Design Optimization Based on Diagnosis Techniques

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

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

Ibrahim N. Hajj
University of Illinois
ECE Department and CSL
Urbana, IL 61801
hajj@uivlsi.csl.uiuc.edu

Abstract

Logic optimization is the step of the VLSI design cycle where the designer performs modifications on the design obtained to satisfy different constraints such as area, power or delay. In this paper we propose a novel ATPG-based optimization methodology that borrows from previous design error diagnosis and correction techniques. We also present examples and experiments that indicate that our approach has additional potential when compared to previous ATPG/simulation-based optimization methods.

1 Logic Design Optimization

The design of digital circuits usually starts with a behavioral description (specification) coded in some high level description language. At some point of the design cycle, logic synthesis is carried out to obtain a multi-level implementation (netlist) using cells (gates) from some technology library. As a next step, **logic optimization** is usually performed, where the objective is to modify the netlist to achieve different constraints such as minimize the area, reduce power consumption, satisfy timing constraints, reduce switching noise or improve the testability of the final circuit. Recently, ATPG/simulation-based optimization techniques [6] [7] [8] [9] [10] [11] [16] have gained increasing popularity. Their main strength, compared to formal Boolean function (*i.e.* symbolic) [4] based optimization methods, lies in their flexibility since they are memory efficient, they have good failure characteristics and although, theoretically, they can be exponential in time, in practice this is rarely the case. In this research we propose a new approach to ATPG-based design optimization. The proposed approach borrows from our previous efforts on **Design Error Diagnosis and Correction (DEDC)** and it provides a more "powerful" route to ATPG-based optimization.

To illustrate the central points of the proposed research we need to take a closer view on existing ATPG-based optimization techniques and the problem of DEDC. This is the topic of sections 2 and 3, respectively. In section 4 we examine the relationship of these two problems and present our method. Section 5 contains the experimental results and section 6 concludes this paper.

2 Design Optimization Via Rewiring

ATPG-based design optimization techniques are usually based on design rewiring as they try to optimize a given netlist with a sequence of simple **logic transformations** such as adding/deleting a single wire, a gate ..., etc. A significant amount of optimization through rewiring is performed through *redundancy addition and removal* [6] [7] [8] [9] [10] [16]. These methods target a wire that violates certain specification constraints and attempt to remove it by either adding redundant logic (so that the target wire becomes itself redundant), or by blocking all possible sensitized paths that exist from the target wire to primary outputs. The following example, taken from [9], outlines the major steps of the single wire redundancy addition/removal design optimization method.

Example: Consider the simple circuit in Figure 1(a) where wire $g5 \rightarrow g9$, indicated by a dotted line, is not part of the original netlist. Also assume that wire $w_T = g1 \rightarrow g4$, named *target wire* hereafter, needs to be removed because it violates some specification constraint(s). In redundancy addition/removal, the target wire w_T is removed without altering the function(s) implemented at the primary output(s) by adding a new redundant connection that makes w_T redundant. Such a redundant connection is wire $w_R = g5 \rightarrow g9$, shown as a dotted line. Under the presense of w_R, wire w_T becomes redundant and new redundancies are introduced in the circuit such as wire $g_6 \rightarrow g_7$. As a next step, w_R is added and redundancies w_T and $g_6 \rightarrow g_7$ are removed as well as logic that no longer has an influence on the primary outputs such as gates g_6 and g_4, wire $g_4 \rightarrow g_8$, etc. The new, simplified, circuit is shown in Figure 1(b) where wire w_T has been removed and the gate count has been reduced leading to an optimized design.

