TRACE DRIVEN LOGIC SYNTHESIS— APPLICATION TO POWER MINIMIZATION

by

Luca P. Carloni, Patrick C. McGeer, Alexander Saldanha,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M96/62

21 October 1996

TRACE DRIVEN LOGIC SYNTHESIS— APPLICATION TO POWER MINIMIZATION

by

Luca P. Carloni, Patrick C. McGeer, Alexander Saldanha,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M96/62

21 October 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Trace Driven Logic Synthesis - Application to Power Minimization

Luca P. Carloni Patrick C. McGeer[†]

Alexander Saldanha[†] Alberto L. Sangiovanni-Vincentelli

Department of EECS
University of California at Berkeley
Cory Hall, Berkeley, CA 94720
[†] Cadence Berkeley Laboratories,
1919 Addison Street, Ste. 303
Berkeley CA 94704-1144

Abstract

A trace driven methodology for logic synthesis and optimization is proposed in this paper. 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. Algorithms are devised to perform trace driven two-level and multi-level logic optimization to minimize switching power. We demonstrate the advantages this approach provides over traditional logic synthesis by showing large reductions in switching power on a set of benchmark circuits.

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 completely 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. 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 to a large extent by the switching activity of 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.

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 is most appropriate when applied to designs that are specific to a chosen application. This paper is devoted to the case where the cost function is to minimize the switching activity of the circuit under the sequence. The approach is novel since it is the first to propose synthesis and optimization at the micro-architecture (logic) level tuned to an application specific stimulus. This paper focuses on the development of two-level and multi-level algorithms for combinational logic synthesis to minimize the switching activity under a user provided trace set. The approach is applied to sequential circuits by performing the optimizations on the combinational logic between the latches.

The motivation and application of the proposed methodology to power minimization at the logic level is justified in two steps: (1) Why trace driven logic optimization is essential to obtain useful power reductions in logic circuits, and, (2) How trace driven synthesis is used in logic synthesis and what modifications of existing algorithms and design flows are required to accommodate the proposed methodology. Each of these issues are illustrated in the next two sub-sections.

1.1 Trace driven logic optimization for low power

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 Table 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. Thus, neither implementation is always preferred over the other with respect to switching

	Trace $\mathcal{T}_1$	Values		Trace $\mathcal{T}_2$	Values	
vector	$x_1x_2x_3$	g_1	g_2	$x_1x_2x_3$	g_1	g_2
v_1	1 1 1	1	0	1 1 0	0	0
v_2	1 0 0	0	0	1 0 1	0	0
v_3	1 1 1	1	0	1 1 0	0	0
v_4	0 0 0	0	0	0 0 1	0	0
v_5	1 1 1	1	0	1 1 0	0	0
v_6	0 0 0	0	0	0 0 1	0	0
v_7	1 1 1	1	0	1 1 0	0	0
v_8	1 0 0	0	0	1 0 1	0	0
v_9	1 1 1	1	0	1 1 0	0	0
v_{10}	0 0 0	0	0	0 0 1	0	0
v_{11}	0 1 1	1	1	0 1 0	0	0

Table 1: Example illustrating trace driven synthesis. Note that $g_1 = x_2 \cdot x_3$ and $g_2 = \overline{x_1} \cdot x_2 \cdot x_3$

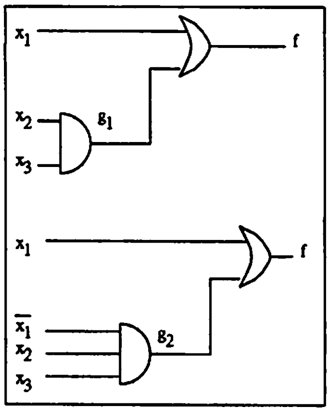

Figure 1: The two implementations of $f = x_1 + x_2 \cdot x_3$.

activity, and the desirable implementation can only be differentiated using trace driven synthesis.

Note that the previous approaches described by Najm [6] or Iman and Pedram [4] using signal and/or transition probabilities calculated for each of the given traces $\mathcal{T}_1$ and $\mathcal{T}_2$ always causes the first implementation to be preferred over the second.

It is a widely accepted fact 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 forcing this conclusion is the variation in the estimated power using probabilistic or statistical techniques from the power dissipated under the actual vectors applied to the circuit. Specifically the spatial correlations between the values on input signals within a vector as well as the spatio-temporal correlations between signals across vectors cannot be captured by existing probilistic and statistical methods. 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 substantially more room for significant reduction in the power dissipation due to trace driven optimization algorithms.

1.2 Trace driven methodology

Having demonstrated the potential improvement available from trace driven logic synthesis, the next major question to address is the generation of the input trace sets to use during logic synthesis and optimization.

