Simulation-Based Design Error Diagnosis and Correction in Combinational Digital Circuits*

Debashis Nayak
Cadence Design Systems
270 Billerica Rd.
Chelmsford, MA 01824
Tel: (978) 262-6341
Fax: (978) 262-6030
Email: dnayak@cadence.com

D. M. H. Walker
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
Tel: (409) 862-4387
Fax: (409) 847-8578
Email: walker@cs.tamu.edu

Abstract

This paper describes an approach to design error diagnosis and correction in combinational digital circuits. Our approach targets small errors introduced during the design process or due to specification changes. We incrementally use simulation to identify suspect nets, and then attempt correction based on our error model. We use multiple iterations to handle multiple errors. Experimental results on ISCAS'85 benchmarks are shown for circuits containing up to four random errors. Diagnosis and correction can be done quickly, with the bulk of the time going to diagnosis. Our tool is accurate in that even with multiple errors present, the corrected circuit is identical to the original most of the time.

1 Introduction

Most digital circuits today are designed using automated synthesis tools. The resulting design may be incorrect due to software bugs in the synthesis tools. In addition, error-prone manual changes are often done to improve performance, reduce circuit size, or implement small specification changes. In order to detect erroneous implementations, formal verification tools are used to verify the equivalence between the specification and implementation, or between the original implementation and a new optimized one. When the two design descriptions are not equivalent, the verification tools provide a list of counter examples in the form of input patterns or a Binary Decision Diagram (BDD) [7] that detect the errors. The designer must then use these counter examples to locate and manually correct the errors. The verification-diagnosis-correction cycle is repeated until a correct implementation is obtained. An automated process of Design Error Diagnosis and Correction (DEDC) would greatly reduce this effort.

*This research was supported in part by the National Science Foundation under grant MIP-9406946. Debashis Nayak was a graduate student at Texas A&M University when this work was performed.

In this paper we describe a new simulation-based approach to the DEDC problem for combinational digital logic circuits (or networks) where small modifications to a small number of nets (also called lines or signals) are sufficient to correct the design. The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 introduces the basic definitions and terminology. Section 4 describes the concepts and approach of our diagnostic and correction system. The results are given in Section 5, and our conclusions are presented in Section 6.

2 Prior Work

Most combinational error diagnosis and correction approaches can be classified into two categories: (1) simulation-based approaches [10, 14, 15, 20, 21, 22, 25, 26, 30], and (2) symbolic approaches [1, 9, 16, 17, 18]. The simulation-based approaches first derive a number of erroneous vectors. By simulating each erroneous vector, the potential error region can be trimmed down gradually. The conditions for eliminating those signals that cannot be an error source vary from one approach to another. Symbolic approaches do not enumerate the erroneous vectors. They primarily rely on OBDDs to characterize the necessary and sufficient conditions of a potential error source as a boolean formula. Based on this formulation, every potential error source can be precisely identified. More recently, a method combining the above two approaches has been proposed [27, 28]. The symbolic approaches are accurate and extendible to multiple errors. However constructing the required BDD representations may cause memory explosion when applied to large circuits. On the other hand, the simulation-based approaches, although scalable with circuit size, are usually not accurate enough to handle multiple errors.

Most of the methods that consider the DEDC problem are limited in the sense that they use a restricted error model or there is no guarantee that they will find a correct solu-

tion even if one exists. Therefore, we need a tool that is applicable to large circuits and is accurate enough to rectify multiple design errors. A hybrid technique using both simulation and symbolic algorithm has been suggested [11]. But it requires the existence of a gate-level model for both the specification and the implementation with significant structural similarity. In the next section, we present a simulation-based procedure for correcting multiple design errors in large circuits, which does not rely on structural similarity between the specification and the implementation.

3 Terms and Definitions

3.1 Gate-Level Model

Let $N(G,W)$ denote a gate-level implementation of a combinational network with primitive gates $G = G_1,\ldots,G_g$ and lines $W = W_1,\ldots,W_w$ interconnecting these gates. $PO_1,\ldots,PO_n \subseteq W$ are the n primary output lines and $PI_i,\ldots,PI_m \subseteq W$ are the m primary input lines of N. A set of Boolean variables $x = (x_{PI_1},\ldots,x_{PI_m})$ is assigned to the input lines $PI_i,\ldots,PI_m$. Based on the structure of N and the primitive functions of G, each line $W_i \in W$ computes some boolean function $f_{W_i}(x)$.

A given functional specification of network N assigns to each output PO_i an a priori correct function $F_{PO_i}(x)$. Let us assume some verification technique is used to compare the specification with the implementation for a set of input patterns $V = V_1,\ldots,V_p, 1 \le p \le 2^m$. The network N is said to be correct w.r.t. V if and only if: $f_{PO_i}(V_j) = F_{PO_i}(V_j), 1 \le i \le n, \forall V_j \in V$, otherwise N is incorrect.

Definition 1: (Erroneous Vector) Given a network N which implements a set of functions $f_{PO_i}(x),\ldots,f_{PO_n}(x)$ and their functional specification $F_{PO_i}(x),\ldots,F_{PO_n}(x)$, an input pattern $V_j \in V$ is called an *erroneous vector* if $\exists$ some $PO_i \in PO$, such that $f_{PO_i}(V_j) \neq F_{PO_i}(V_j), 1 \le i \le n$.