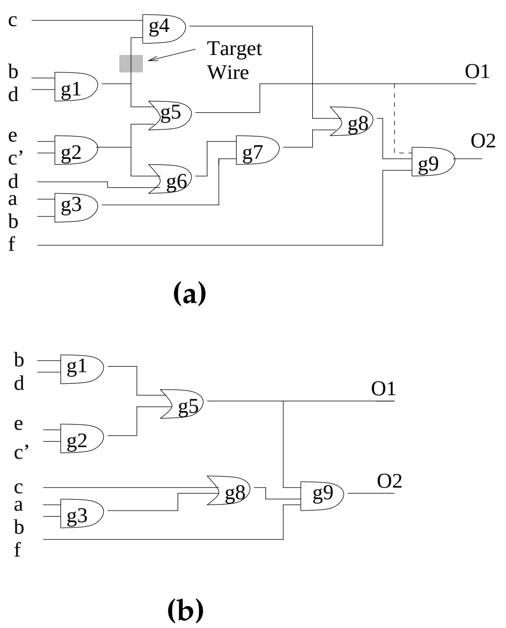

Figure 1: Optimization through rewiring

3 Design Error Diagnosis and Correction

Logic *design errors* may occur during the design cycle of a VLSI chip when the designer has to adjust manually the netlists generated by synthesis tools in order to satisfy specific design goals. Design errors are functional mismatches between the specification and the gate–level description. Experimental data has shown that the nature of the design errors usually involves the functional misbehavior of some gate element(s) and/or some interconnection error(s). The average number of design errors is 2. Most of the literature, including our work in the field [12] [13], uses a design error (correction) model, *i.e.* a small predetermined set of possible error types, proposed by Abadir et al. [1]. This model contains ten different types of possible design errors such as missing inverter, extra inverter, gate replacement, extra gate, extra input wire, missing input wire, ... etc. For example, removing w_T from the circuit in Figure 1(a) gives a missing input wire design error.

DEDC is the problem where given an erroneous design and a specification we need to identify lines in the design that are potential sources of error (diagnosis) and suggest appropriate modifications, from the design error model, that rectify it (correction).

In [12] [13] we present a run-time efficient method for multiple DEDC. The input to the algorithm is an erroneous design and its specification. The output of the algorithm is a list of *all* applicable corrections that rectify the design from the design error model and the test vector set that it is used. Since there may be more than one way to synthesize a particular function, there may be *equivalent* corrections, along with the actual one, that rectify the design. The existence of equivalent corrections is of paramount importance for the potential of the proposed optimization method. With respect to the example in Figure 1(a), removing target wire w_T and running the DEDC algorithm described in [12] [13], it returns connection $g_5 \rightarrow g_9$, among others, in the list of equivalent corrections.

3.1 The Signle Error Case

In this subsection we review our DEDC [12] [13] methodology for single design errors as it will be used for the optimization process described in section 4.

Diagnosis uses the path–trace procedure, a line marking algorithm developed for fault diagnosis by Venkataraman et al. [14] [15] that is based on the critical path tracing algorithm [2]. In detail, let circuit C corrupted with a design error and input test vector v with failing primary output response(s) for C. Path–trace [14] [15] starts from an *erroneous* primary output and traces backwards toward the primary inputs marking lines as follows: if the output of a gate G has been marked and G has one or more fan–in(s) with controlling values then the procedure randomly marks any one controlling fan–in; if G has all fan–ins with non–controlling inputs, then all fan–ins are marked; if a branch is marked, then the algorithm automatically marks the stem of the branch.

For example, consider the circuit of Fig. 2 with primary outputs O_1 and O_2. If O_1 has been marked by path–trace, it can proceed by marking set of lines $S_1 = \{O_1, G_4, G_3, B_1, I_2\}$. If path–trace begins from O_2, set of lines $S_2 = \{O_2, B_4, G_2, B_2, I_2\}$ are marked.

