May 1994

UILU-ENG-94-2216
DAC-46

Analog and Digital Circuits

Timing and Area Optimization for VLSI Circuit and Layout

Wei-Tong Chuang

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) (DAC-46)	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 VLSI Circuit and Layout

12. PERSONAL AUTHOR(S)
Chuang, Wei-Tong

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 8/91 TO 4/94	94 May 10	149

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Time-area optimization; VLSI layout; linear programming; delay estimation; sequential circuits; physical design; placement; gate sizing

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

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

This thesis considers two problems in computer-aided design of VLSI circuits: (1) discrete gate sizing and (2) timing-driven placement improvement.

The discrete gate-sizing problem is described as follows. 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 solution 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 and by adjusting the delays between the central clock distribution node and individual flip-flops. Existing methods treat these two problems separately, which may lead to very suboptimal solutions in some cases. We develop a novel unified approach to tackle them simultaneously. We also address the problem of making this work applicable to very large synchronous sequential circuits by partitioning these circuits to reduce the computational complexity.

Traditionally, gate sizing is performed before the actual physical design steps are performed. A drawback of such an approach is that the interconnect wire lengths are not available at the gate-sizing stage. The gate sizes selected to be optimal at that stage may no longer be optimal later in the physical design process in which large interconnect capacitances are introduced at the output of each gate. To remedy this problem, we propose a novel algorithm which performs delay and area optimization for a given compact placement by resizing and relocating cells in the circuit layout. The algorithm combines gate sizing with the placement adjustment procedure into one formulation. Since the gate-sizing procedure is embedded within the placement adjustment process, interconnect capacitance information is included in the gate-size selection process.

TIMING AND AREA OPTIMIZATION FOR VLSI CIRCUIT AND LAYOUT

BY

WEI-TONG CHUANG

B.S., National Taiwan University, 1986
M.S., University of Maryland at College Park, 1989

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

TIMING AND AREA OPTIMIZATION FOR VLSI CIRCUIT AND LAYOUT

Wei-Tong Chuang, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1994
I.N. Hajj, Advisor

This thesis considers two problems in computer-aided design of VLSI circuits: (1) discrete gate sizing and (2) timing-driven placement improvement.

The discrete gate-sizing problem is described as follows. 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 solution 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 and by adjusting the delays between the central clock distribution node and individual flip-flops. Existing methods treat these two problems separately, which may lead to very suboptimal solutions in some cases. We develop a novel unified approach to tackle them simultaneously. We also address the problem of making this work applicable to very large synchronous sequential circuits by partitioning these circuits to reduce the computational complexity.

Traditionally, gate sizing is performed before the actual physical design steps are performed. A drawback of such an approach is that the interconnect wire lengths are not available at the gate-sizing stage. The gate sizes selected to be optimal at that stage

may no longer be optimal later in the physical design process in which large interconnect capacitances are introduced at the output of each gate. To remedy this problem, we propose a novel algorithm which performs delay and area optimization for a given compact placement by resizing and relocating cells in the circuit layout. The algorithm combines gate sizing with the placement adjustment procedure into one formulation. Since the gate-sizing procedure is embedded within the placement adjustment process, interconnect capacitance information is included in the gate-size selection process.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and thanks to my advisor, Professor Ibrahim Hajj, for all of his assistance, technical or otherwise, patience and support that he has provided during my studies. I would like to acknowledge him for sharing his knowledge, insight and experience with me. I would also like to express my thanks to Professor Sachin Sapatnekar of the Iowa State Univertisy for initially suggesting the gate-sizing problem and for many fruitful discussions on various matters relating to the contents of this thesis. Thanks are also due to Professors Chung Laung Liu, Farid Najm, and Resve Saleh for serving on my committee.

I am grateful to my family for all of their help, support and tutelage. I would also like to thank all of my friends (who are too numerous to enumerate here), particularly Yao-Jen Chang, Shu-Ling Cheng, Terry Lee, Ping-Chung Li, Jaidip Singh, and Chin-Chi Teng for making my stay in Champaign-Urbana a pleasant one.

Finally, I thank all of the members of the Digital and Analog Circuits Group at the Coordinated Science Laboratory of the University of Illinois.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

The influence of very large scale integrated (VLSI) circuit technology on our society during the past decade has been overwhelming, in application areas ranging from consumer products to personal computers, to business management, to defense electronics. The functional capability of the modern integrated circuit (IC) has increased in scope and complexity *exponentially* with time over the past two decades. The exponential growth pattern in IC functions over time was first described by Gordon Moore [1], and the projection he made based on this pattern is known as Moore's law.

The creation of large, complex electronic systems has grown beyond the capabilities of many engineers without the aid of computers. Successful completion of large design projects requires that computers be used in virtually all aspects of the design process. This trend toward automation will accelerate as improved circuit fabrication technologies permit higher levels of integration and as more powerful computers allow more sophisicated tools. These tools must span the spectrum of the design process, including partitioned design entry, logic synthesis, circuit design, circuit simulation and verification,

physical design, process simulation, and the design for testability and manufacturability. These tools are commonly implemented and termed as computer-aided design (CAD) or electronic design automation (EDA) programs. The evolution of integrated circuit development has become heavily dependent on the development of CAD and EDA resources for design support.

In this thesis, we examine two problems in the field of electronic design automation: (1) gate sizing for combinational circuits and sequential circuits, and (2) timing and area optimization for a compact placement. Before we go into details of our work, we give a brief description of the electronic systems design process.

1.2 The Process of Electronic System Design

A typical IC design process, shown in Figure 1.1, is composed of the four following phases [2, 3]: system design, logic design, circuit design, and physical design. They are briefly described in the following.

1.2.1 System design

System design is the process of defining the circuit functionality and the input-output behavior. A behavior representation describes how a particular design should respond to a given set of inputs. Behavior may be specified by Boolean equations, tables of input and output values, or algorithms written in high-level computer languages, or hardware description languages (HDL).

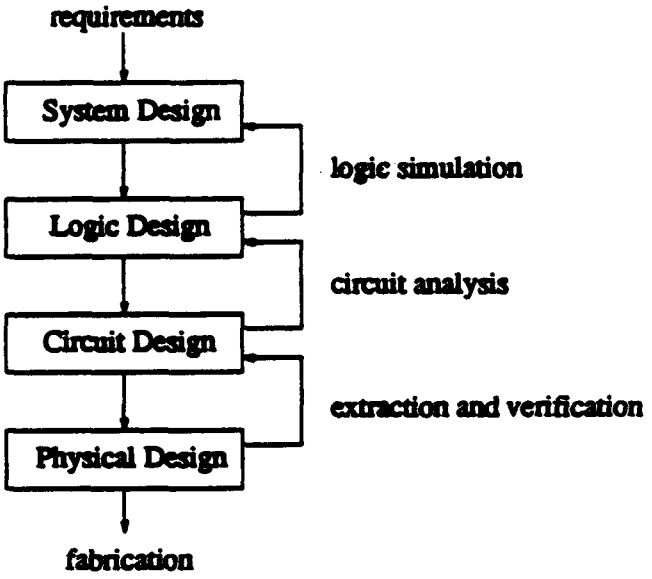

Figure 1.1 A typical IC design process.

As far as the physical aspect of the design is concerned, at this level one is concerned with connecting the major subsystems and communication interfaces with the external world; global wiring strategies; selecting layers for carrying global control, data, and power; placement of major subsystem; and routing strategies.

1.2.2 Logic design

Logic design is the process of transforming the register transfer level (RTL) specification of a design into a netlist of logic gates such as NAND gates, NOR gates, inverters, AOI gates and latches. This process begins with logic descriptions given by the RTL specification or generated by designers directly at the logic level, and optimizes the network of gates that are required to implement the function specified by the logic descriptions.

The design of random logic has objectives such as:

- minimize overall layout area of the fabricated chip;
- minimize critical path delay time;
- maximize testability of the synthesized logic.

Generally, a logic design system divides the design problem into two steps [4]:

- A technology-independent step, which manipulates general Boolean functions to optimize the logic, using algebraic and/or Boolean techniques.
- A technology-mapping step, which translates the technology-independent description derived in the first step to a set of logic gates that can be implemented in the design method of choice (e.g., standard-cells, gate-arrays, field-programmable gate arrays).

1.2.3 Circuit design

The circuit design phase concerns the electrical laws that govern the detailed behavior of the basic circuit elements such as transistors, resistors, capacitors, and inductors. It transforms the basic logic components into networks of transistors and interconnects.

Delay, power consumption, charge sharing problem, and reliability are among the major concerns in this phase.

1.2.4 Physical design

Physical design consists of transforming a circuit design description into a physical representation that can be used to manufacture the specified electronic circuit. Once the circuit description of a network is available, it can be converted into a layout. Behavioral or structural representations from the previous phases are transformed into geometric shapes that are used in the fabrication of the system. Placement and routing are the two major tasks in this phase.

Placement is the task of placing modules adjacent to each other on a chip to minimize area or delay. The placement procedure determines the locations of components within the circuit being designed, subject to the constraints imposed by the designers and the design rules imposed by the fabrication process and by physical principles.

Following placement, components are arranged on the chip, and the task remains to insert the electrical connections among the components to make them function correctly. A router takes a module placement and a list of connections and connects the components with wires.

Often, an iterative process of placement and routing is used to optimize certain objectives, such as performance or layout area, of the design.

1.3 Design Styles

There are various chip design options that may be used to implement a system design, such as sea-of-gate, gate array, standard-cell design, and full-custom design. These VLSI

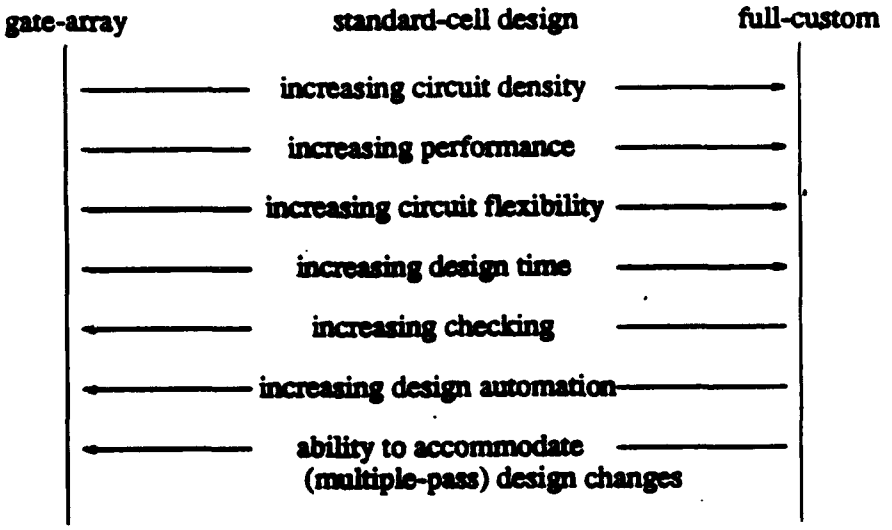

Figure 1.2 Engineering trade-offs among different design styles.

design approaches require different trade-offs and impose different constraints on the chip physical design in an attempt to make the design more manageable, while maintaining sufficient design flexibility.

The trade-offs among these different approaches are illustrated in Figure 1.2 [5]. In the following, we briefly discuss the advantages and disadvantages of different design styles, namely, gate-array, standard-cell, and full-custom designs.

To develop a full-custom design, engineering groups are assembled to cover the wide range of skills required to design the part virtually from scratch. These groups may include experts in process engineering, device modeling, circuit design, physical design layout, logic design, and system architecture design. The final design is optimized for the best density and performance. However, the design turnaround time is usually large. In addition, due to the large design effort required, a full-custom design is desired only for

high volume, for which the initial engineering expense can be compensated over a long, active product life.

In the gate-array design, several of the lithographic patterning levels are standardized, except for the interconnections and via geometries. The devices used to implement circuit designs are prefabricated on a chip, but are left unconnected after the initial processing step. A circuit design/logic block is placed at a specified cell location by assigning the appropriate pattern of wires to coordinate inside that cell area; these wires connect the devices to form the selected logic gate. Different logic blocks are then connected to implement the desired logic function. The disadvantage of gate arrays is that they are not optimal for any task. There are usually blocks that are not used. Since block placement is done in advance, interconnect routing can become complex and the resulting long wires can slow down the circuit. Also, the design will not be compact since interblock spacing is fixed to allow worst-case routing needs. Another problem with the gate-array approach is that the transistor patterns are predefined. Therefore, the transistors cannot be tuned to the specified application. This leads to inferior performance compared to full-custom design.

Between these two extremes lies the standard-cell approach which strives for high design system support for chip physical design and the capability to locally optimize circuit designs using hand-crafted cells and layouts. This approach involves the use of a library of basic functional elements, each of which has been fully characterized. In the standard-cell design approach, a division is made between the tasks of handcrafting circuit designs and placing and wiring those circuit blocks together. This separation is

based on the assumption that the time-consuming task of handcrafting a custom layout is best restricted to small circuit designs only. The initial circuit design and layout are done once, and the resulting shapes stored in a technology library for repeated use across many designs.

1.4 Standard-cell Design

The standard-cell approach has the advantage of greatly simplifying the automated synthesis process because it separates the synthesis system from the details of cell layout issues. The cell library presents models for individual cells, which are useful for performing circuit and timing analyses. With the aid of many advanced CAD tools, the performance of a standard-cell designed circuit is fairly high, while the design turnaround time is fast. Therefore, this approach has become the mainstay in Application-Specific Integrated Circuits (ASICs).

A crucial issue in the cell library approach is the size of the library. If the library is too small, much time is spent in converting the logic into a format that can be supported by the small library. On the other hand, if the library size is too large, the issues of database maintenance, pattern matching and searching become significant [6]. Moreover, the useful life of a library is relatively short as dictated by the lifetime of the technology in use [7]. For these reasons, the cell libraries tend to remain relatively small in size.

The prevalent use of complex gates such as AOI or OAI further complicates the library issue. As shown in Table 1.1 [8], as many as 3,503 different complex gates can be

Table 1.1 Number of (s,p)-gates.

(s,p)	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	7	18	42	90	186
3	3	18	87	396	1677	6877
4	4	42	396	3503	28435	222943
5	5	90	1677	28435	425803	6084393
6	6	186	6877	222943	6084393	154793519

configured for $(s,p) = (4,4)$, where the gates are constrained to have at most s transistors from output to ground and p transistors from output to power supply. This number dramatically increases to 425,803 for $(s,p) = (5,5)$ and 154,793,519 for $(s,p) = (6,6)$. It is apparent that a moderately sized library cannot support all of the possible circuit configurations for complex gates.

The other problem with the standard-cell approach is that even if the individual cells in the library are nearly optimal in performance and in terms of compactness of the layout, the whole circuit is often suboptimal after all cells are put together. For flexibility, many standard-cell libraries contain multiple versions of some cells with different driving powers.

1.5 Discrete Gate-Sizing Problem

1.5.1 Optimization for combinational circuits

In general, circuit delay and circuit area are the primary concerns of any logic design optimization. In many cases, a reduction in the number of stages (gates) between an input

and an output node can reduce the circuit area and delay. Such an optimization is usually made during the logic synthesis stage. This reduction is not, however, guaranteed to reduce the circuit delay. A standard-cell library typically contains several versions of any given gate type. Cells of identical gate type differ from each other in attribution, such as driving-capability, gate area, and input capacitive load. Because of these differences, the selection of cell versions for each individual gate in the circuit has a profound impact on the characteristics (i.e., delay, circuit area, and power consumption) of the whole circuit. By means of gate sizing, a fixed-topology logic circuit can be significantly optimized. In this thesis, we assume that a logic-level circuit description is provided, and the objective is to perform gate-size selection in an optimal way. The logic synthesis stage is usually performed before the technology mapping stage. Hence, we do not address this issue in this thesis.

Given a netlist of a logic circuit and a cell library, an automatic gate-size optimization algorithm chooses, from the library, one version of a logic gate for each cell such that

(1) the total circuit delay is under a constraint and an objective function (such as circuit area or power consumption) is minimized;

(2) the total circuit area (or power consumption) is under a constraint and the circuit delay is minimized.

The former is called the area (power) optimization problem while the latter is called the timing optimization problem. In this thesis, we concentrate on the first problem; specifically, we minimize total circuit area. It should be mentioned that the algorithm

presented in this thesis can be extend to the power optimization problem and timing optimization under area or power constraint as well.

1.6 Optimization for Sequential Circuits

Optimization for synchronous sequential circuits, on the other hand, is different from combinational circuit optimization. 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.

After developing an optimization algorithm for combinational circuits in Chapter 2, we present an optimization technique for synchronous sequential circuits in Chapter 3. We examine the following problem: Given a clock period specification, how can the area of a synchronous sequential circuit be minimized 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 and individual flip-flops?

