July 1993

UILU-ENG-93-2228
DAC-39

Analog and Digital Circuits

TIMING AND AREA OPTIMIZATION FOR STANDARD-CELL VLSI CIRCUIT DESIGN

Weitong Chuang
Sachin S. Sapatnekar
Ibrahim N. Hajj

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2228 (DAC-39)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 1308 W Main St Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Joint Services Electronics Program 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-90-J-1270
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Timing and Area Optimization for Standard-Cell VLSI Circuit Design

12. PERSONAL AUTHOR(S)
Chuang, Weitong; Sapatnekar, Sachin S.; and Hajj, Ibrahim N.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 7/92 TO 7/93	93/07/08	48

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Discrete gate sizing; clock skew optimization; partitioning; MOS VLSI circuits

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

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. 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. A heuristic metric to measure the objective function of the partitioning problem is proposed. A multiple-way synchronous sequential circuit partitioning algorithm is then developed.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

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

Weitong Chuang† Sachin S. Sapatnekar‡ Ibrahim N. Hajj†

†Coordinated Science Laboratory and
Dept. of Electrical & Computer Engineering
University of Illinois at Urbana-Champaign

‡Department of Electrical Engineering
and Computer Engineering
Iowa State University

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. Traditional methods treat these two problems separately, which may lead to very sub-optimal solutions in some cases. We develop a novel unified approach to tackle them simultaneously. Experimental results show that by considering the two problems together, 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. A heuristic metric to measure the objective function of the partitioning problem is proposed. A multiple-way synchronous sequential circuit partitioning algorithm is then developed.

[1]This work was supported by Joint Services Electronics Program.

Manuscript received _____

Affiliation of authors:

Weitong Chuang
Coordinated Science Laboratory and
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign,
Urbana, IL 61801.
Tel : (217) 333-7498. E-mail : chuang@uivlsi.csl.uiuc.edu

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 transistor sizing problem is formulated as

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

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 strategy proposed for Boolean tree networks involves propagating the set of delay constraints, and pruning those that are infeasible. For general DAG's (directed acyclic graph), a cloning procedure is used to convert the DAG into an equivalent tree, whereby a vertex of fanout 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. As pointed out in [6], this procedure does not necessarily provide the optimal solution for a general DAG; moreover, this algorithm is of exponential complexity in the worst case.

The approach of Lin *et al.* [7] 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. Another algorithm proposed by Li *et al.* [6] is exact for series-parallel graphs, but is of exponential complexity. This work is extended to non-series-parallel circuits, whose structures are represented by general DAG's, and several heuristic techniques are used in conjunction with the algorithm, but no guarantees on optimality are made for such circuits. Both of these approaches are heuristics, 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 that is well-known to give optimal or near-optimal solutions.

The algorithm proposed in [8] does use simulated annealing; however, since simulated annealing is computationally expensive, a technique for variable pruning is used by this algorithm to reduced the computational complexity. An initial configuration is obtained using an algorithm similar to TILOS [1]. The set of gates that are left at minimum size at the end of this algorithm are

eliminated from the parameter space, under the assumption that these cells would not be sized in the final configuration. The sizes of the remaining cells are determined using a simulated annealing algorithm. One argument against such an algorithm is that it would have very large run-times for tight timing specifications, where a large number of cells would be sized by the TILOS-like heuristic.

Recently, Chuang *et al.* [9] proposed an efficient approach for solving the gate sizing problem under double-sided timing constraints. The approach first approximates delay curves of each gate in the circuit by piecewise-linear functions. With these piecewise-linear delay characteristics, the gate sizing problem can be formulated as a linear program. The obtained solution, which may contain impermissible gate sizes from the library, is then mapped onto the permissible set. This approach has been shown to be able to obtain near-optimal solutions (compared to simulated annealing) in a reasonable amount of time. However, the approach in [9] assumes the output capacitance of each gate is constant, which is not so in reality, since gate resizing alters the output capacitance of driving gates.

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. Unlike the approach used in [9] which assumes constant load capacitance of each gate, our approach handles the fanout capacitance problem properly. As will soon be obvious, this is not a straightforward exercise, as it greatly increases the number of constraints in the optimization problem. 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 [7].

1.2 Optimization for Synchronous Sequential Circuits

Optimization for synchronous sequentiall 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 (FF's) 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.

Example 1: Consider the circuit shown in Figure 1. If the gates in Block 1 are sized substantially, while those in Block 2 are close to their minimum sizes, then by allowing a clock skew at FF B, it is possible to increase the delay specification for Block 1 and decrease that for Block 2. This could reduce the area of Block 1 greatly, at the expense of a small increase in the area of Block 2. □

Example 2: Consider the synchronous sequential circuit shown in Figure 2. In addition to adjust clock skews at boundary latches (which will be defined in Section 6) as in Example 1, we can adjust clock skews at internal latches. By doing so, it is also possible to reduce the circuit area of the combinational block. □

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:

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

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

where $Maxdelay(i,j)$ and $Mindelay(i,j)$ are, respectively, the maximum and the minimum combinational delays between the two flip-flops, and P is the clock period. Fishburn [10] 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 P, with the constraints given by the inequalities in (2) and (3) above. In real design situations, however, P 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 FF's outside the system, and are clocked with zero (or constant) skew.

We first present an algorithm for small synchronous sequential circuits, and then show how it can be extended to arbitrarily large circuits. The algorithm works in three phases to solve the problem. In the first phase, the combined gate sizing and clock skew optimization problem is formulated as an LP. The solution of this LP provides us with a set of gate sizes that does not necessarily belong to the set of allowable sizes. Hence, in the second phase, we move from the LP solution to a set of allowable gate sizes, using heuristic techniques. At the end of the second phase, the set of allowable sizes obtained may not satisfy (2) and (3) simultaneously. Hence in the third stage, we fine-tune the longest path to satisfy (2) and satisfy the short path constraints in (3) by appropriately inserting delay buffers in the short path.

Finally, we consider arbitrarily large synchronous sequential circuits for which the size of the formulated LP's are 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

We assume that each gate in a standard cell library can be represented by an equivalent inverter such that the ratio of the p-transistor size to the n-transistor size of that inverter is a constant. Hence, the size of each gate can be parameterized by a single number, which we refer to as the gate size. An obvious choice for the gate size, which we use in this work, is the size of the n-transistor of the equivalent inverter. As in [1], the equivalent inverter is replaced by an RC circuit; the delay of this circuit is taken to be the delay of the inverter. In this case, the Elmore delay [11] of a cell G of size x is given by

$$D(x) = \frac{R_u}{x} \times C_{out} = R_{out} \times C_{out}, \tag{4}$$

Here R_u represents the on-resistance of a unit transistor, and C_{out} is the load capacitance of G. Since the gate terminal capacitance of a cell is proportional to its size, we have

$$C_{out} = \alpha \cdot y_1 + \beta + \alpha \cdot y_2 + \beta \cdots + \alpha \cdot y_f + \beta. \tag{5}$$

where $y_1, y_2, \ldots, y_f$ is the sizes of the cells to which G fans out; α and β are related to transistor gate terminal area and perimeter capacitances [3]. Thus, the delay function $D(x)$ of G is a function of $x, y_1, y_2, \ldots, y_f$.