If the intended design is to be used to perform a specific task as part of a larger system the trace set is composed of the sequence of values that appear on the input signals of the design. This sequence is obtained by performing logic simulation of the part within the expected environment of application. This case covers most *application specific* integrated circuits (ASIC's). An interesting sub-class of these designs are controllers which are implemented as finite state machines (FSM's). Assume that the sequence of values that appear on the primary inputs of the FSM is hard to characterize. In such cases, a relevant and useful trace set may be obtained by performing logic simulation on the FSM using random values on the primary inputs. The sequence of values of the state registers will convey the important correlations between the state variables of the controller. In fact, our experimental results for FSM's are obtained by generating characteristic sequences using this technique.

As discussed later in the sequel, 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 the naive simulation based procedure outlined above 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 represented as a set of vector pairs with a frequency count indicating the number of occurrence of each pair in the trace set; thus recurring vector pairs and sequences are automatically compacted, (2) the size of the trace set only impacts the running time used in simulations 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.

1.3 Related work

We provide a brief summary of previous work in low power synthesis to compare and contrast the contributions of this paper.

The early approaches to power estimation were based on propagating switching probabilities from inputs through a combinational logic netlist. Given the inaccuracy due to logic re-convergence, the technique has been modified to use pairwise correlations, spatial correlations and temporal correlations approximated by time-homogeneous Markov chains. Although the authors have claimed that these approximate techniques yield only small losses in accuracy, none of these approaches have been feasibly applied in logic synthesis procedures. Power estimation for sequential logic circuits have been mostly based on calculation of the steady state transition probabilities using the Chapman-Kolmogorov equations, and the majority of approaches are unable to handle circuits with larger than a handful of memory elements. Similar to the combinational logic case, none of the techniques have as yet been feasibly adapted in sequential logic synthesis procedures.

In the logic synthesis domain greedy heuristic techniques for simplification using don't cares, decomposition and factorization have been proposed. None of these techniques presumably provide a guarantee of realistic power reduction since the techniques are usually based on naive assumptions about the switching activity of individual nodes. The authors report reductions in the 5%-15% range on benchmark circuits assuming completely random traces. No technique until now have been presented to address the trace driven logic synthesis. We proceed to the formulation of this problem in the next section.

2 Trace driven logic optimization problem

The problem of trace driven logic optimization for minimum switching activity may be stated as follows: 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.

There are several significant aspects to note in the statement of the problem. 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. We use an accurate capacitance model during the estimation of power dissipated due to switching activity. In particular 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.

Some basic notation and definitions are introduced to facilitate the presentation of trace driven logic synthesis.

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. Note that every input either has value 0 or 1 in each vector in a trace.

The switching activity of a Boolean function f for a given trace $\mathcal{T}$ can be computed by simply determining the value of f under each vector in $\mathcal{T}$. A switch occurs when the value of f changes from 0 to 1 or 1 to 0. The switching activity of f under $\mathcal{T}$ may also be 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 $\mathcal{F}_{\mathcal{T}}(v_i, v_j)$.

As an example, consider the three traces shown in Table 2. We have:

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$
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 2: Example traces

$$\mathcal{P}_{\mathcal{T}_1} = \{(0000, 0001), (0001, 0010), (0010, 0011), (0011, 0100), (0100, 0101),$$

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

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

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

For the example, we have $\mathcal{F}_{\mathcal{T}_1}^{g} = 2$, $\mathcal{F}_{\mathcal{T}_2}^{g} = 9$, and $\mathcal{F}_{\mathcal{T}_3}^{g} = 9$, 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 that are described in the sequel.

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 in Table 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:

$$\begin{aligned}
\mathcal{OSA}_{\mathcal{T}}(q) &= 0.5 \cdot C_{load} \cdot V_{dd}^2 \cdot \mathcal{F}_{\mathcal{T}_1}^{q} & (2) \\
\mathcal{ISA}_{\mathcal{T}}(q) &= \Sigma_{x_k \in q} 0.5 \cdot C_{x_k} \cdot V_{dd}^2 \cdot \mathcal{F}_{\mathcal{T}_1}^{x_k} & (3)
\end{aligned}$$

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 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 $\mathcal{Q}^*_{\mathcal{T}}$.

Following [1, 9], 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. In fact, the assumption [1] "*an implicant costs no more than any implicant which it contains*" is not satisfied and there is no guarantee that there exists a minimum cover consisting only of prime implicants. Since it is practically infeasible to solve the problem considering the set $\mathcal{I}$ of all the implicants [2], one wishes to identify a set $\mathcal{I}^*_{\mathcal{T}} \subseteq \mathcal{I}$ of implicants sufficient to find a minimum cover. $\mathcal{I}^*_{\mathcal{T}}$ is termed the set of candidate implicants.

The following definitions identify which implicants are elements of $\mathcal{I}^*_{\mathcal{T}}$. 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 $\mathcal{I}^*_{\mathcal{T}}$ 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.

Recall that an implicant q of f is a cube which has empty intersection with the vertices of the *off-set* of f, while a prime implicant p is an implicant which is not contained by any implicant. A prime implicant p is **essential** if it contains at least one minterm which is not contained in any other prime. Given f :

[1] See [9], assumption 2.3.1 at page 14.

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

$B^N \rightarrow B$, 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 $\mathcal{I}^\star$. 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.

