Power Optimization of Technology-Dependent Circuits Based on Symbolic Computation of Logic Implications

R. IRIS BAHAR
Brown University
ERNEST T. LAMPE
Brown University
and
ENRICO MACII
Politecnico di Torino

This paper presents a novel approach to the problem of optimizing combinational circuits for low power. The method is inspired by the fact that power analysis performed on a technology mapped network gives more realistic estimates than it would at the technology-independent level. After each node's switching activity in the circuit is determined, high-power nodes are eliminated through redundancy addition and removal. To do so, the nodes are sorted according to their switching activity, they are considered one at a time, and learning is used to identify direct and indirect logic implications inside the network. These logic implications are exploited to add gates and connections to the circuit; this may help in eliminating high-power dissipating nodes, thus reducing the total switching activity and power dissipation of the entire circuit. The process is iterative; each iteration starts with a different target node. The end result is a circuit with a decreased switching power. Besides the general optimization algorithm, we propose a new BDD-based method for computing satisfiability and observability implications in a logic network; furthermore, we present heuristic techniques to add and remove redundancy at the technology-dependent level, that is, restructure the logic in selected places without destroying the topology of the mapped circuit. Experimental results show the effectiveness of the proposed technique. On average, power is reduced by 34%, and up to a 64% reduction of power is possible, with a negligible increase in the circuit delay.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; *Optimization*

General Terms: Design

Additional Key Words and Phrases: Aids, automation, design synthesis, logic design

Authors' addresses: R. I. Bahar, Division of Engineering, Box D, Brown University, Providence, RI 02912; E. T. Lampe, Division of Engineering, Box D, Brown University , Providence, RI 02912; E. Macii, Dip. di Automatica e Informatica, Politecnico di Torino, Torino, 10129, Italy.

1. INTRODUCTION

Excessive power dissipation in electronic circuits reduces reliability and battery life. The severity of the problem increases with the level of transistor integration. Therefore, much work has been done on power optimization techniques at all stages of the design process. During high-level design, power dissipation can be reduced through algorithmic transformations [Chandrakasan et al. 1995]; architectural choices [Chandrakasan and Brodersen 1995]; and proper selection of high-level synthesis tools [Macii et al. 1997]. At the logic level, the focus of this paper, the main objective of low-power synthesis algorithms is reducing the logic's switching activity, weighted by the capacitive load. Logic optimization may occur at both the technology-independent and the technology-dependent stages of synthesis flow. At the technology-independent stage, combinational circuits are optimized by two-level minimization [Bahar and Somenzi 1995; Iman and Pedram 1995b]; don't care based minimization [Shen et al. 1992; Iman and Pedram 1994]; logic extraction [Roy and Prasad 1992; Iman and Pedram 1995a]; and selective collapsing [Shen et al. 1992]. At the technology-dependent stage, technology decomposition [Tsui et al. 1993] and technology mapping [Tsui et al. 1993; Tiwari et al. 1993; Lin and de Man 1993] methods have been proposed. Finally, after an implementation of the circuit is available, power can still be reduced by applying technology remapping [Vuillod et al. 1997] and gate resizing [Bahar et al. 1994; Coudert et al. 1996].

It is difficult to measure the power dissipation of technology-independent circuits with a dependable level of accuracy. Therefore, we propose a method that can be applied to technology mapped circuits, based on the idea of reducing the total switching activity of the network through redundancy addition and removal.

Previous work on this subject includes the methods proposed in Cheng and Entrena [1993], Entrena and Cheng [1993], and Chang and Marek-Sadowska [1994], in which a set of mandatory assignments is generated for a given target wire. A set of candidate connections is then identified. Each candidate connection, when added to the circuit, causes the target fault to become untestable, and thus the faulty connection to become redundant. However, since the additional connection may change the circuit's behavior, a redundancy check is needed to verify that the new connection itself is redundant before it is added to the circuit.

Another ATPG-based approach is proposed in Rohfleish et al. [1996]. This technique uses an analysis tool introduced in Rohfleish et al. [1995] to identify permissible transformations on the network that may reduce power dissipation. The method for finding permissible transformations is simulation-based; implications are classified into C1-, C2-, and C3-clauses; and bit-parallel fault simulation is performed to eliminate most of the invalid clauses. The remaining *potentially valid clauses* are combined to create different clause combinations, each of which is checked for validity

using ATPG. As more complex clauses are included, the number of combinations to consider can increase dramatically.

Other work in the area of redundancy addition and removal uses *recursive learning* as a guide in the process. For instance, the work proposed in Kunz and Menon [1994] introduces an ATPG-based method for identifying *indirect implications*, which may indicate useful transformations of a circuit. Once an implication is identified, it may be used to add a redundant connection, which is guaranteed not to change the behavior of the circuit. An additional redundancy elimination step is required to identify what other redundant connections, if any, have been created by adding this new connection.

In our method, we start from a circuit that is already implemented in gates from a technology library and perform power analysis on it, so as to identify its high- and low-power dissipating nodes. We use a sophisticated learning mechanism (related to those of Trevillyan et al. [1986], Kunz and Pradhan [1992], Kunz [1993], and Jain et al. [1995]) to find satisfiability and observability implications in the circuit in the neighborhood of the target nodes. Such implications are used to identify network transformations that add and remove connections in the circuit (as in Kunz and Menon [1994]), with the object of eliminating the high-power nodes or connections from them. The method is innovative in two main aspects; first, it uses a powerful learning procedure based on symbolic calculations rather than ATPG-based methods. This approach allows the identification of very general forms of logic implications; second, it operates at the technology-dependent level, which allows more accurate power estimates to drive the overall resynthesis process. Experimental results, obtained on a sample of the `Mcnc 91` benchmarks [Yang 1991], show the viability and the effectiveness of the proposed approach.

The rest of this manuscript is organized as follows. Section 2 gives definitions for subsequent usage. In Section 3, we propose a symbolic procedure to compute logic implications using learning. Section 4 describes a power optimization procedure that is based on redundancy addition and removal. Section 5 is dedicated to experimental results; and finally, Section 6 gives conclusions and directions for future work.

2. BACKGROUND

In this section, we provide definitions of terms and introduce concepts used in the rest of the paper. We first define characteristic functions and relations; next, we discuss the concept of untestable faults and show how these faults may be eliminated through redundancy removal. Finally, we show how logic implications may be used to create new untestable faults in a circuit that may be subsequently removed to create an overall better optimized circuit.

2.1 Characteristic Functions and Relations

Given a set of points $\mathcal{S}$ in the Boolean space, $B^n = \{0, 1\}^n$, it is possible to define a function $\chi_{\mathcal{S}} : B^n \rightarrow B$, called the characteristic function of $\mathcal{S}$, that evaluates to 1 exactly for the points of B^n that belong to $\mathcal{S}$. Formally:

$$\forall x \in B^n, x \in \mathcal{S} \Leftrightarrow \chi_{\mathcal{S}}(x) = 1 \tag{1}$$

This definition can be extended to arbitrary finite sets, provided that the objects in the set are properly encoded with binary symbols.

Since characteristic functions are Boolean functions, they can be represented and manipulated very efficiently with binary decision diagrams (BDDs) [Bryant 1986]. Consequently, it is usually possible to handle much larger sets of characteristic functions if they are represented with BDDs rather than an explicit enumeration of all the elements in the sets.

In this paper we restrict our attention to relations, that is, to sets that are subsets of some Cartesian product. Let $\mathcal{S}$ and $\mathcal{Q}$ be two sets and let $R \subseteq \mathcal{S} \times \mathcal{Q}$ be a binary relation (i.e., the elements of R are pairs of elements from $\mathcal{S}$ and $\mathcal{Q}$). Using different sets of variables, $x = (x_1, \ldots .x_n)$ and $y = (y_1, \ldots, y_m)$, to encode the elements of $\mathcal{S}$ and $\mathcal{Q}$, we can represent this relation through its characteristic function:

$$\chi_R(x, y) \leq \chi_{\mathcal{S}}(x) \cdot \chi_{\mathcal{Q}}(y) \tag{2}$$

As an example, consider sets $\mathcal{S} = \{$`red, blue, green`$\}$ and $\mathcal{Q} = \{$`WHITE, BLACK, GREY, RED, ORANGE`$\}$, and relation $R = \{(s, q) \in \mathcal{S} \times \mathcal{Q} : s =$ **lower_case**$(q)\}$. Let us encode the elements of $\mathcal{S}$ using variables $x = (x_1, x_2)$ as `red` $= 00$, `blue` $= 01$ and `green` $= 10$. Similarly, the elements of $\mathcal{Q}$ are encoded using variables $y = (y_1, y_2, y_3)$ as `WHITE` $= 000$, `BLACK` $= 001$, `GREY` $= 010$, `RED` $= 011$, and `ORANGE` $= 100$. We have

$$\chi_{\mathcal{S}}(x) = x' + x'$$

$$\chi_{\mathcal{Q}}(y) = y' + y'_2 y'_3$$

The characteristic function of relation R is then:

$$\chi_R(x, y) = x'_1 x'_2 y'_1 y_2 y_3$$

That is, $R = \{($`red, RED`$)\}$. Obviously, the definition of a binary relation given above can be easily extended to the case of n-ary relations, that is, relations that are subsets of Cartesian products of order n.