Therefore the Elmore delay of a cell is a sum of functions of $g(x, y) = y/x$ and $h(x) = 1/x$. Figure 3 shows surface plots of the function y/x. Since the function $g(x, y) = y/x$ is relatively

smooth, it can be approximated by a convex piecewise linear function with q regions, of the form

$$PWL(x,y) = \begin{cases} a_1 \cdot x + b_1 \cdot y + c_1 & (x,y) \in \text{Region } \mathbf{R}_1 \\ a_2 \cdot x + b_2 \cdot y + c_2 & (x,y) \in \text{Region } \mathbf{R}_2 \\ \vdots & \\ a_q \cdot x + b_q \cdot y + c_q & (x,y) \in \text{Region } \mathbf{R}_q \end{cases} \tag{6}$$

$$= \max_{1 \le i \le q} (a_i \cdot x + b_i \cdot y + c_i) \qquad \forall (x,y) \in \bigcup_{1 \le i \le q} \mathbf{R}_i. \tag{7}$$

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

Similarly, we can approximate the function $h(x) = 1/x$ with a convex piecewise linear function.

Therefore, the gate delay $D(x, y_1, \ldots, y_f)$ of a gate with size x, and fanout gate sizes $y_1 \cdots y_f$ can be represented using a convex piecewise linear function with q regions, as follows:

$$\hat{D}(x, y_1, \cdots, y_f) = \begin{cases} \hat{a}_1 \cdot x + \hat{b}_{1,1} \cdot y_1 + \cdots + \hat{b}_{1,f} y_f + \hat{c}_1 & (x, y_1 \cdots y_f) \in \text{Region } \mathbf{R}_1 \\ \hat{a}_2 \cdot x + \hat{b}_{2,1} \cdot y_1 + \cdots + \hat{b}_{2,f} y_f + \hat{c}_2 & (x, y_1 \cdots y_f) \in \text{Region } \mathbf{R}_2 \\ \vdots & \\ \hat{a}_q \cdot x + \hat{b}_{q,1} \cdot y_1 + \cdots + \hat{b}_{q,f} y_f + \hat{c}_q & (x, y_1 \cdots y_f) \in \text{Region } \mathbf{R}_q \end{cases} \tag{8}$$

$$= \max_{1 \le i \le q} (\hat{a}_i \cdot x + \hat{b}_{i,1} \cdot y_1 + \cdots \hat{b}_{i,f} y_f + \hat{c}_i) \qquad \forall (x, y_1 \cdots y_f) \in \bigcup_{1 \le i \le q} \mathbf{R}_i. \tag{9}$$

It is worth pointing out that although we use Elmore delay model to estimate gate delays, our approach is not limited to this model. Given a standard-cell library, as long as the gate delay curve is relatively smooth (which is true for almost all practical designs), we can always approximate the delay function by a convex piecewise linear function.

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, 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 [1–8].

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; where 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

$$m_j + d_i \leq m_i, \quad \forall\, j \in Fanin(i). \tag{10}$$

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

$$\begin{array}{lll}
\textit{minimize} & \displaystyle\sum_{i=1}^{\mathcal{N}} \gamma_i \cdot x_i & \\
\textit{subject to} & \text{For all gates } i = 1 \cdots \mathcal{N} & \\
& m_j + d_i \leq m_i & \forall\, j \in Fanin(i) \\
& m_i \leq T_{spec} & \forall\ gates\ i\ at\ PO's \\
& d_i \geq \hat{D}(x_i, x_{i,1}, \ldots x_{i,fo(i)}) & \\
& x_i \geq Minsize(i) & \\
& x_i \leq Maxsize(i) &
\end{array} \tag{11}$$

where γ_i is the *area coefficient*, a constant associated with gate i. The area of gate i is $\gamma_i \cdot x_i$ if gate i has size x_i. The value of γ can be calculated based on the data given by the standard-cell library. $x_{i,1}, \ldots x_{i,fo(i)}$ are the sizes of the gates to which gate i fans out.

The above is a linear program in the variables x_i, d_i, m_i. It is worth noting that the entries in the constraint matrix are very sparse, as can be seen above, which makes the problem amenable to fast solution by sparse linear program approaches. Notice that the equalities of (8) are replaced here by inequalities, so as to satisfy (9).

3 Phase II : The Mapping Algorithm

The set of permissible sizes for gate i is $\mathcal{S}_i = \{x_{i,1} \cdots x_{i,p_i}\}$, where p_i is the cardinality of $\mathcal{S}_i$. The solution of the linear program would, in general, provide a gate size, x_i, that does not belong to $\mathcal{S}_i$. If so, we consider the two permissible gate sizes that are closest to x_i; we denote the nearest larger (smaller) size by x_{i+} (x_{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:

$$\begin{aligned} \text{For all } i = 1 \cdots \mathcal{N}: \quad & \text{Select } x_i = x_{i+} \text{ or } x_{i-}, \\ & \text{such that } \mathit{Delay} \leq T_{spec} \end{aligned}$$

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 algorithm first places all $\mathcal{N}$ gates in a queue, Q, in decreasing order of their worst-case signal arrival time, m_i. The longest path, P, from PI to the gate at the head of Q is found. The unmapped gates along P are mapped to permissible gate sizes using an implicit enumeration approach [12]. Once a gate size has been mapped onto a permissible size, it is said to be *processed*, and remains unchanged during the remainder of the enumeration process. A processed gate is removed from the queue Q.

After P has been processed, the process is repeated for the longest path to the gate that is now at the head of Q, until Q is empty. Thus, although the circuit could have an exponentially large number of paths, our algorithm needs to handle at most $\mathcal{N}$ of those paths.

Let G_1 be the gate that is currently at the head of the queue. Let $P = G_1, G_2, \ldots, G_{|P|}$ be the longest path from any PI to gate G_1, where $|P|$ is the number of gates on the path. The order of gates on the path is such that G_i fans out to G_{i-1}, $2 \leq i \leq |P|$. The *predecessor* (*successor*) of gate G_i on the path P is the gate G_{i+1} (G_{i-1}). Note that $G_{|P|}$ has no predecessor and G_1 has no successor.

Starting from G_1, we form a state space tree. Each node at level i in the state space tree is a *cell configuration*, which represents a possible realization of gate G_i. To help define a cell configuration, we introduce the following notation. Let

$C(i,j)$: the jth node at level i,

$anc(i,j)$: the ancestor node of $C(i,j)$,

$FO(i)$: the set of gates that gate i fans out to,

$area(i,x_i)$: the cell area of gate i when its size is x_i, $area(i,x_i) = \gamma_i \cdot x_i$ (see (11)),

$R^i_{out}(x_i)$: the equivalent resistance of gate i, corresponding to size x_i, that drives its load capacitances, $R^i_{out}(x_i) = R_u/x_i$ (see (4)),

$cap(i,j)$: the sum of the transistor gate terminal capacitances of gate j that are driven by gate i (see Figure 4 for an example),

$\Gamma^i(x_j)$: $cap(i,j)$, given that gate j is the predecessor of gate i on path P, and the size of gate j is x_j.

Definition 1 A *cell configuration*, $C(i,j)$ is a triple (X_{ij}, A_{ij}, D_{ij}),

$$X_{ij} = X_{C(i,j)} \in \{x_{i+}, x_{i-}\},$$

$$A_{ij} = A_{C(i,j)} = area(i, X_{ij}) + A_{anc(i,j)},$$

$$D_{ij} = D_{C(i,j)} = d_{ij} + D_{anc(i,j)},$$

where $d_{ij} = R^i_{out}(X_{ij}) \cdot \left[\sum_{k \in FO(i), k \neq i-1} cap(i,k) + \Gamma^i(X_{anc(i,j)}) \right]$.

