# Fast Postplacement Optimization Using Functional Symmetries

Chih-Wei (Jim) Chang, Ming-Fu Hsiao, Bo Hu, Kai Wang, Malgorzata Marek-Sadowska, *Fellow, IEEE*, Chung-Kuan Cheng, *Fellow, IEEE*, and Sao-Jie Chen

***Abstract*—The timing-convergence problem arises because estimations made during logic synthesis may not be met during physical design. In this paper, an efficient rewiring engine is proposed to explore maximal freedom after placement. The most important feature of this approach is that the existing placement solution is left intact throughout the optimization. A linear-time algorithm is proposed to detect functional symmetries in the Boolean network which are then used as the basis for rewiring. Integration with an existing gate-sizing algorithm further proves the effectiveness of our technique. Three applications are demonstrated: delay, power, and reliability optimization.**

***Index Terms*—Functional symmetry, logic restructuring, logic synthesis, physical synthesis, postlayout optimization, timing closure.**

## I. Introduction

IN THIS PAPER, we present a technique focusing on the identification of symmetries when the target Boolean function is represented as a mapped Boolean network. In general, when two wires are functionally symmetric, they can be swapped without changing the overall circuit functionality. We have established a relationship between implication supergate [24] and functional symmetry. Based on our analysis of supergates, we propose a linear-time algorithm for symmetry identification in a multilevel netlist. We have developed efficient postplacement performance-improvement algorithms which apply symmetry-based rewiring and gate sizing.

Power consumption and speed are two primary cost functions in today's integrated circuit design. As mobile computation devices prevail in the market, the ability to design fast, low-power devices is of paramount importance. However, these two objectives often conflict: a faster circuit consumes more power, but a low-power circuit runs slower. Hence, designers often need to tradeoff power for speed and vice versa to meet the desired specifications. To get the best performance, power and speed are considered at various stages of the design cycle, including architecture, register-transfer level, gate, and layout levels.

Circuit reliability is another emerging design consideration. Design trends, such as device miniaturization, system-on-a-chip integration, and higher operating frequencies, increase concerns about circuit reliability. Hot-carrier effect (HCE) is one of the major failure mechanisms affecting long-term reliability. As the device dimensions shrink to the deep submicron ranges, the electric field in a transistor's channel increases significantly. Electrons and holes traveling in the channel may gain high enough kinetic energy to be injected into the gate oxide and cause permanent changes in the oxide-interface charge distribution. In an NMOS transistor, HCE leads to transconductance degradation, shift in the threshold voltage, and decrease in the drain-current driving capability. The performance degradation of particular devices leads to degradation in the overall circuit performance. The transistor degradation behavior is a function of time, the number of transitions, its fanins' driving capability, and geometric dimensions. The effects accumulate while the device is in operation. As a result, circuits age.

Our proposed rewiring technique will be used to target delay, power, and reliability optimization specifically at the postplacement level when the gate level netlist is already placed on a two-dimensional plane. The rationale behind this methodology is that delay, power, and reliability cannot be determined without physical information. The optimization effort could be misled by inaccurate estimation of the objective function. However, since the placement is already done, optimization techniques used at this level must not perturb the existing placement solution too much, in order to guarantee timing closure. Buffer insertion and gate sizing are traditionally the only two techniques that are suitable for this purpose. However, these two techniques are limited to the existing netlist, not enabling us to explore a larger solution space by restructuring the logic. In recent years, rewiring techniques, such as redundancy-addition-and-removal [11] have been successfully applied at postlayout stages to restructure the logic. This technique has the property that only wires are reconnected without disturbing the existing placement solution. This important property allows logic changes to be guided by accurate delay information. Our proposed functional symmetry-based rewiring technique explores another degree of freedom compared with the existing techniques.

## II. Preliminaries

A Boolean network [1] is a *directed acyclic graph* (DAG) $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set

Manuscript received August 24, 2001; revised February 25, 2002, July 9, 2002, and March 7, 2003. This paper was recommended by Associate Editor M. Sarrafzadeh.

C.-W. Chang is with Cadence Design Systems, San Jose, CA 95134 USA (e-mail: cwchang1@yahoo.com).

M.-F. Hsiao is with the Faraday Technology Corporation, Sunnyvale, CA 94085 USA.

B. Hu, K. Wang, and M. Marek-Sadowska are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA.

C.-K. Cheng is with the Department of Computer Science and Engineering, University of California, San Deigo, La Jolla, CA 92093 USA.

S.-J. Chen is with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan.

Digital Object Identifier 10.1109/TCAD.2003.819904

of edges connecting vertices. Let $v$ be a vertex in $V$. The (immediate) *fanout* of $v$ is a set of nodes $u$ such that there is an edge from $v$ to $u$. Similarly, the (immediate) *fanin* of $v$ is a set of nodes $k$, such that there is an edge from $k$ to $v$. A node $v$ is called an *internal node* if it is neither a primary input (PI) nor a primary output (PO). The *fanout cone* of $v$, or *transitive fanout* of $v$, is a set of nodes $u$, such that there exists a directed path from $v$ to $u$. Similarly, the *fanin cone* of $v$, or *transitive fanin* of $v$, is a set of nodes $k$ such that there exists a directed path from $k$ to $v$. The terms "vertex" and "node" are used interchangeably.

Each vertex $v$ has a function $f$ associated with it. Function $f$ maps the Boolean space spanned by its fanins to the space spanned by its fanouts. In practice, we are interested only in the case when $f$ is a single-output function. All fanouts of $v$ are fed by the same $f$. Even though $f$ is a single-output function, the number of fanouts of $v$ can be more than one.

Let $L$ be a library of logic gates. A Boolean network is *mapped* when the function associated with each vertex is implemented by a gate from $L$. Otherwise, it is *unmapped*. Let $g$ be a gate in $L$. An *in-pin* of $g$ is the connector of $g$ to which outputs of other gates can connect. An *out-pin* of $g$ is the connector that can drive other gates' in-pins. The logic type of $g$ is denoted type$(g)$. We do not distinguish between the name of a gate and its *out-pin*.

Let $T$ be a fanout-free network rooted at $f$. By fanout-free, we mean that each node inside $T$ has only a single fanout. A *path* is an alternating sequence of pins and gates such that for any two consecutive gates the former is driving the latter. For example, a path $P = (p_0, g_0, \ldots, p_{n-1}, g_{n-1})$ starts from the in-pin $p_0$ of gate $g_0$ and ends at the in-pin $p_{n-1}$ of gate $g_{n-1}$. A path is fanout-free if all gates along the path are fanout-free. An input $p_j$ to a gate $g_i$ on the path is a *side input* if the gate driving $p_j$ is not on the path.

*Definition 1:* An *input-controlling value* of an in-pin $p$ of a gate $g$, denoted $\mathrm{icv}(p)$, is the logic value which, when set at $p$, uniquely determines the output of $g$ regardless of the logic values on other inputs. An *output-controlled value* of a gate $g$, denoted $\mathrm{ocv}(g)$, is $g$'s output value when one of its input pins is set to its input controlling value.

For example, when type$(g)$ = NAND, the input-controlling value of an in-pin of $g$ is 0, and output-controlled value of $g$ is 1. For buffers and inverters, both 0 and 1 are input-controlling values, since they uniquely determine the output. The input-controlling value of the in-pins of an exclusive-or (XOR) gate is undefined since no logic value at any single input can uniquely determine the output. In this case, the output controlled value is also undefined.

*Definition 2:* An *output noncontrolled value* of the gate $g$, denoted $\mathrm{oncv}(g)$, is the logic value which, when set at the output of $g$, uniquely implies the logic values at the inputs of $g$. An *input noncontrolling value* of an in-pin $p$ of $g$, denoted $\mathrm{incv}(p)$, is the logic value inferred when the output of $g$ is set to its output noncontrolled value.

For example, when type$(g)$ =NAND, the output noncontrolled value of $g$ is 0, since a 0 at $g$'s output uniquely implies that all inputs to $g$ have to be set to 1, the input noncontrolling value. For buffers and inverters, both 0 and 1 are output noncontrolled values since they also imply the only input's logic value. The output noncontrolled value and input noncontrolling values are undefined for exclusive-or gate.

*Logic implication* is a process of inferring consistent logic values based on known logic values. Given a logic value $v$ assigned at the out-pin of gate $g$, the direction of implication can be forward or backward, until no more logic values can be inferred. If $v = \mathrm{oncv}(g)$, all in-pins $p_i$ of $g$ can be inferred with logic value $\mathrm{incv}(p_i)$. This process is called *direct backward implication*. For example, let type$(g)$ AND and $v = 1$. All in-pins of $g$ are inferred with logic value 1. Direct backward implication stops at a gate $g_j$ when the value $v_j$ assigned at the out-pin of $g_j$ is not equal to $\mathrm{oncv}(g_j)$ and hence no logic value at the in-pins of $g_j$ can be further inferred. $imp\_value(p)$ is the value set at a pin $p$ during direct backward implication.

Let $\xi$ be an ordered list of nodes from a Boolean network. If for any $u$ preceding $v$ in $\xi$ $u$ is not in the transitive fanout cone of $v$, then $\xi$ is in a *topological order*. $\xi$ is in *reverse topological order* if for any $u$, $v$ in $\xi$, $u$ is not in the transitive fanin cone of $v$.

We will use the following two types of symmetries.

*Definition 3:* $x_i$ and $x_j$ are *nonequivalence symmetric* (NES) [8] in $f(X)$ if and only if $f_{\overline{xi}xj} = f_{xi\overline{xj}}$. That is, $f(\ldots, x_i, \ldots, x_j, \ldots) = f(\ldots, x_j, \ldots, x_i, \ldots)$.

By plugging in all four possible value combinations to $x_i$ and $x_j$ in the above equation, we can derive the following four relationships:

$$\begin{aligned} f(\ldots,0,\ldots,0,\ldots) &= f(\ldots,0,\ldots,0,\ldots)\\ f(\ldots,0,\ldots,1,\ldots) &= f(\ldots,1,\ldots,0,\ldots)\\ f(\ldots,1,\ldots,0,\ldots) &= f(\ldots,0,\ldots,1,\ldots)\\ f(\ldots,1,\ldots,1,\ldots) &= f(\ldots,1,\ldots,1,\ldots). \end{aligned}$$

The first and fourth relationships are clearly tautological. Hence, for $x_i$ and $x_j$ to be NES, the cofactors $f_{\overline{xi}xj}$ and $f_{xi\overline{xj}}$ have to be equivalent. The name "nonequivalence symmetric" was derived from the fact that in the second and third relationships, $x_i$ and $x_j$ are of opposite logic values.

*Definition 4:* $x_i$ and $x_j$ are *equivalence symmetric* (ES) [8] in $f(X)$ if and only if $f_{\overline{xi}\,\overline{xj}} = f_{xi\,xj}$. That is, $f(\ldots, x_i, \ldots, x_j, \ldots) = f(\ldots, \overline{x}_j, \ldots, \overline{x}_i, \ldots)$.

Again, we could plug in all four possible value combinations to $x_i$ and $x_j$ in the above equation and derive the following four relationships:

$$\begin{aligned} f(\ldots,0,\ldots,0,\ldots) &= f(\ldots,1,\ldots,1,\ldots)\\ f(\ldots,0,\ldots,1,\ldots) &= f(\ldots,0,\ldots,1,\ldots)\\ f(\ldots,1,\ldots,0,\ldots) &= f(\ldots,1,\ldots,0,\ldots)\\ f(\ldots,1,\ldots,1,\ldots) &= f(\ldots,0,\ldots,0,\ldots). \end{aligned}$$

This time, the second and third relationships are tautological. Hence, for $x_i$ and $x_j$ to be ES, the cofactors $f_{\overline{xi}\,\overline{xj}}$ and $f_{xixj}$ have to be equivalent. The name "equivalence symmetric" was derived from the fact that in the first and fourth relationships, $x_i$ and $x_j$ are of the same logic values.

*Definition 5:* $x_i$ and $x_j$ are *symmetric* in $f(X)$ if they are either NES or ES in $f(X)$. That is, when the distinction between NES and ES is not of importance in the context, we use the term "symmetric" to mean the relationship is either NES or ES.

Detecting symmetry has long been an active area of research in switching theory. In [16], necessary conditions for the existence of symmetry in a completely specified Boolean function are identified using the structural properties of ROBDD [1]. Chang *et al.* [4] present an extension to handle incompletely specified functions. In [19], symmetry detection is transformed into a test generation problem and solved using automatic test pattern generation (ATPG) techniques.

We briefly review some terminology used in test generation. After a chip has been designed and fabricated, it must be tested to determine whether it is working correctly. This is done by applying input vectors and then capturing and analyzing the output response. For a sequential circuit, up to $2^{n+p}$ vectors may be tested, where $n$ is the number of primary inputs and $p$ is the number of flip-flops. Applying all of the possible input vectors may take too much time. The *single stuck-at-fault model* assumes that the physical defects manifest themselves as wires, which are permanently connected to either V$dd$ or GND, and that only one such stuck-at wire exists in a given circuit.

