

# VERILAT: Verification Using Logic Augmentation and Transformations

Debjyoti Paul, Mitrajit Chatterjee, and Dhiraj K. Pradhan, *Fellow, IEEE*

**Abstract**—This paper presents a new framework for formal logic verification. What is depicted here is fundamentally different from previous approaches. In earlier approaches, the circuit is either not changed during the verification process, as in ordered binary decision diagram (OBDD) or implication-based methods, or the circuit is progressively reduced during verification. Whereas, in our approach, we actually enlarge the circuits by adding gates during the verification process. Specifically, introduced here is a new technique that transforms the reference circuit as well as the circuit to be verified, so that the similarity between the two is progressively enhanced. This requires addition of gates to the reference circuit and/or the circuit to be verified. In the process, we reduce the dissimilarity between the two circuits, which makes it easier to verify the circuits.

In this paper, we first introduce a method to identify parts of the two circuits which are dissimilar. We use the number of implications that exist between the nodes of one circuit and the nodes of the other circuit as a metric of similarity. As demonstrated, this can be a very useful metric. We formulate transformations that can reduce the dissimilarity. These are performed on those parts of the circuits which are found to be dissimilar. These admissible transformations are functionality-preserving and based on certain Boolean difference formulations. The dissimilarity reduction transformations introduce new logical relationships between the two circuits that did not previously exist. These logical relationships are extracted as new implications, which are then used to reduce the complexity of the verification problem. These two steps are repeated in succession until the verification process is complete. A complete procedure is presented which demonstrates the power of our logic verification technique. The concept presented in this paper can be useful in accelerating verification frameworks which rely on structural methods.

**Index Terms**—Combinational logic circuits, design verification, equivalence checking, logic circuit testing, logic function.

## I. INTRODUCTION

**D**ISTINCT methods for formal combinational logic-level verification include binary decision diagram (BDD)-based functional methods [9]–[13] and structural methods [2]–[8]. These methods are formulated in the equivalence checking framework where a reference design is to be checked for equivalence against its implementation. The func-

tional methods are known to fail for certain types of circuits, whereas structural methods [also referred to as automatic test pattern generation (ATPG)-based methods] have been shown recently to provide more stable performance over a broad spectrum of circuit types [2]–[8]. Structural methods utilize *similarity* between the circuits and are not memory-intensive. Some of these methods have only linear memory requirements [2]–[6].

Often, synthesis tools significantly transform the unoptimized circuit, resulting in an optimized circuit with minimal similarity with the original. As a result, these structural methods which rely on similarity, are unable to verify some circuits. This appears to be the major shortcoming of these methods which are sometimes broadly referred to as ATPG methods. The suggested remedies have been to take breakpoints in the synthesis process and “spit out” intermediate circuits bearing more similarity to each other, or to synthesize locally, leaving module inputs and outputs unchanged. Often, the full capability of a synthesis tool is not used, as the synthesized circuits may be difficult to verify. Hence, better verification tools will not only ensure correctness but also pave the way for more efficient designs.

The proposed framework introduces a powerful and a more general solution and is based on the following.

- 1) A *metric of similarity* based on *indirect* implications.
- 2) A technique to *identify* those regions where the two circuits under verification and the reference circuit are significantly dissimilar from each other.
- 3) Once these regions which constitute the bottlenecks for the verification process are identified, *similarity-enhancing transformations* are performed. These augment both the reference circuit and the circuits under verification in such a way as to *induce* similarity. This makes it easier to verify.
- 4) Essentially the proposed approach *augments* the circuit during the verification process in such a way that as the circuit gets larger, it gets easier to verify.
- 5) The method is further strengthened by using learning procedures at specific regions where potential equivalent nodes exist.

Here, we first introduce a metric of similarity through indirect implications between nodes of one circuit and the nodes of the other circuit whose equivalence is being checked. We then propose to identify regions which are significantly *dissimilar* using this metric. This information is then used for performing certain similarity-enhancing transformations by addition of gates and connections in a guided way. This is followed by renewed search for logical implications which are induced

Manuscript received August 25, 1997; revised October 6, 1999. This work was supported in part by the National Science Foundation (NSF) under Grant MIP9406946, and is subject of U.S. patent [2], [3]. An initial version of the paper appears in the proceedings of the IEEE/ACM International Conference on CAD, 1996. This paper was recommended by Associate Editor M. Fujita.

D. Paul is with the Verification Technology Group, Synopsys Inc., Mountain View, CA 94043 USA (e-mail: debjyoti@synopsys.com).

M. Chatterjee is with the Design Automation Group, Integrated Device Technology Inc., Santa Clara, CA 95054 USA.

D. K. Pradhan is with the Computer Systems Lab, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 0278-0070(00)07471-6.

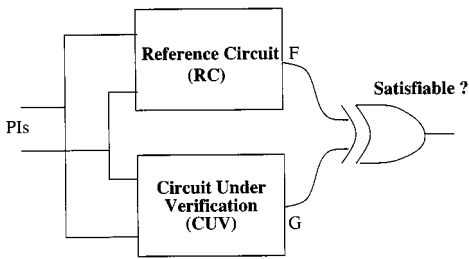


Fig. 1. Miter formulation.

by these transformations. These are used in our verification process to make it possible to verify and accelerate the process. It may be noted that the proposed approach, like other structural methods, has only linear memory requirements. However, our approach does not require strong similarity since it introduces similarity through transformations where none exists. The experimental results not only demonstrate that our approach is able to verify circuits where other approaches fail, but also completes faster. First, we attempt verification using implication. If this fails, we then identify regions in the reference circuit and circuit under verification where no similarity exists between the two. We then perform transformations, connecting a similar region in one circuit to a dissimilar region in another. This is performed in conjunction with certain heuristics so as to reduce dissimilarity. The transformations are designed to be admissible (function-preserving). Thus, although the miter undergoes structural changes, the original functionality of the miter remains unaltered throughout the verification process. After each transformation, we recompute similarities which are defined through logical implications. Using in succession newly found implications between the transformed circuits, we attempt to verify again. This two-step process is repeated until the verification is complete. During the process of verification, the circuits actually get enlarged—a counter-intuitive solution to the verification problem. Previous methods either preserved the circuit [6], [13] or attempted to reduce it physically [7] or logically [5].

This paper is organized as follows. Section II enumerates the prior work in this area. Section III explains the proposed method. Section IV discusses the experimental results. The last section presents our conclusions.

## II. PRIOR WORK

Given two circuits, one a known good design termed the reference circuit (RC) and the other, referred to as a circuit under verification (CUV). A miter of the circuits is formed, as shown in Fig. 1. The problem of verification then reduces to the problem of satisfiability.