Definition 2: (Erroneous Output) The set of Erroneous Outputs for an input vector $V_j \in V$, denoted as $Error\_PO(V_j)$ is defined as: $Error\_PO(V_j) = \{(PO_i) | f_{PO_i}(V_j) \neq F_{PO_i}(V_j), \forall PO_i \in PO\}$. Any primary output not belonging to the $Error\_PO$ set, is included in the $Correct\_PO$ set.

Definition 3: Difference Set for PO_i, denoted as $DIFF_i$ is the set of input vectors for which PO_i is an Erroneous Output. $DIFF_i = \{(V_j) | f_{PO_i}(V_j) \neq F_{PO_i}(V_j), \forall V_j \in V\}$.

Definition 4: (Counter Example) The set of *counter examples* (CEX) for the set of input patterns V is defined as: $CEX = \{(PO_i, V_j) | f_{PO_i}(V_j) \neq F_{PO_i}(V_j), V_j \in V, 1 \le i \le n\}$.

Example 2.1 Figure 1 gives an example of an erroneous network. Let us assume the design was wrongly implemented by adding an inverter at net g. Out of eight possible input vectors. only two (101 and 111) are erroneous. The outputs y and z are erroneous for both the *erroneous vectors.* The corresponding set of *counter examples* contains (y, 101), (y, 111), (z, 101) and (z, 111). Hence the *difference set* for both outputs is $\{101, 111\}$.

Definition 5: (Sensitization Set) For a line f and PO_i, the sensitization set, denoted as $SEN_i(f)$, is the set of input vectors than can sensitize a discrepancy from f to PO_i. The Boolean Difference dPO_i/df is the characteristic function of the sensitization set $SEN_i(f)$. $SEN_i(f)$ represents those input vectors for which signal f determines the value at PO_i. For the network in example 2.1, $SEN_1(h) = \{010, 101, 100, 110, 011, 111\}$ and $SEN_2(m) = \{001, 101\}$.

A line f is called a *sensitizable net* iff $SEN_i(f)$ is not an empty set $\forall i$. The nets *sensitizable* for some of the counter examples constitute the *suspect list.* In example 2.1, only nets g, h and i are sensitizable by all the four counter examples and so form the best suspects.

Definition 6: Current value $CV(G,V_j)$ is the *current value* at the output of a gate G, when the pattern V_j is applied to the implementation.

Definition 7: Required Value at a gate output. Let $V_j \in V$ be an erroneous vector and $PO_i \in PO$ be an erroneous output, when the pattern V_j is applied to the primary inputs ($PO_i \in Error\_PO(V_j)$). The *required value* at the output of a gate G, $RV(G,PO_i,V_j)$ is the value required at G to make $PO_i \in Correct\_PO(V_j)$. In example 2.1, for erroneous vector 101, $CV(i,101) = 0$. The vector will not remain erroneous if the value of line i is complemented. Hence, the required value of line i for vector 101 is 1.

Definition 8: Observed Signature Bit-list(OSB) at a gate output is a list of *current values* at the gate output for the given set simulated vectors. It is just a binary code of length j, where j is the number vectors simulated.

Definition 9: Required Signature Bit-list(RSB) at a gate output is a list of *required values* at the gate output for the given set simulated vectors. The RSB of any internal line, gives the set of values necessary to rectify the network for the given set of input vectors.

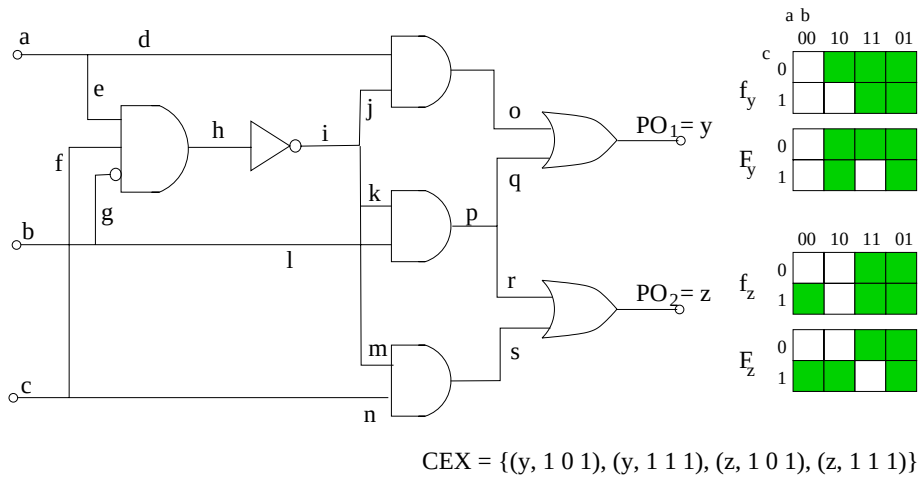
3.1.1 Correctability

Definition 10: (Correctable Vector) An erroneous vector $V_j \in V$ is *correctable* by signal f if there exists a new function for signal f such that V_j is *not* an erroneous vector for the resulting new circuit.

Proposition 1: Let $V_j \in V$ be an erroneous vector and f be a line in the circuit. Then V_j is correctable by f if and only if the following two conditions are satisfied:

- V_j can sensitize a discrepancy from f to every erroneous primary output in N, i.e., for every PO_i in $Error\_PO(V_j)$, $V_j \in SEN_i(f)$.
- V_j *cannot* sensitize a discrepancy from f to any correct primary output, i.e., for every PO_i in $Correct\_PO(V)_j$), $V_j \notin SEN_i(f)$.