For the single stuck-at-fault model, a wire $w$ from a node $n_x$ to a node $n_y$ could be stuck at either 1 or 0. Let $f$ be a multiple-input multiple-output Boolean function implemented by a combinational circuit C. The functions $f_{\text{faulty}}$ and $f_{\text{good}}$ are implemented by the faulty and good circuits, respectively. We use the D-notation [21] to represent the fault effect. D (1/0) means that in the good circuit, the value of a particular wire is 1 whereas in the faulty circuit the value of the same wire is 0. $\overline{\text{D}}$ denotes the opposite case. The fault on $w$ is testable if there exists a primary input vector $v$ such that $f_{\text{good}}(v) \neq f_{\text{faulty}}(v)$. That is, the difference between the good and faulty circuits can be observed at primary outputs when the primary input vector $v$ is applied. When no vector exists, which can distinguish the faulty circuit from the good one, the fault on $w$ is redundant and can be removed by assigning the constant stuck-at-value on $w$. The process of finding such a vector $v$ through algorithmic means is called automatic test pattern generation (ATPG). The processes of test generation and redundancy identification are known to be NP-hard.

In test generation, each node in the network could assume one of five different logic values which are 0, 1, $D$, $\overline{D}$, or *unassigned*. Logic operations work in a bit-wise fashion. For example, the logic-OR operation (+) between 1 and D can be determined from the individual operations in the good and faulty circuits, that is, $1 + 1$ in the good circuit, and $1 + 0$ in the faulty circuit. The result is 1 in the good circuit, and 1 in the faulty circuit. In short, $1 + \text{D} = 1$. Other operations can be similarly deduced.

We now reiterate the main results of [19] in the following lemma.

*Lemma 1:* Two inputs $x_i$ and $x_j$ are of NES if and only if no test exists that sets $x_i$ to D, sets $x_j$ to $\overline{\text{D}}$, and propagates D or $\overline{\text{D}}$ to the output of $f$. Furthermore, $x_i$ and $x_j$ are of ES if and only if no test exists that sets $x_i$ to D, sets $x_j$ to D, and propagates D or $\overline{\text{D}}$ to the output of $f$. Here we assume $f$ is represented as a mapped Boolean network.

Lemma 1 simply conveys Definitions 3 and 4 from the viewpoint of test generation. For example, when no D or $\overline{\text{D}}$ can propagate to the output of $f$ when $x_i = \text{D}$ and $x_j = \overline{\text{D}}$, it means the function $f(\ldots, x_i = 1, \ldots, x_j = 0, \ldots)$ is indistinguishable from the function $f(\ldots, x_i = 0, \ldots, x_j = 1, \ldots)$, which is exactly Definition 3. Lemma 1 establishes the link between the theory of test generation and the traditional definition of functional symmetry.

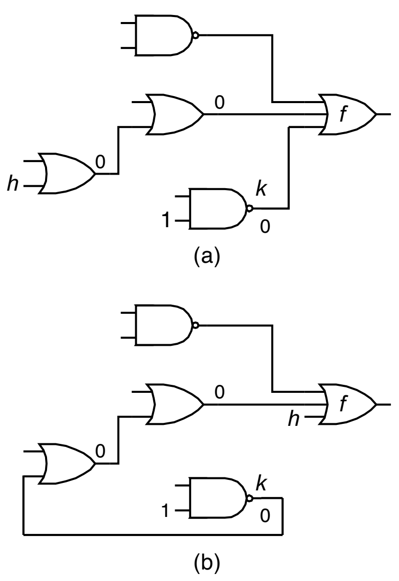


Fig. 1. $h$ and $k$ are swappable.

All previous attempts at symmetry detection have focused on finding symmetries in the primary inputs of a given function. Let $h : \{0,1\}^{\text{n}} \rightarrow \{0,1\}^{\text{m}}$ be a multiple-input, multiple-output Boolean function defined on $X = \{x_0, x_1, \ldots, x_{n-1}\}$ represented by a mapped Boolean network $N$. Also, let $N^{\text{sub}}$ be a single-output subnetwork of $N$ and let $k$ be the corresponding Boolean function defined on $Y = \{y_0, y_1, \ldots, y_r\}$, where $Y$ is a set of internal signals of $N$. Instead of finding symmetries for $x_i, x_j \in X$ with respect to $h$, we focus on the identification of symmetries for $y_i$, $y_j \in Y$ with respect to $k$. The number of detected symmetries increases dramatically since $k$ is only a subfunction of $h$. This analysis forms the basis of our rewiring technique.

Symmetries detected inside a Boolean network immediately provide ways to restructure the network for better performance. In Fig. 1(a), if we know $h$ and $k$ are symmetric, they can be swapped without changing the overall circuit functionality. Depending on the optimization goal, one of the two circuits might be better than the other. We now present our approach to detect functional symmetries inside a Boolean network.

## III. Detection of Symmetries in a Network

In this section, we use the theory of ATPG as a tool for the proofs. Our algorithm does not use ATPG.

### A. Symmetry Detection

Let $T$ be a fanout-free network rooted at $f$. Let $a$ and $b$ be two in-pins in $T$ as illustrated in Fig. 2. Since $T$ is fanout-free, there exists a unique path from $a$ to $p$, where p is an in-pin of $f$. Similarly, there exists a unique path from $b$ to $q$, where $q$ is an in-pin of $f$. We use the notation $(a \rightarrow p)$ and $(b \rightarrow q)$ for these two paths, respectively. All of the following lemmas and

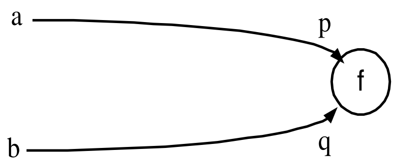


Fig. 2. Two unique paths in a fanout-free network.

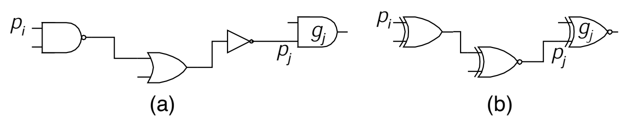


Fig. 3. (a) AND-OR reachability. (b) XOR reachability.

theorems in this section are considered under the assumption that the underlying structure is fanout-free. We also assume that $(a \rightarrow p)$ and $(b \rightarrow q)$ do not properly contain each other.

*Definition 6:* Let $p_i$ be an in-pin of a gate $g_i$ and $g_j$ be a gate in the fanout cone of $g_i$. $\mathrm{P} = (p_i, g_i, \ldots, p_j, g_j)$ is a path from $p_i$ to $g_j$. $p_i$ is AND-OR-*reachable* from $g_j$, if $p_i$ has an inferred logic value via direct backward implication when $g_j$ is set to $\mathrm{oncv}(g_j)$. $p_i$ is XOR-*reachable* from $g_j$ if all gates along P (including $g_j$) are of type either XOR, INV, or BUF, with at least one gate of type XOR.

Since an XOR gate has no controlling value, it is clear that these two definitions are mutually exclusive. That is, if $p_i$ is AND-OR-reachable from $g_j$, it cannot be XOR-reachable from $g_j$. Also, if $p_i$ is XOR-reachable from $g_{\mathrm{j}}$, it cannot be AND-OR-reachable from $g_j$.

Fig. 3 illustrates the definition of AND-OR and XOR reachability. In Fig. 3(a), when the output of $g_j$ is set to 1, it implies $p_j$ to 1, which further infers a 0 through the inverter. The backward implication ends at $p_i$ with a 1 implied. By definition, we say $p_i$ is AND-OR reachable from $g_j$. In Fig. 3(b), the path from $p_i$ to $g_j$ consists of an XOR gate followed by two XNOR gates. An XNOR gate can be viewed as an XOR gate followed by an inverter. Since all gates on the path are of either XOR or INV, $p_i$ is XOR-reachable from $g_j$ by definition.

In order to detect symmetries of two pins $(a, b)$ with respect to $f$, consider the case when $a$ is assigned D and $b$ is assigned $\overline{\mathrm{D}}$. We use $value(a)$ to denote the logic value assigned at pin $a$ and value$(a) \in \{0, 1, \mathrm{D}, \overline{\mathrm{D}}\}$.

*Lemma 2:* Let $a$ be AND-OR-reachable from the gate $f$ with input pin $p$. When value$(a)$ is set to D, it is logically inconsistent if value$(p)$ is set to $\mathrm{incv}(p)$. That is, when value$(a)$ is set to D, value$(p)$ cannot be set to $\mathrm{incv}(p)$.

*Proof:* Assume value$(p)$ is assigned $\mathrm{incv}(p)$. Since $a$ is AND-OR-reachable from $f$, assigning $\mathrm{incv}(p)$ at $p$ results in either 0 or 1 being implied at $a$, depending on the number of inversions along the path. This is in conflict with $a$ being assigned D initially. By contradiction, value$(p)$ cannot be set to $\mathrm{incv}(p)$. QED.

Take Fig. 3(a) as an example. $p_i$ is AND-OR reachable from $g_j$. Let a logic value D be set at $p_i$. At $p_j$, if we set a logic value 1, which is the input noncontrolling value of the in-pin $p_j$, it immediately implies a 1 at $p_i$. This would contradict the fact that a D has already been set at $p_i$. As a result, when we set a D at $p_i$, we cannot set an $\mathrm{incv}(p_j)$ at the input pin $p_j$.

*Lemma 3:* Let $a$ be XOR-reachable from gate $f$ and value$(a) = \mathrm{D}$, value$(p)$ can only be either D or $\overline{\mathrm{D}}$.

*Proof:* Since $x \oplus 1 = \overline{x}$ and $x \oplus 0 = x$, it is clear that the propagation of D (or $\overline{\mathrm{D}}$) from $a$ can only be inverted along the path. That is, value$(p)$ is either D or $\overline{\mathrm{D}}$, depending on the number of inversions and value assignments on the side inputs along the path; value$(p)$ cannot be assigned to 0 or 1. QED.

In Lemma 3, whether value$(p)$ is D or $\overline{\mathrm{D}}$ depends on the number of inversions and side input assignments along the path. Note that this condition differs from that of Lemma 2, where value$(p)$ is D or $\overline{\mathrm{D}}$ depending only on the number of inversions along the underlying path.

*Lemma 4:* Consider a fanout-free network $T$ rooted at $f$. Let $a$ be neither AND-OR-reachable nor XOR-reachable from gate $f$ and value$(a) = \mathrm{D}$; $\mathrm{value}(p)$ can always be assigned 0 or 1.

*Proof:* We first assume there exists no XOR gate along the path $(a \rightarrow p)$. Since $(a \rightarrow p)$ is not AND-OR-reachable, direct backward implication from $f$ stops before reaching the pin $a$. Under the assumption that the underlying structure is fanout-free, the propagation of D from $a$ can always be stopped at the gate where direct backward implication from $p$ stops. Hence $\mathrm{value}(p)$ of either 0 or 1 is always justifiable. In the case when there exists an XOR gate along the path, since $(a \rightarrow p)$ is not XOR-reachable, there exists at least one AND/OR gate $\mathrm{g_k}$ along the path. The propagation of D from $a$ can always be stopped by assigning the side input pin $p_{\mathrm{side}}$ of $\mathrm{g_k}$ with $\mathrm{icv}(p_{\mathrm{side}})$. This assignment is also justifiable because $T$ is fanout-free. Hence, $\mathrm{value}(p)$ being 0 or 1 is also justifiable. QED.

We now discuss the conditions on pin $p$ and pin $q$ such that a fault effect D or $\overline{\mathrm{D}}$ can appear at the output of the gate $f$. First, when type$(f)$ $=\{$AND, OR$\}$, we have the following two conditions.

*Condition 1:* Both value$(p)$ and value$(q)$ are D or both are $\overline{\mathrm{D}}$.

*Condition 2:* One of value$(p)$ and value$(q)$ is assigned their corresponding input noncontrolling value and the other is D or $\overline{\mathrm{D}}$.

Table I lists value assignments on pin $p$ and $q$ such that either a D or $\overline{\mathrm{D}}$ is implied at the output of $f$, when type$(f)$ $=$AND. Other value assignments, such as value$(p) = \mathrm{D}$ and value$(p) = \overline{\mathrm{D}}$, or value$(p) = 0$ and value$(q) = \mathrm{D}$, imply constant logic value 0 at the output of $f$.

Second, when type$(f)$ $=\{$XOR$\}$, the condition on pin $p$ and pin $q$ becomes the following.

