Diagnosis and Correction of Logic Design Errors in Digital Circuits

Pi-Yu Chung, Yi-Min Wang, and Ibrahim N. Hajj
Coordinated Science Lab. and Dept. of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Abstract

This paper describes a method for the diagnosis and correction of logic design errors in an erroneous gate-level implementation. Our method is robust and covers more types of design errors than previous work. Our major contribution is providing significant improvement in efficiency, which is most crucial for practical applications. The notion of immediate dominator set is introduced for efficient error diagnosis. Implicit enumeration of the function space is developed for achieving fast error correction. 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

Logic verification has been studied intensively for finding out whether a gate-level circuit implementation of a design is functionally equivalent to its functional specification. Although synthesis tools have been increasingly used, design changes are still being done manually to improve timing performance, to obtain more compact structure, or to carry out small specification changes. With the increase in circuit complexity, logic design errors can easily occur. It is critical to correct these errors early in the design cycle. This paper provides a 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.

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, *Gate correction* and *line correction*, are available for correcting the error. 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 error assumption. This can be detected if no potential error location exists or none of them can be corrected.

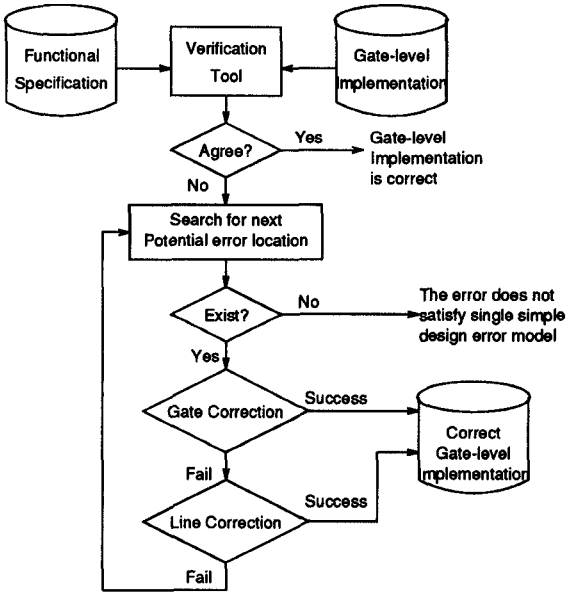

Figure 1: Overview of logic design verification and correction procedures

Several previous papers have discussed this problem [1, 2, 3, 4, 5, 6]. However, either their methods are not robust or they can only handle a small class of errors. 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. We adopt the formal method proposed in [4, 5] for diagnosis, but we use a different formulation of boolean equations which allows incremental calculation of the equations and provides a much stronger pruning condition for significantly reducing the search time. Gate correction and line correction are described in section 4. Section 5 presents the experimental results.

2 Problem Description

We assume that a *functional specification* is given which defines the input and output relationship for

This research was supported by the Joint Services Electronics Program, contract number N00014-90-J-1270.

30th ACM/IEEE Design Automation Conference®

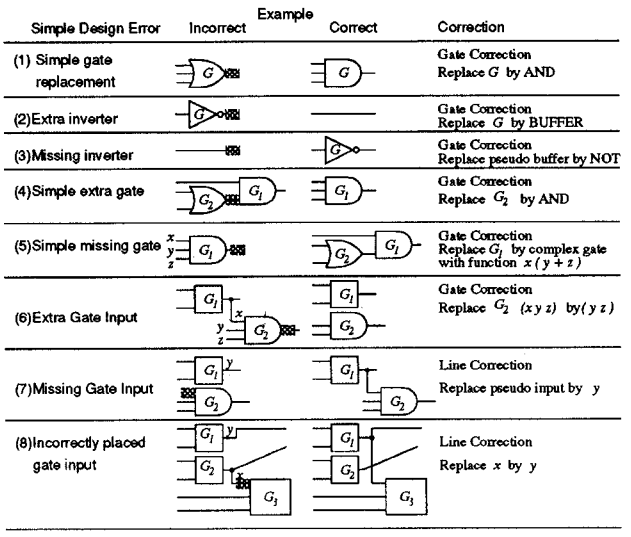

Figure 2: Simple Design Error Models

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 $s_i(X) = f_i(X)$ 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 [7] 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[1].
2. Only one *simple design error* [7] exists in the gate-level implementation. The simple logic design error model is adopted from [7]. The model includes eight commonly encountered design errors as listed in Fig. 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.

[1]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 [7].

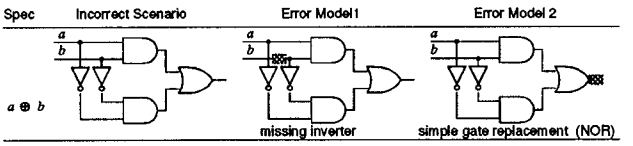

Figure 3: Equivalent errors

2.1 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 0 if the gate type is OR, NOR or XOR, or connected to 1 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.

2.2 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 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.