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.

Let $f : B^5 \rightarrow B$ be the completely-specified Boolean function illustrated in Figure 2: the *on-set* of f is $\{m_1, m_3, m_5, m_7, m_9, m_{13}, m_{17}, m_{24}, m_{25}\}$ The set of prime implicants is $\mathcal{PI}(f) = \{p_1, p_2, p_3, p_4\}$. We perform two-level logic optimizations for minimum switching power with respect to the input trace $\mathcal{T}$. Table 3 reports the input vectors for this trace, showing the correspondences with the vertices of the Boolean space B^5 as they are defined in Figure 2. Table 4 reports all the implicants of the function f and their power costs which can be

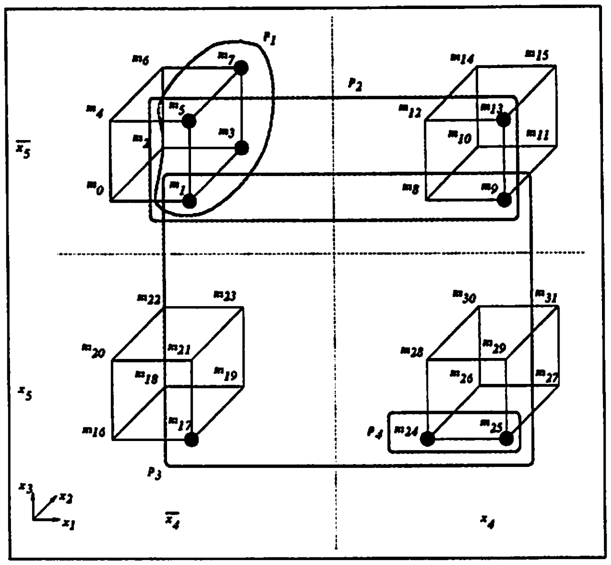

Figure 2: The function $f = x_1\overline{x_4x_5} + x_1\overline{x_2}x_5 + x_1\overline{x_2x_3} + \overline{x_2x_3}x_4x_5$

Input Trace $\mathcal{T}$		
vector	$x_1x_2x_3x_4x_5$	vertex
v_1	1 0 1 0 0	m_5
v_2	0 1 1 0 0	m_6
v_3	0 0 1 0 0	m_4
v_4	1 1 1 0 0	m_7
v_5	1 1 0 0 0	m_3
v_6	1 0 0 1 0	m_9
v_7	1 0 0 0 0	m_1
v_8	0 0 0 0 0	m_0
v_9	1 0 0 0 0	m_1
v_{10}	0 1 1 1 0	m_{14}
v_{11}	1 1 1 0 0	m_7
v_{12}	0 1 1 0 0	m_6
v_{13}	1 0 0 0 0	m_1
v_{14}	0 0 0 0 0	m_0
v_{15}	0 0 1 0 0	m_4
v_{16}	1 0 0 0 0	m_1
v_{17}	0 0 0 0 0	m_0
v_{18}	1 1 0 1 0	m_{11}
v_{19}	0 1 1 1 0	m_{14}
v_{20}	1 0 0 0 0	m_1
v_{21}	1 0 0 1 0	m_9
v_{22}	0 0 0 0 0	m_0

Table 3: Input trace $\mathcal{T}$.

obtained applying equation (1). The column labeled with $\mathcal{I}^\star$ reports a $\star$ if the implicant q is a candidate implicant, otherwise a candidate implicant which dominates q is referred to. A set of candidate implicants which has cardinality equal to 8 is obtained using Theorem 3.1 and 3.2.

$$\mathcal{I}^\star = \{p_1, p_2, p_3, p_4, p_5, c_2, c_4, c_6\}$$

These implicants are the columns of the covering matrix, while the rows are the elements of $\mathcal{M}(f)$. The solution of the unate covering problem is the minimum cover $\mathcal{Q}^\star = \{c_2, p_2, p_4, p_5\}$ having power cost $W_\mathcal{T}(\mathcal{Q}^\star) = 122$. Notice that the area minimum cover is $\mathcal{Q} = \{p_1, p_2, p_3, p_4\}$ which has power cost equal to $W_\mathcal{T}(\mathcal{Q}) = 135$. Hence, we have obtained a 10% power saving.

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

Table 4: Implicant Power Costs.

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 is naturally the introduction of trace based weights for the implicants of f. This implies an important fact which contradicts the original *Espresso* philosophy: 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* proceeds entering the typical *Espresso* loop: the first step is *reduce* which attempts to move the current solution out of a local minimum. This routine is not changed in *Elp*.

Elp differs significantly from *Espresso* in the next step, called *expand*. 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 has a smaller power cost. As before, this step is still characterized by the fact that the cardinality of the cover does not increase. While both algorithms attempt to maximally decrease the number of implicants in the current cover, *Elp* leaves a non-prime implicant untouched if there is no a prime with a smaller power cost that contains it.

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.

As sample of experiments discussed in Section 5 show the effectiveness of *Elp* in comparison with the exact algorithm.

