Timing and Area Optimization for Standard-Cell VLSI Circuit Design [1]

Weitong Chuang Sachin S. Sapatnekar Ibrahim N. Hajj

Abstract

A standard cell library typically contains several versions of any given gate type, each of which has a different gate size. We consider the problem of choosing optimal gate sizes from the library to minimize a cost function (such as total circuit area) while meeting the timing constraints imposed on the circuit.

After presenting an efficient algorithm for combinational circuits, we examine the problem of minimizing the area of a synchronous sequential circuit for a given clock period specification. This is done by appropriately selecting a size for each gate in the circuit from a standard-cell library, and by adjusting the delays between the central clock distribution node and individual flip-flops. Experimental results show that by formulating gate size selection together with the clock skew optimization as a single optimization problem, it is not only possible to reduce the optimized circuit area, but also to achieve faster clocking frequencies. Finally, we address the problem of making this work applicable to very large synchronous sequential circuits by partitioning these circuits to reduce the computational complexity.

[1] This work was supported by Joint Services Electronics Program, grant number N00014-90-J-1270.

Manuscript received ____________________

Affiliation of authors:

Weitong Chuang
AT&T Bell Laboratories
Room 3B-426B
600 Mountain Ave.
Murray Hill, NJ 07974
Tel : (908) 582-4220 E-mail : chuang@mhcnet.att.com

Sachin S. Sapatnekar
Department of Electrical Engineering and Computer Engineering
Iowa State University
Ames, IA 50011
Tel : (515) 294-1426 E-mail : sachin@iastate.edu

Ibrahim N. Hajj (corresponding author)
Coordinated Science Laboratory and
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Tel : (217) 333-3282 E-mail : hajj@uivlsi.csl.uiuc.edu

1 Introduction

1.1 Gate Sizing Problem

The delay of a MOS integrated circuit can be tuned by appropriately choosing the sizes of transistors in the circuit. While a combinational MOS circuit in which all transistors have the minimum size has the smallest possible area, its circuit delay may not be acceptable. It is often possible to reduce the delay of such a circuit, at the expense of increased area, by increasing the sizes of certain transistors in the circuit. The optimization problem that deals with this area-delay trade-off is known as the sizing problem.

The rationale for dealing with only combinational circuits in a world which is rampant with sequential circuits is as follows. A typical MOS digital integrated circuit consists of multiple stages of combinational logic blocks that lie between latches, clocked by system clock signals. Delay reduction must ensure that the worst-case delays of the combinational blocks are such that valid signals reach a latch in time for a transition in the signal clocking the latch. In other words, the worst-case delay of each combinational stage must be restricted to be below a certain specification.

For a combinational circuit, the sizing problem is formulated as

$$\begin{aligned} \textsf{minimize} \quad & Area \\ \textsf{subject to} \quad & Delay \leq T_{spec}. \end{aligned} \tag{1}$$

The problem of continuous sizing, in which transistor sizes are allowed to vary continuously between a minimum size and a maximum size, has been tackled by several researchers [1–4]. The problem is most often posed as a nonlinear optimization problem, with nonlinear programming techniques used to arrive at the solution.

A related problem that has received less attention is that of discrete or library-specific sizing. In this problem, only a limited number of size choices are available for each gate. This corresponds to the scenario where a circuit designer is permitted to choose gate configurations for each gate type from within a standard cell library. This problem is essentially a combinatorial optimization problem, and has been shown to be NP-complete [5].

Chan [5] proposed a solution to the problem that was based on a branch-and-bound strategy. The algorithm is exact for tree networks. For general networks that are not tree-structured, a backtracking-based algorithm is proposed for finding a feasible solution. For general DAGs (directed acyclic graph), a cloning procedure is used to convert the DAG into an equivalent tree, whereby a vertex of fan-out m is implicitly duplicated m times, followed by a reconciliation step in which a single size that satisfies the requirements on all of the cloned vertices is selected.

The approach of Lin et al. [6] uses a heuristic algorithm that is an adaptation of the TILOS algorithm [1] for continuous transistor sizing, with further refinements. The approach is based on a greedy algorithm that uses two measures known as sensitivity and criticality to determine which cell sizes are to be changed. The sensitivity of a cell indicates how much local delay per unit area can be decreased if we pick another template for this specific cell, while criticality tells us whether a cell has to be replaced by a larger template to fulfill the delay constraints of the circuit. A weighted sum of a cell's sensitivity and criticality is used to guide the algorithm to select a certain number of gates to be replaced with a different template.

Another algorithm proposed by Li et al. [7] is exact for series-parallel circuits. (See [7] for a formal definition of series-parallel circuits.) For a chain circuit (serial circuit), a dynamic programming technique is used to obtain optimal solution. For a simple parallel circuit, a number of transformations are repeated to obtain the optimal implementation. The optimal implementation of any series-parallel circuit is obtained by repeatedly using the chain and simple parallel circuit transformation on subcircuits of the given series-parallel circuit. This work is extended to nonseries-parallel circuits, whose structures are represented by general DAGs, and several heuristic techniques are used in conjunction with the algorithm, but no guarantees on optimality are made for such circuits.

Both of the above two approaches [6, 7] are heuristics (Chan's approach [5] is also heuristic for general circuits), and hence no concrete statements can be made on how close their solutions are to the optimal solution. Moreover, neither work shows comparisons with a technique such as simulated annealing [8] that is known to give optimal or near-optimal solutions.

In the first part of this paper, we present a new algorithm for solving the gate sizing problem for combinational circuits that takes into consideration the variations of gate output capacitance with gate resizing. In the first stage, the gate sizing problem is formulated as a linear program. The solution of this linear program provides us with a set of gate sizes that does not necessarily belong to the set of allowable sizes. Therefore, in the second phase, we move from the linear program solution to a set of allowable gate sizes, using heuristic techniques. In the third phase, we further fine-tune the solution to guarantee that the delay constraints are satisfied. Finally, to illustrate the efficacy of our algorithm, we present a comparison of the results of this technique with the solutions obtained by simulated annealing as well as by our implementation of the algorithm in [6].

1.2 Optimization for Synchronous Sequential Circuits

Optimization of synchronous sequential circuits, on the other hand, is different. An additional degree of freedom is available to the designer in that one can set the time at which clock signals arrive at various flip-flops (FFs) in the circuit by controlling interconnect delays in the clock signal distribution network. With such adjustments, it is possible to change the delay specifications for the combinational stages of a synchronous sequential circuit to allow for better sizing. However, consideration of clock skew in conjunction with sizing increases the complexity of the problem tremendously, since it is no longer possible to decouple the problem and solve it on one subcircuit at a time.

In general, given a combinational circuit segment that lies between two flip-flops i and j, if s_i

and s_j are the clock arrival times at the two flip-flops, we have the following relations:

$$s_i + Maxdelay(i,j) + T_{setup} \leq s_j + CP \quad (2)$$

$$s_i + Mindelay(i,j) \geq s_j + T_{hold} \quad (3)$$

where $Maxdelay(i,j)$ and $Mindelay(i,j)$ are, respectively, the maximum and the minimum combinational delays between the two flip-flops, and CP is the clock period. Fishburn [9] studied the clock skew problem, under the assumption that the delays of the combinational segments are constant, and formulated the problem of finding the optimal clock period and the optimal skews as a linear program. The objective was to minimize CP, with the constraints given by the inequalities in (2) and (3) above. In real design situations, however, CP is dictated by system requirements, and the real problem is to reduce the circuit area.

In the second part of the paper, we examine the following problem: Given a clock period specification, how can the area of a synchronous sequential circuit be minimized by appropriately selecting gate size for each gate in the circuit from a standard-cell library, and by adjusting the delays between the central clock and individual flip-flops? For simplicity, the analysis will use positive-edge-triggered D-flip-flops. In the following, the terminologies *flip-flop (FF)* and *latch* will be used interchangably. We assume that all primary inputs (PI) and primary-outputs (PO) are connected to FFs outside the system, and are clocked with zero (or constant) skew.

We first present an optimization algorithm for small synchronous sequential circuits. Then we consider arbitrarily large synchronous sequential circuits for which the size of the formulated problems is prohibitively large, and present a partitioning algorithm to handle such circuits. The partitioning algorithm is used to control the computational cost of the linear programs. After the partitioning procedure, we can apply the optimization algorithm to each partitioned subcircuit.

This paper is organized as follows. We describe the linear programming approach in Section 2, followed by the two post-processing phases in Sections 3 and 4. In Section 5, we formulate the synchronous sequential circuit area optimization problem and present the algorithms to tackle the problem. The partitioning algorithm that allows us to handle large circuits is presented in Section 6. Experimental results are given in Section 7. Finally, Section 8 concludes this paper.

2 Problem Formulation

2.1 Formulation of Delay Constraints

The delay of a gate in a standard-cell library can be characterized by

$$delay \;=\; R_{out} \times C_{out} + \tau = \frac{R_u}{w} \times C_{out} + \tau_1 \cdot w + \tau_2 \tag{4}$$

where R_{out} is the equivalent resistance of the gate, C_{out} is the load capacitance of the gate, τ is the intrinsic delay of the gate, R_u represents the on-resistance of a unit transistor, and w_i is called the nominal gate size of g_i. Therefore, the size of each gate can be parameterized by a number, w, referred to as the (nominal) gate size.