It can be shown [12] [13], that given an erroneous circuit C and a set of input test vectors $v_1, v_2, \ldots, v_k$ each of them failing primary output responses for C, every line that may contain a design error is guaranteed to be in the intersection of the lines marked by distinct path-trace runs for different erroneous primary outputs

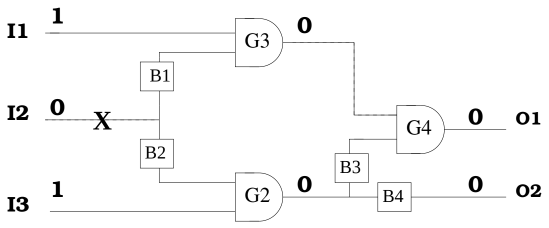

Figure 2: Single design error diagnosis and correction

and vectors v_j, $j = 1, \ldots, k$. Again, consider the circuit of Fig. 2 and assume that there is an inverter missing on line I_2. In such case, input vector $(1, 0, 1)$ produces erroneous responses at both primary outputs O_1 and O_2 and every line with a design error belongs in the intersection of the set of lines S_1 and S_2 marked by path–trace, namely $S_1 \bigcap S_2 = \{I_2\}$.

To compile a list of candidate error sites during *diagnosis*, we simply run path-trace for the vectors $v_1, v_2, \ldots, v_k$ and different failing primary outputs and we return the intersection of the lines marked at every distinct run of the procedure.

For error *correction*, the algorithm exhaustively compiles a list of all possible corrections for every candidate that qualified diagnosis. In detail, for every candidate error line l it applies all modifications of the model of Abadir et al. [1], one at a time, on the gate driving l and keeps the correction if and only if it gives complemented logic values for every vector v_j, $j = 1, \ldots, k$ at the output of the gate.

4 Design Optimization Based on Diagnosis Techniques

A closer view of the two problems described above shows that the procedure of rewiring can be viewed from a DEDC perspective as follows. Suppose that target wire w_T needs to be removed. Instead of attempting to find a redundant connection that makes w_T redundant, we can actually remove the target wire and artificially introduce a "design error". Unlike DEDC where the location of the design error is unknown and test vector generation (in the general case) is a difficult problem [1] [3] [13], in the situation described above, the location of the error is known and we can derive input test–vectors with a PODEM-like procedure [5] to detect it. These vectors can be used to run a DEDC algorithm and obtain a set of "corrections" that rectify the design. Subsequently, the algorithm selects an appropriate correction that satisfies the optimization criteria from the list of *equivalent* corrections and simulation-based verification of the final design is performed.

It should be noted that in the DEDC context, verification, along with test vector generation, of the final design with the use of test vector simulation is a "hard" problem [1] [3] [13]. In detail, it is theoretically [1] and experimentally [3] [13] proven that unless exhaustive test vector simulation is performed, which is practically unrealistic for most commercial designs, a logic verifier is required at the back-end of a DEDC method to guarantee the "quality" of the proposed corrections. However, this is not practical for a DEDC-based optimization algorithm that iterates for a significantly large number of target logic and a formal verifier at the back-end of each iteration will degrade the overall performance significantly. To alleviate this problem we develop run-time/memory efficient ATPG-based techniques that guarantee the correctness of the final design. These techniques are described in subsection 4.1.

In a sense, the optimization procedure described above (remove/add logic) works in the opposite direction from what existing ATPG/simulation-based optimization techniques do (add/remove logic). However, when operating in this new direction, our approach has the potential to lead to greater optimization gains since, as illustrated in the example of subsection 4.2, it can return logic transformations ("corrections") that are *not* necessarily redundant in the presense of the target wire ("design error") w_T.

4.1 Method Overview

The proposed optimization method comprises of four main steps: (1) introduce a design error by removing the target logic, (2) derive test vectors for this design error, (3) use a DEDC algorithm to search for a correction that rectifies the design, and (4) verify the correctness of the final design. All four steps are based on the results of an ATPG-like algorithm.

In the paragraphs that follow we discuss the implementation of the different steps of our method. During this discussion, we assume that non-redundant target wire w_T is fan-in to gate G_T and G'_T is the name of that gate when w_T is removed. We also assume that G_A is a gate where we can fan-in a wire w_A, resulting in gate G'_A.

As explained earlier, the first step of the algorithm artificially introduces a design error in the circuit by removing target wire w_T. Next, a 2-input multiplexer is added in the circuit with fan-ins the outputs of gates G_T and G'_T. To derive vectors that detect the design error we run a PODEM-like [5] procedure for S stuck-at 1 or 0, where S is the select line of the multiplexer. Each of the vectors returned by the ATPG algorithm will also be a vector that distinguishes between the old (correct) and new (erroneous) circuit.