4 Trace driven multi-level minimization

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

4.1 Trace driven node simplification

The node simplification strategy adopted for multi-level logic optimization is based on performing a two-level logic minimization at each node. Note that 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. As already indicated a two-level minimization is effective only in reducing the output switching activity ($\mathcal{OSA}$) of the implicants of the function. This is achieved by changing the fanout of the input variables. These effects have to be carefully managed to avoid reaching a negative result at the end of the procedure. In this paper we simply adopt the logic simplification strategy already existing for area minimization (of course modified to use the procedure described for heuristic power minimization), and this remains an interesting problem for future work.

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).

We assume that the two-level function at each node n of the network has been simplified for minimum switching activity using the procedure of the previous section. The decomposition starts with the creation of an AND gate for each cube and the replacement of the original node n with an OR gate connected to the AND gates. Generally this results in gates with large fanin. Consequently, we want to find the decomposition of a large-fanin gate into 2-input gates that minimizes the switching power for the given trace.

Murgai, *et al.* [5] analyze the problem of decomposing a large-fanin gate for minimum switching activity using transition probabilities of the input variables (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. First we build the **decomposition matrix** $\mathcal{A}$ having K rows and K 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 build the best *AND* gates. 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 $\mathcal{T}$.

The algorithm is presented in Figure 3 and the extensions with respect to the original Huffman algorithm are illustrated on the example below.

Input Trace $\mathcal{T}$		**Output Traces of relevant gates**								
vector	$x_1x_2x_3x_4x_5$	g_1	g_1x_3	g_1x_4	g_1x_5	g_2	g_2x_4	g_2x_5	g_3	g_4
v_1	1 1 1 1 1	1	1	1	1	1	1	1	1	1
v_2	0 1 0 0 0	0	0	0	0	0	0	0	0	0
v_3	1 1 1 0 1	1	1	0	1	1	0	1	0	0
v_4	1 1 0 1 1	1	0	1	1	0	0	0	0	0
v_5	0 0 1 1 0	0	0	0	0	0	0	0	0	0
v_6	0 1 1 1 1	0	0	0	0	0	0	0	0	0
v_7	1 0 1 0 1	0	0	0	0	0	0	0	0	0
v_8	1 0 1 1 0	0	0	0	0	0	0	0	0	0
v_9	1 1 0 0 0	1	0	0	0	0	0	0	0	0

Table 5: The input trace $\mathcal{T}$ for decomposition example

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 5.

The initial decomposition matrix is as follows:

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

Since the entry $(4,3)$ for the variable pair (x_1, x_2) has the minimum value for $\mathcal{F}_\mathcal{T}$, we create the gate $g_1 = (x_1, x_2)$ while deleting the corresponding rows and columns. After computing the output values for this gate under $\mathcal{T}$, as illustrated in Table 5, a new row and column are added to the matrix.

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

There are three choices which have the minimum $\mathcal{F}_\mathcal{T}$ values, namely (x_3, g_1) and (x_4, g_1) with entries $(3,5)$ and (x_5, g_1) with entry $(3,4)$.

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, either of the first two choices may be made. Assume we select $g_2 = (g_1, x_3)$. The decomposition matrix is updated as follows:

$$\begin{array}{c c} & \begin{array}{ccc} x_4 & x_5 & g_2 \end{array} \\ \begin{array}{c} x_4 \\ x_5 \\ g_2 \end{array} & \left[\begin{array}{ccc} (\infty,0) & (5,3) & (1,7) \\ (5,3) & (\infty,0) & (3,5) \\ (1,7) & (3,5) & (\infty,0) \end{array}\right] \end{array}$$

The last two choices are trivial and produce the final decomposition which has 9 switches under $\mathcal{T}$. This decomposition happens to be the optimum decomposition for the given trace, but note that the algorithm is not exact in general.

$$\begin{aligned} g_1 &= x_1 \cdot x_2 \rightarrow \mathcal{F}_\mathcal{T}^g = 4 \\ g_2 &= g_1 \cdot x_3 \rightarrow \mathcal{F}_\mathcal{T}^g = 3 \\ g_3 &= g_2 \cdot x_4 \rightarrow \mathcal{F}_\mathcal{T}^g = 1 \\ g_4 &= g_3 \cdot x_5 \rightarrow \mathcal{F}_\mathcal{T}^g = 1 \end{aligned}$$