A_{ij} is called the *accumulated area* from the root to $C(i,j)$, D_{ij} is called the *accumulated delay* from the root to $C(i,j)$, and d_{ij} is called the *configuration delay* associated with $C(i,j)$. Physically, d_{ij} corresponds to the delay of gate i, given that gate i has size X_{ij}, and gate $(i-1)$ has size $X_{anc(i,j)}$.

In the state space tree, each node has no more than two successors since there are at most two choices for the gate size. Every node in the tree corresponds to an assignment of sizes to those gates which lie on the path from the tree root to that node.

The root of the tree is, by definition, assigned a null cell configuration $(0,0,0)$. We begin with the unprocessed gate on the current path, P, that is closest to the POs, and implicitly enumerate the two possible realizations of each gate i, x_{i+} and x_{i-}. 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_i has been enumerated, the delay associated with the predecessor of G_i on path P can be calculated, and it can be enumerated. The process continues until all gates along P have been processed.

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, (X_{ij}, A_{ij}, D_{ij}), is *bounded* if there exists a cell configuration, (X_{ik}, A_{ik}, D_{ik}), at the same level of the tree such that

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

(2) $area(i, X_{ik}) \leq area(i, X_{ij})$, $A_{ik} < A_{ij}$ and $D_{ik} \leq D_{ij}$.

A somewhat similar procedure, which uses dynamic programming approach, was used in [6]; however, that procedure uses enumerative techniques on a much larger tree, where every gate

size is permissible (unlike our case of a binary tree). Since the number of possibilities is much larger, the heuristics that are required by that method to control the computational complexity are necessarily more ad hoc and hence, more inaccurate. Moreover, the procedure used in [6] fails to consider fanout capacitance load effects, since the sizes of the fanout modules of a certain module, $\mathcal{M}$, are not considered when the optimal implementations of $\mathcal{M}$ are being enumerated.

Example 1. In Figure 5, let G_1 be the current head of the queue, Q. Let G_2 be the predecessor of G_1, and G_3 that of G_2 on the longest path from a PI to G_1. There are two possible realizations for G_1, namely,

(1) one with $area(1, X_{1,1}) = 1.2$ and delay $d_{1,1} = 0.9$, and

(2) one with $area(1, X_{1,2}) = 0.8$ and delay $d_{1,2} = 1.1$.

If neither of node $C(1,1)$ or $C(1,2)$ is bounded, we proceed to construct the second level for both cell configurations. The two successors of node $C(1,1)$ in the tree represent two possible configurations of G_2 *if* G_1 is chosen to have the size with $area(1, X_{1,1}) = 1.2$.

Further, node $C(2,1)$ represents the configuration if G_2 is chosen to have size with $area(2, X_{2,1}) = 1.5$. Here, if the corresponding configuration delay of G_2, $d_{2,1} = 0.8$, then

- Accumulated delay of G_1 and G_2, $D_{2,1} = 1.7$
- Accumulated area of G_1 and G_2, $A_{2,1} = 2.7$

Similarly, node $C(2,2)$ represents the situation if G_1 is chosen to have size with cell area 1.2 and G_2 with cell area 1.0. If the configuration delay of G_2, $d_{2,2} = 1.2$, then

- Accumulated delay $D_{2,2} = 2.1$
- Accumulated area $A_{2,2} = 2.2$

The entries of node $C(2,3)$ and $C(2,4)$ can be calculated similarly.

Now, notice that nodes $C(2,1)$ and $C(2,3)$ have the same gate area for G_2, while node $C(2,3)$ has less accumulated area and accumulated delay than node $C(2,1)$. Therefore, node $C(2,1)$ is bounded and it is not necessary to enumerate the descendants of $C(2,1)$. Similarly, $C(2,2)$ is bounded since $C(2,4)$ has superior configuration to $C(2,2)$. □

For every path P in the circuit, we define a quantity known as *maximum path delay*, (MPD), as follows:

$$MPD(P) = \begin{cases} \min\limits_{j \in FO(i)} (m_j - d_j), & \text{if gate } i \text{ is not at a PO.} \\ \min[\min\limits_{j \in FO(i)} (m_j - d_j),\ T_{spec}], & \text{if gate } i \text{ is at a PO.} \end{cases} \tag{12}$$

where gate i is the gate that lies at the end of path P. Note that even 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 MPD. Maximum path delay physically corresponds to the maximal delay that can be assigned to path P before its effect is propagated beyond the gate G_i at the end of the path.

After the state space tree for the longest path P has been constructed, the algorithm examines the cell configurations at the leaf nodes of the tree. The cell configuration, $C(|P|,n)$, which satisfies the following requirements, is selected.

(1) $D_{|P|,n} \leq MPD(P)$,

(2) $D_{|P|,n} \geq D_{|P|,i}$ $\forall\ C(|P|,i)$ such that $D_{|P|,i} \leq MPD(P)$.

In requirement (2), instead of using $A_{|P|,n} \leq A_{|P|,i}$ as the criterion, we use $D_{|P|,n} \geq D_{|P|,i}$. This is because we do not want to perturb the solution obtained from the linear programming too much. This way, it is expected that no change in gate size takes the circuit delay radically away from T_{spec}.

By performing a trace-back from $C(|P|, n)$ to the root of the tree, the size of each gate along P is determined from the cell configurations at each traversed node of the tree.

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, to G_p. The *path slack* of P is defined as

$$Pslack(P) = T_{spec} - m_p \tag{13}$$

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

$$delay(G_i) = R_{out}^{i+1} \cdot C_{out}^{i+1} + R_{out}^{i} \cdot C_{out}^{i} \tag{14}$$

$$\Delta delay(G_i) = R_{out}^{i+1} \cdot \Delta C_{out}^{i+1} + \Delta R_{out}^{i} \cdot C_{out}^{i} \tag{15}$$

where R_{out}^{i} and C_{out}^{i} 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_{out}^{i} and C_{out}^{i+1}.

Example 2 [13]. Consider the chain of three CMOS inverters shown in Figure 6(a). Let the width of both the n-type and p-type transistors in gate 2 be w_2, and let D be the total delay through the three gates. Consider the effect of increasing w_2, while keeping the size of the transistors in gates 1 and 3 fixed. This causes the magnitude of output current of gate 2 to increase, thus the time required, d_2, for gate 2 to drive its output signal will decrease monotonically (Figure 6(b)).

However, increasing w_2 also increases the capacitive load on the output of gate 1, thus slowing down the output transition of the first gate. Beyond a certain point $w_2 = A$, the total delay, D, starts to increase with respect to w_2, which shows the nonmonotonicity of the delay-area relationship. □

From the above example, it is clear that for each of the gate along P, we must consider either increasing or decreasing its size (unless, of course, a gate is already of the largest or smallest possible size). After calculating the local delay difference associated with each of the gates along path P, we select the largest one, $\Delta delay(G_n)$, which satisfies

$$\Delta delay(G_n) < Pslack(P) \tag{16}$$

and change the size of G_n accordingly. If none of the local delay differences satisfies (16), 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 x_{i+} and x_{i-} here.