In general, given a combinational subcircuit that lies between two FFs i and j, with clock arrival times s_i and s_j, respectively, we have the following relations:

$$s_i + Maxdelay(i,j) + T_{setup} \leq s_j + P \tag{1.1}$$

$$s_i + Mindelay(i,j) \geq s_j + T_{hold} \tag{1.2}$$

where $Maxdelay(i,j)$ and $Mindelay(i,j)$ are, respectively, the maximum and the minimum combinational delays between the two FFs, and P is the clock period. Fishburn [9] studied the clock skew problem under the assumption of constant combinational gate delays, and formulated the problem of finding the optimal clock period and the optimal skews as a linear program (LP). The objective was to minimize P, with the constraints given by the inequalities in (1.1) and (1.2) above. In real design situations, however, P is dictated by system requirements, and the real problem is to reduce the circuit area.

We first consider optimizing circuits of moderate size. Then, in Chapter 4, we consider arbitrarily large synchronous sequential circuits for which the size of the formulated optimization problems becomes prohibitively large, and present a partitioning algorithm to handle such circuits. The partitioning algorithm is used to control the computational cost of the optimization problems. After the partitioning procedure, we can apply the optimization algorithm to each partitioned subcircuit individually.

1.7 Performance-driven Placement

To ensure that high-quality designs are produced, a CAD system must take two important issues into consideration while designing a circuit:

- Layout efficiency: producing a compact circuit layout.
- Performance: satisfying the timing specifications dictated by the clocking scheme.

With the increasing drive for high-performance chips, the timing-driven layout has become more and more important.

Conventional (area-driven) placement tools try to place modules in a chip to minimize the total wire length. However, as device geometries continue to shrink, interconnect delays become increasingly significant. As a result, the reduction of interconnect wire length, which heavily influences the interconnect delay, has become increasingly important.

Recently, there has been extensive research on performance-driven placement [10–13]. Performance-driven placement techniques can be broadly divided into two categories: net-oriented and path-oriented. In the net-oriented approach, the acceptable delay of each gate (cell) is calculated and translated into bounds on the delay associated with each net. These bounds then serve as constraints during the subsequent placement step. In the path-oriented approach, timing analyses of critical paths are performed dynamically during the placement step. All paths, or a subset of them, are taken into account implicitly in the formulation.

Conventionally, gate sizing is performaed after technology mapping, and before the physical placement step. A drawback of such an approach is that accurate interconnect wire lengths are not available during the gate-sizing procedure. The gate size selected optimally at that stage may no longer be optimal after the physical design stage in which large interconnect capacitances are introduced at the output of each gate. To deal with this problem, an iteration procedure is usually followed. After global placement, the capacitance associated with each net is extracted, and the gate-sizing procedure is repeated. However, in such an iterative approach, the variation of net capacitance between iterations may be large and cause large perturbation in the solutions. Thus, a

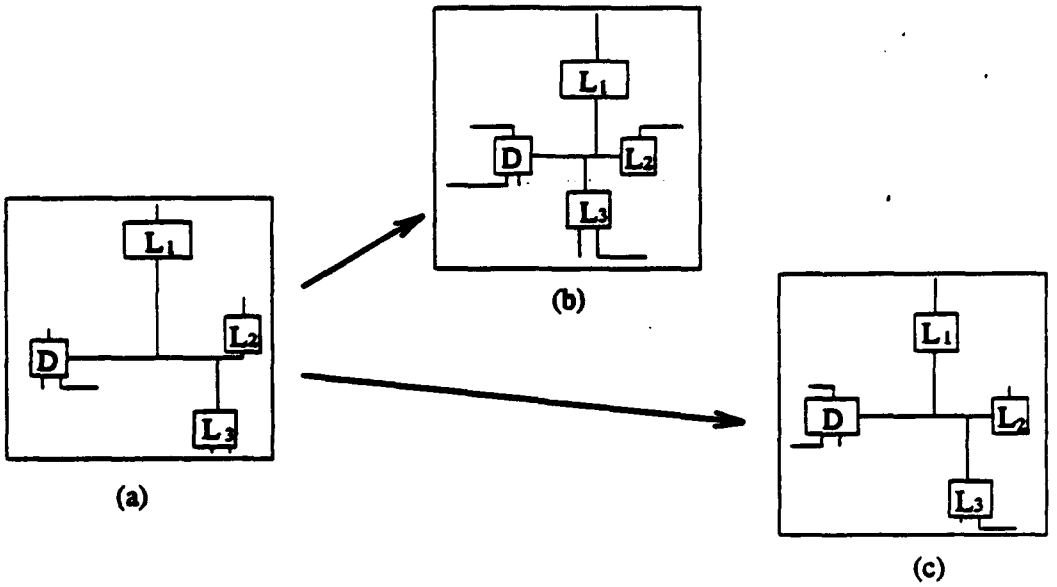

Figure 1.3 Advantage of gate sizing together with placement.

number of iterations may be required, making this approach quite expensive. To deal with this problem, it is desirable that gate sizing and placement be incorporated into a single procedure.

As an illustration, consider a layout placement shown in Figure 1.3(a). Gate D fans out to gates L_1, L_2 and L_3. Assume that the delay of this circuit under such layout violates timing constraints imposed on it. Moreover, D and L_2 lie on a long path whose delay exceeds the timing constraint. Conventional performance-driven placement would move D, L_1, L_2 and L_3 closer to each other to decrease the delay of gate D, as shown in Figure 1.3(b). This may increase the wire lengths of other nets attached to cells D, L_1, L_2 and L_3. But if automatic gate sizing is incorporated with performance-driven placement, a possible solution would be to replace D with a template with a higher driving capacity, and L_1 with one with a smaller loading capacitance with respect to D. As a result,

some of the cells could be moved to better locations, as shown in Figure 1.3(c). The overall effect is a reduction of the long path delay, while the increase in area is kept to a minimum.

In Chapter 5, we propose an algorithm which combines the gate-sizing problem and performance-driven placement into one procedure. By considering these two problems together, the value of interconnect capacitance is known during the selection stage of the automatic sizing procedure. Therefore, optimal gate sizes can be chosen for each gate based on layout information, thus reducing the number of iterations required in the conventional approach.

1.8 Organization of the Thesis

Chapter 2 of the thesis deals with discrete gate sizing for combinational circuits. In Chapter 3, we formulate the synchronous sequential circuit area optimization problem and present the algorithms to tackle the problem. The partitioning algorithm presented in Chapter 4 allows us to handle large circuits. Chapter 5 of the thesis discusses a novel approach to timing-driven placement. Finally, concluding remarks are made in Chapter 6.

CHAPTER 2

DISCRETE GATE-SIZING PROBLEM

2.1 Introduction

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. In general, the interaction between the size of a certain gate and the delay of the whole circuit is very complicated. A larger cell usually has a larger driving capability and a larger input capacitive load. Therefore, using a large template tends to speed up the gate itself while slowing down the predecessor gates that drive it.

Example 2.1 Consider the chain of three CMOS inverters shown in Figure 2.1(a) [14]. 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

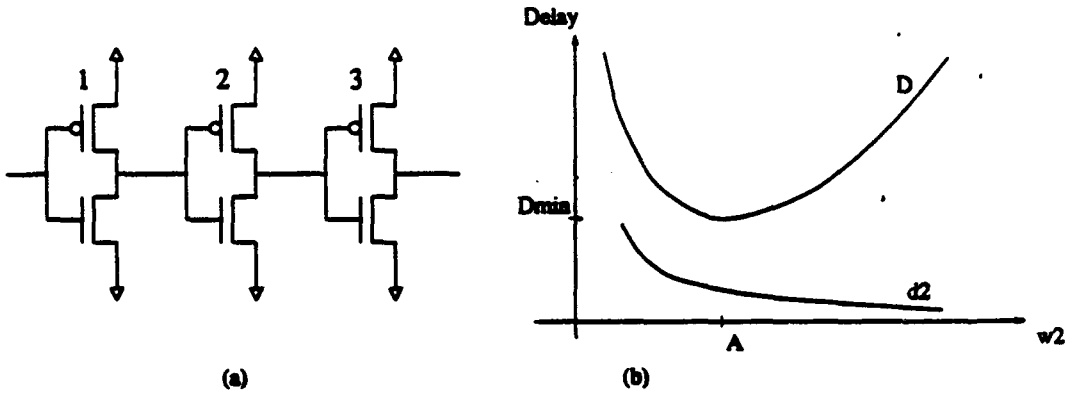

Figure 2.1 (a) A chain of three inverters. (b) Effect of transistor sizes on delay for the three-inverter chain.

the size of the transistors in gates 1 and 3 fixed. This causes the magnitude of the 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 2.1(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. □

The rationale for dealing with only combinational circuits in a world 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.

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 [14–21]. The problem is most often posed as a nonlinear optimization problem, with nonlinear programming techniques used to arrive at the solution. The solutions found by these techniques are then rounded to the nearest integer. The continuous model works well for sizing transistors in a full-custom layout, but does not work well for designing with macrocells and standard cells, for which only a small number of choices are available. Transistor sizing on a gate array, where transistor sizes have to be multiples of the standard transistor, is also poorly realized by the continuous model.

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 in which 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 [22].

In this chapter, 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. There are three phases in our algorithm. 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 [23].

It is worth mentioning that rounding solutions of the linear program to the nearest available sizes may not produce good solutions. In a tightly constrained problem, rounding continuous sizes to the nearest discrete size may not even give a feasible solution. The only reason that the continuous model works so well for transistor sizing is that the performance measures are rather insensitive to small changes in transistor sizes, and the steps between possible sizes are small compared to the sizes themselves. In our problem, however, the change between each step is large. Consequently, the solution obtained by rounding a linear program solution may violate timing constraints, or the objective value may be much larger than the optimal solution. Therefore, a more sophisicated algorithm is needed to handle the problem.

This chapter is organized as follows. We briefly describe previous approaches to the discrete gate-sizing problem in Section 2.2. Then we describe the linear programming approach that we propose in Section 2.3, followed by two postprocessing phases described in Sections 2.4 and 2.5. Experimental results are given in Section 2.6. Finally, we conclude this chapter in Section 2.7

2.2 Previous Work

Chan [22] proposed a solution to the problem that was based on a branch-and-bound strategy. The algorithm is exact for Boolean tree networks. For general networks that are not tree-structured, a backtracking-based algorithm is proposed for finding a feasible solution. The algorithm for solving the optimal discrete sizing problem on a Boolean tree network consists of two phases. In the first phase, timing requirements for each vertex in the network are generated and propagated through the network. All of the timing requirements at the fan-in of each vertex are intersected to prune infeasible timing requirements of the vertex's predecessors. In the second phase, backward substitution is used to assign optimal sizes to each vertex to minimize the total cost. 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. As pointed out in [24], 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. [23] uses a heuristic algorithm that is an adaptation of the TILOS algorithm [15] 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. At the beginning of the algorithm, all cells in the circuit are set to their minimum sizes. The algorithm consists of a series of iterations, each iteration in turn having two phases. In the first phase (*increasing phase*), a *quantum* number of cells are replaced with larger templates, such that the delay constraints can be satisfied. After the timing constraints are satisfied, in the second phase (*decreasing phase*), a *quantum* number of cells are replaced with templates with smaller cell areas to reduce total circuit area. The value of *quantum* is determined emperically and is reduced by one-half over each iteration. The iteration continues until *quantum* becomes 1 or no improvement is possible. However, 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. On the other hand, 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.

Another algorithm proposed by Li et al. [24] is exact for series-parallel circuits. A *simple parallel circuit* is a basic circuit that is comprised of several chains that have the same first and last module. A *series-parallel circuit* is a basic circuit recursively defined as [24]:

- A chain of basic modules is a series-parallel circuit.

- A simple parallel circuit is a series-parallel circuit.
- A circuit obtained from a series-parallel circuit C by replacing any interconnect of C by another series-parallel circuit is also a series-parallel circuit.

The algorithm uses a dynamic programming technique to find solutions for a chain of modules. For a simple parallel circuit, a number of transformations are repeated to obtain the optimal implementation. Finally, 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. Moreover, their algorithm does not consider the capaticances of fan-out modules (gates). Therefore the results may not be accurate since, in reality, the gate delay is a function of the fan-out gate sizes as well.

Both of the above two approaches [23,24] 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 [25] that is known to give optimal or near-optimal solutions.

The algorithm proposed in [26] does use simulated annealing; however, since simulated annealing is computationally expensive, a technique for variable pruning is used by this algorithm to reduce the computational complexity. An initial configuration is obtained using an algorithm similar to TILOS [15]. 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 runtimes for tight timing specifications, in which a large number of cells would be sized by the TILOS-like heuristic.

2.3 Problem Formulation

For a combinational circuit, the discrete gate-sizing problem is formulated as

$$\begin{aligned} &\text{minimize} \quad Area \\ &\text{subject to} \quad Delay \leq T_{spec} \end{aligned} \tag{2.1}$$

Alternatively, we can formulate the following problem:

$$\begin{aligned} &\text{minimize} \quad Delay \\ &\text{subject to} \quad Area \leq A_{spec} \end{aligned} \tag{2.2}$$

In this chapter, we concentrate on the first problem, although the same algorithm can be applied to the second problem with minor changes.

2.3.1 Formulation of delay constraints

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

$$delay = R_{out} \times C_{out} + \tau = \frac{R_u}{w} \times C_{out} + \tau_1 \cdot w + \tau_2 \tag{2.3}$$

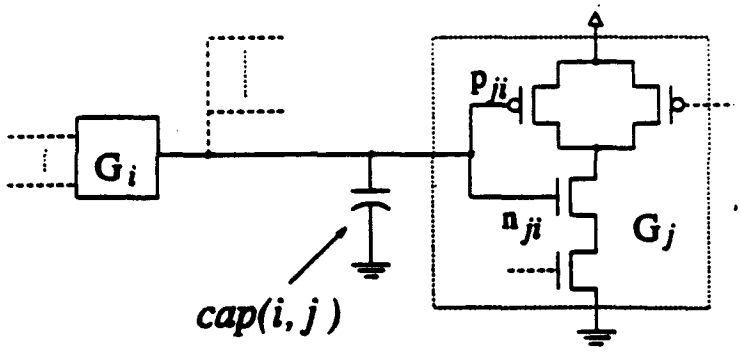

Figure 2.2 An example illustrating calculation of the output load capacitance of a gate.

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. For the time being, we ignore the interconnect capacitance. In Chapter 5, we will discuss how to combine layout information with our formulation to obtain more accurate results.

Consider a gate G_i which fans out to several gates including gate G_j, as shown in Figure 2.2. The output node of logic gate i is connected to the n-type transistor n_{ji} and p-type transistor p_{ji} of logic gate G_j. Let the transistor n_{ij} have the geometry as shown in Figure 2.3.

As illustrated in Figure 2.3, the parameter L stands for the length of the channel, xn_{ji} is the channel width of transistor n_{ji}, and d_d and d_s refer to the lengths of the drain

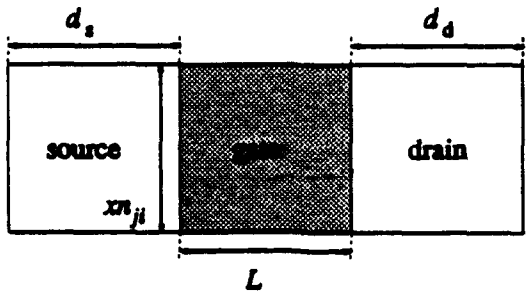

Figure 2.3 Top view of the geometry of a typical transistor.

and source terminals. The gate terminal capacitance, C_g, of the transistor n_{ji} can be expressed as

$$C_g = C_{\text{GTA}} \cdot L \cdot xn_{ji} + 2 \cdot C_{\text{GTP}} \cdot (L + xn_{ji}) \tag{2.4}$$

where

C_{GTA} : Gate terminal area capacitance (pF/μm^2)

C_{GTP} : Gate terminal perimeter capacitance (pF/μm)

Since the channel length, L, of transistors in a typical standard-cell library is fixed, the output load capacitance of logic gate i with respect to logic gate j can be expressed as

$$cap(i,j) = K_1 \cdot xn_{ji} + K_2 \tag{2.5}$$

where

$$\begin{aligned} K_1 &= C_{\text{GTA}} \cdot L + 2 \cdot C_{\text{GTP}} \\ K_2 &= 2 \cdot C_{\text{GTP}} \cdot L \end{aligned} \tag{2.6}$$

In general, the gate terminal capacitances of a certain transistor in different versions of a logic gate may not be linearly proportional to the nominal size of that logic gate.

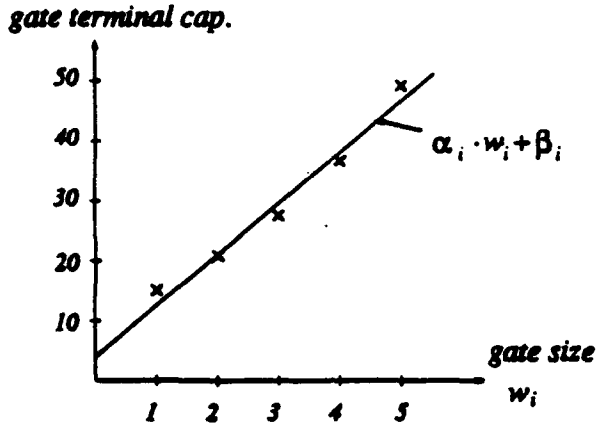

Figure 2.4 Approximating gate terminal capacitance by an affine function.