2.2 Circuits and Faults

A combinational circuit, C, is an acyclic network of combinational logic gates. If the output of a gate g_i is connected to an input of a gate g_j, then g_i is a *fanin* of g_j and gate g_j is a *fanout* of gate g_i.

A combinational circuit may have a failure due to a wire being shorted to the power source or ground. Such a failure may be seen as a *stuck-at fault*. That is, the circuit behaves as if the wire were permanently stuck-at-1 or stuck-at-0. We assume that single stuck-at faults are used to model failures in a circuit. Let C be a combinational circuit and let C_f be the same circuit in which fault f is present. Fault f is *untestable* if and only if the output behaviors of C and C_f are identical for any input vector applied to both C and C_f.

2.3 Redundancy Addition and Removal

Any automatic test-pattern generation program may be used to detect untestable faults (e.g., Sentovich et al. [1992]). The computed information may be used to simplify the network by propagating the constant values (zero or one), due to untestable stuck-at connections, throughout the circuit.

Redundancy removal is one of the most successful approaches to logic optimization (e.g., Cho et al. [1993]). However, its effectiveness greatly depends on the number of redundancies: For circuits that are 100% testable, redundancy removal does not help, and hence techniques based on redundancy addition and removal have been proposed.

The concept of redundancy addition and removal is best explained through an example. In Figure 1(a), all stuck-at faults are testable. Therefore, no simplification through redundancy removal is possible on the network as is. If gate $G10$ is transformed into a 2-input NOR gate and the additional input is connected from the inverted output of gate $G6$ (see shaded logic in Figure 1(b)), then the behavior of the circuit at its primary outputs will remain unaltered; however, three previously testable faults (shown with X's in Figure 1(b)) now become untestable. Through redundancy removal, gate simplification, collapse of the inverter chain, and DeMorgan transformations, the circuit can be simplified as shown in Figure 1(e).

Adding redundant gates and connections to a circuit may increase area, delay, and power consumption by an amount that may not be recoverable by the subsequent step of redundancy removal. Furthermore, not all redundancies in a network are necessarily due to suboptimal design; automatic synthesis and technology mapping tools sometimes resort to inserting redundant gates to increase the speed of a digital design [Keutzer et al. 1991]. As a consequence, adding and removing redundancy are delicate operations that should be performed within the constraints of the objective function being minimized.

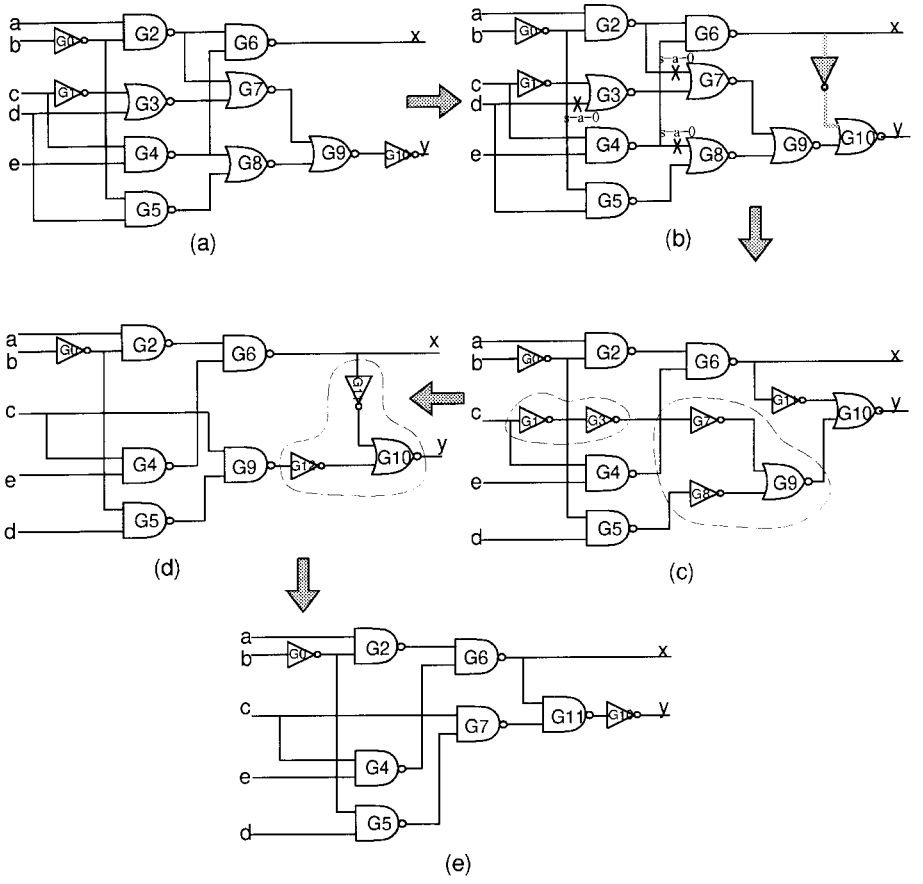

Fig. 1. Example of redundancy addition and removal.

2.4 Logic Implications

In general, there may be a large variety of choices available in selecting the new connections (and logic gates) that may be added to the original circuit so as to introduce redundancies. Kunz and Menon have proposed an effective solution for selecting these connections through a method derived from *recursive learning* [Kunz and Menon 1994]. Recursive learning is the process of determining all value assignments necessary for detecting a single stuck-at fault in a combinational circuit. This process is equivalent to finding *direct* and *indirect implications* in the circuit, that is, finding all the value assignments necessary for a given signal to take on a specific value (satisfiability implications) or to make a given signal observable (observability implications).

2.4.1 *Direct Implications*. If a value assignment can be determined by simple propagation of other signal values through a circuit, then this is known as a *direct satisfiability implication*. For example, consider the

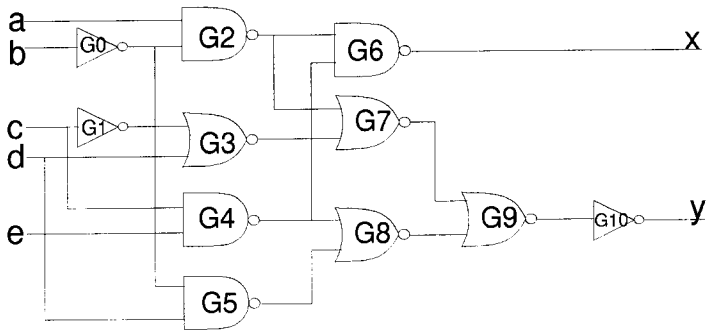

Fig. 2. Example circuit with satisfiability implications.

circuit in Figure 2, where the signal assignment $G9 = 1$ has been made. This assignment directly implies the value assignments $G7 = 0$ and $G8 = 0$, as these are the only assignments for $G7$ and $G8$ that will justify the output of gate $G9$ to a value of 1.

Similarly, if a value assignment such that a given node is observable can be determined by simple propagation of other signal values through a circuit, then this is known as a *direct observability implication*. For example, consider the circuit in Figure 3. For $G2$ to be observable, $G1 = 1$ and $G6 = 1$ are the only assignments for $G1$ and $G6$ that will make gate $G2$ observable. That is, the observability of $G2$ directly implies $G1 = 1$ and $G6 = 1$.

2.4.2 *Indirect Implications*. It was shown in Kunz and Menon [1994] that resorting to direct implications only to remove or add redundancy may not provide enough options to achieve significant improvement on the circuit being optimized. Thus it is interesting to look at another type of implication, namely *indirect implication*.

To illustrate the concept of indirect implication, consider again the circuit of Figure 2, and suppose that the signal assignment $G9 = 0$ has been made. We may optionally assign either $G7 = 1$ or $G8 = 1$ to justify this assignment; however, neither assignment is essential. We therefore conclude that there are no essential direct implications we can make from this assignment. However, upon closer inspection, we can determine that the value assignment $G9 = 0$ indirectly implies the value assignment $G6 = 1$. If we temporarily assign $G7 = 1$, then $G6 = 1$ and $G2 = G3 = 0$ are both essential to satisfy $G7 = 1$. Likewise, by temporarily assigning $G7 = 0$ we find that $G6 = G8 = 1$ and $G4 = G5 = 0$ are essential assignments. In either case, $G6 = 1$ is an essential assignment; we can conclude that $G6 = 1$ is an essential assignment for, and is indirectly implied by, $G9 = 0$.