4.3 Trace driven extraction

The problem of logic extraction for low power has been addressed in the past in a couple of different ways. In [8], a kernel extraction algorithm to generate a network with lower power dissipation is presented. The algorithm performs 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 [3] 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 [9, 10], 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

```
Decompose_Gate($g, \mathcal{T}$)
{
    /* Prepare matrix $\mathcal{A}$ computing its elements $a_{ij} = (\mathcal{F}_{\mathcal{T}}^{g_{ij}}, \mathcal{Z}_{\mathcal{T}}^{g_{ij}})$ */
    $\mathcal{A} \leftarrow$ Build_Decomposition_Matrix($g, \mathcal{T}$)
    while ($\mathcal{A}$ is not an empty matrix) {
        /* Pick up element $(i, j)$ having lowest $\mathcal{F}_{\mathcal{T}}^{g_{ij}}$ */
        /* breaking ties choosing the highest $\mathcal{Z}_{\mathcal{T}}^{g_{ij}}$ */
        $(i, j) \leftarrow$ Choose_Best_Element($\mathcal{A}$)
        /* Insert a new 2-AND gate $g_{ij}$ with output $q_{ij} = x_i \cdot xj$ */
        $q_{ij} \leftarrow$ New_And_Gate($\mathcal{N}, i, j$)
        /* Delete rows $i$ and $j$ from $\mathcal{A}$ */
        $\mathcal{A} \leftarrow$ Delete_Row($\mathcal{A}, i$)
        $\mathcal{A} \leftarrow$ Delete_Row($\mathcal{A}, j$)
        /* Add to $\mathcal{A}$ a new row and a new column for to variable $q_{ij}$ */
        $\mathcal{A} \leftarrow$ Add_Row($\mathcal{A}, q_{ij}$))
        $\mathcal{A} \leftarrow$ Add_Column($\mathcal{A}, q_{ij}$)
    }
    return $\mathcal{N}$
}
```

Figure 3: The *Decompose_And_Gate* algorithm.

the maximum power reduction. As pointed out by the authors, 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.

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 proposed in this paper is trace driven, signal probabilities are not employed. Instead, exact switching activity for any node can be 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 into gates with fanin two on the fly using the procedure outlines in Section 4. This represents a structure much closer to the final network after technology mapping. Finally, our algorithm derives directly from the the algorithm of Rajski and Vasudevamurthy [7] which is widely considered the most efficient method for the factorization of Boolean expression.

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 [7]. 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. 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, based on the computation of values denoting the power dissipation due to single-cube and double-cube divisors. These values are computed for the given trace $\mathcal{T}$ and represent the power saving which is obtained if the extraction of the divisor is performed. Recall that all the information of a trace needed for calculating the switching power is compactly represented as a set of vector pairs and that, for a given function x, $\mathcal{F}_{\mathcal{T}}^{x}$ denotes its exact switching activity. Hence, if we consider a two literal single cube divisor $d = x_1x_2$ with coincidence equal to $K(d)$ [3], then the **power value** of d under $\mathcal{T}$ is given by:

$$PV_{\mathcal{T}}(d) \quad = \quad (K(d) - 1) \cdot [\mathcal{F}_{\mathcal{T}}^{x_1} + \mathcal{F}_{\mathcal{T}}^{x_2}] - K(d) \cdot \mathcal{F}_{\mathcal{T}}^{d} + \mathcal{FISAD} \qquad (7)$$

To understand this equation, suppose that extraction of d has been performed creating a new node n_d within $\mathcal{N}$. The first term of the equation accounts for the power saving related to the variables x_1, x_2 which now feed just node n_d instead of $K(d)$ nodes of $\mathcal{N}$. The term $K(d) \cdot \mathcal{F}_{\mathcal{T}}^{d}$ represents the switching activity of the output variable of n_d multiplied by its fanout. Finally the last term $\mathcal{FISAD}$ denotes "fanout internal switching activity difference" and is an estimation of the balance of internal power dissipation for each fanout n_j of n_d due to the extraction of the divisor. The efficient and accurate computation of this term is deferred until later in this section.

[3]Coincidence K means that if d is extracted to create a new node n_d in the network the fanout of this node will be K.

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 $b_1 = z_1 \cdot z_2 \cdot \ldots \cdot z_{M_{b_1}}, \ldots b_B = z_1 \cdot z_2 \cdot \ldots \cdot z_{M_{b_2}}$.

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

$$\begin{aligned} PV_{\mathcal{T}}(d) &= (\mathcal{FO}(d) - 1) \cdot \left[\sum_{k=1}^{K} (\mathcal{F}_{\mathcal{T}}^{x_k}) + \sum_{k=1}^{H} (\mathcal{F}_{\mathcal{T}}^{y_k}) \right] + \\ &- \mathcal{FO}(d) \cdot \mathcal{F}_{\mathcal{T}}^{d} + \sum_{i=1}^{B} \left[\sum_{j=1}^{M_{b_i}} (\mathcal{F}_{\mathcal{T}}^{z_j}) \right] + \mathcal{FISAD} + \mathcal{CG} \qquad (8) \end{aligned}$$

As before, to understand this equation, suppose that the extraction of d has been performed creating a new node n_d within $\mathcal{N}$. The first term of the equation 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 of $\mathcal{N}$. The term $\mathcal{FO}(d) \cdot \mathcal{F}_{\mathcal{T}}^{d}$ represents the switching activity of the output variable of n_d multiplied by its fanout. The third term is the power 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 term $\mathcal{CG}$ stands for "complementary gain" and it 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 $PV_{\mathcal{T}}(d)$.

