Trace Driven Logic Synthesis - Application to Power Minimization

Luca P. Carloni† Patrick C. McGeer‡ Alexander Saldanha‡

Alberto L. Sangiovanni-Vincentelli†

† University of California at Berkeley
Berkeley, CA 94720
{lcarloni,alberto}@ic.eecs.berkeley.edu

‡ Cadence Berkeley Laboratories
Berkeley, CA 94704-1103
{mcgeer,saldanha}@cadence.com

Abstract

A trace driven methodology for logic synthesis and optimization is proposed. Given a logic description of a digital circuit $\mathcal{C}$ and an expected trace of input vectors $\mathcal{T}$, an implementation of $\mathcal{C}$ that optimizes a cost function under application of $\mathcal{T}$ is derived. This approach is effective in capturing and utilizing the correlations that exist between input signals on an application specific design. The idea is novel since it propose synthesis and optimization at the logic level where the goal is to optimize the **average case** *rather than the* **worst case** *for a chosen cost metric. This paper focuses on the development of algorithms for trace driven optimization to minimize the switching power in multi-level networks. The average net power reduction (internal plus I/O power) obtained on a set of benchmark FSMs is 14%, while the average reduction in internal power is 25%. We also demonstrate that the I/O transition activity provides an upper bound on the power reduction that can be achieved by combinational logic synthesis.*

1 Introduction

Logic synthesis tools have traditionally been applied to problems where the cost function used in optimization depends only on the Boolean function representing the circuit to be implemented. This approach, of course, is appropriate for minimizing the area or longest path delay of a circuit. However, there are certain cost criteria which cannot be measured just by analysis of the Boolean function denoting the circuit. This is typically the case when the **average case** rather than the **worst case** cost is of interest. One such example is switching power minimization where it is well known that the correlations between signals have a profound impact on the switching activity of a circuit. Another example is event driven logic simulation where the speed of simulation is directly proportional to the event activity within the circuit which is determined by the correlations in the input stimulus. An even more compelling example is the approach used by computer architects in designing a modern microprocessor. Given the goal of improving the SPECmark performance of a processor, all architectures are tuned to perform optimally for the given instruction traces generated from the benchmark suites. All three examples correspond to optimization of an average case cost function.

Trace $\mathcal{T}_1$ $x_1x_2x_3$	Values g_1	g_2	Trace $\mathcal{T}_2$ $x_1x_2x_3$	Values g_1	g_2
1 1 1	1	0	1 1 0	0	0
1 0 0	0	0	1 0 1	0	0
1 1 1	1	0	1 1 0	0	0
0 0 0	0	0	0 0 1	0	0
1 1 1	1	0	1 1 0	0	0
0 0 0	0	0	0 0 1	0	0
1 1 1	1	0	1 1 0	0	0
1 0 0	0	0	1 0 1	0	0
1 1 1	1	0	1 1 0	0	0
0 0 0	0	0	0 0 1	0	0
0 1 1	1	1	0 1 0	0	0

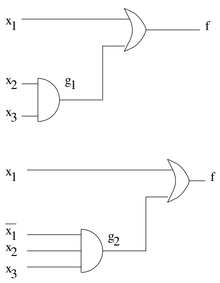


Figure 1: The two implementations of $f = x_1 + x_2 \cdot x_3$.

In this paper we propose a trace driven methodology for logic synthesis and optimization of Boolean circuits. Given an initial logic description of a digital circuit $\mathcal{C}$ and an expected trace of input vectors $\mathcal{T}$, an implementation of $\mathcal{C}$ that optimizes a cost function with respect to $\mathcal{T}$ is derived. This approach is effective in capturing and utilizing the correlations that exist between input signals and minimizes an average (or amortized) cost function rather than a worst case cost function. This paper is devoted to the case where the cost function is to minimize the average switching activity of the circuit under the sequence. The approach is mainly applicable to the combinational logic of finite state networks, where capturing the correlations between the state variables can play a significant role in power minimization.

Consider the simple example of a Boolean function f with *on-set*:

$$\mathcal{ON}(f) = \{x_1\overline{x_2}\,\overline{x_3}, x_1\overline{x_2}x_3, x_1x_2\overline{x_3}, x_1x_2x_3, \overline{x_1}x_2x_3\}$$

The minimum area implementation is $f = x_1 + g_1$ with $g_1 = x_2 \cdot x_3$. A second implementation of f is $f = x_1 + g_2$ with $g_2 = \overline{x_1} \cdot x_2 \cdot x_3$.

Consider the two traces $\mathcal{T}_1$ and $\mathcal{T}_2$ shown in Figure 1. Assume momentarily that only the number of gate switches is of concern, and the absolute power due to capacitance loading are being ignored. The first implementation generates 41 switches while the second yields 37 switches under trace $\mathcal{T}_1$. On the trace $\mathcal{T}_2$ the first circuit generates 30 switches while the second yields 35.

Now, assuming the input signals are uncorrelated, the transition probabilities [10] for both traces are identical, *i.e.*, $p(x_1) = 0.5$, $p(x_2) = 1.0$, $p(x_3) = 1.0$. Thus, previous approaches described by Najm [10] or Iman and Pedram [6] using signal and/or transition probabilities cannot distinguish between the two traces. Hence, the same implementation is selected for both traces. Our approach, in contrast, differentiates between the two traces and always selects the desirable implementation.

It is a widely accepted that the only effective approach to accurate power estimation is to perform logic simulation to obtain the switching activity from which the switching power can be estimated. The main reason behind this conclusion is the inaccuracy of the estimated power using probabilistic or statistical techniques. Specifically the correlations between the values on input signals within a vector as well as across vectors cannot be captured by existing probabilistic and statistical methods [9, 7]. Our approach extends the use of the actual vectors employed in power estimation to logic synthesis and optimization as well, thus narrowing the discrepancy in estimated power between analysis and synthesis tools while providing room for reduction in the power dissipation due to trace driven optimization.

Past approaches have suggested techniques to account for the effects of correlated inputs. One approach to approximating transition probabilities in the presence of correlated inputs is suggested in [15]. Another approach relevant to FSMs is the derivation of the steady state probabilities (*e.g.* using the Chapman-Kolmogorov equations [16]). However, capturing accurate transition probabilities on the inputs of a combinational network (even augmented with correlation values between pairs of inputs) still does not easily allow the derivation of accurate transition probabilities (and correlations between signals) on internal nodes of the network. This is the major drawback we seek to remedy using the trace-driven approach. All forms of signal correlations are captured by this method. Of course, the assumption required for successful application is that the trace set be representative of the input values during actual operation.

