Technology Decomposition and Mapping Targeting Low Power Dissipation

Chi-Ying Tsui, Massoud Pedram, Alvin M. Despain

CENG Technical Report 92-16

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-6006

Technology Decomposition and Mapping Targeting Low Power Dissipation

Chi-Ying Tsui, Massoud Pedram, Alvin M. Despain

23 October 1992

Abstract

In this paper, we address the problem of minimizing the average power dissipation during the technology dependent phase of logic synthesis. Our approach consists of two steps. In the first step, we generate a NAND decomposition of an optimized Boolean network such that the sum of average switching rates for all nodes in the network is minimum. Our power-efficient decomposition procedure is optimal for dynamic CMOS circuits with uncorrelated input signals and produces very good results for static CMOS. In the second step, we perform a power efficient technology mapping that finds an optimal power-delay trade-off value (subject to the unknown load problem) for given timing constraints. We obtain an average of 22% improvement in power at the expense of 13% increase in area and without degradation in performance on a number of benchmarks.

Contents

1 Introduction

1.1 Problem Definition

With the recent advances in microelectronic technology, more and more functions are being put in an embedded microsystem. For many embedded systems such as notebook computers, portable phones or other consumer electronics, long battery life and low heat dissipation are important design objectives. To achieve these goals, designers are willing to trade off area and/or performance for low average power dissipation.

The average power dissipation of a circuit may be improved by architectural or technological changes. Low power dissipation can also be achieved by lowering the supply voltages on the chips. However, lowering the supply voltage may create other design problems such as reduced noise margin, increased cross talk, etc.

In CMOS design, energy is dissipated during the charging and discharging of the gate capacitances. To reduce the power dissipation, we can lower the internal load capacitances and/or the average switching activity of the internal nodes. Many researchers have attempted to reduce the maximum power consumption by resizing the transistors to reduce the load capacitance [7]. In today's designs, reducing the average power consumption which depends on the average switching activity is however more important.

In this paper, we present techniques to lower the average power dissipation of a circuit. In particular, we address the problem of minimizing the average power dissipated in the synthesized circuits during technology-dependent phase of combinational logic synthesis (e.g. technology decomposition and mapping). The idea is to generate a NAND decomposition of the Boolean network with minimum switching activity and later *hide* the high switching nodes with large capacitive loads within complex gates during the technology mapping.

1.2 Our Power Dissipation Model

The average power consumption for a single gate in a synchronous CMOS circuit is given by

$$P_{avg} = 0.5 \times C_{load} \times \frac{V_{dd}^2}{T_{cycle}} \times E(transitions) \tag{1}$$

where C_{load} is the load capacitance of the gate, V_{dd} is the supply voltage, T_{cycle} is the clock cycle time and $E(transitions)$ is the expected switching probability at the output of the gate.

The switching probability of a gate depends on the design style, its logic function in terms of the circuit primary inputs, gate delays, and the switching probabilities of the primary inputs. For domino logic designs, the switching probability of a node is equal to probability of being 1 (for *p-type* circuit) or 0 (for *n-type* circuit) . For static designs, the switching probability of a node is equal to the sum of probabilities of *0->1* and *1->0* transition.

The primary input switching probabilities are often assumed to be independent for data path modules since their input data is often random. For some circuits such as finite state machine or instruction decoder of a microprocessor, the input switching probabilities are correlated and the correlations can be obtained from the opcode/state assignment or the state transition diagram. We will consider both correlated and uncorrelated input signals.

1.3 Prior Work

Previous work has mostly focused on the estimation of signal probabilities and the average power consumption. Cirit [4] estimates the dynamic power dissipation at the transistor level. The probability for the drain of a transistor to make a power consuming transition is calculated by multiplying the source's probability of being 1 or 0 with the gate's transition probability. This method assumes all signals are independent and thus does not consider the reconvergent fanout and input correlation.

Burch et al, [2] introduce the concept of a probability waveform. Given such waveforms at the primary input nodes, they derive the corresponding waveforms at the internal circuit nodes. Then, with some convenient partitioning of the circuit, they examine every sub-circuit and derive its expected current waveform based on the waveforms at its inputs. The inputs to the subcircuits are assumed to be independent.

Najm [12] describes an efficient technique to propagate the transition densities (average switching rates) at the circuit primary inputs into the circuit to give the transition densities at all internal and output nodes. His timing model assumes that the propagation delay from an input to the output of a logic block is independent of values at other inputs. The Boolean difference operator is used to express the output transition density in terms of the input transition densities. Transition densities are propagated through combinational logic without regard to its structure. Correlations between internal lines, which are due to reconvergence, are ignored during the propagation.

Ghosh et al, [6] propose symbolic simulation to produce a set of Boolean functions that represent the conditions for switching at each gate in the circuit. Given the input switching rates, the switching probability at each gate is calculated by performing a linear traversal of the Binary Decision Diagrams (BDDs) representation of the corresponding Boolean function [12]. A general delay model which correctly computes the

Boolean conditions that cause glitchings is used and correlations due to the reconvergence of input signals are taken into account.

The above methods are primarily concerned with the estimation of signal probabilities and analysis of the circuits. Shen et al, [14] present algorithms for reducing power dissipation during technology independent phase of logic synthesis. At first an optimized multi-level implementation of the circuit is obtained by the standard optimization script. Each node is then simplified in such a way that reduces the output switching probability. Logic transformations which realize each node in the network as a disjoint cover are applied to further reduce the average power dissipation. Nodes that are not on the critical paths are partially collapsed and re-implemented in two-levels of logic. The rationale is that in general two-level sub-networks dissipate less power than their multi-level counterparts.

Shen's work focuses on the technology independent phase. In contrast, we present methods that can optimize average power consumption during the technology dependent phase, e.g. technology decomposition and mapping.