TABLE I
VALUE ASSIGNMENT WHEN type($f$) =AND

| value($p$) | value($q$) | implied value($f$) | satisfies |
|---|---|---|---|
| D | D | D | Condition 1 |
| $\overline{\mathrm{D}}$ | $\overline{\mathrm{D}}$ | $\overline{\mathrm{D}}$ | Condition 1 |
| 1 | D | D | Condition 2 |
| 1 | $\overline{\mathrm{D}}$ | $\overline{\mathrm{D}}$ | Condition 2 |
| D | 1 | D | Condition 2 |
| $\overline{\mathrm{D}}$ | 1 | $\overline{\mathrm{D}}$ | Condition 2 |

*Condition 3:* Only one of value$(p)$ and value$(q)$ is D or $\overline{\mathrm{D}}$; the other can be either 0 or 1.

For example, when value$(p)$ = D and value$(q) = 1$, the output at gate $f$ is $\overline{\mathrm{D}}$. On the other hand, when Condition 3 is not satisfied, such as the case when both value$(p)$ and value$(q)$ are D, the output is a constant logic value.

If Conditions 1 and 2 fail when the type of $f$ is either AND or OR, or Condition 3 fails when $f$ is XOR type, there exists no consistent value assignment that can propagate either a D or $\overline{\mathrm{D}}$, to the output of $f$. This situation, by Lemma 1, implies $(a, b)$ are symmetric with respect to $f$.

A direct consequence of Lemma 4 is the following.

*Lemma 5:* Let $a$ be neither AND-OR-reachable nor XOR-reachable from a gate $f$; then $(a, b)$ is not symmetric with respect to $f$.

*Proof:* When value$(b)$ is set to D, it is always possible to propagate this value to $q$ such that value$(q)$ is either D or $\overline{\mathrm{D}}$, since the underlying structure T is fanout-free. Whether the value at $q$ is D or $\overline{\mathrm{D}}$ depends on the number of inversions along the path from $b$ to $q$. From Lemma 4, we know that value$(p)$ can be assigned either 0 or 1 when $a$ is neither AND-OR-reachable nor XOR-reachable from gate $f$. When type$(f)$ =AND, we can assign value$(p)$ to 1, which is the $\mathrm{NCV}$ of $f$. Similarly, we can assign value$(p)$ to 0, when type$(f)$ =OR and to either 0 or 1, when type$(f)$ =XOR. In any case, either Condition 2 or Condition 3 is satisfied and hence $(a, b)$ is not symmetric with respect to $f$. QED.

An important result is stated in the following theorem.

*Theorem 1:* $(a, b)$ are symmetric in $f$ realized as a fanout-free network if and only if $a$ and $b$ are AND-OR-reachable from the root of $f$ or $a$ and $b$ are XOR-reachable from the root of $f$.

*Proof:* The proof of the *if* part is trivial and, hence, omitted.

We prove that if $(a, b)$ are symmetric in $f$, then $a$ and $b$ are AND-OR-reachable or XOR-reachable from the root of $f$. From Lemma 5, we know that if $a$ and $b$ are not both AND-OR-reachable nor XOR-reachable, $(a, b)$ cannot be symmetric. By the law of contraposition, we conclude that $a$ and $b$ are both AND-OR-reachable or XOR-reachable if $(a, b)$ are symmetric. QED.

The importance of Theorem 1 is twofold. First, it expresses functional symmetry in terms of AND-OR, XOR reachability in a fanout-free network. Second, it provides the theoretical foundation for an efficient linear time algorithm for symmetry detection. Generally speaking, the condition of AND-OR and XOR reachability leads to the identification of NES and ES among the input pins. Knowing NES and ES directly allows us to swap pins with or without adding inverters. Details of the algorithm will be presented in the next section.

## B. Generalized Implication Supergate (GISG)

To improve the efficiency of the test generation process, *implication supergate extraction* has been proposed by Tsai *et al.* in [24]. The extraction starts by assigning output noncontrolled value to each of the primary outputs, and direct backward implications are performed as far as possible. Gates at which implications stop are called *implication supergate roots* and are assigned their corresponding output noncontrolling value to start another round of direct backward implication. The gates that are reached by the same round of direct backward implication form an *implication supergate* rooted at the output of the gate from which the process has begun. The concept of implication supergate is extended in the following definition:

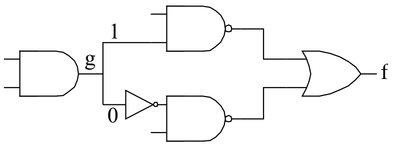


Fig. 4. Implication conflicts at the fanout stem.

*Definition 7:* Let T be a fanout-free subnetwork of N rooted at gate $f$. A GISG of $f$ is the set of gates in T that are either AND-OR-reachable or XOR-reachable from $f$. A gate $g$ is *covered* by the GISG rooted at $f$ if $\mathrm{g} \in \mathrm{GISG}(f)$. An in-pin is *covered* by a GISG if the gate to which it is attached is covered by the GISG.

The *boundary*-in-pins of a GISG are the in-pins covered by the GISG$(f)$ whose fanins are neither AND-OR-reachable nor XOR-reachable from the supergate's root $f$.

Here, the original definition of an implication supergate [24] has been extended to include XOR gates. Even though the property of AND-OR and XOR reachability can cross multiple fanout points, we restrict GISG to fanout-free regions for the purpose of keeping symmetry detection easy.

To extract the maximal GISG from a given netlist, we start from the primary outputs and process each gate in a reverse topological order. At each primary output, depending on its gate type, we attempt either direct backward implication or XOR propagation. Multiple-fanout nodes, or nodes where backward propagation stops, are treated as new GISG roots, and the propagation process continues. This procedure stops when all primary inputs are reached. After the extraction, the network is uniquely partitioned into AND, OR, and XOR supergates with inverters and buffers at their pins.

*Definition 8:* In a GISG network, each gate represents a root in the extracted network which contains all nodes that are covered by the same round of backward propagation originating from this root. A GISG is *trivial* if it covers only one gate. The *type* of a GISG is the same as the type of its root.

During GISG extraction, redundancy can often be easily found. We show two cases in the following.

- Case 1: backward implication conflicts at a fanout stem (see Fig. 4)

  In this case, we can write the following propositions:

$$f = 0 \rightarrow g = 1$$
$$f = 0 \rightarrow g = 0$$

  so, $g = 0 \rightarrow f = 1$ and $g = 1 \rightarrow f = 1$. That is, the value of f is independent of the value of $g$. This means the s-a-fault at g is untestable, and hence $g$ is redundant.
- Case 2: backward implication does not conflict at a fanout stem $f = 0 \rightarrow g = 1$. That is, $g = 0 \rightarrow f = 1$. So, one of the fanout stems of $g$ is s-a-1 untestable and hence redundant (see Fig. 5).

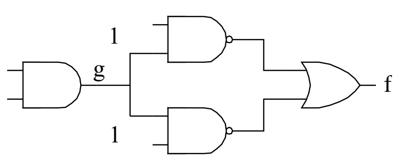


Fig. 5. Implication agrees at the fanout stem.

## IV. Swappable Pins

*Definition 9:* Let $p_i$ be an in-pin of $g_i$ and $p_j$ be an in-pin of $g_j$ in a mapped Boolean network $N$. Assume that the out-pin of $k_i$ connects to $p_i$ and the out-pin of $k_j$ connects to $p_j$. If connecting $k_i$ to $p_j$ and $k_j$ to $p_i$ does not change the functionality of $N$, then $p_i$ and $p_j$ are *noninverting swappable*. If connecting $k_i$ through an inverter to $p_j$ and $k_j$ through an inverter to $p_i$ does not change the functionality of $N$, then $p_i$ and $p_j$ are *inverting swappable*.

The notion of noninverting and inverting swappable pins corresponds to NES and ES. When there is no ambiguity, we use "swappable" to denote both noninverting swappable and inverting-swappable.

### A. Identification of Swappable Pins

The main purpose of the GISG extraction is to explore the functional symmetry inside a network. Equipped with the information of functional symmetry, we can find wires that can be exchanged without changing the functionality of the network.

*Lemma 6:* If two in-pins $p_i$ and $p_j$ are covered by the same GISG rooted at $f$ and $(p_i \rightarrow f)$ and $(p_j \rightarrow f)$ do not properly contain each other, they are swappable.

The reason for the nonproper containment constraint is as follows. Since the underlying structure is fanout-free, if one path properly contains the other, it implies that swapping of these two pins will create loops and hence cause malfunction. Take Fig. 3(a) as an example. Pins $p_i$ and $p_j$ are covered by the same GISG rooted at $g_j$. However, the path from $p_j$ to $g_j$ is properly contained by the path from $p_i$ to $g_j$. Swapping of these two pins will create a cycle in the circuit. In the following, we implicitly assume that target pins fulfill the nonproper containment constraint.

Let the in-pin $p_i$ of a gate $g_i$ is AND-OR-reachable from the gate $f$. $imp\_value(p_i)$ is the value set at $p_i$ during direct backward implication from $f$.

*Lemma 7:* Let in-pins $p_i$ and $p_j$ both be AND-OR-reachable from gate $f$. If $imp\_value(p_i) \neq imp\_value(p_j)$, then $(p_i, p_j)$ are inverting swappable. If $imp\_value(p_i) = imp\_value(p_j)$, then $(p_i, p_j)$ are noninverting swappable.

*Proof:* Assume the output noncontrolling value at $f$ is $v_o$, $\mathrm{imp\_value}(p_i) = v_i$, and $\mathrm{imp\_value}(p_j) = v_j$. By definition of direct backward implication, we have

$$(f = v_o) \Rightarrow (p_i = v_i)$$
$$(f = v_o) \Rightarrow (p_j = v_j). \tag{1}$$

By the law of contraposition, the above propositions can be rewritten as

$$(p_i = \overline{v}_i) \Rightarrow (f = \overline{v}_o)$$
$$(p_j = \overline{v}_j) \Rightarrow (f = \overline{v}_o). \tag{2}$$

Without loss of generality, assume $v_i = 0$ and $v_j = 1$ in the case when $v_i \neq v_j$. Equation (2) can be written as

$$(p_i = 1) \Rightarrow (f = \overline{v}_o)$$
$$(p_j = 0) \Rightarrow (f = \overline{v}_o). \tag{3}$$

In other words, $f|_{p_i=1} = f|_{p_j=0}$. This in turn implies $f|_{p_i=1,p_j=1} = f|_{p_i=0,p_j=0}$, which is the definition of ES. That is, when $imp\_value(p_i) \neq imp\_value(p_j)$, then $(p_i, p_j)$ are inverted swappable. Other value combinations of $p_i$ and $p_j$ can be similarly proved. QED.

*Lemma 8:* Let the in-pins $p_i$ and $p_j$ both be XOR-reachable from the gate $f$; then $p_i$ and $p_j$ are both inverting and noninverting swappable.

*Proof:* By applying $p_i = \mathrm{D}, p_j = \overline{\mathrm{D}}$, or $p_i = \mathrm{D}, p_j = \mathrm{D}$, the inputs to $f$ will always have value combinations such as D and D, $\overline{\mathrm{D}}$ and $\overline{\mathrm{D}}$, or D and $\overline{\mathrm{D}}$, because XOR gates along the path can never stop the fault effect. In either case, no fault effect can be seen at the output of the $f$. That is, inputs to XOR-reachable gates are always both ES and NES. QED.

Fig. 1(a) shows an GISG rooted at $f$. There, $imp\_value(k) = 0$ and $imp\_value(h) = 0$. By Lemma 7, we know $h$ and $k$ are noninverting swappable. That is, they can be swapped without introducing inverters. This is shown in Fig. 1(b).

### B. Cross-Supergate Swapping

Previous theorems show that pins that are covered by the same GISG are symmetric and, hence, swappable. Further analysis shows that groups of pins belonging to different implication supergates may also be swappable.

*Definition 10:* (DeMorgan transformation on an implication supergate) Let SG1 be an implication supergate and $\mathrm{type(SG1)} \in \{\text{AND, OR}\}$. We define operator DeMorgan (SG1) as the addition of inverters to all inputs and the output of SG1.

*Theorem 2:* Let SG1 and SG2 be two implication supergates. $fanin1$ and $fanin2$ are the sets of fanins to SG1 and SG2 and $|fanin1| = |fanin2|$. If the outputs of SG1 and SG2 are symmetric and type(SG1) and type(SG2) $\in \{\text{AND, OR}\}$, $fanin1$ and $fanin2$ are swappable under DeMorgan transformation of SG1 and SG2.

*Proof:* Without loss of generality, assume $\mathrm{type(SG1)} =$ AND and $\mathrm{type(SG2)} =$ OR. Since output(SG1) and output(SG2) are symmetric, these two pins are swappable. However, instead of swapping these two pins, we can connect fanins(SG1) to DeMorgan(SG2) and connect fanins(SG2) to DeMorgan(SG1). That is, we make type(DeMorgan(SG2)) equal to type(SG1) and type(DeMorgan(SG1)) equal to type(SG2). If both SG1 and SG2 are of the same type, no DeMorgan transform is necessary. This swapping and transformation procedure clearly preserves the functionality of the network. QED.

The example in Fig. 6 shows the process of cross-supergate swapping.