The output load capacitance of a gate can be calculated by summing the gate terminal capacitances of its fan-out gates and interconnect wiring capacitance, assuming that layout information is given. In general, the gate terminal capacitances of a certain transistor in different versions of a logic gate may not be exactly linearly proportional to the nominal size of that logic gate. In spite of this, we can approximate the data points by an affine function using linear least-squares approximation. In other words, the output load capacitance of logic gate i that drives logic gate j is

$$cap(i,j) = \alpha_{ij} \cdot z_j + \beta_{ij} \tag{5}$$

where z_j is the size of logic gate j.

Therefore, the output load capacitance of gate i can be found to be

$$C_{out} \;=\; cap(i,1) + cap(i,2) + \cdots + cap(i,f)$$

$$= \alpha_{i1} \cdot z_1 + \beta_{i1} + \alpha_{i2} \cdot z_2 + \beta_{i2} + \cdots + \alpha_{if} \cdot z_f + \beta_{if} \quad (6)$$

where $z_1, z_2, \ldots, z_f$ are the sizes of the cells which logic gate i fans out to.

Thus, the delay function $D(w)$ of gate i with nominal size w can be represented as

$$\begin{aligned} D(w) &= \frac{R_u}{w} \cdot C_{out} + \tau_1 \cdot w + \tau_2 \\ &= R_u \cdot \frac{\alpha_{i1} \cdot z_1 + \beta_{i1} \cdots + \alpha_{if} \cdot z_f + \beta_{if}}{w} + \tau_1 \cdot w + \tau_2 \quad (7) \end{aligned}$$

Therefore the delay of a cell is a sum of functions of $g(w, z) = z/w$ and $h(w) = 1/w$ as well as w. Since the function $g(w, z) = z/w$ is relatively smooth, it can be approximated by a convex piecewise linear function with q regions of the form

$$PWL(w,z) = \begin{cases} a_1 \cdot w + b_1 \cdot z + c_1 & (w,z) \in \text{Region } \mathbf{R}_1 \\ a_2 \cdot w + b_2 \cdot z + c_2 & (w,z) \in \text{Region } \mathbf{R}_2 \\ \vdots & \\ a_q \cdot w + b_q \cdot z + c_q & (w,z) \in \text{Region } \mathbf{R}_q \end{cases} \quad (8)$$

$$= \max_{1 \le i \le q} (a_i \cdot w + b_i \cdot z + c_i) \qquad \forall\, (x, y) \in \bigcup_{1 \le i \le q} \mathbf{R}_i \quad (9)$$

The second equality follows from the first since $PWL(w, z)$ is convex.

Similarly, we can approximate the function $h(w) = 1/w$ with a convex piecewise linear function of the form

$$pwl(w) = \begin{cases} d_1 \cdot w + e_1 & w \in \text{Region } \mathbf{R}_1 \\ d_2 \cdot w + e_2 & w \in \text{Region } \mathbf{R}_2 \\ \vdots & \\ d_q \cdot w + e_q & w \in \text{Region } \mathbf{R}_q \end{cases} \quad (10)$$

$$= \max_{1 \le j \le q} (d_j \cdot w + e_j) \qquad \forall\, w \in \bigcup_i \mathbf{R}_i \quad (11)$$

Therefore, the gate delay $D(w, z_1, \ldots, z_f)$ of a gate with size w, and fan-out gate sizes $z_1 \cdots z_f$ can be represented using a convex piecewise linear function with q regions, as follows:

$$
\begin{aligned}
& \hat{D}(w, z_1, \cdots, z_f) \\
= & \begin{cases}
\hat{a}_1 \cdot w + \hat{b}_{1,1} \cdot z_1 + \cdots + \hat{b}_{1,f} z_f + \hat{c}_1 + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_1 \\
\hat{a}_2 \cdot w + \hat{b}_{2,1} \cdot z_1 + \cdots + \hat{b}_{2,f} z_f + \hat{c}_2 + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_2 \\
\vdots & \\
\hat{a}_q \cdot w + \hat{b}_{q,1} \cdot z_1 + \cdots + \hat{b}_{q,f} z_f + \hat{c}_q + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_q
\end{cases} \\
= & \max_{1 \le i \le q} (\hat{a}_i \cdot w + \hat{b}_{i,1} \cdot z_1 + \cdots \hat{b}_{i,f} z_f + \hat{c}_i) + \tau_1 \cdot w + \tau_2 \qquad \forall\, (w, z_1 \cdots z_f) \in \bigcup_{1 \le i \le q} \mathbf{R}_i
\end{aligned}
\tag{12}
$$

2.2 Formulation of the Linear Program

The formal definition of the gate-sizing problem for a combinational circuit is as given in (1). Since the objective function, namely, the area of the circuit, is difficult to estimate, we approximate it as the sum of the gate sizes, as has been done in almost all work on sizing [1–7, 10–13].

Therefore, the cell area of a gate i can be expressed as

$$area(i) = \gamma_i \cdot w_i + \varepsilon_i \tag{13}$$

where w_i is the nominal size of gate i.

The delay specification states that all path delays must be bounded by T_{spec}. Since the number of PI-PO paths could be exponential, the set of constraining delay equations could potentially be exponential in the number of gates; unless certain additional variables, m_i, $i = 1 \cdots \mathcal{N}$ (where $\mathcal{N}$ is the number of gates), are introduced to reduce the number of constraints [11]. The *worst-case signal arrival time* m_i corresponds to the worst-case delay from the primary inputs to gate i. Using these variables, for each gate i with delay d_i, we have

$$m_i = \max\{m_j + d_i \mid \forall\, j \in Fanin(i)\} \tag{14}$$

where $Fanin(i)$ is the set of fan-in gates of gate i. Equivalently, we have

$$m_j + d_i \leq m_i, \quad \forall\, j \in Fanin(i). \tag{15}$$

This reduces the number of constraining equations to $\sum_{i=1}^{\mathcal{N}} Fanin(i)$, which, for most practical circuits, is of the order $O(\mathcal{N})$.

We now formulate the linear program as

$$\begin{array}{lllll}
\textsf{minimize} & & \displaystyle\sum_{i=1}^{\mathcal{N}} \gamma_i \cdot w_i & & \\
\textsf{subject to} & \multicolumn{4}{l}{\text{For all gates } i = 1 \cdots \mathcal{N}} \\
& m_j + d_i & \leq m_i & \forall\, j \in Fanin(i) & \\
& m_i & \leq T_{spec} & \forall\, gate\ i\ at\ POs & \\
& d_i & \geq \hat{D}(w_i, w_{i,1}, \ldots w_{i,fo(i)}) & & \\
& w_i & \geq Minsize(i) & & \\
& w_i & \leq Maxsize(i) & &
\end{array} \tag{16}$$

where $w_{i,1}, \ldots w_{i,fo(i)}$ are the sizes of the gates to which gate i fans out, and $Minsize(i)$ and $Maxsize(i)$ are the minimum and maximum sizes of gate i in the library, respectively. Notice that in the objective function, the constant term in (13) is omitted since it does not affect the result.

Eq. (16) is a linear program in the variables w_i, d_i, m_i. It is worth noting that the entries in the constraint matrix are very sparse, which makes the problem amenable to fast solution by sparse linear program approaches. Notice that the equalities of (12) are replaced here by inequalities so as to satisfy (13).

3 Phase II : The Mapping Algorithm

The set of permissible sizes for gate i is $\mathcal{S}_i = \{w_{i,1} \cdots w_{i,p_i}\}$, where p_i is the cardinality of $\mathcal{S}_i$. The solution of the linear program would, in general, provide a gate size, w_i, that does not belong to

$\mathcal{S}_i$. If so, we consider the two permissible gate sizes that are closest to w_i; we denote the nearest larger (smaller) size by w_{i+} (w_{i-}). Since it is reasonable to assume that the LP solution is close to the solution of the combinatorial problem, we formulate the following smaller problem:

$$\text{For all } i = 1 \cdots \mathcal{N}: \quad \text{Select } w_i = w_{i+} \text{ or } w_{i-}, \quad \text{such that } Delay \leq T_{spec}$$

Although the complexity has been reduced from $O(\prod_{i=1}^{\mathcal{N}} p_i)$ for the original problem to $O(2^{\mathcal{N}})$, this is still an NP-complete problem. In this section we present an implicit enumeration algorithm for mapping the gate sizes obtained using linear programming onto permissible gate sizes. The algorithm is based on a breadth-first branch-and-bound approach.

It is worth pointing out that the solution to this problem is not necessarily the optimal solution; however, it is very likely that the final objective function value at a solution arrived using good heuristics will be close to the linear program solution, and hence close to the optimal solution. This supposition is borne out by the results presented in Section 7.1.

3.1 Implicit Enumeration Approach

The rationale behind our global enumeration algorithm is based on the following observation. Given the solution of the linear programming, the majority of the gates remain at their smallest sizes. Only a small portion of the gates in the circuit are moved to a larger size because, for a typical circuit, although there may be a huge number of long paths, the number of gates on these long paths is, in general, relatively small.

Based on this observation, during the implicit enumeration procedure we may ignore those gates which are assigned to have their smallest size by the solution of the linear programming, and concentrate on those gates that have been assigned larger sizes and are probably on long paths.

