

## Short Paper

---

# Logic Rewiring for Delay and Power Minimization\*

ANDREAS VENERIS

*Department of Electrical and Computer Engineering  
and Department of Computer Science  
University of Toronto  
Toronto, Ontario M5S 3G4, Canada  
E-mail: veneris@eecg.toronto.edu*

An application of the ATPG-based method by Veneris *et al.* [11] to multi-level combinational logic circuit delay and power optimization is presented. A number of theoretical results and various heuristics are described to allow for an efficient implementation of the algorithm. Experiments confirm the robustness of the approach.

**Keywords:** VLSI, logic optimization, low power, design performance, diagnosis

## 1. INTRODUCTION

Recent considerations in semiconductor portability and increasing clock cycle rates require designs that consume little power but meet strict performance requirements. Due to these facts, it becomes apparent that circuit optimization remains an important task in the overall design cycle. Traditionally, logic optimization is carried in two steps. In the first step, technology independent optimization is performed to produce an optimum design in terms of some general criteria such as gatecount or literal count. Symbolic-based techniques [8] have been very successful for this step. In the second step, technology dependent optimization is carried through an iterative sequence of successive *design rewiring* operations [1, 2, 4, 6, 7, 9, 10]. During each iteration of this procedure, a single *target wire* is identified for removal because it violates some specification constraints and some logic is added to eliminate the target wire. This process is repeated until the required optimization goals are achieved.

In this work, we describe an application of the method by Veneris *et al.* [10] to multi-level combinational circuit technology dependent optimization. Unlike most rewiring techniques [1, 2, 4, 6, 7, 9] that eliminate a target wire by adding redundant logic, the method in [10] treats rewiring using a sequence of design error diagnosis and correction [11] steps. Under this perspective, the task of design rewiring is performed by intro-

---

Received November 1, 2002; revised September 16, 2003; accepted November 10, 2003.

Communicated by Liang-Gee Chen.

\* Parts of this work are also presented in: A. Veneris, M. Amiri, and I. Ting, "Design Rewiring for Power Minimization," in IEEE International Symposium on Circuits and Systems, 2002.

ducing simple design errors that eliminate the target logic and correcting those errors in a less crucial part of the design. An obvious advantage of this approach lies in its flexibility to handle a variety of logic transformations that other methods cannot handle in favor of design optimization. Furthermore, we tailor the approach in [10] to delay optimization and design for low-power for digital logic designs mapped to a technology library. We also examine different trade-offs, present a number of theorems and illustrate heuristics that allow for an efficient implementation. Experiments on benchmark circuits demonstrate its effectiveness.

The remaining paper is organized as follows. The next section describes the delay optimization and design for low-power algorithms. In this presentation, we assume the reader is familiar with the work in [10, 11]. Experimental results are found in section 3 and section 4 concludes this work.

## 2. DESIGN OPTIMIZATION

The general flow of the optimization algorithm is the same for both optimization procedures. First, we identify the target logic  $w_T$  and introduce a design error by eliminating  $w_T$ . Next, the design rewiring method from [10] returns some equivalent valid correction(s). If no valid correction(s) is found we select a new objective, otherwise, for each equivalent correction, we compute the optimization gain, delete any new generated redundancies and keep the logic transformation with the highest improvement, if any. Since the algorithm operates on a (structural) gate level representation of the design, technology libraries are available to compute optimization trade-offs and optimization gains.

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. It should be noted, type (i) transformations have been performed by other logic rewiring methods [1, 2, 4, 7, 9] but type (ii) modifications are novel to this method. The particular implementations for delay and power optimization differ in the criteria and heuristics employed during the selection of the target wire  $w_T$  and the replacement wire  $w_R$ , as described later in this section.