1.4 Calculation of the Switching and Correlation Probabilities

We calculate the signal probability at the output of a node by first building a BDD corresponding to the global function of the node and then performing a linear traversal of the BDD as follows [12]. For a function $y = f(x_1, ...x_n)$ where the inputs are independent, probability of y is given by

$$w_{y=1} = w_{x_1=1} w_{f_{x_1}=1} + w_{\bar{x_1}=1} w_{f_{\bar{x_1}}=1} \tag{2}$$

where f_{x_i} and $f_{\bar{x_i}}$ are the cofactors of f with respect to x_i and $\bar{x_i}$. $w_{f_{x_i}=1}$ is calculated in a similar manner and hence a depth-first-traversal of the BDD, with a post-order evaluation of w at every node is all that is required to give $w_{y=1}$.

In this paper, we assume the zero delay model where gate delays are assumed to be zero [6] and ignore the signal transitions due to glitching. Primary inputs are assumed to be uncorrelated. For static circuits, we assume that the present input signal value is independent of the previous value. Hence, the transition probabilities are given by

$$w_{i_{0->1}} = w_{i=0} w_{i=1} \tag{3}$$

$$w_{i_{1(0->1)}|i_{2(0->1)}} = w_{i_1=0|i_2=0} w_{i_1=1|i_2=1} \tag{4}$$

respectively where $w_{i=x}$ is the probability of signal i assuming value x and $w_{i_1=x|i_2=y}$ is the probability that signal i_1 assumes x when signal i_2 assumes y.

1.5 Organization of the Paper

The rest of this paper is organized as follows. In Section 2 we discuss how a NAND-decomposed network that has minimum total average switching activity is constructed. Section 3 presents a power efficient technology mapping paradigm which exploits the power/delay tradeoff curves to generate a mapped network with minimum total switching activity subject to given timing constraints. Experimental results and conclusion remarks are presented in Sections 4 and 5. Proofs are omitted for sake of brevity.

2 Power-Efficient Technology Decomposition

Technology decomposition (the procedure for converting an optimized Boolean network into a NAND-decomposed network) is the precursor to the technology mapping step. It is an open problem to determine which of the possible NAND-decomposed networks yields an optimum solution when an optimum covering algorithm is applied [1].

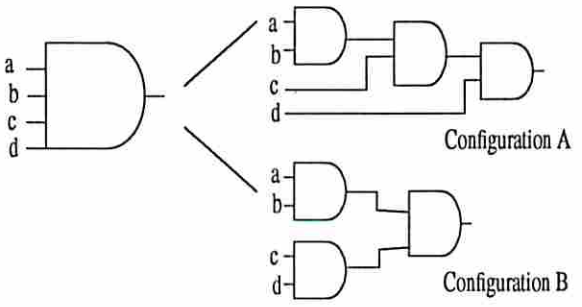
It is our belief and observation that a decomposition scheme which minimizes the sum of the switching activities at the internal nodes of the network, is a good starting point for power-efficient technology mapping. We illustrate the importance of decomposition on the average switching rate with a simple example (Figure 1). A four-input AND gate can be decomposed into a tree of three two-input AND gates in two ways. These two decompositions have different total switching activities. In particular, configuration A is better than Configuration B since the sum of the switching activities at the internal nodes for configuration A is smaller.[1]

We denote the problem of generating a NAND-decomposed network with minimum total switching activity as the MINPOWER decomposition. The performance-oriented version of the above problem requires that the increase in the height of the decomposed network (compared to the undecomposed network) be bounded. We refer to this problem as the BOUNDED-HEIGHT MINPOWER decomposition.

2.1 MINPOWER Tree Decomposition

We describe algorithms for solving the MINPOWER decomposition for a fanout-free logic function (i.e. a function that has a tree realization).

[1]Indeed, if we assume that the cell library has 2-input and 3-input AND gates but not a 4-input AND gate, the minimum power mapping with value 2.026 is obtained from configuration A and not from configuration B.



Assume p-type dynamic circuit is used and inputs are independent,
let P(i)= probability of node$_i$ being 1, P(a)=0.3, P(b)=0.4, P(c)=0.7
and P(d)=0.5. Total switching activities SR for A and B are

SR(A)=P(a)*P(b) + P(a)*P(b)*P(c)+P(a)*P(b)*P(c)*P(d) + P(a)+P(b)+P(c)+P(d) = 2.146

SR(B)=P(a)*P(b) + P(c)*P(d)+P(a)*P(b)*P(c)*P(d) + P(a)+P(b)+P(c)+P(d) = 2.412

Figure 1: Effect of technology decomposition on total switching activity

The basic algorithm is similar to the Huffman's algorithm for constructing a binary tree with minimum average weighted path length. We denote the leaves of a binary tree by $v_1, v_2, \ldots, v_n$, the "path length" from the root to v_i by l_i, and the weight of leaf v_i by w_i. Assuming that the root is at level zero (the highest level), leaf v_i is at level l_i. Given a set of weights w_i, there is a simple $O(nlogn)$-time algorithm due to Huffman [8] for constructing a binary tree such that the cost function $\sum_{i=1}^{n} w_i l_i$ is minimum.

Algorithm 2.1 (Huffman)

> Among the n nonnegative weights $w_1, w_2, \ldots, w_n$, find the two smallest weights w_1, w_2, say. Replace the two nodes by one node having the weight $W_1 = w_1 + w_2$ and two sons with weights w_1, w_2. Do this recursively for the $n-1$ weights $W_1, w_3, \ldots, w_n$. The final single node with weight $W_{n-1} = w_1 + w_2 + \ldots + w_n$ is then the root of the binary tree.