For example, Figure 2.4 shows a typical plot of the gate terminal capacitance of a certain transistor with respect to different sizes of a logic gate. Inspite of this, however, we can approximate the data points by an affine function using linear least-squares approximation, as shown in the figure. In other words, the output load capacitance of logic gate i as seen by logic gate j is

$$cap(i,j) = \alpha_{ij} \cdot z_i + \beta_{ij} \tag{2.7}$$

where z_j is the size of logic gate j.

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

$$\begin{aligned} C_{out} &= cap(i,1) + cap(i,2) + \cdots + cap(i,f) \\ &= \alpha_{i1} \cdot z_1 + \beta_{i1} + \alpha_{i2} \cdot z_2 + \beta_{i2} + \cdots + \alpha_{if} \cdot z_f + \beta_{if} \end{aligned} \tag{2.8}$$

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

$$D(w) = \frac{R_u}{w} \cdot C_{out} + \tau_1 \cdot w + \tau_2$$

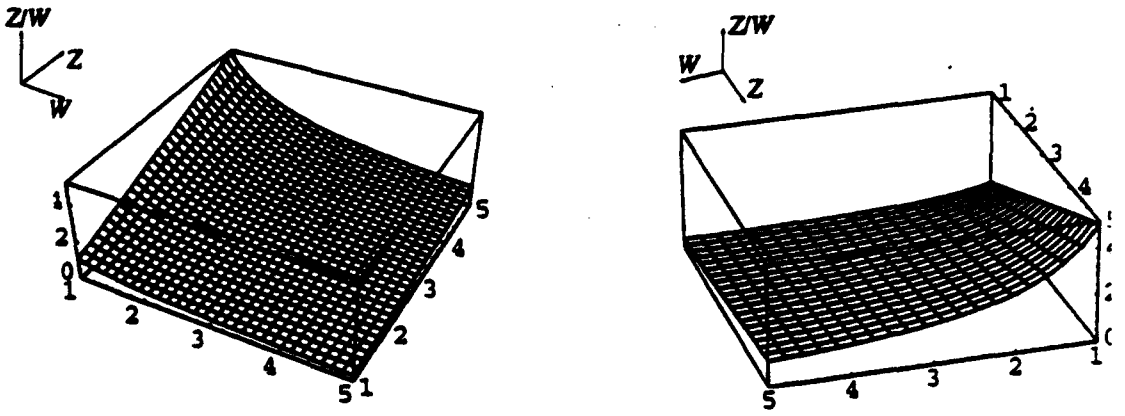

Figure 2.5 Surface plots of the function $g(w,z) = z/w$ from two different viewpoints.

$$= R_u \cdot \frac{\alpha_{i1} \cdot z_1 + \beta_{i1} \cdots + \alpha_{if} \cdot z_f + \beta_{if}}{w} + \tau_1 \cdot w + \tau_2 \tag{2.9}$$

Therefore the delay of a cell is a sum of functions of $g(w,z) = z/w$ and $h(w) = 1/w$. Figure 2.5 shows surface plots of the function z/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

$$PWL(w,z) = \begin{cases} a_1 \cdot w + b_1 \cdot z + c_1 & (w,z) \in \text{Region } \mathbf{R}_1 \\ a_2 \cdot w + b_2 \cdot z + c_2 & (w,z) \in \text{Region } \mathbf{R}_2 \\ \vdots & \\ a_q \cdot w + b_q \cdot z + c_q & (w,z) \in \text{Region } \mathbf{R}_q \end{cases} \tag{2.10}$$

$$= \max_{1 \le i \le q} (a_i \cdot w + b_i \cdot z + c_i) \qquad \forall (x,y) \in \bigcup_{1 \le i \le q} \mathbf{R}_i \tag{2.11}$$

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

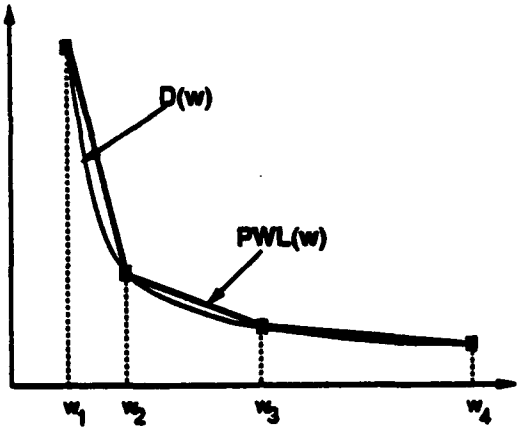

Figure 2.6 Approximation of the function $1/w$ by a piecewise linear function.

The function $1/w$ is shown in Figure 2.6. Similarly, we can approximate the function $h(w) = 1/w$ with a convex piecewise linear function of the form

$$pwl(x) = \begin{cases} d_1 \cdot w + e_1 & w \in \text{Region } \mathrm{r}_1 \\ d_2 \cdot w + e_2 & w \in \text{Region } \mathrm{r}_2 \\ \vdots & \\ d_q \cdot w + e_q & w \in \text{Region } \mathrm{r}_q \end{cases} \tag{2.12}$$

$$= \max_{1 \le j \le q} (d_j \cdot w + e_j) \qquad \forall\, w \in \bigcup_i \mathrm{r}_i \tag{2.13}$$

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:

$$
\begin{aligned}
&\hat{D}(w, z_1, \cdots, z_f) \\
&= \begin{cases}
\hat{a}_1 \cdot w + \hat{b}_{1,1} \cdot z_1 + \cdots + \hat{b}_{1,f} z_f + \hat{c}_1 + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_1 \\
\hat{a}_2 \cdot w + \hat{b}_{2,1} \cdot z_1 + \cdots + \hat{b}_{2,f} z_f + \hat{c}_2 + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_2 \\
\vdots & \\
\hat{a}_q \cdot w + \hat{b}_{q,1} \cdot z_1 + \cdots + \hat{b}_{q,f} z_f + \hat{c}_q + \tau_1 \cdot w + \tau_2 & (w, z_1 \cdots z_f) \in \text{Region } \mathbf{R}_q
\end{cases} \\
&= \max_{1 \le i \le q} (\hat{a}_i \cdot w + \hat{b}_{i,1} \cdot z_1 + \cdots \hat{b}_{i,f} z_f + \hat{c}_i) + \tau_1 \cdot w + \tau_2 \qquad \forall (w, z_1 \cdots z_f) \in \bigcup_{1 \le i \le q} \mathbf{R}_i
\end{aligned} \tag{2.14}
$$

2.3.2 Formulation of the linear program

The formal definition of the gate-sizing problem for a combinational circuit is as given in (2.1). 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 [14–21].

Similarly, in general, the cell area of a logic gate may not be linearly proportional to the cell size, as shown in Figure 2.7. Nevertheless, we can approximate those data points by an affine function using linear least-squares approximation as shown in the figure.

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

$$area(i) = \gamma_i \cdot w_i + \varepsilon_i \tag{2.15}$$

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

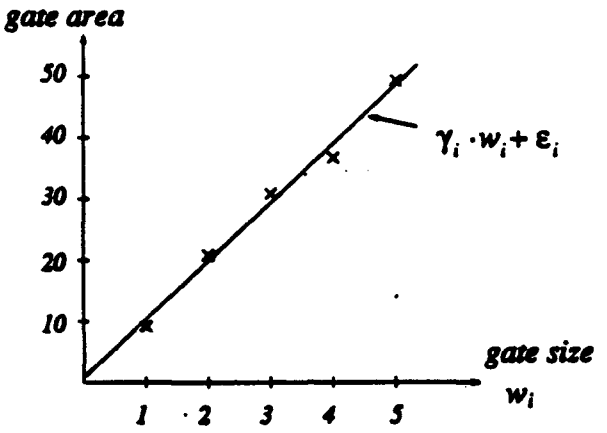

Figure 2.7 Approximating gate area by an affine function.

potentially be exponential in the number of gates; unless certain additional variables, m_i, $i = 1 \cdots M$ (where M is the number of gates), are introduced to reduce the number of constraints. 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

$$m_i = \max\{m_j + d_i \mid \forall\, j \in Fanin(i)\} \tag{2.16}$$

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

$$m_j + d_i \leq m_i, \quad \forall\, j \in Fanin(i). \tag{2.17}$$

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

For example, consider a part of a circuit as shown in Figure 2.8. Gates 1, 2, and 3 fan out to gate 4. The worst-case signal arrival times of gate 1, 2, and 3 are 4.5, 3.0, and

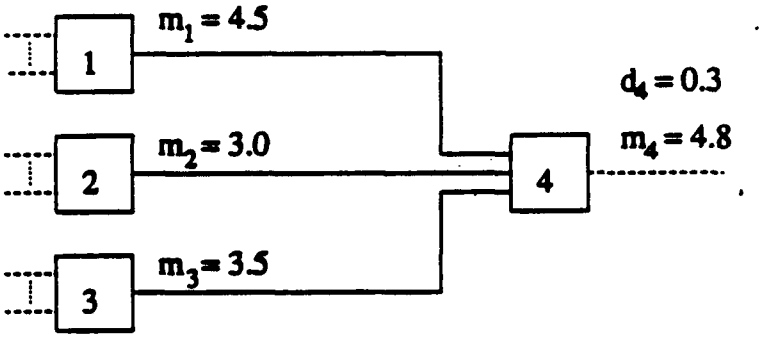

Figure 2.8 An example illustrating the definition of m_i.

3.5, respectively. The gate delay of logic gate 4 is 0.3. Then the worst-case signal arrival time of gate 4 is $m_4 = \max(4.5,\ 3.0,\ 3.5) + 0.3 = 4.8$.

We now formulate the linear program as

$$
\begin{aligned}
&\textbf{minimize} && \sum_{i=1}^{M} \gamma_i \cdot w_i && \\
&\textbf{subject to} && \text{For all gates } i = 1 \cdots M && \\
& && m_j + d_i \leq m_i && \forall\, j \in Fanin(i) \\
& && m_i \leq T_{spec} && \forall\ gate\ i\ at\ PO's \\
& && d_i \geq \hat{D}(w_i, w_{i,1}, \ldots w_{i,fo(i)}) && \\
& && w_i \geq Minsize(i) && \\
& && w_i \leq Maxsize(i) &&
\end{aligned}
\tag{2.18}
$$

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 (2.15) is omitted since it does not affect the result.

The preceding 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 (2.14) are replaced here by inequalities so as to satisfy (2.15).

It should be emphasized that our approach is able to handle different timing specifications at different primary outputs. However, for the sake of simplicity, we use the same timing specification for all of the primary outputs in the circuit.

2.4 Phase II : The Mapping Algorithm

The set of permissible sizes for gate i is $S_i = \{w_{i,1} \cdots w_{i,p_i}\}$, where p_i is the cardinality of S_i. The solution of the linear program would, in general, provide a gate size, w_i, that does not belong to 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-}). Note that in any standard cell library, w_{i+} has a smaller delay than 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:

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

Although the complexity has been reduced from $O(\prod_{i=1}^{M} p_i)$ for the original problem to $O(2^M)$, 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 for a solution arrived at 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 2.6.

In Section 2.4.1, we present an implicit enumeration mapping algorithm which is single-path oriented. Although the execution time is fast, the result may not be satisfying. Therefore, in Section 2.4.2, we propose an improved implicit enumeration mapping algorithm using a global approach.

2.4.1 Implicit enumeration approach

The algorithm first places all M gates in a queue, Q, in decreasing order of their worst-case signal arrival time, m_i. The longest path, P, from any 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 [27]. 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 has 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,

$area(i,w_i)$: the cell area of gate i when its size is w_i, $area(i,w_i) = \gamma_i \cdot w_i$ (see (2.18)),

$R^i_{out}(w_i)$: the equivalent resistance of gate i, corresponding to size w_i, that drives its load capacitances, $R^i_{out}(w_i) = R_u/w_i$ (see (2.2)),

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

Definition 2.1 *A <u>cell configuration</u>, $C(i,j)$ is a triple (W_{ij}, A_{ij}, D_{ij}),*

$$W_{ij} = W_{C(i,j)} \in \{w_{i+}, w_{i-}\},$$

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

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

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

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

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, w_{i+} and w_{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 (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) $area(i, W_{ik}) \leq area(i, W_{ij})$, $A_{ik} \leq A_{ij}$ and $D_{ik} < D_{ij}$, or

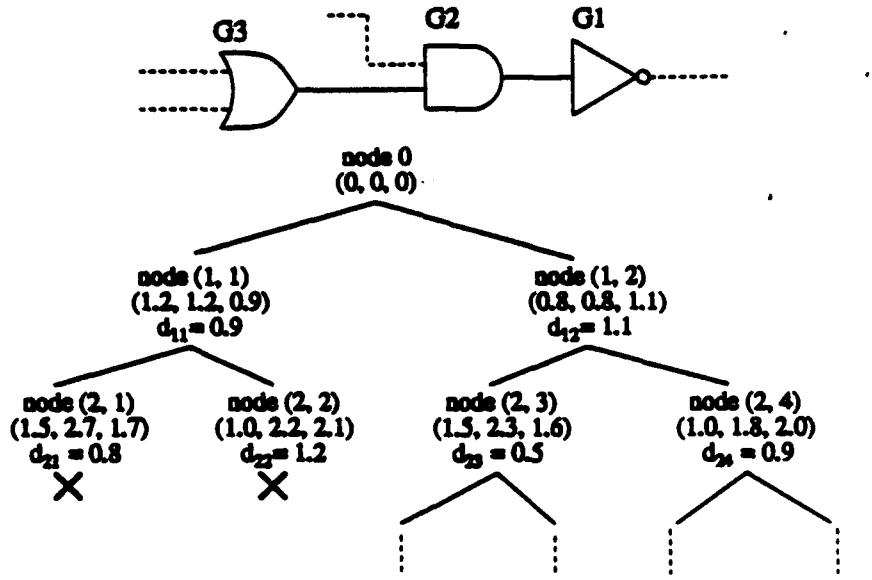

Figure 2.9 An example illistrating the construction of a state-space tree in the mapping algorithm.

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

Example 2.2 In Figure 2.9, 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 $area(1, W_{1,1}) = 1.2$ and delay $d_{1,1} = 0.9$, and

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

If neither node $C(1,1)$ nor $C(1,2)$ 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 be of the size with $area(1, W_{1,1}) = 1.2$.

Further, node $C(2,1)$ represents the configuration if G_2 is chosen to have a template with $area(2, W_{2,1}) = 1.5$. 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 be of the 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 the *maximum path delay*, (MPD), as follows:

$$MPD(P) = \begin{cases} \min\limits_{j \in FO(i)} (m_j - d_j), & \text{if gate } i \text{ is not at a PO} \\ \min[\min\limits_{j \in FO(i)} (m_j - d_j),\ T_{spec}], & \text{if gate } i \text{ is at a PO} \end{cases} \tag{2.19}$$

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 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) $D_{|P|,n} \geq D_{|P|,i} \quad \forall\, C(|P|, i)$ 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.

2.4.2 Global 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 2.2 *A critical gate is a gate whose size is larger than its smallest possible size.*

Notice that the determination of critical gates, in general, can be very difficult to obtain, since whether a gate is critical or not heavily depends on the circuit structure as well as the tightness of the delay bounds. However, using an analytical approach such as linear programming, whether a gate is critical or not can be determined easily.

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* [28]. That is,

$$mds(i) = \max_{j \in FO(i)} \{mds(j) + d_j\} \tag{2.20}$$

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.[1] 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, where $k = \mathcal{F}(i)$. 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.

Definition 2.3 *A cell configuration, $C(i,j)$ is a triple (W_{ij}, A_{ij}, D_{ij}),*

$W_{ij} = W_{C(i,j)} \in \{w_{\mathcal{F}(i)+}, w_{\mathcal{F}(i)-}\}$,

$A_{ij} = A_{C(i,j)} = \gamma_{\mathcal{F}(i)} \cdot W_{ij} + A_{anc(i,j)}$,

$D_{ij} = D_{C(i,j)} =$ *max* $\{mds(k)\}$, *where k is a gate in the circuit (not necessarily a critical gate), which satisfies* $c\_level(k) = c\_level(\mathcal{F}(i)) + 2$.

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

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 $w_{\mathcal{F}(i)-}$.[2] 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 and denoted by D'_{ij}. The cell configuration

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

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.

2.5 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

$$Pslack(P) = T_{spec} - m_p \tag{2.21}$$

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

$$delay(G_i) = R^{i+1}_{out} \cdot C^{i+1}_{out} + R^{i}_{out} \cdot C^{i}_{out} \tag{2.22}$$

$$\Delta delay(G_i) = R^{i+1}_{out} \cdot \Delta C^{i+1}_{out} + \Delta R^{i}_{out} \cdot C^{i}_{out} \tag{2.23}$$

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, we select the largest one, $\Delta delay(G_n)$, which satisfies

$$\Delta delay(G_n) < Pslack(P) \tag{2.24}$$

and change the size of G_n accordingly. If none of the local delay differences satisfy (2.24), 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.