There is no *a priori* restriction on the length of the trace set that is used in the proposed methodology. Although it is conceivable that the trace sets generated using a naive simulation based procedure may be represented by a smaller trace set without much impact on the information essential for logic optimization, our approach is only marginally impacted by large trace sets for two reasons: (1) a trace set is compactly represented as a set of vector pairs - each vector pair has an associated frequency count of its occurrence in the trace set, (2) the size of the trace set only impacts the time for simulations needed during the logic optimization algorithms. Most of these simulations are performed in a pre-processing step and the running time can be improved by employing state-of-the-art logic simulation techniques. Nonetheless, trace driven synthesis can be performed with traces which are produced by vector compaction tools: for example, Tsui *et al.* [14] propose a technique with a compaction ratio of 100X while preserving most correlations.

2 Trace driven logic optimization problem

The problem of trace driven logic optimization for minimum switching activity is: Given a logic circuit represented as a set of Boolean functions and a sequence of input vectors, synthesize a circuit which dissipates minimum power due to switching activity under the application of the given sequence.

The focus of our work is on reduction of the switching power. We assume that the application of proper physical design techniques will minimize the power dissipated due to short-circuit and leakage currents. The switching power is estimated using a zero-delay model during logic optimization. While the principal reasons guiding this decision are the speed of the zero delay logic simulation versus timing simulation and the lack of availability of good delay models at the technology independent stage of logic synthesis, any other model that accounts for glitch occurrence may be utilized. At the technology independent level we account for the fanout of each gate as well as the size of the gate by using a decomposition into two-input gates. The accuracy of the model is verified by reporting results for the power dissipation obtained by running timing simulation using delays and capacitance loads obtained after technology mapping of the synthesize circuits.

B^n is the set of all vertices (or vectors) in the n-dimensional Boolean space. A **trace** $\mathcal{T} = \{v_1, v_2, ..., v_L\}$ of dimension n is an ordered sequence of vectors in B^n. Every input either has value 0 or 1 in each vector in a trace. A switch occurs when the value of f changes from 0 to 1 or 1 to 0. The switching activity of f under $\mathcal{T}$ is deduced by considering pairs of vectors. Let $\mathcal{P}_\mathcal{T}$ denote the set of pairs of adjacent vectors of $\mathcal{T}$, *i.e.*,

$$\mathcal{P}_\mathcal{T} = \{(v_i, v_j) \mid v_i \in \mathcal{T}, v_j \in \mathcal{T}, i = 1, ..., L{-}1, j = i{+}1\}$$

Associated with each element $(v_i, v_j) \in \mathcal{P}_\mathcal{T}$ is the frequency of occurrence of v_i and v_j as successive vectors in $\mathcal{T}$, denoted $\mathcal{F}_\mathcal{T}(v_i, v_j)$.

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$
vector	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$	$x_1x_2x_3x_4$
v_1	0 0 0 0	0 1 1 1	1 1 1 1
v_2	0 0 0 1	1 0 0 1	0 0 0 0
v_3	0 0 1 0	0 1 1 1	1 1 1 1
v_4	0 0 1 1	1 0 0 1	0 0 0 0
v_5	0 1 0 0	0 1 1 1	1 1 1 1
v_6	0 1 0 1	1 0 0 1	0 0 0 0
v_7	0 1 1 0	0 1 1 1	1 1 1 1
v_8	0 1 1 1	1 0 0 1	0 0 0 0
v_9	1 0 0 0	0 1 1 1	1 1 1 1
v_{10}	1 0 0 1	1 0 0 1	0 0 0 0

Table 1: Example traces

As an example, consider the three traces shown in Table 1. We have:

$$\begin{aligned}
\mathcal{P}_{\mathcal{T}_1} &= \{(0000, 0001), (0001, 0010), (0010, 0011), \\
&\quad (0011, 0100), (0100, 0101), (0101, 0110), \\
&\quad (0110, 0111), (0111, 1000), (1000, 1010)\} \\
\mathcal{P}_{\mathcal{T}_2} &= \{(0111, 1001), (1001, 0111)\} \\
\mathcal{P}_{\mathcal{T}_3} &= \{(0000, 1111), (1111, 0000)\}
\end{aligned}$$

Suppose we wish to calculate the switching activity of a gate $g = x_2 \cdot x_3$ on each of the traces. Define

$$\mathcal{P}_{\mathcal{T}}(g) = \{(v_i, v_j) \mid (v_i, v_j) \in \mathcal{P}_{\mathcal{T}} \wedge g(v_i) \neq g(v_j)\}$$

and

$$\mathcal{F}^g_{\mathcal{T}} = \Sigma_{(v_i,v_j)\in\mathcal{P}_{\mathcal{T}}(g)} \mathcal{F}_{\mathcal{T}}(v_i, v_j)$$

For the example, we have $\mathcal{F}^g_{\mathcal{T}_1} = 2$, $\mathcal{F}^g_{\mathcal{T}_2} = 9$, and $\mathcal{F}^g_{\mathcal{T}_3} = 9$, each of which denotes exactly the switching activity of g under the respective trace set. The switching activity of an input variable x is obtained by setting $g = x$ in the above formulae.

The key transformation performed by the formulae above is the representation of all the information of a trace needed for calculating the switching power by a set of vectors pairs, each associated with a count of the frequency of occurrence of the vector pair in the trace. This representation of a trace by a set of vector pairs is critical in the formulation and implementation of the algorithms for exact and heuristic two-level logic minimization and heuristics for multi-level logic optimization algorithms for decomposition and factorization.

3 Trace driven two-level minimization

In this section we address the problem of synthesis of a two-level logic circuit for a Boolean function with minimum power dissipation for a given trace. We first provide an exact formulation of the two-level covering problem for minimum power dissipation and then describe a heuristic procedure to allow the solution of larger problems. As already indicated by the example of Figure 1, the minimum power cover may contain non-prime implicants. The cost of an implicant is computed according to the following definition:

Definition 3.1 *Given trace $\mathcal{T} = \{v_1, v_2, \ldots, v_L\}$ and an implicant $q = x_1 \cdot x_2 \cdot \ldots \cdot x_K$ of f, the* **power cost** *of q under $\mathcal{T}$, denoted $W_{\mathcal{T}}(q)$, is*

$$W_{\mathcal{T}}(q) = \mathcal{ISA}_{\mathcal{T}}(q) + \mathcal{OSA}_{\mathcal{T}}(q) \tag{1}$$

with:

$$\mathcal{OSA}_{\mathcal{T}}(q) = 0.5 \cdot C_{load} \cdot V_{dd}^2 \cdot \mathcal{F}^q_{\mathcal{T}_1} \tag{2}$$

$$\mathcal{ISA}_{\mathcal{T}}(q) = \Sigma_{x_k \in q} 0.5 \cdot C_{x_k} \cdot V_{dd}^2 \cdot \mathcal{F}^{x_k}_{\mathcal{T}_1} \tag{3}$$

The term $\mathcal{OSA}_{\mathcal{T}}(q)$ represent the power dissipation due to the output switching activity of a gate g_q implementing q. C_{load} is the output load capacitance of g_q and V_{dd} is the supply voltage. Since there is an output transition only if $g_q(v_i) \neq g_q(v_{i+1})$, $\mathcal{OSA}_{\mathcal{T}}(q)$ sums the output power dissipation due to the entire trace $\mathcal{T}$. Similarly, the term $\mathcal{ISA}_{\mathcal{T}}(q)$ is the sum of the contributions to the total power dissipation due to the switching activity on each gate input x_k with capacitance C_{x_k}. Notice that $\mathcal{ISA}_{\mathcal{T}}(q)$ can be computed by simply observing the input trace and using the inputs that appear in q regardless of the specific function f.

