Design Rewiring 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

Ivor Ting
Alcatel Corporation
4190 Still Creek Drive
Burnaby, BC
ivor.ting@alcatel.com

Abstract

Logic optimization is the step of the VLSI design cycle where the designer performs modifications on a design to satisfy different constraints such as area, power or delay. Recently, ATPG-based design rewiring techniques for logic optimization have gained increasing popularity. In this paper we propose a novel ATPG-based design rewiring methodology that borrows from previous design error diagnosis and correction techniques. We also present examples and experiments that indicate the added potential of our approach which is expected to provide a "powerful" route to design optimization.

1 Introduction

Logic optimization is the step of the VLSI design cycle where the designer modifies the netlist obtained by synthesis tools 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-based optimization techniques [5] [6] [7] [8] [9] [11] have gained increasing popularity. Their main strength when compared to symbolic [3] 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 general, ATPG-based techniques attempt to optimize the design through a sequence of *design rewiring* operations. During each iteration of this procedure, some simple redundant logic is added in the design so that the target logic becomes redundant itself and it can be removed. This process is repeated until the desired optimization constraints are satisfied.

In this paper we propose a new approach for ATPG-based multilevel combinational circuit design rewiring. This new approach borrows from previous *Design Error Diagnosis and Correction (DEDC)* techniques [2] [10]. In detail, the proposed methodology works in the opposite direction to what existing ATPG-based design rewiring procedures do as it first removes the target logic, thus introducing a design error, and then it attempts to correct the design with the use of a simulation-based DEDC algorithm.

It should be noted that in this paper we do not propose an algorithm that targets a specific optimization goal but we introduce a new methodology to design rewiring. We also show that this approach has the potential to provide a "powerful" route to design optimization that can add and complement on existing techniques. It is among our future research plans to extend the results presented here and apply this new methodology to specific design optimization problems.

The contribution of the work presented here is twofold. First, as we demonstrate in subsection 4.2, the logic transformations returned by this new approach are always a superset of the ones suggested by existing ATPG-based design rewiring techniques. In other words, the proposed approach has the potential to do a better use of the internal don't cares of the design when it is applied to design optimization. Next, as discussed in section 6, we expect that such a method will help perform efficiently the process of multiple simultaneous logic transformations which has traditionally been computationally intensive for existing ATPG-based optimization techniques [6].

To effectively describe the proposed approach we need to take a closer view on existing ATPG-based design rewiring 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

*This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Contract #227044–00.

we present our method. Section 5 contains the experimental results and section 6 concludes this paper.

2 Previous Work

ATPG-based design optimization techniques try to optimize a given netlist with a sequence of simple *logic transformations* [5] [6] [7] [8] [9] [11]. During each iteration, a target wire that violates some specification constraint(s) is selected for removal and a redundant logic transformation is performed to remove that wire. These transformations usually include the addition of some simple redundant logic, such the introduction of a new gate with support from existing lines in the circuit or an additional fan-in to an existing gate. The effect of this extra redundant logic is to make the target wire redundant so it can be removed. Existing ATPG-based methods mainly differ in the structure of the new logic that they introduce at each iteration of this design rewiring process and in the underlying ATPG-based algorithm that is used to derive it.

The following example, taken from [8], outlines the major steps of a 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 $w_A = g5 \rightarrow g9$, shown as a dotted line. Under the presense of w_A, wire w_T becomes redundant and new redundancies are also introduced in the circuit such as wire $g_6 \rightarrow g_7$. As a next step, w_A 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, optimized, circuit is shown in Figure 1(b) where wire w_T has been removed and the gate count has been reduced.

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 [1] [2]. These 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) [1]. In addition, it has been reported that the average number of design errors that may occur during a manual resynthesis procedure is less than or equal to 2 [1].

