An Exact Gate Decomposition Algorithm for Low-Power Technology Mapping

Hai Zhou and D.F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

Abstract

With the remarkable growth of portable application and the increasing frequency and integration density, power is being given comparable weight to speed and area in IC designs. In technology mapping, how decomposition is done can have a significant impact on the power dissipation of the final implementation. In the literature, only heuristic algorithms are given for the low-power gate decomposition *problem. In this paper, we prove many properties an optimal decomposition tree must have. Based on these optimality properties, we design an efficient exact algorithm to solve the low-power gate decomposition problem. Moreover, the exact algorithm can be easily modified to a heuristic algorithm which performs much better than the known heuristics.*

1 Introduction

With the remarkable growth of portable application and the increasing frequency and integration density, power is being given comparable weight to speed and area in IC designs. Power dissipation in digital CMOS circuits is dominated by the *dynamic dissipation*, which is mainly the charging and discharging of the node capacitances [5]. It can be modeled as

$$P = 0.5V_{dd}^2 f_{clk} C_L E_{sw}$$

where V_{dd} is the supply voltage, f_{clk} is the clock frequency, C_L is the physical capacitance at the output of the node, and E_{sw} (referred to as the *switching activity*) is the average number of output transitions per clock cycle. As we can see, V_{dd} and f_{clk} are fixed by the technology, but C_L and E_{sw} can be controlled in design process.

In technology mapping, the subject netlist is usually first decomposed into a netlist composed of only inverters and two-input NAND gates. How the decomposition is done can have a significant impact on the power dissipation of the final implementation [4, 6, 7]. We deal with the *low-power gate decomposition* problem in this paper.

The problem appears in a few recent papers. Tiwari *et al.* [6] mentioned the importance of a good decomposition on the final result of technology mapping, but did not give any solution. At the same time, Tsui *et al.* [7] analyzed the problem and found that Huffman's algorithm [3] can only be used in domino dynamic logic. For static logic which is more important in low-power applications, only a greedy heuristic called the *modified Huffman algorithm* is given. Murgai *et al.* [4] also considered the decomposition problem, but their minimization objective was the power consumptions due to glitches.

Since the problem for dynamic logics can be easily solved, we only consider static logics. In our approach, we first study the structure of an optimal decomposition tree. This is given by a set of properties an optimal tree must have. Then, based on these properties, we designed an exact algorithm for the construction of an optimal decomposition tree. The time complexity of the algorithm is $O(n2^n)$, which, though still exponential, should be regarded as efficient considering the total of more than $(2n-1)^{n-1}$ trees in the solution space.

As a by-product, a heuristic algorithm can be easily derived from the exact algorithm. Its running time is $O(n \log n)$, which is much faster than the $O(n^2 \log n)$ running time of the modified Huffman algorithm [7]. Since the heuristic is strongly based on the optimality properties, it also performs much better than the modified Huffman algorithm. In fact our experimental results show that our heuristic gives optimal results in most cases.

The rest of the paper is organized as follows. In section 2, we define the low-power gate decomposition problem. In section 3, we describe Huffman's algorithm for tree construction and identify two special cases of the problem which can be solved. Section 4 studies the properties of an optimal decomposition tree. Based on these properties, section 5 presents two algorithms: one exact algorithm and one heuristic. Section 6 gives the experimental results and some concluding remarks. Due to space limit, most of the proofs are omitted. All of them can be found in [9].

2 Problem formulation

In technology decomposition, we need to decompose a multi-input gate into a tree of two-input gates. Since an OR gate can be treated as a NAND gate with negations of

0-89791-993-9/97 $10.00 © 1997 IEEE

the inputs, what we need to solve is how to decompose an n-input AND gate into a tree of 2-input AND gates. We call this *gate decomposition*.

We will treat the signals in a circuit as random variables and define the *signal probability* of a signal x as the probability of x being 1, denoted by $p(x)$. We use the same model as in [6, 7], that is, we assume the *zero delay model* where gate delays are assumed to be zero and thus signal transitions due to glitching are ignored; primary inputs are assumed to be uncorrelated (*spatial independent*); and the present input signal value is independent of those in the past (*temporal independent*). Under these assumptions, given the input signal probabilities and a decomposition tree, the probabilities of internal signals can be computed as follows. Start from the primary inputs, for each $z = x$ AND y, let $p(z) = p(x)p(y)$. Thus, the signal probability of any node v is equal to the product of all leaf probabilities in the subtree rooted at v. For example, Figure 1 shows one gate decomposition and all signal probabilities of the nodes.