The indirect implication shown in the previous example falls in the category of *satisfiability implications*. As an example of an indirect *observ-*

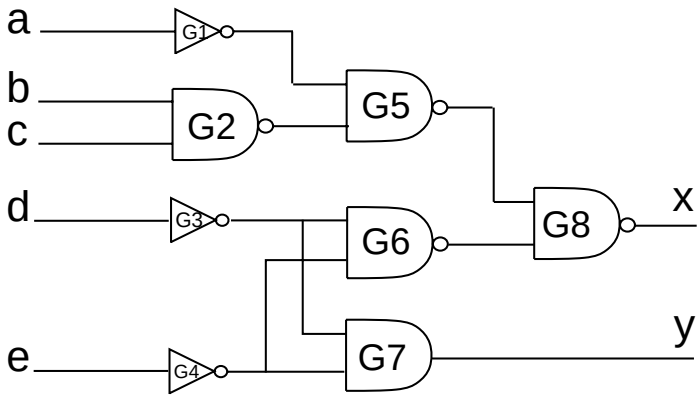

Fig. 3. Example circuit with observability implications.

ability implication, consider again the circuit in Figure 3. For $G3$ to be observable, $G4 = 1$ must be true (and vice versa). Therefore, the observability of $G3$ indirectly implies $G4 = 1$.

Satisfiability implications have a bidirectional property, which does not apply to observability implications. For example, for the circuit shown in Figure 2, it can be shown that the satisfiability implication $(G8 = 1) \Rightarrow (G2 = 1)$ can be reversed and complemented so that $(G2 = 0) \Rightarrow (G8 = 0)$. This bidirectionality is a general property of satisfiability implications. However, this property does not hold for observability implications. Referring to the previous example of Figure 3, although the observability of $G2$ (that is, both $G2 = 0$ and $G2 = 1$) implies $G1 = 1$ and $G6 = 1$, the observability of $G1$ does imply $G2 = 1$, but it does not imply $G2 = 0$. This is because it cannot be assumed that $G1$ and $G2$ are always observable under the same conditions. This lack of bidirectionality makes adding redundant logic to the circuit more constrained when observability rather than satisfiability implications are used. For this reason, implications of the two types must be handled separately when applying optimizations based on redundancy addition and removal.

Kunz and Menon [1994] observed that the presence of indirect implications is a good indication of suboptimality of a circuit. This is especially true for satisfiability implications. We have taken inspiration from this approach to implement the power optimization algorithm we propose in this paper. However, certain implications should not be eliminated from consideration simply because they were found through direct propagation of logic values. In fact, most observability implications are found "directly," since they are determined primarily through forward propagation of implications. In the next section we discuss how direct, as well as indirect implications can be computed symbolically using BDD-based data structures. Then, in Section 4, we outline the overall optimization procedure.

3. COMPUTING IMPLICATIONS SYMBOLICALLY

In this section we introduce our symbolic procedure to compute logic implications through recursive learning. In what follows, a *literal* is either a variable or its complement; a *cube* is a product of literals.

We start with a set $T = \{T_j\}$ of relations $T_j \subseteq B^n \times B$, where $B = \{0, 1\}$ within a universe of n Boolean variables. Each T_j can be thought of as a characteristic function $y_j \equiv f_j(y)$ for the gate j describing its functional behavior. For example, if j is a NAND gate with inputs y_i and j_k, $T_j = y_j(y_i y_k)' + y'_j(y_i y_k)$, whereas if j is a NOR gate, then $T_j = y_j(y_i + y_k)' + y'_j(y_i + y_k)$. If the Boolean variables y_j, y_i, and y_k are assigned in such a way that T_j evaluates to 0, then the variable assignments violate the required behavior of the gate and are invalid. In this way, an entire network may be described within this set $\{T_j\}$.

T_j may also be used to express the *observability* relation for a gate. For instance, consider gate $G4$ in Figure 2. For the signal at the output of gate $G4$ to be observable at either primary output x or y, $G5 = 0$ or $G2 = 1$. That is, $T^{obs}_{G4} = G5' + G2$ and $T^{sat}_{G4} = G4'ce + G4(ce)'$.

Now, if we are given an initial assignment (i.e., assertion $A(y)$), we may compute its implications by applying $A(y)$ to the set $\{T_j\}$:

$$I(y) = A(y) \prod_j T_j(y) \tag{3}$$

Furthermore, we may wish to extract only the necessary, or *essential*, literals from the implication $I(y)$:

$$c(y) = \texttt{Essential}(I(y)) \tag{4}$$

For example, if the assertion $A = (y_0 = 1)$ is applied to the characteristic function $T_0 = y_0 \cdot (y_1 + y_2)y_3 + y'_0 \cdot (y'_1 y'_2 + y'_3)$, the resulting implication becomes $I = y_3(y_1 + y_2)$. However, in order to satisfy the characteristic function given this assertion, only $y_3 = 1$ is essential (i.e., neither $y_1 = 1$ nor $y_2 = 1$ is strictly necessary). Therefore, the implication $y_3 = 1$ is stored separately in $c(y)$.

In this way, given an initial assertion $A(y)$, $c(y)$ is a list of essential gate assignments over the entire network. Implemented using BDDs, $c(y)$ is represented as a single cube. As shown later, essential literals require simple redundant logic to be added to the network, and therefore it may be beneficial to store them separately.

3.1 Direct Satisfiability Implications

Given a set of characteristic functions, $T^{Sat} = \{T^{Sat}_j\}$, and an initial assertion, A, we compute all direct satisfiability implications using the procedure `impSatDirect` shown in Figure 4. Recall that a direct implication is one that can be found by propagating the immediate effects of the logic

```
    procedure impSatDirect(A,T^Sat) {
1.      imp = Essential(A);
2.      Q = {k|T_k^Sat is fanout of some x_j ∈ imp} ∪ {x_j|x_j ∈ imp};
3.      while (Q ≠ ∅) {
4.          k = select_and_remove_one(Q);
5.          T_k^Sat = Cofactor(T_k^Sat,imp);
6.          if (T_k^Sat ≡ zero) return(inconsistent,T^Sat);
7.          t = Essential(T_k);
8.          if (t ≡ one) continue;
9.          imp = imp · t;
10.         Q = Q ∪ {k_t|k_t is a fanout of some x_j ∈ t} ∪ {x_j|x_j ∈ t};
            }
        return(imp,T^Sat);
    }
```

Fig. 4. Procedure `impSatDirect`.

assertion forward and backward through the specified set of relations T, *without case analysis.* The essential implications are returned separately in the cube *imp*. In our implementation, T^{Sat}, A, and *imp* are all represented as BDDs.

The direct implications are computed as follows. First, we initialize the list of implications to be the cube-free, or essential, part of A. Next, we consider all possible direct implications, one at a time, on the gates from the set Q (line 2). The set Q contains all fanouts of variables in the support of the cube *imp*, plus all the variables in *imp* itself.

The first step of the **while** loop selects one gate from Q, and removes it from the set. Inside the loop (lines 4 to 10), procedure `Essential` is called, according to Eq. (4), to discover any new essential implication. We exit the loop and return (line 6) whenever T_k^{Sat} reduces to *zero*, implying that the assertion A is logically inconsistent with one of the relations in the set of subrelations $\{T_i^{Sat}\}$.

The literal function t (line 7) represents newly discovered essential implications. On each pass through the **while** loop, these new implications are added to the global implications cube *imp* (line 9), thereby accumulating all the implications of the original assertion A. As these implication variables are found, they are also appended to the set Q (line 10) in order to evaluate their effect on the rest of the network.

In addition to the cube *imp*, the procedure `impSatDirect` also returns T^{Sat}, which has now become a reduced set of relations comprised of the cofactor of each original relation with respect to the essential implications *imp* (that is, $T^{Sat}(imp)$). Notice that $T^{Sat}(imp)$ is itself a set of implications, albeit nonessential ones. These more general implications may also

be useful for optimizing a network, though they are not as straightforward to apply to the network. This is discussed in more detail later in the paper.

3.2 Direct Observability Implications

If observability implications are also to be used in the evaluation of direct implications, procedure `impSatDirect` may be expanded such that relation T_j^{obs} is evaluated along with T^{Sat} whenever signal j is on the observability frontier. The procedure is now renamed `impDirect`, shown in Figure 5.

Lines 1 to 10 are almost identical to those of procedure `impSatDirect`. The observability implications are computed beginning on line 12. If the gate k is on the frontier, we find new implications in the same way as in `impSatDirect`, only this time using the observability characteristic functions, T^{Obs}. Updating the frontier makes the code more complicated.

