Boolean Techniques for Low Power Driven Re-Synthesis *

R. Iris Bahar Fabio Somenzi
Dept. of Electrical and Computer Engineering
University of Colorado, Boulder

Abstract

We present a boolean technique to reduce power consumption of combinational circuits that have already been optimized for area and delay and then mapped onto a library of gates. In order to achieve a better optimization, we cluster *gates by collapsing two or more levels of gates into a single node. When optimizing each cluster, our method extends the algorithms used in* ESPRESSO*, by adding heuristics that bias the minimization toward lowering the power dissipation in the circuit. The results of our method, on a number of benchmark circuits, show an average of 16% improvement in power savings compared to existing boolean techniques.*

1 Introduction

Designing electronic devices with lower power dissipation is increasingly critical in today's market. As the density, size, and complexity of VLSI chips continue to increase, providing adequate cooling, long running batteries for portable devices, and inexpensive packaging may no longer be possible. Techniques for IC design should address the design problem as a three-dimensional one, optimizing for area, performance, and power simultaneously.

Under a simplified model of the energy dissipated by CMOS devices, the power consumption of a CMOS gate is directly related to its switching activity factor [1]. We assume that the energy dissipated by a CMOS gate each time its output switches equals the change in the energy stored in the capacitor associated with the output of the gate. The average power, P_{avg}, dissipated by the gate is given by:

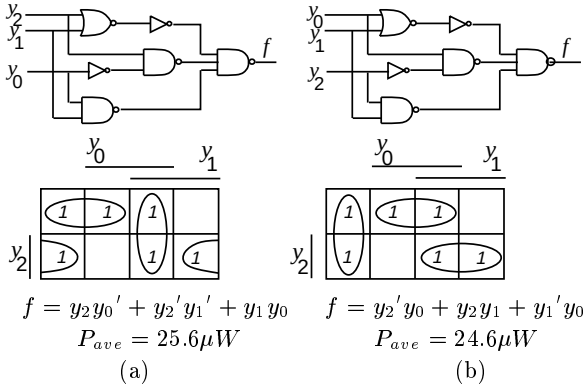
$$P_{avg} = \frac{1}{2} \cdot C_{load} \cdot \frac{V_{dd}^2}{T_C} \cdot E(transitions), \qquad (1)$$

where C_{load} is the capacitance of the output load, V_{dd} is the supply voltage, T_C is the global clock period, and *E(transitions)* is the expected number of gate output transitions per clock cycle. *E(transitions)* represents the *switching activity* of the output of the gate.

During the high-level design stage, power dissipation in ICs can be reduced by means of precise architectural choices [2]. At the logic level, it is the switching activity of the logic, weighted by the capacitive loading, that is the prime focus of the optimization. Several methods have been proposed to reduce power dissipation for combinational circuits at the logic level. Some techniques are applied at the technology independent stage of logic synthesis; they try to reduce the switching activity of the logic functions using an approach similar to that of [3]. Both [4] and [5] use the don't care set to minimize a linear combination of the number of product terms and switching activity of a function. Another method applied at the technology independent stage uses a kernel extraction procedure to generate multi-level circuits with lower power consumption [6]. In [7], technology decomposition and mapping methods are introduced that try to minimize the total switching activity using a procedure similar to Huffman's algorithm and dynamic programming. Technology mapping using an area-delay-power tradeoff curve has been used in methods described in [8] and [9]. In addition to technology independent and technology mapping techniques, at the gate re-sizing stage, symbolic methods used in [10] may be applied to more accurately measure gate slack and size gates down to save power.

The main problem with applying low power optimization to technology independent circuits is that power dissipation is difficult to measure with a dependable level of accuracy so that the results may not be very predictable. We propose a method that can be applied to technology mapped circuits, but uses technology independent boolean techniques for optimizing the circuit. If we start with a circuit that is already implemented in gates from a technology library, we can extract from it more accurate information on the switching activity of the gates, and thereby more effectively apply low power driven boolean optimization techniques to re-synthesize the circuit.