Definition 1 *A critical gate is a gate whose size is larger than its smallest possible size.*

We modify the circuit topology by adding a source node so and a sink node si. A dummy edge is added from node so to each of the input nodes and from each of the output nodes to the node si. Next, for each gate i we define *max-delay-to-sink*, denoted by $mds(i)$, to be the maximum of the delays of all possible paths starting from gate i to the sink node si [14]. That is,

$$mds(i) = \max_{j \in FO(i)} \{mds(j) + d_j\} \tag{17}$$

The method for finding max-delay-to-sink is a topological sort. That is, $mds(i)$ of a gate i can be calculated only after all of the mds's of its fan-out gates have been computed. Therefore, the computation of mds's starts from sink node si and proceeds backwards until we reach the source node so.

A breadth-first search is applied to levelize the circuit from the sink node backwards.[2] The level of a gate i in this levelization is called its *backward circuit level*, $c\_level(i)$. By definition, the backward circuit level of the sink node si is 0, while the source node so has the largest backward circuit level. Starting from si, we form a state-space tree by implicitly enumerating critical gates. During the enumeration, noncritical gates remain at their minimum size and need not be enumerated. Each level in the state-space tree corresponds to a critical gate. The corresponding critical gate of level i is gate k. We also define a function $\mathcal{F}(i)$, which is used to indicate the corresponding critical gate of level i. That is, the corresponding critical gate of level i is indicated by $\mathcal{F}(i)$. Therefore, if gate k is the corresponding critical gate of level i, then $k = \mathcal{F}(i)$. Similarly, the corresponding level of a critical gate k in the state-space tree is called the gate's *tree level*, $t\_level(k)$. Therefore $t\_level(\mathcal{F}(i)) = i$. Each node at level i in the state-space tree is a *cell configuration*, which represents a possible realization of its corresponding gate. Let $C(i,j)$ denote the jth node at level i, and $anc(i,j)$ be its ancestor node.

[2]This is different from a traditional levelizing scheme which is done starting from the source node and proceeds *forwards*.

Definition 2 *A* <u>*cell configuration,*</u> *$C(i,j)$ is a triple (W_{ij}, A_{ij}, D_{ij}),*

$$
\begin{aligned}
W_{ij} = W_{C(i,j)} &\in \{w_{\mathcal{F}(i)+}, w_{\mathcal{F}(i)-}\}, \\
A_{ij} = A_{C(i,j)} &= \gamma_{\mathcal{F}(i)} \cdot W_{ij} + A_{anc(i,j)}, \\
D_{ij} = D_{C(i,j)} &= max\ \{mds(k)\}, \text{ where } k \text{ is a gate in the circuit (not necessarily a} \\
&\qquad \text{critical gate), which satisfies } c\_level(k) = c\_level(\mathcal{F}(i)) + 2.
\end{aligned}
$$

where A_{ij} is the *accumulated area* from the root to $C(i,j)$. (Notice that $\gamma_{\mathcal{F}(i)} \cdot W_{ij}$ is the cell area of gate $\mathcal{F}(i)$, given that its size is W_{ij}.)

In the state-space tree, each node has no more than two successors since there are at most two choices for the gate size. The root of the tree is, by definition, assigned a null cell configuration $(0,0,0)$. We begin with the critical gate that has the smallest backward circuit level and implicitly enumerate the two possible realizations of each gate $\mathcal{F}(i)$, $w_{\mathcal{F}(i)+}$ and $w_{\mathcal{F}(i)-}$.[3] The delay of each gate is dependent on its own size and on the size of the gates that it fans out to. Therefore, once $g_{\mathcal{F}(i)}$ has been enumerated, the delay associated with the predecessor of $g_{\mathcal{F}(i)}$ can be calculated, and the remaining critical gates can be enumerated. During the enumeration process, it is possible to eliminate several of the possibilities to prune the search space. A node $C(i,j)$ with a cell configuration (W_{ij}, A_{ij}, D_{ij}) is *bounded* if there exists a cell configuration (W_{ik}, A_{ik}, D_{ik}), at the same level of the tree such that

(1) $A_{ik} \leq A_{ij}$ and $D_{ik} < D_{ij}$, or

(2) $A_{ik} < A_{ij}$ and $D_{ik} \leq D_{ij}$.

After all of the critical gates have been implicitly enumerated, we keep calculating max-delay-to-sink for each remaining gate. However, since noncritical gates have fixed sizes, no enumeration is necessary. Rather, we simply propagate the values toward the source node. For each leaf node of the state-space tree, the max-delay-to-sink of the source node corresponding to that node is calculated

[3] If there is more than one critical gate which has the same backward circuit level, one of them is randomly chosen.

and denoted by D'_{ij}. The cell configuration which has the largest D'_{ij} and satisfies $D'_{ij} \leq T_{spec}$ is selected. By performing a trace-back from the selected leaf node to the root of the tree, the size of each critical gate is determined from the cell configurations at each traversed node.

4 Phase III : The Adjusting Algorithm

After the mapping phase, if the delay constraints cannot be satisfied, some of the gates in the circuit must be fine-tuned. For each PO which violates the timing constraints, we identify the longest path to that PO. For example, if gate p at the PO has a worst case signal arrival time $m_p > T_{spec}$, we first find the longest path, P_l, to gate p. The *path slack* of P_l is defined as

$$Pslack(P_l) = T_{spec} - m_p \tag{18}$$

For each gate along that longest path, we calculate the *local delay difference* for each of the gates along path P_l. Assume that G_{i-1}, G_i, G_{i+1} are consecutive gates, in order of precedence, on path P_l. The local delay and local delay difference associated with G_i are defined as

$$delay(G_i) = R^{i+1}_{out} \cdot C^{i+1}_{out} + R^{i}_{out} \cdot C^{i}_{out} \tag{19}$$

$$\Delta delay(G_i) = R^{i+1}_{out} \cdot \Delta C^{i+1}_{out} + \Delta R^{i}_{out} \cdot C^{i}_{out} \tag{20}$$

where R^i_{out} and C^i_{out} are, respectively, the equivalent driving resistance of gate i, and the capacitive load driven by gate i. Therefore, $\Delta delay(G_i)$ is the difference between the original local delay of G_i and the new local delay of G_i after we replace it with a different gate size that has a different value of R^i_{out} and C^{i+1}_{out}.

After calculating the local delay difference associated with each of the gates along path P_l, we select the largest one, $\Delta delay(G_n)$, which satisfies

$$\Delta delay(G_n) < Pslack(P_l) \tag{21}$$

and change the size of G_n accordingly. If none of the local delay differences satisfy (21), we select the most negative one and replace the gate with a new realization. This process continues until the delay constraints are all satisfied. Also, notice that unlike in the mapping algorithm, we do not restrict our choices to w_{i+} and w_{i-} at this phase.

5 Optimization for Sequential Circuits

The techniques described so far are valid for the sizing problem for combinational circuits. We now consider the optimization problem for synchronous sequential circuits.

5.1 Formulation of Constraints

In a synchronous sequential circuit, a data race due to clock skew can cause the system to fail [15]. Consider a synchronous sequential digital system with flip-flops (FFs). Let s_i denote the individual delay between the central clock source and flip-flop FF_i, and let CP be the clock period. Assume there is a data path, with delay d_{ij}, from the output of FF_i to the input of FF_j for a certain input combination to the system. There are two constraints on s_i, s_j and d_{ij} that must be satisfied:

Double Clocking : If $s_j > s_i + d_{ij}$, then when FF_i is clocked, the data races ahead through the path and destroys the data at the input to FF_j before the clock arrives there.

Zero Clocking : This occurs when $s_i + d_{ij} > s_j + CP$, i.e., the data reaches FF_j too late.

It is, therefore, desirable to keep the maximum (longest-path) delay small to maximize the clock speed, while keeping the minimum (shortest-path) delay large enough to avoid clock hazards.

In [9], Fishburn developed a set of inequalities which indicates whether either of the above hazards is present. In his model, each FF_i receives central clock signal delayed by s_i by the delay element imposed between it and central clock. Further, in order for a FF to operate correctly when the clock edge arrives at time t, it is assumed that the correct input data must be present and stable during the time interval $(t - T_{setup}, t + T_{hold})$, where T_{setup} and T_{hold} are the set-up time and hold time of the FF, respectively. For all of the FFs, the lower and upper bounds $MIN(i,j)$ and $MAX(i,j)$ $(1 \le i,j \le \mathcal{L}$, $\mathcal{L}$ being the total number of FFs in the circuit) are computed, which are the times required for a signal edge to propagate from FF_i to FF_j.

To avoid double-clocking between FF_i and FF_j, the data edge generated at FF_i by a clock edge may not arrive at FF_j earlier than T_{hold} after the latest arrival of the same clock edge arrives at FF_j. The clock edge arrives at FF_i at s_i, the fastest propagation from FF_i to FF_j is $MIN(i,j)$. The arrival time of the clock edge at FF_j is s_j. Thus, we have

$$s_i + MIN(i,j) \ge s_j + T_{hold}. \tag{22}$$

Similarly, to avoid zero-clocking, the data generated at FF_i by the clock edge must arrive at FF_j no later than T_{setup} amount of time before the next clock edge arrives. The slowest propagation time from FF_i to FF_j is $MAX(i,j)$. The clock period is CP, so the next clock edge arrives at FF_j at $s_j + CP$. Therefore,

$$s_i + T_{setup} + MAX(i,j) \le s_j + CP. \tag{23}$$

Inequalities (22) and (23) dictate the correct operation of a synchronous sequential system.