Given the trace $\mathcal{T}$, the power cost of a cover $\mathcal{Q}$ of f, is given by

$$W_{\mathcal{T}}(\mathcal{Q}) = \sum_{q\in\mathcal{Q}} W_{\mathcal{T}}(q)$$

Thus we have the typical logic minimization problem: Given a Boolean function f and a trace $\mathcal{T}$, find the cover $\mathcal{Q}_{\mathcal{T}}$ of f which has the minimum power cost $W_{\mathcal{T}}(\mathcal{Q})$. A **minimum cover** is denoted $\mathcal{Q}^\star_{\mathcal{T}}$.

Following [1, 13], a minimum-weight unate covering problem (UCP) is solved using $W_{\mathcal{T}}(q)$ as the cost of each implicant q. In the case of trace driven two-level power minimization a minimum cover may contain an implicant which is not a prime. Since it is practically infeasible to solve the problem considering the set $\mathcal{I}$ of all the implicants [1], one wishes to identify a set $\mathcal{I}^\star_{\mathcal{T}} \subseteq \mathcal{I}$ of implicants sufficient to find a minimum cover. $\mathcal{I}^\star_{\mathcal{T}}$ is termed the set of candidate implicants.

The following definitions identify which implicants are elements of $\mathcal{I}^\star_{\mathcal{T}}$. An implicant q is **dominated** by an implicant $\tilde{q}$ if for every minimum cover which contains q there is another minimum cover which contains $\tilde{q}$. From Definition 3.1 it follows that an implicant q is dominated by another implicant $\tilde{q}$ if $q \subseteq \tilde{q}$ and $W_{\mathcal{T}}(q) \geq W_{\mathcal{T}}(\tilde{q})$ Given a function f and an input trace $\mathcal{T}$, a **candidate implicant** is an implicant of f that is not dominated by another implicant.

3.1 Generation of candidate implicants

An exact algorithm to generate $\mathcal{I}^\star_{\mathcal{T}}$ is presented in this section. The algorithm proceeds by generating implicants by *reduction* of larger candidate implicants thus eliminating implicants that are not candidate implicants as soon as possible.

Given $f : B^N \rightarrow B$, an implicant $q = x_1 \cdot x_2 \cdot \ldots \cdot x_K$ of f that is not a minterm is said to be **reducible**. The set $\mathcal{Y}$ of literals of the $N - K$ variables on which q does not depend is called the **external variable set** of q. A **reduced implicant** of q is an implicant $q \cdot y_1 \cdot y_2 \cdot \ldots \cdot y_H$ obtained by lowering q using one or more literals in $\mathcal{Y}$.

If q is a reducible implicant and qy is the reduced implicant obtained by lowering q using literal y, the following result holds for each possible trace $\mathcal{T}$:

$$\mathcal{ISA}_{\mathcal{T}}(qy) = \mathcal{ISA}_{\mathcal{T}}(q) + \mathcal{ISA}_{\mathcal{T}}(y)$$

Theorem 3.1 *Let q be a reducible candidate implicant and let qy be a reduced implicant of q obtained by lowering q using the literal y. qy is dominated by q if*

$$\Phi_{\mathcal{T}}(q, y) \geq 0 \tag{4}$$

where

$$\Phi_{\mathcal{T}}(q, y) = \mathcal{OSA}_{\mathcal{T}}(qy) - \mathcal{OSA}_{\mathcal{T}}(q) + \mathcal{ISA}_{\mathcal{T}}(y) \tag{5}$$

Theorem 3.1 provides a basic rule for the generation of the set of candidate implicants $\mathcal{I}^\star$. Starting from the set of primes of f, which are considered *a priori* candidate implicants, equation (4) is applied to decide if a reduced implicant obtained by lowering a prime with a single literal is a candidate implicant. The same equation is applied recursively to each new candidate implicant. However, Theorem 3.1 does not apply to implicants that can be generated by reduction from implicants not inserted in $\mathcal{I}^\star_{\mathcal{T}}$. Note that qy could still be part of $\mathcal{I}^\star_{\mathcal{T}}$ even if q is not. The following theorem applies to this case.

[1] For an n-input Boolean function f, $\mid \mathcal{I} \mid = O(2^{2^n})$.

Theorem 3.2 *Let q be a reducible candidate implicant and let $q_h = qy_h$ be a reduced implicant of q such that $\Phi_{\mathcal{T}}(q, y_h) > 0$. Consider $q_{h+1} = q_h y_{h+1}$, which is another reduced implicant of q obtained by lowering q_h with a literal y_{h+1} different from y_h. q_{h+1} is dominated by q if*

$$\Phi_{\mathcal{T}}(q, y_h) + \Phi_{\mathcal{T}}(q_h, y_{h+1}) \geq 0 \tag{6}$$

Recursive application of Theorems 3.1 and 3.2 yields all the implicants in $\mathcal{I}^{\star}_{\mathcal{T}}$. Note that memorizing dominated implicants ensures that a dominated implicant is not inserted later into $\mathcal{I}^{\star}_{\mathcal{T}}$. We conclude this sub-section presenting a simple example of trace driven two level minimization.

Consider the Boolean function f and trace $\mathcal{T}$ in Figure 2. The table shows the correspondences between the input vectors and the vertices of the Boolean space B^5. Table 2 reports the power cost of the implicants of f obtained using equation (1). The column labeled with $\mathcal{I}^{\star}$ is obtained using Theorem 3.1 and 3.2 (a $\star$ is indicated if the implicant is a candidate implicant, otherwise a dominating implicant is indicated). The minimum power cost cover $\mathcal{Q}^{\star} = \{c_2, p_2, p_4, p_5\}$ has $W_{\mathcal{T}}(\mathcal{Q}^{\star}) = 122$, while the area minimum cover is $\mathcal{Q} = \{p_1, p_2, p_3, p_4\}$ has power cost $W_{\mathcal{T}}(\mathcal{Q}) = 135$.