Recently, new approaches to verification have been proposed that use ATPG techniques combined with random simulation [7], and implication combined with ordered binary decision diagram (OBDD) [5]. These are broadly categorized as ATPG techniques. However, there are some important differences between these different techniques. The technique proposed in [7]

reduces the miter by finding nodes in the RC which are equivalent to nodes in the CUV. Then it replaces the nodes in the RC with nodes in the CUV progressively, working from inputs to outputs. Ultimately, if the two circuits are equivalent, then each output node in the RC can be replaced with the corresponding output node in the CUV. The search for intermediate equivalent nodes is guided by simulation and ATPG is used to prove their equivalence. The techniques proposed in [5] use indirect implications derived by a recursive learning technique [2], [4], [6]. The implications, combined with the D-implication proposed in [11], can provide a technique more powerful than that proposed by [7]. Various logical relationships between the two circuits, which can be extracted as implications, provide a super set of those extracted by [7]. Essentially, equivalent nodes [4] are a form of implication. For example if  $a$  and  $b$  are equivalent then  $a = 0 \rightarrow b = 0$  and  $a = 1 \rightarrow b = 1$ . If  $a$  and  $b$  are complements then  $a = 0(1) \rightarrow b = 1(0)$ . Equivalent or complement relationships can be translated into two implications. Structural methods like [5], [7] rely on identifying equivalent nodes between the two circuits. They may also identify nodes which are equivalent *under don't care conditions*. Replacing a node in one circuit with the corresponding node in the other circuit reduces the size of the miter physically [7] or logically [5]. The smaller circuit eases the final task of verification, using ATPG concepts. However, these methods do not exploit *asymmetric relationships* [6] that may exist between signal nodes. For example, a value on node  $y$  may imply a signal value in  $f$ , but not vice versa.

Two methods using BDDs have been presented in [9], [10]. In [9], equivalences are identified and the circuit is reduced by connecting these together and eliminating the cones of logic in the RC/CUV. The reduced circuit is more easily verified. The equivalence proving is done using BDDs. It is a very fast and efficient implementation. In [10], equivalent nodes are identified efficiently using hashing. Efficient BDD techniques are used to manipulate the functional representation of the circuit and verify them. However these methods do not exploit asymmetric relationships [6] as explained above. Also, the methods do not address on verifying circuits that have very less number of equivalent points among them.

Our approach differs from the previous approach conceptually in that, not only do we exploit whatever logical relationships may exist between RC and CUV, but we also induce new logical relationships through introduction of new connections and extra gates between the two circuits. As seen later, as the number of gates in the circuits increases during verification, it becomes easier to verify. As the circuit size grows during verification, it gets easier to verify, somewhat counter to the conventional wisdom of reducing the circuit [1] or partitioning the circuit [5].

Fig. 2 shows a classification of combinational logic verification methods based on how the circuit structure changes during the verification process. The first category keeps the circuit unchanged [5], [12], [13], [10]. The second category tries to simplify the verification process by reducing the circuit size physically [7], [9] or logically [5]. The final category is the proposed approach which enlarges the circuit. Although increasing the circuit size seems counter-intuitive, it actually eases the verification process if guided properly, as will be shown later.

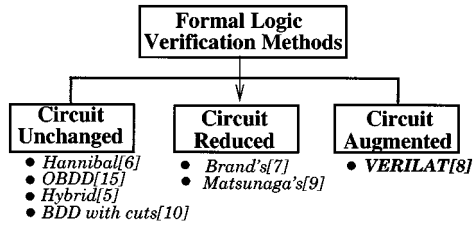


Fig. 2. Classification of logic verification methodologies based on structural modifications during verification.

The proposed approach removes a shortcoming of the structural methods by incorporating additional logical relations between similar and dissimilar parts of the circuit, in a cost-effective manner. The underlying framework relies on identification of *dissimilar* regions which are the bottlenecks of the verification process. The dissimilar regions are then transformed into similar regions by adding gates and connections. Specifically, the proposed verification procedure uses basically two *verification modes*. The first mode consists of verification using well-known techniques of implications, simulation and equivalences. When this fails to verify within a certain resource bound, the circuits, the procedure changes to the second *verification mode*. This consists of performing similarity-enhancing transformations to induce similarity in an ordered manner. Because structurally transforming a circuit is a step which cannot be reversed, this process is guided using certain heuristics [3] which are also crucial. Experimental results have shown that this approach is powerful to verify dissimilar circuits efficiently.

### III. PROPOSED VERIFICATION PROCEDURE

Our verification procedure aims to identify structurally dissimilar regions of the circuit. It uses certain transformations to enhance similarity in these regions.

#### A. Identifying Dissimilar Regions

It has been observed that structural methods need a large amount of computational time for dissimilar circuits. To perform more efficient verification, it is necessary to identify dissimilar regions so that additional effort is spent in selected regions of the circuit which constitute the bottleneck in verification.

We formulate the metric of similarity using logical implications. This formulation of similarity is shown to be quite useful in identifying precisely those regions which have the least functional similarity. We then target our search for potential transformations of these regions so as to induce similarity.

There are two types of implications: direct and indirect. Direct implications are derived from the truth tables of the states and are of limited use. Indirect implications, on the other hand, provide a powerful tool for identifying logical relationships. Recursive learning procedure identifies all indirect implications, given sufficient levels of recursion [4]. These indirect implications form the basis of our secondary metric and illustrated below.

*Example 3.1:* In Fig. 3, consider the primary output,  $t = 1$ . We can invoke recursive learning to determine any indirect implications of this value assignment. The value  $t = 1$  makes the

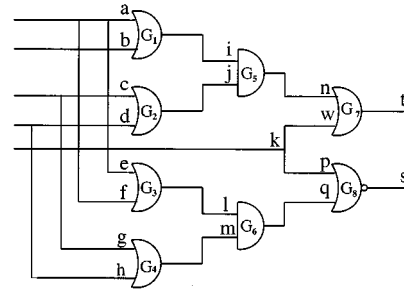


Fig. 3. Example circuit to explain behavior of recursive learning.

gate  $G_7$  unjustified. We enter level-1 recursion with two possible justifications:  $J_1 = \{n = 1, w = X\}$  and  $J_2 = \{n = X, w = 1\}$ . For  $n = 1$ , we obtain  $i = j = 1$  through direct implication. With  $i = 1$  and  $j = 1$ , we enter the second level of recursion, and obtain two sets of justification. For gate  $G_1$ ,  $J_3 = \{a = 1, b = X\}$  or  $J_4 = \{a = X, b = 1\}$  are the two possible justifications. Through direct implication, we deduce  $l = 1$ . Similarly with level-2 recursion, we can deduce that  $G_2 = 1 \rightarrow G_4 = 1$ . Thus, by direct implication, we obtain that  $n = 1 \rightarrow q = 1$ . Similarly, with  $w = 1$ , we have  $p = 1$ . In either case, we have  $s = 0$ , which is the consequence of the intersection implications, resulting from the two possible justifications,  $J_1$  and  $J_2$ , at gate  $G_7$ . It can be seen that the intermediate of implications, such as  $G_5 = 1$  implying  $G_6 = 1$ , are not learned because the only value assignment at hand is  $t = 1$ .  $\square$

It may be noted that given a large enough recursion level, one is guaranteed to find all implications. Since time complexity is exponential in terms of the number of levels of recursion, it is not practical to allow an arbitrarily large recursion level. However, it may be noted that the *maximum recursion level* is bounded by the levels of logic and importantly *the memory requirements* for recursive learning are linear in terms of size of the circuit [4]. Furthermore, for certain types of circuits or given a fixed level of recursion, the complexity is only polynomial. Also, recursive learning is *self-guided* in that the search is automatically guided by the topology of the circuit to precisely those nodes that are indirectly implied. BDD-based learning methods [12] require simulation for guiding the search.