If the reduced observability relation T_k^{Obs} evaluates to 1, then the fault is observable through at least one fanout of k, and the frontier should be pushed forward to include these fanout gates (line 24). Furthermore, if all the fanouts of gate k have been implied, then k is removed from the frontier (line 27). Finally, if the frontier ever becomes 0 (i.e., empty), then the assertion is not observable, and never will be. This case is checked in line 16.

3.3 Indirect (Recursively Learned) Implications

Using the symbolic direct implications procedure from the previous section, we can find indirect, or recursively learned, implications by setting temporary orthogonal constraints on the initial assertion, finding implications based on these constraints, and extracting the common implication as the full set of implications. We define orthogonal constraints as a set of functions $\{f_i\}$, $0 \leq i < n$, such that

$$\sum_{0}^{n-1} f_i = 1 \text{ and } f_i \cdot f_j = 0, \forall i \neq j \tag{5}$$

Although we may use any orthogonal set of functions, to simplify the recursive implication procedure, we use the orthogonal constraints $f(y)$ and $f'(y)$.

Say that we extract a function $f(y)$ from the network, and add it to the original assertion $A(y)$ such that

$$A_1(y) = A(y) \cdot f(y), A_0(y) = A(y) \cdot f'(y) \tag{6}$$

We apply each assertion $A_1(y)$ and $A_0(y)$ separately to the network using Eq. (3) to get two different sets of implications, $I_1(y)$ and $I_0(y)$, respectively. For each variable y_j we combine these implications such that

$$I(y) = \prod_j (I_1(y_j) \cdot I_0(y_j)) \tag{7}$$

```
    procedure impDirect(A,T^Sat,T^Obs) {
1.      imp = Essential(A);
2.      frontier = imp;
3.      Q = {k|x_k is fanout of some x_j ∈ imp} ∪ {x_j|x_j ∈ imp} ∪
            {k|x_k is on frontier and has same fanout as x_j ∈ imp};
4.      while (Q ≠ ∅) {
5.          k = select_and_remove_one(Q);
6.          T_k^Sat = Cofactor(T_k^Sat,imp);
7.          if (T_k^Sat ≡ zero) return(inconsistent,T^Sat);
8.          t = Essential(T_k);
9.          if (t ≡ one) continue;
10.         imp = imp · t;
11.         Q = Q ∪ {k|x_k is a fanout of some x_j ∈ t} ∪ {x_j|x_j ∈ t} ∪
                {k|x_k is on frontier and has same fanout as x_j ∈ t};
            /* Begin Observability Calculations. */
12.         if (k is on frontier) {
13.             T_k^Obs = Cofactor(T_k^Obs,imp);
14.             if (T_k^Obs = zero) {
15.                 frontier = Cofactor(frontier,k);
16.                 if (frontier ≡ zero) return (unobservable,T^Sat,T^Obs);
                }
17.             else {
18.                 t = Essential(T_k^Obs);
19.                 imp = imp · t;
20.                 T_k^Obs = Cofactor(T_k^Obs,t);
21.                 if (T_k^Obs = one) {
                        /* Frontier is pushed forward */
22.                     foreach (s ∈ fanout(k)) {
23.                         if (k is observable from s) {
24.                             frontier = frontier · s;
25.                             Q = Q ∪ s;
                            }
                        }
26.                     if (∀s ∈ fanout(k), s ∈ imp)
27.                         frontier = Cofactor(frontier,k);
                    }
                }
            }
        }
        return(imp,T^Sat,T^Obs);
    }
```

Fig. 5. Procedure impDirect.

```
procedure indirectImps(A,T^Sat,T^Obs,level) {
1.   (directImps,T^Sat,T^Obs) = impDirect(A,T^Sat,T^Obs);
2.   if (∀i, (T_i^Sat ≡ one)) return(directImps);
3.   if (level ≥ maxlevel) return(directImps);
4.   y_j = selectVariable(directImps);
5.   (posImps) = indirectImps (y_j,T^Sat,T^Obs,level+1);
6.   (negImps) = indirectImps (y'_j,T^Sat,T^Obs,level+1);
7.   indirectImps = cubeIntersect(posImps, negImps);
8.   impCube = directImps · indirectImps;
     return(impCube);
}
```

Fig. 6. Procedure `indirectImps`.

to obtain the new set of implications (both direct and indirect). That is, we retain only the implications that are common to both $I_1(y)$ and $I_0(y)$. In addition, we may choose to save the essential implications separately by applying Eq. (4) to the new set of implications. Notice that we may recursively apply new orthogonal constraints to the set of transition relations to potentially find even more implications. This is handled easily within the BDD environment, as shown in the pseudocode of Figure 6.

The procedure `indirectImps` takes, as inputs, the assertion A and the relation sets T^{Sat} and T^{Obs}. In addition, it takes an input, *level*, representing the recursion level of the recursive call, initially set to 0. When *level* exceeds a specified limit *maxlevel*, the search for further implications is abandoned. Procedure `indirectImps` returns a cube, *impCube*, representing the set of all variables (direct and indirect) that are implied to constant values.

Note that an indirect implication discovered by propagating implications backwards is often found to be a direct implication by propagating implications forward. For example, the indirect satisfiability implication found in Figure 2, $(y = 0) \Rightarrow (x = 1)$ can be found as a direct implication $(x = 0) \Rightarrow (y = 1)$. Due to the bidirectional property of satisfiability implications, we know that this is the same implication. As previously mentioned, we make the distinction between indirect and direct implications only because it is often a good way of sorting out the more promising implications. Indeed, we may not want to eliminate an implication simply because it is not "indirectly obtained." If implications are found only by backward propagation, this may be a reasonable filter to use. However, if we are interested in observability-based implications as well, these can only be found in the forward direction, so using such a filter may not be a good solution. This point is discussed further in Section 5.

Using BDDs to compute and store indirect implications may seem inefficient compared to doing a simple analysis of the topology of a circuit. This may in fact be true if all we are interested in are single-variable implica-

tions derived from satisfiability assignments for single-literal assertions (for example, $(y_i = a) \Rightarrow (y_j = b)$ for $a, b \in \{0, 1\}$.) However, by computing the implications symbolically, we are better equipped for finding more general implications. That is, our procedure can store, manipulate, and compute general (i.e., more complex) expressions with similar complexity than if expressions were restricted to simple cubes (in fact, by separating the essential implications from the nonessential ones, both are available).

4. POWER OPTIMIZATION PROCEDURE

We now describe our implication-based optimization procedure for reducing power dissipation. The procedure consists of four main steps, described in detail in Sections 4.1 to 4.4.

4.1 Selecting an Assertion Function and Finding Its Implications

Computing all the indirect implications of a large network, as shown in Section 3, can be computationally expensive. Hence it is important to prune the search for implications by limiting the recursion level and carefully selecting the assertion function $A(y)$ upon which the implications are found. To reduce the cost even farther, in Bahar et al. [1996] we proposed extracting a subnetwork, and found the implications only within the confines of this subnetwork. Note that, although the implications can be found only within the boundaries of the subnetwork, all implications must hold in the context of the *entire* network.

Indirect implications are often present specifically in circuits containing *reconvergent fanout*. Reconvergent fanout is the presence of two or more distinct paths with a common input gate (or *fanout stem*), leading to a common output gate, and with no other gate in common. The gate where the paths reconnect is called the *reconvergence gate*. An example of a subnetwork with reconvergent fanout is shown in Figure 7. Two distinct paths from inputs c and d reconverge at gate $G9$.

The experiments in Bahar et al. [1996] suggested that to invest the time spent finding implications only where it is most useful, the search for indirect implications should be limited to subnetworks containing reconvergent fanout, where the reconvergence gate itself is used as the initial assertion $A(y)$. In this way, implications may be found through (predominantly) backward propagation of signal values toward the primary inputs.

While the above approach may work well if we are concerned with satisfiability implications only, it may be too limiting if observability implications are also to be exploited. Furthermore, as mentioned in Section 3, distinguishing between indirect and direct implications becomes a less useful filter in sorting out the more promising implications, since many of the observability ones are found through direct forward propagation of signal values.

Instead of asserting the reconvergent gate, our new strategy selects a gate with relatively low power dissipation (due to either low switching

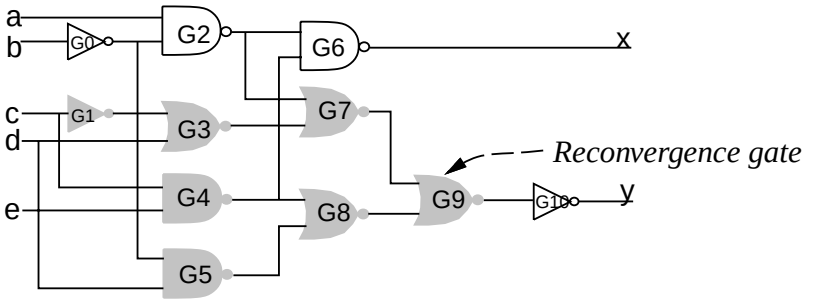

Fig. 7. A network of gates with an extracted subnetwork (shown in grey).

activity, low capacitive load, or both). Once a suitable implication is found, the low-power assertion gate is included in the added redundant logic. Using a low-power assertion gate has a minimal impact on potentially increasing its own power dissipation (switching activity and capacitive load are already low). In addition, using this signal as an input to other gates may have a dampening effect on the switching activity of other gates. For example, if an AND gate has a high switching activity, then connecting a signal that tends toward a 0 value most of the time (and switches infrequently) may prevent the output of the AND gate from switching as frequently. Moreover, these additions may allow the removal of other high-power connections or gates. Therefore, although the assertion gate's power is increased, the net result is an overall decrease in power consumption.