Our problem requires us to represent path delay constraints between *every* pair of FFs. This may be achieved by performing PERT [16] on the circuit and setting all FFs except the FF of interest (say FF_i) to $-\infty$ (∞) for the longest (shortest) delay path to from FF_i to all FFs, and the arrival time at the FF of interest is set to 0 [9]. Therefore in addition to longest-path delay variable, m_k, for the shortest-path delay, we introduce new variables, p_k, $k = 1 \cdots \mathcal{N}$, correspond to the shortest delay from PIs (the outputs of FFs are considered as pseudo PIs) up to the output of G_k.

$$p_j + d_k \ge p_k, \quad \forall\, j \in Fanin(k). \tag{24}$$

To represent path delays between every pair of FFs, we need intermediate variables m_k^i (p_k^i) to represent the longest (shortest) delay from FF_i to the k^{th} gate. The number of constraints so introduced may be prohibitively large. An efficient procedure for intelligent selection of intermediate m_k^i and p_k^i variables to control the number of additional variables and constraints *without* making approximations has been developed. Deferring a discussion on these procedures to Section 5.2, we now formulate the linear program for a general synchronous sequential circuit as

$$
\begin{array}{lll}
\textsf{minimize} & \displaystyle\sum_{k=1}^{\mathcal{N}} \gamma_k \cdot w_k & \\
\textsf{subject to} & d_k \geq D(w_k, w_{k,1}, \ldots w_{k,fo(k)}), & 1 \leq k \leq \mathcal{N} \\
& w_k \geq Minsize(k), & 1 \leq k \leq \mathcal{N} \\
& w_k \leq Maxsize(k), & 1 \leq k \leq \mathcal{N} \\
& \text{For all FF } i, & 1 \leq i \leq \mathcal{L} \\
& \quad s_i + p_k^i \geq s_j + T_{hold} & 1 \leq j \leq \mathcal{L}, \quad k = Fanin(FFj) \\
& \quad s_i + T_{setup} + m_k^i \leq s_j + CP & 1 \leq j \leq \mathcal{L}, \quad k = Fanin(FFj) \\
& \quad \text{For all gates } \; k = 1, \cdots, \mathcal{N} & \\
& \qquad m_l^i + d_k \leq m_k^i, & \forall\, l \in Fanin(k) \\
& \qquad p_l^i + d_k \geq p_k^i, & \forall\, l \in Fanin(k)
\end{array}
\tag{25}
$$

The above is a linear program in the variables w_i, d_i, m_i, p_i and s_i. Again, the entries in the constraint matrix are very sparse, which makes the problem amenable to fast solution by sparse linear program approaches.

5.2 Symbolic Propagation of Constraints

We begin by counting the number of LP constraints in (25). We ignore the constraints on the maximum and minimum sizes of each gate since these are handled separately by the simplex method. The d_k inequalities impose q constraints for each of the gates in the circuit to the LP formulation (see Eq.(10)). Let $\mathcal{F} = \sum_{i=1}^{\mathcal{N}} Fanin(i)$, where $\mathcal{N}$ is the total number of gates in the circuit. Then for each FF i, there are $O(\mathcal{F} + \mathcal{L})$ constraints, where $\mathcal{L}$ is the total number of FFs in the circuit. Therefore the total number of constraints could be as large as $O(\mathcal{N} \cdot q + \mathcal{L} \cdot (\mathcal{F} + \mathcal{L}))$. Assume that the average number of fanins to a gate is 2.5 and $q = 4$. Then $\mathcal{F} = 2.5\mathcal{N}$, and $\mathcal{L} \cdot \mathcal{F}$ is the dominant term in the expression above. For real circuits, $\mathcal{L}$ is large, and hence the number of constraints could be tremendous. In this section, we propose a symbolic propagation method to prune the number of constraints by a judicious choice of the intermediate variables m and p, without sacrificing accuracy. Basically, for any PI, we introduce m and p variables for those gates that are in that PIs fanout cone. Also, we collapse constraints on chains of gates wherever possible (line 6 in Figure 1).

The synchronous sequential circuit is first levelized. For this purpose, the inputs of FFs are considered as pseudo POs the outputs of FFs are considered as pseudo PIs. Two string variables, $mstring(i)$ and $pstring(i)$, are used to store the long-path delay and short-path delay constraints associated with gate i, respectively. For each gate and each FF, an integer variable $u_i \in \{0, 1\}$ is introduced to indicate its status. u_i has the value 1 whenever $mstring(i)$ and $pstring(i)$ are non-empty, i.e., when the constraints stored in $mstring(i)$ and $pstring(i)$ must be propagated; otherwise, $u_i = 0$.

The algorithm for propagating delay constraints symbolically is given in Figure 1. In the following discussion of the algorithm, we elaborate on the formation of $mstring$; the formation of $pstring$ proceeds analogously. At line 2, for each gate j, u_j and $mstring(j)$ are initialized by setting $u_j = 0$, and $mstring(j)$ to the null string. The status variable of primary input i, u_i, is set to 1, however, since we are to process delay constraints with respect to that particular PI (line 3). At line 6, we check if $u_l = 0$ for all $l \in fanin(k)$, i.e., if all of gate k's input gates have a null $mstring$. If so, no constraints need to be propagated, and no operations are needed. Hence we continue to process the next gate. Next, at line 7, we check whether exactly one of all of gate

ALGORITHM Symbolic_propagation()

1. for i = 1 to $\mathcal{L}$
2. $u_j \leftarrow 0$, $mstring(j) \leftarrow$ "", $ptring(j) \leftarrow$ "" for all gates and PIs;
3. $u_i \leftarrow 1$;
4. for j = 1 to max_level
5. for each gate k at level j
6. if ($u_l = 0$ for all $l \in fanin(k)$) continue; /* do nothing */
7. if (among all $l \in fanin(k)$, exactly one $u_l = 1$, others equal 0)
8. $mstring(k) \leftarrow mstring(l') +$ "d_k", $pstring(k) \leftarrow pstring(l') +$ "d_k", $u_k \leftarrow 1$;
   /* $u_{l'} = 1$, $l' \in fanin(k)$ */
9. else
10. $u_k \leftarrow 1$, $mstring(k) \leftarrow$ "m_k^i", $pstring(k) \leftarrow$ "p_k^i";
11. for all $u_l = 1$, $l \in fanin(k)$
12. write down the two constraints,
13. $mstring(l) + d_k \leq m_k^i$, $pstring(l) + d_k \geq p_k^i$,

Figure 1: The symbolic constraints propagation algorithm.

k's input gates, say gate l', has a non-empty *mstring*, others all have null *mstring*'s. If so, we simply continue to propagate the constraint. This is implemented by concatenating $mstring(l')$ and "d_k", and storing the resulting string in $mstring(k)$. Also u_k is set to 1 to indicate that further propagation is required at this gate (line 8). Finally, if more than one of gate k's input gates have non-empty *mstring*, we add a new intermediate variable, m_k^i, and the string "m_k^i" is stored at $mstring(k)$ (line 10). For each input gate whose *mstring* is non-empty ($u_l = 1$), we need a delay constraint (line 13).

Example 1 Figure 2 gives an example that illustrates the symbolic delay constraints propagation algorithm. Assume that $mstring(11)$ = "m_{11}^1", $mstring(12) = mstring(13)$ = "" (null string). Therefore, from lines 6 and 7 of the pseudo-code, $mstring(14)$ = "$m_{11}^1 + d_{14}$" and $u_{14} = 1$. Propagating this further, we find that similarly, $mstring(15)$ = "$m_{11}^1 + d_{14} + d_{15}$", and $u_{15} = 1$. Finally, for gate 16, we apply lines 9 through 12, and find that we must introduce a variable m_{16}^1,

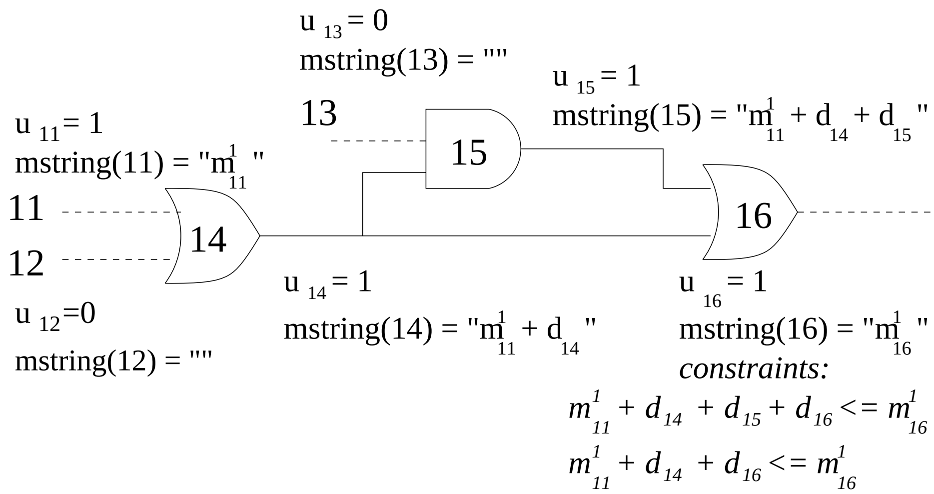

Figure 2: An example illustrating symbolic delay propagation algorithm.

and set $u_{16} = 1$. We also write down the two constraints shown in the figure and add these to the set of LP constraints. $\square$

Using the symbolic constraints propagation algorithm, although the actual reduction is dependent on the structure of the circuit, experimental results show that this algorithm can reduce the number of constraints to less than 7% of the original number on the average for the tested circuits.