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 [14]. Consider a synchronous sequential digital system with flip-flops (FF's). Let s_i denote the individual delay between the central clock source and flip-flop FF_i, and let P 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 $s_i + d_{ij} > s_j + P$, 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 [10], 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 FF's, the lower and upper bounds $MIN(i,j)$ and $MAX(i,j)$ $(1 \leq i,j \leq \mathcal{L}$, $\mathcal{L}$ being the total number of FF's 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

$$s_i + MIN(i,j) \geq s_j + T_{hold}. \tag{17}$$

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 P, so the next clock edge arrives at FF_j at $s_j + P$. Therefore,

$$s_i + T_{setup} + MAX(i,j) \leq s_j + P. \tag{18}$$

Inequalities (17) and (18) dictate the correct operation of a synchronous sequential system.

Our problem requires us to represent path delay constraints between *every* pair of FF's. This may be achieved by performing PERT [15] on the circuit and setting all FF's except the FF of interest (say FF_i) to $-\infty$ (∞) for the longest (shortest) delay path to from FF_i to all FF's, and the arrival time at the FF of interest is set to 0 [10]. 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 PI's (the outputs of FF's are considered as pseudo PI's) up to the output of G_k.

$$p_j + d_k \geq p_k, \quad \forall\, j \in Fanin(k). \tag{19}$$

To represent path delays between every pair of FF's, 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

$$\begin{aligned}
&\textit{minimize} && \sum_{k=1}^{\mathcal{N}} \gamma_k \cdot x_k && \\
&\textit{subject to} && d_k \geq D(x_k, x_{k,1}, \ldots x_{k,fo(k)}), && 1 \leq k \leq \mathcal{N} \\
& && x_k \geq Minsize(k), && 1 \leq k \leq \mathcal{N} \\
& && x_k \leq Maxsize(k), && 1 \leq k \leq \mathcal{N} \\
& && \text{For all FF } i, && 1 \leq i \leq \mathcal{L} \\
& && \quad s_i + p_k^i \geq s_j + T_{hold} && 1 \leq j \leq \mathcal{L},\ k = Fanin(FFj) \\
& && \quad s_i + T_{setup} + m_k^i \leq s_j + P_{spec} && 1 \leq j \leq \mathcal{L},\ k = Fanin(FFj) \\
& && \quad \text{For all gates } k = 1, \cdots, \mathcal{N} && \\
& && \quad\quad m_l^i + d_k \leq m_k^i, && \forall\, l \in Fanin(k) \\
& && \quad\quad p_l^i + d_k \geq p_k^i, && \forall\, l \in Fanin(k)
\end{aligned} \tag{20}$$

The above is a linear program in the variables x_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 (20). 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 8). Let $\mathcal{F} = \sum_{i=1}^{\mathcal{N}} Fanin(i)$, where $\mathcal{N}$ is 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 FF's 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 $q = 5$. 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 PI's fanout cone. Also, we collapse constraints on chains of gates wherever possible (line 6 in Figure 7).

The synchronous sequential circuit is first levelized. For this purpose, the inputs of FF's are considered as pseudo PO's the outputs of FF's are considered as pseudo PI's. 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 $w_i \in \{0, 1\}$ is introduced to indicate its status. w_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, $w_i = 0$.

The algorithm for propagating delay constraints symbolically is given in Figure 7. 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, w_j and $mstring(j)$ are initialized by setting $w_j = 0$, and $mstring(j)$ to the null string. At line 5, we check if $w_l = 0$ 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. Next, at line 6, we check whether exactly one of all of gate k's input gates, say gate l', has a non-empty *mstring*, others are have null *mstring*'s. If so, we may continue to propagate the constraint. This is implemented by concatenating $mstring(l')$ and "d_k", and storing the resulting string in $mstring(k)$. Also w_k is set to 1 to indicate that further propagation is required at this gate. 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 9). For each input gate whose *mstring* is non-empty ($w_l = 1$), we need a delay constraint (line 12).

Example 3: Figure 8 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 $w_{14} = 1$. Propagating this further, we find that similarly, $mstring(15)$ = "$m_{11}^1 + d_{14} + d_{15}$", and $w_{15} = 1$. Finally, for gate 16, we apply lines 9 through 12, and find that we must introduce a variable m_{16}^1, and set $w_{16} = 1$. We also write down the two constraints shown in the figure and add these to the set of LP constraints. □

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

$$\begin{aligned} \text{For all } k = 1 \cdots \mathcal{N}: \quad & \text{Select } x_k = x_{k+} \text{ or } x_{k-}, \text{ such th at} \\ & \text{for all FF's } \; 1 \leq i\,, j \leq \mathcal{L} \\ & \quad s_i + Maxdelay(i,j) + T_{setup} \leq s_j + P_{spec} \\ & \quad s_i + Mindelay(i,j) \geq s_j + T_{hold} \end{aligned}$$

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 PO's at the inputs of FF's), the required maximum (minimum) signal arrival times, $req_l(j)$ ($req_s(j)$), can be expressed as

$$\begin{aligned} req_l(j) &= s_j + P_{spec} - T_{setup} \\ req_s(j) &= s_j + T_{hold} \end{aligned} \tag{21}$$

The path slack then can be defined as

$$Pslack(P_l(n)) = req_l(n) - m_n \tag{22}$$

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

$$Gslack(i) = \begin{cases} \min\limits_{j \in FO(i)} \{m_j + Gslack(j) - (d_j + m_i)\}, & \text{if gate } i \text{ is not at a PO.} \\ \min\{\min\limits_{j \in FO(i)} [m_j + Gslack(j) - (d_j + m_i)],\ (req_l(i) - m_i)\}, & \text{if gate } i \text{ is at a PO.} \end{cases} \tag{23}$$

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 PO's or FF's, in terms of long path delay. Therefore, it also tell us the maximum delay that a delay buffer can have if we are to insert a delay buffer at the output of gate i.

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 PO's and all FF's. The gate slack of each gate is determined recursively using (23).

The algorithm for inserting buffers is shown in Figure 9. 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 $delay'(G_{ni})$.

Example 4: In Figure 10, let gate 4 be connected to some FF. The required maximum arrival time (req_l) is 4.8, and the required minimum arrival time (req_s) is 1.3. The actual long-path delays (m_i) and short-path delays (p_i) for all gates are as indicated. The gate slack of each gate is calculated and shown in the figure. Since gate 4 violates shortest-path delay requirement, the shortest-path to it, $P_s(4)$, is found; this can be seen to include gate 3. Since the gate slack of gate 3 is 1.0, we can insert a delay buffer between gate 3 and 4. If $delay(3) = 0.5$, the delay after introducing the

buffer, $delay'(3) = 0.4$, and $delay(bf) = 0.3$, then the new value of p_4 is 1.4, which satisfies $req_s(4)$.

□

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 FF's 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 [16,17]. 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 one shown in Figure 2.