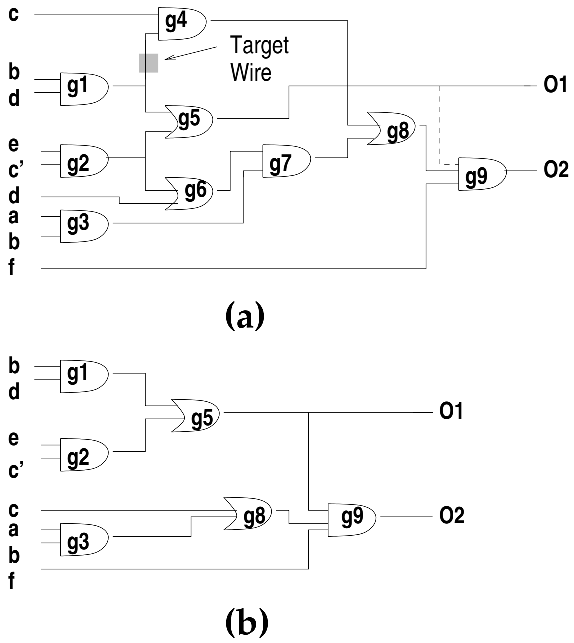

Figure 1: Optimization through rewiring

Most of the literature in the field 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 ten different types of possible design errors such as missing or extra inverter, gate replacement, extra gate, extra or 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 it is used, to rectify it (correction) [2].

There has been a significant amount of published work on the DEDC problem. In Section 4 we make use of a simulation based DEDC algorithm for single design errors. The input to the algorithm is an erroneous netlist, its specification and a set of vectors with failing primary output responses. The output of the algorithm is a list of *all* applicable corrections from the design error model presented in [2] that rectify the design. 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.

With respect to the example in Figure 1(a), removing target wire $w_T = g_1 \rightarrow g_4$ and running the DEDC algorithm described in [10], it returns $g_5 \rightarrow g_9$ as the only correction in the set of equivalent corrections. The existence of equivalent corrections is of paramount importance for the potential of the design rewiring method presented next.

4 Design Optimization Based on Diagnosis Techniques

A closer view of the two problems described in Sections 2 and 3 shows that the procedure of design rewiring with simple logic transformations 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". Since the location of the error is known, we can derive input test–vectors with an ATPG procedure such as the one described in [4] 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 from the list of *equivalent* corrections and ATPG-based verification of the final design is performed. In the context of design optimization, an appropriate correction is the one that satisfies the current optimization constraint(s).

It should be noted that in the DEDC context, test vector generation for the erroneous design along with simulation-based verification of the final design is a "hard" problem [2] [10] because the location of the design error is not known. In detail, it is theoretically [2] and experimentally [10] proven that unless exhaustive test vector simulation is performed 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 design rewiring algorithm that iterates for a large search space. To alleviate this problem we propose an ATPG-based technique to guarantee the correctness of the final design. This technique is described in subsection 4.1 and it is based on the fact that in DEDC-based design rewiring the location of the design error is actually known.

In a sense, the design rewiring procedure described above (remove/add logic) works in the opposite direction from what existing ATPG-based design rewiring techniques do (add/remove logic). However, as shown in subsection 4.2, when design rewiring operates in this new direction it can return logic transformations ("corrections") that are *not* necessarily redundant in the presense of the target wire ("design error") w_T. In other words, the proposed methodology takes better advantage of the internal don't cares of the design.

4.1 Method Overview

The proposed design rewiring method comprises of four main steps:

Step 1: introduce a design error by removing the target logic

Step 2: derive test vectors for this design error

Step 3: use a DEDC algorithm to search for a correction that rectifies the design

Step 4: verify the correctness of the final design.

In this subsection and in the example presented in subsection 4.2, we discuss the implementation of the different steps of our method in terms of a simple logic transformation where we remove a target wire ("design error") and we attempt to rectify the design by adding some other wire ("correction"). In our experiments, presented in Section 5, we allow for a larger family of permissible logic transformations during correction, all of them a subset of the design error model proposed by Abadir et al. [2].