As an example of the optimization problem we are addressing, consider the function shown in Figure 1, where we have two possible implementations shown in the Karnaugh maps of Figures 1(a) and 1(b). Both implementations are optimal in terms of number of product terms and number of literals, however, given *non-uniform* switching activity at the primary inputs to the circuit, selection of the cover in 1(b) reduces power dissipation in the circuit.



$f = y_2 y_0' + y_2' y_1' + y_1 y_0$ $\quad$ $f = y_2' y_0 + y_2 y_1 + y_1' y_0'$
$P_{ave} = 25.6\mu W$ $\quad$ $P_{ave} = 24.6\mu W$
(a) $\quad$ (b)

Figure 1: Different Cube Coverings for Node f.

Given a technology mapped circuit (i.e., a netlist of gates) our approach to re-synthesis is based on extracting clusters of gates; deriving a two-level cover for each cluster; optimizing the cover with low power as the objective using both observability and satisfiability don't cares; and finally remapping the cluster onto the given library. While dealing with one cluster at the time, we can rely on the rest of the circuit to produce meaningful statistical information on the switching activity, because the information is obtained from a technology mapped circuit.

Our approach to solving the minimization problem is an extension of the ESPRESSO algorithm [11, 12]. We have

*This work is supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 94-DJ-560.

developed our procedures within the SIS [13] environment and compare our results to existing minimization procedures, oriented toward area minimization. We show that our method reduces power, often at no cost in terms of increased area or delay.

To re-synthesize our circuit for lower power we need an accurate measure of switching activity (including glitching) of the output of each gate in the circuit. To do this we use existing tools that measure average power; specifically, we use the tool described in [14], which assumes the transition probabilities of the primary inputs to be given, and calculates the switching activity factor of each gate output using symbolic simulation.

2 Preliminaries

It is customary to classify logic optimization techniques into *technology-independent* methods (e.g.,extraction of common subexpressions) and *technology dependent* methods (e.g. buffer sizing), according to whether they manipulate logic formulae without reference to a specific technology or library, or they consider those technology-related factors. Effective optimization strategies combine technology-independent techniques with technology-dependent ones. The boundaries between these categories are not sharply defined, and that techniques that straddle those boundaries often produce very good results.

A boolean optimization technique for multi-level circuits that is of particular interest to us consists of applying two-level minimization to the nodes of the multi-level circuit. Following the nomenclature of SIS [13], we shall call this technique *simplify*. For simplify to be effective, the functions of the nodes subjected to minimization (in terms of their local inputs) should not be too small. It is therefore customary to precede its application by a partial collapsing of the circuit, which produces a circuit with fewer, more complex, nodes.

A global optimization procedures uses information about its global behavior during the optimization process. *Simplify* can be a global optimization procedure, depending on whether *don't cares* are used in solving the two-level minimization problems. The don't cares express conditions that may not occur at the inputs of the node being simplified (*satisfiability don't cares*,) or conditions under which the primary outputs of the circuit are not sensitized to the value of the node being simplified (*observability don't cares*.)

The extraction of don't cares from a multi-level network can be done by first computing compatible subsets of observability don't cares in terms of the primary inputs; by then adding the external don't cares; and by finally computing the image of the don't care conditions at the local inputs of the node being simplified. To avoid wasteful recomputation when observability don't cares are used, it is convenient to apply *simplify* to the nodes in topological order from the primary outputs to the primary inputs [3].

A *cover* for a function F is a set of *cubes* or *implicants*, such that the disjunction of the cubes is a sum of product representation of the cover. An incompletely specified function is represented to the two-level logic minimizer as a pair of covers, (F, D). F is a cover of the ON-set of the function, and D is a cover of the don't care set. The OFF-set R can be obtained as the complement of $F \cup D$.