## V. Delay Optimization

Two types of postplacement performance optimizations are made possible by exploiting GISGs.

- Wire length reduction: Fig. 7(a) shows a set of placed gates and two signals $a$ and $b$ coming from geometrically fixed

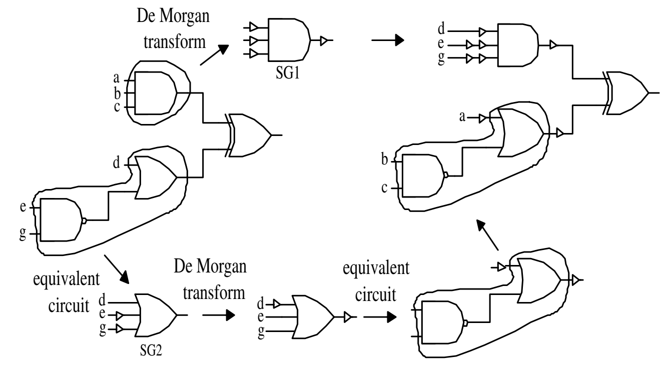


Fig. 6. Cross-supergate swapping.

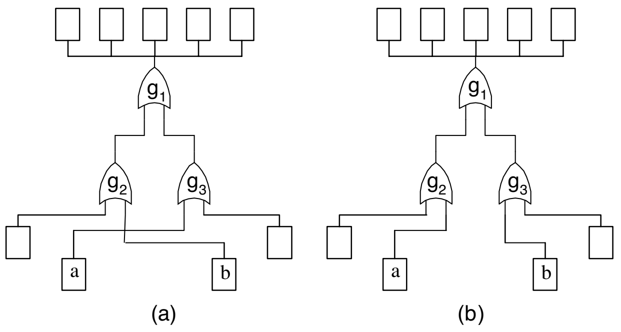


Fig. 7. Application of swapping.

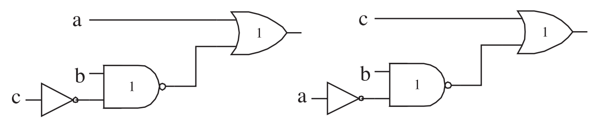


Fig. 8. Logic-level reduction.

locations. Replacement of gates $g_1$, $g_2$, and $g_3$ cannot lead to better solutions because of the connectivities from other fixed instances. However, swapping $a$ and $b$ can clearly reduce the wire length. If either net $a$ or $b$ is critical, reducing wire length directly contributes to loading reduction. Congestion may also be relieved.

- Logic level reduction: In Fig. 8, let $c$ be the late-arriving signal. Swapping $c$ with $a$ reduces the number of logic levels the late signal has to travel and, hence, reduces the overall delay.

## A. Timing Model

We assume that final routing has not been done yet, that only placement has been completed. Therefore, a net model is necessary to estimate the delay along the interconnect. We adopt the analytical model proposed in [20]. Assume all pins have known coordinates after placement. Each net is modeled as a star: the center of the star is the center of gravity of all its terminals. A net is divided into several segments: from the source to the star center and from the star center to each sink. Each segment is modeled by a lumped $RC$. We use the Elmore model [9] for delay calculation. Since the distance from the star center to each sink may vary, each sink may have a different delay from the source.

We use a load-dependent model for gate delay. The delay from an input pin $i$ to an output pin $g$ is

$$\delta(i,g) = \alpha_{i,g} + \beta_{i,g} c_g.$$

Here, $c_g$ is the load capacitance at the output of a gate $g$, and $\alpha_{i,g}$ is the intrinsic delay from in-pin $i$ to the out-pin of $g$, and $\beta_{i,g}$ is the load-dependent coefficient. Each $\alpha$ and $\beta$ has two values corresponding to the rise and fall transitions, respectively.

## B. Problem Formulation and the Algorithm

Timing optimization at postplacement stage is crucial in today's very large scale integration design flow. At this stage, all gates have already been assigned fixed locations. It is possible to use some logic restructuring techniques to further speedup the critical path. For example, a certain part of the logic close to the critical path could be collapsed and technology mapping could be redone for better performance [1]. These logic structures are first removed from the existing placement and then placed back to available slots after resynthesis. However, there could potentially be many cell overlapping in the placement that would need to be resolved with an engineering-change-order (ECO) placer. This would inevitably introduce undesirable perturbation to the existing placement because the resynthesis is based on the timing constraints from that placement.

We consider GISG-based logic restructuring to be best suited for a postplacement scenario since the restructuring involves only wire swapping and some inverter insertion. Our goal is to minimize the maximum arrival time among all primary outputs while limiting any perturbation of the existing placement solution.

We have observed that GISG-based rewiring for performance optimization is similar to the gate-sizing problem. To use gate-sizing for performance optimization, each gate in the netlist can be sized either up or down to its logically equivalent gates from the technology library. In our case, we first perform generalized supergate extraction to get a netlist of GISGs. For each GISG, a set of swappable pins is identified. Each swap can be viewed as a different library implementation of the GISG. Thus, the problem of performance-driven GISG-based rewiring is transformed into a gate sizing problem on the GISG netlist. Cross-supergate swapping is not considered in the current formulation since the occurrences of such supergates is relatively less in the test suite we used.

Our algorithm is based on the gate-sizing heuristics proposed by Coudert [6]. The idea is to maximize the minimum slack through iterative neighborhood search and relaxation. Our overall algorithm is shown in Fig. 9. The function to the left labeled $gsg\_sizing$ is called by the function to the right labeled $gsg\_opt$ as an internal routine. We first discuss the function $gsg\_sizing$. Each gate-resizing and wire-swapping choice is viewed as a possible ***move*** for the optimization and is annotated with a value called ***fitness***, which is the potential gain in terms of the cost function in the local neighborhood when the move is executed. The type of the cost function is specified by an external variable $T$, which can be either "S" or "TS." When $T =$ S, the cost of a neighborhood is defined as the minimum slack among all nodes in the neighborhood. When $T =$ TS,

```
function gsg_sizing (N: input netlist, L: technology
                     library,T: cost type)
update ← all gsg in the circuit;
moves ← ∅;
loop
  old_cost ← Cost(N, T);
  foreach gsg ∈ update
    gsg.move ← ∅;
    gsg.fit ← Fitness(N, T);
    if gsg is trivial
      foreach gates g ∈ L implementing gsg
        fit← Fitness(N[gsg ← g], T);
        if (fit > gsg.fit)
          gsg.fit ← fit;
          gsg.move ← g;
    else {//non-trivial gsg
      foreach swap s of gsg
        fit ← Fitness(N[gsg ← gsg(s), T];
        if (fit > gsg.fit)
          gsg.fit ← fit;
          gsg.move ← s;
  //end of "foreach gsg ∈ update"
  moved← BestMultipleMoves(N);
  update← GetPerturbedNodes(N, moved);
until Converge(old_cost, Cost(N, T), moved)
```

```
function gsg_opt (N: input netlist,
                  L: technology library)
best_slack ← -INFINITY;
gsg_sizing (N, L, S);
while Slack(N) > best_slack
  best_slack ← Slack(N);
  best_solution ← N;
  gsg_sizing (N, L, TS);
  gsg_sizing (N, L, S);
//end of while
return best_solution;
```

Fig. 9. Algorithm for implication supergate-based optimization.

the cost of a neighborhood is defined as the summation of slacks of all nodes in the neighborhood. While $T = \mathrm{S}$ is the direct target for delay optimization, it has been observed in [6] that $T = \mathrm{TS}$ offers a good measure for relaxation. For each GISG $gsg$ in the netlist, we find the best move based on the fitness value calculated over the local neighborhood. For $gsg$ that is trivial, we consider resizing as the set of the candidate moves. After finding best moves for each $gsg$, the algorithm sorts all of the $gsg$s into a sequence with respect to their fitness values. A series of best moves is determined by traversing the sequence of moves. We do not stop at the first maximum found when traversing the sequence. Instead, we traverse the whole sequence and determine the best sequence in order to escape from local optima. This is implemented in the BestMultipleMoves(N) function. After applying the moves, the set of gates that are in the neighborhood of the perturbed gates is put into the *update* list as candidates for the next iteration. The function stops when convergence conditions are met—either the iteration limit has been exceeded or the improvement is lower than a given threshold.

The second function $gsg\_opt$ calls $gsg\_sizing$ in an iterative fashion by switching the optimization goal between S and TS. In the $T = \mathrm{S}$ phase, we seek the best move which maximizes the minimum slack in its neighborhood. In the $T = \mathrm{TS}$, the best move is taken to maximize the summation of all slacks in its neighborhood. The goal of this phase is to speed up the network globally and escape from a local minimum. These two phases iterate until no further improvement is possible [6].

## C. Experimental Results

Our prototype tool Rewiring After Placement usIng easily Detectable Symmetries (RAPIDS) has been implemented on top of SIS 1.3 [23] and tested on both Microelectronics Center of North Carolina'91 and International Symposium on Circuits and Systems'89 benchmark suites. Sequential circuits are treated as combinational ones with all sequential elements removed. All benchmarks are optimized by SIS *script.rugged* and mapped by command "map -n 1 -AFG." We use a commercial 0.35-$\mu$m standard cell library consisting of INV, BUF, NAND, NOR, XOR, and XNOR with number of inputs ranging from 2 to 4. Each type has four different implementations. The mapped netlist is fed to a commercial timing-driven placer. We set the required time at primary outputs by taking 80% of the preplacement arrival time. This figure is used as the timing constraint to the placer. Cell locations are extracted after placement. To model interconnect, we use 2 pf/cm for unit capacitance and $2.4\ \mathrm{K}\Omega/$ cm for unit resistance. All experiments are performed on a Sun Ultra10 with 128 MB of memory. We do not perform cross-supergate swapping in our experiment. Also, we do not utilize the redundancies found during supergate extraction.

To evaluate the effect of using GISG-based rewiring for delay optimization, three algorithms have been implemented:

- $gsg$: Use only GISG-based rewiring;
- $GS$: Use only gate sizing [6];
- $gsg + GS$: For gates covered by nontrivial GISGs, use GISG-based rewiring. Otherwise, consider gate sizing for that gate. To couple these two choices tightly, our algorithm works on a netlist of extracted GISGs. Each possible swap in a GISG can be viewed as an electrically different instance of a functionally equivalent implementation from a virtual library for this GISG.

Table II shows the results of our experiments. Column 3 shows the initial critical path delay after placement. Columns 4–6 show the delay improvement by $gsg$, $GS$, and $gsg + GS$, respectively. Columns 7–9 show the runtime (in seconds) for these three algorithms. Columns 10 and 11 show the percentage of increase/decrease in the area. We consider only the area taken by gates in the netlist. Column 12 shows the percentage of gates covered by nontrivial GISGs. On average, 27.6% gates are covered. Column 13 shows the largest number of inputs among all GISGs in the netlist. In benchmark $k2$, a GISG with