(Proof): Let F be a boolean function that disagrees with the original function of f only on input vector v. Then after re-



CEX = {(y, 1 0 1), (y, 1 1 1), (z, 1 0 1), (z, 1 1 1)}

Figure 1: Example of erroneous network (inverter incorrectly added at net g).

placing signal f with the new function F, a discrepancy is injected at f. If the above two conditions are satisfied, then the response of every erroneous output toggles and thus becomes correct (because *v can* sensitize a discrepancy from f to every one of them). At the same time, every originally correct output remains correct (because vector *v* cannot sensitize a discrepancy from f to any of them). On the other hand, it can be shown that if the above two conditions are not satisfied, then at least one erroneous primary output will remain erroneous, or one originally correct output will become erroneous. (Q.E.D)

Definition 11: (Single Signal Correctable) If the circuit N can be completely corrected by re-synthesizing a signal f, then N is single-signal correctable, and the signal f is called a *single-fix signal*.

3.2 Single Error

Given a network N if there exists an internal line f s.t. $SEN_i(f)$, covers $DIFF_i$, then N can be fixed by changing the function of f. On the other hand, if the sensitization set of f does not cover $DIFF_i$, then f is not a single-fix signal.

The new function for f (*required value* at f) in order to fix N should satisfy two conditions: (1) it should *disagree* with the original function of f for input vectors in the difference set, $DIFF_i$, and (2) it should *agree* with the old function of f for input vectors in the sensitization set but not in the difference set, $(SEN_i - DIFF_i)$. Let the on-set (off-set) of the original function of f be $f^{ON}(f^{OFF})$. Then, we can represent the on-set and off-set of the incompletely specified new function for f as follows:

$$f^{new-ON} = (f^{OFF} \cap DIFF_i) \cup (f^{ON} \cap (SEN_i - DIFF_i))$$

$$f^{new-OFF} = (f^{ON} \cap DIFF_i) \cup (f^{OFF} \cap (SEN_i - DIFF_i))$$

For circuits with multiple outputs, each primary output would impose a constraint on the on-set and off-set. These constraints should be combined together to derive the final on-set and off-set of the new function.

Proposition 2: (*Necessary and sufficient conditions for single fix signal*) [9, 17, 18] Let i be the index of the primary outputs, and f be an internal signal of the implementation. The set of input vectors f^{new-ON} and $f^{new-OFF}$ are defined as follows.

$$f^{new-ON} = \bigcup_i (f^{OFF} \cap DIFF_i) \cup (f^{ON} \cap (SEN_i - DIFF_i))$$

$$f^{new-OFF} = \bigcup_i (f^{ON} \cap DIFF_i) \cup (f^{OFF} \cap (SEN_i - DIFF_i))$$

If the intersection of f^{new-ON} and $f^{new-OFF}$ is empty, then f is a single fix signal.

Definition 12: We define *Correction* as a functional change or modification on some line in the circuit.

Definition 13: The *Rectification* of a logic network is a series of *changes (corrections)* to make the network functionally equivalent to its specification.

Definition 14: The *Correctability* of a suspect line is its potential to reduce the number of counter examples.

Definition 15: We define the *accuracy* of a rectification algorithm as its ability to rectify a logic network without making any radical changes to the network structure. Thus, accuracy is directly related to the the number of corrections made in the rectification process. Too many corrections may cause the rectified circuit to look completely different from the original circuit. Even when a network is rectified in a minimum number of steps, it is not guaranteed that the rectified network will look similar to the original network. Due to the existence of mutually equivalent corrections, there is no way a tool can determine which correction

is better. We measure the accuracy of a rectification tool by the number of equivalent lines between the original and rectified circuits, which is a good estimate of similarity.

3.3 Multiple Errors

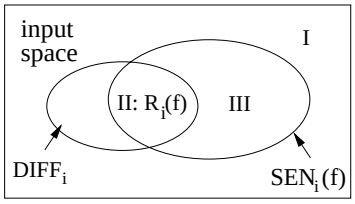
For circuits with multiple errors, the algorithm based on the single signal correctable condition may not be applicable. To overcome this limitation, we first define the term *correctable set* and then formulate a partial correction condition in the following.

Definition 16: The *Correctable Set* for an internal signal f w.r.t. a primary output PO_i denoted as $R_i(f)$, is the intersection of its corresponding sensitization set $SEN_i(f)$, and difference set $DIFF_i$. Intuitively, the correctable set $R_i(f)$ represents the *maximal set of difference vectors* w.r.t. PO_i that can be corrected by changing the function of signal f.

Definition 17: Partial Correction: A signal f is called a *partial-fix signal* if there exists a new function at f s.t.: (1) no new difference vector is created for any primary output, and (2) the difference set for at least one primary output is *reduced*.

Based on this definition, the rectification process can be viewed as a sequence of partial corrections. During this process, the difference set w.r.t. each primary output is monotonically shrinking until it is empty. Each partial correction consists of finding a partial fix signal and synthesizing the new function to replace the old function.

Definition 18: Strong partial correction: A signal f is called a *strong partial-fix signal* if there exists a new function at f such that the following two conditions are satisfied for each primary output: (1) no new difference vector is created, and (2) the *correctable set* for every primary output is reduced to the empty set.