Once an assertion gate is selected, its output value is alternately set to both 0 and 1, and the implication procedure finds any relation that exists given one of the assertions. Since the assertion gate may exist anywhere within the network (or subnetwork), values will be propagated both "backward" and "forward" through the logic.

4.2 Finding the Right Addition

Once we have found the implications for the given assertions on the selected gate, we can use this information to add gates and/or connections to the circuit while retaining the behavior of the original one at the primary outputs. We use a method similar to *dataflow* analysis [Trevillyan et al. 1986] to determine what these modifications are for a given assertion $(x = i)$ and its implication $(y = j)$ for $i, j \in \{0, 1\}$, where the implication gate y is in the transitive fanin of the assertion gate x.

Consider first the case where $(x = 0) \Rightarrow (y = 0)$. This implication can also be expressed as $x' \cdot y = 0$. Given the function $F(x) = x$, the implication can be expressed as the don't care condition, $F(x)_{DC}$, for $F(x)$. That is, $x' \cdot y \in F(x)_{DC}$. We may transform F to $\tilde{F}$ by adding this don't care term to the output of F without changing the behavior at the primary outputs of the circuit:

$$\tilde{F}(x) = F(x) + x' \cdot y = x + y = x, \text{ since } y = 0 \text{ if } x = 0 \tag{8}$$

In other words, the original circuit is modified by ORing the don't care term (i.e., the implicant gate y) with the output of gate x. Similarly,

$$\tilde{F}(x) = x + y' \text{ if } x = 0, y = 1 \tag{9}$$

For the case $(x = 1) \Rightarrow (y = 1)$, instead of using the don't care expression $x \cdot y' \in F(x)_{DC}$, or $x \cdot y' = 0$, we use the analogous expression $x' + y = 1$ and transform F to $\tilde{F}$ as

$$\tilde{F}(x) = F(x) \cdot (x' + y) = x \cdot y = x, \text{ since } y = 1 \text{ if } x = 1 \tag{10}$$

In other words, the circuit is modified by ANDing the implicant gate y with the output of gate x.

As an example of how this method is applied, refer back to the circuit in Figures 1(a) and (b). We show the additional connection added due to the implication $(G9 = 0) \Rightarrow (G6 = 1)$. According to the implication, we can modify the function at the output of gate $G9$ without changing the behavior of the circuit by inserting the OR function $\widetilde{G9} = G9 + G6'$ in the circuit. Notice that this OR gate added to the network is "absorbed" by the inverter $G11$, which now becomes a 2-input NOR gate. Note that it is essential that the assertion gate not be in the transitive fanin of the implication, since replacing the function $F(x)$ with $\tilde{F}(x)$ would create a cyclic network.

4.3 Finding and Removing Redundancies

Once the redundant circuitry is added, we use the automatic test pattern generation (ATPG) procedure implemented in SIS [Sentovich et al. 1992] to find the new redundancies created in the network. Whether implications are found using the entire network or only within the boundaries of a subnetwork, finding and removing redundancies should be done on the *entire* network.

We generate a list of possibly redundant connections. Since the newly added gates are themselves redundant, we need to make sure that they are not included in the list. The result of redundancy removal is order dependent; removing a redundant connection from a network may create new redundancies, and/or make existing ones no longer redundant. Since our primary objective is reducing power dissipation, we sort the redundant connections in order of decreasing power dissipation and remove them starting from the top of the list.

After ATPG, the identified redundant faults can be removed with the ultimate goal of eliminating fanout connections from the targeted high-power dissipating node. Redundancy removal procedures such as the one implemented in SIS cannot be used for this purpose for two main reasons. First, optimization occurs through restructuring of the Boolean network. As

a consequence, even if redundancy removal operates on a technology mapped design, the end result of the optimization is a technology-independent description that requires remapping onto the target gate library. This may lead to significant changes in the structure of the original network. This is undesirable in the context of low-power resynthesis, since the network transformations made during resynthesis are based on the original circuit implementation. Second, redundancy removal usually targets area minimization, and this may obviously affect circuit performance.

We have implemented our own redundancy removal algorithm, which resembles the `Sweep` procedure implemented in SIS, but operates on the gates of a circuit rather than on the nodes of a Boolean network. In addition, it performs a limited number of transformations. Namely, the procedure (a) simplifies gates whose inputs are constant, and (b) collapses inverter chains only when the original circuit structure and performance are preserved.

4.3.1 *Gate Simplification*. Three simplifications are applicable to a given gate, G, when one of its inputs is constant:

(1) If the constant value is a controlling value for G, then G is replaced by a connection to either V_{dd} or *Ground*, depending on the function of the gate.

(2) If the constant value is a noncontrolling value for G, and G has more than two inputs, then G is replaced by a gate, $\tilde{G}$, taken from the library and implementing the same logic function as G but with one less input.

(3) If the constant value is a noncontrolling value for G, and G is a two-input gate, then G is replaced by an inverter or buffer.

Cell libraries usually contain several gates implementing the same function, but differing in size, and therefore, in their delays, loads, and driving capabilities. We select, as replacement gate $\tilde{G}$, the gate that has approximately the same driving strength as the original gate G.

4.3.2 *Collapsing Inverter Chains*. Inverter chains are commonly encountered in circuits, especially where speed is critical. Collapsing inverters belonging to these "speed-up" chains, though advantageous from the point of view of area and,possibly power, may have a detrimental effect on the performance of the circuit. On the other hand, the simplification of gates due to redundancy removal may produce inverter chains that may be easily eliminated without slowing down the network. We eliminate inverter chains only in the cases where the transformation does not increase the critical delay of the original circuit. For each inverter, $\tilde{G}$, obtained through simplification of a more complex gate, we first check if $\tilde{G}$ belongs to a chain that can be eliminated. If certain constraints are satisfied, both the inverter $\tilde{G}$ and the companion inverter in the chain (i.e., the inverter

feeding $\tilde{G}$ or the inverter fed by $\tilde{G}$) are removed. In particular, in order to safely remove the inverter chain:

(1) The first inverter cannot have multiple fanouts.

(2) The load at the output of the inverter chain must not be greater than the load currently seen at the gate preceding the inverter chain.

The first restriction may be unnecessarily conservative; however, removing it implies that sometimes extra inverters need to be inserted on some of the fanout branches of the first inverter, thereby possibly introducing area, power, and delay degradation.

4.4 Choosing the Best Network

Adding redundant gates and connections to a circuit may increase area, delay, and power consumption by an amount that may not be recoverable by the subsequent step of redundancy removal. Given an assertion, a network is created for each literal in the implication cube that we elected to save while running the implication procedure. (We may choose to eliminate an implication from the list of possible candidates because it will create a cyclic network or may add connections to an already high-dissipating node.) This new network is obtained by adding the appropriate redundant logic according to the chosen implication (Section 4) and finding and removing the newly created redundancies (Section 4). Power and delay estimations are then run on each new network. The best network is selected from them and used to replace the existing network. The criteria we have used to carry out the network selection are based on a combination of delay and power consumption, and are discussed in detail in Section 5.

5. EXPERIMENTAL RESULTS

In this section we present the results obtained by applying our optimization procedure to some combinational circuits from the `Mcnc'91` benchmark suite. Experiments were run within the SIS environment on a SUN UltraSparc 170 workstation with 300 MB of memory.

The circuits are initially optimized using the `SIS` script `script.rugged` and mapped for either area (using `map`) or for delay (using `map -n 1-AFG`). The library used to map the circuits contains NAND, NOR, and inverter gates, each of which allows up to 4 inputs and 5 drive options. In general, gates with larger drive strength have larger cell areas, however, these two values do not increase at the same rate. After mapping, the Bahar et al. [1994] method is used to resize gates with smaller gates where no circuit delay penalty is incurred. This ensures that any gain made during the experiment is the result of our optimization procedure and not of an improperly sized gate. The statistics for these circuits are reported in Table I. In particular, the number of gates, the area (in μm^2), the delay (in $nsec$), and the power consumption (in μW) are shown. Power dissipation is estimated using the symbolic simulation method of Ghosh et al. [1992].