TABLE II
TIMING OPTIMIZATION RESULTS

| benchmark | # of gates | initial worst slack (ns) | gsg worst slack (%) | GS worst slack (%) | gsg+ GS worst slack (%) | gsg cpu (second) | GS cpu (second) | gsg + GS cpu (second) | GS area (%) | gsg + GS area (%) | gsg coverage (%) | L | # of red. | # of symmetries |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alu2 | 516 | 7.6 | 6.9 | 2.7 | 9.7 | 3.5 | 1.6 | 6.8 | -2.7 | -2.1 | 23.4 | 9 | 7 | 760 |
| alu4 | 1004 | 10.2 | 6.8 | 8.0 | 11.1 | 14.2 | 4.5 | 22.5 | -3.1 | -3.0 | 27.5 | 12 | 14 | 1827 |
| c432 | 291 | 8.6 | 4.5 | 1.4 | 6.8 | 2.0 | 0.3 | 2.9 | -1.1 | -3.1 | 49.5 | 9 | 6 | 674 |
| c499 | 625 | 6.1 | 2.8 | 4.9 | 10.6 | 1.7 | 2.0 | 5.1 | -0.9 | +1.2 | 20.8 | 3 | 2 | 538 |
| c1355 | 625 | 6.0 | 2.3 | 7.3 | 10.3 | 1.4 | 1.8 | 6.8 | -0.3 | +0.9 | 20.8 | 3 | 2 | 538 |
| c1908 | 730 | 9.7 | 1.5 | 7.1 | 7.4 | 2.9 | 2.2 | 11.4 | -3.2 | -3.4 | 32.6 | 8 | 5 | 999 |
| c2670 | 911 | 7.0 | 2.6 | 2.8 | 8.8 | 2.6 | 1.9 | 4.5 | -4.5 | -4.5 | 21.5 | 20 | 23 | 1483 |
| c3540 | 1809 | 11.7 | 2.9 | 4.2 | 7.2 | 13.5 | 11.2 | 29.8 | -2.4 | -2.4 | 25.4 | 10 | 33 | 3064 |
| c5315 | 2379 | 9.8 | 2.8 | 5.1 | 6.5 | 5.6 | 13.5 | 16.3 | -2.6 | -3.4 | 25.7 | 9 | 103 | 2977 |
| c6288 | 5000 | 34.4 | 1.4 | 5.9 | 7.6 | 16.5 | 71.0 | 103.2 | -5.3 | -5.8 | 28.7 | 3 | 52 | 4262 |
| c7552 | 2565 | 9.3 | 1.8 | 5.1 | 7.5 | 5.5 | 8.5 | 13.9 | -2.8 | -2.7 | 18.3 | 7 | 26 | 2147 |
| i10 | 3397 | 15.3 | 0.1 | 7.4 | 11.0 | 11.3 | 17.2 | 44.4 | -0.7 | -1.3 | 24.6 | 11 | 40 | 4472 |
| x3 | 1010 | 4.8 | 5.8 | 9.5 | 14.2 | 2.4 | 3.2 | 8.6 | -2.2 | -3.4 | 27.1 | 10 | 46 | 1227 |
| i8 | 1229 | 4.8 | 3.9 | 4.5 | 8.0 | 10.2 | 5.6 | 14.6 | -2.4 | -2.8 | 30.5 | 7 | 229 | 2510 |
| k2 | 1484 | 6.7 | 8.0 | 3.0 | 10.1 | 91.2 | 3.2 | 59.9 | -0.6 | -0.7 | 43.6 | 43 | 16 | 11306 |
| s5378 | 1811 | 5.9 | 2.0 | 4.8 | 7.6 | 5.1 | 3.7 | 13.6 | -2.9 | -2.7 | 24.4 | 9 | 112 | 2194 |
| s13207 | 2900 | 9.7 | 2.3 | 6.2 | 10.2 | 35.8 | 8.0 | 76.2 | -2.1 | -1.9 | 27.7 | 24 | 90 | 8032 |
| s15850 | 4640 | 12.4 | 0.1 | 7.2 | 8.2 | 54.1 | 18.4 | 135.2 | -2.4 | -1.8 | 25.8 | 20 | 366 | 10822 |
| s38417 | 10090 | 14.7 | 0.7 | 4.8 | 7.7 | 81.6 | 35.4 | 140.6 | 0 | -0.4 | 25.8 | 21 | 1474 | 18579 |
| ave. | | | 3.1 | 5.4 | 9.0 | | | | -2.2 | -2.3 | 27.6 | | | |

43 inputs exists. Column 14 shows the number of redundancies found during GISG extraction. The last column shows the number of symmetries found in each benchmark circuit—a considerable number of them. The ability to explore such flexibility suggests a huge potential for optimization.

The results show that GISG-based rewiring and gate-sizing complement each other. Applying $gsg + GS$, the total improvement is often larger than the sum of the individual improvements. Our explanation is that $gsg$ or $GS$ may easily get stuck in local optima because critical paths often conflict with each other. On the other hand, $gsg$ and $GS$ can help extricate each from local optima by exploring a much larger solution space.

The results also show that the area is often reduced after either $GS$ or $gsg+GS$. For most benchmarks, $gsg+GS$ achieves better delay improvement than $GS$ alone while reducing the area more. This further confirms our approach of sizing only gates covered by trivial GISGs. We assume that such minimal area perturbation can be easily resolved by an ECO placer. Also, all benchmark runs finish within three minutes of CPU time.

Although in our experiment we resized only gates covered by trivial supergates, we can also resize gates covered by nontrivial supergates. This resizing can potentially enlarge the solution space for better performance, a potential that is shaping the direction of our future study.

## VI. DELAY-CONSTRAINED POWER OPTIMIZATION

In this section, we present a delay-constrained power optimization algorithm using functional symmetries. Some power estimation techniques are reviewed first.

### A. Power Model

The average power dissipation in a CMOS gate consists of three major factors

$$P_{\text{av}} = P_{\text{load}} + P_{\text{short}} + P_{\text{leak}}.$$

The first term $P_{\text{load}}$ is the power consumed when charging and discharging the output load of the gate. It depends on the output loading capacitance and the toggle rate (number of transitions per time unit). The second term $P_{\text{short}}$ indicates the short circuit current during the CMOS gate's switching. It depends on the input transition time, internal load, and the toggle rate. The last term $P_{\text{leak}}$ is the power consumed due to the device leakage current. Since $P_{\text{short}}$ and $P_{\text{leak}}$ are more device-related, we need only consider the optimization of $P_{\text{load}}$, which is the dominating factor of $P$ [17].

The toggle rate depends on the relative delays of signals propagating through the circuit. A gate can undergo a series of transitions before settling into a steady state. However, it is computationally very expensive to determine this effect, as it involves an event-driven simulator with all timing information taken into consideration [17]. In order to use the estimation as a subroutine inside our algorithm, we choose to neglect the effect of glitching and use a zero-delay model instead.

Najm has introduced the notion of equilibrium probability and transition density for power estimation [18]. The equilibrium probability of a signal $x$, denoted by $\mathrm{P}(x)$, is the fraction of time when $x$ is evaluated to logic 1. The transition density of $x$, denoted by $\mathrm{D}(x)$, is the average number of transitions per unit time. Under spatial and temporal independency assumptions, an efficient algorithm was introduced to propagate the density values from the primary inputs throughout the circuit. To see how the propagation algorithm works, recall the concept of Boolean difference: if $f$ is a Boolean function that depends on $x$, then the Boolean difference of $f$ with respect to $x$ is defined as

$$\frac{\partial f}{\partial x} = f|_{x=0} \oplus f|_{x=1} = f_{\bar{x}} \oplus f_x.$$

Here, $\oplus$ represents the Boolean exclusive-or function. The Boolean difference is the XOR of the positive and negative cofactors with respect to $x$. Essentially, $\partial f/\partial x$ is the condition that if there is a transition at $x$, there is a corresponding transition at $f$. For example, let $f$ be a two-input AND gate (i.e., $f = x_1 \cdot x_2$). The Boolean difference of $f$ with respect to $x_1$ is $(\partial f/\partial x_1) = 0 \oplus x_2 = x_2$. So, when $x_2 = 1$, any transition at $x_1$ will cause a corresponding transition at $f$.

It has been shown in [18] that under the spatial independency assumption, the transition density at the output of an $n$-input function $f$ can be calculated by the following:

$$D(f) = \sum_{i=1}^{n} P\left(\frac{\partial f}{\partial x_i}\right) D(x_i).$$

Intuitively, $D(f)$ is the summation of each of the inputs' transition densities multiplied by the probability of setting other side inputs for the propagation of the transition. The overall power consumption estimation under this measure is then

$$P_{\text{av}} = \frac{1}{2} V_{\text{dd}}^2 \sum_{i=1}^{k} C(x_i) D(x_i)$$

where $\mathrm{V_{dd}}$ is the supply voltage, $\mathrm{C}(x_i)$ is the load capacitance seen from node $x_i$, and $k$ is the total number of nodes in the circuit.

## *B. Property of Functional Symmetry*

We now analyze the effect of wire swapping on transition density.

*Theorem 3:* Let $f$ be a function defined over support set $\mathrm{X} = \{\mathrm{x_1, x_2, \ldots, x_n}\}$ and $f$ be of NES (ES) with respect to variables $\mathrm{x_i, x_j} \in \mathrm{X}$. That is, $\mathrm{x_i}$ and $\mathrm{x_j}$ can be swapped without (with) adding inverters. Let $D^{\text{new}}(f)$ be the transition density at $f$ after swapping $\mathrm{x_i}$ and $\mathrm{x_j}$. Then, the transition density after the swap will equal the transition density before the swap. That is, $D^{\text{new}}(f) = D(f)$.

*Proof:* Without loss of generality, we assume $f$ is of NES with respect to variables $\mathrm{x_i, x_j}$. That is,

$$f_{\bar{x}_i x_j} = f_{x_i \bar{x}_j}. \tag{4}$$

The case for ES can be proved similarly. The swap is illustrated in Fig. 10.

By definition, the transition density of $f$ before swap is

$$\begin{aligned} D(f) &= \sum_{k=1\ldots n} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) \\ &= \sum_{\substack{k=1\ldots n \\ k\neq i,j}} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) + P\left(\frac{\partial f}{\partial x_i}\right) D(x_i) \\ &\quad + P\left(\frac{\partial f}{\partial x_j}\right) D(x_j). \end{aligned}$$

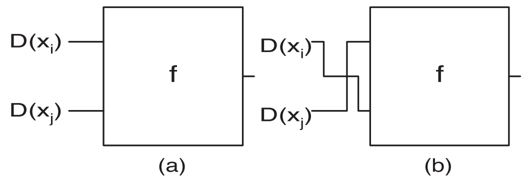


Fig. 10. Swap effect on transition density (a) before swap and (b) after swap.

For simplicity, we denote $\mathrm{D_t}(f)$ as the last two terms of $\mathrm{D}(f)$. That is $D_t(f) = P(\partial f/\partial x_i)D(x_i) + P(\partial f/\partial x_i)D(x_j)$.

The new transition density of $f$ after the swap is

$$\begin{aligned} D^{\text{new}}(f) &= \sum_{k=1\ldots n} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) \\ &= \sum_{\substack{k=1\ldots n \\ k\neq i,j}} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) + P\left(\frac{\partial f}{\partial x_i}\right) D(x_j) \\ &\quad + P\left(\frac{\partial f}{\partial x_j}\right) D(x_i). \end{aligned}$$

We denote the last two terms of $\mathrm{D}^{\text{new}}(f)$ as $D_t^{\text{new}} = P(\partial f/\partial x_i)D(x_j) + P(\partial f/\partial x_j)D(x_i)$. Observe, that $P(\partial f/\partial x_i)$ is now associated with $D(x_j)$. It is clear that $D(f) = D^{\text{new}}(f)$ if and only if $D_t(f) = D_t^{\text{new}}(f)$. Now, we proceed by expanding $\partial f/\partial x_i$ and $\partial f/\partial x_j$

$$\begin{aligned} \frac{\partial f}{\partial x_i} &= f_{x_i} \oplus f_{\bar{x}_i} \\ &= (f_{x_i} \oplus f_{\bar{x}_i})|_{x_j} + (f_{x_i} \oplus f_{\bar{x}_i})|_{\bar{x}_j} \quad \text{(Shannon Expansion)} \\ &= (f_{x_i x_j} \oplus f_{\bar{x}_i x_j}) + (f_{x_i \bar{x}_j} \oplus f_{\bar{x}_i \bar{x}_j}) \end{aligned} \tag{5}$$

$$\begin{aligned} \frac{\partial f}{\partial x_j} &= f_{x_j} \oplus f_{\bar{x}_j} \\ &= (f_{x_j} \oplus f_{\bar{x}_j})|_{x_i} + (f_{x_j} \oplus f_{\bar{x}_j})|_{\bar{x}_i} \quad \text{(Shannon Expansion)} \\ &= (f_{x_i x_j} \oplus f_{x_i \bar{x}_j}) + (f_{\bar{x}_i x_j} \oplus f_{\bar{x}_i \bar{x}_j}). \end{aligned} \tag{6}$$

By assumption, $x_i$ and $x_j$ are of NES, so (4) holds. Plugging (4) into (5) and (6), we obtain $(\partial f/\partial x_i) = (\partial f/\partial x_j)$. This result once more proves the equivalence between $D_t(f)$ and $D_t^{\text{new}}(f)$. Finally, we conclude that the new transition density after the swap is the same as the one before the swap. QED.

The importance of Theorem 3 is twofold. First, it provides the theoretical foundation for the effect of symmetric swapping on transition density. Changes in transition density are guaranteed to be bounded inside the associated implication supergate. Second, at each of the GISG roots, transition densities serve as a set of fixed points throughout the optimization. As a result, our algorithm can take a global view of the whole optimization process. The detailed algorithm will be discussed in the next section.

## *C. Algorithm*

The algorithm for delay-constrained power optimization is an extension of the algorithm developed for delay optimization. Wire swapping contributions to delay-constrained power optimization are as follows.