Region I : ~($SEN_i(f)$)

Region II : $R_i(f)$

Region III : $SEN_i(f)$ - $R_i(f)$

Figure 2: Requirements of a strong partial fix signal.

Figure 2 illustrates the different regions in the input space: (I) *don't care region,* (II) *fix region,* and (III) *don't touch region.* For every primary output in the implementation these constraints should be combined together. The necessary and sufficient condition is given as follows.

Proposition 3: *(Necessary and sufficient condition for strong partial correction)* Let i be the index of the primary outputs, and f be an internal signal in the implementation. The set of input vectors f^{new-ON} and $f^{new-OFF}$ are defined as follows.

$$f^{new-ON} = \bigcup_i ((f^{OFF} \cap R_i) \cup (f^{ON} \cap (SEN_i - R_i)))$$

$$f^{new-OFF} = \bigcup_i ((f^{ON} \cap R_i) \cup (f^{OFF} \cap (SEN_i - R_i)))$$

If the intersection of f^{new-ON} and $f^{new-OFF}$ is empty, then f is a strong partial fix signal, and the new function is an incompletely specified function defined by f^{new-ON} and $f^{new-OFF}$.

3.4 Error Model

Many types of design errors have been classified in the literature. These error types are not necessarily complete, but they are believed to be common in the design process. We condense the errors identified by Abadir et al. [1] into four categories:

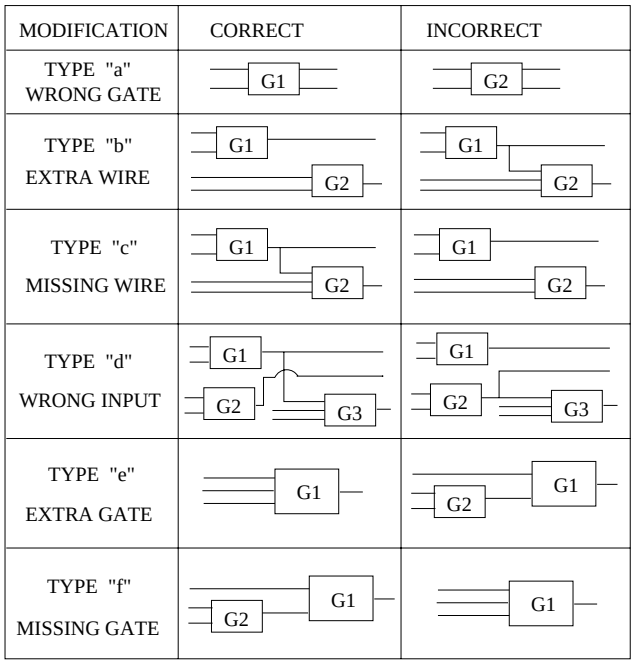


Figure 3: Error model.

- *Wrong Gate (WG):* mistakenly replacing one gate type by another gate type with the same number of inputs (Figure 3, type "a"). Extra and missing inverters are considered a substitution of an inverter for a buffer and vice-versa.
- *Extra/Missing Wire (EW/MW):* using a gate with more or fewer inputs than required (Figure 3 types "b" and "c").
- Wrong Input (WI): connecting a gate input to a wrong signal (Figure 3 type "d"). A WI may be viewed as a

combination of EW and MW, but this cannot model a WI in an inverter.

- *Extra/Missing Gate (EG/MG):* incorrectly adding or removing a gate (Figure 3 types "e" and "f"). This category is combined with gate substitution in [8], where unlike here, XOR and XNOR gates are not considered.

Sometimes the term *Misplaced Wire* is used to denote *EW*, *MW* and *WI* errors collectively.

4 Simulation-Based Error Correction

Our approach to the DEDC problem uses a test simulation procedure to identify potential modification locations. Correction is based on the results of test simulation together with Boolean function manipulation. Unlike some other error correction methods [1, 27, 28], our approach does not assume that only one net can be the source of error.

Once a design has been verified to be erroneous, we apply test vectors to the functional specification, and the gate-level implementation of the design, that produce erroneous primary output responses. These vectors provide information on the location of possible candidate modifications. A number of observations listed in this section, allow us to greatly reduce the set of candidate modification locations in a fast and efficient manner. The correction phase of our algorithm returns a set that contains a set of possible modifications, if they exist in our logic design error model, that rectify the design.

Simulation-based design error correction is not guaranteed to be complete unless the vector set is exhaustive. In addition, since the actual error may not consist of one or more errors in our error model, the correction may be invalid. Therefore, after correction, the circuit must be reverified against the original. We will discuss this further in Section 5.

4.1 Candidate Selection

The first step of our algorithm involves the identification of partial-fix signals for making corrections. We use a simulation-based diagnosis procedure suitable for generating a suspect list in the presence of multiple errors. In this situation, making a correction can introduce new erroneous vectors while removing others. Therefore, it is essential not to discard a suspect net even if it is not capable of removing all erroneous vectors. As defined earlier, the *Required Value* of a suspect net determines the type of correction needed to remove an erroneous vector. The following definitions will help us locate the best candidates for error correction.

Definition 19: The *Required Value* of a line l for V_j and PO_i is related to the *Current Value* as:

- *Required Value = Current Value*, if l is sensitizable at PO_i and PO_i is not an *erroneous output*.
- *Required Value* = NOT *Current Value*, if l is sensitizable at PO_i and PO_i is an *erroneous output*.
- *Required Value* = DONTCARE, if l is not sensitizable at PO_i for V_j.