2.6 Experimental Results

The preceding algorithms were implemented in a program GALANT (GAte sizing using Linear programming ANd heuricTics) on a Sun Sparc10 station. The test circuits include several ISCAS85 combinational benchmark circuits [29]. Each cell in the standard-cell library has four different sizes of realization with different driving capabilities.

To prove the efficacy of the approach, a simulated annealing algorithm and Lin's algorithm [23] 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 [26]. However, unlike in [26], all gate sizes were allowed to change during the simulated annealing procedure; while the runtimes 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 2.1. The test circuits include five ISCAS85 benchmarks, and vary in size from 160 gates (824 transistors) to 3512 gates (15,396 transistors). It can be seen that the accuracy of the results of our approach ranges from being as good as simulated annealing for c432 to a discrepancy of less than 2.0% in comparison with simulated annealing. The average discrepancy is less than 1.0%, and the run times are considerably smaller than those for simulated annealing.

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 a solution at all. As mentioned previously, Lin's algorithm essentially is an adaptation of the TILOS algorithm [15] 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 2.2 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 2.1. It is clear that for all circuits, the chief component (over 95%) of the runtime

Table 2.1 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			

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 2.2, and the total runtime in Table 2.1 is attributable to the preprocessing step which performs miscellaneous administrative steps such as reading in the circuit description and levelizing the circuit.

Table 2.2 Execution times for the Linear Program and the Mapping and Adjusting Algorithms.

Circuit	# of gates	T_{spec}	LP solution	Mapping and Adjusting
c432	160	12.0	6.98s	0.75s
c1355	546	14.0	1min 4s	7.33s
c2670	1193	14.0	6min 50s	13.68s
c5315	2307	17.0	18min 29s	32.51s
c7552	3512	16.0	1h 10min	1min 21s

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 2.3 and is plotted in Figure 2.10. 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 tighter, 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 runtime 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, in which most

Table 2.3 Performance comparison of GALANT with Lin's algorithm and simulated annealing for c432.

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	17.5	2331	24min 29s	2331	4.63s	1.000	2331	0.09s	1.000
	17.0	2337	24min 28s	2337	4.66s	1.000	2337	0.08s	1.000
	16.5	2350	25min 11s	2350	4.72s	1.000	2368	0.09s	1.013
	16.0	2372	25min 40s	2376	4.81s	1.002	2376	0.17s	1.002
	15.5	2394	25min 45s	2402	5.02s	1.003	2450	0.16s	1.023
	15.0	2420	26min 28s	2424	4.96s	1.002	2465	0.13s	1.019
	14.5	2467	26min 17s	2467	5.02s	1.000	2719	0.16s	1.102
	14.0	2515	26min 32s	2515	5.39s	1.000	2749	0.23s	1.092
	13.5	2563	27min 47s	2563	5.68s	1.000	2929	0.17s	1.143
	13.0	2645	27min 57s	2658	6.05s	1.005	3024	0.15s	1.143
	12.5	2801	28min 33s	2801	7.24s	1.000	3332	0.26s	1.190
	12.0	2950	24min 27s	2983	7.72s	1.011	-	-	-
	11.5	3096	36min 4s	3139	8.40s	1.014	-	-	-
	11.0	3300	38min 28s	3315	13.35s	1.005	-	-	-
	10.5	3546	43min 5s	3583	10.95s	1.010	-	-	-
Average Area Ratio						1.009			1.206

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.

2.7 Conclusions

In this chapter, an efficient algorithm is presented to minimize the area taken by cells in standard-cell designed combinational circuits under timing constraints. We present

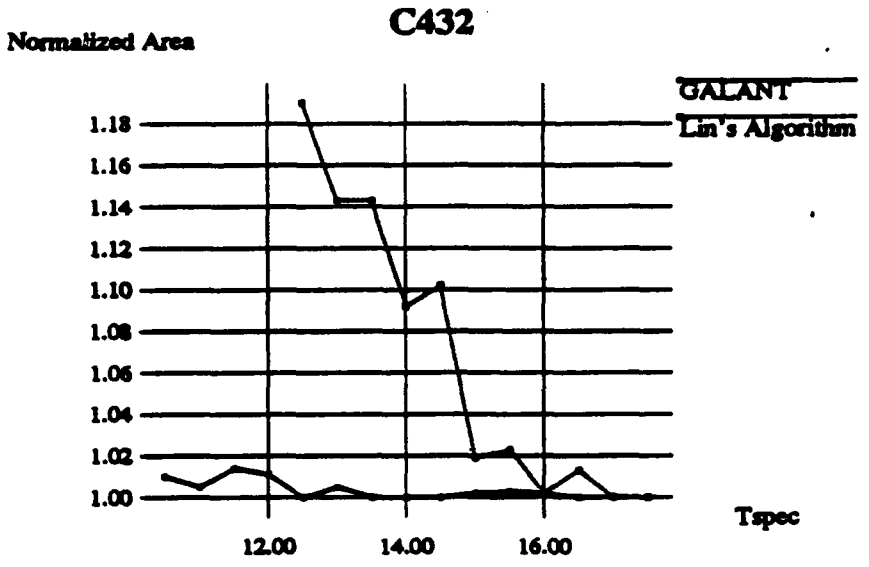

Figure 2.10 Comparison of Galant and Lin's algorithm against simulated annealing for c432.

a comparison of the results of our algorithm with the solutions obtained by our implementation of Lin's algorithm [23] and by simulated annealing. In [23], it was shown that Lin's algorithm is able to obtain better results than the technology mapping of MIS2 [8]. 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 a 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.

The major bottleneck of our approach is the time required to solve the linear program. Our approach uses a linear program which is solved using a package available in the

public domain [31], 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 [31], the CPU usage can be reduced by about 40% on an Alliant FX/8 machine. Although the computational complexity of the 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 [32]. Other polynomial-time linear programming algorithms such as Karmarkar's algorithm [33] may also be employed; however, in practice, its average run-time has been found to be similar to that of the simplex algorithm.

Finally, to increase the accuracy of the results, instead of using the RC delay model, one can use fast timing simulation to evaluate delay of the circuit during implicit enumeration. The run time will be greater. However, as we have mentioned, the number of critical gates is likely to be relatively small. Therefore, the size of state-space tree is usually small. This means that the number of times that we have to perform fast timing simulation would also be small.

CHAPTER 3

OPTIMIZATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

3.1 Introduction

The delay-area optimization problem for a combinational circuit is examined in Chapter 2. Optimization for 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. This effect is even more important in the standard-cell environment, where the granularity of available choices for gate sizes is coarse, and the delay of an optimally sized combinational subcircuit may differ significantly from its delay specification. 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.

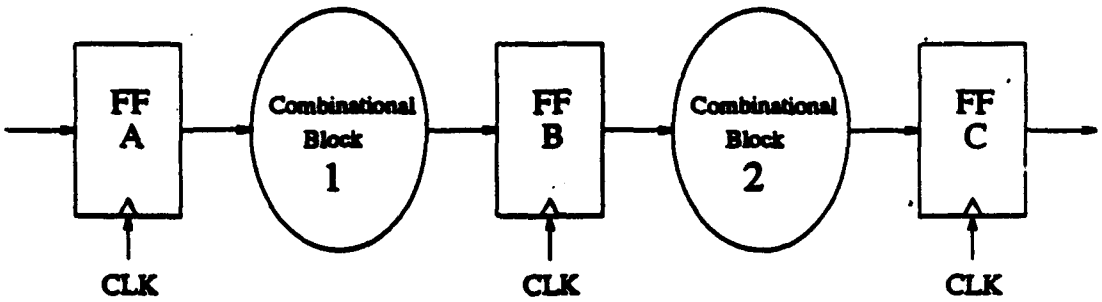

Figure 3.1 The advantages of nonzero clock skew.

Example 3.1 Consider the circuit shown in Figure 3.1. 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 3.2 Consider the synchronous sequential circuit shown in Figure 3.2. In addition to the possibility of adjusting clock skews at boundary latches as in Example 3.1, we can adjust clock skews at internal latches as well. 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:

$$s_i + Maxdelay(i,j) + T_{setup} \leq s_j + P \tag{3.1}$$

$$s_i + Mindelay(i,j) \geq s_j + T_{hold} \tag{3.2}$$

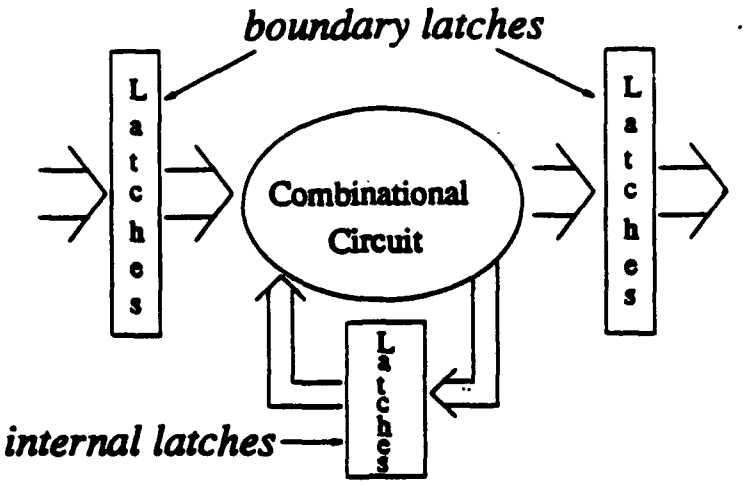

Figure 3.2 An example illustrating the definition of a synchronous block.

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 [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 P, with the constraints given by the inequalities in (3.1) and (3.2) above. In real design situations, however, P is dictated by system requirements, and the real problem is to reduce the circuit area.

In this chapter, 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 (PIs) and primary outputs (POs) are connected to FFs 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 (3.1) and (3.2) simultaneously. Hence in the third stage, we fine-tune the longest path to satisfy (3.1) and satisfy the short path constraints in (3.2) by appropriately inserting delay buffers in the short path.

In Chapter 4, we consider arbitrarily large synchronous sequential circuits for which the sizes of the formulated LPs 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 chapter is organized as follows. We briefly discuss previous work on clock skew optimization in Section 3.2. In Section 3.3, we formulate the synchronous sequential circuit area optimization problem. To reduce the number of constraints in our formulation, we propose a pruning algorithm in Section 3.4. A buffer insertion algorithm is presented

in Section 3.5, which is used to satisfy short-path constraints without violating long-path constraints. Experimental results are given in Section 3.6. Finally, Section 3.7 concludes this chapter.

3.2 Previous Work on Clock Skew Optimization

Synchronous circuit designers usually try to eliminate clock skew. Clock skew is referred to as the variations in the delays from the central clock source to individual flip-flops of the system. This effort can involve equalization of wire length [34] or wire width [35], symmetric design of the distribution network, and design guidelines to eliminate skew due to process variations [36]. Clock skew can limit the clock speed of a synchronous system or cause clocking hazards leading to malfunction at any clock rate.

In a synchronous sequential circuit, a data race due to clock skew can cause the system to fail [37]. Consider a synchronous sequential digital system with flip-flops (FFs) as shown in Figure 3.3. 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 that 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. As illustrated in Figure 3.4, 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 the positive clock edge arrives at FF_i, the data race ahead through the path and destroy the data at the input to FF_j before the clock arrives there. When the clock edge finally arrives at FF_j, the

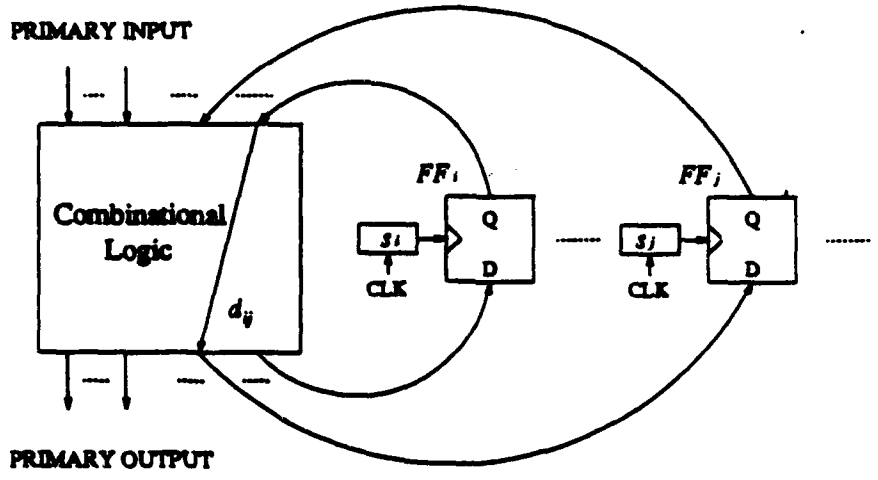

Figure 3.3 A synchronous sequential system.

wrong data are clocked through. Since the data are through two FF's with one clock edge, this has been called *double-clocking.*

Zero Clocking : This occurs when $s_i + d_{ij} > s_j + P$, i.e., the data reach FF_j too late. When the clock edge arrives at FF_j, the correct data are not ready yet. Since no correct data are clocked in by a FF, this is called *zero-clocking.*

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 the central clock signal delayed by s_i by the delay element imposed between it and the 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 $(t-T_{setup}, t+T_{hold})$,

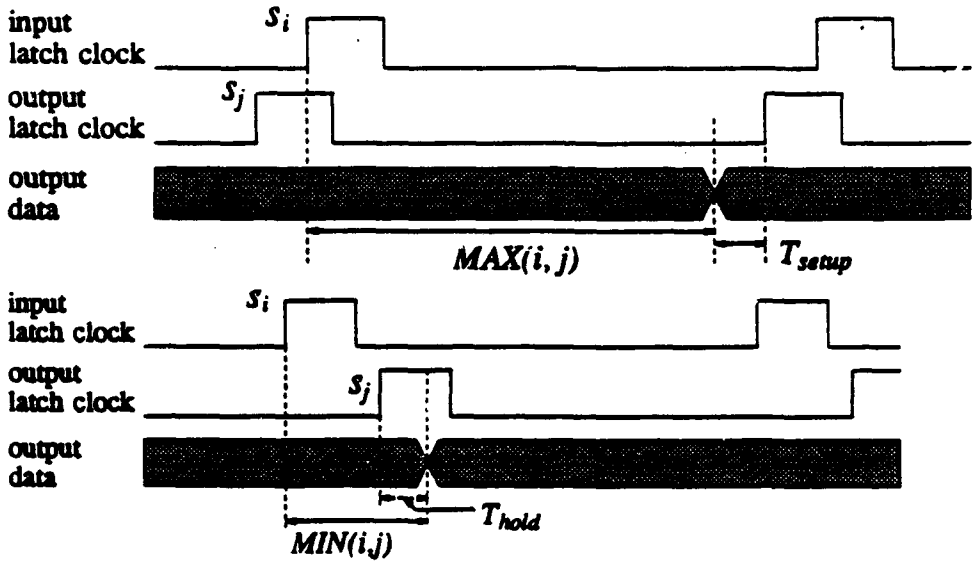

Figure 3.4 Double-clocking and zero-clocking.

where T_{setup} and T_{hold} are the setup time and hold time of the FF, respectively. For all of the FFs, the lower and upper bounds $MIN(i,j)$ and $MAX(i,j)$ (where $1 \leq i,j \leq \mathcal{L}$, $\mathcal{L}$ is 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. Since it is possible that multiple paths exist from FF_i to FF_j, $MIN(i,j)$ and $MAX(i,j)$ must be computed as the minimum and maximum of these path delays; if no such path exists, define $MIN(i,j) = \infty$ and $MAX(i,j) = -\infty$.

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

$$s_i + MIN(i,j) \geq s_j + T_{hold}. \tag{3.3}$$

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

$$s_i + T_{setup} + MAX(i,j) \leq s_j + P. \tag{3.4}$$

Inequalities (3.3) and (3.4) dictate the correct operation of a synchronous sequential system.

Two different optimization problems are then formulated [9] with regard to clock skew optimization. They are discussed briefly in the following.