As explained earlier, the first step of the algorithm artificially introduces a design error in the circuit by removing target wire w_T. In the next paragraphs, we assume that non-redundant target wire w_T is input 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 add as an input wire w_A, resulting in gate G'_A. Next, a 2-input multiplexer is added in the circuit with inputs the outputs of gates G_T and G'_T and select line S. To derive vectors that detect the design error for the second step we run an ATPG algorithm [4] for S stuck-at 1. Each of the vectors returned by the ATPG algorithm will also be a vector that distinguishes between the old (correct) and new (erroneous) circuit since, by construction, the output of the multiplexer implements the function at G_T when $S = v$ and the function at G'_T when $S = \overline{v}$ ($v = 0$ or 1).

To implement the third step of the method we run a simulation-based DEDC algorithm that returns all equivalent corrections. 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. Let wire w_A as an additional fan-in to gate G_A be a member of this list of equivalent corrections proposed by the algorithm to rectify the design.

Nevertheless, as explained earlier in this section, considering that DEDC bases its results on a small subset

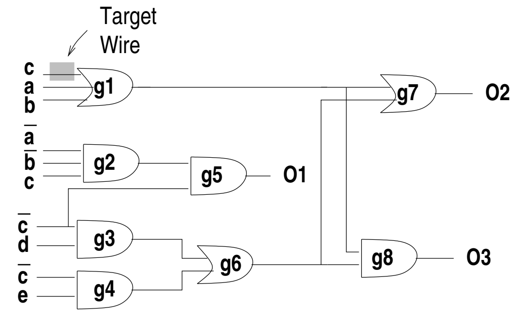

Figure 2: Original Circuit

of the complete input test vector space, we need to verify the correctness of the final design when w_A is added in the circuit. To perform this verification process (step 4 of the algorithm), we attach a second 2-input multiplexer to the fan-outs of G_A and G'_A with the same select line S so that when $S = v$ we select the old circuit and when $S = \overline{v}$ we select the new circuit. It is clear that if we run an ATPG procedure [4] for S stuck-at 1 that 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 test set is empty (*i.e.,* S stuck-at 1 is redundant) then that guarantees the correctness of the new design.

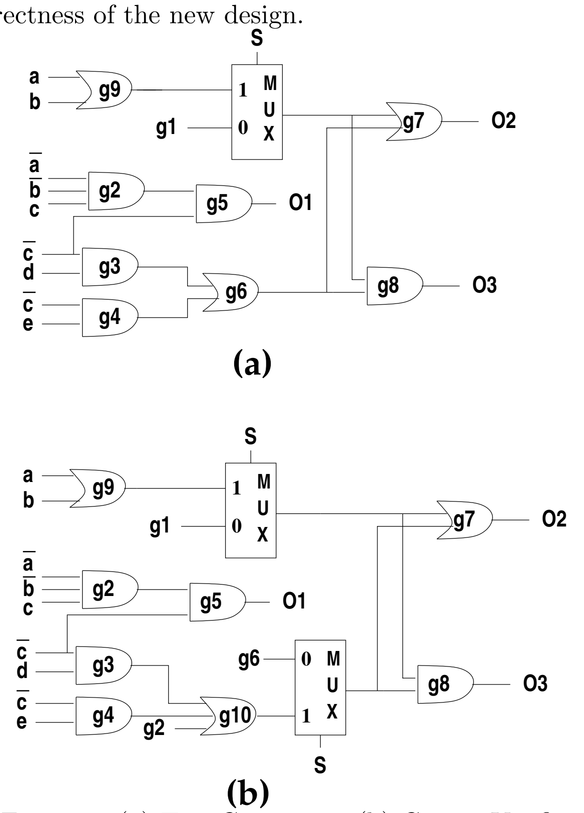

Figure 3: (a) Test Generation (b) Circuit Verification

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. With respect to the circuit in Fig. 2 assume that line c, input to gate $G_T = g_1$, is the target wire, *i.e.* $w_T = c$ needs to be removed.