Let  $F$  be the set of all nodes in the RC and  $G$  the set of all nodes in the CUV, as shown in Fig. 1. We define *similarity index* of a node  $g \in G$  ( $f \in F$ ) is equal to number of nodes  $f \in F$  ( $g \in G$ ), such that  $g \rightarrow f$  or  $f \rightarrow g$ . We will denote the similarity index as  $\delta$ . Consider for example the circuit shown in Fig. 4, in particular, the output of node 3. The following indirect implications exist which can be identified using level 1 recursive learning [4]

$$\begin{aligned} (3 \text{ in } F) = 0 &\rightarrow (3 \text{ in } G) = 0 \\ (3 \text{ in } G) = 0 &\rightarrow (3 \text{ in } F) = 0. \end{aligned}$$

Thus, it can be seen that the node representing output of gate 3 in both circuits has a similarity index of  $\delta = 2$ .

Given a set of implications of the nodes of the circuits  $F$  and  $G$  in a miter, we formulated the following framework for the *similar* and *dissimilar* regions with respect to the implications. We define *dissimilar* region as a group of connected nodes

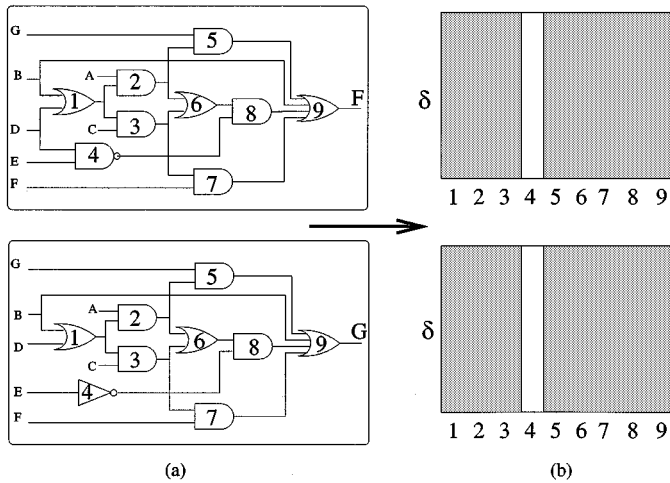


Fig. 4. Similarity profiles of example circuit pair.

TABLE I  
NUMBER OF IMPLICATIONS PER NODE  
USING RECURSION LEVEL 1 (FIG. 3)

Circuit Names	OBDD	Hybrid		VERILAT		
	Time	Rec Lev	Time	Rec Lev	Time	# SETs
c432	0:61	1	0:02	1	0:02	0
c499	1:29	1	0:02	1	0:05	0
c1355	2:24	1	0:07	1	0:20	0
c1908	0:30	1	0:12	1	0:22	0
c2670	unable	1	1:08	1	1:01	17
c3540	unable	2	6:13	1	4:41	20
c5315	0:21	1	0:21	1	3:10	20
c6288	unable	1	2:31	1	0:40	0
c7552	unable	1	93:33	1	6:52	20

where all the nodes have zero similarity index,  $\delta = 0$ . Conversely, a group of connected nodes with  $\delta > 0$  forms a similar region.

Table I depicts similarity index of all nodes in  $F$  and  $G$ . It is clear that the node 4 constitutes a dissimilar region. The set of similarity indices for all nodes in the miter is called the similarity profile.

Dissimilar regions are identified using a given level of recursion in the recursive learning procedure. Therefore, it is always possible that a region identified to be dissimilar using a particular level of recursion may be found to be similar at a higher level of recursion. For the circuits in Fig. 3, with recursion level of one, the value of  $t = 1$  does not imply any value on  $s$ . However, as we have seen above, given the maximum recursion level = 2,  $t = 1 \rightarrow s = 0$ . Therefore, given a recursion level of two,  $G_7$  and  $G_8$  form a similar region.

B. Similarity Enhancing Transformations

The recursive learning analysis phase identifies logical relationships between nodes in the miter. These logical relationships represented as implications are used to identify dissimilar regions. Our first verification tool attempts to verify using these implications. If it fails, then transformations are targeted on dissimilar regions of the circuits, and designed to induce similarity in these. These transformations are called similarity enhancing transformations (SET).

TABLE II  
NUMBER OF IMPLICATIONS PER NODE TO NODES IN THE OTHER CIRCUIT  
AFTER TRANSFORMATION

Number of implications in F		Number of implications in G	
Node	Similarity Index $\delta$	Node	Similarity Index $\delta$
1	2	1	2
2	2	2	2
3	2	3	2
4	0	4	0
5	2	5	2
6	2	6	2
7	2	7	2
8	2	8	2
9	2	9	2
-	-	10	2

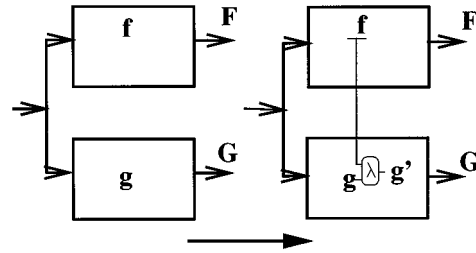


Fig. 5. Similarity enhancing transformation in a miter.

Theorem 3.1: Consider two combinational circuits  $F$  and  $G$  with their inputs tied, as shown in Fig. 5. Let  $f$  be a subfunction in circuit  $F$  and  $g$  be a subfunction in circuit  $G$ . The subfunction  $g$  in circuit  $G$  can be replaced by:  $g' = \lambda(f, g)$ , where

$$\lambda(f, g) = \begin{cases} f + g & \text{if } \bar{g} \frac{dG(f=1)}{dg} = 0 \\ \bar{f} + g & \text{if } \bar{g} \frac{dG(f=0)}{dg} = 0 \\ f \cdot g & \text{if } g \frac{dG(f=0)}{dg} = 0 \\ \bar{f} \cdot g & \text{if } g \frac{dG(f=1)}{dg} = 0. \end{cases}$$

Proof: Following, we present a proof for the first case. Proofs for other cases are similar. It may be noted that the transformation  $g' = \lambda(f, g) = f + g$  forces  $g' = 1$  whenever  $f = 1$ . This can be captured in

$$G(f = 1) \oplus G(f = 1, g = 1) = 0.$$

Performing Shannon expansion of the first term we have

$$g \cdot G(f = 1, g = 1) \oplus \bar{g} \cdot G(f = 1, g = 0) \oplus G(f = 1, g = 1) = 0.$$

$(g \oplus 1)G(f = 1, g = 1) \oplus \bar{g} \cdot G(f = 1, g = 0) = 0$  yields  $\bar{g} \cdot G(f = 1, g = 1) \oplus \bar{g} \cdot G(f = 1, g = 0) = 0$ , which is equivalent to  $\bar{g}dG(f=1)/dg = 0$  Q.E.D.

