Error Diagnosis for Transistor-Level Verification

Andreas Kuehlmann* David I. Cheng** Arvind Srinivasan* David P. LaPotin*

*IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y.

**Dept. of Electrical and Computer Engineering
University of California
Santa Barbara, C.A

Abstract

This paper describes a diagnosis technique for locating design errors in circuit implementations which do not match their functional specification. The method efficiently propagates mismatched patterns from erroneous outputs backward into the network and calculates circuit regions which most likely contain the error(s). In contrast to previous approaches, the described technique does not depend on a fixed set of error models. Therefore, it is more general and especially suitable for transistor-level circuits, which have a broader variety of possible design errors than gate-level implementations. Furthermore, the proposed method is also applicable for incomplete sets of mismatched patterns and hence can be used not only as a debugging aid for formal verification techniques but also for simulation-based approaches. Experiments with industrial CMOS circuits show that for most design errors the identified problem region is less than 3% of the overall circuit.

1 Introduction

Different techniques have been developed for verifying that an implemented design has the same behavior as a given "correct" specification. Classical design verification by simulation proves the correctness for only a limited set of input patterns. A complete coverage for all possible patterns can be accomplished by exhaustive comparison methods like BDD-based formal verification [1], test vector approaches [2] or probabilistic methods [3]. In the case of a miscompare, all methods provide a partial or complete list of counter examples in the form of mismatched input patterns. However, from a usage point of view the designer is primarily interested in locating, and subsequently correcting, the error.

Previous work in this area has focused on error diagnosis and correction as being one problem, where a single occurrence of a design error from a predefined set of possible models (e.g missing or superfluous connection, wrong gate type, etc.) is assumed. The idea of error correction is to apply these hypothetical error models to potential error locations and to repeat the verification procedure until the design is correct.

A general approach to check whether the modification of a single internal net function could correct a design can be formulated by means of a set of implicit Boolean equations [4, 5]. If a solution exists for these equations, then the reimplementation of the tested net is sufficient to make the erroneous design correct. Although different pruning techniques, such as input cone intersection [4] or elimination of dominated nets [5, 6] have been proposed, the solution of the Boolean equations is considerably expensive. In [7], a set of pairs each consisting of a mismatched and a matched input pattern, differing in one input value only, are used to identify those gates where the propagation of the pattern pair is interrupted. The identified gates are used as an initial guess for backtracking the error. The drawback of this approach is again the limitation to single errors. Moreover, the required pairs of input patterns do not necessarily exist. In [8], a specific method for identifying misordered inputs and outputs is presented. This approach does not deal with internal errors.

These previous approaches work on gate-level representations and are intended as a correction mechanism for errors which were induced by a synthesis tool or for updating a previously synthesized result after some manual change. This paper focuses on error diagnosis for transistor-level verification of CMOS designs which implies the following specific problems:

- High-performance CMOS circuit design is most often done manually, where automatic error correction would not be acceptable. A general debugging technique which is able to identify potential error regions in a general way is preferred.
- The variety of possible design errors introduced

at the transistor level is much broader than at the gate level. After extracting an equivalent Boolean network from the transistor-level representation, errors typically appear in multiple locations of the Boolean network. A diagnosis model which is restricted to a single error occurrence from a fixed set of templates is too simple.

- In hierarchical implementations, a single error introduced in a frequently used subnetwork is replicated many times over the whole design. The common error source needs be identified.

This paper describes a new diagnosis technique for the identification of possible error locations in incorrect transistor-level implementations of combinatorial circuits. The proposed *Error Coverage Algorithm* (EC-algorithm) works on the equivalent Boolean structure of a CMOS circuit, where mismatched patterns are propagated from incorrect outputs backward into the network. These patterns are associated with internal nets which most likely cause the errors. The back propagation can be done implicitly for all patterns simultaneously (e.g. BDD-based) or explicitly for one pattern at a time (e.g. simulation-based).

We show that the resulting number of collected error patterns at internal nets is an excellent base for precisely identifying single design errors. We also show that, in case of multiple errors, this number can be used as a good metric for identifying problematic circuit regions where the designer should look first.

This paper is structured as follows: Section 2 presents a new gate-level diagnosis model which is applied to a Boolean network extracted from a transistor-level circuit. Section 3 describes the EC-algorithm for the back propagation of error patterns through the network. Section 4 summarizes the transistor-level verification extraction procedure and describes the application of the diagnosis technique to the extracted model. Sections 5 and 6 present results and conclusions, respectively.

