# Design Rewiring for Power Minimization

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

Mandana Amiri
University of Toronto
Dept ECE
Toronto, ON M5S 3G4
amiri@eecg.toronto.edu

Ivor Ting
Alcatel
4190 Still Creek Drive
Burnaby, BC V5C 6C6
ivor.ting@alcatel.com

## Abstract

A new ATPG-based approach to multi-level combinational logic circuit power optimization is presented. This method borrows from existing simulation-based design error diagnosis and correction techniques. At every step of the optimization procedure, a design error is introduced by removing logic with high switching activity and it is corrected by modifying the design in a less critical area. The experiments presented here indicate the added flexibility and potential of this approach.

## 1 Introduction

During **logic optimization** a digital design is modified to satisfy different constraints. Increasing demand for portability and reliability impose strict constraints on power consumption. Therefore logic optimization for low power is an active research area. [5] [6] [10].

In general, logic optimization tools can be classified to those that use Boolean function manipulation methods and those that use Automatic Test Pattern Generation (ATPG). Recently, ATPG-based optimization techniques have gained increasing popularity. In general, ATPG-based techniques attempt to optimize the design through a sequence of *design rewiring* operations based on the concept of redundancy addition/removal [3] [5] [10]. During each iteration of this redundancy addition/removal procedure, the algorithm identifies some target logic and then it attempts to remove it. Most of the times, the target logic is a single wire that violates some optimization constraint(s). Redundancy addition/removal methods attempt to remove this wire with the addition of some simple redundant logic somewhere else in the design. This new logic usually includes the addition 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 that it can be removed. This iterative design rewiring process is repeated until the desired optimization goals are met.

In this paper we describe a new ATPG-based design rewiring methodology for low-power design of multi-level combinational circuits. The proposed method borrows from Design Error Diagnosis and Correction (DEDC) techniques [1] [9]. This method works in the opposite direction to what existing ATPG-based design rewiring procedures do as it first removes target logic with high switching activity, thus introducing a design error, and then it attempts to correct the design in some less critical area with the use of a simulation-based DEDC algorithm. The main advantage of this approach, as explained in [8], is that the corrections returned are a superset of the ones returned by existing techniques.

The rest of this paper is organized as follows. Section 2 briefly reviews the rewiring method used here. Next, in Section 3, we describe an efficient application of this design rewiring methodology to design for low power. Experimental results can be found in Section 4. Finally, Section 5 concludes this paper.

## 2 Design Rewiring Based on Diagnosis Techniques

In order to effectively describe the design rewiring algorithm we utilize for low-power design, we need to take a closer look on the problem of DEDC. We do this in the paragraphs that follow. Next, we review our DEDC-based design rewiring methodology [8].

Logic *design errors* are functional mismatches between the specification and the gate–level description. Most of the literature 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, missing input wire, extra input wire, gate replacement etc. 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) [1]. Note that the set of corrections returned by a DEDC algorithm may contain some *equivalent* corrections along with the actual one [9]. The existence of equivalent corrections is of paramount importance for the potential of the design rewiring method.

In the remainder of this Section we review our design rewiring methodology in terms of a single wire removal, *i.e.* we assume that the target logic to be removed is a single wire $w_T$. The idea can be easily generalized for removal of more "complex" logic, as explained in Section 3.

In our ATPG-based design rewiring methodology the process of eliminating some target logic $w_T$ is viewed from a DEDC perspective. Instead of attempting to find a redundant connection that makes $w_T$ redundant, we actually remove the target wire and artificially introduce a "design error". Subsequently, we run a DEDC algorithm and select an appropriate correction that satisfies the optimization constraint(s) from the list of equivalent corrections, if any. Therefore, the method performs the following four steps to remove $w_T$: *(i)* introduce a design error by removing $w_T$, *(ii)* use ATPG [2] to derive test vectors for this design error, *(iii)* use a simulation-based DEDC tool to find equivalent correction(s) that rectify the design, and *(iv)* choose a correction and use ATPG [2] to verify the correctness of the final design.

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, when it works in this new direction, it has the potential to return corrections that are *not* necessarily redundant in the presense of the target wire $w_T$ [8]. In other words, the proposed methodology takes better advantage of the internal don't cares of the design.

# 3 Design for Low Power

In this section we describe the application of the design rewiring method to design for low power of combinational circuits that have been already mapped to an existing technology library. This procedure is an iterative execution of the design rewiring algorithm presented in Section 2 with some modifications as outlined below:

**Step I:** If the list of candidate target logic is empty exit otherwise randomly select target logic $w_T$ and introduce a design error to eliminate it.