The equation can be interpreted as, if  $f = 1$ , a  $0 \rightarrow 1$  change on  $g$  cannot propagate to output  $G$ . It also can be viewed that when  $f = 1$  the fault  $g$  stuck-at-1 is undetectable. Both conditions are equivalent. The former has been stated as a









INITIAL STRUCTURE	CONDITION	TRANSFORMATION
1. 	$\bar{g} \cdot \frac{dG(f=0)}{dg} = 0$	
2. 	$\bar{g} \cdot \frac{dG(f=1)}{dg} = 0$	
3. 	$g \cdot \frac{dG(f=0)}{dg} = 0$	
4. 	$g \cdot \frac{dG(f=1)}{dg} = 0$	

Fig. 6. Similarity enhancing transformations.

D-implication [11]. In essence, Boolean difference captures this conditional undetectability.

There are four such basic transformations based on “simple” conditions. Other transformations with NAND, NOR, or EXOR gates require more complex Boolean difference formulation. These basic  $\lambda(f, g)$ s are shown with their corresponding Boolean difference conditions in Fig. 6. Note that the condition for any arbitrary  $\lambda(f, g)$  can be derived as a combination of above Boolean difference conditions. This formulation of admissible transformations can be nicely implemented using ATPG techniques, as seen below.

Since computing Boolean difference and checking its satisfiability can be complex, we suggest following ATPG techniques. Consider the case  $\lambda(f, g) = f + g$ . Let the value assignment,  $f = 0$ , be a necessary assignment to test  $g$  s-a-1; this implies transformation  $g' = f + g$  is a permissible function of  $G$ . Checking for this is accomplished by first assuming a fault s-a-1 at a node  $g$ , and then identifying those *necessary assignments* that are required to detect this fault. A node  $f \in F$  for which  $f = 0$  is a necessary assignment and, thus, a candidate for the transformation  $g' = f + g$ . Whether this transformation is actually performed or not depends on certain heuristics.

1) *SET Implementation*: The following example illustrates the generality of the SET conditions given in Theorem 3.1. These capture not only controllability and observability implications but also more general conditions for admissibility. Controllability and observability implications are special cases of this formulation of admissibility.

A SET can exist where neither a controllability implication [4] nor an observability implication [11] exists. In this example, the fault is excitable and observable separately, but it cannot be excited and observed simultaneously. Importantly, the following example illustrates that SETs are more powerful than controllability and observability implications treated independently.

*Example 3.2*: Fig. 7 shows the original subcircuit [21] and the circuit to be verified in its nonredundant version. These circuits can be transformed as shown in Fig. 9. There exists a Boolean difference condition which allows the SET shown in Fig. 8. Fig. 7 shows how the equivalent undetectability conditions, as explained above, are satisfied to allow this transformation. Fig. 7(a) shows the setting of values  $f = 1$  and  $g = 0/1$ . The fault justifications and unique sensitization values are shown in Fig. 7(b). The direct implications of the values in the

circuit demonstrate that there is a conflict between excitation and observation conditions, as seen in Fig. 7(c). The conflict reflects that under the condition of  $f = 1$ , the fault  $g = 0/1$  (s-a-1) is undetectable  $\square$ .

2) *Multiple SET Implementation*: As seen above, a single SET can be performed in isolation one at a time. For many circuits, to complete verification, one may need a large number of SETs. However, multiple SETs cannot be performed simultaneously, unless they form a compatible set. This follows from the well-known result that even if two stuck-at faults are undetectable individually, they can be detectable as a multiple fault [1]. Note that the order of choosing SETs is important. Certain heuristics can be used to accept or discard a particular SETs depending on certain cost-benefit function.

Given a set of single SETs, one can form a subset of compatible SETs using fault-detection theory. These compatible SETs can be performed in one step. However performing one SET at a time checking for the admissibility each time poses no particular problem.

3) *Effect of Transformations on Similarity Profile*: A SET can only be useful when it induces the needed amount of similarity between the circuits. A series of SETs adds numerous connections between the two circuits. Therefore heuristics [3] are formulated to decide whether to accept or discard a particular SET and are briefly explained below.

A node, say  $g \in G$ , is selected which is in the dissimilar region. A suitable  $f$  has to be selected from a similar or dissimilar region of  $F$ . If the search fails to find a suitable  $f$ , a new  $g \in G$  is selected using certain heuristics [3]. These heuristics are based on the similarity profile concept which has already been explained. The similarity profile detects regions which are dissimilar and hence a potential area for ATPG to have problems. In these dissimilar regions, absence of implications means that the ATPG search is unguided and boils down to a brute force enumeration of the search space. If this can be avoided, the whole verification process can be speeded up. This can be done by performing a transformation which structurally connects this dissimilar region to a similar or dissimilar region. One or many such transformations are performed. The line selected for transformation always lies in the dissimilar region. The line to which it is getting connected is either in a similar or a dissimilar region. As a result of performing such SET(s), the dissimilar region becomes similar to some region in the RC.

We first try to find SETs proceeding from input to output and then from output to input. The input to output process is the most helpful one and propagates similarity through the circuit. However, the output to input flow is also useful, since it concentrates on the outputs and any possible transformation near the outputs is very helpful for the ATPG process.

The two examples below illustrate how SETs can induce similarity and eliminate dissimilarity.

*Example 3.3*: Consider the SET connecting  $f$  and  $g$  in Fig. 8. It shows that several new additional implications because of the transformation. In fact this SET alone induces the final verifying implications between nodes in  $F$  and  $G$ . This is illustrated further in Fig. 9 which plots the similarity and dissimilarity regions. Node 4 is outside the similarity region, but now it does not impede the verification.  $\square$