2 Gate-Level Model

Let $N(G, W)$ denote a gate-level implementation of some combinatorial network with a set of primitive gates $G = \{G_1, \ldots, G_g\}$ and a set of nets $W = \{W_1, \ldots, W_w\}$ interconnecting these gates. $\{PO_1, \ldots, PO_n\} \subseteq W$ are the n primary output nets and $\{PI_1, \ldots, PI_m\} \subseteq W$ are the m primary input nets of N. Without loss of generality, we assume that each net has a fanout of one. Signals driving several destinations are modeled by multi-output buffer gates. This enables us to distinguish between the destinations for the sake of error diagnosis. A set of Boolean variables $\underline{x} = (x_{PI_1}, \ldots, x_{PI_m})$ is assigned to the input nets $\{PI_1, \ldots, PI_m\}$. Based on the structure of N and the primitive functions of G, each net $W_i \in W$ computes some Boolean function $f_{W_i}(\underline{x})$.

A given functional specification of network N assigns to each output PO_i an a priori correct function $F_{PO_i}(\underline{x})$. Let us assume some verification technique is used to compare the specification with the implementation for a set of input patterns $X = \{X_1, \ldots, X_p\}$, $1 \leq p \leq 2^m$. The network N is said to be correct with respect to X if and only if: $f_{PO_i}(X_j) = F_{PO_i}(X_j)$, $1 \leq i \leq n$, $\forall X_j \in X$, otherwise N is incorrect.

Definition: Given a network N which implements a set of functions $f_{PO_1}(\underline{x}), \ldots, f_{PO_n}(\underline{x})$ and their functional specification $F_{PO_1}(\underline{x}), \ldots, F_{PO_n}(\underline{x})$, the set of *counter examples* (CEX) for the set of input patterns X is defined as: $CEX = \{(PO_i, X_j) \mid f_{PO_i}(X_j) \neq F_{PO_i}(X_j),\ X_j \in X,\ 1 \leq i \leq n\}$.

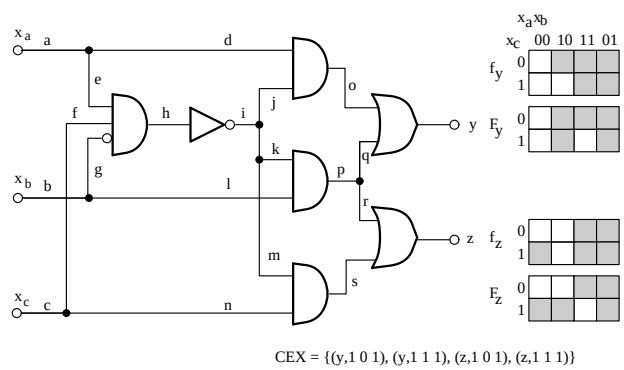

Figure 1: Gate-level example (inverter incorrectly added at net g).

Figure 1 gives an example of an erroneous network. Let us assume the design was wrongly implemented by adding an inverter at net g which results in incorrect functions f_y and f_z at outputs y and z, respectively. The corresponding set of counter examples contains four elements $(y, 101), (y, 111), (z, 101)$ and $(z, 111)$.

Clearly, it is always possible to correct all counter examples by modifying the function f_{W_i} of certain nets $W_i \in W$. In the general case, the reimplementation of the functions $f_{PO_1}, \ldots, f_{PO_n}$ of all output nets results in a completely new design which could correct any problem in N.

Definition: The set of nets $C = \{W_{c1}, \ldots, W_{cp}\} \subseteq W$ is called a *correction* of network N if there exists a set of corrected functions $\{\tilde{f}_{W_{c1}}(\underline{x}), \ldots, \tilde{f}_{W_{cp}}(\underline{x})\}$ such that the replacement of $\{f_{W_{c1}}, \ldots, f_{W_{cp}}\}$ by

$\{\tilde{f}_{W_{c1}}, \ldots, \tilde{f}_{W_{cp}}\}$ results in a correct network N'.

In general, there exist multiple corrections C for an erroneous network N. For example, either net g, h, or i could correct the circuit of figure 1. The various corrections usually differ in: (1) the area and delay performance of the resulting network N', (2) the effort to reimplement the new net functions and (3) the applicability of specific error templates which can automatically be corrected (e.g.[5]).

