Logic Verification Based on Diagnosis Techniques *

Andreas Veneris
University of Toronto
Dept. ECE and CS
Toronto, ON M5S 3G4
veneris@eecg.toronto.edu

Alexander Smith
University of Toronto
Dept. ECE
Toronto, ON M5S 3G4
smith@eecg.toronto.edu

Magdy S. Abadir
Motorola
7700 W. Parmer
Austin, TX 78729
m.abadir@motorola.com

Abstract

We present a formal logic verification methodology for combinational circuits. The method uses simulation, logic diagnosis and ATPG to identify circuit lines that implement equivalent logic functions efficiently. One advantage of the proposed technique is that it identifies line equivalences under controllability and observability don't care conditions, while not suffering from false negatives. The method is easy to implement, and, due to its general nature, existing techniques can benefit from ideas described here. We also give implementation details and present experiments to confirm its potential.

1 Introduction

The digital VLSI design cycle consists of many synthesis stages. Although most of the synthesis process is automated, errors may occur due to bugs in CAD tools and due to human interference [1]. These errors, if not detected early in the design cycle, may have large financial and time-to-market consequences. Hence, design verification is important to ensure the correctness of the final product. Verification is NP-hard, and it is unlikely that a unified solution exists for all hierarchical synthesis stages. For this reason, the research community develops efficient design-specific verification solutions [7].

The focus here is the problem of logic verification of combinational circuits that exhibit some degree of structural similarity to their originals. Such similarities exist because of the incremental nature of the synthesis process where consecutive logic synthesis steps can be viewed as atomic operations on a circuit to obtain a new one [1] [11] [13]. Designs that correspond to atomic operations close in this sequence of events are expected to have some degree of structural similarity.

Techniques that exploit such structural similarities identify pairs of lines in the two circuits that perform the same function. Such *equivalent line pairs* are "matched" progressively to reduce the complexity of the overall verification problem. To identify equivalences, random test vector simulation and implication learning [9] [7] are usually invoked. These methods are not guaranteed to identify all equivalences under controllability and observability don't cares [13]. This is because equivalent lines may have different logic values for the same test vectors as long as these vectors operate in their don't care space. To prove line equivalences, existing techniques use Binary Decision Diagrams (BDD's) [3] [8] [12], ATPG [10] [13], SAT solvers [6] or combinations of the above [11]. A survey of existing work is found in [7].

Motivated by these observations, we present a logic verification methodology for combinational circuits that uses simulation-based *Design Error Diagnosis and Correction (DEDC)* techniques [2] [17] and advances in ATPG [5] [9]. One of its unique characteristics is its ability to identify equivalent lines under *controllability and observability don't care conditions* while avoiding false negatives. This enhances the potential to simplify the verification problem. Furthermore, it may allow for future implementations that identify *multiple sets* of equivalent lines, a topic which has not been addressed by present literature.

The method identifies equivalent line pairs in two steps. At first, random test vector simulation is performed and candidate lines are matched according to their logic values. Additional candidate line pairs are found by injecting an error in one circuit and diagnosing the other circuit for lines that may explain the faulty behavior. A novel construction proves these equivalences

*The research of the first two authors was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Contract #227044–02.

formally using ATPG. It should be noted, the use of ATPG is not restrictive to the method and other formal equivalence tools can be used such as BDDs and SAT solvers. Additionally, due to the general nature of the operations performed by the proposed method, existing techniques can benefit from observations in this paper. Experiments for circuits that do not always bear a large degree of structural similarity demonstrate the effectiveness and practicality of the approach.

This paper is outlined as follows. The next Section reviews background information. Section 3 contains the method and implementation details. Experiments can be found in Section 4 and Section 5 concludes this paper.

2 Background

Logic *design errors* are functional mismatches between the specification and the gate–level description [2] [17]. Most literature uses a design error (correction) model, *i.e.* a small predetermined set of possible error types, proposed by Abadir et al. [2]. This model contains simple error types such as gate replacement, missing input wire, extra input wire etc.

In simulation-based DEDC, given an erroneous design, a specification, a design error model and a set of input test vectors, we need to identify lines in the design that are potential sources of error (*diagnosis*) and suggest appropriate modifications on these lines from the design error model used to rectify it (*correction*). Note that the set of logic transformations returned by a DEDC algorithm may contain some *equivalent* transformations along with the *actual* one. This is true because there may be multiple locations at which one can synthesize a particular function to rectify the design.