The two-level minimization problem that is obtained by appending the don't care conditions to the function of a node is typically solved by a heuristic algorithm similar to ESPRESSO. The basic loop of ESPRESSO consists of three steps: In the first, the cubes of the function being minimized are reduced as much as possible; in the second step they are expanded so that as many cubes as possible are covered by other cubes and can therefore be dropped. In the third and final step, an irredundant set of cubes that cover the function is extracted from the set of those that have survived the expansion phase. If close-to-optimum results are desired, the basic loop is repeated until no improvement is achieved. In *simplify* the basic loop is repeated only once for the sake of speed.

3 Clustering

As pointed out in Section 2, *simplify* yields better results if the functions of the nodes are not too small. Indeed, if the node being simplified corresponds to a simple gate, and no don't cares are used, no simplification can be achieved. If don't cares are used, a simple gate can be simplified only if some of its connections are redundant. When applied to a complex function, on the other hand, *simplify* can escape a local minimum and produce a better cover, even when starting from an irredundant circuit.

In a technology-mapped circuit, nodes tend to be simple. Therefore, the application of techniques similar to *simplify* to circuits that are already expressed in terms of library gates requires that nodes be not considered in isolation, but in *clusters*. We call *clustering* the process of forming groups of nodes to be subjected to two-level minimization. Although similar to the partial collapsing described in Section 2, clustering differs from it because it does not permanently change the structure of the entire network (undesirable for a technology-mapped circuit); rather, it finds groups of nodes from a sub-network forming fanout-free subcircuits that can be extracted, simplified, and remapped onto the gate library, while leaving the rest of the circuit unchanged.

Clustering collapses gates from two or more levels of logic into a single node. The goal is to then find a locally optimal cover for each cluster. By working with more than one level of logic we have more flexibility in choosing a cover that is good in terms of both area and power. On the other hand, we do not want to collapse gates from too many levels of logic into a single node since we may be too removed from a mapped representation of the circuit. The clustering starts at the primary outputs and proceeds to the primary inputs. To keep the algorithm simple, we only collapse the fanin of a node if this fanin does not fanout to multiple sources. The output node of a cluster may have multiple fanouts.

4 Two-Level Optimization

We solve the low power optimization problem by solving a series of two-level minimization problems. For each problem we are given a cluster for a node f described as a two-level cover in terms of the local inputs to the cluster. After optimization we obtain a new cover for the function f, which should produce less switching activity once the function is

mapped onto gates from the library. We accept the new cover of f if the number of literals (or alternatively, number of cubes) has not increased from the original cover. After all clusters have been optimized we remap the entire circuit.

4.1 Cube Probabilities

In order to bias logic minimization toward lowering power dissipation, we need the switching activity, or *transition density*, of each cube in the cover of f. Before clustering our circuit, we have computed the transition densities for all signals in the circuit using an accurate delay model (i.e., we compute transitions due to functional output switching as well as glitches). Therefore, we have available the transition densities for all input signals to each cluster of the circuit. The problem now becomes how to compute the transition densities of each cube c in the cover of function f given the transition densities and signal probabilities for all inputs to the cluster. The signal probability is defined as the probability of a signal being at value 1. The signal probabilities are found by building the functional BDDs for each x_i with respect to the primary inputs of the circuit.

We approximate the cube transition densities by assuming that transitions on different inputs are not simultaneous and that the inputs to the cluster are independent. Under these assumptions, given a cluster with n inputs x_i, $1 \leq i \leq n$ and transition densities $D(x_i)$, we compute the transition densities of all m cubes c_j, $1 \leq j \leq m$, in the cover of f using the following formula due to [1]:

$$D(c_j) = \sum_{i=1}^{n} P\left(\frac{\partial c_j}{\partial x_i}\right) \cdot D(x_i), \qquad (2)$$

where $P\left(\frac{\partial c_j}{\partial x_i}\right)$ is the signal probability of the *Boolean difference* of c_j with respect to input x_i.