In the following, we introduce the concepts of: (1) sensitivity of counter examples and (2) error coverage of internal nets of N. Based on these definitions we state an important theorem which can be used to identify corrections for the network N.

Definition: A counter example $(PO_i, X_j) \in CEX$ is called *sensitive* to net W_k if there exists a set of nets $W' \subset W - W_k$ and an assignment of constant values to all nets contained in W' such that:

(1) The output value $f_{PO_i}(X_j)$ is not affected by the constant assignment.

(2) For the given assignment of constant values, the output value $f_{PO_i}(X_j)$ becomes correct if the value $f_{W_k}(X_j)$ is inverted.

We use $SEN(PO_i, X_j)$ to denote the set of nets to which (PO_i, X_i) is sensitive.

Figure 2 illustrates the concept of sensitivity and error coverage for two counter examples $(y, 101), (z, 111)$ of figure 1. The highlighted nets are those to which the counter examples are sensitive. For example, consider $(y, 101)$ (figure 2a) with respect to net k. If a constant value 1 is assigned to net l then output $f_y(101)$ remains unaffected (incorrect). The output can be corrected if the value of net k is inverted. Therefore, counter example $(y, 101)$ is sensitive to net k. In contrast, $(y, 101)$ is not sensitive to net d, because there is no assignment of constant values to other nets such that the correction of $(y, 101)$ depends directly on the inversion of net d.

Definition: Given a set of counter examples CEX, the *error coverage* of a net W_k is defined as $EC_{W_k} = \{(PO_i, X_j) \mid (PO_i, X_j) \in CEX \text{ and } W_k \in SEN(PO_i, X_j)\}$.

For the circuit in figure 1, all four counter examples are sensitive to net k. Therefore, $EC_k = \{(y, 101), (y, 111), (z, 101), (z, 111)\}$. In contrast $EC_j = \{(y, 101), (y, 111)\}$.

The input cone for output PO_i is defined as the set of nets from which a path to PO_i exists. Obviously, the input cone of PO_i provides an upper bound on $SEN(PO_i, X_j)$. However, this bound is typically loose because, input cones are independent of input patterns. In contrast, the EC-approach derives a specific sensitivity cone for each counter example.

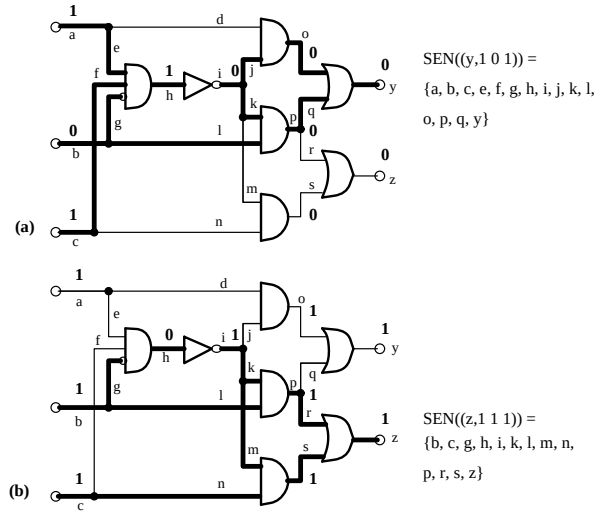

Figure 2: Set of nets to which two counter examples of figure 1 are sensitive.

Based on the error coverage of internal nets we can state the following theorem (proof is provided in [9]):

Theorem: If $C = \{W_{c1}, \ldots, W_{cp}\}$ is a correction of N then:

$$\bigcup_{W_k \in C} EC_{W_k} = CEX.$$

Corollary: If there exists a single net W_i such that its reimplementation can correct the erroneous design N, then $EC_{W_i} = CEX$.

As an example of the EC-concept, figure 3a highlights the nets whose EC covers all counter examples. Note that the stated theorem and corollary provide only a necessary condition for the error identification. For example, the reimplementation of nets b, l, k or p alone can not make N correct. In other words, the coverage of all counter examples by some net W_i does not imply that the reimplementation of W_i alone can correct the circuit. However, our results (section 5) show that the size of EC is a strong measure for identifying problematic circuit regions, and in case of a single error this region often contains one, or only few nets.

