37	269	620	1011	0.1
rot	135	107	691	1640	15598	1.3
des	256	245	4679	11203	20732	4.6

Table 3: Results

	Errors			Total cpu times		
Circuit	Exact	Equivalent	Redundant	Ave.	Std. dev.	Max.
C432	53	45	2	12.7	21.7	106.7
C499	46	44	10	28.4	26.5	118.5
C880	34	64	2	4.5	7.1	30.3
C1355	51	47	2	115.7	120.0	426.1
C1908	27	71	2	22.3	25.7	110.4
C5315	38	62	0	16.6	18.3	82.6
b9	51	48	1	0.1	0.1	0.5
apex6	62	37	1	1.1	1.8	12.8
apex7	43	56	1	0.2	0.2	1.0
rot	55	45	0	2.9	3.5	13.1
des	51	49	0	35.9	53.8	278.1

Table 4: Search for error locations

Circuit	Ave. err. outputs	# lines visited		Search times	
		Ave.	Max.	Ave.	Max.
C432	4.4	11.2	97	11.9	105.7
C499	12.1	4.1	18	22.5	66.7
C880	2.6	2.8	15	3.7	30.0
C1355	18.9	11.2	37	101.3	372.4
C1908	10.9	16.3	77	18.2	95.0
C5315	7.4	6.9	27	2.9	18.5
b9	1.7	2.1	11	0.0	0.2
apex6	2.4	2.3	17	0.0	0.4
apex7	3.0	1.4	9	0.1	0.5
rot	4.9	1.9	10	1.8	12.7
des	2.3	4.8	16	0.4	2.9

Table 5: Error Correction

Circuit	# PEL found		Gate correction times		Line correction times	
	Ave.	Max.	Ave.	Max.	Ave.	Max.
C432	2.1	11	0.3	1.3	0.3	2.3
C499	1.6	16	3.7	52.2	1.0	10.3
C880	1.1	4	0.3	6.4	0.3	1.1
C1355	3.8	12	5.9	87.8	5.5	22.3
C1908	4.2	23	0.6	3.6	2.7	16.9
C5315	3.1	12	0.2	1.2	13.1	75.2
b9	1.7	5	0.0	0.2	0.1	0.3
apex6	1.7	9	0.0	0.1	1.0	12.6
apex7	1.2	6	0.0	0.2	0.1	0.5
rot	1.3	4	0.2	3.0	0.8	9.3
des	2.5	9	0.1	0.3	35.2	277.0