Test vector generation and verification of the rectified design in DEDC are inherently "hard" problems [2] [17] because the error location is not known. On the other hand, given input test vectors with failing output responses, there exist DEDC methods for single errors that are exhaustive on the solution space yet run in *linear time* [17]. In this work we use the simulation-based DEDC algorithm from [17] which is exhaustive on the solution space. The input to the algorithm is an erroneous netlist, its specification, and a set of input test vectors. The output of the algorithm is a list of *all* candidate locations with applicable corrections.

3 Proposed Approach

In principle, there are two components to a logic verification tool: *(i)* how to identify and prove the equivalence of candidate line pairs and *(ii)* how to use these

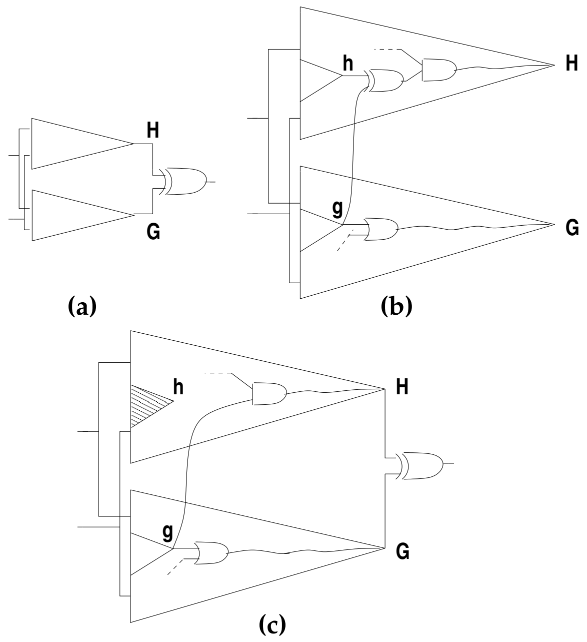

Figure 1: Logic verification in [3]

equivalences to simplify the problem.

For the second component, we follow a variation of the scheme proposed by Brand [3]. In this scheme, line equivalences are used to reduce *physically* the *size* (*i.e.* gatecount) of the circuits under verification. For example, and without loss of generality, consider two single primary output cones of combinational logic (circuits) H and G under verification, as in Fig. 1(a). In [3], an `XOR` gate is added at the primary output to form an *external miter.* It follows that the circuits are equivalent if the stuck-at-0 fault at the output of the miter is redundant.

However, proving such brute-force redundancies is a hard problem with a complexity that depends on the size of the miter and equals that of the original verification problem [3]. To reduce the size of the miter, [3] uses random test pattern simulation to identify potential equivalent lines such as h and g in Fig. 1(b). To prove their equivalence, an *internal miter* is formed on these lines (Fig. 1(b)) and ATPG is invoked for stuck-at-0 fault at the output of this `XOR` gate. If the fault is redundant, lines h and g are equivalent and g can replace h in circuit H. This replacement does not duplicate the logic that feeds g into H, but it simply replaces h with an extra branch that stems from g. Finally, the circuitry that feeds h is deleted, and the size of the external miter is reduced as shown in Fig. 1(c).

As in [3], our technique uses line equivalences to reduce the size of the miter. One main difference is that a *multiplexer* is used in the place of an `XOR` to form internal and external miters. At the end of this Section, we