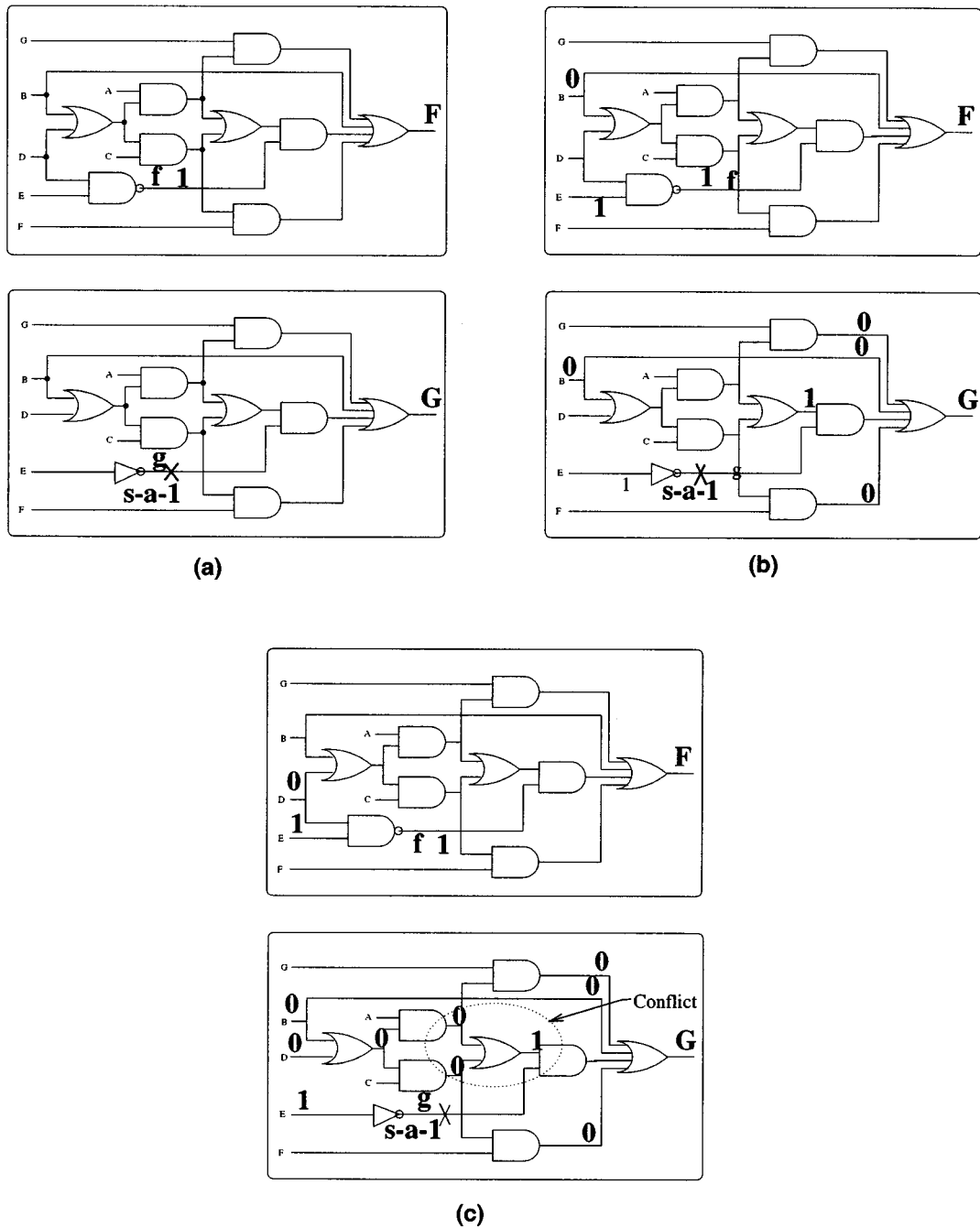


Fig. 7. Justification of the fault in the example circuits.

The following example illustrates the case for multiple SETs.

*Example 3.4:* Consider c432 and o1\_c432. Fig. 10 shows the similarity profiles using recursion level 1 implications. Because o1\_c432 was highly optimized by logic optimization with testability (LOT) [19] and had a region of strong dissimilarity, it needed 20 SETs. At this point, it became possible to verify the circuits with implications derived using only level 1 implications. The similarity profiles of the circuits, after transformations, are shown in Fig. 11. Note that the region before the output of o1\_c432, which had a dissimilar region, is now *eliminated completely* because of transformations. □

We have not studied the effect of SETs on improving the similarity profile theoretically. Our empirical experience shows that

they are very helpful. However, it is true that a SET might alter all existing controllability implications in the transitive fanout of the gate being transformed, i.e., node *g* in Fig. 8. However, we expect new controllability implications to come into existence and this is why SETs are useful.

Also note that introduction of SETs actually enlarges the circuit. Conventionally the complexity of the verification was thought to be in some way proportional to the circuit size. Also note that SETs are redundant logic in that they do not alter the functionality of the circuit. This is also thought to effect the ATPG process adversely. However, the whole miter is a redundant circuit (if the circuits are functionally identical) and any added redundancy will not be harmful if we ensure that it

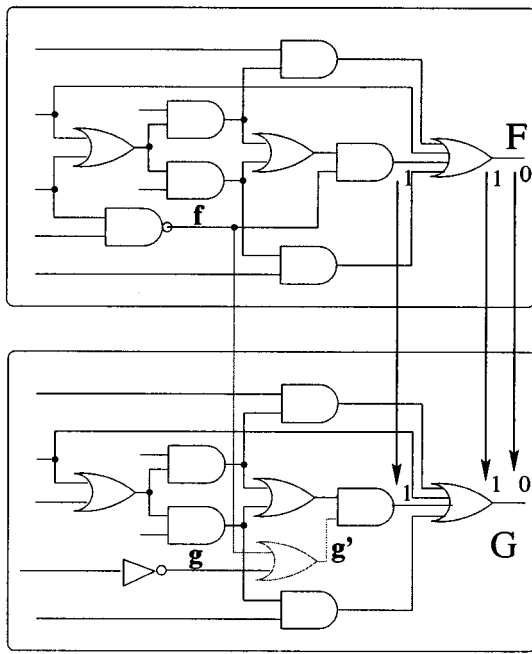


Fig. 8. New implications found after introducing a SET.

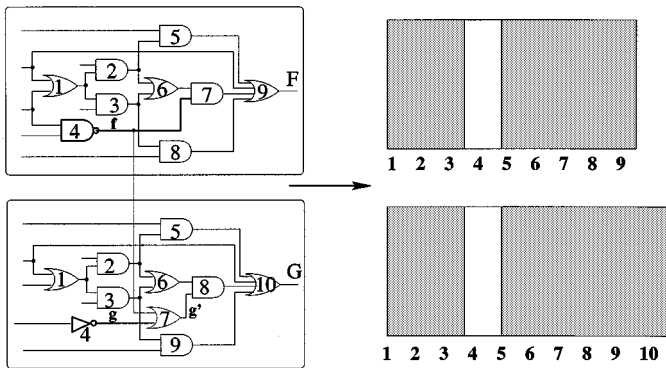


Fig. 9. Effect on similarity profile after a transformation.

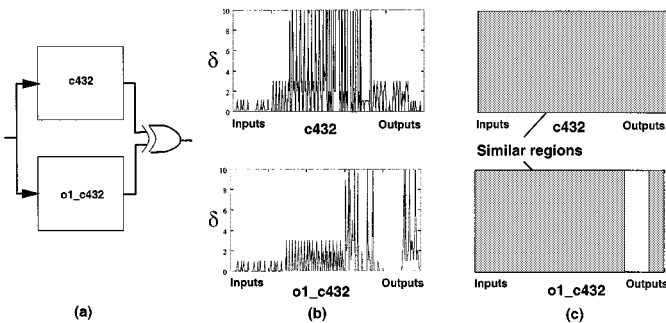


Fig. 10. Similarity profiles for c432 and o1\_c432.

helps the ATPG process. Using the heuristics in Section III-B3, we ensure that each SET is useful to the ATPG process.