Finally, the term $\mathcal{FISAD}$ is calculated as follows. Recall that each node n in $\mathcal{N}$ is decomposed into two-input gates to estimate its switching power P_{orig}^{n} as the sum of the switching activities of all the two-input gates. When a divisor d is evaluated, the same operation is repeated for each fanout n_j of the divisor, under the assumption that the divisor has been extracted. Lets call $P_{new}^{n_j}$ the new power estimation obtained for the node n_j. The term $\mathcal{FISAD}$ is simply the sum of $P_{orig}^{n_j} - P_{new}^{n_j}$ for each fanout n_j of d. The key point to note here is that 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

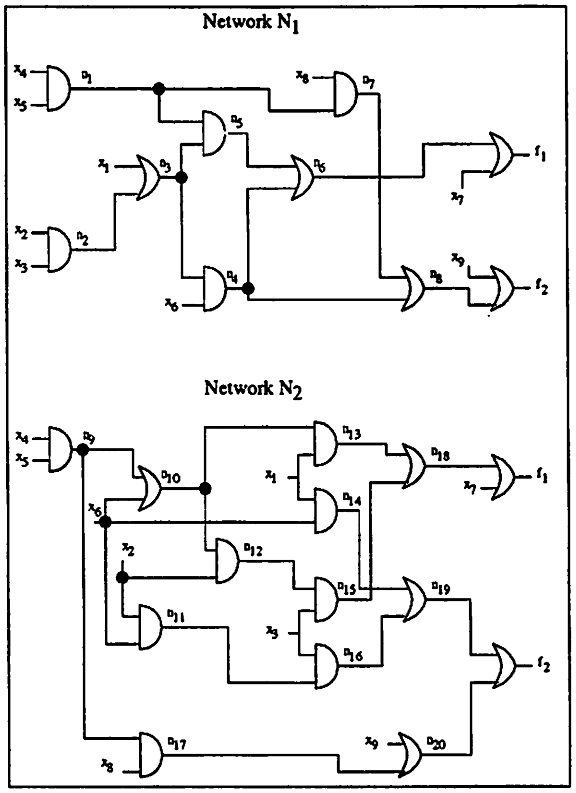

Figure 4: Two alternative extraction results

Input Trace $\mathcal{T}$		Divisor Output Traces				
vector	$x_1x_2x_3x_4x_5x_6x_7x_8x_9$	$\mathcal{F}_{\mathcal{T}}^{(x_2\cdot x_3)}$	$\mathcal{F}_{\mathcal{T}}^{d_1}$	$\mathcal{F}_{\mathcal{T}}^{(x_4\cdot x_5)}$	$\mathcal{F}_{\mathcal{T}}^{d_2}$	$\mathcal{F}_{\mathcal{T}}^{d_3}$
v_1	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_2	1 1 0 0 1 0 0 0 1	0	1	0	0	0
v_3	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_4	1 1 0 0 1 0 0 0 1	0	1	0	0	0
v_5	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_6	1 1 0 1 0 0 1 1 0	0	1	0	0	0
v_7	0 1 0 0 1 0 0 0 0	0	0	0	0	0
v_8	1 1 1 1 0 0 1 1 0	1	1	0	0	0
v_9	0 1 0 0 1 0 0 0 0	0	0	0	0	0
v_{10}	1 1 1 1 0 0 1 0 1	1	1	0	0	0
v_{11}	0 1 0 0 1 0 0 1 1	0	0	0	0	0
v_{12}	1 1 1 1 0 0 1 1 1	1	1	0	0	0
v_{13}	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_{14}	1 1 0 1 0 0 1 1 1	0	1	0	0	0
v_{15}	0 1 0 0 1 0 0 1 1	0	0	0	0	0
v_{16}	1 1 0 0 1 0 0 0 1	0	1	0	0	0
v_{17}	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_{18}	1 1 0 0 1 0 0 0 1	0	1	0	0	0
v_{19}	0 1 0 0 1 0 0 0 1	0	0	0	0	0
v_{20}	1 1 0 0 1 0 0 0 1	0	1	0	0	0

Table 6: Traces for extraction example

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 $\mathcal{N}$, which has 9 inputs and 2 outputs and 2 internal complex nodes f_1 and f_2.

$$\begin{aligned} f_1 &= x_1 \cdot x_4 \cdot x_5 + x_1 \cdot x_6 + x_2 \cdot x_3 \cdot x_4 \cdot x_5 + x_2 \cdot x_3 \cdot x_6 + x_7 \\ f_2 &= x_1 \cdot x_6 + x_2 \cdot x_3 \cdot x_6 + x_4 \cdot x_5 \cdot x_8 + x_9 \end{aligned}$$

Performing the extraction using the area minimal option as described in the algorithm of Rajski and Vasudevamurthy [7] yields a network with a divisor d_1 and the new network is:

$$\begin{aligned} d_1 &= x_1 + x_2 \cdot x_3 \\ f_1 &= d_1 \cdot x_4 \cdot x_5 + d_1 \cdot x_6 + x_7 \\ f_2 &= d_1 \cdot x_6 + x_4 \cdot x_5 \cdot x_8 + x_9 \end{aligned}$$

Decomposition of this network into 2-input AND and OR gates yields the network $\mathcal{N}_1$ illustrated in Figure 4.

