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

\[ P = 0.5 V_{dd}^2 f_{clk} C_L E_{sw} \]

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. 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
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 prob-
ability 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 sig-
nal transitions due to glitching are ignored; primary in-
puts are assumed to be uncorrelated (spatial independent);
and the present input signal value is independent of those
in the past (temporal independent). Under these assump-
tions, given the input signal probabilities and a decompo-
sition tree, the probabilities of internal signals can be com-
puted as follows. Start from the primary inputs, for each
$z = x$ AND $y$, let $p(z) = p(x)p(y)$. Thus, the signal prob-
ability of any node $v$ is equal to the product of all leaf prob-
abilities in the subtree rooted at $v$. For example, Figure 1
shows one gate decomposition and all signal probabilities
of the nodes.

The switching activity $E_{sw}$ depends on the implementa-
tion logic style. In $p$-domino logic designs, the gate out-
puts 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 ob-
jective function we want to minimize in domino logic is
$\sum_{v \in V} p(v)$. Because of this simple objective function, it
is 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 de-
signs, 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

$$E_{sw}(x) = Pr[x: 0 \rightarrow 1] + Pr[x: 1 \rightarrow 0]$$

$$= Pr[x=0]Pr[x=1] + Pr[x=1]Pr[x=0]$$

$$= 2Pr[x=1]Pr[x=0]$$

$$= 2p(x)(1 - p(x))$$

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 ac-
tivities 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 sig-
nal 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 la-
beled oriented binary trees with $n$ leaves is $(2^{n-1})(2n -
2)!/(2^{2n-1})$. In a decomposition tree, only leaves are labeled,
the internal nodes are indistinguishable. Therefore, the
number of different decomposition trees is

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

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 the modified Huffman algorithm. It starts with a
forest composed of all the inputs, and incrementally com-
bines 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 Huff-
man algorithm is shown in Figure 2(a), where the sum-
mation 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 com-
bines two trees with the minimum weights, use the summa-
tion 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.
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 2:** (a) Decomposition tree by modified Huffman has switching activities 1.337; (b) A decomposition tree with switching activities 1.22748.
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 \( p(s) \leq p(s_{n-1}) \).

Another important property is that in a rank-0 tree, the sibling of \( s_n \) must form a separate subtree. 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.

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**.
we can find that, for the two trees, except one leaf, all other
ure 4 are not known until we recursively construct them, we
the only concern comes from the difference between
scribed as follows. Since the structures of
\( T \)
different decision criteria.

This can be tuned into different heuristic algorithms based on

Besides the exact algorithm, the optimality properties
can also be upper bounded by \( O(n \log n) \), which is much faster
HeuDecomp is strongly based on the optimality proper-
ties, its performance should be better than that of the mod-
ified Huffman algorithm. This is supported by our ex-

6 Experimental results

We implement both the exact algorithm \( \text{ExDecomp} \)
and the heuristic algorithm \( \text{HeuDecomp} \) in C++ on a Sun
Sparc 5 workstation. Our experiments focus on two as-
pects: the running time of the exact algorithm and the per-
formance of the heuristic. In order to compare the per-
formance 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 ex-
periments, the input signal probabilities are randomly gen-
erate, and based on the above reason, all signal probabili-
ties 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 \( \text{ExDecomp}, \)
\( \text{HeuDecomp} \) and the modified Huffman algorithm on each
of them. We compute the average running time of \( \text{ExDecomp} \)
on each input size. To measure the performance
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 \( \text{Opt}(I) \) represent the optimal solution, we use the ratio

\[
R = \frac{S(I) - \text{Opt}(I)}{\text{Opt}(I)}
\]

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