### C. Equivalent Node Identification

Nodes in the two circuits might have equivalent functionality. Though these are rarer than implications, they occur frequently as some lines are not changed functionally. This might happen if only modules were changed keeping inputs and outputs in-

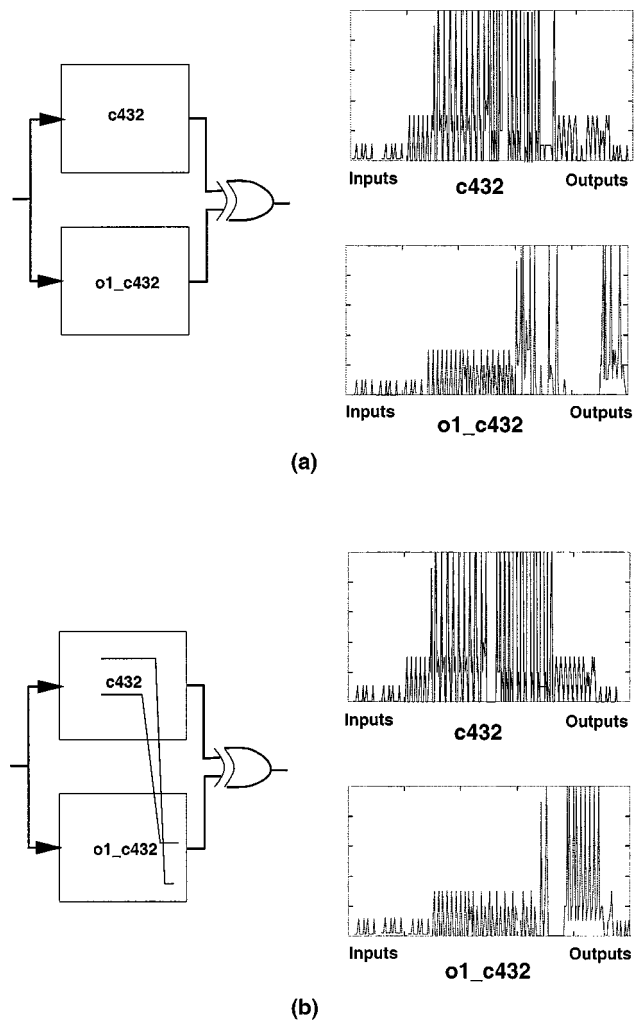


Fig. 11. Effect on similarity profile after SETs.

tact. Another fact is that probable equivalent nodes can be easily identified using simulation. Then they can be checked for equivalence. In our approach, if verification using implications fails, equivalences are tried. Often nodes not detected as equivalent using implications can be detected to be equivalent by this approach as the effort is more directed. Moreover, it is a divide and conquer approach where the task of proving equivalence of the outputs is split into detecting some internal equivalent points and then the final verification step is attempted.

The configuration used to identify equivalences is shown in Fig. 12. Candidates to be tried for equivalence are determined by random simulation. Also, only lines in the dissimilar regions and those between the two circuits are used for equivalence identification. The reason probable equivalent nodes are not interesting is that implications already exist in those regions. The added effort in those regions is probably a waste. Here, all probable equivalent nodes need not be tested as all of them may not be very useful for the verification process. Moreover, identifying only selected equivalent nodes based on the similarity index is not necessarily more difficult as it is done in a topologically sorted manner. If the lines are identified as equivalent, the relation is stored as two-way controllability implication, for future use.

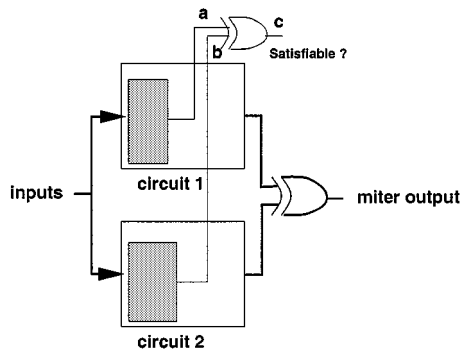


Fig. 12. Configuration to identify equivalent lines  $a$  and  $b$  in the miter.

We use parallel pattern logic simulation to identify probable equivalent nodes. Initially we assume all nodes to be equivalent to each other. Each parallel pattern vector simulated splits the current set of equivalent node groups into smaller groups. Hence it is a monotonic function. The number of random patterns used is based on a differential function. The simulation phase is stopped when the last  $n$  patterns did not alter the current set of equivalent node groups. The number  $n$  can be a small value like ten. This strategy is optimal in the sense that it simulates till it is no longer useful.

After the set of probable equivalent node groups is detected in the dissimilar region, the real equivalences are detected. This is done in a topologically sorted manner. The reason is previously detected equivalences in the input cone of the current probable equivalence being tested, are recycled and help the process. So this process starts from the edge of the dissimilar regions close to the inputs and slowly moves to the end close to the outputs. Starting from the edge closest to the input makes proving equivalences easier and enables the algorithm to identify the region to be actually similar. The controllability implications are reused.

The equivalence proving scheme is simple and is based on configuration shown in Fig. 12. We put a s-a-0 fault on the output of the EXOR and try to justify it using an ATPG process. Note that this ATPG process makes use of the implications already learned. The ATPG process is also resource bound and we give up after a fixed time interval. We then move on to the next probable equivalent pair.

#### D. The Algorithm

Our algorithm basically consists of three steps. First, the verification is attempted using earlier-known techniques of implications and equivalences [5], [6]. The next step involves identifying implications and equivalent nodes in regions. If this fails to verify the circuits, SETs are added in a guided manner. With enhanced similarity, more implications and equivalences exist which ease the verification process. The algorithm, as shown in Fig. 13, uses similarity profile, SETs and equivalence identification for verification. Three different verification modes are used during the verification process.

- *Verification mode = 1*: This verification mode uses implications and equivalences.
- *Verification mode = 2*: This mode uses implications, as well as equivalences identified in regions based on the *similarity profile*.

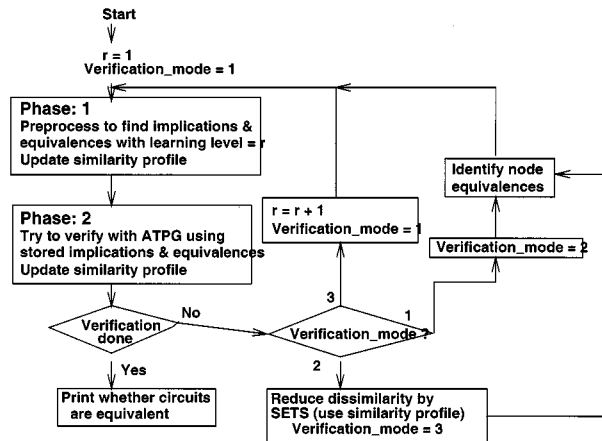


Fig. 13. Flowchart of proposed verifier.

- *Verification mode = 3*: This mode uses the implications, equivalences, transformations based on using SETs in the dissimilar regions, and equivalent node identification.

The procedure begins with the *Verification mode = 1*, and then switches to higher modes if the circuits are not verified. *Verification mode = 3* is the strongest verification phase used at a particular recursion level, as it uses SETs. After the three *verification modes* are tried, the recursion level is incremented. The implications learned in Phase 1 update the *similarity profile*, this updated similarity profile is then used to guide the SET procedure. The verification is attempted all over again with the highest level of recursion.