Table I. Circuit Statistics Before Optimization

Circuit	*Initial Statistics (mapping for area)*				*Initial Statistics (mapping for speed)*			
	Gates	*Area*	*Delay*	*Power*	*Gates*	*Area*	*Delay*	*Power*
5xp1	101	140592	29.52	424	154	226780	17.90	1099
9sym	159	236176	17.79	629	276	404840	10.95	1743
b12	69	94192	10.04	266	97	133748	8.07	495
bw	124	172608	32.65	527	206	288144	17.92	1269
clip	104	146624	17.43	427	167	249632	12.29	1205
inc	83	117856	23.96	356	128	178408	16.45	638
misex1	50	66352	13.79	192	76	111244	9.94	525
misex2	74	104864	10.26	272	133	183164	7.55	552
rd53	33	44080	10.52	150	47	65192	8.52	287
rd73	60	80272	18.76	253	71	101384	14.77	414
rd84	128	175856	18.23	482	185	265524	13.07	1127
sao2	100	142912	19.19	370	172	246036	13.68	963
squar5	53	71456	20.64	240	80	116580	15.04	542
C432	135	196736	34.51	617	249	364472	21.72	1598
C880	343	474672	45.37	1154	468	643916	28.77	1998
alu4	169	234784	19.93	628	247	344056	14.06	1391
cordic	67	90944	11.64	280	85	123192	9.53	474
cps	897	128620	40.09	1932	1272	1694412	13.68	2772

For each set of experiments, our optimization procedure was iteratively applied to the circuits to find implications to be used for redundancy addition and removal. After this step, gates in the circuit are again resized without increasing the critical delay.

5.1 Setting Delay and Power Constraints

The first set of experiments were run to determine how to choose a new network among a choice of several. That is, how to choose the best implication to apply to the network. As in Bahar et al. [1996], network choices may be based solely on which one has the lowest power dissipation with the constraint that the delay of the new network has not increased by more than a fixed percentage (usually 5%). A more robust approach may use a combination of delay and power to select the best network, or may temporarily allow power to increase so that more powerful implications may be applied subsequently, thus having a greater impact on reducing power dissipation. These experiments are discussed in the following sections.

5.1.1 *Power Threshold*. One optimization already mentioned in Section 4 is the addition of a power threshold. The basis of a power threshold is the observation that if the difference in power of two networks is small, better results are obtained by choosing the network with the smaller delay. Then, as a fall back, if the delays of the two networks are equal, the difference in power (no matter how small) is used to make the determination. This heuristic takes advantage of the fact that the final network will be resized based on the delay of the original network. A large improvement in the delay gives the resizing algorithm the ability to make significant additional

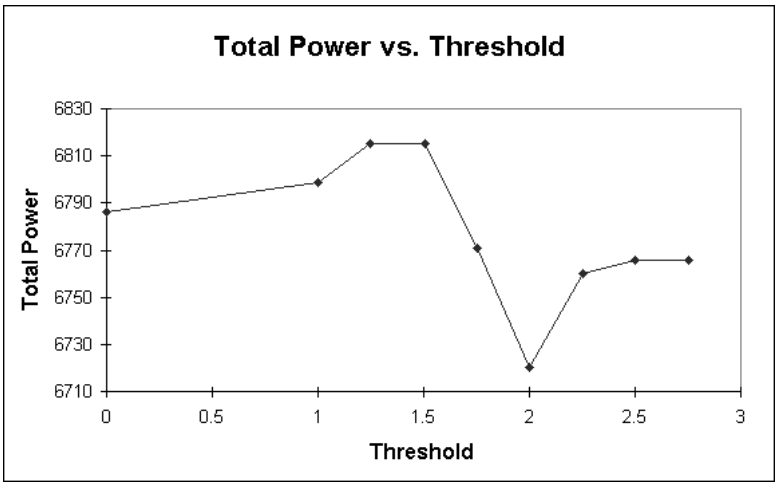

Fig. 8. Power dissipation for given threshold value.

power gains in the layout of the transistors. It also allows transformations that may not have been accepted previously because they increased the delay too much.

A set of experiments was done to determine the value of an optimal threshold value. Tests were done with values of 1.0 to 2.75 μW at intervals of 0.25 μW on all tested circuits for an area mapping. A run was also done at a threshold of 0, which is a run based on power alone. Figure 8 shows an optimal threshold value of 2 μW. This result is reasonable since, as discussed earlier in the paper, the final power is affected by two variables, power and delay. Also, allowing the power to increase slightly may create new implications that lead to even greater decreases in power dissipation.

5.1.2 *Delay Tolerance*. Delay tolerance, defined as the allowable percent increase in the delay of the final circuit from the original circuit, is another parameter that can be varied. The interesting observation is that raising the delay tolerance does not always result in a slower final network. This is because there are often many transformation choices made before the final network is obtained. Transformations that increase the delay are often offset by other transformations that decrease it. Yet the increase in the delay tolerance increases the number of networks to choose from. In other words, similarly to the power threshold, increasing the delay tolerance increases the probability that a power-saving implication will be found.

There are several observations that can be made from the data. First, increasing the delay tolerance does, on average, have the effect of increasing the delay. However, depending on the circuit, allowing more flexibility with delay per iteration can allow us to obtain a final circuit that is both lower in power and faster than the original circuit. Second, greater success was found when testing delay-mapped circuits compared to the area-

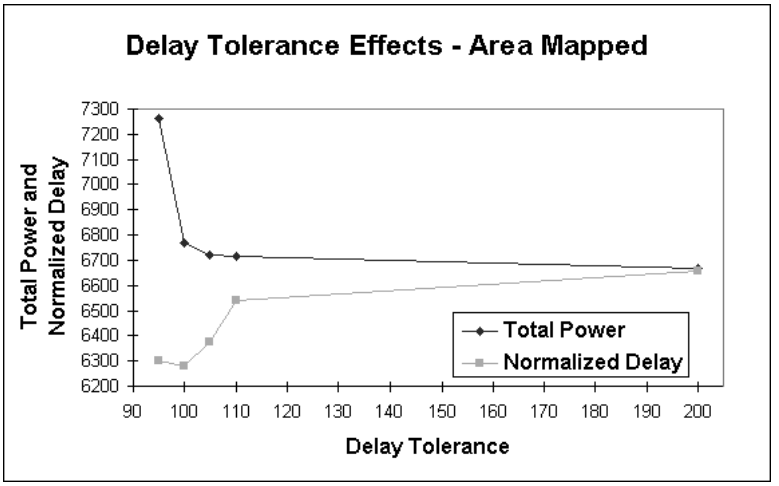

Fig. 9. Power/delay trade-off curves for varying delay tolerance (circuits mapped for area).

mapped circuits. This is reasonable, since the delay-mapped circuits are by definition designed to achieve minimum delay. From this extreme, a small sacrifice in delay produces a relatively large power savings over area-mapped circuits.

The results of the experiments are shown in Figures 9 and 10. We can see that for an increase in delay tolerance, the average delay does go up. Also, after a certain delay tolerance level, we can see that further decreases in power are small to insignificant (on the graphs, a delay tolerance of 200 can be interpreted as infinite.)

5.2 Individual Experiments

From the results in the first set of experiments, we now show the individual power, delay, and area characteristics for each circuit after redundancy addition and removal and resizing is complete. For all these experiments, the recursion level for finding implications was limited to 1 (i.e., *maxlevel = 1* in Figure 6). Implications were applied using a power threshold of $2\mu W$ and a delay tolerance of 5% above the original circuit delay. As before, after redundancy addition and removal, gates in the circuit are resized without increasing the critical delay. That is, the critical delay of the final circuits was never greater than 5% above that reported in Table I. Tables II–V give the final statistics for the circuits. In Tables II and IV we report the results after applying the redundancy addition and removal (Table II starts with circuits mapped for area and Table IV starts with circuits mapped for speed). The number of accepted implications is shown in the column labeled *Imps*. Of these, the *Obs* column shows how many of them are observability implications. The relative changes in power, delay, and area are shown in the columns labeled ΔP, ΔD and ΔA (e.g., a 0.75 in the ΔP column indicates a 25% reduction in power compared to that given in Table I). The results after the final step in gate resizing are shown in Tables III and V,

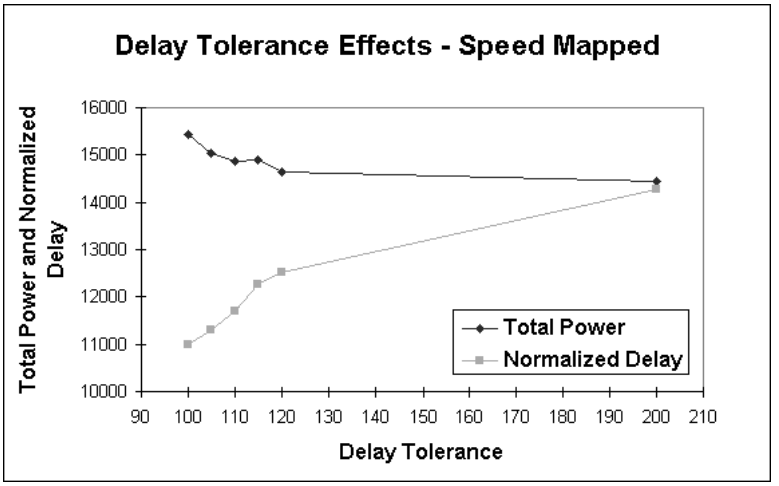

Fig. 10. Power/delay trade-off curves for varying delay tolerance (circuits mapped for speed).