1) The transition density of gates covered by the same GISG can potentially be changed. Thus, it is beneficial to lower

TABLE III
EXPERIMENTAL RESULTS

| | | | | | **fixed delay constraint** | | **fixed power constraint** | | **area change** | | **CPU (second)** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **benchmark** | **initial circuit** | | | | **GS** | **gsg+ GS** | **GS** | **gsg+GS** | **GS** | **gsg+ GS** | **GS** | **gsg+ GS** |
| | **# gates** | **Delay (D)** | **Power (P)** | **Area (A)** | **%P** | **%P** | **%D** | **%D** | **%A** | **%A** | | |
| alu2 | 516 | 9.1 | 2104 | 56526 | -8.1 | -12.1 | 7.7 | -1.3 | 1.1 | 0.6 | 14 | 26 |
| alu4 | 1004 | 12.6 | 3844 | 5111253 | -10.7 | -12.5 | 2.3 | 0.2 | 0.6 | 1.1 | 27 | 55 |
| C432 | 291 | 9.8 | 1194 | 31573 | -7.5 | -9.3 | 2.9 | -0.9 | -0.7 | -0.8 | 4 | 7 |
| C499 | 625 | 5.8 | 4740 | 69992 | -10.3 | -15.6 | 3.8 | 2.4 | -3.6 | -4.0 | 12 | 28 |
| C1355 | 625 | 5.7 | 4855 | 70495 | -4.6 | -5.0 | 10.5 | 7.5 | -4.2 | -1.7 | 14 | 19 |
| C1908 | 730 | 9.4 | 3536 | 79411 | -8.2 | -11.3 | 8.1 | 5.5 | 0.5 | 1.0 | 18 | 21 |
| C2670 | 911 | 6.9 | 5257 | 101777 | -11.1 | -13.8 | 4.5 | 1.9 | -1.2 | 0.4 | 23 | 35 |
| C3540 | 1809 | 12.7 | 11559 | 204197 | -6.3 | -13.5 | 13.2 | 3.1 | 0.1 | 0.5 | 53 | 93 |
| C5315 | 2379 | 9.5 | 16158 | 267644 | -7.2 | -15.9 | 6.7 | 2.1 | -4.1 | -3.6 | 45 | 79 |
| C6288 | 5000 | 39.1 | 167898 | 584906 | -7.1 | -25.5 | 7.0 | 1.2 | -10.9 | -10.1 | 158 | 143 |
| C7552 | 2565 | 9.8 | 15577 | 286032 | -6.8 | -12.5 | 7.5 | 3.3 | -2.3 | -1.9 | 48 | 71 |
| k2 | 1484 | 6.7 | 3087 | 164329 | -6.6 | -11.9 | 12.5 | 0.3 | -0.2 | 0.5 | 30 | 74 |
| i8 | 1229 | 6.4 | 6802 | 139671 | -6.6 | -10.0 | NA† | 5.9 | 0.8 | 0.0 | 101 | 163 |
| i10 | 3397 | 17.4 | 23341 | 384827 | -13.1 | -14.5 | 0.7 | -7.9 | -2.7 | -2.6 | 701 | 125 |
| s13207 | 2900 | 10.7 | 13291 | 329798 | -11.4 | -12.7 | 2.1 | 0.1 | -4.3 | -3.7 | 44 | 106 |
| s15850 | 4640 | 12.8 | 28248 | 536498 | -10.3 | -11.9 | 1.8 | 0.9 | -4.7 | -4.4 | 124 | 288 |
| s38417 | 10090 | 15.1 | 82094 | 1139460 | -5.3 | -6.0 | 0.1 | 0.0 | -2.0 | -1.6 | 732 | 420 |
| average | | | | | -8.3 | -12.6 | 5.7 | 1.4 | -2.2 | -1.8 | | |

† gs-only is unable to reach 10% power reduction.

the transition density at gates with high loading by wire swapping.

2) Gate resizing lowers the power consumption at the risk of delay penalty. When the allowable delay penalty is reached, no further power reduction is possible.

The solution space for tradeoff can be enlarged by covering the delay losses with wire swapping, which has been shown to be good for delay optimization. For GISGs that are nontrivial, we consider each swap as a possible move. For a trivial GISG, implementations of this gate from the technology library form the set of possible moves. Hence, a move in our algorithm can be either gate resizing or wire swapping.

Now, we analyze the effect for each type of move. Basically, the resizing will affect the slack ($\Delta$S) and power ($\Delta$P) of the circuit under optimization. In [6], Coudert observed that the effect on slack tends to be confined within the local neighborhood of the move. Here, we concentrate on the effect of a swap. The change in slack can be calculated by updating the arrival/required time in the local neighborhood. The change in power consumption comes from two sources: 1) the loading capacitances of the swapped pins are changed and 2) the transition densities of the fanouts of the swapped pins are changed up to the root of the supergate. This effect can be efficiently calculated by an event-driven procedure.

We adopted an approach based on a benefit/penalty function for the delay-constrained power optimization problem by defining the fitness function of each move as follows:

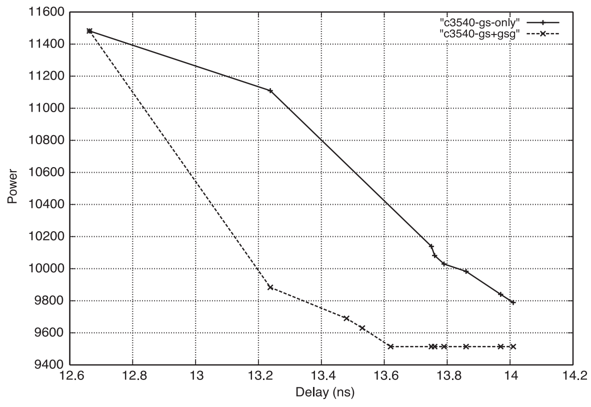
$$\text{Fitness} = \begin{cases} 0, & \Delta S < 0,\ \Delta P > 0 \\ e^{\alpha \Delta S + \beta \Delta P}, & \text{otherwise} \end{cases}$$

where $\Delta$S is the change in minimum slack in the local neighborhood, $\Delta$P is the change of power consumption of the whole circuit, and $\alpha > 0$ and $\beta < 0$ are predefined constants. A move is assigned zero fitness value (gain) if the move causes both the slack and power to become worse. Moves with zero fitness are immediately discarded. Otherwise, the gain is defined as a function depending on both $\Delta$S and $\Delta$P. In general, we want to choose a move that trades as little $\Delta$S to achieve as much $\Delta$P as possible. A move can be either a gate resizing or a wire swapping. They are distinguished only by their fitness values.

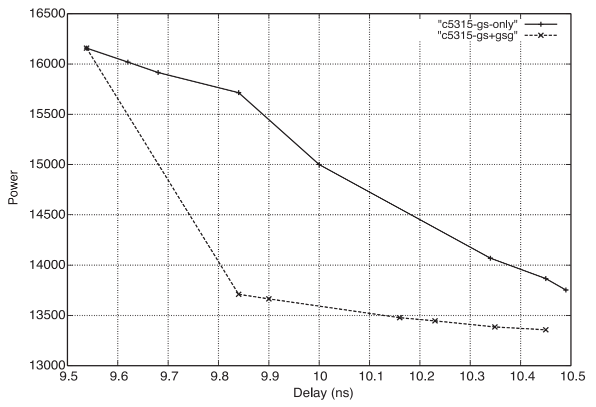
The overall algorithm is similar to the one used for delay optimization shown in Fig. 9. The cost of the network is redefined as the primary optimization goal—power consumption. The fitness function defined above is used for delay-constrained power optimization.

## D. Experimental Results

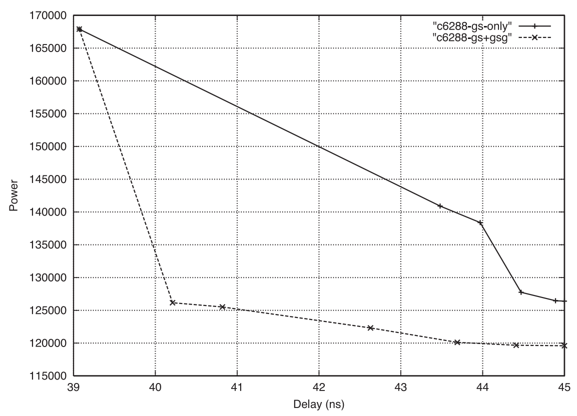
The experimental setting is the same as the one used for delay optimization. Table III shows the results. The first column lists the name of each benchmark. Column 2 shows the number of gates in the mapped netlist. To have a fair comparison with the gate-sizing-only technique, we preprocess the circuit by minimizing the critical path delay, using only gate sizing. Columns 3–5 show the corresponding delay, power, and area after timing optimization. Columns 6 and 7 show the corresponding power reduction for the gate-sizing-only approach and for our hybrid approach when the delay constraint is set at 5% worse than that of the preprocessed circuit. To demonstrate the result from another angle, we set the power constraint to be 10% less than



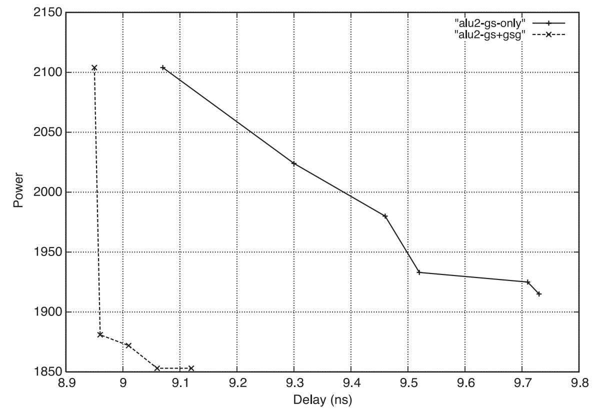
(a) C3540



(b) C5315



(c) C6288



(d) alu2

Fig. 11. Power-delay tradeoff curves (a) C3540, (b) C5315, (c) C6288, and (d) alu2.

that of the preprocessed circuit and show the delay tradeoff. The results for both approaches are shown in Columns 8 and 9, respectively. We also show the percentage of area perturbation and CPU time (in seconds) when deriving the power-delay tradeoff curve from Columns 10 to 13. That is, we show the runtime from the preprocessed circuit with the best possible slack compared with the circuit with the best possible power reduction. The area perturbation is reported for the fixed delay experiment.

The results clearly show the benefits of using functional symmetry together with gate sizing for postplacement power-delay tradeoff. In all benchmark runs, the hybrid approach has achieved better power reduction with less delay penalty. For example, in benchmark C6288, the gate-sizing-only approach reduces power by 7.1% at 5% of delay penalty. At the same delay penalty, the hybrid approach reaches as much as 25.5% reduction in power consumption. On the other hand, delay penalties reach 7.0% and 1.2%, respectively, for the gate-sizing-only and the hybrid approach when the same benchmark is reduced to 90% of its original power consumption. This shows the great potential for applying our approach to trade lower delay penalty for better power reduction. On average, at 5% delay penalty, our hybrid approach achieves 12.6% power reduction, as compared with 8.3% of the gate-sizing-only approach. At 10% power reduction, we tradeoff only 1.4% of delay, whereas using the gate-sizing-only approach we incur 5.7% delay penalty. In our experiment, we considered only trivial supergates for resizing. Further improvements are still possible by relaxing this constraint.

Our approach can potentially explore a much larger solution space than can be obtained by the gate-sizing-only approach. This can be seen in the power-delay tradeoff curves in Fig. 11. It is easily seen from the curves that our hybrid approach can quickly reach a significant power reduction while incurring only a very small delay penalty. In Fig. 11(d), the processed benchmark alu2 has power level at 2104 and delay at 9.07. Our hybrid approach immediately finds a solution with a delay of 8.95, while consuming the same amount of power. This shows that because we have a much larger solution space, we can have much more freedom to trade less delay for more power reduction.

## VII. Delay-Constrained Reliability Optimization

HCEs have been studied extensively in the past few decades [10], [13], [27]. Efficient techniques for accurate transistor-level reliability simulations have been implemented in both academic [25] and commercial tools [3]. However, transistor-level simulations of large industrial circuits are computationally too expensive to be feasible. A probabilistic approach was proposed in [14] to estimate the degradation effects on timing. Recently, a ratio-based gate-level degradation model was proposed in [28] as a higher level abstraction. Each cell from the technology library is precharacterized for its degradation behavior under various stress conditions. For an excellent review, refer to [13].

Design-for-reliability (DfR) techniques considering HCEs fall into two categories. One category includes such techniques as transistor reordering and resizing [7], technology mapping [5], and technology-independent factorization [22] to minimize the maximum hot-carrier degradation effects among all transistors in the circuit. That is, each transistor in the circuit is labeled with a relative degradation factor and the optimization goal is to minimize the maximum of these factors. This goal targets improvement of the mean-time-to-failure (MTTF) under the assumption that if any device in the circuit fails, the whole circuit fails. The other category of techniques includes the method proposed by Li *et al.*, which performs input pin reordering and gate resizing [15] to minimize the impact of performance degradation on the entire circuit. The idea is that not all devices in the circuit are of equal importance as far as overall performance is concerned. Devices not on the critical paths can potentially tolerate more degradation without affecting the overall performance.

However, all of these techniques operate at the gate/transistor level without knowledge of the physical layout information, which has a tremendous impact on device degradation. For example, input slew rate to a transistor and effective output switching are identified as the most important factors [12], [15] determining device degradation. Due to the resistive behavior of deep submicron interconnects, estimation of slew rates at the gate level is very inaccurate when the placement and routing information is unknown. Also, because of the underlying Boolean functionality, some gates experience more switching than others. Switching activity cannot be controlled for optimization purposes without changing the logic structure of the circuit.

## A. Reliability Model

In this section, we first review the ratio-based degradation model from [26] and [28].