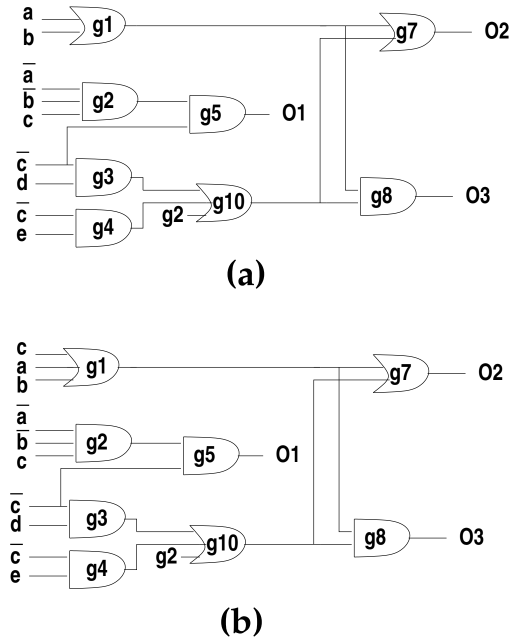

Figure 4: (a) Final Design (b) An Erroneous Design

During the first two steps of the algorithm, shown in Fig. 3(a), gate $G'_T = g_9$ is introduced, that is, a gate similar to g_1 with w_T not present in its support. A multiplexer MUX with inputs the outputs of g_1 and g_9 and select line S is also introduced. For simplicity, in Fig. 3(a) we use g_1 to indicate the circuitry that implements the respective boolean function in Fig. 2. As explained in subsection 4.1, an input test vector set $\mathcal{V}$ that detects the fault S stuck-at 1 is also a set of vectors that each of them gives erroneous primary output responses when w_T is removed from the circuit in Fig. 2.

During the third step of our algorithm, DEDC is performed. The input to the DEDC algorithm is the original circuit (Fig. 2), the erroneous one (Fig. 2 with w_T removed) and vector set $\mathcal{V}$. The DEDC algorithm returns with the actual error (missing input wire c to g_1) and a set of equivalent corrections that contains a missing input wire $w_A = g_2$ as an input to g_6.

To verify the correctness of the final design, we introduce gate g_{10} with input lines g_3, g_4 and g_2, that is, the equivalent to gate g_6 in Fig. 2 with w_A added as an input. We also introduce a second multiplexer with the same select line S as the first one and inputs g_6 and g_{10}, as shown in Fig. 3(b). An ATPG procedure [4] for fault S stuck-at 1 returns with an empty test set indicating that the fault is redundant. Therefore, the circuits in Fig. 2 and Fig. 4(a) implement the same function at the primary outputs and the correction qualifies.

Observe that the solution returned by the proposed method cannot be found by a conventional single wire redundancy addition/removal procedure. This is demonstrated in Fig. 4(b) where both w_A and w_T are present in the circuit. However, this circuit does not implement the same boolean function at the primary outputs with the original one (Fig. 2) as input test vector $(a, b, c, d, e) = (0, 0, 1, 0, 0)$, for example, causes a failing response at O_3. Therefore, w_A is not redundant in presense of the target wire w_T.

5 Experiments

We implemented our algorithm in C and Perl and ran it on an Ultra 5 SUN workstation for the ISCAS'85 benchmark circuits. The details of the ATPG and DEDC algorithms that we employed can be found in [4] and [10], respectively. We ran several hundreds of experiments for these circuits where we randomly picked and removed a target wire and counted the number of logic transformations, other than the original, that correct the design. We allowed three types of permissible logic transformations from the design error model proposed by Abadir et al. [2]:

- *Type 1:* change the type of gate G
- *Type 2:* an additional fan-in to gate G
- *Type 3:* replace an existing fan-in of gate G with some other wire

For logic transformations types 2 and 3, we consider adding wires that do not create loops in the combinational circuitry. In the context of DEDC, we randomly selected and deleted a wire to introduce a design error in the design and then we attempted to find the number of equivalent corrections from a design error model that contains three design error types. During our experiments, if an `AND`/`OR` (`NAND`/`NOR`) gate originally had 2 inputs, we replaced the gate with a `BUFFER` (`NOT`) when we inject the error. The average values of the results of our experiments are reported in Tables 1 and 2. All run–times are in seconds.