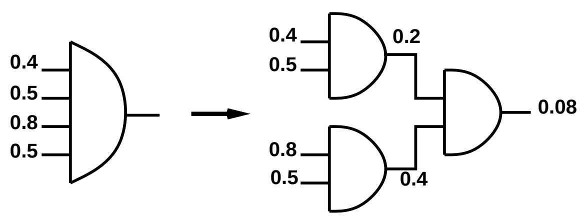

Figure 1: Gate decomposition

The switching activity E_{sw} depends on the implementation logic style. In p-domino logic designs, the gate outputs are pre-discharged to 0, thus the switching activity of a node is equal to the probability of being 1. Let $T = (V, E)$ represent the decomposition tree, and $p(v)$, for any $v \in V$, denote the output signal probability of node v. The objective function we want to minimize in domino logic is $\sum_{v \in V} p(v)$. Because of this simple objective function, it can be shown that Huffman's algorithm can be used to give an optimal decomposition tree in domino logic designs [7].

Because of the pre-discharges or pre-charges, domino logic designs dissipate more power than static logic designs, which never do extra charges or discharges. In static logic, under the temporal independence assumption, the switching activity E_{sw} of signal x can be written as

$$\begin{aligned} E_{sw}(x) &= Pr[x : 0 \to 1] + Pr[x : 1 \to 0] \\ &= Pr[x = 0]Pr[x = 1] + Pr[x = 1]Pr[x = 0] \\ &= 2Pr[x = 1]Pr[x = 0] \\ &= 2p(x)(1 - p(x)) \end{aligned}$$

However, in their recent work [8], Wu *et al.* showed that, even in the absence of temporal independence, $2p(x)(1 - p(x))$ also gives the expected value of the switching activities among all sequences that satisfy the given signal probability.

The problem we will solve in this paper can be defined as follows.

Low-power gate decomposition problem: Given an n-input AND gate with inputs $s_1, s_2, \ldots, s_n$ and their signal probabilities $p(s_1), p(s_2), \ldots, p(s_n)$, construct a tree $T = (V, E)$ of 2-input AND gates with $s_1, s_2, \ldots, s_n$ as its leaves such that

$$E_{sw}(T) = \sum_{v \in V} p(v)(1 - p(v))$$

is minimized.

According to Knuth [2], the number of different *labeled oriented* binary trees with n leaves is $\binom{2n-1}{n-1}(2n-2)!/2^{n-1}$. In a decomposition tree, only leaves are labeled, the internal nodes are indistinguishable. Therefore, the number of different decomposition trees is

$$\frac{\binom{2n-1}{n-1}(2n-2)!}{2^{n-1}(n-1)!} > (2n-1)^{n-1}.$$

Thus, an exhaustive enumeration method is prohibitively expensive. Tsui *et al.* [7] found Huffman's algorithm can not solve this problem. Instead, they gave a heuristic which was called modified Huffman algorithm. It starts with a forest composed of all the inputs, and incrementally combines two trees into one until there is only one tree. It is a greedy algorithm, and each time tries all pairs and chooses the combination which gives the minimum increase on the objective function. The time complexity of the algorithm is $O(n^2 \log n)$ [7].

This algorithm is by far not optimal. This can be shown by a simple example. Here we have six input signals with the following probabilities: 0.4, 0.4, 0.4, 0.94, 0.94, 0.95. The decomposition tree constructed by the modified Huffman algorithm is shown in Figure 2(a), where the summation of switching activities is 1.3337. Nevertheless, a decomposition tree shown in Figure 2(b) has 1.22748 as its total switching activities.

3 Huffman's algorithm

Given n leaves $v_1, v_2, \ldots, v_n$ with their weights $w(v_1), w(v_2), \ldots, w(v_n)$, Huffman [3] gave an algorithm to construct a binary tree with minimum weighted path length $\sum_{i=1}^{n} w(v_i) l_i$, where l_i is the path length from the root to v_i. The algorithm can be described as follows. Starting from a forest composed of all the leaves, it combines two trees with the minimum weights, use the summation of the weights as the weight of the combined tree and substitute the two trees by the combined one; this process is continued until there is only one tree.