Let $T_{\text{fresh}}$ and $T_{\text{aged}}$ be the fresh and aged pin-to-pin signal delays. $\alpha$ is the aged-to-fresh signal delay ratio which characterizes the overall pin-to-pin delay degradation of a gate due to the HCE. These variables are defined for each transition type (rise or fall) in each signal path of the logic gates. The relationship between $T_{\text{fresh}}$, $T_{\text{aged}}$, and $\alpha$ is shown in

$$T_{\text{aged}} = \alpha T_{\text{fresh}}. \tag{7}$$

$\alpha$ is a value larger than one and can be characterized by the following equation:

$$\alpha = \left(\sum_{i=1\ldots n} \alpha_i\right) - n + 1 \tag{8}$$

where $n$ is the number of transistors in series and $\alpha_i$ is the aged-to-fresh delay ratio when only pin $i$ is under stress. It is defined as follows:

$$\alpha_i = \Psi(T_{\text{slew}}, C_L, N_{\text{sw}}). \tag{9}$$

In this equation, $T_{\text{slew}}$ is the slew rate of the input pin. $C_L$ is the load capacitance of the gate output. $N_{\text{SW}}$ is the number of effective switchings of the input pin. By "effective," we mean that the input-pin switching leads to an output-pin switching. We can view $\alpha_i$ as a degradation factor of the $i$th transistor in series in the conducting path. Conceptually, slower slew rates can put transistors in undesirable bias conditions for longer periods of time, larger load capacitances can stress the transistor longer during charging and discharging, and more effective switching can stress the transistors more often. These stresses cause transistors to wear out more frequently. Function $\Psi$ is determined in the process of transistor level simulation. The results are used to build a three-dimensional table for later reference.

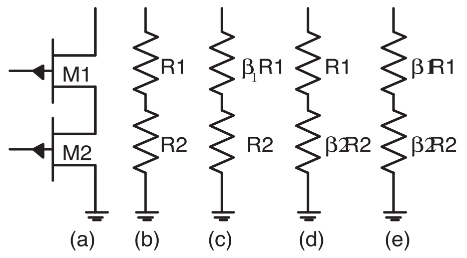


Fig. 12. Ratio derivation.

Equation (8) deserves further explanation. Essentially, it decouples/simplifies the effect of degradation on the conducting path into individual contributions from transistors along the path. For example, consider the high-to-low delay of a two-input NAND gate in Fig. 12(a). As a first-order simplification, transistors in series are regarded as resistors in series in Fig. 12(b). When only M1 switches in the whole lifetime, R1 degrades to $\beta_1$R1 with $\beta_1 > 1$ and the delay degradation ratio $\alpha_1$ can be written as follows:

$$\alpha_1 = \frac{T_{\text{aged(M1)}}}{T_{\text{fresh}}} = \frac{\beta_1 R1 + R2}{R1 + R2}.$$

Fig. 12(c) shows the effect. Similarly, if only transistor M2 switches over the whole lifetime as in Fig. 12(d), $\alpha_2$ can be written as follows:

$$\alpha_2 = \frac{T_{\text{aged(M2)}}}{T_{\text{fresh}}} = \frac{R1 + \beta_2 R2}{R1 + R2}.$$

Now, after characterizing degradation of each of the individual transistors, the effect of both degraded transistors M1 and M2 along the conducting path can be added [see Fig. 12(e)] and the resulting $\alpha$ is as follows:

$$\alpha = \frac{T_{\text{aged}}}{T_{\text{fresh}}} = \frac{\beta_1 R1 + \beta_2 R2}{R1 + R2} = \alpha_1 + \alpha_2 - 1.$$

This equation is for the case when $n$ equals two. When the number of transistors in a series is $n$, this equation can be generalized yielding (8). Based on this extended pin-to-pin delay model, full chip timing/reliability simulation is demonstrated to be 2 to 4 orders of magnitude faster [28], while accuracy is within 1% of the transistor-level counterpart.

## B. Problem Formulation

Even though large-scale, fast-yet-accurate reliability simulation is feasible, there are no systematic ways to correct the timing degradation found. In [12], design-for-reliability considerations at circuit level are given as a set of guidelines. Here, we move one step further by considering the design for reliability issues at the logic level guided by accurate layout information. Our main argument is that circuit level consideration by itself is not adequate. From (9), it is clear that degradation is affected

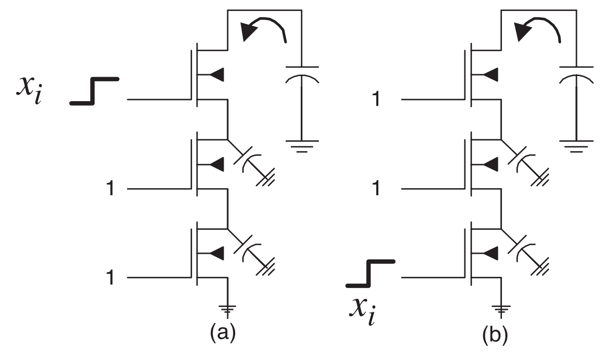


Fig. 13. Pin position affects delay and reliability.

by three variables: namely, $T_{\text{slew}}$, $C_L$, and $N_{\text{SW}}$. In a digital circuit, these three parameters could potentially vary dramatically among different gates. In other words, the stress is uneven. Some transistors wear out faster than others. For example, because of the underlying Boolean logic, some gates in the circuit switch much more frequently than others, and some gates bear larger loads than others. Circuit-level techniques simply cannot address all of these phenomena.

We now discuss the effect of gate sizing, pin reordering, and rewiring on HCE and circuit performance. To improve $T_{\text{slew}}$ of an input pin, the driver of the pin can be sized up for better driving capability to improve the slew rate. However, a larger driver causes larger loading $C_L$ on the preceding stage, which needs to be taken into account in the tradeoff. Rewiring can be used either to reduce the loading of the driver by minimizing the interconnect loading, or simply to replace the driver by another which has better driving capability or which is physically closer to the sink pin, to reduce the interconnect length. The effective switching $N_{\text{SW}}$ can be changed only by rewiring the netlist.

Another HCE effect concerning the ordering of transistors has been observed in [12] and [15]. For example, in Fig. 13(a), the top nMOS transistors that are directly connected to the output node have the potential of experiencing the most damage if they switch last. This is because the stress on nMOS transistor is directly related to $V_{\text{ds}}$ (drain-to-source voltage difference). Suppose there is an effective transition on pin $x_i$ (other inputs have already arrived). When $x_i$ connects to the output node, $V_{\text{ds}}$ is larger than $V_{\text{ds}}$ in the case when $x_i$ is closer to the ground. This effect is due to the charge redistribution on internal nodes. In Fig. 13(b), when $x_i$ is connected close to the ground and the other two transistors are conducting, the charge stored at the output is redistributed to the two internal parasitics. This effectively lowers the $V_{\text{ds}}$-induced HCE damage on $x_i$. However, conventional timing optimization techniques tend to put the last arriving signal closer to the output node to minimize the overall arrival time. This tradeoff also needs to be considered during optimization.

Let $T_{\text{fresh}}(G)$ and $T_{\text{aged}}(G)$ be the nondegraded and degraded critical path delays of a design $G$. We formulate the following problem:

*Fresh Delay Constrained Aged Delay Optimization Problem (FDCADOP):* **Instance**: We assume that we are given a placed and routed standard-cell design $G$ and a hot-carrier degradation, precharacterized, standard cell library $L$. Let $P$ be the family of the sets of pins that are identified as functionally symmetric and which can be swapped without changing the overall functionality of the design $G$.

**Configuration**: Each gate $g \in G$ can be resized by a functionally equivalent though electrically different cell from $L$. Each functionally symmetric pin pair $p \in P$ can be swapped to change the logic structure of $G$. We consider pin reordering to be a special case of functional symmetry.

**Optimization**: Let $G'$ be the new design after gate resizing and pin swapping from the original design $G$. The goal is to find a $G'$ that satisfies the following requirements:

$$\begin{aligned}&\text{minimize} \quad T_{\text{aged}}(G')\\ &\quad \text{s.t.}\\ &T_{\text{fresh}}(G') \le T_{\text{fresh}}(G).\end{aligned}$$

Essentially, we want to minimize the performance degradation of the aged design by redistributing the stress to logic elements that are not on the timing-critical path. However, we are not willing to sacrifice any performance loss in the fresh design. We observe that simply optimizing $T_{\text{fresh}}(G')$ (traditional delay optimization goal) does not necessarily lead to an optimized solution of $T_{\text{aged}}(G')$. This is because the optimized $T_{\text{fresh}}(G')$ might place to unfavorable stress conditions on the transistors along the critical path. As discussed in the previous section, various tradeoffs have to be considered simultaneously, and we need a unified algorithm that takes into account both the performance requirement and the aging effect to resolve this situation. Our algorithm will be detailed in the next section.

## *C. Algorithm*

To solve the FDCADO problem, we adopt a probabilistic approach based on [18] to estimate the effective switching $N_{\text{sw}}$ of each pin of the gate. Under the spatial and temporal independency assumptions, $N_{sw}$ for a pin $x_i$ of gate $f$ under zero-delay model can be expressed as

$$N_{\text{sw}} = D(x_i) P\left(\frac{\partial f}{\partial x_i}\right) T$$

where $D(x_i)$ is the per-unit time transitions number of the input pin $x_{\text{i}}$, $P(\partial f/\partial x_i)$ is the probability of propagating a transition from pin $x_i$ to the output of gate $g$, and $T$ is the total time period. We assume that half of the transitions $N_{sw}$ are low-to-high and half are high-to-low.

The algorithm is very similar to what we have used for delay-constrained power optimization. We change the fitness function to:

$$\text{Fitness} = \begin{cases} 0, & \Delta S_{\text{fresh}} < 0,\ \Delta S_{\text{aged}} < 0 \\ e^{\alpha \Delta S_{\text{fresh}} + \beta \Delta S_{\text{aged}}}, & \text{otherwise} \end{cases}$$

where $\Delta S_{\text{fresh}}$ and $\Delta S_{\text{aged}}$ are the changes of minimum slack caused by the move in the local neighborhood, defined as gates within a user-specified level limit from the source of the move. The upper part of the fitness function defines the situation when the move degrades both the fresh and aged circuits. This undesirable kind of move is assigned zero fitness value, which means it will never be executed. On the other hand, the exponential dependency on $\alpha \Delta S_{\text{fresh}} + \beta \Delta S_{\text{aged}}$ gives priority to

TABLE IV
EXPERIMENTAL RESULT

| | **initial** | | | **PR** | | **GS** | | **Ours** | | **CPU (second)** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Circuit** | **$T_{fresh}$ (D)** | **$T_{aged}$ (D)** | **%** | **$T_{aged}$ (Dí)** | **%** | **$T_{aged}$ (Dí)** | **%** | **$T_{aged}$ (Dí)** | **%** | **PR** | **GS** | **Ours** |
| C432 | 11.0 | 12.1 | -10.4 | 12.1 | -10.2 | 11.3 | -2.6 | 11.00 | 0.0 | 1.0 | 1.4 | 6.3 |
| C499 | 6.4 | 7.2 | -11.8 | 7.2 | -11.5 | 6.8 | -5.4 | 6.6 | -2.1 | 3.2 | 6.4 | 17.1 |
| C880 | 10.7 | 11.3 | -6.1 | 11.1 | -4.6 | 11.0 | -2.7 | 10.7 | -0.2 | 1.5 | 4.5 | 6.0 |
| C1355 | 6.3 | 7.1 | -12.0 | 7.1 | -11.4 | 6.7 | -5.5 | 6.5 | -2.4 | 3.2 | 6.8 | 16.9 |
| C1908 | 10.4 | 10.9 | -5.1 | 10.9 | -5.1 | 10.8 | -4.4 | 10.4 | 0.0 | 2.6 | 3.8 | 15.4 |
| C3540 | 14.2 | 16.0 | -12.2 | 15.8 | -11.3 | 15.0 | -5.4 | 14.5 | -2.1 | 13.2 | 19.0 | 47.5 |
| C5315 | 10.2 | 11.2 | -9.8 | 11.0 | -8.4 | 10.5 | -3.2 | 10.3 | -1.5 | 6.8 | 12.0 | 29.1 |
| C6288 | 40.4 | 44.7 | -10.5 | 44.5 | -9.9 | 44.2 | -9.3 | 40.5 | -0.2 | 18.2 | 36.4 | 104.6 |
| C7552 | 10.7 | 11.5 | -7.7 | 11.5 | -7.1 | 11.0 | -2.9 | 10.7 | -0.4 | 7.8 | 10.4 | 21.3 |
| average | | | -9.5 | | -8.8 | | -4.6 | | -1.0 | | | |

moves which accelerate both the fresh and aged circuits. Typically, $\alpha$ is chosen to be much larger than $\beta$ to penalize the situation in which the fresh delay is degraded while the aged delay is improved.