Table II. Statistics After Redundancy Addition and Removal (Circuits Mapped for Area)

	Gates	*Area*	*Delay*	*Power*	*Imps*	*Obs*	ΔP	ΔD	ΔA
5xp1	86	109968	26.23	274	58	20	0.65	0.89	0.78
9sym	178	241744	18.34	483	98	48	0.77	1.03	1.02
b12	76	97440	10.14	216	46	18	0.81	1.01	1.03
bw	132	170752	23.56	385	85	11	0.73	0.72	0.99
clip	88	117392	15.48	274	43	14	0.64	0.89	0.80
inc	97	127136	22.06	259	67	24	0.73	0.92	1.08
misex1	50	64032	13.12	142	33	13	0.74	0.95	0.97
misex2	76	105328	10.44	188	45	13	0.69	1.02	1.00
rd53	33	40832	10.93	111	15	3	0.74	1.04	0.93
rd73	60	77488	19.65	201	31	6	0.79	1.05	0.97
rd84	109	139664	17.76	290	60	20	0.60	0.97	0.79
sao2	102	139200	19.19	275	52	18	0.74	1.00	0.97
squar5	53	67280	16.47	168	38	11	0.70	0.80	0.94
C432	149	199056	35.09	395	100	32	0.64	1.02	1.01
C880	359	484880	43.22	1066	94	27	0.92	0.95	1.02
alu4	126	163792	19.38	354	74	28	0.56	0.97	0.70
cordic	60	79344	12.12	203	23	12	0.72	1.04	0.87
Average							0.72	0.96	0.93

respectively. Changes in power, delay, and area are given relative to those shown in Table I.

Notice that for circuits shown in Tables II and III, the change in area remain the same before and after gate resizing. Since these circuits were originally mapped for area optimization, most of the gates were already at or near minimum size. Therefore, even after resizing, no additional saving in area is possible, However, a slight improvement in power and delay is still possible with resizing since, in our library, cell areas for some gates are the same for different drive strengths.

Table III. Statistics After Resizing Gates from Table II. Changes in power, delay, and area relative to those in Table I, Cols. 2-5

	Area	*Delay*	*Power*	ΔP	ΔD	ΔA
5xp1	109968	25.58	242	0.57	0.87	0.78
9sym	241744	18.06	470	0.75	1.02	1.02
b12	97440	10.08	213	0.80	1.00	1.03
bw	170752	25.31	363	0.69	0.78	0.99
clip	117392	15.44	255	0.60	0.89	0.80
inc	127136	22.32	236	0.66	0.93	1.08
misex1	64032	13.72	123	0.64	0.99	0.97
misex2	105328	10.40	180	0.66	1.01	1.00
rd53	40832	10.93	111	0.74	1.04	0.93
rd73	77488	19.65	201	0.79	1.05	0.97
rd84	139664	17.98	246	0.51	0.99	0.79
sao2	139200	19.09	269	0.72	0.99	0.97
squar5	67280	16.32	150	0.62	0.79	0.94
C432	199056	35.01	388	0.63	1.01	1.01
C880	484880	45.10	1040	0.90	0.99	1.02
alu4	163792	19.44	346	0.55	0.98	0.70
cordic	79344	12.12	197	0.70	1.04	0.87
Average				0.68	0.96	0.93

Table IV. Statistics After Redundancy Addition and Removal (Circuits Mapped for Speed)

	Gates	*Area*	*Delay*	*Power*	*Imps*	*Obs*	ΔP	ΔD	ΔA
5xp1	115	164604	18.47	656	77	26	0.60	1.03	0.73
9sym	242	358904	11.48	1407	123	47	0.81	1.05	0.89
b12	78	108228	8.42	368	49	14	0.74	1.04	0.81
bw	176	235248	17.92	781	127	33	0.62	1.00	0.82
clip	134	195692	12.50	828	71	24	0.69	1.02	0.78
inc	115	158224	16.09	450	71	16	0.71	0.98	0.89
misex1	68	94656	10.38	370	43	19	0.70	1.04	0.85
misex2	113	160660	7.82	433	65	26	0.78	1.04	0.88
rd53	36	51156	8.89	191	23	8	0.67	1.04	0.78
rd73	62	91292	15.50	383	15	4	0.93	1.05	0.90
rd84	136	192096	13.72	753	77	32	0.67	1.05	0.72
sao2	155	219240	14.34	700	78	34	0.73	1.05	0.89
squar5	66	90944	14.06	330	46	14	0.61	0.93	0.78
C432	208	311112	22.59	1284	101	31	0.80	1.04	0.85
C880	432	598560	30.07	1743	163	75	0.87	1.05	0.93
alu4	172	241860	13.99	932	87	27	0.67	1.00	0.70
cordic	75	109040	9.89	420	22	11	0.89	1.04	0.89
Average							0.73	1.03	0.83

The effectiveness of our method is shown by the results. For example, in the case of mapping for area, a 49% power reduction was obtained for circuit `rd84`. On the other hand, in the case of circuits mapped for speed, a 64% power savings was obtained for benchmark `bw`; on average, a power savings of 34% was obtained for area and speed mapped circuits combined.

It is interesting to point out the relationship between power reduction and circuit delay and area. While the circuits averaged a 36% reduction in

Table V. Statistics After Resizing Gates from Table IV. Changes in Power, Delay, and Area Relative to Those in Table I, cols. 6-9

	Area	*Delay*	*Power*	ΔP	ΔD	ΔA
5xp1	158920	18.13	572	0.52	1.01	0.70
9sym	354844	11.48	1351	0.78	1.05	0.88
b12	105676	8.32	312	0.63	1.03	0.79
bw	213788	17.80	463	0.36	0.99	0.74
clip	170752	12.21	496	0.41	0.99	0.68
inc	146160	16.37	295	0.46	1.00	0.82
misex1	92336	10.27	331	0.63	1.03	0.83
misex2	158108	7.80	423	0.77	1.03	0.86
rd53	50924	8.82	187	0.65	1.04	0.78
rd73	91292	15.50	383	0.93	1.05	0.90
rd84	186760	13.72	646	0.57	1.05	0.70
sao2	217152	14.19	673	0.70	1.04	0.88
squar5	84448	14.92	248	0.46	0.99	0.72
C432	311112	22.59	1282	0.80	1.04	0.85
C880	597400	29.92	1719	0.86	1.04	0.93
alu4	228288	13.99	690	0.50	1.00	0.66
cordic	109040	9.89	420	0.89	1.04	0.89
Average				0.64	1.02	0.80

power, delay increased by only 2% for speed-mapped circuits and *decreased* by 4% for area-mapped circuits. However, for a few examples, delay decreased significantly. For instance, in Table III, circuit `bw` mapped for area showed a 31% decrease in power along with a 22% decrease in delay. These results help emphasize that low power does not need to always come at the expense of reduced performance. In addition, it is not always the case that smaller devices must be used to obtain lower-power dissipation. For example, for circuit `C432` in Table III, area increased by 1%, but power dissipation decreased by 37%. This result emphasizes the need to consider switching activity when optimizing for power.

6. CONCLUSIONS AND FUTURE WORK

To be successful, power optimization needs to be driven by power analysis. By performing power analysis directly on a technology-mapped circuit, we can selectively target the search for implications to areas (i.e., assertion functions) that indicate promise in reducing power dissipation. Starting from an appropriate assertion gate, low-power minimization is achieved through network transformations, which are based on implications obtained using symbolic, BDD-based computation. Our method has been shown to obtain up to a 64% decrease in power dissipation and an average of 34% power reduction for all circuits.

Although all circuits presented in the results were of small to moderate size, this method can be expanded to larger circuits as well. With larger-sized circuits, however, execution time may increase significantly. The execution bottleneck is not the BDD-based algorithm to find the implications, but rather the ATPG algorithm within SIS used to identify redun-

dant connections. Other redundancy identification and removal techniques, such as those in Iyer and Abramovici [1994] may be used to alleviate this bottleneck.