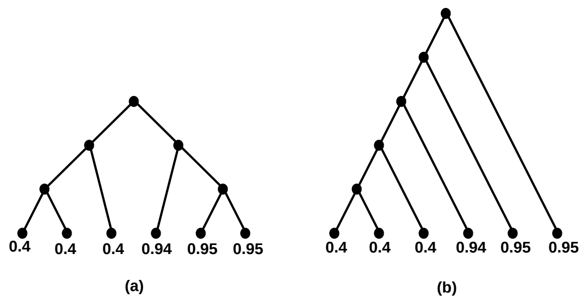

Figure 2: (a) Decomposition tree by modified Huffman has switching activities 1.3337; (b) A decomposition tree with switching activities 1.22748

If, for each internal node r with two children u and v, we define the weight $w(r) = w(u) + w(v)$, then

$$\sum_{i=1}^{n} w(v_i) l_i = \sum_{u \in V} w(u),$$

where V represents the set of internal nodes. This formulation leads us to consider Huffman's algorithm for the low-power gate decomposition problem. Unfortunately, it can not solve the problem in general case [7]. However, we find that under some conditions Huffman's algorithm can give optimal solutions. Before we give these conditions, we will describe two variations of Huffman's algorithm, which are a little different with the original one.

> *Min-Huffman algorithm*: Start with all the input signals; combine the two signals of minimum probabilities and substitute the two signals with the new signal; continue the process until there remains only one signal.

> *Max-Huffman algorithm*: Start with all the input signals; combine the two signals of maximum probabilities and substitute the two signals with the new signal; continue the process until there remains only one signal.

As stated by the following theorem, two special cases can be solved efficiently.

Theorem 1 *If all input signal probabilities are not greater than 0.5, the low-power gate decomposition problem can be solved by the Min-Huffman algorithm; If the product of all input signal probabilities is not less than 0.5, it can be solved by the Max-Huffman algorithm.*

4 Optimality properties

In previous section, we identified two special cases of the low-power gate decomposition problem which can be solved efficiently. In order to solve the general case, in this section, we will study the properties of an optimal decomposition tree.

First, we have the following simple observations.

Lemma 1 *On any path from a leaf to the root in a decomposition tree, the signal probabilities are decreasing. Each subtree in an optimal decomposition tree is also optimal.*

Further analysis gives us the following result.

Lemma 2 *In an optimal decomposition tree, all inputs whose probabilities are not greater than 0.5 must form a separate subtree.*

Lemma 2 tells us, in order to construct an optimal decomposition tree, we can always combine the signals whose probabilities are not greater than 0.5 into a subtree. By Lemma 1, this subtree needs to be an optimal one. According to Theorem 1, it can be constructed by the Min-Huffman algorithm. In fact, since the product of two smallest probabilities is still the smallest, in the Min-Huffman algorithm, signals are combined sequentially from low probability to high probability.

Similar analysis gives the following lemma.

Lemma 3 *In an optimal decomposition tree, the internal nodes whose probabilities are not greater than 0.5 form a path.*

In order to present the next optimality property, we need to define two labels for each node in an optimal decomposition tree. For each v, let *level*(v) be the distance of v from the root. That is, the root has level 0, its children have level 1, etc. For each v, if v is an internal node and $p(v) \leq 0.5$, then let *rank*$(v) = 0$. Otherwise, let *rank*(v) be the minimum distance of v from any node in rank 0. The property can be stated as follows.

Theorem 2 *Let u and v be any two nodes in an optimal decomposition tree. If rank(u) = rank$(v) \neq 0$ and level$(u) <$ level(v), then $p(u) \geq p(v)$.*

This theorem states that, in an optimal decomposition tree, for the nodes in the same rank other than 0, the probabilities are non-increasing with respect to their levels. According to the definition, the probability of each internal node in rank 1 is greater than 0.5. By Theorem 1, each subtree rooted at rank 1 node can be constructed by the Max-Huffman algorithm. Therefore, it is possible to arrange each subtree in such a way that, in each rank, the probabilities is non-decreasing from left to right. Under these arrangements, an optimal decomposition tree can be visualized in Figure 3, where the nodes in rank 0 form a path, and the probabilities in other ranks are non-decreasing along the arrows.