5.3 Inserting Delay Buffers to Satisfy Short Path Constraints

The solution of the LP would, in general, provide a gate size, w_k that does not belong to the permissible set, $\mathcal{S}_k = \{w_{k,1} \cdots w_{k,q_k}\}$. If so, we consider the two permissible gate sizes that are closest to w_k; we denote the nearest larger (smaller) size by w_{k+} (w_{k-}). As in Section 3, we formulate the following smaller problem:

$$
\begin{aligned}
\text{For all } k = 1 \cdots \mathcal{N}: \quad & \text{Select } w_k = w_{k+} \text{ or } w_{k-}, \text{ such th at} \\
& \text{for all FFs } \; 1 \leq i\,, j \leq \mathcal{L} \\
& \quad s_i + Maxdelay(i,j) + T_{setup} \leq s_j + CP \\
& \quad s_i + Mindelay(i,j) \geq s_j + T_{hold}
\end{aligned}
$$

The mapping algorithm described in Section 3 can be used to obtain a solution for this problem.

After the mapping phase, if some of the delay constraints cannot be satisfied, we have to fine-tune some gate sizes in the circuit. In Section 4, we have discussed the approach to resolve violation of long path delay constraints. The same strategy can be applied for synchronous sequential circuit optimization, except the definition of path slack must be modified.

For each PO j (including pseudo POs at the inputs of FFs), the required maximum (minimum) signal arrival times, $req_l(j)$ ($req_s(j)$), can be expressed as

$$\begin{aligned} req_l(j) &= s_j + CP - T_{setup} \\ req_s(j) &= s_j + T_{hold} \end{aligned} \tag{26}$$

The path slack then can be defined as

$$Pslack(P_l(n)) = req_l(n) - m_n \tag{27}$$

Violations of short path delay constraints, on the other hand, can be resolved by inserting delay buffers. However, buffer insertion cannot be carried out arbitrarily, since one must simultaneously ensure that the changes in the circuit do not violate any long path constraints.

For every gate i in the circuit, we define the *gate slack*, $Gslack(i)$, as

$$Gslack(i) = \begin{cases} \min\limits_{j \in FO(i)} \{m_j + Gslack(j) - (d_j + m_i)\}, & \text{if gate } i \text{ is not at a PO.} \\ \\ \min\{\min\limits_{j \in FO(i)} [m_j + Gslack(j) - (d_j + m_i)],\ \ (req_l(i) - m_i)\}, & \text{if gate } i \text{ is at a PO.} \end{cases} \tag{28}$$

Note that if gate i is at a PO, it could still fan out to other gates in the circuit; this is reflected in the definition of the gate slack. Physically, a gate slack corresponds to the amount by which the delay of gate i can be increased before its effect will be propagated to any POs or FFs, in terms of long path delay. Therefore, it also tells us the maximum delay that a delay buffer can have if we are to insert a delay buffer at the output of gate i.

ALGORITHM Insert_buffer($n1$)

1. Let $P_s(n1)$ be the shortest path to gate $n1$, and $G_{n1}, G_{n2}, \cdots, G_{nk}$ be on path $P_s(n1)$ (G_{ni} fans out to $G_{n(i-1)}$, $2 \le i \le k$, $k =$ # of gates along $P_s(n1)$.);
2. $i \leftarrow 1$;
3. while ($p_{n1} < req_s(n1)$)
4.   if ($\exists$ a (smallest) buffer, *bf*, in the library such that: $delay(G_{ni}) < delay'(G_{ni}) + delay(bf) \le delay(G_{ni}) + slack(G_{ni})$)
5.   insert *bf* at the output of G_{ni};
6.   incrementally update $slack(j)$, m_j, p_j for each gate j in the circuit;
7.   if ($p_{n1} \ge req_s(n1)$) stop;
8.   else goto 1.
9. $i \leftarrow i + 1$;

Figure 3: The buffer insertion algorithm.

If output gate G_{n1} violates the hold time constraint, its shortest path, $P_s(n1)$, to some PI is first identified. If p_{n1} is the worst-case shortest path signal arrival time of gate $n1$, and $req_s(n1)$ is the required shortest path delay, then the delay of $P_s(n1)$ must be increased by at least $req_s(n1) - p_{n1}$.

At the beginning of this phase, we first back-propagate gate slacks from POs and all FFs. The gate slack of each gate is determined recursively using (28).

The algorithm for inserting buffers is shown in Figure 3. In line (4) of the algorithm, beginning from the smallest buffer in the library, we try to insert a buffer at the output of gate G_{ni}. The delay of the buffer is denoted by $delay(bf)$. Since the output capacitance of G_{ni} is changed during this process, we have to recalculate its delay, which is denoted by $delay'(G_{ni})$.

6 Partitioning Large Synchronous Circuits

As indicated above, the number of constraints in our formulation of the LP is in the worst proportional to the product of the number of gates and the number of FFs in the circuit. Ideally for a given synchronous sequential circuit, all variables and constraints should be considered together to obtain an optimal solution. However, for large synchronous sequential circuits, the size of the LP could be prohibitively large even with our symbolic constraint propagation algorithm. Therefore, it is desirable to partition large synchronous sequential circuits into smaller, more tractable subcircuits, so that we can apply the algorithm described in Section 5 to each subcircuit. While this would entail some loss of optimality, an efficient partitioning scheme would minimize that loss; moreover the reduction of execution time would be very rewarding.

It is well-known that multiple-way network partitioning problems are NP-hard. Therefore, typical approaches to solving such problems find heuristics that will yield approximate solutions in polynomial time [17–24]. Traditional partitioning problems usually have explicit objective functions; for example, in physical layout it is desirable to have minimal interface signals resulting from partitioning the circuit, and hence the objective function to be minimized there is the number of nets connecting more than two blocks. Our synchronous sequential circuit partitioning problem, however, is made harder by the absence of a well-defined objective function; since our ultimate goal is to minimize the total area of the circuit, there is no direct physical measure that could serve as an objective function for partitioning. In the remainder of this section, we develop a heuristic measure that will be shown to be an effective objective function for our partitioning problem.

To help us describe our partitioning algorithm, we introduce the following terminology. For a synchronous sequential circuit, such as the one shown in Figure 4:

An **internal latch** is a latch whose fanin and fanout gates belong to the same combinational block.

A **sequential block** consists of a combinational subcircuit and its associated internal latches.

Boundary latches are latches that act as either a pseudo PI or a pseudo PO (but not both) to a combinational block, i.e. latches whose fanin and fanout gates belong to different combinational blocks.

A partition of a synchronous sequential circuit **N** is a partition of the sequential blocks of **N**

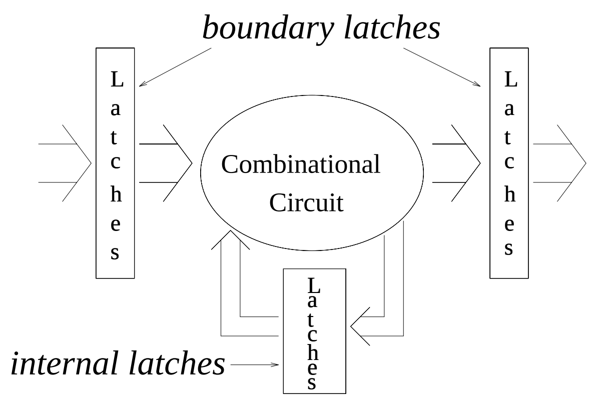

Figure 4: An example illustrating the definition of a synchronous block.

into disjoint groups. A b-way partitioning of the network is described by the b-tuple $(\mathbf{G}_1, \mathbf{G}_2, \ldots \mathbf{G}_b)$ where the $\mathbf{G}_i's$ are disjoint sets of sequential blocks whose union is the entire set of blocks in the network. Each $\mathbf{G}_i$ is said to be a *group* of the partition.

For a given sequential block $\mathbf{B}$, let $L_{\mathbf{B}}$ denote the set of boundary latches incident on $\mathbf{B}$, and for a given boundary latch L, $\mathbf{B}_L$ denotes the set of sequential blocks that L is connected to. For each boundary latch L, we define *input tightness* τ_{in}, *output tightness* τ_{out}, and the *tightness ratio* τ as

$$\begin{aligned} \tau_{in}(L) &= \text{maximum combinational delay from any boundary latch to } L \text{ in the unsized circuit,} \\ \tau_{out}(L) &= \text{maximum combinational delay from } L \text{ to any boundary latch in the unsized circuit,} \\ \tau(L) &= \begin{cases} \tau_{in}/\tau_{out} & \text{if } \tau_{in} \geq \tau_{out} \\ \tau_{out}/\tau_{in} & \text{if } \tau_{in} < \tau_{out} \end{cases} \end{aligned} \tag{29}$$

where the adjective “unsized” implies that all gates in the subcircuit are at the minimum size. The tightness ratio $\tau(L)$ provides a measure of how advantageous it would be to provide a skew at L.