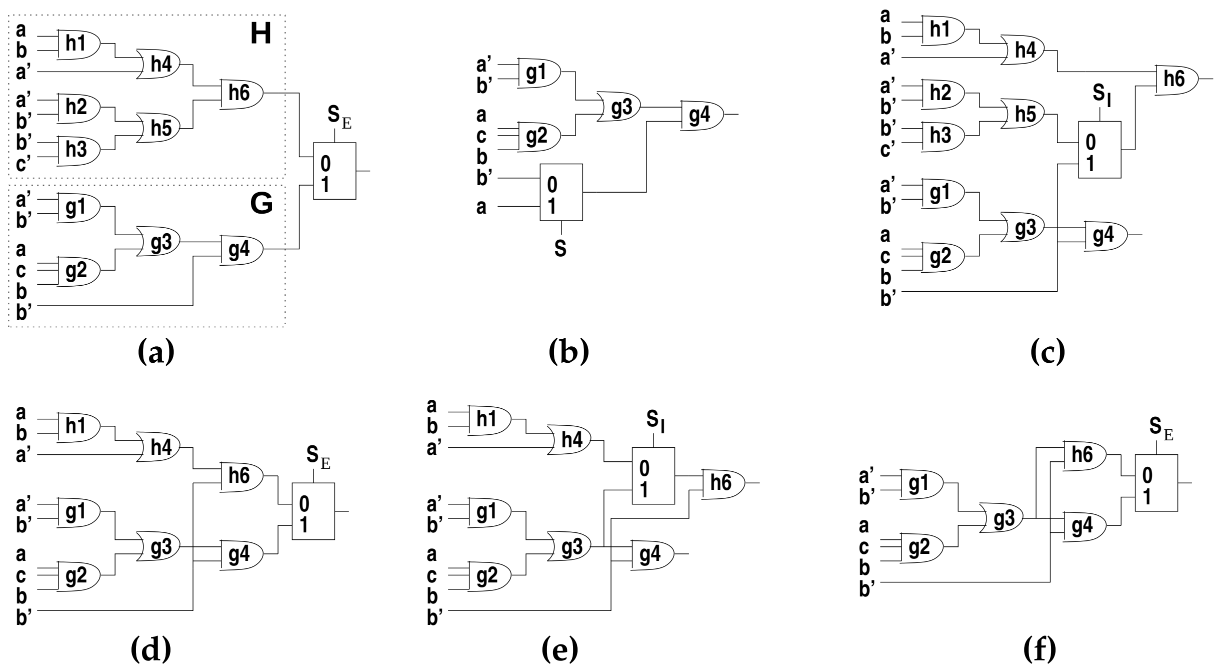

Figure 2: (a) Original external miter (b) Computing vectors for DEDC (c) Verifying (b', h_5) pair equivalence (d) New miter configuration when b' replaces h_5 (e) Verifying (g_3, h_4) pair equivalence (f) Final miter

discuss the advantages of this construction. In the subsection that follows, we present the overall strategy. In this discussion, we assume circuits H and G are under verification, where circuit H is always pruned to ease the verification process. We also use the same notation to indicate the name of a line as well as the function it implements.

3.1 Line Pair Selection and Verification

In the proposed approach, selecting a list of candidate equivalent line pairs and proving their equivalence (component *(i)*) is done in *three* phases. In the first phase, *parallel logic simulation* is performed for a small number of vectors (usually less than 1000 vectors). Indexed arrays of logic values are updated at each line during simulation, and they are hashed into a hash table. Lines that hash in the same hash table entry form the initial set of candidate equivalent line pairs as in [3] [8].

Although random simulation is useful, it does not guarantee the identification of equivalences under sets of controllability and observability don't cares [13]. To increase the number of candidate line pairs and improve the overall performance, in the second phase, the algorithm performs a sequence of *diagnosis-based operations* to discover more potential pairs. It should be noted that a diagnosis-based RTL-to-logic line equivalence searching technique is also presented in [14]. However, the purpose, use and implementation of the algorithm presented here stem from the diagnosis-based design rewiring work in [18] [19] and the technique is radically different from the one presented in [14]. In more detail, the algorithm operates as follows.

Let line $g \in G$ for which potential equivalent lines in circuit H need to be found. To find these equivalences, a (non-redundant) design error is injected at the gate driving g to implement erroneous function g'. In experiments, we introduce simple wire-related errors such as a wire removals or wire replacements. These error types usually alter the functionality of the circuit less than gate related errors and there is an increased likelihood to find equivalent transformations [17].

Following the error introduction, a multiplexer with select line S is attached; the inputs to the multiplexer are lines g and g' and the output is the original output of g. It can be shown [18] [19] that test vectors returned by ATPG for the S stuck-at-1 fault [1] are also vectors that detect the design error g'. These vectors are input to a DEDC engine and diagnosis is performed on circuit H for the vectors derived for the error in G. Clearly, any location $h \in H$ returned by DEDC is a line potentially equivalent to g. By construction [17], DEDC returns error locations that account for controllability and observability don't cares in linear time.

Given candidate equivalent line pair (g, h) we need prove their equivalence. To do this, the algorithm performs the construction in Fig. 1(b) in circuit H, with the exception that a multiplexer with select line S is used instead of an `XOR`. If ATPG confirms that a stuck-at 1 fault on S is redundant, then lines g and h are equivalent

[1]An alternative approach runs ATPG for S stuck-at 0 but the net effect is the same.

and the replacement (Fig. 1(c)) is performed. Additionally, since ATPG is employed for the complete circuit, the algorithm avoids false negatives [11].