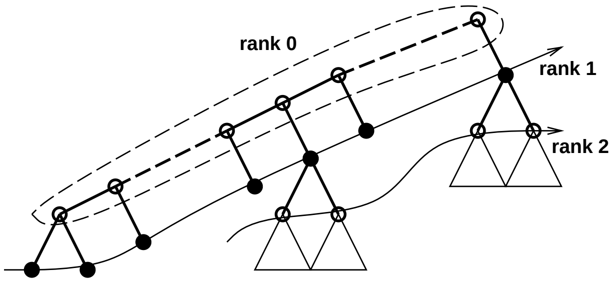

Figure 3: Probabilities are non-decreasing along the arrows in an optimal tree

The following theorem gives another important property of an optimal decomposition tree.

Theorem 3 *Let u and v be any siblings in an optimal decomposition tree such that $0.5 < p(u) < p(v)$, there can not exist node y in the tree such that $p(u) < p(y) < p(v)$.*

5 Decomposition algorithms

In the previous section, we have derived some properties an optimal decomposition tree must have. Since these properties are necessary conditions of an optimal tree, other trees which do not observe them need not to be considered during the optimization process. This can reduce the search space and help us to design an efficient algorithm for the low-power gate decomposition problem.

The following theorem combines all optimality properties given in previous section and is the basis of our exact algorithm.

Theorem 4 *Given n input signals $s_1, s_2, \ldots, s_n$ such that $p(s_1) \leq p(s_2) \ldots \leq p(s_n)$, there is an optimal decomposition tree where s_n either is combined with s_{n-1} or is a direct child of the root.*

Proof: We have two cases based on $p(s_{n-1})$.

Case 1. $p(s_{n-1}) \leq 0.5$. We claim s_n must be a direct child of the root in an optimal tree. Here we have $p(s_i) \leq 0.5$ for all $1 \leq i \leq n-1$. If $p(s_n) \leq 0.5$, according to Theorem 1, the optimal tree can be constructed by the Min-Huffman algorithm and s_n will be a direct child of the root. On the other hand, if $p(s_n) > 0.5$, according to Lemma 2, signals $s_1, s_2, \ldots, s_{n-1}$ must form a separate subtree, which will finally be combined with s_n. This also means s_n is a direct child of the root.

Case 2. $p(s_{n-1}) > 0.5$. We show there is an optimal tree where s_n either is combined with s_{n-1} or is a direct child of the root. Denote the sibling of s_n in an optimal tree by s. According to Lemma 1, we have $p(s) \leq p(s_{n-1})$. If $p(s) > 0.5$ then it must be that $p(s) = p(s_{n-1})$. Otherwise, we will have $0.5 < p(s) < p(s_{n-1}) < p(s_n)$, which contradicts with Theorem 3. But if $p(s) = p(s_{n-1})$, we can always exchange the subtree rooted at s with s_{n-1} and get an optimal tree where s_n is combined with s_{n-1}. On the other hand, if $p(s) \leq 0.5$, then, let v be the parent of s and s_{n-1}, we will have $p(v) \leq 0.5$. This means *rank*$(v) = 0$ and hence *rank*$(s_n) = 1$. Since $p(s_n)$ is the maximum, according to Theorem 2, *level*(s_n) must be the minimum. Therefore, *level*$(s_n) = 1$, which means s_n is a direct child of the root. $\square$

In other words, the theorem says that there is always an optimal tree between the two configurations shown in Figure 4. More specifically, if $p(s_{n-1}) \leq 0.5$ it must be configuration I; otherwise, it can be either configuration I or configuration II.

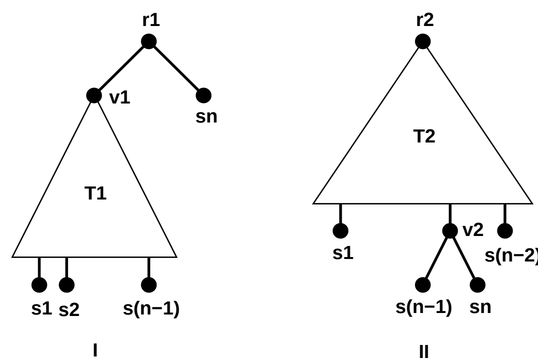

Figure 4: Two configurations of an optimal decomposition tree

Based on Theorem 4, we can design an exact algorithm for the low-power gate decomposition problem as follows. Given n input signals, we first sort them according to their probabilities such that $p(s_1) \leq p(s_2) \ldots \leq p(s_n)$. If $p(s_{n-1}) \leq 0.5$, we construct configuration I; otherwise, we construct both configurations I and II and output the one with the minimum switching activities. According to Lemma 1, the subgraphs T_1 and T_2 in Figure 4 must also be optimal. Since their input sizes are both only $n-1$, we can construct them recursively. This algorithm is called **ExDecomp** and its pseudo-code is given in Figure 5.