Since the input cones of erroneous outputs give an upper bound on $SEN(PO_i, X_j)$, their intersection provides an upper bound on the set of nets which cover all counter examples (see figure 3b). Again this bound is very loose because, input cones are independent of input patterns. In contrast, the EC-approach intersects the specific sensitivity cone for each counter example. This technique is powerful because it is typi-

cal for design errors to cause a large number of mismatched input patterns and multiple incorrect outputs.

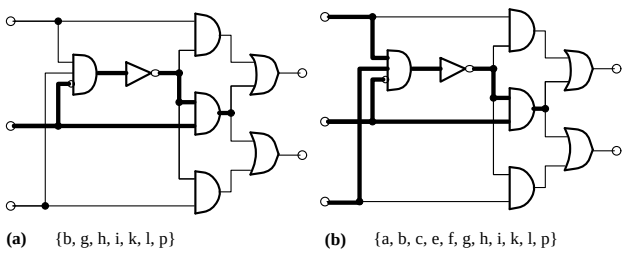

Figure 3: (a) Nets which cover all counter examples ($EC_{W_i} = CEX$). (b) Intersection of the input cones of both outputs.

Corollary: If $C = \{W_{c1}, \ldots, W_{cp}\}$ is a correction of N, then there exists at least one net $W_k \in C$ such that: $\mid EC_{W_k} \mid \geq \frac{|CEX|}{p}$.

In other words, if network N is assumed to have less than p errors, we can identify a set of nets which contains at least one of these errors. Although generally the size of the set of such nets increases for a growing number of errors, their identification provides a valuable debugging aid to handle multiple design errors.

3 EC-Algorithm

The idea of the EC-algorithm is to compute the error coverage of individual nets by propagating the counter examples from erroneous outputs backward into the network. The logic gates of the network will act as propagation filters where, depending on the input and resulting output values at these gates, specific counter examples are further propagated or blocked.

Let W_I be an input and W_O be an output of some gate G_k. For a fixed primary input pattern X_i, let v_{W_I} and v_{W_O} denote the evaluated values at nets W_I and W_O, respectively.

Definition: Output W_O is called *gate-sensitive* to input W_I, with respect to the pair of values (v_{W_I}, v_{W_O}), if there exists an assignment of constant values to all other inputs of G_k such that:

(1) The output value v_{W_O} is not affected by the constant assignment.

(2) For the given constant assignment the output value v_{W_O} changes if the value v_{W_I} is inverted.

We use $PROP(G_k, W_I, W_O)$ to denote the set of value pairs for which output W_O is sensitive to input W_I.

Figure 4 gives the gate-sensitivity for some primitive gate functions. For example, consider the gate-sensitivity of an AND gate for the value pair $(1, 0)$. In