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 $\mathbf{N}$ is a partition of the sequential blocks of $\mathbf{N}$ 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

$\tau_{in}(L)$ = maximum combinational delay from any boundary latch to L in the unsized circuit,

$\tau_{out}(L)$ = maximum combinational delay from L to any boundary latch in the unsized circuit,

$$\tau(L) = \begin{cases} \tau_{in}/\tau_{out} & \text{if } \tau_{in} \geq \tau_{out} \\ \tau_{out}/\tau_{in} & \text{if } \tau_{in} < \tau_{out} \end{cases} \tag{24}$$

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

$$\mu_{ij} = \sum_{\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j} \tau(L_k) \tag{25}$$

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:

$$\begin{aligned} &max && \sum_{k=1}^{N} M(\mathbf{G}_k) \\ &subject\ to && C(\mathbf{G}_k) < \alpha \cdot MaxConstraints. \end{aligned} \tag{26}$$

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 [16].

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

$$\mathbf{\Phi}_{G_k}(L) = |\{\mathbf{B} | \mathbf{B} \in \mathbf{G}_k \ and \ \mathbf{B} \in \mathbf{B}_L\}|. \tag{27}$$

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

$$\mathbf{\Gamma}_{ij}(\mathbf{B}) = \sum_l (\tau(L_l) | L_l \in L_{\mathbf{B}} \ and \ \mathbf{\Phi}_{G_j}(L_l) = 1) - \sum_n (\tau(L_n) | L_n \in L_{\mathbf{B}} \ and \ \mathbf{\Phi}_{G_i}(L_n) = 2) \tag{28}$$

The first term of (28) 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,

$$N' = \left\lceil \frac{TotalConstraints}{MaxConstraints} \right\rceil, \tag{29}$$

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, [18], 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 (29).

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 (26). 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 (26), 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 [19] and ISCAS89 synchronous sequential circuits [20]. Each cell in the standard-cell library has five different sizes of realization with different driving capabilities. 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 [7] 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 [8], which is briefly described in Section 1.1. However, unlike in [8], 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. Although simulated annealing does not guarantee the global optimal solution, a well-designed algorithm and a very slow annealing procedure can provide a solution that is very close to the global optimum.

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 160 gates (824 transistors) to 3512 gates (15,396 transistors). It can be seen the accuracy of the results of our approach ranges from being as good as simulated annealing for c499 to an discrepancy of 7.4% in comparison with simulated annealing; the run times are considerably smaller than those for simulated annealing. It is also worth pointing out that this procedure finds the solution for the circuit c6288, a 16-bit multiplier with a large number of paths, in a very reasonable amount

of time. It is likely that such a circuit would cause immense problems for an approach such as [5] which depends on path enumeration.

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. For somewhat tight specifications, however, its solution becomes excessively pessimistic. For even tighter delay constraints, it cannot obtain 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.

Table 2 shows the amount of time taken by the mapping and adjusting algorithm in comparison with the time required to solve the linear program, for some of the results in Table 1. It is clear that for all circuits, the chief component (over 98%) of the run-time was the linear programming algorithm; the heuristic was extremely fast in comparison. The discrepancy between the sum of LP solution time and the time required for mapping and adjusting in Table 2, and the total run-time in Table 1 is attributable to the preprocessing step which performs miscellaneous administrative steps such as reading in the circuit description and levelizing the circuit.

A comparison of the run-times for GALANT, Lin's algorithm, and simulated annealing on the circuit c432, for various timing specifications, is shown in Table 3. It is clear that GALANT is orders of magnitude faster than simulated annealing, with results of comparable quality. It can be seen that as the timing specification becomes more tight, the area increases; the increase in area is very rapid for tighter timing specifications. 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.

The run-time of GALANT is seen to go up as the timing specifications become tighter. This can be ascribed to the fact that there are many more solutions of the linear program that are close to the optimal solution, and hence the simplex procedure takes a longer time. This is in contrast with the case for a loose timing specification, where most gates are at minimum size at the solution, and the vertices of the feasible region where these gates are at nonminimum sizes are clearly suboptimal.

Finally, the circuit areas obtained using GALANT after LP phase as well as after mapping and adjusting phases are shown in Table 4. It can be seen that our mapping and adjusting algorithms are very efficient in that the final total areas are close to those given by LP. On the average, the final circuit areas after mapping and adjusting phases are within 1.7% of those obtained at the conclusion of the linear programming phase. Also notice that for some cases, the area given by the linear programming is slightly larger than that by simulated annealing. This could be attributed to the deficiency of the piecewise linear approximations to the actual delay curves.

7.2 Experimental Results for Synchronous Sequential Circuits

In Table 5, the experimental results of fifteen ISCAS89 circuits are listed. For information on the number of PI's, PO's, FF's, and logic gates in the circuits, see [20]. 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 (P_{spec}), 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 ranges from about the same for some circuits, to less than double or triple for most circuits.

One may raise the question of whether it is worthwhile to minimize circuit area through clock skew optimization, since the reduction of area is not very significant for some circuits. However, Table 6 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 7 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 FF's in those circuits are shown in Table 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.

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 [7] and by simulated annealing. In [7], it was shown that Lin's algorithm is able to obtain better results than the technology mapping of MIS2 [21]. 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 [9], 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 [22]. 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 schema. 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.

The major bottleneck of our approach was the time required to solve the linear program. Our approach used a linear program which is solved using a package available in the public domain [23], whose base is a sparse matrix dual simplex linear program solver. It is possible to reduce the CPU usage using vector processors; as pointed out in [23], the CPU usage can be reduced by about 40% on an Alliant FX/8 machine. Although the computational complexity of simplex method can be exponential in the worst case, it has been observed that for most practical problems, the complexity ranges from $O((1/n + 1/(m - n_1))^{-1})$ to $O((1/n + 1/(m - n + 1) - 1/m)^{-1})$ for m inequality constraints and n variables [24]. Other polynomial-time linear programming algorithms such as Karmarkar's algorithm [25] may also be employed; however, in practice, its average run-time has been found to be similar to that of the simplex algorithm.

Finally, the clock skew scheme may appear similar to *maximum-rate pipelining* technique used in pipelined computer systems [26]. 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 [10]. 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] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, Jan. 1948.

[12] E. Horowitz and S. Sahni, *Fundamemtals of Computer Algorithms.* Rockville, Maryland: Computer Science Press, 1978.

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

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

[15] T. Kirkpatrick and N. Clark, "PERT as an aid to logic design," *IBM Journal of Research and Development*, vol. 10, pp. 135–141, Mar. 1966.

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

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

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

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

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

[21] 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.

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

[23] M. Berkelaar, *LP_SOLVE USER'S MANUAL*, June 1992.

[24] R. G. Parker and R. L. Rardin, *Discrete Optimization.* San Diego, California: Academic Press, Inc., 1988.

[25] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *16th Annual ACM Symposium on Theory of Computing*, pp. 302–311, 1984.

[26] 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.