The correctness of the algorithm comes directly from Theorem 4 and can be stated as the following corollary.

Corollary 4.1 *The* **ExDecomp** *algorithm exactly solves the low-power gate decomposition problem.*

At each recursion in **ExDecomp**, we need to store the current configuration, which is upper bounded by n. Since the recursion depth is at most n, the space usage in the worst case is n^2. Let $T(n)$ represent the running time of

```
Input: a set of signals S = {s_1, ..., s_n}
        such that p(s_1) ≤ ... ≤ p(s_n)
Output: a decomposition tree T
ExDecomp(S)
{
   if |S| = 1
     return s_n;
   T_1 = ExDecomp(S − {s_n});
   if (p(s_{n−1}) ≤ 0.5)
     return combine(T_1, s_n);
   s = combine(s_{n−1}, s_n);
   T_2 = ExDecomp(S + {s} − {s_{n−1}, s_n});
   if (E_sw(combine(T_1, s_n)) < E_sw(T_2))
     return combine(T_1, s_n);
   else return T_2;
}
```

Figure 5: Pseudo-code of **ExDecomp**

```
Input: a set of signals S = {s_1, ..., s_n}
        such that p(s_1) ≤ ... ≤ p(s_n)
Output: a decomposition tree T
HeuDecomp(S)
{
   if (p(s_{n−1}) ≤ 0.5 or choose configuration I) {
     T_1 = HeuDecomp(S − {s_n});
     return combine(T_1, s_n);
   }
   else {
     s = combine(s_{n−1}, s_n);
     T_2 = HeuDecomp(S + {s} − {s_{n−1}, s_n});
     return T_2;
   }
}
```

Figure 6: Pseudo-code of **HeuDecomp**

ExDecomp on an instance of size n. It is easy to see that

$$T(n) = 2T(n-1) + n.$$

This gives us $T(n) = O(n2^n)$. Although in the worst case it is still need exponential time, compared with the total of more than $(2n-1)^{n-1}$ decomposition trees, it is efficient.

Besides the exact algorithm, the optimality properties can also be used to derive a set of efficient heuristic algorithms. As we can see, the complexity of **ExDecomp** comes from the fact that it is not known beforehand which configuration in Figure 4 will give the minimum switching activities. Trade accuracy for speed, we can use heuristics to choose only one configuration at each recursion. This gives us the algorithm scheme shown in Figure 6 which can be tuned into different heuristic algorithms based on different decision criteria.

The heuristic we used in our implementation can be described as follows. Since the structures of T_1 and T_2 in Figure 4 are not known until we recursively construct them, we can not compare their switching activities beforehand. But we can find that, for the two trees, except one leaf, all other $n-2$ leaves are the same. Therefore, we can assume the difference between the internal switching activities of T_1 and T_2 is not too much. We also know $p(r_1) = p(r_2)$. So the only concern comes from the difference between v_1 and v_2. Our decision criteria then is: if $E_{sw}(v_1) < E_{sw}(v_2)$, choose configuration I, otherwise choose configuration II.

Since only one configuration is chosen at each recursion in **HeuDecomp**, we need only keep one copy of the tree structure, hence the space usage is only n. Implemented by the priority queue data structure [1], the running time can also be upper bounded by $O(n \log n)$, which is much faster than the modified Huffman algorithm. Furthermore, since **HeuDecomp** is strongly based on the optimality properties, its performance should be better than that of the modified Huffman algorithm. This is supported by our experimental results.

6 Experimental results

We implement both the exact algorithm **ExDecomp** and the heuristic algorithm **HeuDecomp** in C++ on a Sun Sparc 5 workstation. Our experiments focus on two aspects: the running time of the exact algorithm and the performance of the heuristic. In order to compare the performance of the heuristic, we also implement the modified Huffman algorithm [7].

According to Lemma 2 and Theorem 1, the input signals whose probabilities are not greater than 0.5 can be easily combined into a subtree by the Min-Huffman algorithm. Therefore, the complexity only depends on the number of signals whose probabilities are greater than 0.5. In our experiments, the input signal probabilities are randomly generate, and based on the above reason, all signal probabilities are generated to be greater than 0.5.