It is well known that the resulting binary tree minimizes $\sum_{i=1}^{n} w_i l_i$ over binary trees whose leaves have these weights. We denote this tree as the MINSUM tree.

In a more general setting, a *weight combination function* F is used to produce the weight of internal nodes during tree construction. For each tree T, a *tree cost function* $G(W_1, W_2, \ldots, W_{n-1})$ gives the cost.[2] Parker [9] characterized a wide class of weight combination functions, *quasi-linear functions*, for which the Huffman's algorithm produces optimal trees under corresponding tree cost functions. [3]

[2] In Huffman's original paper, $F(x,y) = x + y$ and $G = \sum_{i=1}^{n-1} W_i$. It is easy to show that $G = \sum_{i=1}^{n} w_i l_i$.

[3] A weight combination function F is quasilinear if $F(x,y) = \phi^{-1}(\lambda\phi(x) + \lambda\phi(y))$ where λ is a nonzero

It can be shown that $F(x,y) = max(x,y) + 1$ is a quasi-linear function and its corresponding tree cost function is $G = max_{i=1}^{n-1} W_i$. It is not hard to show that G is equal to $max_{i=1}^{n}(w_i + l_i)$ which is the weighted height measure of T.

If $F(x,y)$ is not a quasi-linear function, then the Huffman's algorithm does not, in general, produce the optimal solution. We propose the following $O(n^2 logn)$ greedy algorithm to solve the decomposition problem for the general weight combination functions. (See Section 4 for experimental results of this algorithm.)

Algorithm 2.2 (Modified Huffman)

> For every pair w_i and w_j of the n non-negative weights $w_1, w_2, \ldots, w_n$, compute $F_{ij}(w_i, w_j)$ and store in list L . Find the smallest F_{ij}, say F_{12}. Replace the two nodes by a single node having the weight $W_1 = F_{12}$ and two sons with weight w_1 and w_2. Eliminate all $F_{1k}(w_1, w_k)$ and $F_{2k}(w_2, w_k)$ from L. Compute $F_{1j}(W_1, w_j)$ and insert it into L. Do this recursively for the $n-1$ weights $W_1, w_3, \ldots, w_n$. The final single node with weight W_{n-1} is then the root of the binary tree.

To solve the MINPOWER tree decomposition problem, we must use appropriate weight combination and tree cost functions. We thus distinguish among two cases as follows.

2.1.1 Domino Dynamic Logic

For p logic block dynamic CMOS circuits (p circuits), the gate outputs are precharged to zero and switching occurs when the output changes to 1 during the evaluation phase. The switching probability for the output of a 2-input AND gate (without input signal correlations) is given by

$$W_o = w_{i_1} w_{i_2} \tag{5}$$

where W_o and w_{i_x} values are probabilities of the output and inputs assuming value 1. For n circuits since gates are precharged to 1 and transition occurs when output evaluates to 0, the corresponding formula for the switching probability is given by

$$W_o = 1 - (1 - w_{i_1})(1 - w_{i_2}) \tag{6}$$

where the W_o and w_{i_x} values are now probabilities of the output and inputs assuming value 0.

The merging function $F(w_{i_1}, w_{i_2}) = W_o$ is used during the AND decomposition. The corresponding tree cost function G is equal to $\sum_{i=1}^{n-1} W_i$.

Lemma 2.1

constant and ϕ is an invertible function.

W_o's given in (5) and (6) above are quasi-linear functions.

Proof

For (5), since w_i is within the range of [0,1] we can take $\phi(x) = -log(x)$ which is a convex decreasing function and λ= 1. This shows that W_o is quasilinear. Similar proof can be derived for (6).

Theorem 2.2

MINPOWER tree decomposition for dynamic CMOS circuits with uncorrelated input signals can be solved optimally by Huffman's algorithm using the weight combination functions (5) and (6).

If the input signals to the AND gates are correlated, (5) and (6) cannot be used as the merging function. The switching probabilities are then given by

For p circuits :

$$W_o = w_{i_1} w_{i_2|i_1} \tag{7}$$

For n circuits :

$$W_o = 1 - (1 - w_{i_1})(1 - w_{i_2|i_1}) \tag{8}$$

where $w_{i_2|i_1}$ is the conditional probability of i_2 given i_1. (7) and (8) are *not* quasi-linear functions. Hence, we must use the Modified Huffman's algorithm. After merging nodes i, j, the conditional probability between the newly formed node A and the remaining nodes (say k) is heuristically calculated by

$$W_{Ak} = ((w_{k|i} + w_{k|j}) \times w_{ij}/2 + (w_{j|k} + w_{j|i}) \times w_{ik}/2 + (w_{i|j} + w_{i|k}) \times w_{jk}/2)/3. \tag{9}$$

Alternatively, W_{Ak} can be calculated using BDDs and the procedure described in Section 1.4.

2.1.2 Static Logic

For static CMOS circuits, we need to minimize sum of the probabilities for output switchings from 0 to 1 and 1 to 0. Thus, the merging function F for a 2-input AND gate is equal to $W_{o_{0->1}} + W_{o_{1->0}}$ where

$$W_{o_{0->1}} = w_{i1_{0->1}} w_{i2_{0->1}} + w_{i1_{1->1}} w_{i2_{0->1}} + w_{i1_{0->1}} w_{i2_{1->1}} \tag{10}$$

and

$$W_{o_{1->0}} = w_{i1_{1->1}} w_{i2_{1->0}} + w_{i1_{1->0}} w_{i2_{1->1}} + w_{i1_{1->0}} w_{i2_{1->0}} \quad (11)$$

If input signals are correlated, conditional probabilities between the input signal transitions have to be used in order to calculate the output transition probability. $W_{o_{0->1}}$ and $W_{o_{1->0}}$ are then given by

$$W_{o_{0->1}} = w_{i1_{0->1}} w_{i2_{0->1}|i1_{0->1}} + w_{i1_{1->1}} w_{i2_{0->1}|i1_{1->1}} + w_{i1_{0->1}} w_{i2_{1->1}|i1_{0->1}} \quad (12)$$

and

$$W_{o_{1->0}} = w_{i1_{1->1}} w_{i2_{1->0}|i1_{1->1}} + w_{i1_{1->0}} w_{i2_{1->1}|i1_{1->0}} + w_{i1_{1->0}} w_{i2_{1->0}|i1_{1->0}} \quad (13)$$

Unfortunately, F is not quasi-linear and hence we must use the Modified Huffman's algorithm.