Circuit	T_{spec}	Simulated Annealing		GALANT			Lin's Algorithm		
		Area (A_{SA})	Runtime	Area (A_G)	Runtime	$\frac{A_G}{A_{SA}}$	Area (A_L)	Runtime	$\frac{A_L}{A_{SA}}$
c432	16.0	2360	20m 29s	2389	4.75s	1.012	2429	0.28s	1.029
	14.0	2475	20m 38s	2513	4.91s	1.015	2715	0.20s	1.097
	12.0	2793	24m 7s	2887	5.50s	1.025	5996	0.14s	2.147
c499	9.0	3809	30m 3s	3809	8.60s	1.000	3809	0.16s	1.000
	8.0	4039	35m 34s	4039	9.14s	1.000	4791	0.48s	1.186
	7.0	4916	37m 54s	5279	11.05s	1.074	6467	0.33s	1.316
c880	12.0	5972	55m 51s	5980	23.57s	1.001	6445	0.77s	1.079
	11.0	6106	57m 25s	6177	25.57s	1.010	-	-	-
	10.0	6377	1h 1m	6479	27.59s	1.016	-	-	-
c1355	17.0	7522	2h 51m	7527	42.16s	1.000	7704	0.87s	1.024
	15.0	7700	3h 5m	7700	44.81s	1.000	7856	0.89s	1.020
	14.0	8226	3h 29m	8621	49.66s	1.048	8875	2.07s	1.079
c1908	20.0	11245	5h 3m	11248	1m 40s	1.000	11375	2.59s	1.011
	18.0	11439	5h 15m	11476	1m 52s	1.003	17938	3.20s	1.563
	15.0	13663	6h 1m	13740	3m 28s	1.006	-	-	-
c2670	20.0	17451	7h 15m	17459	3m 5s	1.000	17617	2.48s	1.010
	18.0	17518	7h 32m	17533	3m 12s	1.000	19281	4.06s	1.101
	15.0	17977	8h 1m	18098	4m 19s	1.007	-	-	-
c3540	24.0	24430	10h 2m	24448	5m 57s	1.001	24486	3.01s	1.002
	20.0	25040	10h 19m	25127	8m 25s	1.003	29767	10.89s	1.189
	18.5	25611	10h 47m	26199	10m 2s	1.023	-	-	-
c5315	22.0	36651	12h 2m	36662	11m 5s	1.000	37296	15.96s	1.020
	20.0	36853	12h 58m	36957	12m 48s	1.003	44701	26.79s	1.213
	17.0	38269	13h 21m	38756	17m 22s	1.013	-	-	-
c6288	75.0	32886	15h 42m	32908	14m 47s	1.000	36889	12.90s	1.122
	72.5	32976	16h 4m	33026	14m 52s	1.002	57557	8.57s	1.745
	70.0	33118	16h44m	33296	16m 4s	1.005	61634	9.40s	1.861
c7552	20.0	50123	20h 24m	50152	27m 26s	1.001	50910	25.70s	1.016
	18.0	50425	21h 16m	50469	32m 15s	1.001	62965	59.54s	1.025
	16.0	51968	22h 1m	52376	57m 34s	1.008	-	-	-
Average Area Ratio						1.009			1.206

Table 2: Execution times for the Linear Program and the Mapping & Adjusting Algorithms.

Circuit	T_{spec}	LP solution	Mapping & Adjusting
c432	12	5.37s	0.05s
c499	7.0	10.73s	0.24s
c880	10	27.21s	0.19s
c1355	14	45.82s	3.46s
c1908	15	3m 26s	1.83s
c2670	15	4m 17s	0.89s
c3540	20	8m 4s	1.45s
c5315	17	17m 18s	3.71s
c6288	70	15m 56s	4.20s
c7552	16	57m 21s	9.48s

Table 3: Performance comparison for c432.

Circuit	T_{spec}	Simulated Annealing		GALANT			Lin's Algorithm		
		Area (A_{SA})	Runtime	Area (A_G)	Runtime	$\frac{A_G}{A_{SA}}$	Area (A_L)	Runtime	$\frac{A_L}{A_{SA}}$
c432	17.5	2330	8m 3s	2331	4.57s	1.000	2363	0.15s	1.014
	17.0	2334	9m 37s	2335	4.71s	1.000	2440	0.21s	1.045
	16.5	2341	10m 27s	2346	4.66s	1.002	2526	0.27s	1.079
	16.0	2360	10m 51s	2389	4.75s	1.012	2429	0.28s	1.029
	15.5	2379	9m 54s	2390	4.79s	1.005	2549	0.21s	1.071
	15.0	2411	11m 31s	2421	4.87s	1.004	2645	0.21s	1.097
	14.5	2439	12m 27s	2445	4.91s	1.002	2645	0.22s	1.084
	14.0	2475	12m 36s	2513	4.91s	1.015	2715	0.20s	1.097
	13.5	2553	12m 34s	2608	5.15s	1.022	2829	0.29s	1.097
	13.0	2616	12m 53s	2689	5.21s	1.028	3200	0.19s	1.108
	12.5	2685	13m 35s	2750	5.35s	1.024	3869	0.23s	1.441
	12.0	2816	19m 57s	2887	5.50s	1.025	5996	0.14s	2.129
	11.5	3043	20m 49s	3400	5.97s	1.117	-	-	-
	11.0	3302	21m 1s	3533	7.02s	1.070	-	-	-
	10.5	3619	27m 57s	3683	6.89s	1.018	-	-	-
	10.0	3915	29m 50s	4370	7.51s	1.116	-	-	-
Average Area Ratio						1.022			1.191

Table 4: Performance comparison of GALANT's Mapping and Adjusting Algorithms.

Circuit	T_{spec}	GALANT			Simulated Annealing
		Area after LP (A_{LP})	Final Area (A_G)	$\frac{A_G}{A_{LP}}$	Area
c432	16.0	2345	2389	1.019	2360
	14.0	2468	2513	1.018	2475
	12.0	2741	2887	1.053	2793
c499	9.0	3796	3809	1.003	3809
	8.0	3948	4036	1.022	4039
	7.0	4711	5279	1.121	4916
c880	12.0	5952	5980	1.005	5972
	11.0	6072	6177	1.017	6106
	10.0	6387	6479	1.014	6377
c1355	17.0	7507	7527	1.003	7522
	15.0	7670	7700	1.004	7700
	14.0	8015	8621	1.076	8226
c1908	20.0	11233	11248	1.001	11245
	18.0	11436	11476	1.003	11439
	15.0	12918	13740	1.064	13663
c2670	20.0	17451	17459	1.000	17451
	18.0	17515	17533	1.001	17518
	15.0	17895	18098	1.011	17977
c3540	24.0	24423	24448	1.001	24430
	20.0	24848	25127	1.011	25040
	18.5	25443	26199	1.030	25611
c5315	22.0	36650	36662	1.000	36651
	20.0	36832	36957	1.003	36853
	17.0	38280	38756	1.012	38269
c6288	75.0	32890	32908	1.000	32886
	72.5	32980	33026	1.001	32976
	70.0	33118	33296	1.005	33118
c7552	20.0	50122	50152	1.001	50123
	18.0	50377	50469	1.002	50425
	16.0	51620	52376	1.015	51968
Average Area Ratio				1.017	

Table 5: Performance comparison with and without clock skew optimization for ISCAS89 benchmark circuits.

Circuit	longest-path constraints			P_{spec}	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

Table 6: Improving possible clocking speeds using clock skew optimization.

Circuit	# of PI's	# of PO's	# of FF's	# of gates	P_{spec}	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	-	-	-	-	-

Table 7: 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	P_{spec}	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.