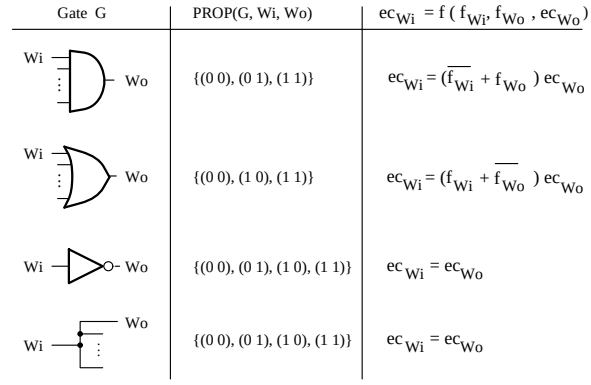

Figure 4: Gate-sensitivity for AND, OR, INVERTER, and multi-output buffers

this case it is assumed that some primary input pattern X_j causes a 1 at the input and (because some other input is 0) a 0 at the output. We say W_O is not sensitive to W_I, because inverting the value at W_I can never cause a change of the value at W_O for any constant value assignment to the other inputs of the AND gate. Therefore, $(1, 0) \notin PROP(AND, W_I, W_O)$. The idea is that if the output value 0 of an AND gate is incorrect, then all inputs which are 1 can not be held responsible for the error. The EC-algorithm uses $PROP(G_k, W_I, W_O)$ as a "gate-filter" to decide whether for the corresponding input pattern a counter example is to be back propagated through G_k. The following pseudo code details the EC-algorithm:

```
Algorithm EC (N, F)
  /* network: N(G,W), specification: F(x) */
  FOR all outputs PO_i of N DO
    EC_{PO_i} = {(PO_i, X_j) | f_{PO_i}(X_j) ≠ F_{PO_i}(X_j), X_j ∈ X};
  END FOR;
  FOR all outputs PO_i of N DO
    Push_Backward (PO_i);
  END FOR;
END;

Algorithm Push_Backward (W_i)      /* net: W_i */
  IF net W_i not marked as backward_done DO
    Pull_Backward (W_i);
    FOR all inputs W_j of gate G_k driving net W_i DO
      Push_Backward (W_j);
    END FOR;
    mark net W_i as backward_done;
  END IF;
END;
Algorithm Pull_Backward (W_i)          /* net: W_i */
  IF net W_i not marked as forward_done DO
    FOR all outputs W_j of gate G_k driven by net W_i DO
      Pull_Backward (W_j);
    END FOR;
    FOR all inputs W_j of gate G_k driving net W_i DO
      EC_{W_j} = EC_{W_j} ∪ {(PO_u, X_v) | (PO_u, X_v) ∈ EC_{W_i}
                 and(f_{W_i}(X_v), f_{W_j}(X_v)) ∈ PROP(G_k, W_j, W_i)}
    END FOR;
```

```
    mark net W_i as forward_done;
  END IF;
END;
```

The implementation of the EC-algorithm can be done implicitly for all input patterns or explicitly for one pattern at a time. The explicit approach complements a simulation based verification method by an additional step of backward-simulating mismatched input patterns. The implicit approach can be done by a BDD-based implementation of the set operations for simultaneously back propagating all mismatched input patterns. The corresponding Boolean expressions for the "gate-filter" $PROP(G_k, W_I, W_O)$ are given in the third column of figure 4. For implementation purposes, to distinguish between mismatched patterns of different counter examples, additional output specific BDD-variables must be inserted.

Theorem: If $F(\underline{x})$ is the functional specification of some incorrect design N, then the EC-algorithm computes the error coverage EC_{W_i} for the internal nets of N (see[9] for detailed proof).

4 Transistor-Level Error Diagnosis

Algorithms for formally verifying the correctness of a CMOS implementation versus its gate-level specification are usually based on a switch-level interpretation of the transistor circuit [10, 11, 12]. Such approaches check the static equivalence of the function without considering the timing or delay dependent behavior of the circuit. A common approach is to first extract a functionally equivalent Boolean network from the transistor-level design, and then prove the correctness of the Boolean network against the gate-level specification.

In the following, we informally summarize the extraction procedure of the Boolean network from transistor-level designs and demonstrate the application of the described diagnosis technique. To simplify the discussion, and to focus on the main idea of transistor-level error diagnosis, we exclude both ratio-logic (i.e. MOS circuits where the function depends on the strength ratio of certain transistors) and combinatorial loops from consideration. In contrast to the gate-level model presented in section 2, we relax the definition of a net to include connections with multiple fanouts.

The extraction of the equivalent Boolean network from a CMOS design is based on *functional nets* in the transistor circuit. These nets include all primary inputs, primary outputs and the nets which control gates of MOS transistors. The extraction procedure assigns two Boolean functions f^1 and f^0 to each functional net. f^1 and f^0 define the cases for which the net is logically 1 and 0, respectively. Consider the erroneous CMOS circuit of figure 5 for which figure 6 gives the corresponding Boolean network. Nets a, b, c, s, and y establish the set of functional nets because they are either circuit terminals or drive some transistor gate. The corresponding nets in the Boolean representation are marked as $a^0, a^1, b^0, b^1, c^0, c^1, y^0$, and y^1. In contrast, d and e (figure 5) are not functional nets and therefore they have no counter part in the Boolean network.

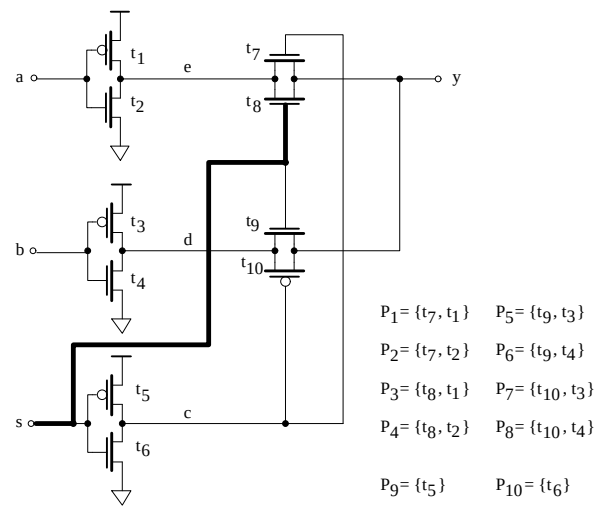

Figure 5: Erroneous multiplexer implementation (incorrect transistor type for t_8).

