ATPG Driven Logic Synthesis for Area and Power Minimization

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

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

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

Abstract

In this paper we present the application of our ATPG-based design rewiring approach to multi-level combinational logic circuit optimization. At every step of this optimization procedure, we introduce a design error by removing the logic that violates the optimization constraint(s) and then we attempt to correct the design by modifying the logic somewhere else. We give heuristics and describe the application of this method to delay optimization and to design for low power. Experiments are also presented to support the potential of our method.

1 Introduction

Logic optimization is the step of the VLSI design cycle where a netlist is modified to satisfy 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 have gained increasing popularity [3] [4] [5] [7] [9] [10] [12]. In general, ATPG-based techniques attempt to optimize the design through a sequence of *design rewiring* operations. During each iteration of this 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). For example, this wire may be part of the critical path or it may have inappropriately high switching activity. ATPG-based optimization methods attempt to remove this wire with the addition of some simple logic somewhere else in the design. This iterative design rewiring process is repeated until the desired optimization constraints are satisfied.

In [9] [10] we present a *diagnosis-based* design rewiring method. This method works in the opposite direction than *redundancy addition/removal* techniques [3] [4] [5] [7] [12] as it first removes the target logic, thus introducing a design error, and then it uses a Design Error and Diagnosis and Correction algorithm (DEDC) [11] to rectify the erroneous design. In the same work [9] [10] it is also shown that this method always returns a set of logic transformations that is a superset of the ones returned by redundancy addition/removal methods. In this paper we describe the application of this diagnosis-based design rewiring method [9] [10] to multi-level combinational circuit optimization. In detail, we apply our approach to delay optimization and to design for low power and we present heuristics that improve its performance. This work also contains experiments that exhibit the potential of this new logic optimization methodology.

The rest of this paper is organized as follows. In the Section that follows, we review the steps of our design rewiring method. Next, in Section 3, we describe its application to design optimization for area and power minimization. Experimental results can be found in Section 4. Section 5 contains the conclusions and future research directions.

2 Design Rewiring Based on Diagnosis Techniques

In order to effectively describe our design rewiring methodology 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 [9] [10].

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 [11]. 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 as follows. 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 [9] [10]. In other words, the proposed methodology takes better advantage of the internal don't cares of the design.

3 Design Optimization

In this section we describe the application of the design rewiring method presented in Section 2 to delay optimization and to design for low power. In both cases the algorithm is the same as we first identify the target logic w_T and we introduce a design error by eliminating w_T. Next, we run DEDC to search for an equivalent correction(s). Once a correction is found, we calculate the new optimization objective for the design and we keep the correction if there is an improvement. This procedure is repeated for different target logic up to the point that no further optimization gain can be obtained.

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. During the *correction* stage, DEDC tries three types of logic transformations: (i) addition of a new fan-in to a gate G, (ii) deletion of an existing fan-in of a gate G, and (iii) replace an existing fan-in of a gate G with some other wire. The algorithms for delay and power optimization, presented in the subsections that follow, differ in the criteria and heuristics during the selection of the target wire w_T and the replacement wire w_R.

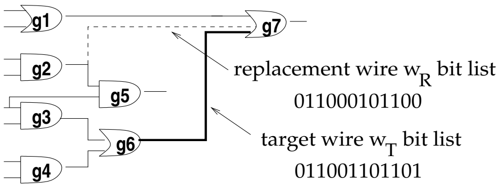

Figure 1: Delay Optimization

In the presentation that follows, we assume a preprocessing step where we simulate a number of random vectors 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 $g_6 \rightarrow g_7$ when we simulate 12 input vectors. The logic value of $g_6 \rightarrow g_7$ is 0 for the 1st vector, 1 for the 2nd vector, 1 for the 3rd 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 2,000-4,000 random input vectors and we update them during every iteration of the algorithm with parallel logic simulation at the fanout 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.

3.1 Delay Optimization

The determining factor for the delay of a circuit is its critical path(s). In order to shorten the critical path, the number of levels of logic [1] needs to be reduced. Therefore, during every iteration of the algorithm, w_T is a wire on some critical path of the circuit. Throughout the execution of the algorithm we also use the heuristics presented in the following paragraphs.

Given a set of lines on a critical path, we first target the input wires of a high fan-in gate(s). The intuition is that the more inputs a gate has, the less reliant this

[1] We assume that the logic level of the primary inputs is 0 and the level of a gate G is the maximum level of its fan-ins plus 1.

gate is on any particular input and it will be easier to find equivalent corrections elsewhere in the circuit. To reduce the levels of logic, we also attempt to replace w_T with a replacement wire w_R that has a a *lower* logic level and then correct the design in some less critical region. For example, consider the circuit in Fig. 1. If we replace wire $w_T = g_6 \rightarrow g_7$, shown in boldface, which is on the critical path with wire $w_R = g_2 \rightarrow g_7$ (dotted line) and we are also able to fix this error, the critical path is reduced from three levels of logic to two levels and so does the delay.

In addition, w_R should have similar logic values to w_T. The 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 candidates that have very different bit-lists and then we introduce an inverter.

3.2 Design for Low Power

One characteristic of the power consumption of a circuit is the transition density [6] of the gates it contains. The transition density of a gate is a measure of the switching activity of the gate [6]. Reducing the transition density of the circuit gates results in a lower power consumption. Therefore, we give priority to target logic w_T that fan-outs gates with high values of transition density. In addition, power gain may result because of the removal of the dangling circuitry dominated by the gate driving w_T once removed [7] [12]. We say that gate G dominates gate G' if and only if every path from G' to any primary output goes through G. Next, we outline the heuristics that guide our power minimization algorithm.