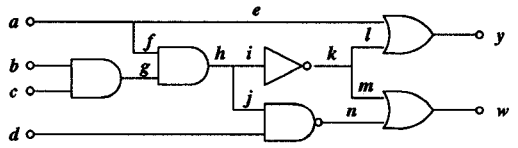

Figure 4: Example 1

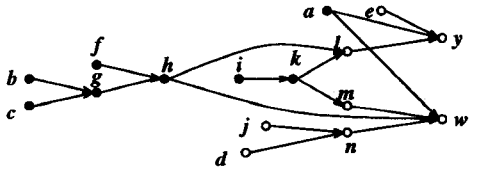

Figure 5: *I-DAG* and $\mathcal{I}_s$ of Example 1

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, let specification at (y,w) be $(a + \bar{b} + \bar{c}, \bar{a}(\bar{b} + \bar{c}) + \bar{d})$. The error equation at k is calculated as follows: $f_y^k(X,z) = a + z$, $f_w^k(X,z) = \overline{abcd} + z$. $E^k(X,z) = (a+z) \oplus (a + \bar{b} + \bar{c}) + (\overline{abcd} + z) \oplus (\bar{a}(\bar{b} + \bar{c}) + \bar{d})$.

Lemma 1 [17] *If l is an error location, then $E^l(X,z) = 0$ is consistent.*

The 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 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 k are $\{l,m\}$, $\{l,w\}$, $\{y,m\}$, and $\{y,w\}$.

Theorem 1 [17] *Let $\{e_1, e_2, ..., e_k\}$ be a dominator set of line l.*

(a) *$E^l(X,z) = \sum_{j=1}^{k} E^{e_j}(X, f_{e_j}^l(X,z))$.*

(b) *If $E^l(X,z) = 0$ is consistent, then $E^{e_j}(X,z) = 0$ is consistent for $1 \leq j \leq k$.*

In the following discussions, we say line l is consistent if $E^l(X,z) = 0$ is consistent. Theorem 1(a) allows an incremental calculation of $E^l(X,z)$ from $E^{e_j}(X,z)$. The calculation is carried out backwards from the primary output side to the primary input side. Once $E^{e_j}(X,z)$ is 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. By Theorem 1(b), a line cannot be a potential error location if any line in its dominator sets is shown to be not consistent.

The following lemma is used to determine an error equation is consistent or not.

Lemma 2 *$E^l(X,z) = 0$ is consistent if and only if $E^l(X,0)\overline{E^l(X,1)} = 0$, 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. The immediate dominator set can be found by using the same algorithm for the supergates[1] [14]. For example, The immediate dominator sets oF b, h, and k in Fig. 4 are $\{g\}$, $\{l,w\}$ and $\{l,m\}$, respectively.

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. 5 shows the *I-DAG* for the circuit in Fig. 4.

[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.

search k	
check l's consistency	check m's consistency
check y's consistenty	check w's consistency
y is PO→consistent $E^y(X,z) =$ $z \oplus (a + \bar{b} + \bar{c})$	w is PO→consistent $E^w(X,z) =$ $z \oplus (\bar{a}(\bar{b} + \bar{c}) + \bar{d})$
$f_y^l(X,z) = a + z$ $E^l(X,z) =$ $(a + z) \oplus (a + \bar{b} + \bar{c})$ $E^l(X,0)E^l(X,1) = 0$ l is consistent	$f_w^m(X,z) = \overline{abcd} + z$ $E^m(X,z) =$ $(\overline{abcd} + z) \oplus (\bar{a}(\bar{b} + \bar{c}) + \bar{d})$ $E^m(X,0)E^m(X,1) \neq 0$ m is not consistent
k is not consistent, i is pruned	

search h
l and w are consistent
$f_l^h(X,z) = \bar{z}, f_w^h(X,z) = \bar{d} + \bar{z}$
$E^h(X,z) = (a + \bar{z}) \oplus (a + \bar{b} + \bar{c}) +$
$(\bar{d} + \bar{z}) \oplus (\bar{a}(\bar{b} + \bar{c}) + \bar{d})$
$E^h(X,0)E^h(X,1) = 0$
h is consistent$\Longrightarrow$ potential error location

Figure 6: Searching for potential error locations in example 1

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 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, $E^l(X,z)$ can then be calculated incrementally to determine l's consistency.

Example 1. In Fig. 4, A *simple gate replacement* error has been inserted by changing the OR gate driving h to an AND gate. The primary output functions derived from the incorrect circuit structure is $(1, \bar{a} + \bar{b} + \bar{c} + \bar{d})$. Both outputs are erroneous. The intersection of their backtrace cones consists of $\{k, i, h, f, g, c, b, a\}$, indicated by the black nodes in Fig. 5. $\mathcal{I}_s$ is the subgraph induced by these grey nodes, in which $\{k, h, a\}$ are the three sink nodes. Fig. 6 shows the steps of finding potential error locations.

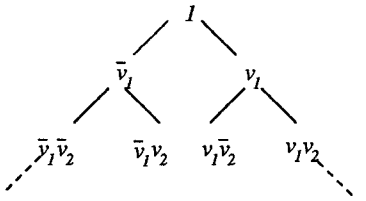

Figure 7: Binary tree for implicit enumeration

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 [1] 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 2.