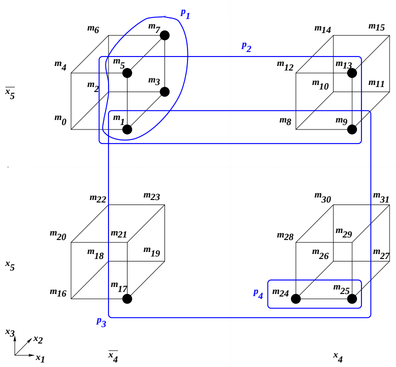
Although the approach of enumerating only candidate implicants and early discarding of dominated implicants performs efficiently in practice, the final step of finding a minimum cost unate covering problem is often a bottleneck. Even extending the classical branch-and-bound based covering algorithm with *lower-bound* computation techniques recently presented in [2] do not provide much improvement. As an alternative, we have adapted the two-level heuristic minimization program *espresso* [1] to perform trace driven two-level minimization. This new heuristic program, called *elp*, is described briefly in the next sub-section.

3.2 Trace driven two-level heuristic minimization

elp inherits two basic principles from *espresso*: the use of the *unate recursive paradigm* and the presence, at the core of the algorithm, of an optimization loop which is performed until no improvement in cost function is seen. The main difference, naturally, is the introduction of trace based weights for the implicants of f. This implies an important difference from *espresso*: essential primes and prime implicants are not necessarily the target of the minimization procedure. Instead, the goal of *elp* is to determine those primes which are "essential" from a power minimization point of view and then determine sub-covers for the remaining minterms which produce low switching activity.

Let p be an essential prime of f and let $\mathcal{EM}(p)$ be the set of minterms of f which are covered only by p. Consider all possible covers of $\mathcal{EM}(p)$ composed of reduced implicants of p : let $\mathcal{C}^{\mathcal{EM}}$ be the cover which has the minimum power cost $W_{\mathcal{T}}(\mathcal{C}^{\mathcal{EM}})$. If $W_{\mathcal{T}}(\mathcal{C}^{\mathcal{EM}}) \geq W_{\mathcal{T}}(p)$ then p is in the minimum cover $\mathcal{Q}^{\star}$ of f. To make this decision it is necessary to solve a *local* and *usually small* unate covering problem. Once the essential primes from a power standpoint have been extracted, *elp* enters the typical *espresso* loop.

The first step is *reduce* which attempts to move the current solution out of a local minimum. This routine is unchanged in *elp*. *elp* differs significantly from *espresso* in the *expand* step. While *espresso* examines each cube q of the current cover and replaces it with a prime implicant d such that $q \subseteq d$, *elp* looks for the largest cube d' (not necessarily prime) which contains q and that has smaller power cost. This step is still characterized by the fact that the cardinality of the cover does not increase. The final step derives an *irredundant* cover using a greedy covering problem. *elp* follows the same strategy as *espresso* using the power costs of the implicants as the weights in the covering problem.



Input Trace $\mathcal{T}$

$x_1x_2x_3x_4x_5$	vertex
1 0 1 0 0	m_5
0 1 1 0 0	m_6
0 0 1 0 0	m_4
1 1 1 0 0	m_7
1 1 0 0 0	m_3
1 0 0 1 0	m_9
1 0 0 0 0	m_1
0 0 0 0 0	m_0
1 0 0 0 0	m_1
0 1 1 1 0	m_{14}
1 1 1 0 0	m_7
0 1 1 0 0	m_6
1 0 0 0 0	m_1
0 0 0 0 0	m_0
0 0 1 0 0	m_4
1 0 0 0 0	m_1
0 0 0 0 0	m_0
1 1 0 1 0	m_{11}
0 1 1 1 0	m_{14}
1 0 0 0 0	m_1
1 0 0 1 0	m_9
0 0 0 0 0	m_0

Figure 2: The function f and input trace $\mathcal{T}$.

implicant q	$\mathcal{ISA}_{\mathcal{T}}(q)$	$\mathcal{OSA}_{\mathcal{T}}(q)$	$W_{\mathcal{T}}(q)$	$\mathcal{I}^{\star}$
$p_1 = x_1\overline{x_4}\overline{x_5}$	15	23	38	$\star$
$p_2 = x_1\overline{x_2}\overline{x_5}$	11	23	34	$\star$
$p_3 = x_1\overline{x_2}\overline{x_3}$	10	30	40	$\star$
$p_4 = \overline{x_2}\overline{x_3}x_4x_5$	0	23	23	$\star$
$p_5 = x_1\overline{x_2}\overline{x_3}x_5$	0	30	30	$\star$
$c_1 = x_1\overline{x_2}\overline{x_4}\overline{x_5}$	11	31	44	p_2
$c_2 = x_1x_2\overline{x_4}\overline{x_5}$	4	31	35	$\star$
$c_3 = x_1\overline{x_3}\overline{x_4}\overline{x_5}$	12	30	42	p_1
$c_4 = x_1x_3\overline{x_4}\overline{x_5}$	5	30	35	$\star$
$c_5 = x_1\overline{x_2}\overline{x_3}\overline{x_5}$	10	30	40	p_2
$c_6 = x_1\overline{x_2}x_3\overline{x_5}$	1	30	31	$\star$
$c_7 = x_1\overline{x_2}x_4\overline{x_5}$	4	31	35	p_2
$c_8 = x_1\overline{x_2}\overline{x_3}\overline{x_4}$	10	38	48	p_3
$c_9 = x_1\overline{x_2}\overline{x_3}x_4$	4	38	42	p_3
$m_1 = x_1\overline{x_2}\overline{x_3}\overline{x_4}\overline{x_5}$	10	38	48	p_1
$m_3 = x_1x_2\overline{x_3}\overline{x_4}\overline{x_5}$	2	38	40	p_1
$m_5 = x_1\overline{x_2}x_3\overline{x_4}\overline{x_5}$	1	38	39	p_1
$m_7 = x_1x_2x_3\overline{x_4}\overline{x_5}$	4	38	42	p_1
$m_9 = x_1\overline{x_2}\overline{x_3}x_4\overline{x_5}$	4	38	44	p_2
$m_{13} = x_1\overline{x_2}x_3x_4\overline{x_5}$	0	38	38	p_2
$m_{17} = x_1\overline{x_2}\overline{x_3}\overline{x_4}x_5$	0	38	38	p_5
$m_{24} = \overline{x_1}\overline{x_2}\overline{x_3}x_4x_5$	0	38	38	p_4
$m_{25} = x_1\overline{x_2}\overline{x_3}x_4x_5$	0	38	38	p_4

Table 2: Implicant Power Costs.