The basic motivation in using SETs, is to transform dissimilar regions to become more similar regions. The conditions which allow SETs are more general than finding implications and observability conditions. Our basic approach is to try to connect a node in the *dissimilar region* to a node in the *similar or dissimilar region* in the *other circuit* (as in Fig. 9). However, after a connection is made, existing implications involving the nodes in the transitive fanout of the new gate can get invalidated. Therefore, validation of the stored implications after performing SET(s) is necessary.

We try to introduce a fixed number (20) of SETs in each pass (The “reducing dissimilarity by SETs” box in Fig. 13). The number 20 is arbitrary and a more optimal number may be found. Note that we can add more than one SET sequentially, checking for admissibility each time with an ATPG engine. The ATPG engine takes into account the structural changes from previous SETs. We have observed that changing the order of SETs can affect the verification speed, but there is not fixed pattern that can be used. Heuristics based on the number of SETs in a dissimilar region have been found to be effective. New implications induced by these SETs are discovered using the same recursion level. These are stored and verification is attempted again.

It is possible to do more than one pass through mode 3 and add a fixed number of SETs before increasing the recursion level (mode 1). This strategy can be useful since SETs are more powerful and less expensive than learning all implications.

It should be noted since dissimilar regions are identified using a given recursion level, that it is always possible that a region identified to be dissimilar using a particular level of recursion



TABLE III  
COMPARISON OF PROPOSED METHOD ON NON-REDUNCANT CIRCUITS  
(TIMES ARE IN [mm : ss])

Circuit Names	OBDD	Hybrid		VERILAT		
	Time	Rec Lev	Time	Rec Lev	Time	# SETs
c432	0:61	1	0:02	1	0:02	0
c499	1:29	1	0:02	1	0:05	0
c1355	2:24	1	0:07	1	0:20	0
c1908	0:30	1	0:12	1	0:22	0
c2670	unable	1	1:08	1	1:01	17
c3540	unable	2	6:13	1	4:41	20
c5315	0:21	1	0:21	1	3:10	20
c6288	unable	1	2:31	1	0:40	0
c7552	unable	1	93:33	1	6:52	20

TABLE IV  
COMPARISON OF PROPOSED METHOD ON SYNTHESIZED CIRCUITS  
(TIMES ARE IN [mm : ss])

Circuit Names	OBDD	Hybrid		VERILAT		
	Time	Rec Lev	Time	Rec Lev	Time	# SETs
c432	0:60	1	0:53	1	0:05	0
c499	0:36	1	0:03	1	0:04	0
c880	1:05	1	0:20	1	0:14	11
c1355	1:02	1	5:24	1	0:07	0
c1908	0:27	1	7:03	1	1:36	0
c2670	fail	2	fail	1	0:58	9
c3540	fail	2	fail	1	2:19	20
c5315	0:13	2	10:24	1	1:37	38
c6288	fail	2	fail	1	1:09	0
c7552	fail	2	fail	1	5:07	20

may be determined similar at a higher recursion level. Therefore, each time we increase the recursion level, we re-examine the dissimilar regions.

The algorithm is our way to use the concepts of implications, equivalences and SETs together. This can be altered and may yield better results. Our aim was to demonstrate the power of the concept of SETs and we got very encouraging results with this algorithm, as we show in the next section.

#### IV. RESULTS

First set of experiments was carried out on the ISCAS85 benchmark circuits [21]. These circuits were verified against the nonredundant versions, as well as against LOT optimized versions [19]. This tool LOT makes significant changes to the circuits for testability, destroying much of the similarity. The performance of our tool has been compared with a OBDD-based verifier [22], and a hybrid method [5]. Results are reported in Tables III and IV obtained on a Sun Sparc 5.

In Table III, it can be observed that all the circuits were verified using only level 1 of recursion. As expected, c7552 took the longest time, of about only 7 min. This clearly demonstrates the accelerating power of SETs proposed here.

Next, the LOT optimized version [19] of ISCAS85 circuits, given in, was verified; the results are shown in Table IV. As before, it can be seen that our verification was significantly faster, requiring only level 1 recursion. As stated before we are able to limit the level of recursion to just one because of the SETs and,

TABLE V  
COMPARISON OF PROPOSED METHOD ON SIS OPTIMIZED CIRCUITS USING  
SCRIPT.RUGGED (TIMES ARE IN [mm : ss])

Circuit Names	VERILAT		
	Rec Lev	Time	# SETs
c432	1	0:03	40
c499	1	0:10	0
c880	1	0:08	0
c1355	1	0:04	0
c1908	1	0:07	40
c5315	1	0:43	40
c6288	1	0:34	16
c7552	1	3:04	20

thus, producing a significant speed up. These experimental results signify that if the verification process can be completed in a given recursion level without resorting to higher levels, it translates to an exponential gain in time. VERILAT clearly outperforms the other techniques, which demonstrates its capability. As the circuits were very dissimilar both of the other techniques OBDD [22] and Hybrid [5] failed. Regarding the number of SETs applied, for c5315 we went through two passes of mode 3 (we tried to introduce  $2 \times 20 = 40$  SETs). For all other circuits, we went through one pass of mode 3 and tried to introduce 20 SETs. For some circuits we found less than 20 desired number of SETs at that recursive learning level. However, it should be noted that the OBDD based verifier used for comparison is not the best available; the latest BDD packages can verify more circuits. Note that the results using our tool is without any significant performance tuning or any trial and error processes. Hence there is room for further improvement of run times to some degree. We were primarily interested in experimenting with the algorithmic speedups.

Experiments on these benchmark circuits also provide the following insight. Circuits optimized using a large number of observability don't care conditions may result in needing of large number of SETs to overcome the dissimilarity.

VERILAT being oriented toward difficult-to-verify circuits, should be used only after the other simpler approaches [2]–[8] fail. Importantly, it has a stable performance over a wide variety of circuits. Our procedure and underlying heuristics [3] seem to provide maximum benefit for difficult-to-verify circuits with little similarity with the reference circuit. No more than approximately 5 min was needed to verify any of the optimized circuits.

#### A. Additional Results

We have some additional results on ISCAS 85 benchmark circuits optimized by SIS-1.2 [23] using the *script.rugged*. The results are shown in Table V. These are reported for 8 of the 10 ISCAS 85 benchmark circuits as we had some difficulty in translating c2670 and c3540 to our format. The results demonstrates that the proposed tool can easily verify circuits altered/optimized by other synthesis tools. We obtained similar results for circuits optimized using *algebraic.script* and *boolean.script*, which was very encouraging. This proves that the proposed method is robust over a wide variety of dissimilar circuits.