To implement the third step of our method we run the simulation-based DEDC algorithm outlined in subsection 3.1. The input to this algorithm is the correct and erroneous circuit along with the test vectors derived in step (2). The output of this algorithm is a list of all (actual and equivalent) corrections that rectify the design for all the test vectors returned by the ATPG tool. Missing input wire w_A is contained in this list of equivalent corrections.

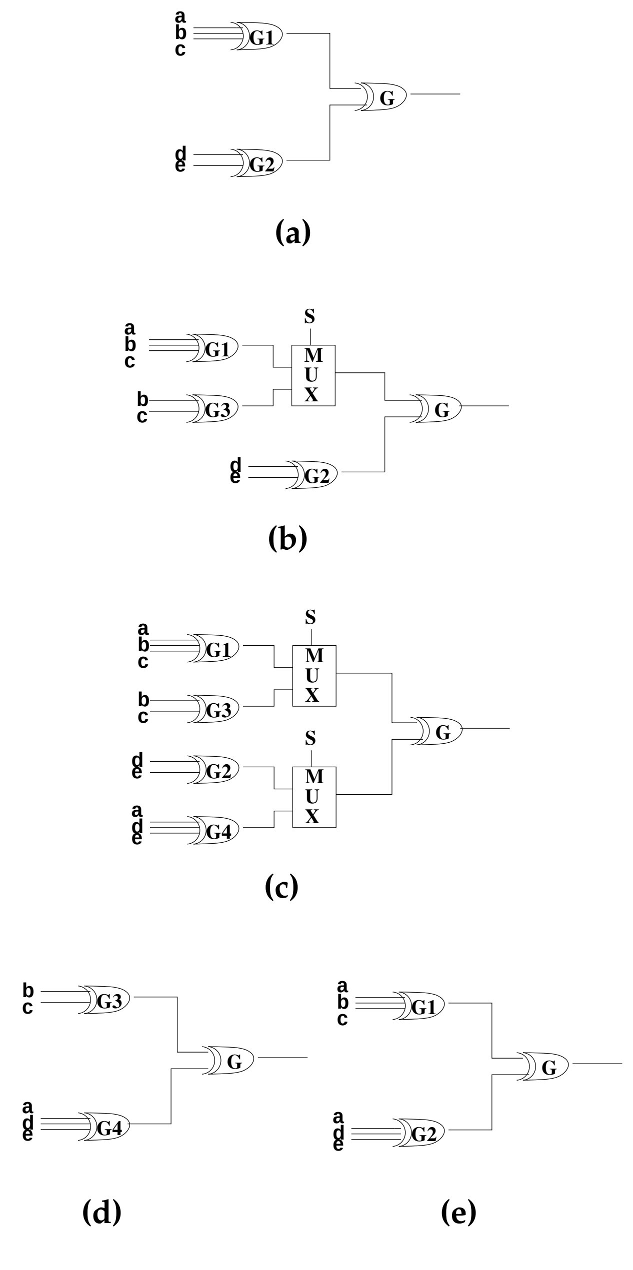

Figure 3: A Circuit with XOR gates (a) Original Design (b) Deriving Test Vectors (c) Circuit Correctness (d) Final Design (e) Incorrect Design

As a last step, we need to verify the correctness of the final design when w_A is added and w_T is removed. To perform this verification step, we attach a second 2-input multiplexer to the fan-outs of G_A and G'_A with the same select line S and run the PODEM-like [5] procedure for S stuck-at 1 or 0. It is clear that if the procedure returns with a non-empty test-vector set it indicates that the new circuit (w_T removed, w_A added) is *incorrect*. However, if the procedure returns with failure, *i.e.* cannot find vectors for the stuck-at fault, then we guarantee the correctness of the new design.