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 Table 6. 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 $d_1 = x_1 + x_2 \cdot x_3$ and $d_2 = x_6 + x_4 \cdot x_5$ and one single divisor $d_3 = x_4 \cdot x_5$ which have power value greater than zero. Using the equations presented in this section and observing the traces of Table 6, we compute:

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

d_2 is extracted instead of d_1 to obtain:

$$\begin{aligned} d_2 &= x_6 + x_4 \cdot x_5 \\ f_1 &= d_2 \cdot x_6 + d_2 \cdot x_2 \cdot x_3 + x_7 \\ f_2 &= x_1 \cdot x_6 + x_2 \cdot x_3 \cdot x_6 + x_4 \cdot x_5 \cdot x_8 + x_9 \end{aligned}$$

Using the trace-driven decomposition algorithm yields the network $\mathcal{N}_2$ illustrated in Figure 4. While $\mathcal{N}_2$ has 14 gates, $\mathcal{N}_1$ has 11.

In $\mathcal{N}_1$ all the internal nodes have zero switching activity under $\mathcal{T}$ except n_3 which has 19 and n_2 which has 6. In $\mathcal{N}_2$ all the internal nodes have zero switching activity $\mathcal{T}$, except n_{20} which has 2 transitions. The sum of the input transition activity is 65, while the total transition activity for the two output nodes f_1, f_2 is equal to 10. Accounting for the fanout of two for n_3 in $\mathcal{N}_1$ and x_1 and x_3 in $\mathcal{N}_2$ yields a 14% power reduction between the two networks.

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

5 Experimental Results

This section describes a sample of experimental results to demonstrate the proposed methodology for trace driven low power logic synthesis. Three distinct sets of results are presented. Each set develops and indicates a particular aspect of the methodology that we believe is essential in understanding and appreciating the contributions of this paper.

The first set of results shown in Table 7 compares the two-level heuristic and exact minimization algorithm against *espresso.* As with all results presented

PLA	I/O	ratio: heuristic-elp / espresso $\mathcal{ISA}_\mathcal{T}$	$\mathcal{OSA}_\mathcal{T}$	$W_\mathcal{T}(Q)$	cputime	ratio: exact-elp / espresso $W_\mathcal{T}(Q)$	cputime
alu4_F3	14/1	1.02	0.05	0.70	6.75	0.68	6.7
alu4_F4	14/1	1.02	0.14	0.86	150.8	0.85	150.8
apex4_F16	9/1	1.04	0.13	0.88	0.21	0.87	0.25
apex4_F17	9/1	1.01	0.13	0.89	0.07	0.87	0.08
apex4_F18	9/1	1.10	0.13	0.90	0.05	0.86	0.05
apex4_F19	9/1	1.07	0.10	0.91	0.08	0.87	0.09
duke2_F1	22/1	1.07	0.10	0.90	0.08	0.72	0.15
duke2_F2	22/1	1.01	0.64	0.92	0.68	0.89	0.17
duke2_F7	22/1	0.96	0.60	0.89	0.06	0.89	748.79
in3_F3	35/1	1.07	0.36	0.91	0.27	-	-
in3_F9	35/1	1.00	0.80	0.91	0.03	0.91	0.16
in3_F11	35/1	1.01	0.80	0.91	0.03	0.91	0.17
in3_F13	35/1	0.98	0.39	0.86	1.75	-	-
in3_F14	35/1	1.00	0.81	0.91	0.03	-	-
in3_F15	35/1	0.77	0.74	0.63	0.74	-	-
in3_F16	35/1	1.40	0.23	0.82	3.79	-	-
in3_F19	35/1	1.31	0.01	0.83	0.12	-	-
in3_F20	35/1	1.01	0.09	0.69	0.08	0.69	0.32
in3_F29	35/1	1.07	0.30	0.83	0.24	0.83	2.87
seq_F12	41/1	1.00	0.50	0.93	0.41	0.93	2.54
seq_F23	41/1	1.09	0.28	0.87	8.53	-	-
seq_F24	41/1	1.10	0.11	0.84	57.1	-	-
seq_F32	41/1	1.01	0.17	0.72	0.22	0.72	0.35

Table 7: Experimental results on MCNC two-level circuits

in this section, all area and power values are presented as ratios between the new algorithms and the original versions. The traces for these examples were 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. Thus the trace length is p^2. The trace was intentionally created artificially and 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 1 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 can be inferred:

1. Dramatic reductions can be achieved by selection of the appropriate implicants for minimal power switching compared to *espresso* (column marked $\mathcal{OSA}_{\mathcal{T}}$).