Definition 20: The *correctability measure*, $CM^{ij}(l)$, of a line l for V_j and PO_i is defined as:

- $CM^{ij}(l) = 0$, if *Required Value* = *Current Value* OR *Required Value* = DONTCARE.
- $CM^{ij}(l) = 1$, if *Required Value* = NOT *Current Value*.

Definition 21: We define the *accumulated correctability measure* $ACM(l)$ of a line l over all primary outputs, and all vectors as:

$$ACM(l) = \sum_{V_j} \sum_{PO_i} CM^{ij}(l)$$

Intuitively, $ACM(l)$ denotes the potential of l to change the incorrect circuit closer to the correct one. A line with positive *ACM* is capable of reducing the number of counter examples. A line with *ACM* equal to the number of counter examples is capable of completely correcting the circuit. The sensitizable lines for any vector can be computed using the *critical path tracing* algorithm [3].

Theoretically, any line sensitizable for any counter example is a possible correction candidate. But the correctability of each line varies with their *ACM* value. We sort the candidate lines in decreasing *ACM* order and start applying correction to the line with maximum *ACM*. In case of a failure, we move down the list.

Since there is more than one way to synthesize a given function, there may be more than one way to model the error in an incorrect implementation, i.e. the correction can be made at different error locations. Our algorithm diagnoses errors to within a functional equivalent class, and chooses the suspects closest to the inputs for correction.

4.2 Single Error Correction

Every simple design error in the error model can be corrected by modifying either a gate or a line. We apply the following five types of corrections for single error rectification: (a) change gate type, (b) add extra gate, (c) replace wrong wire, (d) add extra wire, (e) remove extra wire.

As defined in Section 3, the *correctability* of a line is its ability to reduce the number of counter examples. Let there be m counter examples for an implementation which changes to n after the correction is made on line l. Then, $correctability(l) = m - n$, iff $m > n$, otherwise 0. A single-fix signal is capable of reducing the number of counter examples to zero. Hence, a single-fix signal has the maximum possible *correctability*, which is same as the number of counter examples (Proposition 2).

In order for a line l to satisfy Proposition 2, it should be possible to synthesize the new function at l by applying one or more of the five simple modifications described above. Let the old and new function at l be given by $f_l(x)$ and $f_l^{'}(x)$. The function $f_l(x)$ corresponds to the *Current Value* and $f_l^{'}(x)$ should correspond to the *Required Value* at line l. Our aim is to choose a correction whose $f_l^{'}(x)$ best matches the required values of line l for all input vectors. This is implemented by an efficient cube-matching algorithm that also handles the don't care conditions. This is implemented by matching signature bit-lists as described below.

We examine each suspect line one by one (starting with the suspect with highest ACM value closest to the inputs), and check, if it is possible to synthesize the new function at l by changing the type of gate that feeds l. The gate retains the same number of inputs. If successful, the algorithm reports the correction and terminates. Otherwise, we look for a misplaced wire. For Extra Wire (EW) correction, we look for a fanin of l, which can be removed to make l satisfy Proposition 2. For Missing Wire (MW) correction, we consider every line l' that can become an extra fanin to line l without creating any asynchronous cycles. In Wrong Wire (WI) correction, the line l' replaces one fanin of l.

In order to reduce the search space and perform a missing wire search efficiently, we use the *observed/required signature bit-list*(OSB/RSB) compiled during the vector simulation. In the presence of a large number of suspect lines, the task of matching the OSBs to RSB for every possible correction could be time consuming. But fortunately, most suspects can be eliminated by matching only a few bits. Because of this, a linear search through the bit-lists takes less time than constructing a hash table. Substantial performance gains can also be achieved by using more sophisticated pattern matching algorithms [4].

If both Wrong Gate and Misplaced Wire correction types fail to provide a solution, we look for an Extra/Missing Gate type of Error. Here, we try to synthesize a line from the existing lines in the network, making use of a new gate to match the required bit-list of l. In our current implementation, the only new gate type we consider is an inverter, so we look for lines whose current bit-list is the complement of the required bit-list for l, via bit-list comparison

If the algorithm does not find a solution after trying all of the above corrections, the network N is not a *single-fix* type. In that case, we need to apply a sequence of corrections as described below.

4.3 Incremental Rectification

As shown in Figure 4, the rectification of circuits with multiple errors is an iterative process. In every iteration, all the *partial fix* signals are listed as possible correction candidates. Then, we sort the suspects by their correctability (using *ACM* values) and try correcting the suspect with highest correctability. We try all possible types of *gate* and *wire*