4.2 Example

In this section we give an example to illustrate the steps of the algorithm described in subsection 4.1. This example also indicates the added potential of our approach compared to previous redundancy addition/removal logic optimization methods. With respect to the circuit in Fig. 3(a) assume that $w_T = a$, that is, fan-in a of gate $G_T = G_1$, needs to be removed.

During first two steps of the algorithm, gate $G'_T = G_3$ is introduced where w_T is removed along with a multiplexer MUX that fan-ins the outputs of G_1 and G_3 and has select line S, as shown in Fig. 3(b). As explained in the previous subsection, input test vectors that detect the fault S stuck-at 1 or 0 are also vectors that give erroneous primary output responses when w_T is removed from the circuit in Fig. 3(a). Without loss of generality, we assume that the ATPG tool returns with vector $V = (a, b, c, d, e) = (1, v, v, v, v)$, where v indicates some well-defined value (0 or 1).

During the third step of our algorithm, DEDC [12] [13] is performed. The input to the DEDC algorithm is the original circuit (Fig. 3(a)), the erroneous one (Fig. 3(a) with w_T removed) and vector set V. The DEDC algorithm returns with the actual error (missing input wire a to G_1) and equivalent correction missing input wire $w_A = a$ fan-in to G_2.

To verify the correctness of the final design when target logic w_T is removed and the equivalent correction is applied, step (4) requires the addition of gate G_4 and a second multiplexer with the same select line S as the first one that fan-ins G_2 and G_4, as shown in Fig. 3(c). A PODEM-like [5] procedure for fault S stuck-at 1 or 0 returns with failure, indicating that the fault is redundant. Therefore, the circuits in Fig. 3(a) and Fig. 3(d) implement the same function at the primary output and the correction qualifies.

Note, that the solution returned by the proposed method cannot be found by a redundancy addition/removal optimization procedure, as demonstrated in Fig. 3(e). It is clear that added wire w_A is not redundant in presense of w_T and our solution would

Table 1: Experimental Results

ckt name	# of primary inputs/outputs	# of lines	# w_A same gate	# w_A diff. gate	success ratio	max. # of w_A returned	average CPU time
C432	36/7	545	8.4	10.0	100 %	31	5.7
C880	60/26	880	1.3	2.0	85 %	9	7.9
C499	41/32	1224	0.0	4.8	40 %	15	10.7
C1355	41/32	1355	0.6	5.0	40 %	12	14.2
C1908	33/25	1908	3.4	4.5	90 %	10	19.4
C2670	157/63	2670	4.6	26.3	80 %	192	17.3
C5315	178/123	5315	3.3	2.8	45 %	14	21.9
C7522	207/108	7552	4.5	1.6	70 %	9	35.1

not be considered by existing single redundancy addition/removal methods.

5 Experiments

We implemented our algorithm in C language and ran it on an Ultra 5 Sparc workstation for the ISCAS'85 benchmark circuits. We ran 20 experiments for each circuit. During each experiment we randomly picked and deleted an input wire to a gate and we counted the number of alternate wires that can correct the design. In the case that the gate originally had 2 fan-ins we replaced the `AND/OR` (`NAND/NOR`) gate with a `BUFFER` (`NOT`). The average values of the results of our experiments are reported in Table 1. All run–times are in seconds.

The first three columns of Table 1 contain circuit characteristics, *i.e.* the name, number of primary inputs/outputs and the number of internal lines for each circuit. In each of our experiments, we first randomly pick the target wire w_T which is fan-in to gate G_T and run the algorithm described in subsection 4.1 to find alternate wires w_A that can fan-in to a gate G_A and "correct" the design. In the context of DEDC [12, 13], we randomly select and delete a wire, introducing a design error in the design, and then we attempt to find equivalent missing input wire corrections. We divide the corrections we find in two distinct families, the ones that fan-in to the original gate ($G_T = G_A$) and the ones that fan-in to a different gate ($G_T \neq G_A$). The average number of wires that fan-in to the same gate can be found in column 4 of Table 1 while the average number of wires that fan-in to a different gate is contained in column 5. These numbers are the average values over the number experiments where our algorithm returned with a non-empty number of equivalent corrections (leq 20). The number of times that our approach could not find alternate wires to correct the design is contained in column 6. It is among our future research plans to develop a family of more complex logic transformations, such as adding multiple input wires and/or gates, to perform correction when a single wire addition is not sufficient to rectify the design.