4 Trace driven multi-level minimization

In this section we describe the algorithms developed for the four main technology independent logic optimization procedures to achieve minimum power dissipation for a given trace.

4.1 Trace driven simplification and elimination

The node simplification strategy adopted for multi-level logic optimization is based on performing a two-level logic minimization at each node. Although this is an effective strategy for area minimization, since a local optimization at each node yields a local minimization of the area, it is less clear how simplification should be used for power reduction. A two-level minimization is effective only in reducing the output switching activity ($\mathcal{OSA}$) of the implicants of the function. This may change the fanout of the input variables and has to be carefully managed to avoid reaching a negative result. We simply adopt the logic simplification strategy already existing for area minimization, modified to use the procedure described for heuristic power minimization; alternatives to this naive approach remain an interesting problem for future work. To attempt to climb out of a local minimum, a trace-driven elimination command [17] has been developed using as the figure of merit the change in switching activity under the given trace. Due to the compact representation of traces by a set of vector pairs it is efficient to compute the local trace for the fanin of node n as well as to evaluate its output transition activity.

4.2 Trace driven decomposition

In multi-level logic optimization it is typical to decompose a gate with large fanin into a tree of gates with bounded fanin; this is performed for example during delay optimization or as a pre-process step of technology mapping. For our purposes the decomposition of a complex gate is also used to guide the estimation of the internal switching activity during the logic extraction step (Section 4.3).

Murgai, *et al.* [8] analyze the problem of decomposing a large-fanin gate for minimum switching activity using input transition probabilities (assuming independence between the input signals) and demonstrate that it is equivalent to the problem of finding a minimum weighted binary tree, where the weight of each node is its 1-controllability. An elegant solution to this problem was given by Huffman and can be exactly applied in the zero-delay case for general circuits. A Huffman style algorithm is not exact for a trace-driven decomposition. In fact, the trace driven setting allows us to achieve better decompositions than that obtained by a vanilla implementation of Huffman's algorithm.

We describe the decomposition for an AND gate, and the procedure can be easily dualized for the case of an OR gate. We build a **decomposition matrix** having K rows and columns corresponding to the K input variables of gate g. We use the rows to guarantee that all the variables x_k are "covered" and the columns to choose the next gate in the decomposition. Each column j, $1 \leq j \leq K$ is associated with a variable x_j. For each pair of distinct variables x_i, x_j the entry a_{ij} is a pair of numbers $(\mathcal{F}_\mathcal{T}, \mathcal{Z}_\mathcal{T})$ for the gate $g_{ij} = x_i \cdot x_j$. The first number is the output switching activity of g_{ij} under the trace $\mathcal{T}$. The second counts the number of pairs of successive zero values on the output of g_{ij} under application of $\mathcal{T}$. On each step of the algorithm the minimum entry a_{ij} is selected. An AND gate with inputs corresponding to i and j is created and the rows i and j are deleted from the matrix while a new column for the AND gate is added and the values of its entries are obtained via a local simulation of $\mathcal{T}$.

Let $f = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5$ be the Boolean function for which we must find the best decomposition with respect to the trace $\mathcal{T}$ listed in Table 3. The initial decomposition matrix is as follows:

Input Trace $\mathcal{T}$	Output Traces of relevant gates						
$x_1x_2x_3x_4x_5$	g_1	g_1x_3	g_1x_4	g_1x_5	g_2	g_2x_4	g_2x_5
1 1 1 1 1	1	1	1	1	1	1	1
0 1 0 0 0	0	0	0	0	0	0	0
1 1 1 0 1	1	1	0	1	1	0	1
1 1 0 1 1	1	0	1	1	0	0	0
0 0 1 1 0	0	0	0	0	0	0	0
0 1 1 1 1	0	0	0	0	0	0	0
1 0 1 0 1	0	0	0	0	0	0	0
1 0 1 1 0	0	0	0	0	0	0	0
1 1 0 0 0	1	0	0	0	0	0	0

Table 3: The input trace $\mathcal{T}$ for decomposition example

$$
\begin{array}{c c}
 & \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\
\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} &
\left[\begin{array}{ccccc}
(\infty,0) & (4,3) & (5,2) & (5,3) & (5,2) \\
(4,3) & (\infty,0) & (5,3) & (5,3) & (5,2) \\
(5,2) & (5,3) & (\infty,0) & (5,2) & (5,2) \\
(5,3) & (5,3) & (5,2) & (\infty,0) & (5,3) \\
(5,2) & (5,3) & (5,2) & (5,3) & (\infty,0)
\end{array} \right]
\end{array}
$$

Since the entry $(4,3)$ has the minimum value, we create the gate $g_1 = (x_1, x_2)$; the new matrix is shown in Table 3,

$$
\begin{array}{c c}
 & \begin{array}{cccc} x_3 & x_4 & x_5 & g_1 \end{array} \\
\begin{array}{c} x_3 \\ x_4 \\ x_5 \\ g_1 \end{array} &
\left[\begin{array}{cccc}
(\infty,0) & (5,2) & (5,2) & (3,5) \\
(5,2) & (\infty,0) & (5,3) & (3,5) \\
(5,2) & (5,3) & (\infty,0) & (3,4) \\
(3,5) & (3,5) & (3,4) & (\infty,0)
\end{array} \right]
\end{array}
$$

Three entries, namely (x_3, g_1), (x_4, g_1), and (x_5, g_1), have minimum value of $\mathcal{F}_\mathcal{T}$. A vanilla implementation of Huffman using only the first field would consider all three choices equal. However, in our procedure ties on the first field of an entry are broken by choosing the entry with the largest value of $\mathcal{Z}_\mathcal{T}$. The intuition behind this is that for an AND gate a larger value on the second field implies that the gate output is 0 more often than with a smaller value. This heuristic provides a better upper bound on the total number of switches that may occur for the remainder of the decomposition.

For the example, assume we select $g_2 = (g_1, x_3)$. After updating the decomposition matrix, the last two choices are trivial and produce a final decomposition which has 9 switches under $\mathcal{T}$. The unmodified Huffman algorithm yields a decomposition with 11 switches.

4.3 Trace driven extraction

The problem of logic extraction for low power has been previously addressed in two ways. Roy and Prasad [12] present a kernel extraction algorithm to reduce power dissipation by common sub-expression extraction guided by power values for nodes computed on the given factored form expressions for the nodes. A potential weakness of this approach is that the initial factored forms are not altered, leading to sub-optimum extractions. In [5] an alternate approach is proposed to solve this problem: the power values of the common sub-expressions are computed using the power cost of a node represented using a sum-of-products representation. Since this conforms to the underlying assumptions for traditional algebraic extraction and decomposition algorithms [13, 17], the authors can rely on these algorithms to extract the set of all single cube intersections and kernel intersections among the network nodes in order to choose the sub-expression having the maximum power reduction. The shortcoming of this approach is that after each extraction the switching activity of all the affected internal nodes must be re-estimated by computing the global BDD and the signal probabilities of each function. This operation is generally time consuming and is sped-up by using an approximation for the signal probabilities on the immediate fanin of a node [5].