2.2 BOUNDED-HEIGHT MINPOWER Tree Decomposition

Here, the objective is to construct a MINPOWER binary tree for a given list of weights (signal switching probabilities) with the restriction that its height (defined as $max_i\ l_i$) does not exceed a given integer L. The best known algorithm for solving BOUNDED-HEIGHT MINSUM problem is an $O(nL)$ algorithm due to Larmore and Hirschberg [11].

Algorithm 2.3 (Larmore)

Let n be the number of leaf nodes and L be the height bound. Define a node to be an ordered pair(i, l) such that $i \in [1, n]$,which is called the index of the node and $l \in [1, L]$, which is called the level of the node. Define $width(i, l) = 2^{-l}$. If T is a tree, define $node\_set(T) = \{(i, l)|1 =< l =< l_i\}$ where l_i is the depth of ith leaf of T. Let T_i be the tree we want to decompose with depth no more than L and let each node in the node_set be an item which has width less than 1. For each $l \in [1, L]$, the list of nodes with width 2^{-l} is initialized as $((n, l), (n-1, l), ..., (1, l))$. The optimal Huffman tree with height bound L is thus equivalent to finding a minimal weight node_set of width $n-1$ which can then be reduced to an instance of the Coin Collector's problem of size nL. An instance(I, X) of the Coin Collector's problem is defined as given a set I of m items each of which has a *width* and *weight*, find a subset of S of I whose widths sum to X and has minimum total weight. The problem is solved by *Package-Merge* algorithm which is described by the following pseudo-code.

Package-Merge Algorithm(I, X)

$S = 0$
for all d, L_d = list of items having width 2^d, sorted by weight
while $X > 0$ loop
  $minwidth$ = the smallest term in the diadic expansion of X
  if $I = 0$ then
   return No solution
  else d = the minimum such that L_d is not empty
    $r = 2^d$
    if $r = minwidth$ then
      return No solution
    else if $r > minwidth$ then
      Delete the minimum weight item from L_d and insert into S
      $X = X - minwidth$
    end if
    P_{d+1} = PACKAGE(L_d)
      (PACKAGE merge items in consecutive pairs in L_d to form a new item in L_{d+1})
    discard L_d
    L_{d+1} = MERGE(P_{d+1}, L_{d+1})
  end if
end loop

This algorithm is optimal for trees with quasi-linear weight combination functions.

Theorem 2.3

> The BOUNDED-HEIGHT MINPOWER tree decomposition problem for dynamic circuits with uncorrelated input signals can be solved optimally in time $O(nL)$ by Larmore-Hirschberg's algorithm using the weight combination functions (5) and (6).

For the general merging functions, the Larmore-Hirschberg's algorithm has to be modified as follows. We replace the PACKAGE step with an algorithm similar to Algorithm 2.2. An item at level i is obtained by merging the pair of items at level $i-1$ which has the minimum weight combination value. The MERGE step is unchanged. We therefore perform n^2 weight calculations during the PACKAGE step and the run time is increased to $O(n^2L)$. Using the modified Larmore-Hirschberg's algorithm, the BOUNDED-HEIGHT MINPOWER tree decomposition can be solved heuristically for the general weight combination functions.

2.3 NAND Decomposition

The pseudo-code for power-efficient technology decomposition of a Boolean network Γ, given a vector of arrival times α at the network primary inputs and a vector of required times β at the networks primary outputs, is shown below.[4] Note that during the technology-decomposition step, we use the unit-delay model instead of more accurate timing models such as the library delay model. The reason is that the size and depth of the network after mapping will be quite different from that of the NAND-decomposed network and a rough timing model such as the unit-delay model is all we need (and is indeed more sensible than the library delay model).

```
function power_efficient_network_decomp (Γ, α, β)
Γ is a technology-independent, optimized Boolean network
α is a vector of arrival times at the primary inputs
β is a vector of required times at the primary outputs
begin
   calculate_switching_and_correlation_probabilities (Γ)
   Γ' = network_duplicate(Γ)
   for each node n ∈ Γ' (in postorder) do
      MINPOWER_node_decomp (n)
      update_switching_and_correlation_probabilities (Γ')
   end
   σ = calculate_node_slacks(Γ', Γ,α, β)
   while delay requirement not satisfied and some nodes
      have not yet be redecomposed do
      n = the node having the most negative σ_n value
      BOUNDED_HEIGHT_MINPOWER_node_decomp (n,σ_n)
      update_slack (Γ)
   end
end
```

In order to use the (Modified) Larmore-Hirschberg's algorithm, we must calculate the L values for nodes being decomposed as follows. First, we duplicate network Γ to obtain Γ' and do an unrestricted MINPOWER decomposition of Γ'. We then calculate the arrival times for all nodes in Γ using the NAND-decomposition of Γ' as an estimate. Given the required times β at the primary outputs, we calculate the slacks for all paths

[4] We assume that as a result of technology-independent logic optimization, all nodes in the network are simplified and have tree functions. Otherwise, a preprocessing AND-OR decomposition of complex nodes is used.