In the following presentation, we assume that a pre-processing step is performed by the algorithm where it simulates 2,000-5,000 random vectors on the original circuit and keeps an indexed bit-list on every line  $l$  of the circuit. This bit-list holds the results of the simulation for each vector at the line. We say that two lines have *similar* logic values if most of their respective bit-list entries are the same. For example, Fig. 1 (a) shows the bit-lists of two lines,  $g_2 \rightarrow g_7$  and  $g_6 \rightarrow g_7$  when 12 input vectors are simulated. These two lines have similar logic values since their bit-lists differ in only two positions. In implementation, the degree of line similarity is a *user-specified* parameter usually defined so that 70-80% of the bit positions are identical.

Subsections 2.1 and 2.2 that follow contain heuristics to select  $w_T$  and  $w_R$  in delay and power minimization, respectively. Subsection 2.3 contains theory that allows the correction process to be performed efficiently.

### 2.1 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 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 gate is on any particular input and it will be easier to find equivalent corrections. To reduce the levels of logic, we also attempt to replace  $w_T$  with a replacement wire  $w_R$  that has a *lower* logic level and then correct the design in some less critical region. For example, consider the circuit in Fig. 1 (a). 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.

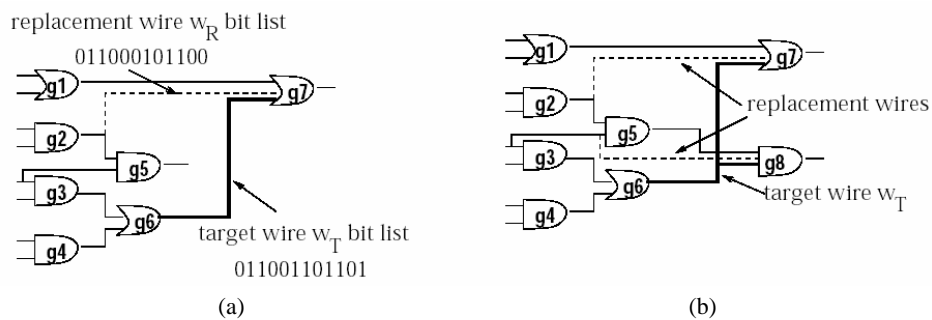


Fig. 1. (a) Delay optimization; (b) Power optimization.

Additionally,  $w_R$  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 rewiring 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.

### 2.2 Power Optimization

One characteristic of the power consumption of a circuit is the transition density [5] of the gates it contains. 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 a high transition density value. In addition, power gain may result because of the removal of the dangling circuitry that is dominated by the gate that drives  $w_T$  [7]. For example, if we eliminate (the fan-out of) gate  $g_6$  in Fig. 1 (b) we can also eliminate gates  $g_3$  and  $g_4$ .

Similar to delay optimization, the first heuristic suggests that  $w_R$  has similar logic values to  $w_T$ . Among all similar wires, 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. 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, as shown in Fig. 1 (b) (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.

### 2.3 Correction

During the *correction* stage, the algorithm attempts four types of logic transformations: (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.

The correction algorithm that we use extends the results presented in [11]. To implement this algorithm, following error introduction we maintain two bit-lists at each line  $l$  in the circuit, the  $l_{corr}$  and  $l_{err}$  list. These bit-lists are generated by “splitting” the original bit-list on  $l$ , described at the beginning of this section. The  $i$ -th bit of  $l_{corr}$  ( $l_{err}$ ) contains the logic value of  $l$  during simulation of the  $i$ -th vector with correct (failing) primary output responses that also produces a sensitized path from  $l$  to some primary output. These lists are re-calculated via bitwise parallel logic/fault simulation at each iteration of the optimization algorithm for different target logic.

To perform type (i) and (ii) corrections, for every candidate error line  $l$  that qualified diagnosis, the algorithm applies the logic transformation, one at a time, 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 [11].

The procedure above can be also applied to type (iii) and (iv) logic transformations [11]. Considering that the number of candidate new fan-ins may be large, we use the following theorem to check whether 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 definition of  $l_{err}$  it 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 indicates that some existing fan-in(s) of  $G$  already has the

controlling value 0 when this vector is simulated and no single additional fan-in to  $G$  can 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 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  $\bar{c}$  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 consider. Finally, we treat type (iv) logic transformations as a two step process. In the first step, we remove the suspicious fan-in  $l'$  from  $G$  and update lists  $l_{err}$  and  $l_{corr}$  at the output of  $G$ . Next, we treat it as a type (iii) logic transformation.