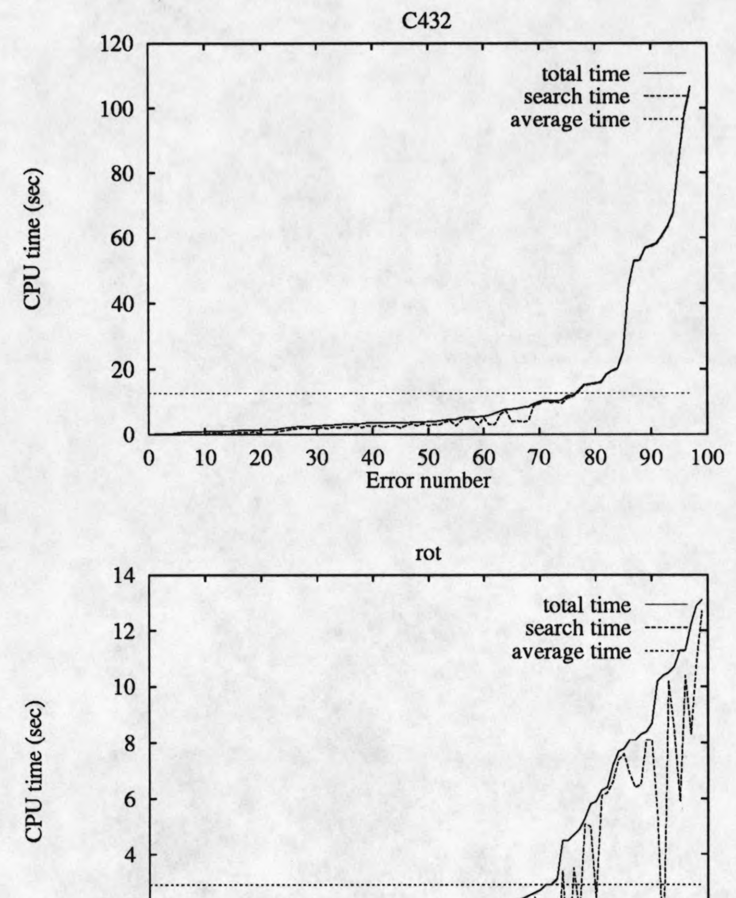


Figure 16: CPU time distribution

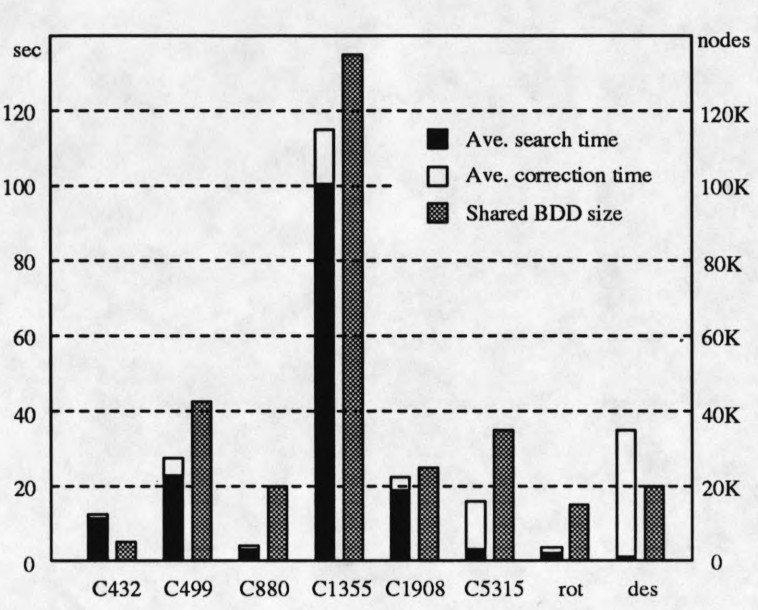


Figure 17: CPU times and BDD size

circuits such as C432, the search time dominates. For mid-size circuits such as *rot*, neither the search time nor the correction time always dominates. For large-size circuits, such as C5315, the correction time, or more precisely, the line correction time, dominates. This is because the number of lines in such a circuit is enormous, so the line correction becomes very expensive due to the large search space. Fig. 17 shows that there is a strong relationship between the cpu time and the shared BDD size.

6 Conclusions

In this paper we have presented a robust method for the diagnosis and correction of a single simple logic design error in digital circuits. The diagnosis is accomplished by an efficient search and pruning algorithm based on the notion of immediate dominator set. The gate correction is implemented by an implicit enumeration process which generates the correct truth table with reduced time complexity. The line correction was implemented by an simple search process. Experimental results on benchmark circuits have shown the effectiveness of our method.

This research is a start for the design correction problem. Future research includes the following two open problems.

1. It is possible for a design to have multiple design errors. In order to handle multiple errors, the search strategy and the error equation need modifications.

2. Some large circuits do not have feasible BDD representations. Circuit partitioning is considered as a solution to this problem. In practice, most of the circuits are designed hierarchically with each module having its own input-output specification. When a design is detected to be incorrect, it is best to verify the modules one by one. This not only reduces the size of the circuit, but also reduce the number of errors to be corrected in each verification process.

References

[1] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic design verification via test generation," *IEEE Trans. CAD,* vol. 7, pp. 138-148, Jan. 1988.

[2] M. Tomita, H.-H. Jiang, T. Yamamoto, and Y. Hayashi, "An algorithm for locating logic design errors," *ICCAD-90,* pp. 468-471, 1990.

[3] M. Fujita, T. Kakuda, and Y. Matsunaga, "Redesign and automatic error correction of combinational circuits," in *Logic and Architecture Synthesis,* ed., G. Saucier. North-Holland: Elsevier Science Publishers B. V., pp. 253-262, 1991.

[4] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method-design of logic networks based on permissible functions," *IEEE Trans. Computers*, Vol. 38, pp. 1404-1424, Oct. 1989.

[5] J. C. Madre, O. Coudert, and J. P Billon, "Automating the diagnosis and the rectification of digital errors with PRIAM," *ICCAD-89,* pp. 30-33, 1989.

[6] H.-T. Liaw, J.-H. Tsaih, and C.-H. Lin, "Efficient automatic diagnosis of digital circuits," *ICCAD-90,* pp. 464-467, 1990.

[7] P.-Y. Chung and I. N. Hajj, "ACCORD: Automatic Catching and Correction of Logic Design Errors in Combinational Circuits," *Proc. ITC-92,* pp. 742-751, 1992.

[8] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers,* vol. C-35, pp.677-691, Aug, 1986.

[9] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham, "Probabilistic Design Verification," *ICCAD-91,* pp.468-471, 1991.

[10] S. Rudeanu, *Boolean Functions and Equations,* North-Holland Publishing Company, 1974.

[11] F. M. Brown, *Boolean Reasoning,* Kluwer Academic Publishers, 1990.

[12] H. A. Curtis, *A New Approach to the Design of Switching Circuits,* D. Ban Nostrand Company, Inc., Princeton, N. J., 1962.

[13] k. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," *Proc. ACM/IEEE DAC-90,* pp. 40-45, 1990.

[14] S. C. Seth, L. Pan, and V. D. Agrawal, "Predict-probabilistic estimation of digital circuit testability," *FTCS-85,* pp. 220-225, 1985.

[15] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *DAC-90,* pp. 52-57, 1990.

[16] M. R. Mercer, R. Kapur, and D. E. Ross, "Functional Approaches to Generating Orderings for Efficient Symbolic Representations," *DAC-92,* pp. 624-627, 1992.

[17] P.-Y. Chung, I. N. Hajj, and J. H. Patel, "Efficient Variable Ordering Heuristics for Shared ROBDD," submitted to *ISCAS-93.*