Consider the cover shown if Figure 2(a). Suppose the signal probabilities of the inputs, $P(y_i)$, are $P(y_0) = 0.9, P(y_1) = 0.3, P(y_2) = 0.5$, and $P(y_3) = 0.8$, and assume, for the sake of simplicity, that the transition densities of the inputs are the same as their signal probabilities in this example. To compute the transition density of cube $c_j = y_3{}' y_2 y_1{}'$, we use Equation 2:

$$\begin{aligned} D(c_j) &= P(y_3{}')P(y_2)\cdot D(y_1) + P(y_3{}')P(y_1{}')\cdot D(y_2) + \\ &\quad P(y_2)P(y_1{}')\cdot D(y_3) \\ &= 0.2\cdot 0.5\cdot 0.3 + 0.2\cdot 0.7\cdot 0.5 + 0.5\cdot 0.7\cdot 0.8 = 0.38 \end{aligned}$$

Transition densities for the other cubes may be computed in a similar fashion producing the values $D(y_3 y_1 y_0) = 0.648$, $D(y_3{}' y_1{}' y_0{}') = 0.242$, $D(y_3 y_2{}' y_0) = 1.08$, $D(y_3{}' y_2 y_0) = 0.54$, and $D(y_2 y_1 y_0) = 0.405$. Notice that due to the non-uniformity of the input transition densities, the transition densities of the cubes in the cover may vary widely.

When processing a cluster, a large amount of simplification is possible if the don't cares are used properly. In order to accurately build the don't care set for a given node in the circuit, we must assume that the nodes in its transitive fanout are not going to be altered. Given this assumption, once a node has been simplified, we can only simplify those gates in its transitive fanin, unless we want to re-evaluate the entire circuit again. For efficiency reasons, we want a simplification process that can be carried out in one pass over the circuit. This implies processing the nodes from primary outputs to primary inputs. The drawback of this approach is that we are making assumptions on the values of the switching activity at the local inputs of the cluster in order to do the simplification. If we change the functional implementation of any transitive fanin to this cluster, we may be changing these switching activity values and the assumptions we made while simplifying this cluster may no longer hold. However, the input switching activity is used mainly as a guide to optimizing the cover, so small changes in switching activity do not usually change the overall quality of the new optimized cluster.

4.2 Reduction

Given a cover pair (F, D) and a cube $c \in F$, reduction is the process of replacing c in F by the smallest $r \leq c$ such that $((F - \{c\}) \cup \{r\}, D)$. Reduction of a cube is possible when it overlaps other cubes in the two covers. For a fixed pair (F, D) the maximal reduction of each cube is unique: If the cube is reduced by adding a set of literals to it, the order in which the literals are added is immaterial. On the other hand, the order in which the cubes of F are considered for reduction does matter. Typically, the cubes that are considered first can be reduced more, because, as the process advances, the overlaps of the cubes decrease.

The purpose of reducing all cubes during minimization is to make it possible to subsequently expand some of them in different directions, so that some other cubes will be entirely covered, and therefore dropped. Since decreasing the number of cubes is advantageous for area, and most of the time also for power, cubes are maximally reduced, regardless of the temporary negative effects this may have on the transition probabilities. To steer the subsequent expansion in the direction of decreasing power consumption, the cubes are processed in order of decreasing switching probability. Hence, the cubes that cause the most activity are given a higher chance of dropping out of the cover.

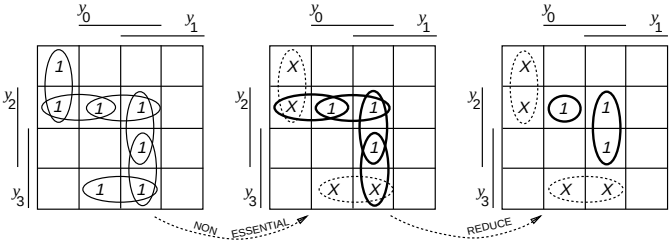


Figure 2: Reduction Process for Node f.