### 3. EXPERIMENTS

We implemented the method and ran it on a Sun Ultra 10 workstation on ISCAS '85 combinational circuits. Results on delay and power minimization are shown in Tables 1 and 2, respectively, where run-times are in minutes, delay figures are in nanoseconds and power is measured in microwatts. The details of the ATPG and DEDC algorithms we use are found in [3] and [11]. We use SIS [8] to detect/delete newly generated redundancies between successive optimization steps.

For delay optimization, we first optimize the circuits using *Design Compiler*<sup>(TM)</sup> by Synopsys and then run the proposed 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 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*<sup>(TM)</sup>) and final delay (after applying our method), respectively. The percentage of delay improvement can be found in column 5. It is seen, in most circuits, delay improvement ranges from about 2 to 5 percent over Synopsys. The last column contains the run-time.

In power minimization, we initially optimize the circuits for area with SIS [8] (*scripted.rugged*) and we use MED [5] to obtain the transition density of each gate. Column 1 of Table 2 contains the circuit name and column 2 has the number of transformations performed on each circuit. Columns 3, 4 and 5 contain the original power, the

**Table 1. Results on delay minimization.**

ckt name	# of modif.	original delay (ns)	final delay (ns)	% decrease	time (min)
C432	3	13.22	12.95	2.04%	3.2
C499	0	11.06	11.06	0%	4.5
C880	5	11.78	9.13	22.6%	7.3
C1355	0	12.42	12.42	0%	5.3
C1908	7	14.69	13.96	4.97%	6.9
C2670	4	10.82	10.69	1.20%	8.1
C3540	4	22.85	21.64	5.30%	12.8
C5315	5	18.38	17.66	3.92%	10.0
C7522	3	21.64	20.89	3.47%	14.3

**Table 2. Results on power minimization.**

ckt name	# of corr.	original pwr (uW)	final pwr (uW)	pwr decr.	area decr.	hit ratio	time (min)
C432	21	0.02529	0.01979	21.01%	11.9%	34%	7.6
C499	37	0.08947	0.07574	15.0%	5.9%	19.5%	12.4
C880	39	0.05595	0.04708	15.6%	4.1%	17.2%	11.0
C1355	55	0.08479	0.07908	7.3%	3.0%	43.7%	14.1
C1908	68	0.08565	0.06316	26.1%	9.7%	29.9%	16.2
C2670	96	0.14712	0.11183	24.0%	18.6%	56%	19.7
C3540	6	0.23360	0.19618	15.9%	7.8%	41.7%	20.1
C5315	33	0.39137	0.33473	14.1%	5.3%	36%	23.9
C7522	7	0.70403	0.58719	16.3%	4.9%	30.6%	29.4

final power and the percentage of improvement, respectively. It is seen, power is reduced by 15.0% on the average. The next column contains the change incurred in the area due to the power optimization transformations. As expected, the area of the circuit is positively affected and decreased by 4.9% on the average. The delay of the designs does not deteriorate, as we skip transformations that reduce the power but increase the delay. The success ratio, that is, the number of times the approach found an equivalent correction(s) during optimization is contained in column 7. It should be noted that although the success ratio for circuit *C499* is low this is also the case with existing design rewiring techniques [1]. The final column contains the CPU time.

To demonstrate the overall effectiveness of the proposed process, we apply a two-stage optimization procedure to three circuits originally optimized for area with SIS. In the first stage, we use our methodology to perform power optimization with no delay consideration. During the second stage, the circuit obtained is further optimized for delay. To measure the overall performance, we compute the power/delay product of the final circuit obtained by multiplying the power consumption and delay. Naturally, the goal is

to reduce this product. The result of this two-stage optimization process is plotted in Fig. 2 where the power delay product is normalized by its original value. It can be seen that the algorithm improves performance noticeably.