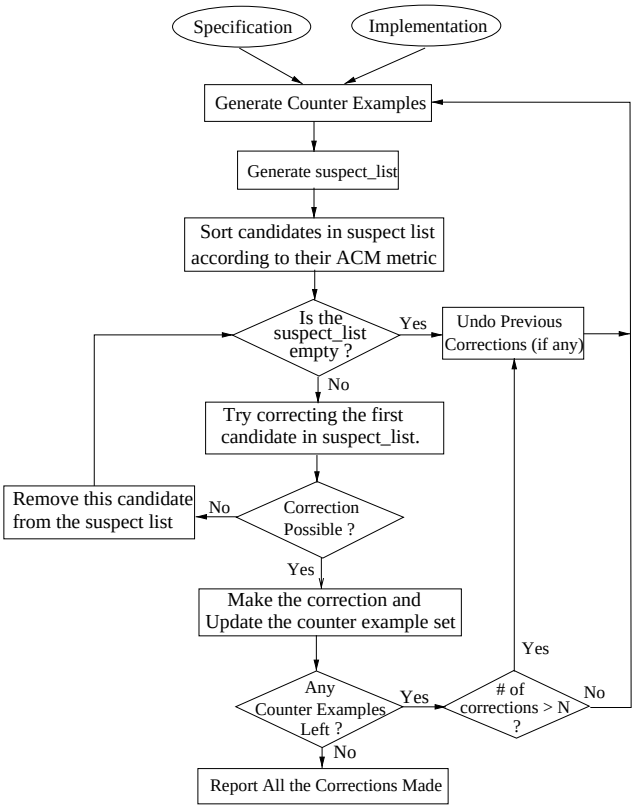


Figure 4: Simulation-based correction procedure.

correction and check if any of those corrections are consistent with the required values at that net. We follow a greedy approach while selecting the best possible correction. The correction that will minimize the number of counter examples is selected and the network is updated. The above procedure is repeated until the set of counter examples becomes empty. Since the algorithm operates on a limited number of input vectors, we cannot assume the network to be corrected even after all the counter examples are removed. The reason that some other counter examples may exist that were not considered by the correction procedure. Therefore, we make use of a formal verifier to check the validity of the rectification. If the specification and the implementation are not equivalent after the rectification process, we restart the diagnosis and the correction procedures, with a larger number of input vectors.

The greedy approach does not guarantee a solution. For example, in Figure 5, there are two errors at locations h (AND gate replaced by XNOR gate) and p (AND gate replaced by XOR gate) and there are five distinguishing vectors between the correct and the incorrect circuits. Because of the mutual interference between the two errors, we find lines y and z have higher *correctability* than either line p or line h. In fact, y and z are the only lines capable of cor-

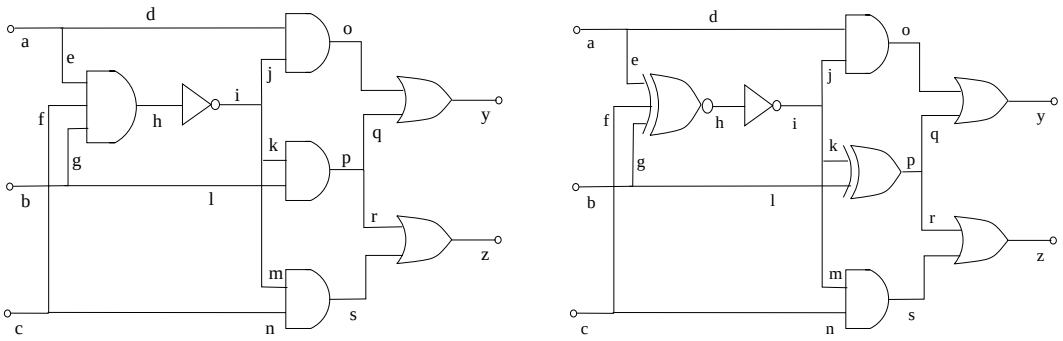


Figure 5: Example of wrong correction.

recting four of the five distinguishing vectors. Therefore, our tool will make a correction at *y* or *z* (change OR gate to XNOR gate, etc.). After this correction is made, it is impossible to get rid of the remaining counter example by any other correction. However, this problem can be solved by backtracking on previously made corrections. Whenever our tool is unable to make any progress, it backtracks and replaces the last correction made with the next possible correction, which didn't look as attractive before the last correction was made. In this way, we can ensure that our tool tries all possible kinds of corrections before giving up. But this may not be desirable as it may take a long time to rectify large circuits. Note that this problem does not arise if the errors are not very close together. However the experimental data is too limited to assume that errors are sparse.

4.4 Algorithm Complexity

Since our algorithm is based on simulation, its complexity is directly related to the p input vectors simulated and the n nets in the circuit. For each vector simulated, the time is dominated by the critical path tracing done to determine net observability. We use the back-trace algorithm [2], with separate simulations for each multiple fanout net. This takes time $\Theta(mn)$ for m multiple fanout lines or $\Theta(n^2)$ in the worst case. The correction time is small compared to the diagnosis time, so the time for one diagnosis and correction step is $\Theta(pn^2)$. Typically one diagnosis and correction step is required for each error, so for s steps, the time will be $\Theta(psn^2)$.

5 Experimental Results

To validate our algorithm, a prototype system has been implemented in *Python* [23] on an UltraSPARC 2 with 1 GB of memory. Experiments were performed using the ISCAS'85 benchmarks [6]. In each experiment, random errors of different types were inserted and the diagnosis and correction algorithm was applied. For these experiments, we use random input patterns rather than error-detecting patterns supplied by a verifier.

Table 1 shows the results of diagnosing and correcting circuits with a single random gate type error. The results are the average of five runs. Since the ISCAS'85 circuits have good testability, only a few random vectors must be simulated to prune the suspect list. We stop the diagnosis when there are still many suspects because our correction procedure is so fast relative to our diagnosis procedure that it takes less time to consider many suspects than to further prune the suspect list. The correction time is small since in most cases the first suspect is used for correction. The total time is less than the corresponding time for the symbolic error correction tool ET [11]. Note that our time does not include the time to reverified the corrected circuit.