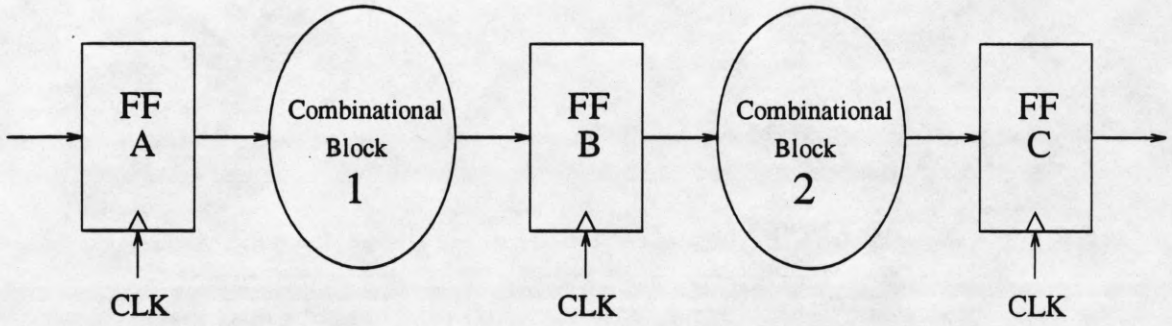

Figure 1: The advantages of nonzero clock skew.

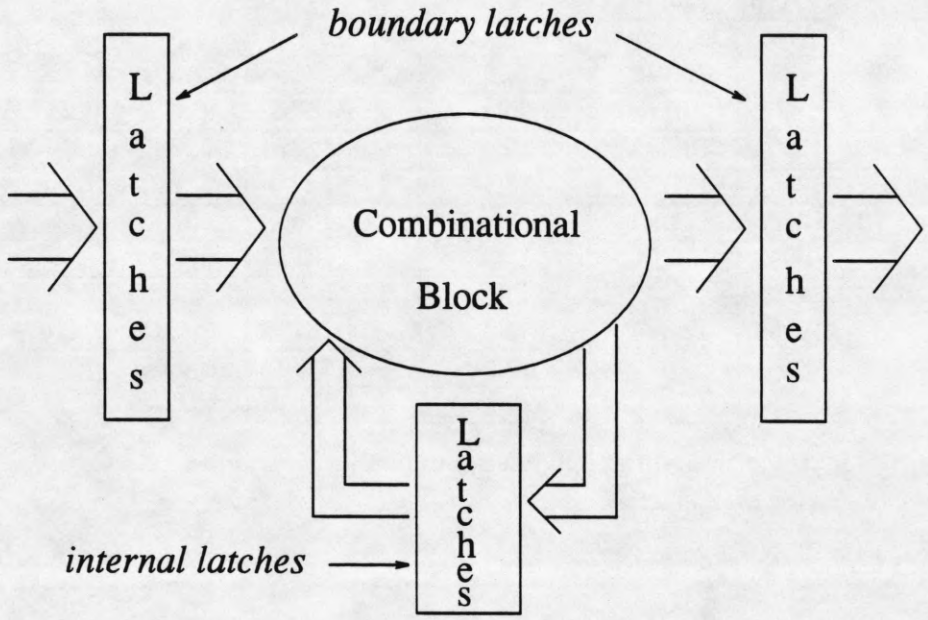

Figure 2: An example illustrating the definition of a synchronous block.

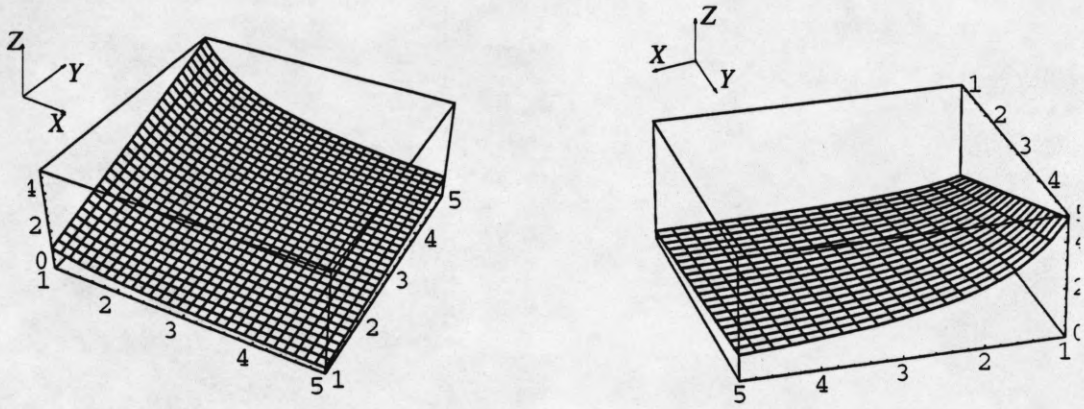

Figure 3: Surface plots of the function $z = y/x$ from two different view points.

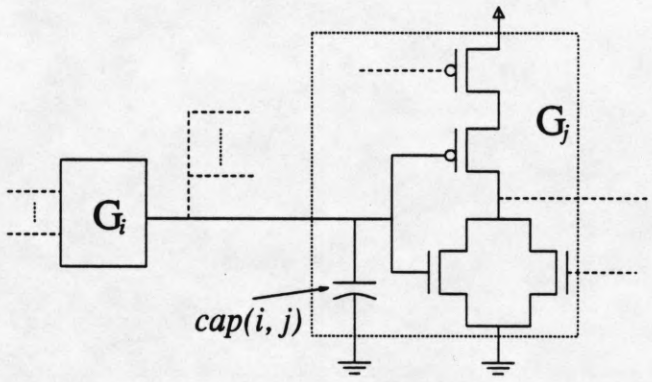

Figure 4: An example illustrating the definition of $cap(i, j)$.

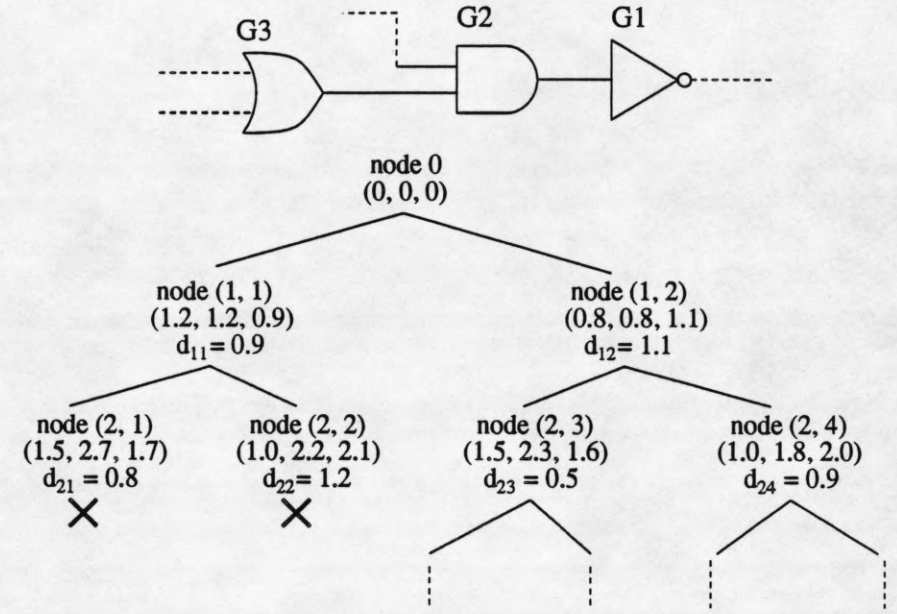

Figure 5: An example illistrating the construction of state space tree in the mapping algorithm.