Again, as in delay optimization, the replacement wire w_R should have similar logic values to w_T. In addition, we give priority to a w_R with a high probability of zeros (ones) if it will be connected to 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 as well. 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 attempt 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. We give preference to the latter approach as it is more flexible and 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 equivalent correction for w_T then 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 very similar to the current one. For this reason, any high switching activity wire w_T that we fail to eliminate, we place it in an ignore list. When all the target wires in a circuit are in the ignore list, the ignore list is scratched and all target wires become free again to operate on.

4 Experiments

We ran the methods described in Sections 2 and 3 using Perl and C on a Sun Ultra 5 workstation and we tested them on ISCAS'85 combinational circuits. The details of the ATPG and DEDC algorithms that we employed can be found in [2] and [11], respectively. Our results on delay optimization and power minimization are shown in Tables 1 and 2, respectively. In these tables, all run-times are in minutes, delay figures are in nanoseconds and power is measured in microwatts. During the second and fourth step of the algorithm (Section 2) we set a maximum limit (user defined) of backtracks the ATPG tool [2] can perform before it aborts the fault.

Table 1: Delay Optimization

ckt name	# of modif.	original delay (ns)	final delay (ns)	% decrease	time (min)
C432	2	13.22	12.95	2.04%	6.2
C499	0	11.06	11.06	0%	7.9
C880	1	11.78	11.13	5.5%	9.1
C1355	0	12.42	12.42	0%	10.2
C1908	7	14.69	13.96	4.97%	14.6
C2670	2	10.82	10.69	1.20%	16.9
C3540	4	22.85	21.64	5.30%	21.1
C5315	5	18.38	17.66	3.92%	24.8
C7522	1	21.64	20.89	3.47%	31.5

For delay optimization, we first optimize the circuits with the use of *Design Compiler*$^{(TM)}$ by Synopsys and then we use our method. We use a high mapping effort and a virtual clock with a small clock period to enforce Synopsys to try its best in terms of delay optimization. The results of our delay optimization efforts can be found in Table 1. Column 1 shows the circuit name and column 2 has the number of transformations the circuit undergoes to reach the final optimized version. Columns 3 and 4 contain the original delay (after optimization by *Design Compiler*$^{(TM)}$) and final delay (after applying our method), respectively. The percentage of delay improvement can be found in column 5. C499 and C1355 are two "difficult" circuits that we fail to improve over Synopsys. For most of the other circuits, our improvement ranges from about 2 to 5 percent. We expect that as we develop more heuristics, as discussed in the end of this section, the results will also improve. The last column of Table 1 contains the overall run-time.

Table 2: Results on Power Minimization

ckt name	# of modif.	original power (uW)	final power (uW)	% decr.	area decr.	delay incr.	time (min)
C432	10	0.02529	0.01975	21.88%	12.9%	5.9%	7.6
C499	7	0.08947	0.08003	11.6%	3.9%	6.1%	12.4
C880	2	0.05590	0.05201	7.0%	2.9%	7.8%	11.6
C1355	12	0.08480	0.07711	9.1%	1.4%	9.4%	23.7
C1908	17	0.08551	0.07280	14.9%	1.6%	13.8%	30.2
C2670	13	0.14691	0.13001	11.6%	4.3%	0.6%	29.1
C3540	6	0.23360	0.21020	10.1%	1.6%	6.9%	33.4
C5315	7	0.38201	0.34902	9.2%	2.1%	8.4 %	29.5
C7522	4	0.70403	0.60110	14.7%	2.6%	5.3%	34.3

During power minimization, we initially use SIS [8] to optimize the circuits for area and MED [6] to obtain the transition density of each gate. The results are shown in Table 2. Column 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. 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 but the delay increases as we do not take into account any delay consideration during power optimization. To include delay consideration, we can modify our algorithm to disregard logic transformations that reduce the power but increase the delay, as described in [7] [12]. The final column of Table 2 contains the CPU run-time.

As with the case of delay optimization, we expect that as more heuristics are developed the results will improve significantly. In more detail, it is among our future research plans to introduce a larger family of permissible transformations during correction since for more than 50% of the suspicious lines returned during diagnosis for power minimization, none of the three permissible types of corrections we allowed was able to rectify the design (*i.e.* correction failed). We expect that a more complex design error model will increase the number of equivalent corrections and thus will improve the optimization gains.

5 Conclusions

We presented a novel ATPG-based design optimization methodology. Our methodology targets a wire and attempts to find alternate connections with the use of existing DEDC procedures. We also apply this technique to delay optimization and to design for low power.

In the future, we intend to develop and apply a family of more complex corrections and we plan to evaluate its performance on circuits where the gates have been decomposed, as described in [4].

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] S. C. Chang, J. C. Chuang, and Z. Z. Wu, "Synthesis for Multiple Input Wires Replacement of a Gate for Wiring Consideration," in *Proc. of ICCAD, pp. 115–118,* 1999.

[5] M. Chatterjee, D. Pradham, and W. Kunz, "LOT: Logic Optimization with Testability – New Transformations Using Recursive Learning," in *Proc. of ICCAD, pp. 115–118,* 1995.

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

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

[8] 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.

[9] A. Veneris, M. S. Abadir and I. Ting, "Design Rewiring based on Diagnosis Techniques," *IEEE ASP Design Automation Conference 2001,* (accepted).

[10] A. Veneris, M. S. Abadir and I. N. Hajj, "Design Optimization based on Diagnosis Techniques," *1st IEEE Latin-American Test Workshop 2000.*

[11] 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.

[12] 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.