Results for diagnosis and correction of multiple random errors are shown in Table 2. Each entry is the average of two runs. The entries with no corrections occurred when the tool was terminated after one hour without a solution. Most of the time the number of steps required for rectification is the same as the number of corrections. As in the single gate errors, almost all the time is spent in critical path tracing during diagnosis.

One concern is that the simulation-based approach will result in invalid corrections or corrections that are significantly different than one might expect. Table 3 compares the number of non-equivalent lines between the original and rectified circuits as a measure of rectification accuracy. In most cases the rectified circuit is identical to the original. However the existence of equivalent errors can prevent this. In C1908, about half of the gates are buffers or inverters which results in many equivalent error locations. In all cases the rectified circuit is functionally equivalent to the original. It appears that a reasonable distinguishing vector set makes the odds of an erroneous correction small.

Table 1: Single Gate Type Error Diagnosis and Correction

Circuit	*Gates*	*Random Vectors*	*Dist. Vectors*	*Diagnosis Time (s)*	*Suspect Nets*	*Rectify Time (s)*	*Total Time (s)*	*ET Total Time (s)*
C432	160	36	4	3.2	38	0.0	3.2	52
C499	202	38	4	3.2	89	0.2	3.4	9
C880	383	32	14	6.0	6	0.2	6.2	5
C1355	546	24	8	13.2	158	1.0	14.2	13
C1908	880	34	19	35.5	16	0.3	35.8	41
C2670	1193	38	18	54.3	47	2.8	57.1	99
C3540	1669	40	8	54.9	73	7.4	62.3	1844
C5315	1993	25	6	86.8	147	0.1	86.9	58
C6288	2406	10	7	89.6	306	0.3	89.9	132
C7552	3512	10	5	87.8	81	23.8	111.6	419

Table 2: Multiple Error Diagnosis and Correction

Circuit	*Two Errors*			*Three Errors*			*Four Errors*		
	Steps	*Vectors*	*Time (s)*	*Steps*	*Vectors*	*Time (s)*	*Steps*	*Vectors*	*Time (s)*
C432	2	150	16.1	3	200	35.8	5	210	57.6
C499	2	200	29.9	4	200	56.2	4	240	66.4
C880	2	100	31.6	3	150	71.4	4	180	94.7
C1355	2	50	56.9	3	200	289.5	4	250	497.3
C1908	2	100	217.4	3	260	682.8	4	250	945.5
C2670	2	100	288.1	3	160	648.1	3	150	633.1
C3540	2	100	439.5	2	195	703.9	3	200	1350.9
C5315	2	100	672.8	3	150	1463.8	–	–	$>$1hr
C6288	2	50	965.6	3	100	2688.2	–	–	$>$1hr
C7552	2	150	2515	3	200	$>$1hr	–	–	$>$1hr

Table 3: Rectification Accuracy

Circuit	*Errors*	*Steps*	*Inequivalent Nets*	
			Before Rect.	*After Rect.*
C432	4	5	56	0
C499	4	4	44	3
C880	4	4	158	2
C1355	3	3	191	0
C1908	3	3	142	19
C2670	3	3	442	0
C3540	2	2	574	0
C5315	3	3	100	0
C6288	2	2	690	0
C7552	2	2	278	0

6 Conclusions and Future Work

An incremental logic rectification algorithm based on simulation has been presented. We defined a design error scenario for combinational circuits, where modifications on some lines of the circuit are sufficient to rectify the corrupted gate-level design. An efficient algorithm that returns all actual modification locations with their respective corrections was presented. The algorithm is based on vector simulation and boolean function manipulation. The experimental results show the applicability of the proposed algorithm to both single and multiple design errors.

We can speed up our tool in several ways. Using a new linear-time critical path tracing algorithm [19] will reduce the algorithm complexity to be linear in circuit size. We could eliminate the linear bit-list scans during correction by using improved pattern matching algorithms, but this is already a small part of the runtime. Recoding from Python to *C* will speed it up by 10 to 100 times [29]. Overall this should provide at least a 100 times speedup. We can further speed up correction of multiple errors by including more complex error models, such as swapping two gate inputs.

These will reduce the number of correction steps and vectors needed. Overall we believe these will allow us to diagnose and correct several errors in the largest ISCAS circuits in a few minutes.

We must explore the behavior of our simulation-based DEDC approach further. This includes behavior with many clustered errors, pathological cases, and DEDC using vectors supplied by a verifier.

References

[1] M. S. Abadir, J. Ferguson and T. E. Kirkland, "Logic Design Verification via Test Generation," *IEEE Trans. Computers*, pp. 138-148, Jan., 1988.

[2] M. Abramovici, M. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Piscataway, NJ: IEEE Press, 1993.

[3] M. Abramovici, P. R. Menon and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," *IEEE Design & Test of Computers*, Vol. 1, No. 1, pp. 83-93, Feb. 1984.

[4] A. Apostolico and Z. Galil, *Pattern Matching Algorithms*, New York, NY: Oxford University Press, 1997.

[5] H. Al-Asaad and J. P. Hayes, "Design Verification via Simulation and Automatic Test Pattern Generation," *IEEE Int'l Conf. on Computer-Aided Design*, San Jose, CA, pp. 174-180, Nov. 1995.