3.2.1 Minimize P subject to clocking constraints

Assume that the value of T_{setup}, T_{hold}, and the maximum and minumum delays between each pair of flip-flops ($MAX(i,j)$, $MIN(i,j)$) are constant, while the clock period P and clock skews to individual flop-flops, s_i, are variable. To make the period P as short as possible while satisfying the system of inequalities Eq. (3.3) and (3.4), a linear program can be formulated as follows:

$$\begin{aligned} \textbf{minimize} \quad & P \\ \textbf{subject to} \quad & s_i - s_j \geq T_{hold} - MIN(i,j), \qquad \forall\, i,j = 1,\cdots,\mathcal{L} \\ & s_j - s_i + P \geq T_{setup} + MAX(i,j), \quad \forall\, i,j = 1,\cdots,\mathcal{L} \end{aligned} \tag{3.5}$$

3.2.2 Maximize minimum margin for error

While manufacturing a circuit, it is inevitable that process variations will cause design parameters, such as component values, to waver from their nominal values. As a result, the manufactured circuit may no longer meet some design specifications, such as the requirements on the delay. On the other hand, a system on the verge of clock hazards might pass system diagnosis but malfunction at unpredictable times due to fluctuations in ambient temperature or power supply voltage. One way to increase reliability of the system and prevent these problems from happening, is to provide a safety margin over all the constraints of the slack, i.e., the amount by which the inequality is satisfied. This converts the problem into a *maximin* problem. This is modeled by introducing a new variable M, which is added to each of the main constraint inequalities so that when M is maximized by the program, it will be the minimum slack over all the inequalities. In this problem formulation, M and s_i are variables to be determined, while P is specified as a constant. The problem can be formulated as:

$$\begin{aligned} \textbf{maximize} \quad & M \\ \textbf{subject to} \quad & s_i - s_j - M \geq T_{hold} - MIN(i,j), \qquad \forall\, i,j = 1,\cdots,\mathcal{L} \\ & s_j - s_i - M \geq T_{setup} + MAX(i,j) - P, \quad \forall\, i,j = 1,\cdots,\mathcal{L} \end{aligned} \tag{3.6}$$

3.3 Formulation of Constraints

In Fishburn's approach [9], it is assumed that circuit delays are fixed. In our problem, since gate sizes are to be determined, individual gate delays, and therefore the total circuit delay, are variables, while the clock period is a user-specified constant. Therefore, the problem becomes much more complicated since the delays $MIN(i,j)$ and $MAX(i,j)$ between each pair of latches are now also variables.

Our problem requires us to represent path delay constraints between *every* pair of FFs. This may be achieved by performing PERT [38] on the circuit and setting all FFs except the FF of interest (e.g., FF_i) to $-\infty$ (∞) for the longest (shortest) delay path from FF_i to all FFs, and the arrival time at the FF of interest is set to 0 [9]. Therefore in addition to the longest-path delay variable m_k, for the shortest-path delay, we introduce new variables, p_k, $k = 1 \cdots \mathcal{N}$, which correspond to the shortest delay from the PIs (the outputs of FFs are considered as pseudo-PIs) up to the output of G_k.

$$p_j + d_k \geq p_k, \quad \forall\, j \in Fanin(k). \tag{3.7}$$

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 reduce the number of additional variables and constraints *without* making approximations has been developed. Deferring a discussion on these procedures to Section 3.4, we now formulate the linear program for a general synchronous sequential circuit as

$$
\begin{array}{lll}
\textbf{minimize} & \displaystyle\sum_{k=1}^{\mathcal{N}} \gamma_k \cdot w_k & \\
\textbf{subjectto} & d_k \geq D(w_k, w_{k,1}, \ldots w_{k,fo(k)}), & 1 \leq k \leq \mathcal{N} \\
& w_k \geq Minsize(k), & 1 \leq k \leq \mathcal{N} \\
& w_k \leq Maxsize(k), & 1 \leq k \leq \mathcal{N} \\
& \text{For all FF } i, & 1 \leq i \leq \mathcal{L} \\
& \quad s_i + p_k^i \geq s_j + T_{hold} & 1 \leq j \leq \mathcal{L}, \;\; k = Fanin(FFj) \\
& \quad s_i + T_{setup} + m_k^i \leq s_j + P_{spec} & 1 \leq j \leq \mathcal{L}, \;\; k = Fanin(FFj) \\
& \quad \text{For all gates } \; k = 1, \cdots, \mathcal{N} & \\
& \quad\quad m_l^i + d_k \leq m_k^i, & \forall\, l \in Fanin(k) \\
& \quad\quad p_l^i + d_k \geq p_k^i, & \forall\, l \in Fanin(k)
\end{array}
\tag{3.8}
$$

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.

3.4 Symbolic Propagation of Constraints

We begin by counting the number of LP constraints in (3.8). 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. (2.15)). 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, 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 fan-ins 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 fan-out cone. Also, we collapse constraints on chains of gates wherever possible (line 6 in Figure 3.5).

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 $w_i \in \{0, 1\}$ is introduced to indicate its status; that is, the variable w_i has the value 1 whenev[illegible]$tring(i)$ and $pstring(i)$ are nonempty, i.e.,

```
ALGORITHM Symbolic_propagation()
 1. for i = 1 to L {
 2.     w_j ← 0, mstring(j) ← "", pstring(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=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 3.5 The symbolic constraints propagation algorithm.

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 3.5. 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 have to be propagated, and no operations are needed. Next, at line 6, we check whether exactly one of gate k's input gates, e.g., gate l', has a nonempty *mstring*; others 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 nonempty *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 nonempty ($w_l = 1$), we need a delay constraint (line 12).

Example 3.3 Figure 3.6 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. □

By 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.

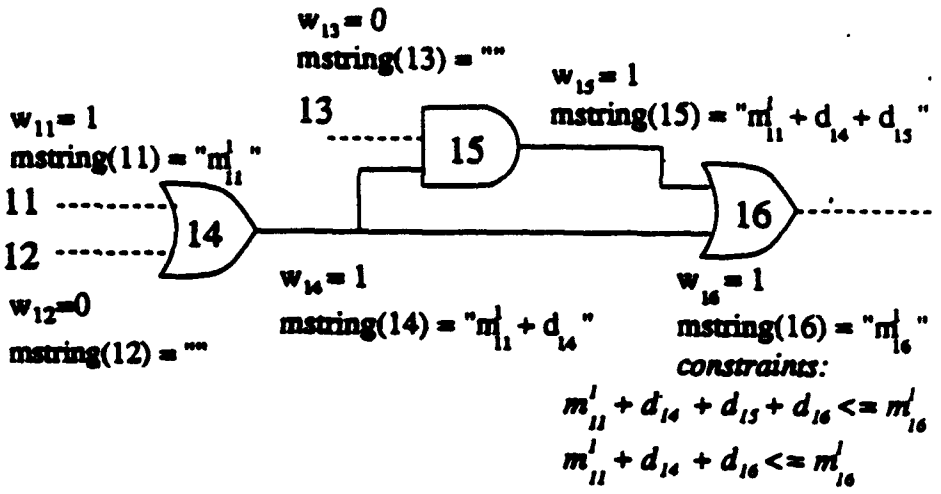

Figure 3.6 An example illustrating symbolic delay propagation algorithm.

3.5 Satisfying Short-Path Delay Constraints

The solution of the LP would, in general, provide a gate size, x_k, that does not belong to the permissible set, $\mathcal{S}_k = \{w_{k,1} \cdots w_{k,q_k}\}$. 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 2.4, we formulate the following smaller problem:

For all $k = 1 \cdots \mathcal{N}$: Select $w_k = w_{k+}$ or w_{k-}, such that

for all FFs $1 \leq i, j \leq \mathcal{L}$

$$s_i + Maxdelay(i,j) + T_{setup} \leq s_j + P_{spec}$$

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

The mapping algorithm described in Section 2.4 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 2.5, we have discussed the approach to resolving the violation of long-path delay constraints. The same strategy can be applied to synchronous sequential circuit optimization, except that 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

$$\begin{aligned} req_l(j) &= s_j + P_{spec} - T_{setup} \\ req_s(j) &= s_j + T_{hold} \end{aligned} \tag{3.9}$$

The path slack then can be defined as

$$Pslack(P_l(n)) = req_l(n) - m_n \tag{3.10}$$

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

$$Gslack(i) = \begin{cases} \min\limits_{j \in FO(i)} \{m_j + Gslack(j) - (d_j + m_i)\}, & \text{if gate } i \text{ is not at a PO} \\ \min\{\min\limits_{j \in FO(i)} [m_j + Gslack(j) - (d_j + m_i)],\ (req_l(i) - m_i)\}, & \text{otherwise} \end{cases} \tag{3.11}$$

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, 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 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.

For example, consider a part of a circuit as shown in Figure 3.7. In the circuit, gates 4 and 5 are connected to flip-flops. Gate 4 has gates 1 and 2 as its fan-in gates, while gate 5 has gate 2 and 3 as its fan-in gates. The required signal arrival times at the inputs of both flip-flops are indicated in the figure. The long-path and short-path signal arrival times of gates 1, 2, and 3 are shown in the figure. The delays of gates 4 and 5 are 0.5 and 0.4, respectively. With this information, the long-path and short-path signal arrival times of gates 4 and 5 can be calculated, and are given in the figure. As we can see, the short-path signal arrival time of gate 4 is 2.1, which is less than the required minimum signal arrival time, 2.3. Therefore, it is necessary to process the short paths to gate 4. Gate slacks are calculated using Eq. (3.11). For the time being, we assume that inserting a delay buffer at the output of a gate will not affect the delay of that gate. Since the gate slack of gate 2 is 0.2, we can insert a delay buffer with 0.2 delay immediately at the output of gate 2.[1] After the buffer insertion, it can be seen that $p_4 = 2.3$, $m_4 = 6.0$, $p_5 = 2.2$, and $m_5 = 5.5$. This way, the required minimum signal arrival time at the input of flip-flop A is satisfied, while none of the required maximum signal arrival times are violated. On the other hand, if we insert a delay buffer with delay time 0.3 at the same

[1] Alternatively, we can insert a delay buffer immediately at the input of gate 4.

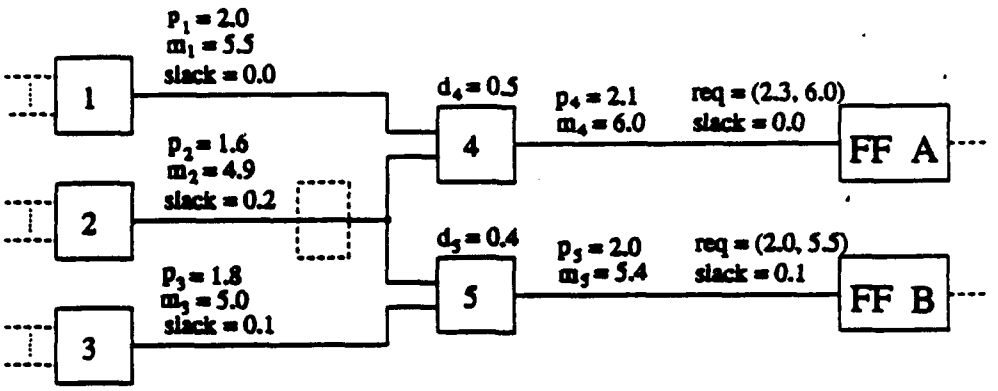

Figure 3.7 An example illustrating the definition of *Gslack*.

location, it can be shown that $p_4 = 2.4$, $m_4 = 6.0$, $p_5 = 2.3$, and $m_5 = 5.6$. Therefore, although the required minimum signal arrival time at the input of flip-flop A is satisfied, the required maximum signal arrival time at the input of flip-flop B is violated due to insertion of the buffer.

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 $req_s(n1) - p_{n1}$.

In a real situation, however, inserting a buffer at the output of a gate will affect the delay of that gate. Therefore, care must be taken when performing buffer insertion to increase short-path delay.

The algorithm for inserting buffers is shown in Figure 3.8. 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 (3.11). In line (4) of the algorithm, beginning

```
ALGORITHM Insert_buffer(n1)