2. The overall reduction in power (column marked $W_{\mathcal{T}}(Q)$) is limited by the input switching activity (column marked $\mathcal{ISA}_{\mathcal{T}}$) 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*. Further note that the reductions obtained in the table were obtained after biasing the trace to expressly "confuse" espresso. Thus, **two-level minimization can provide only an incremental reduction in power once the trace has been determined.**

In the second experiment each circuit is optimized using the standard script, called *script.rugged* in SIS. A version of the same script with the low-power optimization algorithms for decomposition, simplification and extraction was used to compare results. Both technology independent and technology mapped results are reported; note that the technology mapped results serve to validate the two-input simple gate model with consideration for fanout that is employed for power estimation in the technology independent optimization algorithms. The trace for this experiment was also artificially generated as follows: A 16-bit counter is used to generate 10000 vectors in sequence. In case there are more than 16 inputs, the pattern is repeated every 16 bits in the vector. This experiment serves to mimic correlations among inputs that cannot be easily modeled by probabilistic or statistical methods. The circuits are ordered by size (number of two-input gates after optimization using *script.rugged*). The most significant conclusion from these small examples is the validation of the technology independent power model used in simplification, decomposition and extraction. Of the 29 examples, 15 show a reduction in total power, while 7 show no change and 7 show a small increase in total power using the new algorithms.

			technology independent ratio: LP/rugged		technology mapped ratio: LP/rugged	
Circuit	I/O	Opt. Gates	Area	Power	Area	Power
b1	3/4	9	0.88	0.85	0.86	0.85
majority	5/1	11	0.82	0.68	0.85	0.72
cm138a	21/1	16	1.62	0.50	1.47	0.43
cm82a	5/3	17	1.0	1.17	1.29	1.15
cm42a	4/10	18	1.56	0.88	1.32	0.92
tcon	17/16	24	1.0	1.0	1.0	1.0
decod	5/16	26	1.58	1.14	1.72	1.10
cm152a	11/1	27	1.0	0.69	1.03	0.72
cm151a	12/2	28	1.07	0.57	1.22	0.73
cm163a	16/5	37	1.03	0.76	1.06	0.85
cm85a	11/3	40	1.05	0.89	1.08	1.02
x2	10/7	40	0.975	0.62	1.20	0.77
cc	21/20	43	1.40	1.11	1.33	1.11
cu	14/11	43	1.23	0.73	1.27	0.70
cm150a	21/1	44	1.29	1.42	1.29	1.45
parity	16/1	45	1.0	1.0	1.0	1.0
cmb	16/4	46	0.98	0.71	0.96	0.87
cm162a	14/5	49	0.81	0.61	0.85	0.69
mux	21/1	58	1.23	1.09	1.29	1.01
cordic	23/2	60	1.38	1.20	1.34	1.25
pcle	19/9	62	0.90	0.86	0.85	0.95
comp	32/3	85	1.45	1.20	1.33	1.17
lal	26/19	89	0.92	1.23	0.92	1.13
c8	28/18	118	1.25	1.07	1.35	1.07
cht	47/36	120	1.21	0.97	1.21	0.92
b9	41/21	124	1.28	0.69	1.22	0.88
count	35/16	128	1.60	0.98	137	1.02
apex7	49/37	192	1.17	0.88	1.20	0.97
ttt2	24/21	197	0.79	1.23	1.22	0.76

I/O: # inputs / # output
Opt. gates: # of gates in optimized circuit after *script.rugged*
Area: Ratio of area for power script versus *script.rugged*
Power: Ratio of power for power script versus *script.rugged*

Table 8: Experimental results on MCNC multi-level circuits

The final experiment is performed to validate the trace driven approach on FSMs where the trace is not artificially biased. For these FSMs, given a reset state, 10000 random primary input vectors were applied. The trace of the input and latch values obtained by sequential logic simulation served as the trace for power minimization. The results are shown in Table 9. Of the 11 examples reported, 9 show a reduction in power with an average reduction of 13%.

			technology independent ratio: LP/rugged		mapped networks ratio: LP/rugged	
FSM	I/O/L	Opt. Gates	Area	Power	Area	Power
s27	4/1/3	9	1.13	0.90	0.97	0.92
s208	10/1/8	70	1.07	1.05	1.03	1.09
s298	3/6/14	11	0.87	0.63	0.92	0.68
s344	9/11/15	123	1.09	1.07	1.05	1.02
s349	9/11/15	122	1.06	1.03	1.02	0.97
s382	3/6/21	142	1.06	0.84	1.02	0.83
s386	7/7/6	108	1.43	0.97	1.27	0.97
s400	3/6/21	137	1.12	0.81	1.05	0.81
s444	3/6/21	136	1.18	0.75	1.08	0.75
s526	19/7/6	171	1.15	0.80	1.14	0.91
s526n	3/6/21	191	0.99	0.95	1.08	0.95

I/O/L: # inputs / # output / # latches
Opt. gates: # of gates in optimized circuit after *script.rugged*
Area: Ratio of area for power script versus *script.rugged*
Power: Ratio of power for power script versus *script.rugged*

Table 9: Experimental results on *ISCAS89 benchmark suite* FSMs

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.