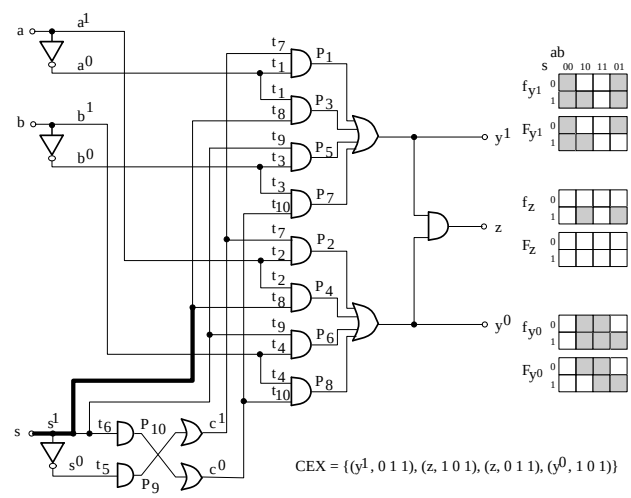

Figure 6: Equivalent Boolean network for circuit of figure 5 (highlighted nets cover all counter examples).

For each primary input, both polarities of the cor-

responding input variable are assigned to f^1 and f^0 (e.g. $a^1 = a$; $a^0 = \overline{a}$). For primary outputs and internal nets, the ON-sets of f^1 and f^0 describe the set of input patterns for which the net is driven by VDD and $GROUND$, respectively. In other words, $f^1(X_i) = 1$ means that there is a path of interconnected transistors from that net to VDD such that all transistors are conducting if pattern X_i is applied at the circuit inputs. In a similar manner $f^0(X_i) = 1$ denotes some path to $GROUND$. The equivalent of a path in the transistor circuit is an AND gate in the Boolean network. The AND inputs are driven by the gate functions f^1 (for NMOS) or f^0 (for PMOS) of the corresponding path transistors. The AND outputs of all paths driving a net to VDD or $GROUND$ are collected by OR-gates to generate f^1 or f^0, respectively.

The circuit of figure 5 contains ten paths, eight driving net y and two driving net c. For example, path P_1 drives y by VDD if transistors t_1 and t_7 are conducting. As shown in figure 6, the AND of path P_1 is fed by a^0 for $t1$ and c^1 for t_7, which are PMOS and NMOS transistors, respectively. Consider that the given multiplexer example (figure 5) is incorrectly implemented where t_8 is an NMOS transistor, instead of a PMOS. This error causes the two paths $P_3 = \{t_8, t_1\}$ and $P_4 = \{t_8, t_2\}$ to be activated by the wrong polarity of input s. In the Boolean representation, this error is reflected by connecting the input of both paths AND's, P_3 and P_4, to s^1 instead of s^0. Note that for this particular example, a single error in the transistor circuit causes two errors in the corresponding Boolean network.

The verification step proves the correct implementation of the circuit outputs against a given specification. f^1 and f^0 of each output must be compared against both polarities F^1 and F^0 of their specified function, respectively. For the multiplexer output y, figure 6 shows the implemented functions f_{y^1} and f_{y^0} and their specification F_{y^1} and F_{y^0}. The mismatched input patterns are (011) and (101), respectively.

In addition to the comparison of the circuit outputs, a set of consistency checks can be formulated for each functional net. For example, if the designer is using a circuit technique which excludes the application of ratio-logic, the intersection of f^1 and f^0 detects collisions where the net is driven simultaneously by VDD and $GROUND$. Similarly, the union of both functions could identify conditions for which a net is floating, i.e. not driven by VDD or $GROUND$. These checks can directly be mapped into some Boolean structure providing additional test points. For the example in figure 6, output z computes the intersection of f_{y^1} and f_{y^0}, finding collisions for input patterns (011) and (101). Typical design errors often cause violations of these consistency checks at many internal nets. This results in a large amount of additional counter examples which significantly improves the discrimination capability of the presented diagnosis technique.