1. Let P_s(n1) be the shortest path to gate n1, and G_n1, ..., G_nk be on path
   P_s(n1) (G_ni fans out to G_n(i-1), 2 ≤ i ≤ k, k = # of gates along P_s(n1).);
2. i ← 1;
3. while ( p_n1 < req_s(n1) ) {
4.     if ( ∃ a (smallest) buffer, bf, in the library such that:
           delay(G_ni) < delay'(G_ni) + delay(bf) ≤ 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 ≥ req_s(n1) ) stop;
8.       else goto 1.
       }
9.     i ← i + 1;
   }
```

Figure 3.8 The buffer insertion algorithm.

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 3.4 In Figure 3.9, 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 the 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

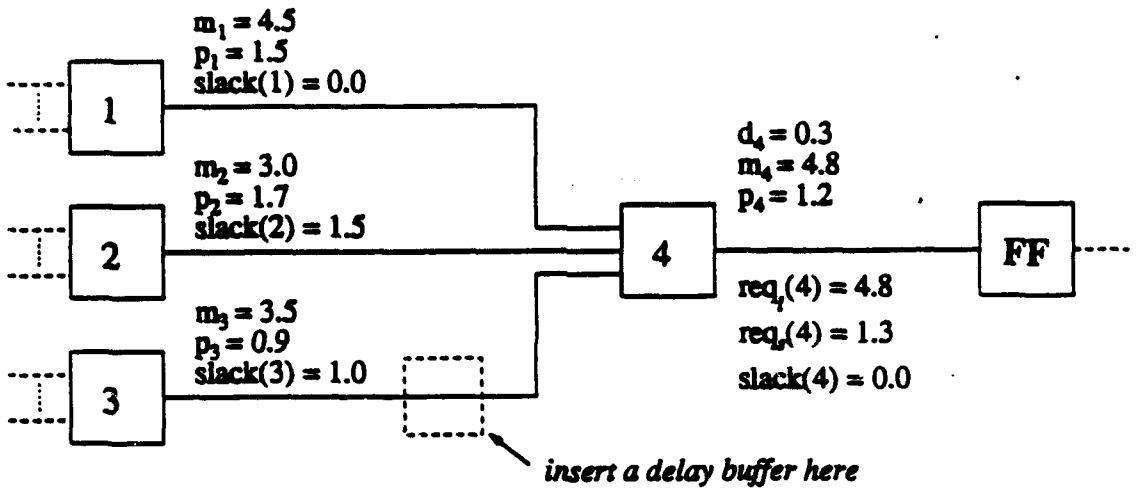

Figure 3.9 An example illustrating buffer insertion algorithm.

gates 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 $req_s(4)$. □

3.6 Experimental Results

The algorithms above were implemented in program GALANT-S on a Sun Sparc10 station. The test circuits include many of the ISCAS85 combinational benchmark circuits [29] and ISCAS89 synchronous sequential circuits [39]. Each cell in the standard-cell library has five different sizes of realization with different driving capabilities.

First, in Table 3.1, the experimental results using the symbolic constraints propagation algorithm are listed. For each circuit, the numbers of primary inputs, primary outputs, flip-flops, and gates are also shown. Both 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 both with and without clock skew optimization are shown in Table 3.2. 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 run time 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 3.3 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 a 30% improvement. Hence, using clock skew optimization can not only reduce the circuit area, but also allows a faster clock speed.

3.7 Comment and Conclusions

3.7.1 Clock tree routing

In [34], Tsay proposed a zero-skew clock tree routing algorithm. In his approach, a clock tree is modeled as an *RC* tree for delay analysis. Based on a lumped delay model

Table 3.1 Experimental results of the symbolic constraints propagation algorithm for ISCAS89 benchmark circuits.

Circuit	# of PIs	# of POs	# of FFs	# of gates	longest-path constraints		
					original	pruned	%
s27	4	1	3	10	133	27	20.3%
s208	11	2	8	104	3276	214	6.5%
s298	3	6	14	137	4556	280	6.1%
s344	9	11	15	160	6720	401	6.0%
s349	9	11	15	160	6816	417	6.1%
s382	3	6	21	158	7488	575	7.7%
s386	7	7	6	171	4758	282	5.9%
s400	3	6	21	162	7824	656	8.4%
s420	19	2	16	196	11830	544	4.6%
s444	3	6	21	181	8592	830	9.7%
s510	19	7	6	211	10775	553	5.1%
s526	3	6	21	229	11688	541	4.6%
s641	35	24	19	379	30402	1331	4.4%
s838	35	2	32	446	55948	2670	4.8%
s953	16	23	29	395	34470	1788	5.2%
s1196	14	14	18	529	32736	2241	6.8%
s1423	17	5	74	657	106379	7953	7.5%
s1488	8	19	6	748	21014	1506	7.2%
s1494	8	19	6	725	20860	1528	7.3%
s5378	35	49	179	2779	911854	6593	0.7%

Table 3.2 Performance comparison with and without clock skew optimization for ISCAS 89 benchmark circuits.

Circuit	P_{spec}	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
		Area (A_1)	Run time	Area (A_2)	Run time	
s27	3.75	151.12	0.32s	179.29	0.30s	0.842
s208	6.8	1404.00	3.32s	1745.25	3.06s	0.805
s298	6.5	2125.50	4.20s	2295.58	4.12s	0.926
s344	8.0	2093.00	7.10s	2400.67	6.91s	0.872
s349	8.0	2128.75	6.18s	2498.17	6.01s	0.852
s382	8.5	2216.50	7.68s	2334.04	6.04s	0.949
s386	6.5	3521.37	7.55s	3577.17	6.14s	0.984
s400	8.4	2314.00	8.19s	2515.50	7.13s	0.920
s420	12.0	2522.00	9.06s	2952.63	8.94s	0.854
s444	8.5	2463.50	11.55s	2724.04	7.22s	0.904
s510	11.0	3219.67	16.13s	3261.37	10.35s	0.987
s526	6.5	3914.08	10.21s	4311.67	9.35s	0.908
s641	22.0	4598.75	51.59s	4747.17	26.49s	0.969
s838	10.5	6162.00	100.67s	7324.42	43.77s	0.841
s953	10.5	5516.87	243.93s	5898.75	67.69s	0.935
s1196	12.0	8550.21	288.15s	8752.42	97.43s	0.977
s1423	35.0	9871.87	1069.75s	10151.38	80.71s	0.972
s1488	10.0	15025.29	148.27s	15322.12	137.61s	0.981
s1494	10.0	14773.96	158.14s	14962.46	115.45s	0.987
s5378	10.0	29219.12	2633.78s	29717.53	1414.49s	0.983

Table 3.3 Improving possible clocking speeds using clock skew optimization.

Circuit	P_{spec}	with clock skew opt.		w/o clock skew opt.		$\frac{A_1}{A_2}$
		Area (A_1)	Run time	Area (A_2)	Run time	
s838	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	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	-	-	-	-	-

and the delay computation method, he found that any two zero-skewed subtrees can be merged into a tree with zero skew by tapping the connection to a specific location of each subtree. The approach is a recursive bottom-up algorithm. To realize the clock routing of a nonzero-skew system as in our approach, Tsay's zero-skew routing algorithm can be modified to handle the problem [34]. This can be done by adding a fictitious delay element on each clock pin.

Let us assume that the optimal clock delay to latch i is $D_0 + D_i$, where D_0 is a common offset value which is unknown until the clock routing is determined. Thus, the skew between latch i and latch j is $D_i - D_j$. Let D_{max} be the maximum clock delay, i.e.,

$$D_{max} = \max_k(D_0 + D_k) = D_0 + \max_k D_k \tag{3.12}$$

Define the fictitious delay of latch i as

$$d_i = D_{max} - (D_0 + D_i) = \max_k D_k - D_i \tag{3.13}$$

In other words, each clock pin attached to a latch is modeled as a lumped delay model with an input loading capacitance and a branch delay, as shown in Figure 3.10. Then the zero-skew routing algorithm is performed on this modified clock tree with the fictitious delay on each clock pin.

3.7.2 Conclusions

In this chapter, a unified approach to minimizing synchronous sequential circuit area and optimizing clock skews has also been presented. Traditionally, the circuit area of a synchronous sequential circuit is minimized one combinational subcircuit at a time. Our

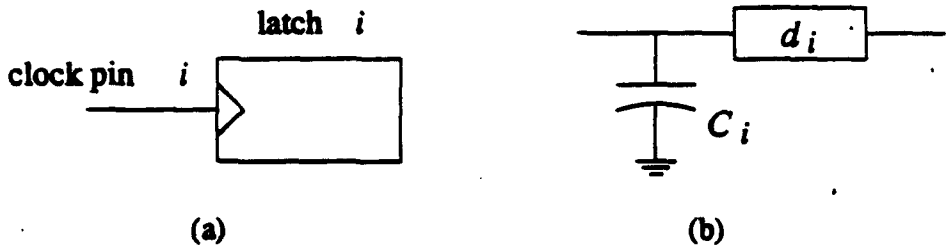

Figure 3.10 (a) A clock pin on a latch. (b) The modified model of a clock pin according to the optimal skew obtained from our algorithm.

experiments have shown that this may lead to very suboptimal solutions in some cases. Experimental results show that using clock skew optimization can not only reduce the circuit area, but also allows a faster clock speed.

In our formulation, for each gate in the circuit, we use the same delay variable (d_k) when calculating longest-path delay and shortest-path delay. In practice, however, the worst-case maximum delay and worst-case minimum delay are different for a specific gate. Nonetheless, our formulation and algorithm described in this chapter can be modified to consider this effect.

In the experimental results, only active circuit area (cell area) is considered. The data do not include the clock tree routing area. It is possible that due to the introduction of clock skew at each latch, the clock tree routing area may be increased. On the other hand, since both positive and negative clock skews are allowed at each latch, it is possible that the net increase in the clock tree routing area may be insignificant. Nevertheless, more thorough study should be conducted before the clock skew optimization technique can be applied to real circuit designs.

Finally, the clock skew scheme may appear similar to the *maximum-rate pipelining* technique used in pipelined computer systems [40]. 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 techniques can be debugged without difficulties.

CHAPTER 4

PARTITIONING FOR OPTIMIZATION

4.1 Introduction

As indicated in Section 3.4, the number of constraints in our formulation of the LP is, in the worst case, 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 Chapter 3 to each subcircuit. While this would entail some loss of optimality, an efficient partitioning scheme would minimize that loss; moreover, the reduction in execution time would be very rewarding.

It is well-known that multiple-way network partitioning problems are NP-hard [41]. Therefore, typical approaches to solving such problems find heuristics that will yield approximate solutions in polynomial time [42, 43]. 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 this chapter, we develop a heuristic measure that will be shown to be an effective objective function for our partitioning problem.

In this chapter, we first briefly discuss previous work on network partitioning in Section 4.2. We develop our partitioning algorithm based on Sanchis' multiple-way partitioning algorithm [42]; details of Sanchis' algorithm are provided in Section 4.3. We present our synchronous sequential circuit partitioning algorithm in Section 4.4. Finally, experimental results are given in Section 4.5, and we conclude this chapter in the same section.

4.2 Previous Work on Partitioning

As VLSI system complexity increases, a divide-and-conquer approach is used to keep the circuit design process tractable. Using this strategy, a complex problem is divided into small subproblems, thus reducing the complexity of the original problem dramatically.

Given a circuit (network) consisting of a set of modules (nodes) connected by a set of signals (nets), the objective of a K-way partitioning is to divide the whole circuit into K subsets such that the number of signals crossing these subsets is minimized.

A network as described above can be modeled as a graph where each edge (net) connects exactly two vertices (node). The graph partitioning problem can be formally stated as follows. We are given an undirected graph, $G = (V, E)$ where vertices $V = \{v_1, v_2, \cdots, v_n\}$ and weighted edges $e = (v_i, v_j)$ represent the cost of putting v_i and v_j in separate partitions. The problem is to divide the vertices into k disjoint sets $\{P_1, P_2, \cdots, P_k\}$ for a given k, such that some cost function is optimized. The cost function can be based on the weights of the edges cut and/or the sizes of the partitions.

Ford and Fulkerson [44] proposed the *max-flow-min-cut* algorithm, which finds the optimum solution between subsets of unconstrained sizes in polynominal time. Using their algorithm, a minimum cut separating designated nodes s and t can be found by flow techniques in $O(n^3)$ time, where $n = |V|$ = number of vertices in the graph. Cut-tree techniques [45] will yield the global minimum cut using $n - 1$ minimum cut computation in $O(n^4)$ time. However, these algorithms tend to generate very unbalanced partitions. Unfortunately, when size balancing constraints are imposed, the problem becomes NP-complete. Because of its importance, many heuristics have been proposed to solve the partitioning problem [42, 43, 46–51]. These heuristics can be classified into the following two categories:

(1) *Iterative method* - Iterative heuristics explore the solution space by making a large number of moves (small changes to the solution) either randomly or greedily in an attempt to discover a global minimum.

(2) *Spectral method* - In spectral partitioning techniques, the eigenvectors and eigenvalues (spectrum) of a graph are computed, and a cost function is shown to be minimized by a function of the spectrum. Some heuristic is used for mapping the information provided by the eigenvectors into an actual partition.

The two approaches are discussed in more detail in the following two subsections.

4.2.1 Iterative partitioning

In [46], Kernighan and Lin described a heuristic procedure for graph partitioning which became the basis for most of the iterative improvement partitioning algorithms generally used. Their algorithm deals with the problem of partitioning a network with n cells (vertices) (n even) into two partitions of $n/2$ cells each. The basic approach is to start with a given partition and to improve it by iteratively choosing one node from each of the blocks (partitions) and exchanging them. The nodes are selected to be switched so that a maximum decrease in cut-set size is obtained (or minimum increase if no decrease is possible). The algorithm consists of a series of *passes*. In each pass, two nodes are interchanged in turn until all n nodes have been moved. In each iteration, the two nodes to be moved are chosen from among the ones which have not been moved during the pass. At the end of each pass, the $n/2$ partitions produced during the pass are examined

and the one with the minimum cut-set size is chosen as the starting partition for the next pass. Passes are performed until no improvement in cut-set size can be obtained.

Fiduccia and Mattheyses [47] modified the Kernighan-Lin algorithm. Their algorithm has a linear worst-case complexity in each pass. In their algorithm, only one cell is moved between two partitions at a time instead of switching pairs. This allows for more flexibility in block sizes. In addition, a method is introduced for keeping the candidates in each partition sorted at all times. Elegant data structures were developed through which they could maintain the sorted candidates, and thus avoiding searching for a candidate to be moved. Hence a linear-time complexity is achieved. Fiduccia and Mattheyses also introduced the idea of preserving balance in the sizes of the blocks. Since only one cell is moved at a time, block sizes cannot be contrained to be constant during the pass. Instead, each block's size is constrained to be within a given range. When choosing the next cell to be moved, the cell with the highest gain (reducing maximum number of cuts across the partition) in each block is examined. It will always be possible to move at least one of these cells while preserving balance. If both may be moved, the one with the highest gain is selected.

Krishnamurthy [48] further improved the Fiduccia-Mattheyses algorithm by refining the method for choosing the best cell to be moved. He introduced the concept of *level gain*. Consider the example shown in Figure 4.1. Moving A would eliminate one net; moving B, however, would not eliminate any net. However, if we move B and C together, two nets can be eliminated. Therefore, the first level gain (γ_1) of A is 1, and that of B is 0; while the second level gain (γ_2) of A is 0, and that of B is 2. The gain vector of a

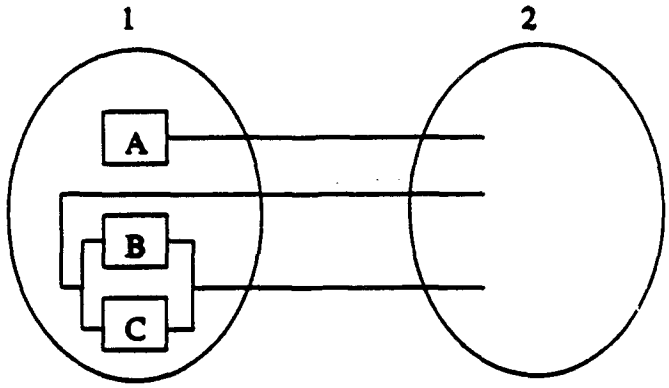

Figure 4.1 An example illustrating the concept of level gain.

cell E is then defined as

$$\Gamma_l(E) =< \gamma_1(E), \cdots, \gamma_l(E) > \tag{4.1}$$

where l is the number of levels used. These vectors are ordered lexicographically. At each iteration, the free cell with the largest gain vector is moved. Computing higher-level gains enables the algorithm to better distinguish between cells whose first-level gains are the same.

In [42], Sanchis further generalized Krishnamurthy's algorithm to deal with the multiple-way partitioning problem. There are several ways in which a two-way partitioning algorithm can be adapted to multiple-way partitioning. For example, one can successively choose pairs of blocks and apply the two-way algorithm to these pairs. However, since eliminating a net from the cut-set formed across a given pair of blocks does not necessarily remove it from the cut-set of the multiple block partition, this method may not be able to obtain good results. The second method consists of a hierarchical use of the two-way algorithm. For example, for a four-way partition we could use the two-way al-

gorithm to partition the cells into two blocks, and then partition each of these two blocks into two blocks each. However, the first partitioning will try to minimize the number of connections between the first two blocks, thus tending to maximize the connections inside these two blocks and making it harder to obtain good partitions thereafter. An alternative for obtaining better solutions is to attempt to improve the partition uniformly at each step. Under such a scheme, we should consider at each iteration during a pass all possible moves of each free cell from its home block to any of the other blocks, and the best of such moves should be chosen. This is the basice approach taken by Sanchis. Since we develop our partitioning scheme based on Sanchis' algorithm, details of the algorithm will be discussed in Section 4.3.