Our technique is similar to the previous approaches in that a value is associated with each sub-expression to denote the power saving obtained if the logic extraction is performed. On the other hand, it differs from the previous approaches in three aspects. First, since the methodology is trace driven, signal probabilities are not employed. Instead, exact switching activity for any node is computed for the given trace by performing a local logic simulation. Since the global function of any node does not change during extraction a complete simulation of all the nodes on the trace is performed once. All subsequent simulations on sub-expressions are obtained by evaluating the function of the sub-expression on the known fanin values. Second, neither a factored form nor a sum-of-products representation is used to estimate the switching activity inside a node. Instead, each node is decomposed on the fly into gates with fanin two, by using the procedure outlined in Section 4.2. This represents a structure that is better correlated to the final network after technology mapping. Finally, our algorithm derives directly from the algorithm of Rajski and Vasudevamurthy [11] which is widely considered the most efficient method for Boolean extraction.

The basic objects used in extraction are single-cube divisors having exactly two literals, double cube divisors and their complements. The motivation for using objects of size two is that they ensure that the operations have polynomial running times, while they can be used to find single-cube divisors of arbitrary size and multiple-cube divisors. The complete definitions of double-cube divisor, set and subset of double-cube divisors, base of a double-cube divisor, single-cube divisor and its coincidence are found in [11]. We illustrate some of the terms for the benefit of the reader. Given four cubes $q_1 = x_1x_2, q_2 = x_1x_3, q_3 = x_1x_2x_3$ and $q_4 = x_1x_3x_4$, the set of two-literal single-cube divisors contains x_1x_2 which has coincidence 2 and x_1x_3 which has coincidence 3. (Coincidence K means that if d is extracted to create a new node n_d in the network then the fanout of this node will be K). Given a Boolean expression $f = x_1x_4x_5 + x_1x_6 + x_2x_3x_4x_5 + x_2x_3x_6$ the set of all its double cube divisors is $D(f) = \{x_6 + x_4x_5, x_1 + x_2x_3, x_1x_4x_5 + x_2x_3x_6, x_1x_6 + x_2x_3x_4x_5\}$. This set can be partitioned in subsets denoted by the symbol $D_{K,H,K+H}$ where K is the number of literals in the first cube and H the number of literals in the second cube. Hence, $x_6 + x_4x_5 \in D_{1,2,3}$, $x_1 + x_2x_3 \in D_{1,2,3}$, $x_1x_4x_5 + x_2x_3x_6 \in D_{3,3,6}$ and $x_1x_6 + x_2x_3x_4x_5 \in D_{2,4,6}$. Finally $x_6 + x_4x_5$ has two bases, namely x_1 and x_2x_3; $x_1 + x_2x_3$ has two bases, namely x_6 and x_4x_5 and the remaining two double-cube divisors have empty base. The greedy algorithm we propose is an extension of the Rajski and Vasudevamurthy extraction algorithm [11], modified to use cost values denoting the power dissipation due to single-cube and double-cube divisors.

The **power value** of a two literal single cube divisor $d = x_1x_2$ with coincidence equal to $K(d)$ is given by:

$$PV_{\mathcal{T}}(d) = (K(d) - 1) \cdot [\mathcal{F}_{\mathcal{T}}^{x_1} + \mathcal{F}_{\mathcal{T}}^{x_2}] - K(d) \cdot \mathcal{F}_{\mathcal{T}}^{d} + \mathcal{FISAD}$$

To understand this equation, suppose that extraction of d has been performed creating a new node n_d. The first term of the equation accounts for the power saving related to the variables x_1, x_2 which now feeds only node n_d instead of $K(d)$ nodes. The second term represents the switching activity of the output variable of n_d multiplied by its fanout. The third term ("fanout internal switching activity difference") denotes the change in power dissipation for each fanout n_j of n_d due to extraction.

Consider a double cube divisor $d \in D_{K,H,K+H}$ having the following characteristics:

- $d = x_1 \cdot x_2 \cdot \ldots \cdot x_K + y_1 \cdot y_2 \cdot \ldots \cdot y_H$;
- The node n_d which can be extracted has fanout $\mathcal{FO}$ in $\mathcal{N}$; e.g. d divides $\mathcal{FO}$ expressions associated two the nodes $n_1, n_2 \ldots n_{\mathcal{FO}}$;
- d has B bases $b_1 = z_1 \cdot z_2 \cdot \ldots \cdot z_{M_{b_1}}, \ldots b_B = z_1 \cdot z_2 \cdot \ldots \cdot z_{M_{b_2}}$.

Input Trace $\mathcal{T}$ $x_1x_2x_3x_4x_5x_6x_7x_8x_9$	**Div. Output Traces** $\mathcal{F}_{\mathcal{T}}^{d_1}$	$\mathcal{F}_{\mathcal{T}}^{d_2}$	$\mathcal{F}_{\mathcal{T}}^{d_3}$
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 0 1 0 0 0 1	1	0	0
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 0 1 0 0 0 1	1	0	0
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 1 0 0 1 1 0	1	0	0
0 1 0 0 1 0 0 0 0	0	0	0
1 1 1 1 0 0 1 1 0	1	0	0
0 1 0 0 1 0 0 0 0	0	0	0
1 1 1 1 0 0 1 0 1	1	0	0
0 1 0 0 1 0 0 1 1	0	0	0
1 1 1 1 0 0 1 1 1	1	0	0
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 1 0 0 1 1 1	1	0	0
0 1 0 0 1 0 0 1 1	0	0	0
1 1 0 0 1 0 0 0 1	1	0	0
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 0 1 0 0 0 1	1	0	0
0 1 0 0 1 0 0 0 1	0	0	0
1 1 0 0 1 0 0 0 1	1	0	0

Table 4: Traces for extraction example

The **power value** of d under the trace $\mathcal{T}$ is given by:

$$\begin{aligned} PV_{\mathcal{T}}(d) &= (\mathcal{FO}(d) - 1) \cdot \left[\sum_{k=1}^{K} (\mathcal{F}_{\mathcal{T}}^{x_k}) + \sum_{k=1}^{H} (\mathcal{F}_{\mathcal{T}}^{y_k}) \right] + \\ &- \mathcal{FO}(d) \cdot \mathcal{F}_{\mathcal{T}}^{d} + \sum_{i=1}^{B} \left[\sum_{j=1}^{M_{b_i}} (\mathcal{F}_{\mathcal{T}}^{z_j}) \right] + \mathcal{FISAD} + \mathcal{CG} \end{aligned}$$