In summary, the algorithm performs the following three steps to discover and prove equivalent line pairs:

Step 1: Collect candidate pairs using simulation

Step 2: Discover more candidate pairs as follows:

- **Step 2a:** Select line $g \in G$ and inject an error
- **Step 2b:** Derive test vectors for this error
- **Step 2c:** Use DEDC to diagnose H

Step 3: Prove the equivalence of all candidate pairs

The following example illustrates the implementation details for Steps 2 and 3 of the verification process.

Example: Consider circuits G and H under verification as shown in Fig. 2(a). In that figure, the external multiplexer-based miter is also shown. The ultimate goal is to prune the size of circuit H and simplify the complexity of a redundancy check on S_E.

In Step 2a, a line from circuit G is selected and a non-redundant design error is introduced on this line. Assume line $b' \rightarrow g_4$ is selected and "replace $b' \rightarrow g_4$ with $a \rightarrow g_4$" error is introduced. To derive test vectors for this error, a multiplexer is attached in circuit G as in Fig. 2(b). Inputs to this multiplexer are lines b' and a and its output feeds gate g_4. Vectors are collected if ATPG for S stuck-at-1 is performed (Step 2b).

In Step 2c, DEDC diagnoses circuit G using these vectors for candidate equivalent lines. If a DEDC algorithm exhaustive on the solution space is used, line h_5 is returned. In order to verify formally the equivalence of line pair (b', h_5), a multiplexer (internal miter) is attached with inputs h_5 and b' and output gate h_6, as shown in Fig. 2(c). Obviously, the redundancy of the S_I stuck-at 1 fault implies the equivalence of the two lines in H, and b' can replace h_5, as in Fig. 2(d).

Repeating Step 2, one may introduce an error on g_3 and have DEDC return candidate equivalent line h_4. Fig. 2(e) contains the hardware to verify the pair and Fig. 2(f) shows the final miter after this line replacement in H is also performed. It is observed that only one gate from H remains, which greatly simplifies the redundancy check of select line S of the external miter.

Observe that candidate equivalent pairs (b', h_5) and (g_3, h_4) cannot be found by implication-based techniques since no such relationship can be established between lines in the same pair. Similarly, random simulation will miss the equivalence of pair (g_3, h_4) $((b', h_5))$ if some of the test vectors $(a, b, c) = \{(0,1,0), (0,1,1), (1,1,0)\}$ $((1,0,1))$ are included.

During implementation, lines from H are selected to perform Steps 1...3 in topological order. The algorithm works iteratively, and at successive iterations it selects lines three circuit levels apart. If the equivalences discovered in the current level are not sufficient (set by a user-defined parameter), lines from the next immediate level are selected. This heuristic efficiently balances the number of times the ATPG engine needs to be called against the complexity of ATPG, which depends on the circuitry size it operates (Step 3). Finally, in Step 2c, the algorithm considers lines returned by DEDC that are associated with correction(s) rather than lines returned by diagnosis alone. These lines have a higher potential to qualify in Step 3 and reduce the total number of times ATPG is called.

In these last paragraphs we argue for the choice of the multiplexer to implement Step 3. When single line equivalences are considered, a multiplexer is equivalent to an `XOR`. This is not the case for *multiple and simultaneous line equivalence identification.*

For example, consider the two circuits G and H under verification in Fig. 3(a) and (b), respectively. It can be shown that circuit H is synthesized from G if $c \rightarrow g_1$ is deleted and gate h_4 is added. Assume two simple errors are injected on g_1 and g_4 *simultaneously* and vectors are collected as in Step 2b. If multiple error DEDC is performed in H then locations h_1 and h_5 are returned. Fig. 3(c) shows the construction, analogous to Step 3, to verify the equivalence of multiple lines $(< g_1, h_1 >, < g_4, h_5 >)$. For simplicity, in this figure we use line names to represent circuitry that implements respective boolean functions in Fig. 3(a) and (b). Since fault S_I stuck-at 1 is redundant, the line replacement can be performed and the verification problem is simplified considerably, as shown in Fig. 3(d).