The presented EC-algorithm was applied to the Boolean network for the four counter examples $CEX = \{(y^1, 011), (z, 101), (z, 011), (y^0, 101)\}$. As a result only two nets cover all counter examples. These nets and their counter parts in the original transistor circuit are highlighted in figure 6 and figure 5, respectively. Note that the extraction and diagnosis algorithms work directly on the transistor circuit by interpreting it according to the equivalent Boolean network. Therefore, the inputs t_8 for the paths AND gates P_3 and P_4 are not distinguished and treated as one net. This enables the diagnosis algorithm to exactly identify the erroneous gate-connection of t_8 and net s^1.

5 Results

The EC-algorithm is part of a BDD-based verification system, *VERITY*, which is being used for CMOS processor designs within IBM. To evaluate the diagnosis capabilities of the presented approach, random errors were introduced into a set of industrial CMOS circuits of varying complexity. First, the verification step generates the complete set of counter examples. Next, the EC-algorithm is applied to back propagate these counter examples into the network. Based on the size of the resulting error coverage at internal nets, and depending on the number of assumed error locations, circuit regions are identified which contain at least one error.

The first experiment evaluates the discrimination capabilities for single errors. For each circuit example, a single randomly chosen net was manipulated by either connecting it to a different source or setting it to a constant value. Table 1 summarizes the results. We define a suspicious net as one which is possibly causing an error. The column titled "cone intersection" reports the size of the input cone intersection of all erroneous outputs (including test points). For single errors the intersection provides the set of nets which must contain the error. In some cases (e.g. e2 or e3) this method gives a tight bound, producing only few nets. However, it is more typical for the cone intersection to result in a considerably large fraction of the whole circuit (e.g. e9 or e12).

The right portion of the table summarizes the performance for the EC-algorithm. As shown, the number of resulting nets in the suspicious circuit region is significantly smaller than the input cone intersection. The average size of the error region comprises 1.8% of

	design		cone intersection	EC-algorithm			
	# transistors / # nets	# inputs / # outputs	# suspicious nets	# suspicious nets	rel. area %	add. memory kByte	add. CPU sec.
e1	900 / 454	194 / 64	47	5	1.1	131	3.62
e2	74 / 81	10 / 33	4	3	3.7	0	0.1
e3	346 / 259	68 / 68	3	3	1.2	0	0.2
e4	718 / 505	80 / 65	30	15	3.0	0	0.3
e5	134 / 77	8 / 2	20	7	9.1	0	0.2
e6	11254 / 3510	383 / 487	120	4	0.1	0	4.8
e7	915 / 486	163 / 3	9	3	0.6	0	1.9
e8	10948 / 3354	383 / 332	90	1	0.03	0	5.23
e9	5296 / 2031	312 / 26	685	3	0.15	33145	690.0
e10	2636 / 1430	175 / 61	56	4	0.3	0	1.1
e11	1016 / 676	64 / 18	54	10	1.5	2097	50.0
e12	12851 / 4628	258 / 66	638	19	0.4	43147	1567.0

Table 1: Diagnosis results for single errors.

the whole circuit; for most examples it is below 3%. Note that larger circuit examples produce a relatively smaller number of suspicious nets.

The reported numbers for memory usage and CPU time (IBM RS6000, Model 340) consider only the resources for the diagnosis algorithm. A memory usage of 0 indicates that no additional BDD-nodes were allocated by the EC-algorithm, and hence a maximum reusage of existing BDD-nodes from the verification part. Other examples (e.g. e9 and e12) demand a significantly large amount of new BDD-nodes as well as CPU time for the BDD-operations. This increase is caused by the ordering of the BDD-variables. In the implementation, the ordering is optimized for the extraction of the output functions only and does not consider the operations for back propagating error patterns. We expect that in future implementations techniques for dynamically changing the variable ordering [13] will reduce this problem.

The second experiment illustrates the distribution of the error coverage for varying numbers of errors. The EC-algorithm was applied to example e6 containing one, two, four, and eight randomly introduced errors. Figure 7 shows the histogram for the error coverage of all 3510 nets. For example, in case of a single error, 4 nets have an error coverage of 100%, 9 nets cover more than 99%, 11 nets more than 91%, etc. As expected, as the errors increase, the distribution of the error coverage becomes flatter resulting in an increasing number of nets in the suspicious circuit region. According the second corollary, for an assumed number of errors, we can identify some part of the circuit which must contain at least one of them. For each EC distribution, the "$*$" denotes the identified circuit region which must contain at least one of the errors. For the one, two, four, and eight errors, this region comprises 4, 13, 39, and 133 out of 3510 nets, respectively; still a small fraction of the overall network. This clearly illustrates that even for multiple errors, the EC-algorithm provides a valuable debugging aid for the designer.