## V. CONCLUSION

We have presented a new approach for formal logic verification which is fundamentally different from previous approaches. In the new technique, we *increase* the size of the circuit. The similarity between the two circuits is progressively enhanced by transformations. Similarity is measured in terms of number of implications that exist between the nodes of the two circuits. We perform transformations on those parts of the circuits that are diagnosed as highly dissimilar so as to reduce the dissimilarity. These admissible transformations are functionality-preserving and based on certain Boolean difference formulations. The dissimilarity reduction introduces new logical relationships between the two circuits that did not previously exist. These two steps are repeated in succession until the verification process is complete. In our approach we have tried to deal with dissimilarity between circuits and SETs. To comment upon the relationship of optimality of a circuit with dissimilarity and SETs, we have not explored the relationship between number of possible SETs and similarity between the circuits. Note that the number of SETs reported in the results is the minimum required to verify the circuits using our method. More dissimilar circuits would require more SETs, although it requires more effort to find them. There is a proportional relationship between the effort to find SETs and the dissimilarity between circuits. Optimality of the circuits has no direct impact since we construct a miter of the two circuits.

We think that algorithmically our approach is the strongest in its' ability to detect similarity between circuits as compared to other existing approaches [5]–[7], [9], [10], [12]. Additionally, the ability of the proposed method to enhance similarity by performing transformations is novel. The concept presented in this paper can be useful in accelerating verification frameworks which rely on structural methods. The experimental results show the proposed method to be very efficient for dissimilar circuits.

## REFERENCES

- [1] M. Abramovichi, M. A. Breuer, and A. D. Friedman, *Digital Testing and Testable Design*. Piscataway, NJ: IEEE Press.
- [2] D. K. Pradhan and W. Kunz, "Method for circuit verification and multilevel circuit optimization based on structural implications," U.S. Patent 5 526 514, June 11, 1996.
- [3] D. K. Pradhan, D. Paul, and M. Chatterjee, "Method for Comparative Circuit Verification," U.S. Patent 08 671 421, June 1996.
- [4] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to CAD problems-test, verification and optimization," *Trans. Computer-Aided Design*, vol. 13, Sept. 1994.
- [5] W. Kunz, D. K. Pradhan, and S. Reddy, "A novel framework for logic verification in a synthesis environment," *IEEE Trans. Computer-Aided Design*, vol. 15, Jan. 1996.
- [6] W. Kunz, "HANNIBAL: An efficient tool for logic verification based on recursive learning," in *Proc. Int. Conf. of Computer-Aided Design*, 1993, p. 538.
- [7] D. Brand, "Verification of large synthesized designs," in *Proc. Int. Conf. Computer-Aided Design*, 1993, p. 534.
- [8] D. Paul and D. K. Pradhan, "Logic verification using recursive learning, transformations and ATPG," Dept. Comput. Sci., Texas A&M Univ., College Station, Tech. Rep., 1995.
- [9] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *Proc. Design Automation Conf.*, 1996, pp. 629–634.
- [10] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. Design Automation Conf.*, 1997, pp. 263–267.

- [11] W. Kunz and P. R. Menon, "Multilevel logic optimization by implication analysis," in *Proc. Int. Conf. Computer-Aided Design*, 1994, p. 6.
- [12] R. Mukherjee, J. Jain, and D. K. Pradhan, "Functional learning: A new approach to learning in digital circuits," in *Proc. IEEE VLSI Test Symp.*, Apr. 1994, pp. 122–127.
- [13] R. E. Bryant, "Graph based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 667–691, Aug. 1986.
- [14] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *Proc. 33rd Design Automation Conf.*, 1996, p. 629.
- [15] S. Malik *et al.*, "Logic verification using binary decision diagrams in a logic synthesis environment," *Proc. ICCAD*, pp. 6–9, 1988.
- [16] R. E. Bryant and Y. A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *32nd Design Automation Conf.*, 1995, p. 535.
- [17] J. Markoff, "Flaw undermines accuracy of pentium chip," *New York Times*, p. 1, Nov. 23, 1994.
- [18] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw Hill, 1994.
- [19] M. Chatterjee, D. Pradhan, and W. Kunz, "LOT: Logic optimization with testability—New transformations using recursive learning," presented at the ICCAD, 1995.
- [20] L. A. Entrena and K. T. Cheng, "Combinational and sequential logic optimization by redundancy addition and removal," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 909–916, July 1995.
- [21] F. Brglez and H. Fujiwara, "A neural netlist of ten combinational benchmark circuits and a target translator in FORTRAN," presented at the Int. Symp. Circuits and Systems, Austin, TX, June 1985.
- [22] K. Brace, "Efficient implementation of a BDD package," *DAC*, 1989.
- [23] (1994) Design Technology Warehouse. Univ. California, Berkeley. [Online] Available: HTTP: <http://www-cad.eecs.berkeley.edu/Software/software.html> (access date Sept. 17, 1996)



**Debjyoti Paul** received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1994 and the M.S. degree in computer science from Texas A&M University, College Station, in 1996.

He has worked at Mentor Graphics and Motorola among others. He is currently with the Verification Technology Group, Synopsys, Mountain View, CA. His research interests include verification, VLSI design and testing.

**Mitrajit Chatterjee** received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1992 and the M.S. and Ph.D. degrees in computer science, from Texas A&M University, College Station, in 1994 and 1997, respectively.

He has worked at AT&T Bell Laboratories, Princeton, Max Planck Institute, Germany, Synopsys Inc., Beaverton, OR, and Reliable Computer Technology, College Station, TX, among others. He is currently with the IPD System Level Products Design group, Integrated Device Technology, Inc., Santa Clara, CA.

Dr. Chatterjee received the Dr. B. C. Roy Memorial Gold Medal at I.I.T. Kharagpur in 1992. His research interests include VLSI design, timing analysis and testing.



**Dhiraj K. Pradhan** (S'70–M'72–SM'80–F'88) is currently a faculty member in the Department of Electrical and Computer Engineering, Oregon State University, Corvallis. He has held other recent faculty positions at Stanford University, Stanford, CA, and Texas A&M University, College Station. Beforehand, he served as Professor and Coordinator of Computer Engineering at the University of Massachusetts, Amherst. Additionally, he has worked at University of California, Berkeley, Stanford University, Oakland University, Rochester,

MI, and the University of Regina, Regina, SK, Canada. He has contributed to very large scale integrated computer-aided design and test, fault-tolerant computing, computer architecture, and parallel processing research, with major publications in journals and conferences spanning more than 25 years. The co-author and editor of various books, his works include *Fault-Tolerant Computing: Theory and Techniques, Vols. I and II* (Englewood Cliffs, NJ: Prentice-Hall, 1986), *Fault-Tolerant Computer Systems Design* (Englewood Cliffs, NJ: Prentice-Hall, 1996), and *IC Manufacturability: The Art of Process and Design Integration* (Piscataway, NJ: IEEE Press).

Dr. Pradhan has served as Guest Editor of special issues in prestigious journals such as IEEE TRANSACTIONS. Also he has served as an editor for several journals, including IEEE TRANSACTIONS and *JETTA*. He has also served as General Chair and Program Chair for various major conferences. He has received several Best Paper Awards, including the 1996 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award, with W. Kunz, on "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems—Test, Verification, and Optimization." A Fellow of the ACM, he is the recipient of the Humboldt Distinguished Senior Scientist Award/Germany. Also, he was awarded the 1997–1998 Fulbright-Flad Chair in Computer Science.