[6] F. Brglez and H. Fujiwara, "A Neutral Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. Int'l Symp. Circuits and Systems*, pp. 695-698, Jun. 1985.

[7] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, Vol. 35, pp. 677-691, Aug. 1986.

[8] B. Chen, C. L. Lee and J. E. Chen, "Design Verification by Using Universal Test Sets," *Proc. IEEE Third Asian Test Symp.*, Nara, Japan, pp. 261-266, Nov. 1994.

[9] P. Y. Chung, Y. M. Wang and I. N. Hajj, "Logic Design Error Diagnosis and Correction," *IEEE Trans. VLSI Systems*, Vol. 2, No. 3, pp. 320.

[10] S.-Y. Huang, K.-T. Cheng, K.-C. Chen and D.-I. Cheng, "Error Tracer: A Fault Simulation Based Approach to Design Error Diagnosis," *Proc. IEEE Int'l Test Conf.*, Washington, DC, pp. 974-981, Nov. 1997.

[11] S.-Y. Huang, K.-T. Cheng and K.-C. Chen, "Incremental Logic Rectification," *Proc. IEEE VLSI Test Symp.*, Monterey, CA, pp. 455-460, Apr. 1997.

[12] S.-Y. Huang, K.-T. Cheng, K.-C. Chen and J.-Y. Joseph Lu, "Fault-Simulation Based Design Error Diagnosis for Sequential Circuits," *Proc. Design Automation Conf.*, San Francisco, CA, Jun. 1998.

[13] J. Jain, J. Bitner, D. S. Fussel and J. A. Abraham, "Probabilistic Design Verification," in *IEEE Int'l Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 468-471, Nov. 1991.

[14] A. Kuehlmann, D. I. Cheng, A. Srinivasan and D. P. LaPotin, "Error Diagnosis for Transistor-level Verification," *Proc. Design Automation Conf.*, Anaheim, CA, pp. 466-471, Sept. 1992.

[15] S. Y. Kuo, "Locating Design Errors via Test Generation and Don't-Care Propagation," *Proc. European Design Automation Conf.*, Hamburg, Germany, pp. 466-471, Sept. 1992.

[16] H. T. Liaw, J.-H. Tsiah and C.-S. Lin, "Efficient Automatic Diagnosis of Digital Circuits," *IEEE Int'l Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 464-467, Nov. 1990.

[17] C.-C. Lin, K.-C. Chen, S.-C. Chang and M. M. Sadowska, "Logic Synthesis for Engineering Change." *Proc. Design Automation Conf.*, San Francisco, CA, pp. 647-652, Jun. 1995.

[18] J. C. Madre, O. Coudert, J. P. Billon, "Automating the Diagnosis and Rectification of the Design Errors with PRIAM," *IEEE Int'l Conf. on Computer Aided Design*, Santa Clara, CA, pp. 30-33, Nov. 1989.

[19] Z. Navabi, M. Shadfar and A. Peymandoust, "Using VHDL Critical Path Tracing Models for Pseudo Random Test Generation", *Proc. VHDL Int'l Users' Forum*, Santa Clara, CA, pp. 41-50, Apr. 1995.

[20] I. Pomeranz and S. M. Reddy, "On Correction of Multiple Design Errors," *IEEE Trans. Computer-Aided Design*, pp. 255-264, Vol. 14, No. 2, Feb. 1995.

[21] I. Pomeranz and S. M. Reddy, "A Method for Diagnosing Implementation Errors in Synchronous Sequential Circuits and Its Implications on Synthesis," *Proc. European Design Automation Conf.*, Hamburg, Germany, pp. 252-258, Sept. 1993.

[22] I. Pomeranz and S. M. Reddy, "On Error Correction in Macro-based Circuits," *IEEE Int'l Conf. on Computer-Aided Design*, San Jose, CA, pp. 568-575, Nov. 1994.

[23] Python Web Site, http://www.python.org, January, 1994.

[24] "SIS: A System for Sequential Circuit Synthesis," Report M92/41, University of California, Berkeley, 1992.

[25] K. A. Tamura, "Locating Functional Errors in Logic Circuits," *Proc. Design Automation Conf.*, Las Vegas, NV, pp. 185-191, June, 1989.

[26] M. Tomita, H. H. Jiang, T. Tomamoto and Y. Hayashi, "An Algorithm for Locating Logic Design Errors," *IEEE Int'l Conf. on Computer Aided Design*, Santa Clara, CA, pp. 468-471, Nov. 1990.

[27] A. G. Veneris and I. N. Hajj, "A Fast Algorithm for Locating and Correcting Simple Design Errors in VLSI Digital Circuits," *Proc. Great Lake Symposium on VLSI Design*, Champaign, IL, pp. 45-50, Mar. 1997.

[28] A. G. Veneris and I. N. Hajj, "Error Diagnosis and Correction in VLSI Digital Circuits," *Proc. Midwest Symposium on Circuits and Systems*, pp. 1005-1008, 1997.

[29] K. Waclena, "Some Programming Language Benchmarks," http://www.lib.uchicago.edu/keith/crisis/bench.html, June 24, 1997.

[30] A. M. Wahba and D. Borrione, "A Method for Automatic Design Error Location and Correction in Combinational Logic Circuits," *Journal of Electronic Testing: Theory and Applications*, Vol. 8, No. 2, pp. 113-27, Apr. 1996.