in the network.[5] The slacks are then distributed to nodes along the path as follows. Let $depth\_surplus_n$ of node n in Γ denote the difference between the height of the power-efficient NAND decomposition of node n and H_n, the height of a balanced decomposition of n. The height bound for n is estimated as $L_n = H_n + Min_{p \in paths(n)} \hat{S}_p$ where $paths(n)$ denotes the signal paths which go through n and

$$\hat{S}_p = S_p \times \frac{depth\_surplus_n}{\sum_{i=1}^{K} depth\_surplus_i}$$

for a path with total slack S_p and K nodes. After the slack assignment step, the node that has the most negative $\hat{S}_p$ is NAND-decomposed. In case of a tie, the node that is shared by a maximum number of paths is processed first. After the node is redecomposed, slacks of all nodes in Γ are updated and the above procedure is repeated until the delay requirements are satisfied or no further node re-decomposition can be carried out.

3 Power-Efficient Technology Mapping

Given a Boolean network representing a combinational logic circuit optimized by technology independent synthesis procedures and a target library, we bind nodes in the network to gates in the library such that average power consumption of the final implementation is minimized and timing constraints are satisfied.

Our power-efficient technology mapper follows a procedure similar to [3], with the difference that our objective is to minimize the sum over all gates of the average power dissipated in the mapped network subject to given required time constraints. First a postorder traversal is used to determine a set of possible arrival times at the root of the tree. Next, a preorder traversal is performed to determine the mapping solution that minimizes the average power subject to the require time constraints.

3.1 Terminology and Computational Models

Consider a match g at node n of a NAND-decomposed tree. The inputs to node n consist of nodes n_i which fanout to node n (that is, $n = n_1' + n_2'$ if n has two inputs or $n = n_1'$ if n has a single input). The nodes which are covered by match g are denoted by $merged(n, g)$. The nodes which are not in $merged(n, g)$ but fanin to $merged(n, g)$ are denoted by $inputs(n, g)$. The $mapped\text{-}parent(n_i)$ is some node n for which there exists a matching gate g such that $n_i \in inputs(n, g)$. Note that given node n and gate

[5] Slack for a path may be positive (indicating early arriving signal at the path endpoint) or negative (indicating late arriving signal).

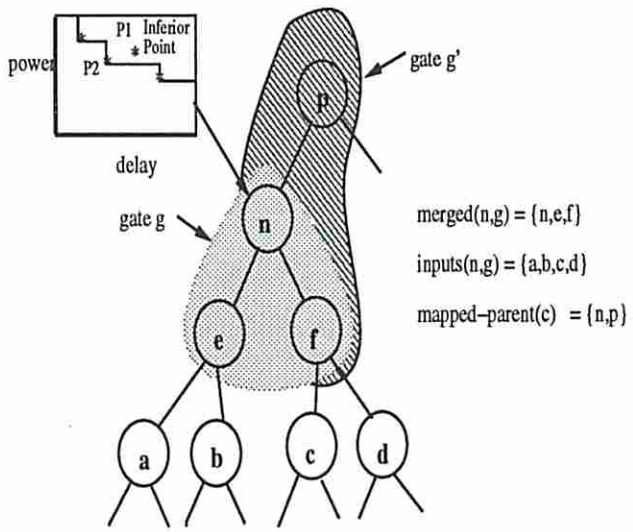


Figure 2: Terminology

g matching at n,$inputs(n,g)$ are uniquely determined. However, n_i may have many distinct mapped-parents (Figure 2).

With each node in the network we store a power-delay curve. A point on the curve represents the arrival time at the *output* of the node and the average power dissipated in its mapped transitive fanin cone (up to but excluding the node). The node itself is excluded for this calculation since the output load has yet to be decided by a later mapping. In addition to the power and delay values, the matching gate and input bindings for the match are also stored with each point on the curve. Points on the curve represent various mapping solutions with different tradeoffs between average power and speed. We are interested in a mapping with minimum average power satisfying delay requirements. Consequently, we can drop point P_1 on the curve if there exists another point P_2 on the curve with lower average power but equal or lower delay. This is possible because the solution associated with P_2 is superior to the solution associated with P_1 in terms of average power, delay or both. By dropping points, the power-delay curve can always be made monotonically non-increasing without loss of optimality. We would refer to P_1 as an *inferior point.* Point $P^* = (t^*, p^*)$ is a *non-inferior point* if and only if there does not exist a point $P = (t, p)$ such that either $t \leq t^*, p < p^*$ or $t < t^*, p \leq p^*$.

Lemma 3.1 *The power-delay function for a node contains the set of all non-inferior points and is monotonically non-increasing.*

In addition, if the difference in delay among two points is small (according to some user-specified parameter ϵ), we drop the point with higher average power without any

noticeable impact on the quality of the result. Similarly, points which are close in terms of their average power are merged together.

The power-delay function is therefore represented by a set of ordered pairs of real positive numbers (t, p), where a piecewise linear function $p = f(t)$ can be constructed which describes the set of all possible accumulated average power dissipations. This function describes all possible arrival time-average power tradeoffs at a given node. The power-delay function at an input node of the NAND-decomposed tree consists of ordered pairs (t, p) where t and p have been specified by the user (in case of primary inputs of the network) or have been previously computed (in case that inputs to this tree are outputs of other trees).

We have adopted the pin-dependent SIS library delay model as follows. Suppose that gate g has matched at node n, then the output arrival time at n is given by

$$arrival(n, g, C_n) = max_{n_i \in inputs(n,g)}(\tau_{i,g} + R_{i,g}C_n + arrival(n_i, g_i, C_i)) \qquad (14)$$

where $\tau_{i,g}$ is the *intrinsic* gate delay from input i to output of g, $R_{i,g}$ is the *drive* resistance of g corresponding to a signal transition at input i, C_n is the load capacitance seen at n, $arrival(n_i, g_i, C_i)$ is the arrival time at input i corresponding to load C_i seen at that input, and g_i is the best match found at input i.