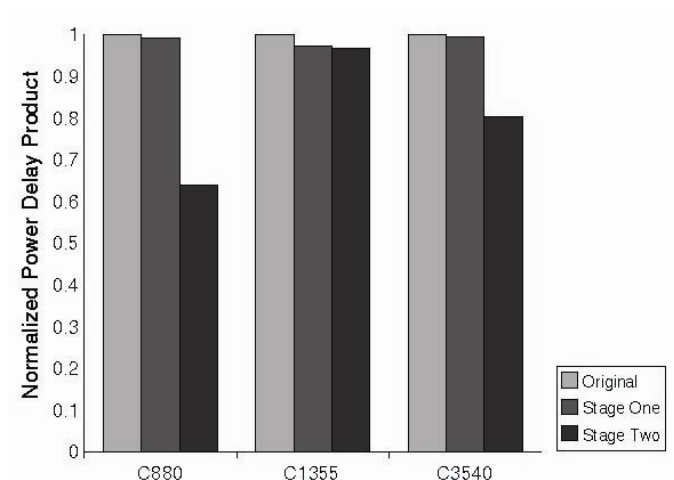


Fig. 2. Delay/Power Product Improvement.

#### 4. CONCLUSIONS

We presented an application of the method in [10] to delay and power minimization. We also described the theory and heuristics that allow for an efficient implementation. Experiments show that the optimization saving is significant in most benchmark circuits.

#### ACKNOWLEDGMENTS

The author thanks Dr. Magdy S. Abadir, Mandana Amiri, Ivor Ting and Brandon J. Liu for technical contributions and insights during early stages of this work.

#### REFERENCES

1. C. W. Chang and M. Marek-Sadowska, "Single-pass redundancy addition and removal," in *Proceedings of IEEE International Conference on Computer Aided Design*, 2001, pp. 606-609.
2. Y. N. Jiang, A. Krstic, K. T. Cheng, and M. Marek-Sadowska, "Post-layout logic restructuring for performance optimization," in *Proceedings of IEEE Design Automation Conference*, 1977, pp. 662-665.
3. W. Kunz and D. K. Pradhan, "Recursive learning: a new implication technique for efficient solutions to CAD problems – test, verification, and optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13,

- 1994, pp. 1143-1158.
4. W. Kunz, D. Stoffel, and P. R. Menon, "Logic optimization and equivalence checking by implication analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, 1997, pp. 266-281.
  5. F. Najm and M. Xakellis, "Statistical estimation of switching activity in digital circuits," in *Proceedings of IEEE Design Automation Conference*, 1994, pp. 728-733.
  6. D. K. Pradhan, M. Chatterjee, M. V. Swarna, and W. Kunz, "Gate-level synthesis for low-power using new transformations," in *Proceedings of IEEE International Symposium on Low Power Electronic and Design*, 1996, pp. 297-300.
  7. B. Rohfleisch, A. Kolbl, and B. Wurth, "Reducing power dissipation after technology mapping by structural transformations," in *Proceedings of IEEE Design Automation Conference*, 1996, pp. 789-794.
  8. E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proceedings of IEEE International Conference on Computer Design*, 1992, pp. 328-333.
  9. G. Stenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes, "Performance optimization by interacting netlist transformations and placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, 2000, pp. 350-358.
  10. A. Veneris and M. S. Abadir, "Design rewiring using ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 21, 2002, pp. 1469-1479.
  11. A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, 1999, pp. 1803-1816.

**Andreas Veneris** was born in Athens, Greece. He received the Diploma in Computer Engineering and Informatics from the University of Patras in 1991, the M.S. degree in Computer Science from the University of Southern California, Los Angeles in 1992 and the Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1998.

He is currently an Associate Professor at the University of Toronto, Department of Electrical and Computer Engineering cross-appointed with the Department of Computer Science. His research interests include algorithms and CAD for synthesis, diagnosis, and verification of digital circuits. He is coauthor to one book and co-recipient of a best paper award in ASP-DAC'01. Prof. Veneris is member of the IEEE, ACM, AAAS, Technical Chamber of Greece and the Planetary Society.