As before, to understand this equation, suppose that the extraction of d has been performed creating a new node n_d. The first term accounts for the power saving related to the input variables $x_1, \ldots, x_K$ and $y_1, \ldots, y_H$ which now feed just the node n_d instead of $\mathcal{FO}$ nodes. The second term represents the switching activity of the output variable of n_d multiplied by its fanout. The third term is the power

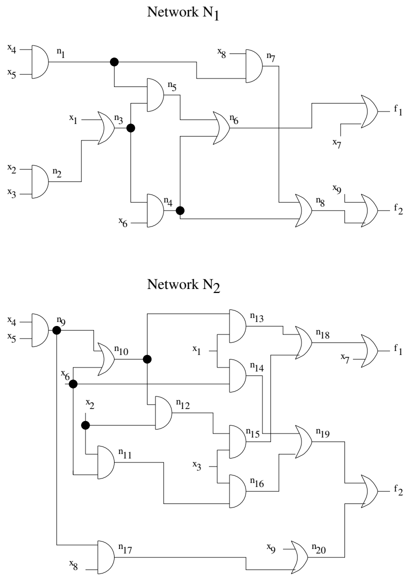


Figure 3: Two alternative extraction results

saving due to the occurrence of each variable z_i in a base of d. Consider a fanout node n_g before the extraction of d and suppose that two cubes of its function g are respectively $(x_1 \cdot x_2 \cdot \ldots \cdot x_K) \cdot b_1$ and $(y_1 \cdot y_2 \cdot \ldots \cdot y_H) \cdot b_1$, with $b_1 = z_1 z_2$. Obviously b_1 is a base of d and, if d is chosen to be extracted, the previous function will be replaced by $d \cdot b_1$. As a consequence, the node n_g will see the input switching activity decreased by a quantity equal to $\mathcal{F}_{\mathcal{T}}^{z_1} + \mathcal{F}_{\mathcal{T}}^{z_2}$. The last term ("complementary gain")is non-zero only if $d \in D_{1,1,2}$, *e.g.* if $d = x_1 + y_1$, where x_1 and y_1 are two distinct literals. In this case, the complement of d is a single cube divisor and the power value of $\overline{d}$ is added to $PV_{\mathcal{T}}(d)$. The term $\mathcal{FISAD}$ is the sum of $P_{orig}^{n_j} - P_{new}^{n_j}$ for each fanout n_j of d. $P_{orig}^{n_j}$ is the power dissipation before extraction, $P_{new}^{n_j}$ the power dissipation after extraction. The representation of traces by a set of vector pairs permits this computation to be performed very efficiently since only a local evaluation of the node functions is involved. Hence it is very feasible to estimate $P_{new}^{n_j}$ for each fanout n_j each time a divisor is evaluated.

Consider the Boolean Network which has 9 inputs, 2 outputs and 2 internal nodes f_1 and f_2.

$$
\begin{aligned}
f_1 &= x_1x_4x_5 + x_1x_6 + x_2x_3x_4x_5 + x_2x_3x_6 + x_7 \\
f_2 &= x_1x_6 + x_2x_3x_6 + x_4x_5x_8 + x_9
\end{aligned}
$$

Performing extraction using the area minimal option described in the algorithm of Rajski and Vasudevamurthy [11], followed by decomposition into 2-input AND and OR gates yields $\mathcal{N}_1$ in Figure 3. Now, suppose that we want to perform the best extraction for low power on $\mathcal{N}$ with respect to the input trace $\mathcal{T}$ specified in the same Figure. First, the power values of all the two-literal single-cube divisors and two-cubes divisors are computed. For the example, there are just two double-cube divisors $d_1 = x_1 + x_2 \cdot x_3$ and $d_2 = x_6 + x_4 \cdot x_5$, and one single divisor $d_3 = x_4 \cdot x_5$ which have positive power value. Using the equations presented in this section,

$$
\begin{aligned}
PV_{\mathcal{T}}(d_1) &= (3-1) \cdot [19+0+6] - 3 \cdot 19 + [10+10] + 0 = 13 \\
PV_{\mathcal{T}}(d_2) &= (2-1) \cdot [0+10+10] - 2 \cdot 0 + [19+0+6] + 0 = 45 \\
PV_{\mathcal{T}}(d_3) &= (2-1) \cdot [10+10] - 2 \cdot 0 = 20
\end{aligned}
$$

d_2 is extracted (preferred over d_1) to obtain:

$$
\begin{aligned}
d_2 &= x_6 + x_4x_5 \\
f_1 &= d_2x_1 + d_2x_2x_3 + x_7 \\
f_2 &= x_1x_6 + x_2x_3x_6 + x_4x_5x_8 + x_9
\end{aligned}
$$

Trace-driven decomposition algorithm yields $\mathcal{N}_2$ in Figure 3. While $\mathcal{N}_2$ has 14 gates, $\mathcal{N}_1$ has 11, but there is a 14% reduction in power since:

$$
\begin{aligned}
\mathcal{F}_{\mathcal{T}}^{\mathcal{N}_1} &= 65 + 10 + 19 \cdot 2 + 6 = 119 \\
\mathcal{F}_{\mathcal{T}}^{\mathcal{N}_2} &= 65 + 10 + 19 + 6 + 2 = 102
\end{aligned}
$$

5 Experimental Results

The first set of experiments (data not reported in this paper for lack of space) compared the trace-driven two-level heuristic and exact minimization algorithm against *espresso*. The traces for these examples were artificially generated as follows: Given a function f with p primes two successive vectors were created for each pair of primes such that the first vector is contained in the first prime and the second is contained in the second prime. The trace was intentionally biased towards obtaining poor results with *espresso* to determine what reductions may be achieved by this step alone. For the two-level case we assume that the output capacitance load on each implicant is equal to one unit, while the input capacitance is one unit for each literal. A more realistic model is not used here since this model is used for multi-level logic simplification where the inputs of a node are the outputs of other nodes. The following conclusions were inferred:

1. Reductions of up to 95% can be achieved for the quantity $\mathcal{OSA}_{\mathcal{T}}$ (*c.f.* Section 3) by selection of the appropriate implicants for minimal switching power compared to *espresso*.
2. The overall power reduction $W_{\mathcal{T}}(Q)$ is limited by the input switching activity $\mathcal{ISA}_{\mathcal{T}}$ which is a function only of the given trace. Two-level minimization cannot impact this significantly, since the number of literals varies little between *espresso* and *elp*. Thus, **two-level minimization can provide only a net incremental reduction in power once the trace has been determined**.