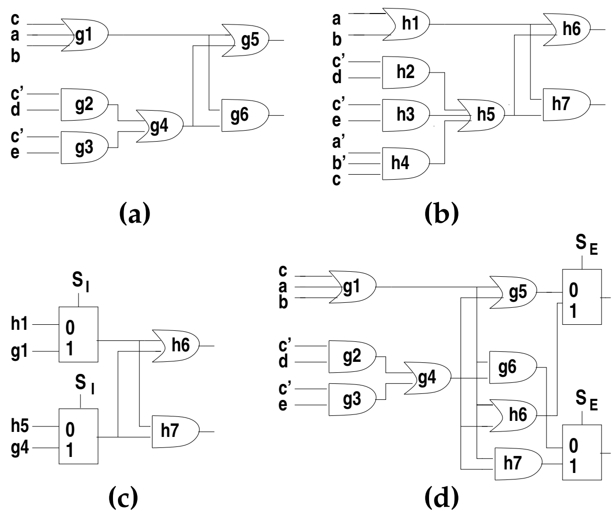

Figure 3: MUX-based verification

The multiplexer construction in Fig. 3(c) *cannot* be emulated with the use of XORs as these gates always produce a 1/0 fault-free/faulty logic value. Furthermore, the reader can verify that no single line equivalence (h_1, g_1) or $(h_5, g4)$ can be established independently. Therefore, multiple and simultaneous line equivalence identification methods, which have been shown to be useful in design rewiring [18] [19], may aid logic verification as well. In the future, we intend to investigate such methods and assess their performance.

4 Experiments

We implemented the algorithm in Section 3 using C and conducted experiments on a Sun UltraBlade 100 workstation with 128Mb of memory for ISCAS'85 combinational benchmark circuits that contain redundancies. Two experiments are reported in which the original circuit is verified against its optimized version. In the implementation, the ATPG and DEDC engines from [9] and [17] are employed.

Table 1: Circuits optimized by SIS

ckt name	gate count	# of repl.	% DEDC based	# all equiv.	# DEDC based	CPU (min:sec)
C432	232/0	35	28%	56	19	00:24
C880	383/0	49	46%	97	51	00:18
C499	618/104	83	33%	97	39	00:40
C1355	546/0	88	27%	504	147	00:19
C1908	880/360	33	24%	102	31	01:38
C2670	1193/0	90	19%	812	294	00:54
C5315	2307/631	155	38%	1433	374	04:03
C6288	2416/1712	62	11%	289	53	14:17

In the first set of experiments, circuits are optimized by SIS [15]. In the second set of experiments, benchmarks are first optimized by SIS and they are further optimized by *Design Compiler*$^{(TM)}$ (Synopsys) [16] for delay minimization. A relatively high mapping effort and a clock with a small period is used. As a result, the circuits are altered considerably and many similarities are eliminated. The average values of these experiments are presented in the next paragraphs.

Table 1 contains results for the first set of experiments. The first column contains the circuit name and the second column contains the number of internal gates for pruned circuit H at the start/end of the experiment. The algorithm always prunes the original circuit because it has more gates and redundancies which are hard for ATPG to tackle.

The *total* number of replacements performed is found in column 3 and the next column contains the percentage of these replacements discovered *only* by DEDC. These are equivalences that cannot be detected with random simulation or with implication techniques. We observe that a significant amount of equivalent pairs can be found only by the proposed diagnosis-based methodology. These equivalences allow for enough replacements to eliminate H and ease the final task of ATPG. This confirms the effectiveness of the approach.

The above observations are confirmed if we exhaustively count the *total number* of equivalences between the two circuits, shown in columns 5 and 6 of Table 1. Column 5 contains the total number of one-to-one equivalences for lines in H and G. The next column contains the number of these equivalences found only by the diagnosis-based method. We observe that many line equivalences are found only by the diagnosis-based method. The last column of Table 1 contains the run time. Most time is spent on redundancy checkings by ATPG and less than 2% of the time is spent in DEDC (Step 2c). The dominant ATPG factor is usually that of the external miter. Therefore, an implication-based preprocessing step for ATPG, such as the one described in [10] [13], is expected to improve these run times.

The first five columns in Table 2 contain data for the second experiment presented in a similar manner as above. Optimized circuits in this experiment undergo a major amount of resynthesis that destroys significant structural similarity. The values in this table indicate the effectiveness of the method in a real life synthesis environment. Column 6 of Table 2 shows the normalized speed-up of the overall verification effort due to the DEDC-based approach. It is seen that the speed-up is considerable in many cases. However, for some circuits, such as C432, C880 and C1355, simulation-based verification alone (Steps 1 and 3) is able to outperform the proposed combined approach (Steps 1...3). This is because the overall gain induced by removing gates using DEDC-based Step 2 is less than the time spent in Step 2 itself.

In the future, we plan to investigate different parameters and heuristics involved with the method. These include the type of errors injected and their effect in the search process as well as different line selection strategies. We also plan to explore the idea of similarity enhancing logic transformations as described in [13]. Finally, we expect to experiment with multiple and simultaneous candidate line equivalences as described earlier in Section 3.