For each pair of blocks $(\mathbf{B}_i, \mathbf{B}_j)$, define merit μ_{ij} as

$$\mu_{ij} = \sum_{\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j} \tau(L_k) \tag{30}$$

where $\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j$ means latch L_k lies between $\mathbf{B}_i$ and $\mathbf{B}_j$. μ_{ij} is defined to be 0 if $\mathbf{B}_i$ and $\mathbf{B}_j$ are disjoint. Physically, μ_{ij} is used to measure the figure of merit if $\mathbf{B}_i$ and $\mathbf{B}_j$ are in the same group. A high μ_{ij} means that the tightness ratio is high and hence $\mathbf{B}_i$ and $\mathbf{B}_j$ should be in the same group.

The cost associated with each block, $\mathbf{B}_i$, is c_i, the number of linear programming constraints required for solving $\mathbf{B}_i$. This number can be calculated very efficiently. Assume that group $\mathbf{G}_k$ consists of blocks $\mathbf{B}_{ki}, i = 1, \ldots |\mathbf{G}_k|$. Then we define the cost of $\mathbf{G}_k$, $C(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} c_{ki}$, and the merit of $\mathbf{G}_k$, $M(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} \sum_{j=i+1}^{|\mathbf{G}_k|} \mu_{ij}$. We now formulate the following optimization problem:

$$\begin{aligned} \textsf{max} \qquad & \sum_{k=1}^{N} M(\mathbf{G}_k) \\ \textsf{subject to} \quad & C(\mathbf{G}_k) < \alpha \cdot MaxConstraints. \end{aligned} \tag{31}$$

where N is the number of groups, $MaxConstraints$ is the maximum number of constraints that one wishes to feed to the LP, and $\alpha \geq 1$ is introduced so that the partitioning procedure becomes more flexible since the cost of a group is allowed to exceed $MaxConstraints$ temporarily. Now that the partitioning problem has been explicitly defined, we develop a multiple-way synchronous sequential circuit partitioning algorithm based on the algorithm proposed by Sanchis [20].

For each group $\mathbf{G}_k$, and each boundary latch L, define the connection number, $\mathbf{\Phi}$, as:

$$\mathbf{\Phi}_{G_k}(L) = |\{\mathbf{B} | \mathbf{B} \in \mathbf{G}_k \textit{ and } \mathbf{B} \in \mathbf{B}_L\}|. \tag{32}$$

Since each boundary latch connects exactly two blocks, $\mathbf{\Phi}_{G_k}(L) \in \{0, 1, 2\}$. In other words, if $\mathbf{B}_i \overset{L}{\leftrightarrow} \mathbf{B}_j$, then (a) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \notin \mathbf{G}_k, \mathbf{\Phi}_{G_k}(L) = 0$, (b) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k$, or vice versa, $\mathbf{\Phi}_{G_k}(L) = 1$, and (c) if $\mathbf{B}_i \in \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k, \mathbf{\Phi}_{G_k}(L) = 2$.

The *gain* associated with moving $\mathbf{B}$ from $\mathbf{G}_i$ to $\mathbf{G}_j$ is defined as

$$\mathbf{\Gamma}_{ij}(\mathbf{B}) \quad = \quad \sum_l (\tau(L_l)|L_l \in L_{\mathbf{B}} \; and \; \mathbf{\Phi}_{G_j}(L_l) = 1) - \sum_n (\tau(L_n)|L_n \in L_{\mathbf{B}} \; and \; \mathbf{\Phi}_{G_i}(L_n) = 2) \qquad (33)$$

The first term of (33) measures the benefit of moving $\mathbf{B}$ to $\mathbf{G}_j$, while the second measures the penalty of moving $\mathbf{B}$ out of $\mathbf{G}_i$.

Before beginning the partitioning procedure, the number of linear programming constraints, c_i, required for each block i is calculated using modified symbolic constraints propagation algorithm. If $c_i \geq MaxConstraints$ for some block $\mathbf{B}_i$, then it is placed in a group alone, and will not be processed later. Let $TotalConstraints = \sum_j (c_j | c_j < MaxConstraints)$. Each remaining block is put into one of the N' groups,

$$N' = \left\lceil \frac{TotalConstraints}{MaxConstraints} \right\rceil, \qquad (34)$$

such that for each group k, $C(\mathbf{G}_k) < MaxConstraints$. This is an integer knapsack problem, and many heuristic algorithms can be used to obtain an initial partition (see, for example, [25], Chapter 2). In some cases, it may be impossible to put all blocks into N groups without violating the restriction on $C(\mathbf{G}_k)$ above; if so, the number of groups may be larger than that given in (34).

Given the initial partition, the algorithm improves it by iteratively moving one block of partition from one group to another in a series of passes. A block is labeled *free* if it has not been moved during that pass. Each pass in turn consists of a series of iterations during each of which the free block with the largest gain is moved. During each move, we ensure that the number of constraints in a group does not violate the limit given by (31). The gain number, $\mathbf{\Gamma}_{ij}(\mathbf{B})$, is updated constantly as blocks are moved from one group to another. At the end of each pass, the partitions generated during that pass are examined and the one with the maximum objective value, as given by (31), is chosen as the starting partition for the next pass. Passes are performed until no improvement of the objective value can be obtained.

After the partitioning, we apply the optimization algorithm described in Section 5 to each group.

7 Experimental Results

The algorithms above were implemented in a program GALANT (**GA**te sizing using **L**inear programming **AN**d heuric**T**ics) on a Sun Sparc10 station. The test circuits include many of the ISCAS85 combinational benchmark circuits [26] and ISCAS89 synchronous sequential circuits [27]. Each cell in the standard-cell library has four different sizes of realization with different driving capabilities. The number of regions of piecewise linear approximation of delay, q, is set to be 4 empirically. Ideally, the larger q is, the more accurately we can approximate the delay function. However, increasing the value of q will directly increase the number of constraints in the LP formulation, which will result in larger run times. Therefore it is desirable to keep q small while maintaining acceptable approximation errors. Section 7.1 provides experimental results for the combination circuit optimization problem. The experimental results for synchronous sequential circuits with clock skew optimization are given in Section 7.2.

7.1 Experimental Results for Combinational Circuits

To prove the efficacy of the approach, a simulated annealing algorithm and Lin's algorithm [6] were implemented for comparison. The parameters used in Lin's algorithm have been tuned to give the best overall results. The simulated annealing algorithm that we have implemented is similar to that described in [28]. However, unlike in [28], all gate sizes were allowed to change during the simulated annealing procedure; while the run-times for this procedure were extremely high, the solution obtained can safely be said to be close to optimal.

The results of our approach, in comparison with Lin's algorithm and simulated annealing, are shown in Table 1. The test circuits include most of the ISCAS85 benchmarks, and vary in size from

Table 1: Performance comparison of GALANT with Lin's algorithm and simulated annealing.

Circuit	T_{spec}	Simulated Annealing		GALANT			Lin's Algorithm		
		Area (A_{SA})	Run time	Area (A_G)	Run time	$\frac{A_G}{A_{SA}}$	Area (A_L)	Run time	$\frac{A_L}{A_{SA}}$
c432	16.0	2372	19min 53s	2376	4.82s	1.002	2376	0.10s	1.002
	14.0	2515	21min 17s	2515	5.38s	1.000	2749	0.15s	1.092
	12.0	2950	24min 27s	2983	7.72s	1.011	-	-	-
c1355	14.0	8276	3h 32min	8276	1min 13s	1.000	8536	0.69s	1.031
	13.0	9258	3h 45min	9412	2min 14s	1.017	10319	1.28s	1.115
	12.5	10224	4h 12min	10417	3min 32s	1.019	-	-	-
c2670	17.0	17623	5h 22min	17623	4min 12s	1.000	18020	11.21s	1.023
	16.0	17772	5h 42min	17790	4min 30s	1.001	20150	19.7s	1.134
	14.0	18929	8h 12min	19079	7min 8s	1.008	-	-	-
c5315	20.0	36906	13h 46min	36954	11min 52s	1.001	37344	2.20s	1.012
	18.5	37438	14h 2min	37457	17min 28s	1.001	41248	4.32s	1.102
	17.0	38618	14h 43min	38863	19min 2s	1.006	-	-	-
c7552	18.0	50557	22h 5min	50604	35min 49s	1.001	51100	9.54s	1.011
	17.0	50740	23h 20min	51254	52min 27s	1.010	53772	34.57s	1.060
	16.0	52069	24h 5min	52563	1h 11min	1.009	-	-	-
Average Area Ratio						1.0057			-

160 gates to 3512 gates. It can be seen the accuracy of the results of our approach ranges from being as good as simulated annealing in some cases to an discrepancy of less than 2% in comparison with simulated annealing; the run times are considerably smaller than those for simulated annealing. It is observed that for all circuits, the chief component (over 95%) of the run-time was the linear programming algorithm; the heuristic was extremely fast in comparison.

Although Lin's algorithm runs much faster than GALANT, it does not always provide good results. For loose timing constraints, its solution is comparable to the result obtained using GALANT.