More recently, Yeh et al. [43] proposed a general-purpose multiple-way partitioning algorithm. In their approach, a top-down clustering is carried out first to group highly connected subsircuits into clusters and then condense these clusters into single nodes prior to the execution of iterative procedure. They also proposed a uniform multipin net model to capture the contributory moves. Consider the example shown in Figure 4.2. Suppose during a pass, nodes A, B and C have not been locked. Moving A would eliminate nets e_i, e_k, and e_l and introduce net e_n to the cut-set. Thus the gain would be 2. Moving B would not eliminate any net, but would allow for nets e_i, e_j, and e_k to be eliminated, provided that node C is moved along with B. If we use the level gain model, the moving of A would be favored, which would depress the movement of B and C. Based upon the above observation, a different approach is introduced. Let us now concentrate on the perspective of nets. If we want to eliminate net e_j, we would have to move B and C

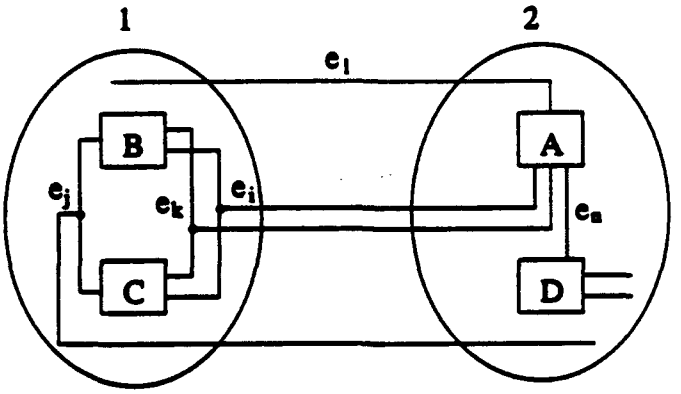

Figure 4.2 An example for multipin net model.

together. This would also introduce the elimination of nets e_i and e_k at the same time. Thus the gain would be 3. On the other hand, if we decide to remove net e_l, we would move A only, and the gain is 2. Therefore, if we view a move as initiated by a net instead of a node, the ambiguity associated with selecting moves would be reduced. Based on this model, a primal-dual iteration is used to enhance the iteration improvement. The primal process is based on the Fiduccia-Mattheyses algorithm. The dual process is similar to the primal process except that it concentrates on net perspective during the pass.

4.2.2 Spectral partitioning

Spectral-based partitioning extracts information about the structure of the graph from the eigenvalues and eigenvectors of the matrices derived from the graph. A graph can be represented by the *adjacency matrix* $A(G)$.

$$A_{ij} = \begin{cases} a_{ij}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

where a_{ij} is the weight of the edge between v_i and v_j. By convention, $A_{ii} = 0$ for all $i = 1, \cdots, n$. If we let $d(v_i)$ denote the degree of node v_i (i.e., the sum of weights of all edges incident on v_i), we obtain the $n \times n$ diagonal *degree matrix* $D(G)$ defined by

$$G_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik}, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{4.3}$$

The *Laplacian* of G is the $n \times n$ symmetric matrix $Q(G) = D(G) - A(G)$. Since the rows (and columns) sum to 0, the Laplacian is singular; it has rank of at most $n-1$ and 0 as an eigenvalue. In fact, the multiplicity of the 0 eigenvalue is the number of connected components of G.

Donath and Hoffman [52] derived a lower bound on the weight of the edges cut (E_c) by a partition satisfying predetermined partition sizes. If $m_1 \geq m_2 \geq \cdots \geq m_k$ are the given partition sizes and $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k$ are the smallest k eigenvalues of the Laplacian, then $E_c \geq \frac{1}{2} \sum_{i=1}^{k} \lambda_i m_i$.

In [53], Hall showed that the eigenvalues/eigenvectors of the Laplacian solve the one-dimensional quadratic placement problem of finding the vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ which minimizes the total weighted squared distance between n points that can be expressed as

$$z = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j)^2 A_{ij} \tag{4.4}$$

subject to the constraints $|\mathbf{x}| = (\mathbf{x}^T \mathbf{x})^{1/2} = 1$.

Here x_i is the coordinate assigned to vertice v_i in a one-dimensional space. The constraint is imposed to avoid the trivial solution in which all x_is are equal. Equation

(4.4) can be rewritten in matrix notation in quadratic form as

$$\begin{aligned} \textbf{minimize} \quad & z = \mathbf{x}^T Q \mathbf{x} \\ \textbf{subject to} \quad & (\mathbf{x}^T \mathbf{x})^{1/2} = 1 \end{aligned} \tag{4.5}$$

To solve this constrained minimization problem, we form the Lagrangian

$$L = \mathbf{x}^T Q \mathbf{x} - \lambda(\mathbf{x}^T \mathbf{x} - 1) \tag{4.6}$$

Taking the partial derivative of L with respect to $\mathbf{x}$ and setting it equal to 0 yields

$$2Q\mathbf{x} - 2\lambda\mathbf{x} = 0 \tag{4.7}$$

which can be rewritten as

$$(Q - \lambda I)\mathbf{x} = 0 \tag{4.8}$$

where I is the identity matrix.

This is an eigenvalue formulation for λ. For a system of n linear equations, there are n possible eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k$. For a connected graph, the Laplacian has rank of $n - 1$. The minimum eigenvalue 0 gives the trivial solution $\mathbf{x} = (1/\sqrt{(n)}, \cdots, 1/\sqrt{(n)}$. Hence the eigenvector corresponding to the second smallest eigenvalue, λ_2 is used. The second smallest eigenvalue is a lower bound on a nontrivial solution to (4.5). In his paper, Hall heuristically derived a k-dimensional generalization in which the eigenvectors are used as the basis for clustering placement.

Recently, Hagen and Kahng [49] established a connection between Hall's formulation and 2-way ratio-cut [54] partitioning. They construct a 2-way partition from v_2 (the

corresponding eigenvector of λ_2) by sorting v_2 and identifying a cut in the sorted v_2 which yields the best ratio-cut value.

In [51], Chan et al. developed a spectral approach to multiple-way ratio-cut partitioning which provides a generalization of the ratio-cut cost metric to k-way partitioning and a lower bound on this cost metric. Their approach involves finding the k smallest eigenvalues/eigenvector pairs to the Laplacian. The eigenvectors provide an embedding of the graph's n vertices into a k-dimensional subspace. A heuristic is then used to enforce the points in the embedding into k partitions.

4.3 Sanchis' Multiple-way Partitioning Algorithm

A cell is labeled *free* if it has not been moved during the pass; otherwise, it is labeled *locked*. Define

$$\begin{aligned} \phi_{A_i}(N) &= |\{C | C \in A_i \text{ and } C \in C_N \text{ and } C \text{ is } free\}| \\ \lambda_{A_i}(N) &= |\{C | C \in A_i \text{ and } C \in C_N \text{ and } C \text{ is locked}\}| \end{aligned} \tag{4.9}$$

Thus $\phi_{A_i}(N)$ is the number of free cells on the net N which are in the block A_i, while $\lambda_{A_i}(N)$ is the number of locked cells on net N which are in the block A_i. For each block A_i and each net N, define the binding numbers β:

$$\beta_{A_i}(N) = \begin{cases} \phi_{A_i}(N) & \text{if } \lambda_{A_i}(N) = 0 \\ \infty & \text{if } \lambda_{A_i}(N) > 0 \end{cases} \tag{4.10}$$

The binding number of a net with respect to a block of a partition indicates how tightly the net is bound to the block. We also define the function β' as follows:

$$\beta'_{A_i}(N) = \sum_{j \neq i} \beta_{A_j}(N) \tag{4.11}$$

That is, $\beta'_{A_i}(N)$ is the sum of all of the binding numbers of net N with respect to all of the blocks of the partition except block A_i; it gives a measure of how tightly N is bound to the partitions other than A_i.

We now define the ith level gain associated with moving cell C from block A_j to block A_k.

$$\begin{aligned} \gamma_i^k(C) = \quad & |\{N \in N_C | \beta'_{A_k}(N) = i \text{ and } \beta_{A_k}(N) > 0\}| \\ & -|\{N \in N_C | \beta'_{A_j}(N) = i-1 \text{ and } \beta_{A_j}(N) > 0\}| \end{aligned} \tag{4.12}$$

The first term in (4.12) measures the ith level benefit of moving cell C from the side of the partition consisting of all blocks except A_k, to A_k. The second term measures the ith level penalty of moving C from A_j to the side of the partition consisting of all blocks except A_j.

The balance requirement for the block sizes can be satisfied as follows. Let $r_1, \cdots, r_b$ be such that $0 \leq r_i \leq 1$ for each i and

$$\sum_{i=1}^{b} r_i = 1 \tag{4.13}$$

We want the size of A_i to be close to $r_i c$, where c is the total number of cells in the network. A parameter w is chosen such that $0 < w \leq \min_{1 \leq i \leq b}(r_i c)$, and we allow the following range for the size of A_i:

$$r_i c - w \leq |A_i| \leq r_i c + w \tag{4.14}$$

That is, a cell move from A_i to A_j is allowed if it preserves the above relationship for A_i and A_j.

Given the initial partition, the algorithm improves the partition by iteratively moving one cell from one block to another in a series of passes. A cell 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 block does not violate the constraints given by (4.14). The gain vector, $\Gamma_i^k(C)$, as defined in (4.12), is updated constantly as cells are moved from one block to another. At the end of each pass, the partitions generated during that pass are examined and the one with the minimum cut-set size is chosen as the starting partition for the next pass. Passes are performed until no improvement of the objective value can be obtained.

4.4 Synchronous Sequential Circuit Partitioning

To help us describe our partitioning algorithm, we introduce the following terminology. For a synchronous sequential circuit, such as one shown in Figure 4.3, we define the following:

An **internal latch** is a latch whose fan-in and fan-out gates belong to the same combinational block.

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

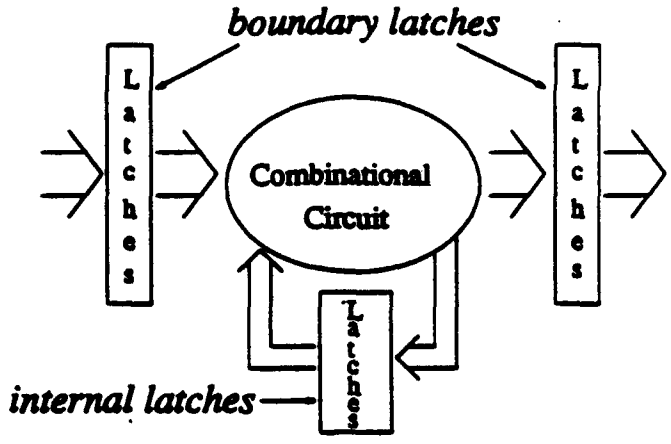

Figure 4.3 An example illustrating the definition of an internal latch, a sequential block, and a boundary latch.

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 fan-in and fan-out gates belong to different combinational blocks.

A partition of a synchronous sequential circuit **N** is a partition of the sequential blocks of **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. After partitioning, boundary latches that lie between groups (that do not belong to any groups) will be set to have constant skews. In other words, we do not have any control on the skews of those latches during the optimization process.

For a given sequential block **B**, let $L_{\mathbf{B}}$ denote the set of boundary latches incident on **B**, and for a given boundary latch L, $\mathbf{B}_L$ denotes the set of sequential blocks that L is

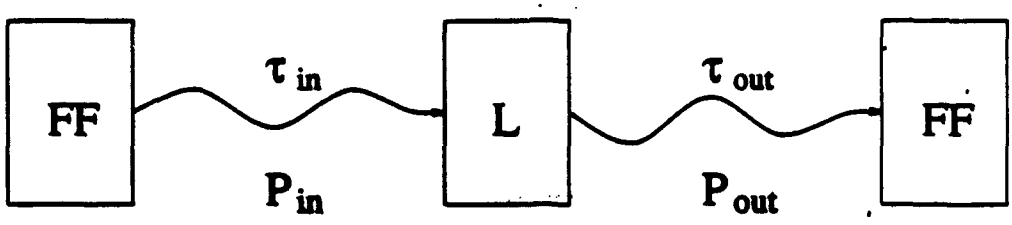

Figure 4.4 Tightness factor.

connected to. For each boundary latch L, we define *input tightness* τ_{in}, *output tightness* τ_{out}, and the *tightness ratio* τ as

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

$$\tau(L) = \begin{cases} \tau_{in}/\tau_{out} & \text{if } \tau_{in} \geq \tau_{out} \\ \tau_{out}/\tau_{in} & \text{if } \tau_{in} < \tau_{out} \end{cases} \tag{4.15}$$

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 example, in Figure 4.4, if the input tightness (of path P_{in}) is 3.0, and output tightness (of path P_{out}) is 1.1, retiming the path $P_{in} \bigcup P_{out}$ would be of great benefit. Therefore, if the circuit contains an FF whose input and output are connected to paths with vastly different tightness factors, the two paths should remain in the same partition.

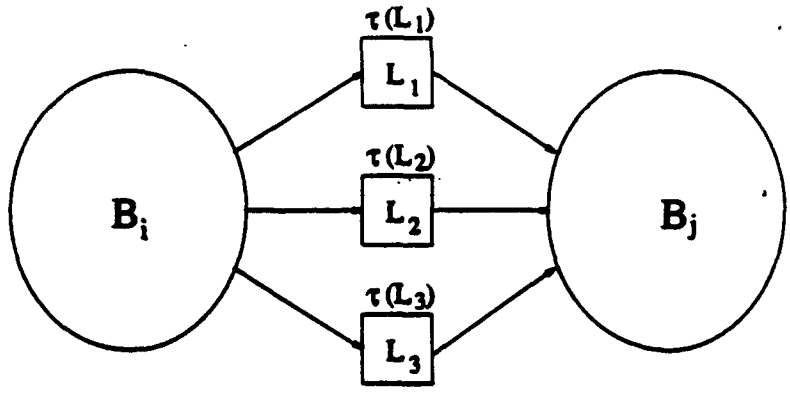

Figure 4.5 Example showing the definition of merit.

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

$$\mu_{ij} = \sum_{\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j} \tau(L_k) \tag{4.16}$$

where $\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j$ means latch L_k lies between $\mathbf{B}_i$ and $\mathbf{B}_j$. The value of μ_{ij} is defined to be 0 if $\mathbf{B}_i$ and $\mathbf{B}_j$ are disjoint. For example, in Figure 4.5, sequential blocks ($\mathbf{B}_i$ and $\mathbf{B}_j$) are connected through three latches L_1, L_2, and L_3 with tightness ratios $\tau(L_1), \tau(L_2)$ and $\tau(L_3)$, respectively. Then the merit between the two blocks is $\mu_{ij} = \tau(L_1)+\tau(L_2)+\tau(L_3)$. 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:

$$max \qquad \sum_{k=1}^{N} M(\mathbf{G}_k)$$

$$subject\ to\ \ C(\mathbf{G}_k) < \alpha \cdot MaxConstraints \tag{4.17}$$

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 [42].

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

$$\Phi_{G_k}(L) = |\{\mathbf{B}|\mathbf{B} \in \mathbf{G}_k \ and\ \mathbf{B} \in \mathbf{B}_L\}| \tag{4.18}$$

Since each boundary latch connects exactly two blocks, $\Phi_{G_k}(L) \in \{0,1,2\}$. In other words, if $\mathbf{B}_i \overset{L}{\leftrightarrow} \mathbf{B}_j$, then (a) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \notin \mathbf{G}_k, \Phi_{G_k}(L) = 0$ (Figure 4.6(a)), (b) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k$, or vice versa (Figure 4.6(b)), $\Phi_{G_k}(L) = 1$, and (c) if $\mathbf{B}_i \in \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k, \Phi_{G_k}(L) = 2$ (Figure 4.6(c)).

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

$$\begin{aligned} \Gamma_{ij}(\mathbf{B}) \ = \ & \sum_l (\tau(L_l)|L_l \in L_{\mathbf{B}} \ and\ \Phi_{G_j}(L_l) = 1) \\ & - \sum_n (\tau(L_n)|L_n \in L_{\mathbf{B}} \ and\ \Phi_{G_i}(L_n) = 2) \end{aligned} \tag{4.19}$$

The first term of (4.19) measures the benefit of moving $\mathbf{B}$ to $\mathbf{G}_j$, while the second measures the penalty of moving $\mathbf{B}$ out of $\mathbf{G}_i$.

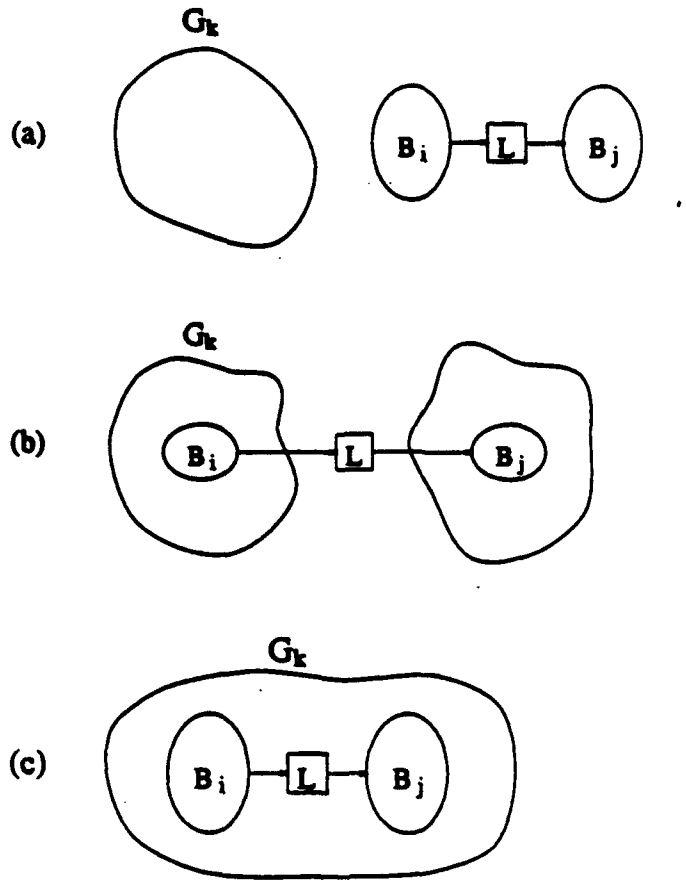

Figure 4.6 Example showing calculation of connection numbers.

As an example, consider a scenario illistrated in Figure 4.7. According to the figure, latch L_m belongs to group G_i, while L_l does not belong to any group. Therefore, we are able to change the skew of latch L_m, but not that of latch L_l. If we move sequential block B from G_i to G_j, latch L_l would be included in group G_j, which means that we are able to adjust the skew of latch L_l when we apply our optimization procedure on group G_j. On the other hand, now L_m does not belong to any group. Therefore, by moving B from G_i to G_j, we obtain control over latch L_m, and the benefit is $\tau(L_l)$. Also we lose control

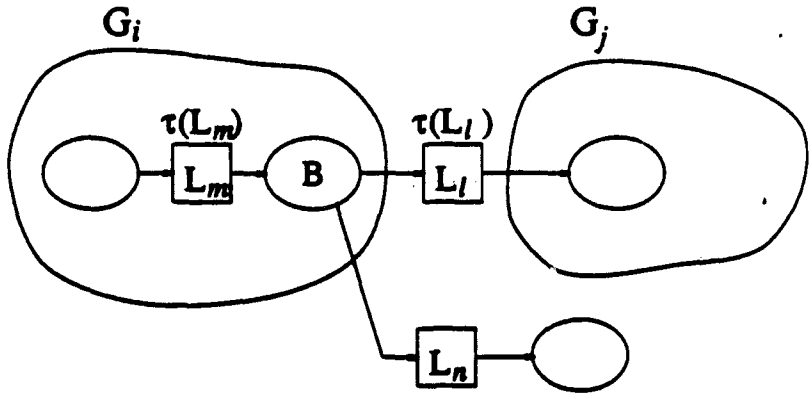

Figure 4.7 Moving gain.

on latch L_m, which gives a penalty of $\tau(L_m)$. Finally, latch L_n does not play a role here, since it does not belong to groups $\mathbf{G}_i$ or $\mathbf{G}_j$, before or after the moving.

Before beginning the partitioning procedure, the number of linear programming constraints, c_i, required for each block i is calculated using the 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

$$TotalConstraints = \sum_j (c_j | c_j < MaxConstraints) \tag{4.20}$$

Each remaining block is put into one of the N' groups,

$$N' = \left\lceil \frac{TotalConstraints}{MaxConstraints} \right\rceil, \tag{4.21}$$

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, [55], 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 (4.21).

After the partitioning, we apply the optimization algorithm described in Chapter 3 to each group.

4.5 Experimental Results and Conclusion

Table 4.1 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 4.1. 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 run time is large. Compared to the first approach, the second approach runs much faster, at a very slight area penalty. Not surprisingly, the third approach gives 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

Table 4.1 Performance comparison of the partitioning procedure.

Circuit	# of PIs	# of POs	# of FFs	# 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.						without	
		w/o partitioning		with partitioning					
		Area	Run time	MxCnst†	N‡	Area	Run time	Area	Run time
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		

† MxCnst = MaxConstraints, the maximum number of contraints.

‡ N, number of groups after partitioning.

using the partitioning procedure improves, while the run time increases only slightly. When $N = 6$, the solution is comparable to that without using partitioning, and the run time is still far less than that without using partitioning.

In summary, in this chapter we develop a synchronous sequential circuit partitioning algorithm. We propose a heuristic measure which is shown to be effective as the objective function of the partitioning problem. Experimental results 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.

CHAPTER 5

DELAY AND AREA OPTIMIZATION FOR PLACEMENT

5.1 Introduction