When designing our algorithm, we intentionally made no assumption about the property of the function $\Psi$ in (9). That is, no matter how $\Psi$ is characterized, either by analytical equation, empirical formula, or simply a table-look-up method, our algorithm still applies. This further demonstrates the robustness of our approach.

### *D. Experimental Results*

The experimental setting is the same as that in delay and delay-constrained power optimization. To characterize the cell library with aging information, we use the transistor level aging simulator BERT [25] together with HSPICE and verify it with analytical equations obtained from [13] for an ten-year period. We characterize the aging effect only on NMOS transistors since the degradation of PMOS transistors is relatively negligible [13] for the technology we are using.

Three algorithms have been implemented to show their relative strength in optimizing the aged circuit under fresh delay constraint: 1) pin reordering; 2) gate sizing; and 3) a hybrid approach discussed in the previous section. Experimental results are shown in Table IV. The first column lists the name of each benchmark. The second and third columns show the fresh and aged delays of the original circuit after placement. This fresh delay is used as the timing constraint for the aged circuit optimization. Column 4 is the percentage of performance degradation due to circuit aging. Columns 5 and 6 show the aged delays after pin reordering and the corresponding degradation percentages as compared with the original fresh delay. Columns 7 and 8 show the aged delays after gate sizing and the corresponding degradation percentages. Column 9 shows the aged delays after our hybrid approach, and column 10 gives the degradation percentages compared with the original fresh delay. Columns 11–13 show the CPU times in seconds for pin reordering, gate sizing, and our approach, respectively.

The results clearly show the advantage of considering logic restructuring in combination with traditional techniques. On an average, the percentage of degradation can be lowered to be within 1% of the original fresh delay using our technique, whereas pin reordering and gate sizing result in degradation of 8.8% and 4.6%, respectively. The percentage of area change caused by gate sizing is within 3% and is assumed to be amenable to corrections by an ECO placer.

## VIII. Conclusion and Future Work

Combining the theory of functional symmetry, ATPG, and supergates, we have developed a unified framework for symmetry identification in Boolean networks. Application for postplacement delay optimization has also been demonstrated. On average, the generalized gate sizing proposed here achieves 9% timing improvement at a very low computational cost and minimum perturbation of the existing placement solution.

Postplacement delay-constrained power optimization is also studied. Theoretical results on the use of functional symmetry and its effect on transition density are formally stated. With the GISG roots serving as fixed transition density points during the logic restructuring, we have developed a restructuring approach that takes a much more global view than existing greedy restructuring approaches. Our technique can be distinguished from the existing techniques in several aspects.

1) Instead of trying to globally change the transition density of the circuit, it keeps a set of fixed transition density points. This enables wire swapping to cover the delay losses when optimizing for power in a global fashion.
2) Performing optimization at postplacement stage allows us to accurately model the interconnect-induced delay and carefully trade it for power.

Even though we use transition density based on [18] as our primary means for power estimation, our approach is not limited to it. Other estimation techniques, such as simulation or symbolic techniques, can be used for better accuracy. Experimental results show that our technique achieves much better power-delay tradeoff when compared with the gate-sizing-only approach. At postlayout stage, trading as little delay penalty as possible for large power reduction is very important, as any delay penalty might lead to failure to meet the performance target.

A timing optimizer targeting directly the circuit-aging behavior is also proposed. Combining functional symmetry based on rewiring, pin reordering, and gate sizing, our approach shows much better results than the individual traditional approaches. On the average, we can minimize the impact of circuit aging to be within one percent of the original design specification.

In [11], a combined buffer insertion and redundancy-addition-and-removal (RAR) technique is proposed for post-layout performance optimization. Supergate-based rewiring, gate sizing, RAR, and buffer insertion can naturally be integrated to form a powerful back-end optimization flow with minimum perturbation on the current placement solution. As designs migrate to the deep submicron technologies, the ability to perform incremental logic restructuring after placement becomes extremely important. Our integrated technique shows great promise for solving the timing closure problem.

## REFERENCES

[1] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, pp. 264–300, Feb. 1990.
[2] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
[3] BTA Technology. [Online] Available: http://www.btat.com.
[4] C.-W. Chang and M. Marek-Sadowska, "Finding maximal symmetric groups of variables in incompletely specified Boolean functions," in *Notes Int. Workshop Logic Synthesis*, 1999.
[5] Z. Chen and I. Koren, "Technology mapping for hot-carrier reliability enhancement," *Proc. SPIE*, vol. 3216, pp. 42–50, 1997.
[6] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Trans. VLSI*, vol. 5, pp. 465–472, Dec. 1997.
[7] A. Dasgupta and R.Ramesh Karri, "Hot-carrier reliability enhancement via input reordering and transistor sizing," in *Proc. Design Automation Conf.*, 1996, pp. 819–824.
[8] C. R. Edwards and S. L. Hurst, "A digital synthesis procedure under functional symmetries and mapping methods," *IEEE Trans. Comput.*, vol. C-27, pp. 985–997, Nov. 1978.
[9] W. C. Elmore, "The transient response of damped linear network with particular regard to wideband amplifier," *J. Appl. Phys.*, vol. 19, pp. 55–63, 1948.
[10] C. Hu *et al.*, "Hot-electron-induced MOSFET degradation—model, monitor, and improvement," *IEEE. Trans. Electron Devices*, vol. ED-32, pp. 375–385, Feb. 1985.
[11] Y.-M. Jiang, A. Krstic, K.-T. Cheng, and M. Marek-Sadowska, "Post-layout logic restructuring for performance optimization," in *Proc. Design Automation Conf.*, 1997, pp. 662–665.
[12] Y. Leblebici, "Design considerations for CMOS digital circuits with improved hot-carrier reliability," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1014–1024, July 1996.
[13] Y. Leblebici and S.-M. Kang, *Hot-Carrier Reliability of MOS VLSI Circuits*. Norwell, MA: Kluwer, 1993.
[14] P.-C. Li, G. I. Stamoulis, and I. N. Hajj, "A probabilistic timing approach to hot-carrier effect estimation," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1223–1234, Oct. 1994.
[15] P.-C. Li and I. N. Hajj, "Computer-aided redesign of VLSI circuits for hot-carrier reliability," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 453–464, May 1996.
[16] D. Moller, J. Mohnke, and M. Weber, "Detection of symmetry of Boolean functions represented by ROBDDs," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 680–684.
[17] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 446–455, Dec. 1994.
[18] ——, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 310–323, Feb. 1993.
[19] I. Pomeranz and S. M. Reddy, "On determining symmetries in inputs of logic circuits," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1428–1434, Nov. 1994.
[20] B. M. Riess and G. G. Ettlt, "Speed: Fast and efficient timing driven placement," in *Proc. Int. Symp. Circuits Syst.*, 1995, pp. 377–380.
[21] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1996.
[22] K. Roy and S. Prasad, "Logic synthesis for reliability: An early start to controlling electromigration & hot-carrier effects," *IEEE Trans. Reliability*, vol. 44, pp. 251–255, June 1995.
[23] "SIS: A system for sequential circuit synthesis," Univ. California, Berkeley, Report M92/41, May 1992.
[24] K.-H. Tsai, R. Tompson, J. Rajski, and M. Marek-Sadowska, "STAR-ATPG: A high speed test pattern generator for large scan designs," in *Proc. Int. Test Conf.*, 1999, pp. 1021–1030.
[25] R. H. Tu *et al.*, "Berkeley Reliability Tools—BERT," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1524–1534, Oct. 1993.
[26] L. Wu *et al.*, "Glacier: A hot carrier gate level circuit characterization and simulation system for VLSI design," in *Proc. Int. Symp. Quality Electron. Design*, 2000, pp. 73–79.
[27] P. Yang and J.-H. Chern, "Design for reliability: The major challenge for VLSI," *Proc. IEEE*, vol. 81, pp. 730–744, May 1993.
[28] H. Yonezawa *et al.*, "Ratio based hot-carrier degradation modeling for aged timing simulation of millions of transistors digital circuits," in *Proc. Int. Electron Devices Meeting*, 1998, pp. 93–96.

**Chih-Wei (Jim) Chang** received the B.S. degree in electronics engineering from the National Chiao-Tung University, Hsinchu, Taiwan, in 1993 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California, Santa Barbara, in 1998 and 2001, respectively.

From 1993 to 1995, he was a Tactical Control Officer in the Army Missile Corporation. In 1995, he was a Lead Teaching Assistant in the Microcomputer Lab, Department of Computer and Information Science, National Chiao-Tung University, Taiwan. From 1996 to 2001, he was a Graduate Student Researcher in the VLSI Design Automation Lab, Department of Electrical and Computer Engineering, University of California, Santa Barbara. He was with Silicon Graphics Inc., Mountain View, CA, in Summer 1997, Mentor Graphics Corporation, San Jose, CA, in Summer 1999, and the Strategic Computer-Aided Design Lab, Intel Corporation, Hillsboro, OR, in Summer 2000. He joined Plato Design Systems, San Jose, CA, in 2001. He is currently with Cadence Design Systems at San Jose, CA. His research interests include logic synthesis, timing analysis, signal integrity, and physical design, with special emphasis on layout-driven logic restructuring and optimization.

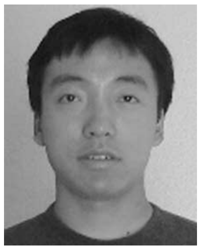
**Ming-Fu Hsiao** received the B.S. degree in electrical engineering from Chung-Yuan University, Taiwan, in 1990 and the M.S. and Ph.D. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1992, and 2003 respectively.

Since 1997, he has been with Faraday Technology Corporation, where he is currently a Senior Technical Manager. In 2001, he was a Visiting Scholar in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His current research interests include very large scale integration physical design automation, system-on-a chip design methodology, and signal integrity problem.

**Bo Hu** received the B.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 1999 and the M.S. degree in computer engineering, in 2002, from University of California, Santa Barbara, where he is currently pursuing the Ph.D. degree in computer engineering.

Since 1999, he has been a Graduate Student Researcher with the VLSI Design Automation Lab, Department of Electrical and Computer Engineering, University of California, Santa Barbara. He was with Quick Logic Corporation, Sunnyvale, CA, in 2000, with Verplex Systems Inc., Milpitas, CA, in 2001, and with Cadence Berkeley Labs, Berkeley, CA, in 2002 as a Summer Intern. His current research interests include logic optimization, technology mapping, physical design and programmable fabrics.

Mr. Hu is a Student Member of the Association for Computing Machinery.



**Kai Wang** received the B.S degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the M.S. degree in computer engineering, in 2002, from the University of California, Santa Barbara, where he is working toward the Ph.D. degree in computer engineering.

His research interests are in the area of computer-aided design of very large scale integration, with an emphasis on power-supply noise analysis and optimization, floor planning, and clock synthesis.

**Malgorzata Marek-Sadowska** (M'87–SM'95–F'97) received the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from the Technical University of Warsaw, Warsaw, Poland, in 1971 and 1976, respectively.

From 1976 to 1982, she was an Assistant Professor with the Institute of Electron Technology, Technical University of Warsaw. She became a Research Engineer in the Electronics Research Laboratory, University of California, Berkeley, in 1982 and continued there until 1990, when she became a Professor in the Department of Electrical and Computer Engineering, University of California, Santa Barbara.

Prof. Marek-Sadowska was the Editor-In-Chief of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 1993 to 1995.

**Chung-Kuan Cheng** (S'82–M'84–SM'95–F'00) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1976 and 1978 and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley, in 1984.

From 1984 to 1986, he was a Senior Computer-Aided Design Engineer at Advanced Micro Devices Inc. In 1986, he joined the University of California, San Diego (UCSD), where he is a Professor in the Computer Science and Engineering Department and an Adjunct Professor in the Electrical and Computer Engineering Department. He served as a Chief Scientist at Mentor Graphics in 1999. His research interests include network optimization and design automation of microelectronic circuits.

Prof. Cheng was an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 1994 to 2003. He is a recipient of the IEEE Transactions on Computer-Aided Design Best Paper Awards in 1997 and 2002, the NCR Excellence in Teaching Award, School of Engineering, UCSD, 1991.

**Sao-Jie Chen** received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1977 and 1982, respectively, and the Ph.D. degree in electrical engineering from the Southern Methodist University, Dallas, Texas, in 1988.

Since 1982, he has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University, where he is currently a Full Professor. During the Fall of 1999, he was a Visiting Scholar in the Department of Computer Science and Engineering, University of California, San Diego. During the Fall of 2003, he held an Academic Visitor position in the Department of System Level Design, IBM T. J. Watson Research Center, Yorktown Heights, NY. His current research interests include very large scale integration physical design automation, wireless LAN and bluetooth IC design, and system-on-a-chip hardware/software codesign.

Dr. Chen is a Member of the Chinese Institute of Engineers and the Association for Computing Machinery, and a Senior Member of the IEEE Circuits and Systems and the IEEE Computer Societies.