5 Conclusions

We presented a method for logic verification of combinational circuits with a degree of structural similarity. To identify equivalent lines between the two designs, novel simulation-based and diagnosis-based techniques

Table 2: Circuits optimized by Synopsys

ckt name	gatecount before/after	# of repl.	% diag. based	CPU (min:sec)	normalized speed-up
C432	232/0	23	44%	00:48	0.82
C880	383/0	40	51%	00:37	0.69
C499	618/412	22	71%	09:22	1.34
C1355	546/270	54	27%	06:34	0.91
C1908	880/595	31	43%	12:11	1.45
C2670	1193/0	117	15%	02:08	1.72
C5315	2307/473	241	33%	10:58	1.41
C6288	2416/2071	41	27%	21:01	1.09

are described. These equivalences are later used to reduce the complexity of the verification problem. Experiments demonstrate the potential of the approach.

In the future, we plan to experiment with different parameters involved such as the error types injected and the line selection process. We also plan to investigate multiple and simultaneous line equivalences as well as similarity enhancing transformations presented in [13].

References

[1] E. J. Aas, K. Klingsheim and T. Steen, "Quantifying design quality: a model and design experiments," in *Proc. of EURO–ASIC*, pp. 172–177, 1992.

[2] M. S. Abadir, J. Ferguson and T. E. Kirkland, "Logic Verification Via Test Generation," in *IEEE Trans. on Computer–Aided Design,* vol. 7, pp. 138–148, January 1988.

[3] D. Brand, "Verification of large synthesized designs," in *Proc. IEEE Int'l Conf. on Computer–Aided Design, pp. 534-539,* 1993.

[4] R. E. Bryant, "Graph–based algorithms for Boolean function manipulation," in *IEEE Trans. on Computers*, vol. C–35, no. 8, pp. 677–691, 1986.

[5] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," in *IEEE Trans. on Computers, vol. C-32, no. 12,* December 1983.

[6] E. Goldberg, M. Prasad and R. Brayton, "Using SAT for Combinational Equivalence Checking," in *Proc. of IEEE Design and Test in Europe (DATE), pp. 114-121,* 2001.

[7] J. Jain, A. Narayan, M. Fujita and A. Sangiovanni-Vincentelli, "A Survey of Techniques for Formal Verification of Combinational Circuits," in *IEEE Int'l Conf. on Computer-Aided Design, pp. 445-454,* 1997.

[8] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. of IEEE Design Automation Conference, pp. 263-267,* 1997.

[9] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems–Test, Verification, and Optimization," in *IEEE Trans. on Computer-Aided Design, vol. 13, no. 9, pp. 1143-1158* September 1994.

[10] W. Kunz, "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning," in *IEEE Int'l Conf. on Computer-Aided Design, pp. 538-543,* 1993.

[11] W. Kunz, D. K. Pradhan and S. M. Reddy, "A Novel Framework for Logic Verification in a Synthesis Environment," in *IEEE Trans. on Computer-Aided Design, vol. 15, no. 1, pp. 20-32* September 1996.

[12] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *Proc. IEEE Design Automation Conference, pp. 629-634,* 1996.

[13] D. Paul, M. Chatterjee and D. K. Pradhan, "VERILAT: Verification Using Logic Augmentation and Transformations," in *IEEE Trans. on Computer-Aided Design, vol. 19, no. 9 pp. 1041-1051,* 2001.

[14] S. Ravi, I. Ghosh, V. Boppana and N. K. Jha, "A Technique for identifying RTL and gate-level correspondences," in *IEEE Int'l Conf. on Computer Design, pp. 591-594,* 2000.

[15] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. of Int'l Conference on Computer Design, pp. 328-333,* 1992.

[16] Synopsys "Design Compiler," available from *http://www.synopsys.com/products/logic /design_compiler.html*, 2002

[17] A. Veneris, and I. N. Hajj, "Design Error Diagnosis and Correction Via Test Vector Simulation," in *IEEE Trans. on Computer–Aided Design,vol. 18, no. 12, pp. 1803–1816,* December 1999.

[18] A. Veneris, M. S. Abadir and I. Ting, "Design Rewiring based on Diagnosis Techniques," in *Proc. of IEEE Asia and South Pacific Design Automation Conf., pp. 479-484,* 2001.

[19] A. Veneris and M. S. Abadir, "Design Rewiring Using ATPG," in *IEEE Trans. on Computer-Aided Design,* in press.