Returning to our example in Figure 2, starting with the initial cover shown in Figure 2(a), we want to maximally reduce the cubes in the cover. To avoid needless work, we remove from our cover any cubes we know are *essential*. These essential cubes must be included in any prime cover of the function; hence there is no point trying to reduce and later expand these cubes. In Figure 2(b) the essential cubes $y_3{}' y_1{}' y_0{}'$ and $y_3 y_2{}' y_0$ are temporarily removed from the cover and replaced with don't care values. We now compute the switching probability of the remaining cubes (as shown in Section 4.1) and obtain the order for cube

reduction as ($y_3y_1y_0$, $y_3'y_2y_0$, $y_2y_1y_0$, $y_3'y_2y_1'$) according to decreasing values of switching probabilities. The cubes are maximally reduced in this order to obtain the cover shown in Figure 2(c). Notice that the two cubes with the highest transition densities ($y_3y_1y_0$ and $y_3'y_2y_0$) have been completely eliminated from the cover.

4.3 Expansion

After reduction, the cubes of F are not, in general, primes. The purpose of *expansion* is to turn a cube $c \in F$ into one of the primes of $F \cup D$ that contain c. The non-uniqueness of the maximal expansion of a cube makes this phase of the minimization algorithm more interesting, and more complex. In expanding a cube, two objectives must be taken into account: One is the quality of the cube taken in isolation. For area, this means that the expansions that lead to the largest cubes are preferable. For power, the switching statistics of the inputs must be taken into account as well. The second objective of expansion is to cover as many other cubes as possible, either totally, or partially. In the case of power minimization, the value of an expansion depends not only on the number of cubes that are covered, but also on the activity of the cubes that are eliminated because they are covered.

The maximal expansions of a single cube do not depend on the order in which the cubes of F are processed. However, the order in which the cubes are processed for expansion is important because of the effect it has on which cubes are covered and hence dropped: Expanding a cube too early may prevent another cube from covering it. In order to minimize power consumption, the switching activity of each reduced cube is computed and we process them according to increasing switching activity. Thus, the cubes with high switching activity are kept last, in the hope that some other cubes will expand to cover them.

In the case when an expanded cube may subsume more than one other cube, we expand the cube in the direction that makes it cover the cubes with higher switching activity. If no other cubes can be entirely covered, ESPRESSO tries to maximize the overlap of the expanded cube and the remaining cubes, so that the number of cubes in the cover may be minimized. While minimizing the number of cubes in the cover of the cluster is still important, in order to bias our cover toward low power, we assign a weight to each variable according to its switching probability and solve the problem as a minimal weighted cover. We give lower weight, i.e., higher advantage, to *not* expanding the cube in the direction of a variable with lower transition density. Given input variable x_i to the cluster, we assign the weights using the following equation:

$$weight\,(x_i) = \alpha \cdot TransitionDensity\,(x_i) + (1 - \alpha), \quad (3)$$

where α is a parameter used to properly weigh the variables so that both transition density and the number of cubes needed to cover the function can have the appropriate importance.

To illustrate the expansion process, we start with a maximally reduced cover as shown in Figure 3(c). We first compute the transition densities of the cubes to obtain the expansion order ($y_3'y_2y_1y_0$, $y_3'y_2y_1'y_0$, $y_3y_2y_1y_0$) according

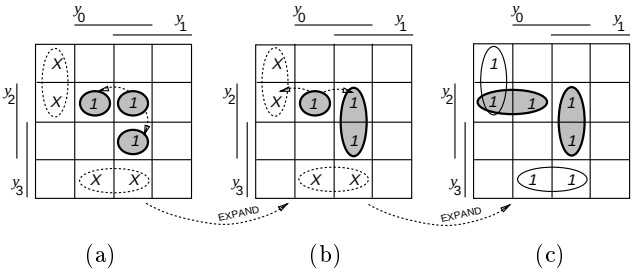


Figure 3: Expansion Process for Node f.

to increasing values of transition densities ($D(y_3'y_2y_1y_0) = 0.189$, $D(y_3'y_2y_1'y_0) = 0.405$, and $D(y_3y_2y_1y_0) = 0.432$.) We then consider the expansion of cube $y_3'y_2y_1y_0$. We can expand this cube in either the direction y_1 or y_3 as shown by the two arrows in Figure 3(a). In either direction we would subsume one cube; however, we conclude that expanding in the direction of y_3 is better since we will remove a cube with a higher transition density from the cover. After expansion of cube $y_3'y_2y_1y_0$ we obtain the cover as shown in Figure 3(b). We now consider the expansion of cube $y_3'y_2y_1'y_0$. Expansion in either direction y_0 or y_1 will not subsume any other cubes; however, if we expand in direction y_0 we remove from the cube a variable with higher transition density. The final cover after expansion is shown in Figure 3(c).