Table 1

ckt name	# of primary inputs/outputs	# of lines	success ratio	avg. CPU time
C432	36/7	545	35.8 %	6.1
C880	60/26	880	10.2 %	7.7
C499	41/32	1224	14.4 %	14.1
C1908	33/25	1908	80.3 %	18.9
C2670	157/63	2670	36.4 %	19.3
C3540	157/63	3540	80.7 %	25.3
C5315	178/123	5315	28.9 %	20.8
C7522	207/108	7552	47.1 %	40.5

Table 2

ckt name	corr. type 1	corr. type 2	corr. type 3	avg. corr. same gate	avg. corr. diff. gate	max # corr.
C432	3%	10%	87%	3.1	2.2	6
C880	2%	3%	95%	3.3	1.5	7
C499	0%	0%	100%	1.8	0.0	14
C1908	0%	2%	98%	4.7	5.2	18
C2670	1%	27%	72%	3.1	0.8	19
C3540	0%	36%	64%	5.6	1.3	25
C5315	0%	1%	99%	2.0	0.5	20
C7522	0%	12%	88%	9.5	9.4	40

The first three columns of Table 1 contain circuit characteristics. The success ratio, that is, the number of times our approach found alternate corrections (*i.e.,* non-empty set of equivalent corrections) from the set of the three types of permissible logic transformations is contained in column 4. It should be noted that although the success ratio for circuit C499, for example, is low, this is also the case with existing redundancy addition/removal techniques [6] which implies that some circuits are hard to optimize through a sequence of simple logic transformations. To increase the success ratio, it is among our future plans to develop a family of more complex logic transformations that will serve specific optimization goal(s), as explained at the end of this section.

The last column in Table 1 contains the average run–time to find *all* equivalent corrections during one experiment. This number contains the CPU time for all four steps of the algorithm as described in subsection 4.1. It should be noted that during the second and fourth step of the algorithm we set a maximum limit (user defined) of backtracks the ATPG tool [4] can perform before it aborts the fault (steps 2 and 4). Throughout our experiments, the ATPG tool was always able to derive test vectors during step 2. In less than 15% of the cases (on the average) it had to abort the fault during step 4. Finally, if we are interested for only one equivalent correction then the CPU time to find this correction is a small fraction of the time reported in column 5.

In Table 2 we give more detailed information on the amount and type of logic transformations performed during our experiments. The numbers presented in this table are the average values over the total number of experiments where the algorithm returned with a non-empty number of equivalent corrections. Columns 2, 3 and 4 contain the percentage of each permissible logic transformation type performed that rectified the design during all the successful experiments. We also divide the equivalent corrections we find in two distinct families, the ones that were performed on the original gate

($G_T = G_A$) and the ones that were performed on a different gate ($G_T \neq G_A$). The average number of logic transformations that can be performed on the original gate can be found in column 5 while the average number of corrections on a different gate is contained in column 6. Finally, column 7 presents the maximum number of equivalent corrections that our approach returned throughout the experiments for every circuit.

As it can be seen from the experimental results of this section, certain types of corrections are more commonly used than others. It is among our future plans to introduce a larger family of permissible transformations during correction so that we can increase the correction success ratio and serve different optimization goals. Once such a larger family of permissible transformations has been established we will also be able to experimentally compute the gains of this method over existing redundancy addition/removal techniques since the three types of corrections we allow our algorithm to perform at this stage of our work are simpler when compared to the re-synthesis performed during redundancy addition/removal [5] [6] [7] [8] [9] [11].

6 Conclusions

We presented a novel ATPG-based design rewiring methodology. Our methodology targets a wire and attempts to find alternate corrections 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 showed that the proposed method exhibits additional potential for optimization gains when compared to existing redundancy addition/removal procedures. In addition, considering the efficiency of existing multiple DEDC techniques, we expect that this new methodology will aid in the development of effective CAD methodologies that perform multiple logic transformations simultaneously.

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 where the gates have been decomposed, as described in [6], and for different optimization goals.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] B. Rohfleisch, A. Kolbl and B. Wurth, "Reducing Power Dissipation after Technology Mapping by Structural Transformations," in *Proc. of Design Automation Conference, pp. 789-794,* 1996.

[10] 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.

[11] 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.