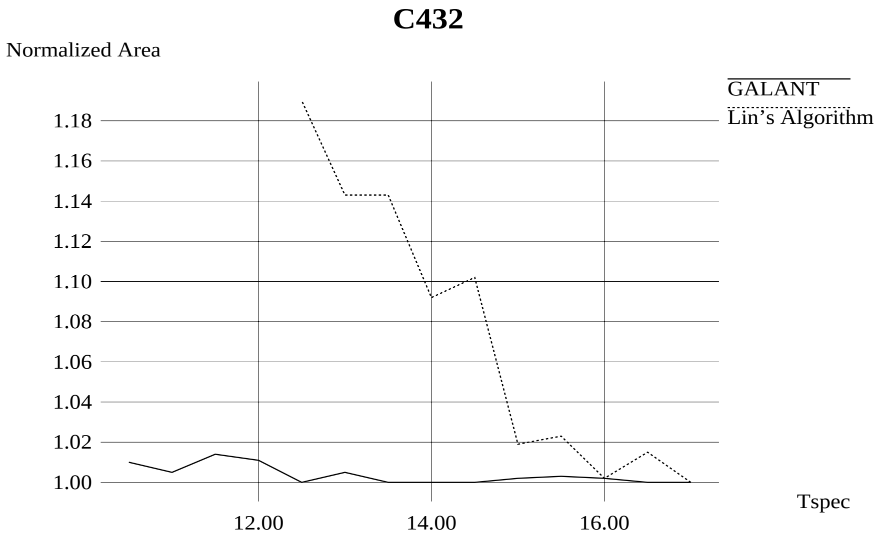

Figure 5: Comparison of Galant and Lin's algorithm against simulated annealing for c432.

For somewhat tight specifications, however, its solution becomes pessimistic. For even tighter delay constraints, it cannot obtain a solution at all. As mentioned previously, Lin's algorithm essentially is an adaptation of the TILOS algorithm [1] for continuous transistor sizing, with a few enhancements. While the TILOS algorithm is known to work reasonably well for the continuous sizing case, the primary reason for its success is that the change in the circuit in each iteration is very small. However, in the discrete sizing case, any change must necessarily be a large jump, and a TILOS-like algorithm is likely to give very suboptimal results.

A comparison of GALANT, Lin's algorithm, and simulated annealing on the circuit c432, for various timing specifications, is shown in Figure 5. In all cases, the solution obtained by GALANT is very close to the solution obtained by simulated annealing. In comparison with the results of Lin's algorithm, we find that GALANT provides results of substantially better quality, with reasonable run-times.

Table 2 shows the areas given by linear programming, Galant, and simulated annealing for C432

Table 2: Experimental results of areas given by linear programming, GALANT and simulated annealing for circuit c432.

Circuit	T_{spec}	Linear Programming	GALANT	Simulated Annealing
		Area	Area	Area
c432	17.0	2335	2337	2337
	16.5	2339	2350	2350
	16.0	2347	2376	2372
	15.5	2367	2402	2394
	15.0	2396	2424	2420
	14.5	2432	2467	2467
	14.0	2468	2515	2515
	13.5	2507	2563	2563
	13.0	2567	2658	2645
	12.5	2654	2801	2801
	12.0	2783	2983	2970
	11.5	2918	3139	3096
	11.0	3127	3315	3300
	10.5	3448	3597	3560

circuit under various delay constraints. The area given by LP could be considered as approximately the minimal area if we have a "continuous" library. A continuous library means we have infinite number of selections of the size of a gate, provided that the size w satisfies $w_{min} \leq w \leq w_{max}$ for each gate, where w_{min} and w_{max} are the minimal and maximal sizes in the original discrete library.

Finally, we compare the areas given by GALANT for two different sets of libraries. The results are shown in Figure 6. The first library, Library A has six different sizes for each logic gate in the library, while Library B has only two templates for each logic gate. The minimal and maximal sizes (w_{min} and w_{max} respectively) are the same for each logic gate in both libraries. That is, we only increase the number of available intermediate sizes in Library A compared to Library B. As we can see, using Library A consistently gives better results compared to using Library B. The improvement is about 10% for all timing constraints.

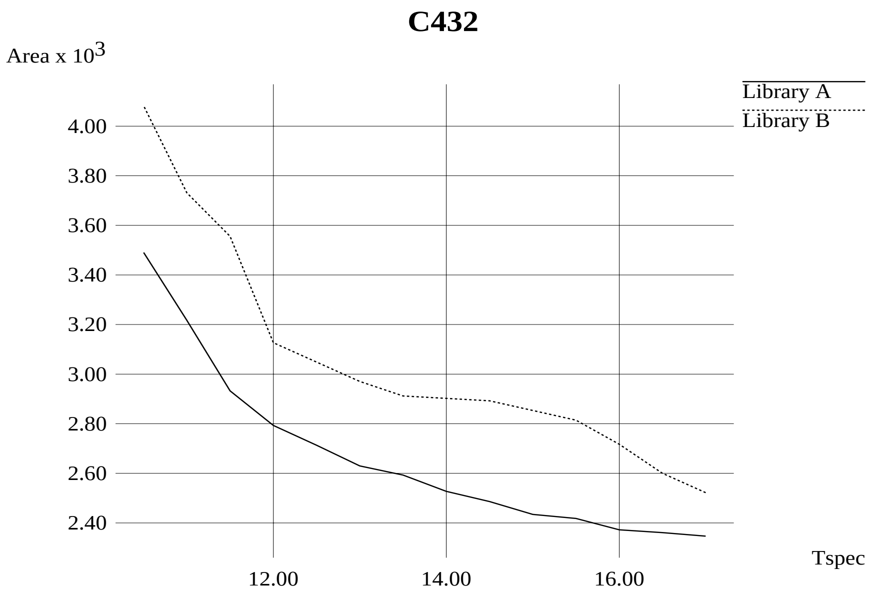

Figure 6: Experimental results of Galant using two different libraries. Library A has six different sizes for each logic gate, Library B has only two sizes for each gate.

7.2 Experimental Results for Synchronous Sequential Circuits

For all the experiments on sequential circuits, the clock skews are restricted to be less than half of the clock cycle to avoid excessively large difference between clock signal arrival times at different latches. In Table 3, the experimental results of fifteen ISCAS89 circuits are listed. For information on the number of PIs, POs, FFs, and logic gates in the circuits, see [27]. For each circuit, the number of longest-path delay constraints without using symbolic constraint propagation algorithm and the number of constraints pruned by the algorithm are given. It is clear that our pruning algorithm is very efficient. The number of delay constraints is reduced by more than 93% on the average. For a given desired clock period, the optimized results for both with and without clock skew optimization are shown. Depending on the structure of the circuits, the improvement over total area of the circuit ranges from 1.2% to almost 20%. As for the execution time, the runtime

Table 3: Performance comparison with and without clock skew optimization for ISCAS89 benchmark circuits.

Circuit	longest-path constraints			CP	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
	original	pruned	%		Area (A_1)	Runtime	Area (A_2)	Runtime	
s27	133	27	20.3%	3.75	151.12	0.32s	179.29	0.30s	0.842
s208	3276	214	6.5%	6.8	1404.00	3.32s	1745.25	3.06s	0.805
s298	4556	280	6.1%	6.5	2125.50	4.20s	2295.58	4.12s	0.926
s344	6720	401	6.0%	8.0	2093.00	7.10s	2400.67	6.91s	0.872
s349	6816	417	6.1%	8.0	2128.75	6.18s	2498.17	6.01s	0.852
s400	7824	656	8.4%	8.4	2314.00	8.19s	2515.50	7.13s	0.920
s420	11830	544	4.6%	12.0	2522.00	9.06s	2952.63	8.94s	0.854
s444	8592	830	9.7%	8.5	2463.50	11.55s	2724.04	7.22s	0.904
s526	11688	541	4.6%	6.5	3914.08	10.21s	4311.67	9.35s	0.908
s641	30402	1331	4.4%	22.0	4598.75	51.59s	4747.17	26.49s	0.969
s838	55948	2670	4.8%	10.5	6162.00	100.67s	7324.42	43.77s	0.841
s953	34470	1788	5.2%	10.5	5516.87	243.93s	5898.75	67.69s	0.935
s1196	32736	2241	6.8%	12.0	8550.21	288.15s	8752.42	97.43s	0.977
s1423	106379	7953	7.5%	35.0	9871.87	1069.75s	10151.38	80.71s	0.972
s5378	911854	6593	0.7%	10.0	29219.12	2633.78s	29717.53	1414.49s	0.983

ranges from about the same for some circuits, to less than double or triple for most circuits.

Table 4 provides some more in-depth experiments of two circuits, s838 and s1423. In this experiment, we try to minimize the area using different specified clock periods. As one can see, for s1423, the minimum clock period without clock skew optimization is about 32.5. On the other hand, using clock skew optimization, the minimum period can be as small as 22, which gives an almost 33% improvement in terms of clock speed. For s838, using clock skew optimization also gives an 30% improvement. Hence, using clock skew optimization can not only reduce the circuit area, but also allows a faster clock speed.

Table 5 gives the experimental results for the partitioning procedure. Since most of the ISCAS89 circuits consist of only one combinational block, we generated some synchronous sequential random logic circuits. The number of gates and FFs in those circuits are shown in Table

Table 4: Improving possible clocking speeds using clock skew optimization.