4.4 Irredundant Covers

Expansion produces a cover of prime implicants. None of them is entirely contained in another cube of the cover. However, some cubes may be covered by the union of other cubes. The purpose of the *irredundant* step is to extract a subset of the prime cover, that is still a cover. A covering problem is solved, in which the cubes in the current cover are selected so as to cover all the minterms of the function. The solution to the covering problem is found by a branch-and-bound algorithm. This reflects the non-uniqueness of the irredundant covers for a given initial cover. When minimizing for low power, the objective of *irredundant* is twofold: To reduce the number of cubes, and to reduce their switching activity. These two objectives can be combined by making the cost of each cube—as seen by the covering problem solver—a convex combination of the area and power dissipation of the cube. (See Equation 3.)

5 Experimental Results

The algorithms and heuristics described in this paper have been implemented and integrated within the SIS logic synthesis package. We conducted experiments on a subset of circuits from the MCNC benchmark set. Power estimation (and switching activity) was computed using the symbolic simulation method of [14]. All experiments were run on a DEC-station 5000/200 with 88MB of memory.

In Table 1 we report the results of our method when used to optimize circuits mapped for area and speed. Randomly generated non-uniform transition densities for the primary inputs are used for all experiments. We start with an initial circuit which has been first optimized using the SIS script `script.rugged`. After this initial optimization the circuit is mapped for speed onto a gate library consisting of NANDs, NORs, and inverters—each type having five different size

Circuit	*Init.Mapping*			*Full Simplify*			*Simplify for LP*				*Savings*	
											Area	*Power*
	Area	*Delay*	*Power*	*Area*	*Delay*	*Power*	*Area*	*Delay*	*Power*	*Depth*	*FS / LP*	*FS / LP*
5xp1	256244	15.88	1201	224576	15.65	902	224576	15.65	902	1	12% / 12%	25% / 25%
9sym	474904	10.94	2359	438944	11.19	2298	438944	11.19	2298	1	8% / 8%	3% / 3%
Z5xp1	216688	20.93	2177	240004	22.08	2594	216572	20.70	2001	3	-11% / 0%	-19% / 8%
b12	164952	8.12	538	141984	7.89	466	133980	8.17	405	3	14% / 19%	13% / 25%
bw	348696	17.58	1512	337444	18.09	1695	316796	17.57	1410	3	3% / 9%	-12% / 7%
clip	279676	12.29	1495	271904	12.74	1438	249632	12.24	1338	4	3% / 11%	4% / 11%
misex1	119828	9.92	772	111476	9.89	704	111476	9.89	704	1	7% / 7%	9% / 9%
rd53	68904	8.52	350	65888	8.25	320	65888	8.25	320	1	4% / 4%	9% / 9%
rd73	131892	15.10	780	112056	15.10	803	103356	14.55	559	4	15% / 22%	-3% / 28%
rd84	301368	13.10	1782	290812	13.38	1881	253924	13.10	1582	2	4% / 16%	-6% / 11%
sao2	312156	15.46	1284	282460	15.78	1223	248936	15.32	894	3	10% / 20%	5% / 30%
squar5	139896	13.75	716	122264	13.75	672	100920	13.05	497	3	13% / 28%	6% / 31%
Average											7% / 13%	3% / 16%

Table 1: Experimental Results on Sample Circuits.