**Step II:** Apply the design rewiring algorithm to get a set of equivalent corrections $\mathcal{S}$.

**Step III :** Select an equivalent correction from $\mathcal{S}$ and apply the correction. If $\mathcal{S}$ is empty then go to Step I.

**Step IV:** Remove new potential circuit redundancies [3][7] and compute power gain. If power gain is positive go to Step I and delete $w_T$ from candidate list otherwise go to Step III.

In subsections 3.1 and 3.2 we present the heuristics used to eliminate the target logic ("design error") and the correction model and algorithm used during DEDC, respectively. During this presentation, we say that a line $l$, fan–in to an `AND` or `NAND` (`OR` or `NOR`) gate, has *controlling value* for input vector $v$ if the value of $l$ is 0 (1).

## 3.1 Elimination of Target Logic

The transition density [4] of a gate is a measure of its switching activity [4] and consequently its power consumption. In order to reduce the power consumption, we try to reduce the transition density. Therefore, when selecting a target wire, we give priority to target logic $w_T$ that fan-outs gates with a high transition density value.

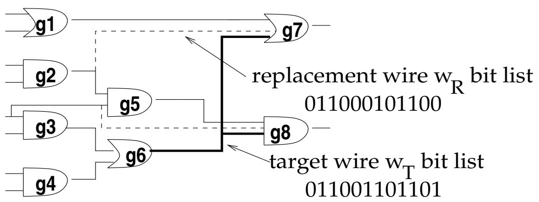


Figure 1: Power Optimization

In addition, power gain may result because of the removal of the dangling circuitry that is dominated [1] by the gate that drives $w_T$ [6] [10]. For example, if we eliminate (the fan-out of) gate $g_6$ in Fig. 1 we can also eliminate gates $g_3$ and $g_4$.

In order to describe our heuristics we assume a preprocessing step where we simulate a number of random vectors on the original circuit and we keep a bit-list on every line $l$ of the design. The $i$-th entry of this list is the logic value of $l$ under simulation of the $i$-th input vector. We say that two lines have *similar* logic values if most of their respective bit-list entries are the same. For example, Fig. 1 shows the bit-lists of two lines, $g_2 \rightarrow g_7$ and $g_6 \rightarrow g_8$ when we simulate 12 input vectors. The logic value of $g_6 \rightarrow g_8$ is 0 for the 1st vector, 1 for the 2nd vector etc. These two lines have similar logic values since their bit-lists differ in only two positions. Throughout our experiments we compile these bit-lists with simulation of 3,000-10,000 random input vectors and we update them during every iteration of the optimization algorithm with parallel logic simulation at the fan-out cone of the design error and the correction. We also specify the desired level of bit-list similarity via a user defined parameter which ranges from 20 to 35 percent.

To eliminate the target logic (*"design error"*) we either ($i$) remove $w_T$ or ($ii$) we replace $w_T$ with some other existing wire $w_R$ that does not cause a loop in the combinational circuitry.

Our first heuristic suggests that $w_R$ should have similar logic values to $w_T$. The underlying intuition is that if we replace $w_T$ with a wire with similar logic values, we alter the functionality of the circuit less and the DEDC algorithm will be able to return with a larger number of equivalent corrections. To increase the number of wires with bit-lists similar to $w_T$ we also consider the inverted version of wires with very different bit-lists. In addition, among all similar wires, we give priority to a $w_R$ with a high probability of zeros (ones) if it will be connected to

[1] We say that gate $G$ dominates gate $G'$ if and only if every path from $G'$ to any primary output goes through $G$.

the input of an AND/NAND (OR/NOR) gate. A wire with a high probability of zeros (ones) will dominate the logic value at the output of an AND/NAND (OR/NOR) gate since this is the controlling value of the gate and it will result in a lower transition density at the output of the gate. A metric of the probability of zeros (ones) of a line $l$ can be easily calculated with the use of the bit-list of $l$.

We also give higher priority to stem candidates for the target logic $w_T$. One approach is to find an alternative wire for the stem itself. Another approach is to attack each branch of the stem separately and find (possibly different) alternate wires $w_R$ for each one of them, as shown in Fig. 1 (dotted lines). We give preference to the latter approach as it guarantees that the set of alternative wires returned is always a superset of those returned by the former.

Finally, when we fail to find a valid correction(s) for $w_T$, we skip $w_T$ during the next few iterations of the algorithm since the same target wire is likely not to perform well in circuits that are structurally similar.