There are two methods to calculate the dynamic value of the average power dissipation at some intermediate node during the mapping process.

Method 1

$$power(n, g) = \sum_{n_i \in inputs(n,g)} (0.5C_{n_i}\frac{Vdd^2}{T_{cycle}}E_{n_i} + power(n_i, g_i)) \qquad (15)$$

where E_{n_i} is the average switching activity at the output of gate n_i, C_{n_i} is the output load capacitance seen at node n_i, T_{cycle} is the cycle time, Vdd is the power supply voltage and $power(n_i, g_i)$ is the average power dissipated at input i.

Method 2

$$power(n, g) = 0.5C_n\frac{Vdd^2}{T_{cycle}}E_n + \sum_{n_i \in inputs(n,g)} power(n_i, g_i) \qquad (16)$$

where C_n is the output load capacitance seen at n.

When calculating the average power dissipation at node n, *Method 1* does not include the power contribution from n while *Method 2* includes its contribution by assuming a *default* load capacitance C_n[6]. In the zero-delay model, E_n is calculated based on the

[6]This is usually referred to as the "unknown load problem".

global function of n and not its particular implementation. Hence, E_n is independent of the gate matching at n. For this reason, it is possible (and indeed desirable) to use *Method 1* to calculate the dynamic power dissipation since it produces more accurate results compared to *Method 2*.

In *Method 1*, the average power contribution of the n's output will be included at *mapped-parent(n)*. When the mapping reaches a primary output, every point on the power-delay curve has a *power* value equal to the total average power dissipation of the mapped tree minus the power dissipation at the primary output. The power dissipation at the primary output depends on E_n at that output and the load capacitance connected to it which are both *independent* of the mapping configuration. Hence, it only causes a fixed shift of the curve along the power axis. It does not affect the selection of the optimal point from the power-delay curve[7]. In *Method 2*, the average power dissipation at a node is a function of the load it is driving. It thus has to be recalculated (similar to the timing recalculation procedure described in Subsection 3.2) in order to account for the difference between the estimated and actual loads.

Furthermore during the DAG mapping, the dynamic power at node n is divided by its fanout count (see Subsection 3.3). *Method I* will not reduce the power contribution due to the fanout edges of n while *Method 2* will do that. *Method I* is better in modeling the multiple fanout situation for the following reason. Preserving n after mapping will not create logic duplication in the transitive fanin of n and hence will reduce their power contribution. On the other hand it cannot reduce the power dissipation at the fanout of n and hence that component should not be divided by the fanout count of n.

We have therefore adopted *Method 1* to calculate the dynamic power dissipation at the intermediate nodes.

3.2 Tree Mapping

In this section, we focus on tree mapping. Later, we shall describe extensions to DAG mapping. In particular, we describe two tree-traversal operations which are applied to a NAND-decomposed tree in order to obtain a technology mapping solution which minimizes the average power dissipation while satisfying the timing constraints.

3.2.1 Postorder Traversal

On the first traversal, we begin at the leaf nodes of the NAND-decomposed tree. Since each leaf node possesses a set of possible arrival time - average power points which are

[7]Note that this argument will *not* hold for arrival time calculation since the shift along delay axis will not be a constant due to $R_{i,g}$ in (14).

reflected in its power-delay function, the power-delay function at any *mapped-parent*(n) must also reflect these possible arrival time - average power tradeoffs. A postorder traversal of the NAND-decomposed tree is performed, where for each node n and for each gate g matching at n, a new power-delay function is produced by appropriately merging the power-delay functions at the *inputs*(n, g). Merging must occur in the common region in order to ensure that the resulting function reflects feasible matches at the *inputs*(n, g). The power-delay functions for successive gates g matching at n are then merged by applying a *lower-bound merge* operation on the corresponding power-delay functions (see [3] for details). At a given node n, the resulting power-delay function will describe the arrival time - average power tradeoffs in propagating a signal from the tree inputs to the output of n.

3.2.2 Preorder Traversal

The user is allowed to select the arrival time - average power tradeoff which is most suitable for his application. Given the required time t at the root of the tree, a suitable (t, p) point on the power-delay function for the root node is chosen. The gate g matching at the root which corresponds to this point and *inputs*$(root, g)$ are, thus, identified. The required times t_i at *inputs*$(root, g)$ are computed from t, g, and the observation that *inputs*$(root, g)$ must now drive gate g. The preorder traversal resumes at *inputs*$(root, g)$ where t_i is the constraining factor and a matching gate g_i with minimum power dissipation satisfying t_i is sought.

3.2.3 Timing Recalculation

The gate delay is a function of the load it is driving. During the postorder tree traversal, the output of current node n_i, is not mapped hence the load capacitance is unknown (unless, all the gates in the library have identical pin capacitances). At this time the load value is assumed to be that offered by the smallest two-input NAND gate in the library. When we come to a node $n =$*mapped-parent*(n_i) with matching gate g, we know the exact load seen by n_i. This load is equal to the input capacitance of g and is, in general, different from the default load. Therefore, in order to calculate the arrival time at node n, the output arrival times for all nodes in *inputs*(n, g) must be adjusted to account for the change in the load capacitance. Similarly, during the preorder tree traversal, when a gate g is selected to match at n, the load seen by *inputs*(n, g) must be recalculated. (See [3] for more details.)