options. We show the effectiveness of our results my comparing the area and power savings we obtain using our new procedure *simplify_for_lp* with the procedure *full_simplify* from the SIS package. Columns 2–10 report the area, delay, and power after mapping, for the initial circuit, after optimization using *full_simplify*, and after optimization using *simplify_for_lp*, respectively. The optimization procedures attempt to keep the delay of the circuits very close to the delay of the original mapping. We have experimented with different depths of clustering ranging from 1 to 4. In Column *Depth* we report the number of levels of clustering that leads to the best reduction in power. The results we report for our method reflect this best level of clustering. Columns 12 and 13 report the area and power savings compared with the initial circuit after optimization using *full_simplify* (*FS*) and *simplify_for_lp* (*LP*). As seen from our results, our simplification procedure often obtains a circuit with better power and area reduction compared to *full_simplify*, and in some cases the difference is quite significant. There is no specific level of clustering that leads to the best optimization; however, our procedure tends to favor less clustering rather than more, suggesting that clustering too many levels moves us too far away from a mapped representation of the circuit. Once the simplification is complete, additional power savings may be obtained by resizing the gates using the method described in [10].

6 Conclusions

We have presented a boolean technique to reduce power consumption of combinational circuits that have already been mapped. Starting from a mapped circuit allows us to obtain more accurate information about the power dissipation of the circuit and then use this information to more effectively re-synthesize the circuit to lower its power dissipation. Comparing our results to those of SIS has shown that our method produces circuits with lower power dissipation in most cases. Clustering alone can be a very effective tool; running *full_simplify* after clustering the circuit can produce results almost as good as those shown in Table 1 for *simplify_for_lp*. In order for the two-level optimization step to more effectively contribute to power savings this step needs to be tightly coupled with technology mapping—mapping should not be done as a post-processing step. Therefore, our future plans include extending our techniques to insure that optimizations made at the technology independent stage are not lost during technology mapping.

References

[1] F. N. Najm, "Transition density, a stochastic measure of activity in digital circuits," in *Proceedings of the Design Automation Conference*, (San Francisco, CA), pp. 644–649, June 1991.

[2] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Transactions on CAD*, vol. 14, pp. 12–31, Jan. 1995.

[3] H. Savoj, R. K. Brayton, and H. J. Touati, "Extracting local don't cares for network optimization," in *Proceedings of the International Conference on CAD*, (Santa Clara, CA), pp. 514–517, Nov. 1991.

[4] A. Shen, S. Devadas, and A. Ghosh, "Probabilistic construction and manipulation of free boolean diagrams," in *Proceedings of the International Conference on CAD*, (Santa Clara, CA), pp. 544–549, Nov. 1993.

[5] S. Iman and M. Pedram, "Multi-level network optimization for low power," in *Proceedings of the International Conference on CAD*, (San Jose, CA), pp. 372–377, Nov. 1994.

[6] S. C. Prasad and K. Roy, "Circuit activity driven multilevel logic optimization for low power reliable operation," in *Proceedings of the European Conference on Design Automation*, (Paris, France), pp. 368–372, Feb. 1993.

[7] C.-Y. Tsui, M. Pedram, and A. Despain, "Technology decomposition and mapping targeting low power dissipation," in *Proceedings of the Design Automation Conference*, (Dallas, TX), pp. 68–73, June 1993.

[8] V. Tiwari, P. Ashar, and S. Malik, "Technology mapping for low power," in *Proceedings of the Design Automation Conference*, (Dallas, TX), pp. 74–79, June 1993.

[9] B. Lin and H. deMan, "Low-power driven technology mapping under timing constraints," in *International Workshop on Logic Synthesis*, (Lake Tahoe, CA), May 1993.

[10] R. I. Bahar, G. D. Hachtel, E. Macii, and F. Somenzi, "A symbolic method to reduce power consumption of circuits containing false paths," in *Proceedings of the International Conference on CAD*, (San Jose, CA), pp. 368–371, Nov. 1994.

[11] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston, Massachusetts: Kluwer Academic Publishers, 1984.

[12] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-6, pp. 727–750, Sept. 1987.

[13] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proceedings of the International Conference on Computer Design*, (Cambridge, MA), pp. 328–333, Oct. 1992.

[14] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proceedings of the Design Automation Conference*, (Anaheim, CA), pp. 253–259, June 1992.