On each different input size ranging from 5 to 20, we randomly generated 100 instances. We run **ExDecomp**, **HeuDecomp** and the modified Huffman algorithm on each of them. We compute the average running time of **ExDecomp** on each input size. To measure the performance

Table 1: Experimental results

#input	ExDecomp time(sec.)	Modified Huffman #bad	MaxRatio	AvgRatio	HeuDecomp #bad	MaxRatio	AvgRatio
5	0.0009	43	5.51%	0.76%	6	0.430%	0.010%
6	0.0014	66	5.59%	1.29%	6	0.187%	0.006%
7	0.0028	83	7.33%	2.30%	9	0.288%	0.008%
8	0.0057	89	11.31%	4.05%	9	0.268%	0.007%
9	0.0105	93	9.95%	5.02%	8	0.254%	0.008%
10	0.0187	93	14.87%	5.97%	5	0.265%	0.005%
11	0.0424	96	14.61%	7.55%	1	0.108%	0.001%
12	0.0752	97	16.17%	8.45%	2	0.111%	0.002%
13	0.1377	100	19.41%	9.78%	0	0.000%	0.000%
14	0.3050	97	19.97%	9.83%	0	0.000%	0.000%
15	0.5465	100	20.44%	11.32%	0	0.000%	0.000%
16	1.1539	98	20.82%	11.09%	0	0.000%	0.000%
18	3.9232	99	25.23%	12.25%	0	0.000%	0.000%
20	14.2497	100	29.55%	12.32%	0	0.000%	0.000%

of **HeuDecomp** and the modified Huffman algorithm, we compare their solutions with the optimal solution given by **ExDecomp**. The number of non-optimal solutions is counted. For each instance I, let $Opt(I)$ represent the optimal solution, we use the ratio

$$R = \frac{S(I) - Opt(I)}{Opt(I)}$$

to measure the performance of solution $S(I)$. For each algorithm, the maximum and average ratios are computed.

Based on the results reported in Table 1, we have the following conclusions. First, **ExDecomp** is efficient in practice. For 20 input probabilities which are greater than 0.5, the average running time is less than 15 seconds. In reality, usually only half of the input probabilities are greater than 0.5. This means a problem with 40 inputs can be solved in less than 15 seconds. Second, the performance of **HeuDecomp** is very good. Among all the 1400 solutions reported in Table 1, only 46 of them are not optimal. Among these non-optimal solutions, the largest deviation from the optimal solution is only 0.43%. Finally, an interesting phenomenon is that, with the increasing of the input size, **HeuDecomp** performs better and better. Starting from 13 inputs, all solutions given by **HeuDecomp** are optimal. Based on this phenomenon and the fact that **ExDecomp** runs very fast when the input size is not too large, we can use the following strategy for the low-power gate decomposition problem: if the input size is not too large, use **ExDecomp**; otherwise, use **HeuDecomp**.

References

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[2] D.E. Knuth, *Fundamental Algorithms*, Volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1968. Second edition, 1973.

[3] D.A. Huffman, A Method for the Construction of Minimum Redundancy Codes. In *Proceedings of the IRE*, volume 40, pages 1098-1101, Sept. 1952.

[4] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli, Decomposition of Logic Functions for Minimum Transition Activity. *Proceeding of the International Workshop on Low Power Design*, 1994.

[5] M. Pedram, Power Minimization in IC Design: Principles and Applications. *ACM Trans. on Design Automation of Electronic Systems*, Jan., 1996.

[6] V. Tiwari, P. Ashar, and S. Malik, Technology Mapping for Low Power. *ACM/IEEE Design Automation Conference*, 1993.

[7] C.-Y. Tsui, M. Pedram, and A.M. Despain, Technology Decomposition and Mapping Targeting Low Power Dissipation. *ACM/IEEE Design Automation Conference*, 1993.

[8] Q. Wu, M. Pedram, and X. Wu, A Note on the Relationship Between Signal Probability and Switching Activity. *Asian and South Pacific Design Automation Conference*, 1997.

[9] H. Zhou and D.F. Wong, An Exact Gate Decomposition Algorithm for Low-Power Technology Mapping. Technical Report TR97-21, Department of Computer Sciences, University of Texas at Austin, 1997.