For future work, we would like to take advantage of the more general implications mentioned in our symbolic algorithm. This enhancement should allow us to reduce power dissipation further. In addition, we are working on identifying more powerful transformations (other than simple inverter chain-collapsing), which may be applied to the circuit in order to further reduce area and power at no delay cost. These transformations may include application of DeMorgan's law or expanding the types of gates included in our library. For instance, it may be advantageous to collapse, say, a NAND and an inverter into an AND, or NAND/inverter clusters into complex gates. Finally, we are working on refining our method of selecting an assertion gate and finding a suitable subnetwork over which to search for implications.

ACKNOWLEDGMENTS

We would like to thank Fabio Somenzi and Gary Hachtel for their many helpful comments and suggestions on the first draft of this paper.

REFERENCES

BAHAR, R., BURNS, M., HACHTEL, G., MACII, E., SHIN, H., AND SOMENZI, F. 1996. Symbolic computation of logic implications for technology-dependent low-power synthesis. In *Proceedings of the 1996 International Symposium on Low Power Electronics and Design* (ISLPED '96, Monterey, CA, Aug 12–14), M. Horowitz, J. Rabaey, B. Barton, and M. Pedram, Eds. IEEE Press, Piscataway, NJ, 163–168.

BAHAR, R. I., HACHTEL, G. D., MACII, E., AND SOMENZI, F. 1994. A symbolic method to reduce power consumption of circuits containing false paths. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '94, San Jose, CA, Nov. 6–10, 1994), J. A. G. Jess and R. Rudell, Eds. IEEE Computer Society Press, Los Alamitos, CA, 368–371.

BAHAR, R. I. AND SOMENZI, F. 1995. Boolean techniques for low power driven re-synthesis. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 428–432.

BRYANT, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput. C-35*, 8 (Aug. 1986), 677–691.

CHANDRAKASAN, A. P. AND BRODERSEN, R. W. 1995. Minimizing power consumption in digital cmos circuits. *Proc. IEEE 83*, 4 (Apr.), 498–523.

CHANDRAKASAN, A. P., POTKONJAK, M., MEHRA, R., RABEY, J., AND BRODERSON, R. W. 1995. Optimizing power using transformations. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 14*, 1 (Jan. 1995), 12–31.

CHANG, S.-C. AND MAREK-SADOWSKA, M. 1994. Perturb and simplify: Multi-level boolean network optimizer. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '94, San Jose, CA, Nov. 6–10, 1994), J. A. G. Jess and R. Rudell, Eds. IEEE Computer Society Press, Los Alamitos, CA, 2–5.

CHENG, K. T. AND ENTRENA, L. 1993. Multi-level logic optimization by redundancy addition and removal. In *Proceedings of the IEEE European Conference on Design Automation* (EURO-DAC '93), IEEE Computer Society Press, Los Alamitos, CA, 373–377.

CHO, H., HACHTEL, G. D., AND SOMENZI, F. 1993. Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration. *IEEE Trans. Comput.-Aided Des. 12*, 7 (July), 935–945.

COUDERT, O., HADDAD, R., AND MANNE, S. 1996. New algorithms for gate sizing: a comparative study. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 734–739.

ENTRENA, L. AND CHENG, K.-T. 1993. Sequential logic optimization by redundancy addition and removal. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD '93, Santa Clara, CA, Nov. 7–11), M. Lightner and J. A. G. Jess, Eds. IEEE Computer Society Press, Los Alamitos, CA, 310–315.

GHOSH, A., DEVADAS, S., KEUTZER, K., AND WHITE, J. 1992. Estimation of average switching activity in combinational and sequential circuits. In *Proceedings of the 29th ACM/IEEE Conference on Design Automation* (DAC '92, Anaheim, CA, June 8-12), D. G. Schweikert, Ed. IEEE Computer Society Press, Los Alamitos, CA, 253–259.

IMAN, S. AND PEDRAM, M. 1994. Multi-level network optimization for low power. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '94, San Jose, CA, Nov. 6–10, 1994), J. A. G. Jess and R. Rudell, Eds. IEEE Computer Society Press, Los Alamitos, CA, 372–377.

IMAN, S. AND PEDRAM, M. 1995a. Logic extraction and factorization for low power. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 248–253.

IMAN, S. AND PEDRAM, M. 1995b. Two-level logic minimization for low power. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 433–438.

IYER, M. A. AND ABRAMOVICI, M. 1994. Low-cost redundancy identification for combinational circuits. In *Proceedings of the 7th International Conference on VLSI Design* (India, Jan. 1994), 315–318.

JAIN, J., MUKHERJEE, R., AND FUJITA, M. 1995. Advanced verification techniques based on learning. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 420–426.

KEUTZER, K., MALIK, S., AND SALDANHA, A. 1990. Is redundancy necessary to reduce delay. In *Proceedings of the ACM/IEEE Conference on Design Automation* (DAC '90, Orlando, FL, June 24-28), R. C. Smith, Ed. ACM Press, New York, NY, 228–234.

KUNZ, W. 1993. HANNIBAL: an efficient tool for logic verification based on recursive learning. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD '93, Santa Clara, CA, Nov. 7–11), M. Lightner and J. A. G. Jess, Eds. IEEE Computer Society Press, Los Alamitos, CA, 538–543.

KUNZ, W. AND MENON, P. R. 1994. Multi-level logic optimization by implication analysis. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '94, San Jose, CA, Nov. 6–10, 1994), J. A. G. Jess and R. Rudell, Eds. IEEE Computer Society Press, Los Alamitos, CA, 6–13.

KUNZ, W. AND PRADHAN, D. K. 1992. Recursive learning: An attractive alternative to the decision tree for test generation in digital circuits. In *Proceedings of the on IEEE International Test Conference* (Sept. 1992), IEEE Computer Society Press, Los Alamitos, CA, 816–825.

LIN, B. AND DE MAN, H. 1993. Low-power driven technology mapping under timing constraints. In *Proceedings of the 1993 Conference on Computer Design* (ICCD '93, Cambridge, MA, Oct. 3-6, 1993), IEEE Computer Society Press, Los Alamitos, CA, 421–427.

MACII, E., PEDRAM, M., AND SOMENZI, F. 1997. High-level power modeling, estimation, and optimization. In *Proceedings of the 34th Annual Conference on Design Automation* (DAC '97, Anaheim, CA, June 9–13), E. J. Yoffa, G. De Micheli, and J. M. Rabaey, Eds. ACM Press, New York, NY, 504–511.

ROHFLEISCH, B., KÖLBL, A., AND WURTH, B. 1996. Reducing power dissipation after technology mapping by structural transformations. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 789–794.

ROHFLEISCH, B., WURTH, B., AND ANTREICH, K. 1995. Logic clause analysis for delay optimization. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 668–672.

ROY, K. AND PRASAD, S. C. 1992. Syclop: Synthesis of CMOS logic for low power applications. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 1992), IEEE Computer Society Press, Los Alamitos, CA, 464–467.

SENTOVICH, E. M., SINGH, K. J., MOON, C. W., SAVOJ, H., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1992. Sequential circuits design using synthesis and optimization. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 1992), IEEE Computer Society Press, Los Alamitos, CA, 328–333.

SHEN, A., GHOSH, A., DEVADAS, S., AND KEUTZER, K. 1992. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '92, Santa Clara, CA, Nov. 8–12), L. Trevillyan, Ed. IEEE Computer Society Press, Los Alamitos, CA, 402–407.

TIWARI, V., ASHAR, P., AND MALIK, S. 1993. Technology mapping for low power. In *Proceedings of the 30th ACM/IEEE International Conference on Design Automation* (DAC '93, Dallas, TX, June 14–18), A. E. Dunlop, Ed. ACM Press, New York, NY, 74–79.

TREVILLYAN, L., JOYNER, W., AND BERMAN, L. 1986. Global flow analysis in automatic logic design. *IEEE Trans. Comput. C-35*, 1 (Jan. 1986), 77–81.

TSUI, C.-Y., PEDRAM, M., AND DESPAIN, A. M. 1993. Technology decomposition and mapping targeting low power dissipation. In *Proceedings of the 30th ACM/IEEE International Conference on Design Automation* (DAC '93, Dallas, TX, June 14–18), A. E. Dunlop, Ed. ACM Press, New York, NY, 68–73.

VUILLOD, P., BENINI, L., AND DE MICHELI, G. 1997. Re-mapping for low power under tight timing constraints. In *Proceedings of the 1997 International Symposium on Low Power Electronics and Design* (ISLPED '97, Monterey, CA, Aug. 18–20), B. Barton, M. Pedram, A. Chandrakasan, and S. Kiaei, Eds. ACM Press, New York, NY, 287–292.

YANG, S. 1991. Logic synthesis and optimization benchmarks user guide version 3.0. Microelectronics Center of North Carolina, Research Triangle Park, NC.

Received: September 1997; accepted: December 1998