In order to account for this load change (Δ_i), the delay curves at the inputs have to be appropriately shifted. In particular, since the drive resistance of gate matching at n_i and giving rise to a point p_j on delay-curve of n_i is stored with that point, the delay shift is computed as $\Delta_i \times p_j.gate.drive$. (See [3] for more details.)

```
function power_efficient_technology_map(η)
η is a NAND-decomposed Boolean network
begin
    for each node n ∈ η (in postorder) do
      if n is a primary output then
        assign_best_gate(n, required_n)
      endif
      compute_power_delay_curve(n)
    end
end
```

3.3 DAG Mapping

Most of the real circuits are not trees, but general DAGs. The problem of mapping a DAG even for the constant load model is NP-hard [1]. Therefore, we resort to heuristics. One heuristic is to decompose the DAG into a number of trees such that the inputs for each tree come from other tree outputs or the primary inputs. During the power-delay curve computation step, entire trees are processed in postorder and power-delay curves are computed for each primary output of the DAG. During the gate assignment step, entire trees are mapped in preorder. This heuristic which does not allow mapping across tree boundaries is similar to that used by DAGON [10].

Alternatively, we could avoid decomposing the DAG into trees as follows. During the power-delay curve computation step, nodes are visited in postorder. For each node, we compute the power-delay curve as in case of trees. However, if the input for a candidate match at the node is coming from a multiple fanout node, we divide the average power contribution of that input by the fanout count of the input node . By reducing the average power contribution we tend to favor a solution in which multiple fanout nodes are preserved after mapping, which reduces logic duplication and improves the final mapped average power. This heuristic which permits tree boundary crossing only for nodes with relatively few fanouts was also adopted by the MIS mapper [5]. During the gate selection step, if we come to a node which has already been mapped, we check if the mapped solution at the node satisfies the timing requirement. If so, we keep the mapping; otherwise, we replace it with another solution from the power-delay curve which satisfies the current timing requirement and has minimum average power. The new solution may have higher average power compared to the previous solution.

The solution for circuits with multiple outputs also depends on the order in which the output cones are processed. During the power-delay curve generation step, when we are computing the signal arrival time for a match g at node n, we need to recalculate

numbers of input	% of getting optimal result
3	100
4	96
5	93
6	88

Table 1: Simulation result for the modified Huffman's algorithm

the load seen at $inputs(n, g)$. For $n_i \in inputs(n, g)$, some of the fanouts of node n_i (other than g) may have already been mapped (because they are part of a logic cone which has been processed), and hence, the contribution of these fanouts to the load can be calculated exactly. This incremental load recalculation will result in more accurate arrival time calculation at the output of n. Similar incremental load recalculation is applied during the gate assignment step.

4 Experimental Results

To evaluate the effectiveness of the Modified Huffman's algorithm, we performed simulations on static AND gate decomposition of a complex node with different number of inputs and assuming uncorrelated, random input probabilities. For each simulation, 500 input patterns were generated. For each pattern, all possible AND decompositions were enumerated to find the minimum power decomposition tree. The results are presented in Table 1. The Modified Huffman's algorithm generated optimal trees for average 94% of the trials.

To evaluate the effectiveness of the power-efficient technology decomposition and mapping, we applied our algorithms on subsets of the ISCAS-89 and the MCNC-91 benchmark sets. Static CMOS circuits were used in the experiments and all primary inputs were assumed to be independent. Power estimation was performed by the power estimation tool described in [6]. 20MHz clock frequency is assumed and all power estimates are in micro-Watts. The delays and loads for circuits were obtained using the pin-dependent SIS library delay model for *lib2.genlib*.

Tables 2 and 3 contain our experimental results using different technology decomposition and mapping combinations. All methods have the same starting point that is the circuits optimized by the SIS rugged script [13]. Methods I to III use area-delay mapping (*ad-map*) algorithm of [3] and Methods IV to VI use power-delay mapping (*pd-map*). Methods I and IV take in a NAND-decomposed network generated by the conventional tree decomposition algorithm. Methods II and V use the MINPOWER technology decomposition (*minpower_t_decomp*) while Methods III and VI use the BOUNDED-HEIGHT MINPOWER decomposition(*bh_minpower_t_decomp*).

circuit	I			II			III		
	gate area	delay	average power	gate area	delay	average power	gate area	delay	average power
s208	77	9.92	38.1	70	10.02	35.2	66	9.96	31.0
s344	156	11.35	110.4	153	12.41	105.1	153	12.41	107.1
s382	147	11.80	131.2	161	11.53	135.0	160	11.05	136.3
s444	161	13.36	141.35	153	14.44	130.3	153	14.44	130.2
s510	271	14.85	229.8	277	16.40	222.5	251	15.53	196.6
s526	182	10.81	139.5	180	10.12	144.9	185	10.03	142.0
s641	214	15.34	149.7	203	16.67	132.5	204	16.61	132.3
s713	204	16.13	145.1	191	18.15	131.7	197	17.84	130.4
s820	285	11.54	168.4	283	11.64	180.3	278	11.16	170.0
cm42a	27	3.01	28.75	27	2.67	30.0	27	2.67	30.0
x1	277	9.03	179.8	271	9.52	159.9	274	9.25	170.6
x2	55	5.40	39.1	51	5.04	30.2	51	4.22	28.5
x3	683	17.56	574.4	698	18.11	548.7	663	18.03	524.6
ttt2	210	11.30	173.0	205	11.37	165.7	206	11.36	168.3
apex7	228	9.83	139.0	235	11.60	151.3	241	10.74	138.4
alu2	311	33.26	277.9	343	31.62	293.8	341	33.61	282.3
ex2	308	12.37	132.2	307	12.59	127.7	301	12.42	130.5

Table 2: I : area-delay mapping with conventional decomposition , II : area-delay mapping with minpower_t_decomp , III : area-delay mapping with bh_minpower_t_decomp

To illustrate the impact of the (*minpower_t_decomp*) on the average power dissipation, we should compare the results of method pairs of I and II, and IV and V. The power dissipation is improved by an average 3.7% at the expense of 1.4We believe the small gain of *minpower_t_decomp* is due to the fact that most nodes in the optimized network are relatively simple due to the fast-extract and quick decomposition operations performed before the technology decomposition step. Thus, the *minpower_t_decomp* does not have much freedom to improve the power efficiency through NAND decomposition. From II and III, and V and VI, we see that *bh_minpower_t_decomp* improves the circuit performance and power dissipation by an average of 1.6% and 1.6% respectively.