The objective 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 [11, 5]

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

By Shannon's expansion, we get

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

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 [5]. Instead of explicitly enumerating all 2^p minterms, implicit enumeration can be used

[1] if l is a fanout branch, we assume that a buffer drives l.

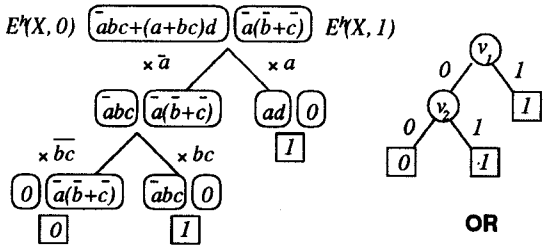

Figure 8: Gate correction for j in example 1

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. 7. 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 as shown in Fig. 7. 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.

Example 2. We show how h in Fig. 5 can be corrected by gate correction. The NAND gate driving j has two fanins with global functions a and bc, respectively. The binary tree expansion are shown in Fig. 8 and the correct gate type is OR. This example shows that 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.

4.2 Line Correction

Line correction is used for correcting *missing gate input* and *incorrectly placed gate input*. Based on the calculated solution interval for l, the line correction for line l search 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 the set of lines that are 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.

Note that for missing gate input, we have to introduce a pseudo input as the error location. In [6], it has been shown 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. The consistency of the pseudo input can be determined by the same method stated in section 3.

5 Experimental Results

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

The following experiments on several ISCAS and MCNC benchmark circuits have been done. 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 the error type are chosen randomly. The error found and corrected is then compared to the inserted error. According to 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 functions at the primary outputs and so no diagnosis and correction are necessary. Table 1 lists the number of errors in each category. It is interesting to notice that a large number of errors found are not the inserted errors. This is because there exist many different realizations for the same boolean function. Table 1 also lists the total cpu times for diagnosis, correction and garbage collection for BDDs. All the errors are corrected in less than 9 minutes on a SPARC-II workstation. The high standard deviation is due to different BDD size and different search and correction cost for different error locations.

Table 2 lists the cpu times for searching for potential error location including the calculation of the error equations. Table 2 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. Table 2 also lists the cpu times for corrections procedures. For circuits C432, C499, C880, C1355, C1908 and *rot*, the search time dominates; however, for circuits C5315 and *des*, the correction times, or more precisely, the line correction time, dominates. This is because the numbers of lines in these two circuits are enormous, so the line correction becomes very expensive due to the large search space.

6 Conclusions

In this paper we have presented a robust algorithm to diagnose and correct single logic design error in digital circuits. By using the concept of the immediate dominator set, we have developed an efficient search and pruning algorithm for error diagnosis. A new implicit enumeration process is developed for fast gate correction. A line correction technique is proposed for covering more connection errors. Experimental results on benchmark circuits have shown the effectiveness of our method.

There are two limitations to the current work. First, 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. Second, 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] K. A. Tamura, "Locating Functional Errors in

Table 1: Results

Circuit	Inputs	Outputs	Gates	SBDD size	Errors			Total cpu times		
					Exact	Equivalent	Redundant	Ave.	Std. dev.	Max.
C432	36	7	160	4855	52	46	2	12.6	21.0	104.9
C499	41	32	202	42890	48	42	10	30.3	27.2	100.4
C880	60	26	383	19168	28	70	2	4.0	6.5	24.7
C1355	41	32	546	134448	56	44	0	122.5	122.0	499.3
C1908	33	25	880	25405	26	74	0	29.5	56.8	535.2
C5315	178	123	2307	35088	27	72	1	18.9	19.5	79.0
rot	135	107	691	15598	55	45	0	2.9	3.5	13.1
des	256	245	4679	20732	51	49	0	35.9	53.8	278.1

Table 2: Search for error locations and correction times (in seconds)

Circuit	# lines visited		Search times		Gate correction		Line correction	
	Ave.	Max.	Ave.	Max.	Ave.	Max.	Ave.	Max.
C432	10.6	97	11.7	103.8	0.3	1.4	0.3	2.8
C499	4.0	18	23.6	56.4	4.0	50.6	1.4	14.8
C880	2.7	15	3.2	24.1	0.2	2.8	0.3	1.9
C1355	10.7	33	106.1	421.9	5.4	33.2	6.5	27.4
C1908	16.2	163	26.0	529.5	0.5	3.0	2.0	21.9
C5315	7.6	34	3.3	46.2	0.2	1.7	14.9	71.5
rot	1.9	10	1.8	12.7	0.2	3.0	0.8	9.3
des	4.8	16	0.4	2.9	0.1	0.3	35.2	277.0

Logic Circuits," *DAC-89,* pp.185-191, 1989.

[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] 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.

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

[6] 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.

[7] 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.

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

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

[17] P.-Y. Chung, Y.-M. Wang, and I. N. Hajj, "Logic Design Error Diagnosis and Correction," *UILU-ENG-93-2207 DAC-35,* Coordinated Science Laboratory, Univ. of Illinois at Urbana-Champaign, 1993.