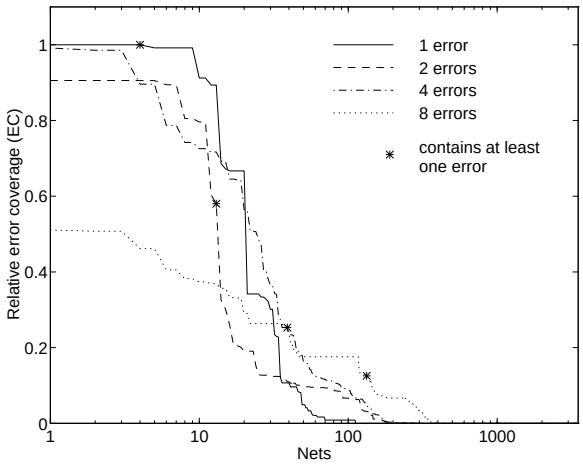

Figure 7: Histogram for the error coverage of example e6 (table 1).

6 Conclusion

This paper describes a technique for locating design errors in incorrectly implemented circuits. As compared to previous approaches, the proposed method is independent of predefined error models and is not restricted to errors which are correctable in terms of given correction templates. The algorithm is based on a single efficient network traversal for back propagating mismatched input patterns from erroneous outputs.

The resulting error coverage at internal nets of the circuit provides an upper bound for the identification of nets whose reimplementation can correct the design.

Practical experiments demonstrate that for single occurrences of design errors this bound is tight, producing only one or few nets as possible error candidates. Results show that, even for multiple errors, the size of the identified region which contains at least one error is sufficiently small, providing a valuable debugging aid in practical design environments.

This paper has mainly focused on error diagnosis for transistor-level designs. Nevertheless, the proposed technique is equally applicable to complement other gate-level error correction techniques. The presented method provides a strong selection mechanism for reducing the set of possible error candidates, which is more general and faster than prior preselection schemes. Since the resulting number of error candidates is small, more expensive and complex correction algorithms can be applied.

Acknowledgements

The authors would like to thank Geert Janssen from the Technical University Eindhoven for providing his BDD-package and Victor Rodriguez from IBM for his technical support. Additional thanks are extended to IBM designers Christopher Durham and David Appenzeller for contributing circuit examples.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, August 1986.

[2] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic design verification via test generation," *IEEE Transactions on Computer-Aided Design*, vol. 7, pp. 138–148, January 1988.

[3] J. Jain, J. Bitner, D. S. Fussel, and J. A. Abraham, "Probabilistic design verification," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 468–471, November 1991.

[4] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the diagnosis and the rectification of design errors with PRIAM," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 30–33, November 1989.

[5] P.-Y. Chung, Y.-M. Wang, and I. N. Hajj, "Diagnosis and correction of logic errors in digital circuits," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, (Dallas, TX), pp. 503–508, June 1993.

[6] H.-T. Liaw, J.-H. Tsaih, and C.-S. Lin, "Efficient automatic diagnosis of digital circuits," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 464–467, November 1990.

[7] M. Tomita and H.-H. Jiang, "An algorithm for locating logic design errors," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 468–471, November 1990.

[8] I. Pomeranz and S. M. Reddy, "On diagnosis and correction of design errors," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 500–507, November 1993.

[9] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin, "Error diagnosis for transistor-level verification," Tech. Rep. Computer Science, RC 19219 (#83668), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, October 1993.

[10] G. Ditlow, W. Donath, and A. Ruehli, "Logic equations for MOSFET circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, (Newport Beach, CA), pp. 752–755, May 1983.

[11] R. E. Bryant, "Boolean analysis of MOS circuits," *IEEE Transactions on Computer-Aided Design*, vol. 6, pp. 634–649, July 1987.

[12] D. T. Blaauw, D. G. Saab, P. Banerjee, and J. A. Abraham, "Functional abstraction of logic gates for switch-level simulation," in *Proceedings of The European Conference on Design Automation*, (Amsterdam, The Netherlands), pp. 329–333, IEEE, February 1991.

[13] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *International Workshop on Logic Synthesis*, (Tahoe City, CA), pp. 3a–1–3a–12, May 1993.