Circuit	# of PI's	# of PO's	# of FF's	# of gates	CP	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
						Area (A_1)	Runtime	Area (A_2)	runtime	
s838	35	2	32	390	10.5	6162.00	100.67s	7324.42	43.77s	0.841
					10.25	6165.25	102.18s	7365.58	45.30s	0.837
					10.0	6182.04	103.25s	-	-	-
					7.5	6637.58	130.20s	-	-	-
					6.75	7417.58	172.31s	-	-	-
					6.5	-	-	-	-	-
s1423	17	5	74	657	35.0	9871.87	1069.75s	10151.38	80.71s	0.972
					32.5	9998.63	1130.89s	10545.71	84.05s	0.948
					30.0	10154.08	1450.03s	-	-	-
					22.0	12178.83	1605.43s	-	-	-
					20.0	-	-	-	-	-

3. For each circuit, we conduct three experiments.

   1. First, we minimize the area using clock skew optimization, but without partitioning.

   2. Secondly, we minimize the circuit area using both clock skew optimization and partitioning.

   3. For comparison, we minimize the circuit with neither clock skew optimization nor partitioning.

From the table, it can be seen that the first approach is able to obtain the best result as expected. Since it considers all variables at the same time, it provides the best solution. However, the runtime is large. Compared to the first approach, the second approach runs much faster, at a very slight area penalty. Not surprisingly, the third approach gets the worst solution. We also note that the introduction of clock skew provides a significantly faster clock speed for circuit m1337. Although it has not been shown here, the same result also holds for m1783. For m1783, we also specify several different *MaxConstraints*. The result shows that as the specified *MaxConstraints* increases, the number of groups after partitioning decreases. As the number of groups decreases, the optimized solution using partitioning procedure improves, while the runtime only increases slightly. When $N = 6$, the solution is comparable to that without using partitioning, and the runtime is still far less than that without using partitioning.

Table 5: Performance comparison of the partitioning procedure.

Circuit	# of PI's	# of PO's	# of FF's	# of gates	# of blocks
m51	8	8	12	51	5
m144	16	2	18	144	9
m1337	51	53	97	1337	42
m1783	90	54	124	1783	43

Circuit	CP	with clock skew opt.						w/o clock skew opt.	
		w/o partitioning		wth partitioning					
		Area	Runtime	MaxCnstr†	N‡	Area	Runtime	Area	Runtime
m51	5.0	731	1.74s	300	2	813	1.50s	849	1.29s
m144	6.2	1872	6.11s	300	5	1953	3.32s	2410	2.87s
m1337	9.5	12364	135.35s	1500	6	12370	58.96s	13055	47.54s
	9.25	12353	151.34s	1500	6	12356	57.91s	-	-
	7.5	12685	171.92s	1500	6	12689	60.74s	-	-
	6.75	13049	186.61s	1500	6	13112	60.94s	-	-
	6.5	-	-	1500	6	-	-	-	-
m1783	9.5	18564	427.14s	300	16	18743	155.07s	21074	140.23s
				1000	8	18708	156.55s		
				2000	6	18572	159.93s		

† MaxCnstr = MaxConstraints, the maximum number of contraints.
‡ N, number of groups after partitioning.

8 Conclusion

In this paper, an efficient algorithm is presented to minimize the area taken by cells in standard-cell designed combinational circuits under timing constraints. We present a comparison of the results of our algorithm with the solutions obtained by our implementation of Lin's algorithm [6] and by simulated annealing. In [6], it was shown that Lin's algorithm is able to obtain better results than the technology mapping of MIS2 [29]. Although Lin's algorithm is fast, its solution becomes excessively pessimistic for tight delay constraints. For very tight timing constraints, it fails to obtain a solution at all. Experimental results show that our approach can obtain near-optimal solution (compared to simulated annealing) in a reasonable amount of time, even for very tight delay constraints. By adding additional linear programming constraints to account for short path delay [30], and slightly modifying the mapping and adjusting algorithm, the same approach can be used to tackle the double-sided delay constraints problem.

A unified approach to minimizing synchronous sequential circuit area and optimizing clock skews has also been presented. The skews at various latches in a circuit may be set using the algorithm in [31]. Traditionally, the circuit area of a synchronous sequential circuit is minimized one combinational subcircuit at a time. Our experiments have shown that this may lead to very suboptimal solution in some cases.

We formulate the discrete gate sizing optimization as a linear program, which enables us to integrate the equations with clock skew optimization constraints, taking a more global view of the problem. Experimental results show that this approach can not only reduce total circuit area, but also give much faster operational clock speed. For large synchronous sequential circuits, we also present a partitioning procedure. Our experiments show that our partitioning procedure is very effective in making our optimization algorithm run at a much faster speed, with no significant degradation in the quality of the solution.

Finally, the clock skew scheme may appear similar to *maximum-rate pipelining* technique used in pipelined computer systems [32]. However, the clock in a maximum-rate pipeline cannot be single-stepped or even slowed down significantly. This makes maximum-rate designs extremely hard to debug. In the clock skew scheme, by constrast, single-stepping is always possible [9]. Therefore circuits implemented using clock skew technique can be debugged without difficulties.

References

[1] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 326–328, 1985.

[2] S. S. Sapatnekar, V. B. Rao, and P. M. Vaidya, "A convex optimization approach to transistor sizing for CMOS circuits," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 482–485, 1991.

[3] J.-M. Shyu, A. Sangiovanni-Vincentelli, J. Fishburn, and A. Dunlop, "Optimization-based transistor sizing," *IEEE J. Solid-State Circuits*, vol. 23, pp. 400–409, Apr. 1988.

[4] M. R. Berkelaar and J. A. Jess, "Gate sizing in MOS digital circuits with linear programming," in *Proc. European Design Automation Conf.*, pp. 217–221, 1990.

[5] P. K. Chan, "Algorithms for library-specific sizing of combinational logic," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 353–356, 1990.

[6] S. Lin, M. Marek-Sadowska, and E. S. Kuh, "Delay and area optimization in standard-cell design," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 349–352, 1990.

[7] W. Li, A. Lim, P. Agrawal, and S. Sahni, "On the circuit implementation problem," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 478–483, 1992.

[8] S. Kirkpatrick, C. G. Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[9] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, pp. 945–951, July 1990.

[10] K. S. Hedlund, "AESOP : A tool for automated transistor sizing," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 114–120, 1987.

[11] D. Marple, "Performance optimization of digital VLSI circuits," Tech. Rep. CSL-TR-86-308, Stanford University, 1986.

[12] D. Marple, "Transistor size optimization in the Tailor layout system," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 43–48, 1989.

[13] H.-Y. Chen and S. Kang, "iCOACH: A circuit optimization aid for CMOS high-performance circuits," *Integration, the VLSI Journal*, vol. 10, pp. 185–212, Jan. 1991.

[14] S. H. Yen, D. H. Du, and S. Ghanta, "Efficient algorithms for extracting the K most critical paths in timing analysis," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 649–654, 1989.

[15] L. Cotten, "Circuit implementation of high-speed pipeline systems," *AFIPS Proc. 1965 Fall Joint Comput. Conf.*, vol. 27, pp. 489–504, 1965.

[16] T. Kirkpatrick and N. Clark, "PERT as an aid to logic design," *IBM J. of Res. Develop.*, vol. 10, pp. 135–141, Mar. 1966.

[17] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, pp. 291–307, Feb. 1970.

[18] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 175–181, 1982.

[19] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.

[20] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62–81, Jan. 1989.

[21] C.-W. Yeh, C.-K. Cheng, and T.-T. Y. Lin, "A general purpose multiple way partitioning algorithm," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 421–426, 1991.

[22] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1074–1085, Sept. 1992.

[23] C. Alpert and A. Kahng, "Geometric embeddings for faster and better multi-way netlist partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 743–748, 1993.

[24] P. Chan, M. D. Schlag, and J. Y. Zien, "Spectral K-way ratio-cut partitioning and clustering," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 749–754, 1993.

[25] M. M. Syslo, N. Deo, and J. S. Kowalik, *Discrete Optimization Algorithms.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

[26] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 663–698, 1985.

[27] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 1929–1934, 1989.

[28] M.-C. Chang and C.-F. Chen, "PROMPT3 - A cell-based transistor sizing program using heuristic and simulated annealing algorithms," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 17.2.1–17.2.4, 1989.

[29] E. Detjens, G. Gannot, R. Rudell, and A. Sangiovanni-Vincentelli, "Technology mapping in MIS," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 116–119, 1987.

[30] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Delay and area optimization for discrete gate sizes under double-sided timing constraints," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 9.4.1–9.4.4, 1993.

[31] R.-S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 242–249, Feb. 1993.

[32] P. Kogge, *The Architecture of Pipelined Computers*. New York, New York: McGraw-Hill, 1981.

Table and Figure Captions

Table. 1 Performance comparison of GALANT with Lin's algorithm and simulated annealing for ISCAS85 benchmark circuits.

Table. 2 Experimental results of areas given by linear programming, GALANT and simulated annealing for circuit c432.

Table. 3 Performance comparison with and without clock skew optimization for ISCAS89 benchmark circuits.

Table. 4 Improving possible clocking speeds using clock skew optimization.

Table. 5 Performance comparison of the partitioning procedure.

Fig. 1 The symbolic constraints propagation algorithm.

Fig. 2 An example illustrating symbolic delay propagation algorithm.

Fig. 3 The buffer insertion algorithm.

Fig. 4 An example illustrating the definition of a synchronous block.

Fig. 5 Comparison of Galant and Lin's algorithm against simulated annealing for c432.

Fig. 6 Experimental results of Galant using two different libraries.