### 3.2 Correction

During the *correction* stage, DEDC tries four types of logic transformations from the design error model of Abadir et al.[1]: (*i*) gate replacement (*ii*) deletion of an existing fan-in of a gate $G$, (*iii*) addition of a new fan-in to a gate $G$, and (*iv*) replace an existing fan-in of a gate $G$ with some other wire. For type (*iii*) and (*iv*) transformations the algorithm considers wires that do not cause a loop in the combinational circuitry.

The correction algorithm that we use extends the DEDC results presented in [9]. To implement this algorithm, we maintain two bit-lists at each line $l$ in the circuit, the $l_{corr}$ and $l_{err}$ list. These bit-lists are created during a test vector simulation step that precedes DEDC. During this step, we simulate a set of vectors $\mathcal{V}$ which comprises of the vectors returned by ATPG plus a number (3,000-10,000) of random vectors. At the end of this simulation step, the $i$-th bit of $l_{corr}$ contains the logic value of $l$ during simulation of the $i$-th vector with correct primary output responses that also produces a sensitized path from $l$ to some primary output. Similarly, the $i$-th bit of the $l_{err}$ list contains the logic value of $l$ during simulation the $i$-th vector with failing primary output responses. These bit-lists need to be re-calculated via simulation at each iteration of the optimization algorithm for different target logic.

For every candidate error line $l$ that qualified diagnosis the algorithm applies logic transformations on the gate that drives $l$. Then it performs one *local* simulation step using the bit-lists at the fan-in of the gate and keeps the correction only if it gives complemented $l_{err}$ values but maintains all $l_{corr}$ values [9].

Considering that the number of candidate new fan-ins may be large, to improve on performance, we use the following correctability theorem to check whether such a transformation has the potential to correct the design.

**Theorem 1:** *Let suspicious* AND/NOR *(*OR/NAND*) gate $G$ that drives line $l$. $G$ can be corrected by a type (iii) transformation if every bit entry in $l_{err}$ is a logic 1 (0).*

*Proof:* Without loss of generality assume that $G$ is an AND gate and let 0 be the value of the $i$-th entry of $l_{err}$ for some $i$. By construction of $l_{err}$ this also means that the $i$-th failing input vector produces a logic 0 on $l$ and there is a sensitized path for this vector from $l$ to some primary output. However, this also indicates that some existing fan-in(s) of $G$ already has the controlling value 0 when we simulate this vector and no single additional fan-in to $G$ could complement the value of $l$ and correct the design.

If gate $G$ qualifies Theorem 1 the algorithm searches for a new fan-in that can be added and correct the design. Theorem 2 provides a necessary and sufficient (for $\mathcal{V}$) condition for a candidate wire to qualify as an additional fan-in. We omit the proof of this theorem as it is along the same lines with the one for Theorem 1.

**Theorem 2:** *Let suspicious gate $G$ with controlling value $c$ that drives line $l$. A new fan-in $l'$ to $G$ corrects the design if and only if it has:*

- *value $c$ in all $l'_{err}$ bit entries, and*
- *value $\overline{c}$ in every $l'_{corr}$ entry where its respective $l_{corr}$ entry is 1 (0) if $G$ is an* AND/NOR *(*NAND/OR*) gate*

To increase the number of candidate fan-in wires we also consider adding a wire which is the fan-out of a new gate with two fan-ins from existing wires in the circuit. We calculate these new functions in a brute-force manner by setting a user-defined limit on the maximum number of the new two input gates we introduce.

We treat type (*iv*) logic transformations as a two step process. First we remove the suspicious fan-in $l'$ from $G$. This removal may complement some of the entries in the $l_{err}$ and $l_{corr}$ lists at the output of $G$. To update these bit-lists, for all lines in the circuit, we place all $l_{err}$ entries that $l'$ had a controlling value for $G$ in $l_{corr}$ and all $l_{corr}$ entries that $l'$ was the *only* fan-in with controlling value for $G$ in $l_{err}$. This step can be performed efficiently with the use of appropriate masks. Once this is performed, we treat this correction as a type (*iii*) logic transformation.

## 4 Experiments

We implemented the method described in Sections 2 and 3 using Perl and C and ran on a Sun Ultra 5 workstation. We used ISCAS'85 combinational circuits. The details of the ATPG and DEDC algorithms that we employed can be found in [2] and [9], respectively. Our results on power minimization are shown in Table 1. In this table, run-times are given in minutes and power consumption is measured in microwatts.

During the second step of the design rewiring algorithm (Section 2) we set a maximum limit (user defined) of back-tracks the ATPG tool [2] can perform before it aborts the fault. We apply the same reasoning during the verification Step 4 of the algorithm but we accept the correction as valid once ATPG [2] reaches this limit. For this reason, we formally verify the designs at the end of the overall optimization process. During our experiments, every final (optimized) design we obtained proved to be functionally equivalent to its original.