The next experiment was performed on 20 FSMs from the *ISCAS89 benchmark suite*. For these FSMs, given a reset state, 10000 random primary input vectors were applied. The trace of the primary input $\mathcal{T}^{PI}$ and the corresponding trace $\mathcal{T}^{L}$ of the latch values, obtained by sequential logic simulation, were combined together to serve as the trace for power minimization. The average power reduction is 16% before technology mapping, and 14% after technology mapping (Table 5).

We also report the fraction of power dissipation due to the transition activity related to primary inputs and latches (column marked "I/O Power Fraction"). These values were obtained by simulating the circuit after technology independent optimization with *script.rugged*. These results prove an important point on logic synthesis for low power: **the I/O power provides an upper bound on the power reduction that can be achieved**, because combinational logic synthesis only reduces power consumed in the internal nodes of the circuit. For example, consider the data reported for FSM *s298* in Table 5. A final power reduction of 25% has been obtained with the *low power* script, even though in the initial circuit 40% of the total power P was due just to primary input and latch switching activity. We have:

$$P \cdot (0.40 + x \cdot 0.60) = P \cdot (1 - 0.25) \quad \Rightarrow x = 0.35/0.60 \; = 0.58$$

Hence, for this example, trace driven logic synthesis provided a reduction of the power due to internal transition activity of 42%. From Table 5, the average reduction of internal power is 25%.

			I/O	tech. independent		mapp. networks	
		Gate	Power	**ratio: LP/rugg.**		**ratio: LP/rugg.**	
FSM	I/O/L	#	Fract.	Area	Power	Area	Power
s27	4/1/3	9	0.46	0.88	0.95	0.97	0.95
s208	10/1/8	70	0.41	0.96	0.61	1.09	0.89
s298	3/6/14	111	0.40	1.00	0.72	0.99	0.75
s344	9/11/15	123	0.23	0.97	0.83	1.06	0.91
s349	9/11/15	122	0.22	1.03	0.84	1.07	0.89
s382	3/6/21	142	0.37	1.13	0.78	1.07	0.76
s386	7/7/6	108	0.26	1.08	0.76	1.04	0.78
s400	3/6/21	137	0.36	1.37	0.86	1.20	0.84
s420	18/1/16	143	0.42	1.04	1.00	1.12	0.84
s444	3/6/21	136	0.35	1.21	0.77	1.12	0.78
s510	19/7/6	255	0.24	0.99	0.87	1.04	0.91
s526	3/6/21	171	0.46	1.36	0.98	1.23	1.07
s526n	3/6/21	191	0.40	1.20	0.92	1.15	0.96
s641	35/23/17	156	0.33	0.90	0.76	0.93	0.83
s713	35/23/17	157	0.33	0.81	0.73	0.91	0.78
s820	18/19/5	298	0.39	1.05	0.90	1.08	0.93
s832	18/19/5	288	0.38	1.11	0.92	1.13	0.99
s838	34/1/32	355	0.43	1.27	0.84	1.11	0.66
s1488	8/19/6	641	0.20	0.99	0.82	1.00	0.81
s1494	8/19/6	641	0.20	1.00	0.84	1.01	0.85

Table 5: Results on *ISCAS89* FSMs

6 Conclusions and Future Work

This paper has proposed a new trace drive methodology for logic synthesis which captures and exploits the correlations that exist between signal values in an application specific design. The idea has been applied to the minimization of the switching activity in the combinational logic portion of finite state machines. In this paper we have focussed on technology independent optimizations. The extension to technology mapping, based on applying trace driven optimization to the methods proposed in [3] remains for the future. Our results have also indicated that additional gains can only be realized by changing the sequential behavior of the FSMs to impact the I/O switching activity. The application of encoding and re-encoding techniques [4] to reduce I/O activity is a promising avenue for exploration.

Acknowledgments

The authors would like to thank Alberto Ferrari and Tiziano Villa for their support and useful discussions.

References

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

[2] O. Coudert. On Solving Covering Problems. In *Proc. of the Design Automation Conf.*, pages 197–202, June 1996.

[3] O. Coudert and R. Haddad. Integrated Resynthesis for Low Power. In *ISLPED Digest of Technical Papers*, pages 169–174, August 1996.

[4] G. D. Hachel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi. Re-Encoding Sequential Circuits to Reduce Power Dissipation. In *Proceedings of the Design Automation Conference*, pages 70–73, June 1994.

[5] S. Iman and M. Pedram. Logic Extraction and Factorization for Low Power. In *Proceedings of the Design Automation Conference*, pages 248–253, June 1995.

[6] S. Iman and M. Pedram. Two-Level Logic Minimization for Low Power. In *Proceedings of the International Conference on Computer-Aided Design*, pages 433–438, November 1995.

[7] R. Marculescu, D. Marculescu, and M. Pedram. Efficient Power Estimation for Higly Correlated Input Streams. In *Proceedings of the Design Automation Conference*, pages 628–634, June 1994.

[8] R. Murgai, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Decomposition for Minimum Transition Activity. In *Proceedings of the Low Power Workshop - Napa Valley*, April 1994.

[9] F. N. Najm. A Survey of Power Estimation Techniques in VLSI Circuits. *IEEE Transactions on VLSI Systems*, 2:446–455, 1994.

[10] F. N. Najm. Feedback, Correlation, and Delay Concerns in the Power Estimation of VLSI Circuits. In *Proceedings of the* 32^{th} *Design Automation Conference*, pages 612–617, June 1995.

[11] J. Rajski and J. Vasudevamurthy. The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Espression. *IEEE Transactions on Computer-Aided Design*, 11:778–793, 1992.

[12] K. Roy and S.C. Prasad. Circuit Activity Based Logic Synthesis for Low Power Reliable Operations. *IEEE Transactions on VLSI Systems*, 1:503–513, 1993.

[13] Richard L. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989. Memorandum No. UCB/ERL M89/49.

[14] C-Y. Tsui, R. Marculescu, D. Marculescu, and M. Pedram. Improving the Efficiency of Power Simulators by Input Vector Compaction. In *Proceedings of the Design Automation Conference*, pages 165–168, June 1996.

[15] C-Y. Tsui, M. Pedram, and A. M. Despain. Efficient Estimation of Dynamic Power Consumption under a Real Delay Model. In *Proceedings of the International Conference on Computer-Aided Design*, pages 224–228, Nov 1993.

[16] C-Y. Tsui, M. Pedram, and A. M. Despain. Exact and Approximate Methods for Calculating Signal and Transition Probabilities in FSMs. In *Proceedings of the* 29^{th} *Design Automation Conference*, pages 18–23, June 1994.

[17] A. Wang. *Algorithms for Multi-Level Logic Optimization*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.