In column 7 we present the maximum number of alternate wires that our approach returned throughout these 20 experiments for each circuit and the last column contains the overall run–time to find all equivalent corrections.

6 Conclusions

We presented a novel ATPG-based logic optimization methodology. Our methodology targets a wire that violates some optimization constraints and attempts to find alternate connections with the use of existing design error diagnosis and correction procedures. We also proposed efficient techniques to guarantee the correctness of the final design and we proved, with the use of an example, that the proposed method exhibits additional potential for optimization gains when compared to existing single wire redundancy addition/removal procedure. This is also confirmed by our experimental results.

In the future, we intend to develop and apply a family of more complex logic transformations during the correction stage. It is also among our future research plans to evaluate the performance of our method on circuits were the gates have been decomposed, as described in [8], and for different optimization goals such as power and delay.

References

[1] 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.

[2] M. Abramovici, P. R. Menon and D. T. Miller, "Critical path tracing: an alternative to fault simulation," in *IEEE Design and Test of Computers*, vol. 1, pp. 89–93, February 1984.

[3] H. A. Asaad, and J. Hayes, "Design verification via simulation and automatic test pattern generation," in *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design*, pp. 174–180, 1995.

[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] P. Goel, "An Implicit Enumeration Algorithm to Generate Test for Combinational Circuits," in *IEEE Trans. on Computers, vol. C-30, pp. 215–222,* March 1981.

[6] S. C. Chang, K. T. Cheng, N. S.Woo, and M. M.-Sadowska, "Layout Driven Logic Synthesis for FPGAs," in *Proc. of ACM/IEEE Design Automation Conference, pp.308–313,* 1994.

[7] S. C. Chang and M. Marek-Sadowska, "Perturb and Simplify:Multi-Level Boolean Network Optimizer," in *Proc. Int'l Conf. on Computer-Aided Design, pp. 2-5,* 1994.

[8] S. C. Chang, J. C. Chuang, and Z. Z. Wu, "Synthesis for Multiple Input Wires Replacement of a Gate for Wiring Consideration," in *Int'l Conf. on Computer–Aided Design, pp. 115–118,* 1999.

[9] L. A. Entrena, and K. T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal," in *IEEE Trans. on Computer–Aided Design of Integrated Circuits and Systems, vol.14, no.7, pp.909-916,* July 1995.

[10] R. C-Y. Huang, Y. Wang, and K. T. Cheng, "Libra — A Libra-Independent Framework for Post-Layout Performance Optimization," in *Proc. Int'l Symp. on Physical Design, pp. 135-140,* 1998.

[11] M. Chatterjee, D. Pradham, and W. Kunz, "LOT: Logic Optimization with Testability – New Transformations Using Recursive Learning," in *Int'l Conf. on Computer–Aided Design, pp. 115–118,* 1995.

[12] A. Veneris, S .Venkataraman, I. N. Hajj and W. K. Fuchs, "Multiple design error diagnosis and correction in digital VLSI circuits," in *Proc. of IEEE VLSI Test Symp.*, pp. 58-63, 1999.

[13] 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.

[14] S. Venkataraman, I. Hartanto and W. K. Fuchs, "Dynamic diagnosis of sequential circuits based on stuck–at faults," in *Proc. of the VLSI Test Symp.*, pp. 198–203, 1996.

[15] S. Venkataraman and W. K. Fuchs, "A deductive technique for diagnosis of bridging faults," in *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design*, pp. 562–567, 1997.

[16] Q. Wang, and S. B. K. Vrudhula, "Multi–Level Optimization for Low Power Using Local Logic Transformations," in *Proc. ACM/IEEE Design Automation Conference, pp.270–277,* 1996.