Table 1: Results on Power Minimization

| ckt name | # of corr. | original pwr (uW) | final pwr (uW) | pwr decr. | area decr. | delay incr. | hit ratio | time (min) |
|---|---|---|---|---|---|---|---|---|
| C432 | 21 | 0.02529 | 0.01975 | 21.88% | 12.9% | 5.9% | 34% | 7.6 |
| C499 | 37 | 0.08947 | 0.08003 | 10.5% | 3.9% | 6.1% | 19.5% | 12.4 |
| C880 | 39 | 0.05595 | 0.04699 | 16.0% | 1.1% | 1.7% | 17.2% | 13.6 |
| C1355 | 55 | 0.08479 | 0.07297 | 13.9% | 1.2% | 5.1% | 43.7% | 21.2 |
| C1908 | 68 | 0.08565 | 0.06906 | 26.3% | 2.7% | 4.7% | 29.9% | 33.2 |
| C2670 | 96 | 0.14712 | 0.10912 | 24.1% | 18.6% | 4.2% | 56% | 48.1 |
| C3540 | 6 | 0.23360 | 0.21020 | 10.1% | 1.6% | 6.9% | 41.7% | 33.4 |
| C5315 | 33 | 0.39137 | 0.37378 | 4.5 % | 0.2 % | 6.3% | 36% | 54.0 |
| C7552 | 7 | 0.70403 | 0.60001 | 14.6% | 2.6% | 5.3% | 30.6% | 34.3 |

During power minimization, we initially optimize the circuits for area with SIS [7], using *script.rugged*. Next, we employ MED [4] to obtain the transition density of each gate. Column 1 of Table 1 contains the circuit name and column 2 has the number of transformations our algorithm performs on every circuit. Columns 3, 4 and 5 contain the original power, the final power and the percentage of improvement, respectively. As it can be seen, on the average, 15.0% of power reduction is achieved. The next two columns contain the change incurred in the area and delay due to the power optimization transformations. As expected, the area of the circuit is positively affected (4.9% on the average) but the delay increases (7.1% on the average) as we do not take into account any delay consideration during power optimization. It is among our future plans to extend this method and respect delay constraints. This can be achieved if we let the method skip logic transformations that reduce the power but violate the delay constraints, as described in [6] [10].

The success ratio, that is, the number of times our approach found a non-empty set of equivalent corrections during the optimization process from the set of permissible logic transformations is contained in column 8. It should be noted that although the success ratio for circuit C499, for example, is low, this is also the case with existing design rewiring techniques [3] [5]. This implies that some circuits are hard to optimize through a sequence of simple logic transformations. The final column of Table 1 contains the CPU time.

As it is shown in Table 1, the optimization gain is significant in all benchmark circuits. We expect that as we develop more heuristics the results will further improve. In more detail, it is among our future research plans to introduce a larger family of permissible transformations during correction. Our goal is to eventually develop a simple simulation-based re-synthesis algorithm during correction and eliminate the need for a fixed correction model. We expect that this will increase the success ratio and improve on optimization gains.

# 5 Conclusions

We presented a new methodology for logic re-synthesis that targets low power. The method is an iterative procedure where logic with high switching activity is removed to introduce a design error. This error is then fixed in some less critical area so that it gives a design with less power consumption. We also described the theory that allows for an efficient implementation of this new methodology and presented experiments that exhibit its effectiveness and robustness.

# References


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

[2] 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.

[3] S. C. Chang and M. Marek-Sadowska, "Perturb and Simplify: Multi-Level Boolean Network Optimizer," in *Proc. ICCAD, pp. 2-5,* 1994.

[4] F. Najm and M. Xakellis, "Statistical Estimation of Switching Activity in Digital Circuits," in *Proc. of DAC, pp. 728–733,* 1994.

[5] D. K. Pradhan, M. Chatterjee, M. V. Swarna and W. Kunz, "Gate-Level Synthesis for Low-Power Using New Transformations," in *Proc. of ISLPED, pp. 297–300,* 1996.

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

[7] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. of ICCD, pp. 328-333,* 1992.

[8] A. Veneris, M. S. Abadir and I. Ting, "Design Rewiring based on Diagnosis Techniques," in *Proc. of ASP–DAC, pp. 479–484,* 2001.

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

[10] Q. Wang, and S. B. K. Vrudhula, "Multi–Level Optimization for Low Power Using Local Logic Transformations," in *Proc. of DAC, pp.270–277,* 1996.