To see the impact of the *pd-map* on the average power dissipation, we should compare the results of method pairs I and IV, II and V, and III and VI. It is seen that with *pd-map*, the power dissipation is improved by an average of 22% over *ad-map*. The active cell area is increased by an average of 12.4% while the performance is improved by an average of 1.1%.

circuit	IV			V			VI		
	gate area	delay	average power	gate area	delay	average power	gate area	delay	average power
s208	81	9.93	32.6	68	10.99	29.4	71	10.26	27.0
s344	167	11.76	101.4	165	11.43	84.9	166	11.43	83.6
s382	172	12.03	113.7	174	11.55	110.6	176	10.31	109.4
s444	171	13.69	102.0	174	14.44	103.9	167	13.58	97.7
s510	304	15.55	212.0	315	16.29	198.2	273	16.11	178.9
s526	218	9.88	112.5	206	9.79	100.7	215	11.17	102.3
s641	208	17.52	131.2	232	16.26	133.2	233	16.48	137.9
s713	207	16.57	118.5	237	19.03	140.4	236	18.45	145.2
s820	329	11.66	154.3	320	11.39	149.8	316	11.89	153.7
cm42a	26.5	3.05	15.3	29	2.68	10.3	27.8	2.93	12.5
x1	307	8.82	134.6	317	8.04	134.6	311	8.40	131.9
x2	54	4.85	30.9	54	4.77	24.3	50	4.29	24.0
x3	833	17.20	375.3	848	17.35	375.1	844	15.34	364.4
ttt2	227	11.71	137.3	231	12.08	127.5	227	11.91	139.1
apex7	275	10.64	129.3	282	11.32	120.7	276	10.60	131.2
alu2	403	30.44	221.8	438	30.52	260.2	414	27.96	236.2
ex2	355	11.83	100.3	365	11.01	107.8	354	11.68	97.7

Table 3: IV : power-efficient mapping with conventional decomposition , V : power-efficient mapping with minpower_t_decomp , VI : power-efficient mapping with bh_minpower_t_decomp

5 Concluding Remarks

We have presented algorithms for low power technology decomposition and mapping. In particular, we generate networks with minimum total switching activity and feed them to a delay constrainted power efficient technology mapper that *hides* the highly active nodes inside the mapped gates. Experimental results show that significant reduction in power dissipation is achieved with little or no performance degradation.

The idea of generating nodes with minimum switching activity can be extended to the technology independent phase of logic synthesis. Shen et al.[14] have already addressed some aspects of this problem. However, more work in node simplification, common sub-expression extraction and node decomposition is still needed. The average power dissipation also depends on the correlation among primary inputs. For finite state machines and instruction decoder, the correlation probabilities depend on the state and input encoding. Further research needs to be done to address these issues.

Acknowledgement –

The authors would like to thank Prof. Srinivas Devadas and Dr. Abhijit Ghosh who provided us with their power estimation tool. This work was supported in part by the National Science Foundation's Research Initiation Award under contract No. MIP-9211668 and the Defence Advanced Research Projects Agency under contract No. J-FBI-91-194.

References

[1] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. In *Proceedings of the IEEE*, volume 78, pages 264–300, February 1990.

[2] A. R. Burch, F. Najm, P. Yang, and D. Hocevar. Pattern independent current estimation for reliability analysis of CMOS circuits. In *Proceedings of the 25th Design Automation Conference*, pages 294–299, June 1988.

[3] K. Chaudhary and M. Pedram. A near-optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings of the 29th Design Automation Conference*, pages 492–498, June 1992.

[4] M. A. Cirit. Estimating dynamic power consumption of CMOS circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 534–537, November 1987.

[5] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Technology mapping in MIS. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 116–119, November 1987.

[6] A. A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of average switching activity in combinational and sequential circuits. In *Proceedings of the 29th Design Automation Conference*, pages 253–259, June 1992.

[7] B. Hoppe, G. Neuendorf, D. Schmit-Landsiedel, and W. Specks. Optimization of high-speed CMOS logic circuits with analytical models for signal delay, chip area, and dynamic power dissipation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9 No. 3:236–247, March 1990.

[8] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, September 1952.

[9] D. S. Parker Jr. Conditions for optimality of the Huffman algorithm. *SIAM Journal of Computing*, 9(3):470–489, August 1980.

[10] K. Keutzer. DAGON: Technology mapping and local optimization. In *Proceedings of the Design Automation Conference*, pages 341–347, June 1987.

[11] L. Larmore and D. S. Hirschberg. A fast algorithm for optimal length-limited Huffman codes. *Journal of the Association for Computing Machinery*, V 37 No. 3:464–473, 1990.

[12] F. Najm. Transition density, a stochastic measure of activity in digital circuits. In *Proceedings of the 28th Design Automation Conference*, pages 644–649, June 1991.

[13] H. Savoj and H. Y. Wang. Improved scripts in MIS-II for logic minimization of combinational circuits. In *Proceedings of the International Workshop on Logic Synthesis*, May 1991.

[14] A. A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proceedings of the IEEE International Conference on Computer Aided Design*, November 1992.