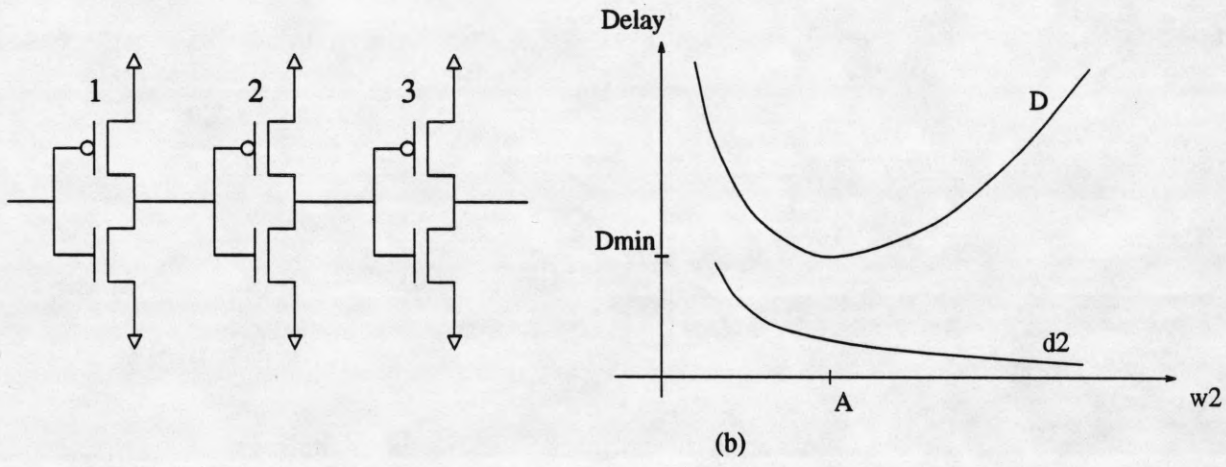

Figure 6: (a) A chain of three inverters. (b) Effect of transistor sizes on delay for the three-inverter chain.

```
ALGORITHM Symbolic_propagation()
 1.    for i = 1 to L
 2.      w_j ← 0, mstring(j) ← "", ptring(j) ← "" for all gates and PI's;
 3.      for j = 1 to max_level
 4.        for each gate k at level j
 5.          if ( w_l = 0 for all l ∈ fanin(k) ); /* do nothing */
 6.          if ( among all l ∈ fanin(k), exactly one w_l = 1, others equal 0 )
 7.            mstring(k) ← mstring(l')+"d_k", pstring(k) ← pstring(l')+"d_k", w_k ← 1;
               /* w_l' = 1, l' ∈ fanin(k) */
 8.          else
 9.            w_k ← 1, mstring(k) ← "m_k^i", pstring(k) ← "p_k^i";
10.            for all w_l = 1, l ∈ fanin(k)
11.              write down the two constraints,
12.                mstring(l) + d_k ≤ m_k^i,  pstring(l) + d_k ≥ p_k^i,
```

Figure 7: The symbolic constraints propagation algorithm.

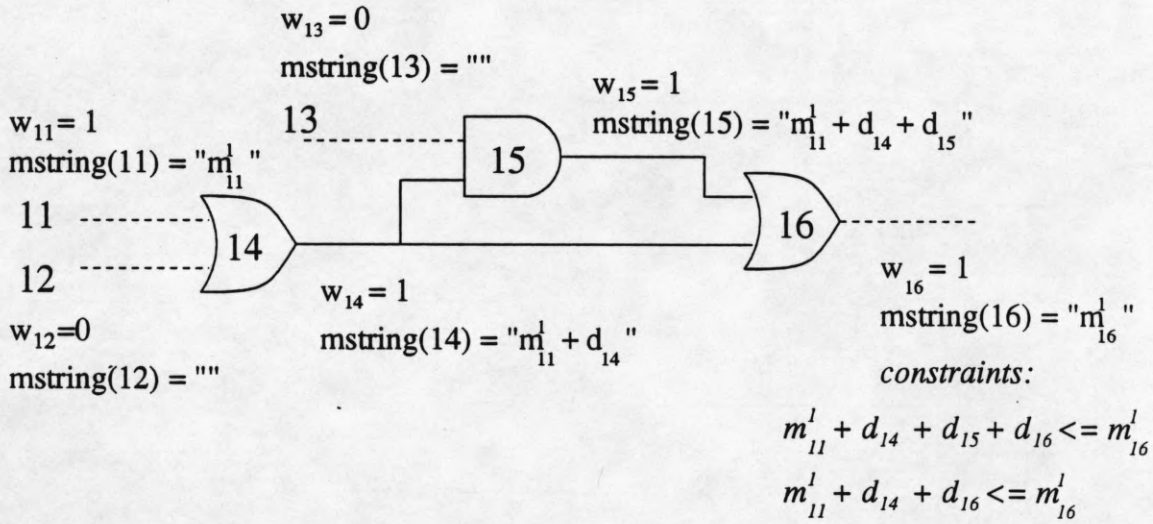

Figure 8: An example illustrating symbolic delay propagation algorithm.

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 $G_{n(i-1)}$, $2 \leq i \leq k$, $k =$ # 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: $delay(G_{ni}) < delay'(G_{ni}) + delay(bf) \leq delay(G_{ni}) + slack(G_{ni})$)
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 ($p_{n1} \geq req_s(n1)$) stop;
8. else goto 1.
9. $i \leftarrow i + 1$;

Figure 9: The buffer insertion algorithm.

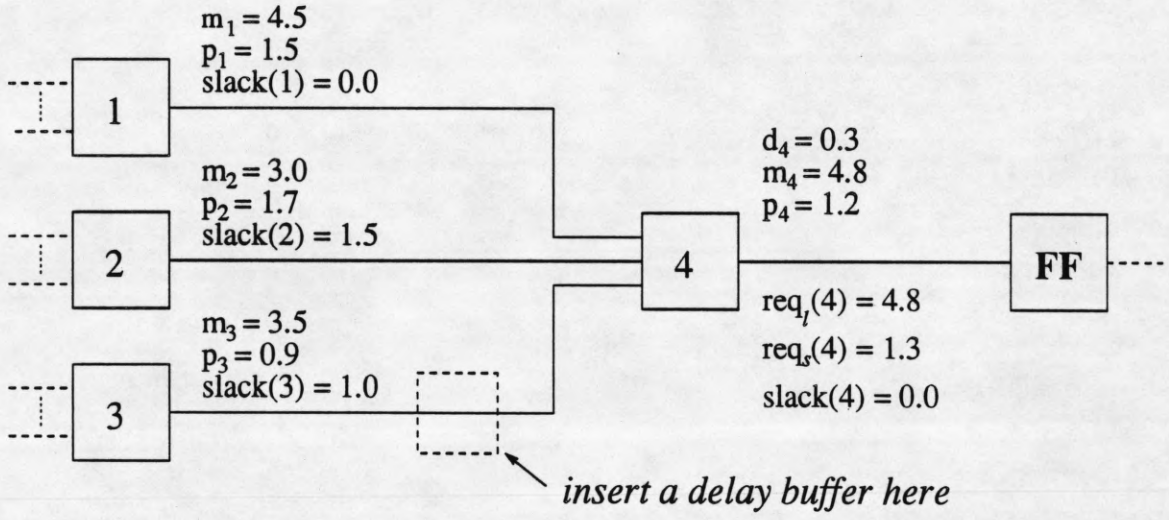

Figure 10: An example illustrating buffer insertion algorithm.