For standard-cell based VLSI circuits, optimization for improving timing performance can be carried out at three levels in the design process: logic synthesis, gate size selection, and layout. In previous chapters, we have concentrated on optimizing the timing performance of a VLSI circuit by gate sizing. Thus far, we have not considered interconnect delay due to wiring capacitances. As the size of today's VLSI circuits becomes increasingly larger and the device size becomes smaller, the delay of a circuit becomes dominated by interconnect delays [56]. For example, in an SSI or MSI chip designed in 5 μm nMOS technology, the gate input capacitance (15 fF per minimum size transistor) dominates the wiring capacitance (200 fF/mm). A typical transistor with $W/L = 10$ has a capacitance of 150 fF, which is equivalent to 0.75 mm of wire. With a typical MSI die size of a few millimeters on a side, most of the nets will be well below 0.75 mm. According to the scaling theory, when the transistors are scaled down, the gate input capacitance is reduced, while the wiring capacitance per unit length is unchanged [56].

Therefore in a 0.5 μm CMOS technology, a minimum-sized transistor has 1.5 fF input capacitance, which yields a typical transistor ($W/L = 10$) input capacitance of 15 fF. This is equivalent to 0.075 mm of wire, which represents a large number of the nets in a typical 12 $\times$ 12 mm VLSI chip. Consequently, as the devices are scaled down further in submicron technology, the node capacitances will not go down as much as they did in a 5 μm nMOS MSI chip because of the increased role of the wiring capacitance, and the delay of a circuit is dominated by interconnect delay.

In this chapter, we extend our work to consider interconnect delay. We coordinate our work on the gate size selection and placement steps. Layout optimization, also referred to as timing-driven layout is concerned with placement and routing. From the overall chip timing viewpoint, the placement steps are more critical than the routing which can affect mostly the local issues such as noise coupling. For this reason, placement has received more attention in timing-driven layout.

Recently, there has been extensive research on timing-driven placement [10–13]. Timing-driven placement techniques can be broadly divided into two categories: net-oriented and path-oriented. In the net-oriented approach, the acceptable delay of each gate (cell) is calculated and translated into bounds on the delay associated with each net. These bounds then serve as constraints during the subsequent placement step. In the path-oriented approach, timing analyses of critical paths are performed dynamically during the placement step. All paths, or a subset of them, are taken into account implicitly in the formulation. Since the delay of a circuit is inherently path-oriented, it is expected that path-based approaches can obtain better solutions [13, 57].

A standard-cell library typically contains several versions of any given gate type, each of which has a different gate size. The gate-sizing problem is that 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. This is usually done after technology mapping, where the logic function of each gate is determined, and before the physical placement step. A drawback of such an approach is that accurate interconnect wire lengths are not available during the gate-sizing procedure. The gate size selected optimally at that stage may no longer be optimal after the physical design stage where large interconnect capacitances are introduced at the output of each gate. To deal with this problem, an iteration procedure is usually followed. After global placement, the capacitance associated with each net is extracted, and the gate-sizing procedure is repeated. However, in such an iterative approach, the variation of net capacitance between iterations may be large and cause large perturbation in the solutions. Thus, a number of iterations may be required, making this approach quite expensive. To deal with this problem, it is desirable that gate sizing and placement be incorporated into a single procedure.

As an illustration, consider a layout placement shown in Figure 5.1(a). Gate D fans out to gates L_1, L_2 and L_3. Assume that the delay of this circuit under such layout conditions violates the timing constraints imposed on it. Moreover, D and L_2 lie on a long path whose delay exceeds the timing constraint. Conventional timing-driven placement would move D, L_1, L_2 and L_3 closer to one another to decrease the delay of gate D, as shown in Figure 5.1(b). This may increase the wire lengths of other nets attached to cells D, L_1, L_2 and L_3. But if automatic gate sizing is incorporated with timing-driven placement, a possible solution would be to replace D with a template with a higher driving capacity, and L_1 with one with a smaller loading capacitance with respect to D. As a result, some of the cells could be moved to better locations, as shown in Figure

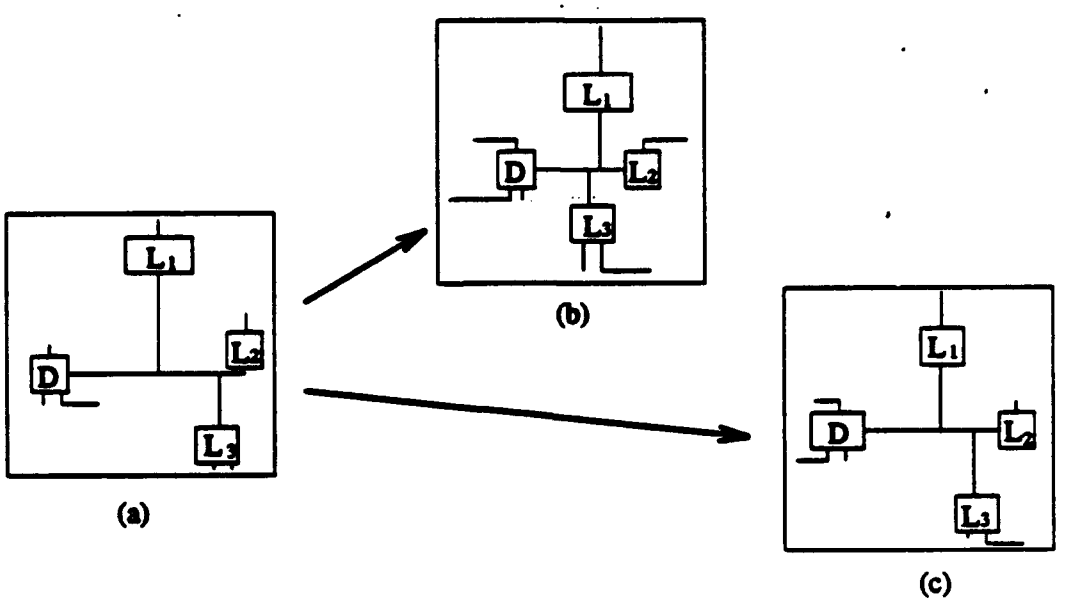

Figure 5.1 Advantage of gate sizing together with placement.

5.1(c). The overall effect is a reduction of the long-path delay, while the increase in area is kept to a minimum.

In [58], Kim et al. propose an area-timing-testability driven placement algorithm. Their algorithm consists of a series of iterations. At the beginning of each iteration, a placement using Timberwolf [59] is done to minimize the total wire length. After placement, a set of partial scan flip-flops is selected, followed by a gate-sizing step [23]. After gate sizing is done, timing bounds are calculated for each net; then Timberwolf is called again to obtain an improved layout. In each iteration of the annealing step inside Timberwolf, cells switch their positions in an attempt to reduce the total wire length and also to meet the timing bound assigned to each net. Therefore, the algorithm is net-based, and the gate sizing and placement steps are treated separately.

In this chapter, we propose an algorithm which combines the gate-sizing problem and timing-driven placement into one procedure. By considering these two problems

together, the value of the interconnect capacitance is known during the selection stage of the automatic sizing procedure. Therefore, optimal gate sizes can be chosen for each gate based on layout information, thus reducing the number of iterations required in the conventional approach. For simplicity, we use a row-based layout style. However, a more general arrangement can be used. In the following, the terminologies "gate" and "cell" are used interchangeably. Both refer to a module in the circuit. Besides, in the following, we consider combinational circuits only. For a sequential circuit, we can apply our algorithm to a combinational block in the sequential circuit one at a time.

This chapter is organized as follows. Section 5.2 briefly discusses previous work on timing-driven placement. In Section 5.3 we formulate the task of timing-driven placement with automatic gate sizing in a single optimization problem. In Section 5.4 we describe a novel algorithm which performs delay and area optimization for a given compact placement by means of gate resizing and relocation. Experimental results are provided in Section 5.5. Finally, we conclude the chapter in Section 5.6.

5.2 Previous Work

For many years, timing-driven layout techniques were net-oriented. The timing constraints derived from higher levels were translated into bounds on delay associated with each net, and timing-driven placement and routing were used to synthesize a layout satisfying those constraints. However, timing is not associated with the nets but with the signal flows along paths which are combinations of nets in the circuit. Therefore, instead of satisfying individual net delays, the constraints on the sum of delays of all of the nets constituting a path must be satisfied.

To take into consideration more accurate timing behavior and achieve globally better solutions, timing analysis of critical paths must be performed dynamically during the placement procedure. Such a technique is proposed by Jackson and Kuh in [10] where a sequence of linear programming steps is used to determine the cell placement in a hierarchical approach. The delay behavior is modeled in a path-oriented manner and considers intercell delays as well as interconnect and pin capacitances.

Sutanthavibul and Shragowitz [60] proposed a hierarchical constructive placement algorithm with look-ahead and adaptive placement capabilities. The delay functions are computed based upon the net geometry, capacitance per unit wire length, and the net loading, to arrive at the path delay values.

Donath et al. [61] introduced an approach in which the timing is evaluated together with routability in the global placement step. The parameterized delay equations are used in the path analysis. During the placement of cells on the critical paths, fast incremental timing analysis is performed to evaluate the feasibility of each move. A complete timing analysis is done after each major step.

Srinivasan et al. [11] proposed an approach based on Lagrangian Relaxation. They observed that only a small subset of timing requirements is active as constraints at one time, thus the problem of a large number of paths can be effectively avoided. They represented timing requirements by a set of linear inequalities. When the corresponding constrained optimization problem is turned into a Lagrangian, these linear inequalities make the Lagrangian nondifferentiable. The subgradient method was used to update Lagrange multipliers on the nondifferentiable Lagrangian.

Most recently, Hamada et al. [13] proposed an algorithm which also transforms the placement with timing constraints into a Lagrangian problem. A primal-dual approach

is then used to find the optimal relative module locations. In each primal dual iteration, the primal problem is solved by a piecewise linear resistive network method, while the dual process is used to update the Lagrange multipliers by using the Newton method.

5.3 Timing-Driven Placement with Gate Sizing

Typically, a path-based timing-driven placement algorithm formulates the placement problem as an optimization problem, with both timing requirement and physical placement requirement as constraints. The constraints are usually linear ones. The objective function can be either a linear function or a quadratic function of the cell coordinates. A quadratic objective function allows efficient quadratic programming techniques to be used, thus the problem can be solved relatively fast. However, in [62], it was observed that a linear objective function tends to reflect the actual wiring demands more accurately than the quadratic objective function. Therefore, we choose to use a linear objective function in our approach.

A circuit can be modeled as a set of M gates (cells), $\mathcal{G} = \{g_1, \cdots, g_M\}$, interconnected by a set of N nets, $\mathcal{N} = \{n_1, \cdots, n_N\}$, that attach to the cells at pins. For the sake of simplicity, we assume that all gates in the circuit are of single output. Therefore, net n_i is associated with gate g_i. Hence, the same index i can be referred to as both a gate and a net. We also assume that all pins are located in the centers of cells. Therefore, the physical location of a cell i on the chip is represented by (x_i, y_i), where x_i (y_i) is the x (y) coordinate of cell i. The positions of the I/O pads are fixed and located on the perimeter of the chip. These constraints act as the boundary conditions.

Figure 5.2 Approximating wire length using bounding box method for 2- and 3-pin nets.

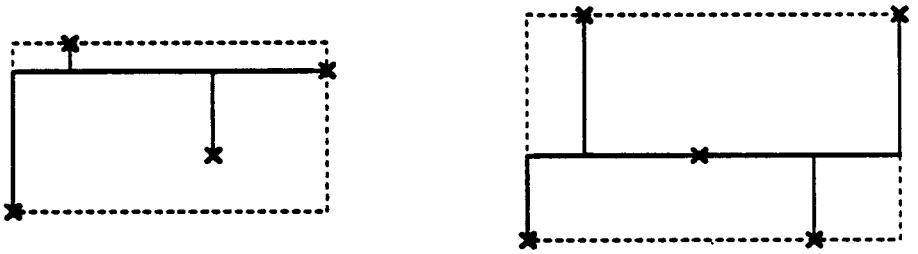

Figure 5.3 Approximating wire length using bounding box method for 4- and 5-pin nets.

There are three categories of constraints in our LP formulation, namely, physical, timing, and sizing constraints.

5.3.1 Physical constraints

We approximate the wire length of an individual net by the half-perimeter of the smallest rectangle enclosing the pins of the net [10]. This approximation is the same as the rectilinear, minimal Steiner tree length for two- or three-pin nets (Figure 5.2). The approximation error for four- and five-pin nets is within the width of the bounding box of the Steiner tree length [63] (Figure 5.3). The bounding box for net i is denoted by four

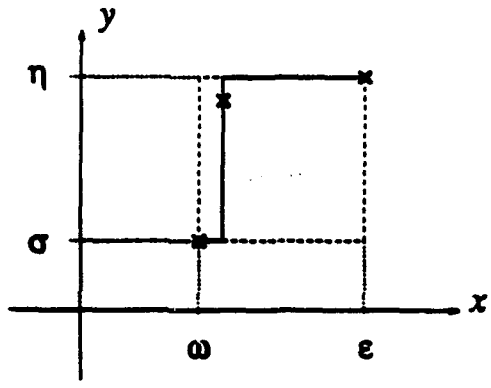

Figure 5.4 Approximating wire length using bounding box method.

parameters, the northernmost (η_i), southernmost (σ_i), easternmost (ε_i), and westernmost (ω_i) extents of the pins of the net (Figure 5.4). Mathematically, the bounding box constraints can be expressed as follows:

$$\begin{aligned} \varepsilon_i &\geq x_{ij}, \\ \omega_i &\leq x_{ij}, \\ \eta_i &\geq y_{ij}, \\ \sigma_i &\leq y_{ij}, \qquad \forall\, 1 \leq j \leq p_i \end{aligned} \tag{5.1}$$

where p_i is the number of pins associated with net i and j is a pin of net i.

Let C_h and C_v denote the unit length wire capacitance in horizontal and vertical layers, respectively. Then the interconnect capacitance, C_i, of net i can be estimated as

$$C_i = C_h(\varepsilon_i - \omega_i) + C_v(\eta_i - \sigma_i) \tag{5.2}$$

Similarly, the length of net i, l_i, is

$$l_i = (\varepsilon_i - \omega_i) + (\eta_i - \sigma_i) \tag{5.3}$$

Therefore, the total wire length of the layout is

$$\sum_{i=1}^{N} l_i \tag{5.4}$$

5.3.2 Timing and sizing constraints

Consider a single-output gate i with $fi(i)$ inputs, and gate i fans out to $fo(i)$ gates. The worst-case signal arrival time at the output of gate i, m_i, can be expressed as

$$m_i \geq m_{ij} + d_i, \qquad 1 \leq j \leq fi(i) \tag{5.5}$$

Now the delay of gate g_i can be contributed to the loading capacitance of its fan-out gates, plus the wire capacitance of its fan-out net n_i. Let CL_{ij}^{i} represent the loading capacitance of gate g_{ij} with respect to gate g_i. Then the delay of gate g_i is

$$d_i = \frac{R_u}{w_i} \times C_{out} + \tau \tag{5.6}$$

$$= \frac{R_u}{w_i} \times (C_i + \sum_{j=1}^{fo(i)} CL_{ij}^{i}) + \tau_1 \cdot w_i + \tau_2 \tag{5.7}$$

$$= \frac{R_u}{w_i} \times \{C_h(\epsilon_i - \omega_i) + C_v(\eta_i - \sigma_i) + \sum_{j=1}^{fo(i)} (\alpha_{ij} \cdot w_{ij} + \beta_{ij})\} + \tau_1 \cdot w_i + \tau_2 \tag{5.8}$$

where α_{ij} and β_{ij} are related to the (transistor) gate terminal area capacitance and (transistor) gate terminal perimeter capacitance of the transistor of cell j to which cell i fans out [17].

As in Section 2.3.1, this is a sum of functions of the form y/w. Therefore, it can be approximated by a piecewise linear function.

5.3.3 Objective function

The objective function of our optimization problem can be formulated as

$$\min(\sum_{i=1}^{M} \gamma_i \cdot w_i + \Upsilon \cdot \sum_{i=1}^{N} l_i) \tag{5.9}$$

where l_i is the length of net i, Υ is a constant. In general, we may want to set Υ equal to the sum of the width of interconnect wire and the minimum distance between two adjacent wires. That way, Υ is the minimum width a wire occupies on the chip.

The objective function in this formulation represents two important quantities to be minimized in physical design. The first term is the total area of the cells. The second term represents the total area taken by the interconnect wires.

5.3.4 Slot constraints

For most placement algorithms using mathematical programming techniques, the solutions in general would yield a placement which could have many cell overlaps. Therefore, placement is usually alternated with partitioning steps that generate constraints for the next step. During each step, the following constraint is introduced for each region:

$$r_i^x = \frac{1}{M_i} \cdot \sum_{j \in \mathcal{M}_i} x_j$$

$$r_i^y = \frac{1}{M_i} \cdot \sum_{j \in \mathcal{M}_i} y_j \tag{5.10}$$

where r_i^x (r_i^y) is the x (y) coordinate of the center of the i th region, $\mathcal{M}_i$. M_i is the number of cells in that region.

Equation (5.10) forces the center of gravity of all cells in the region to be equal to the center of the region. Therefore, the cells are distributed better over the whole placement region.

5.3.5 Final LP

After introducing the constraints and objective function, we are in a position to formulate the following linear programming:

$$\text{minimize} \quad (\sum_{i=1}^{M} \gamma_i \cdot w_i + \Upsilon \cdot \sum_{i=1}^{N} l_i)$$

$$\begin{aligned}
\text{subject to} \quad & \text{For all gates } i = 1 \cdots M \\
& m_j + d_i \leq m_i && \forall\, j \in Fanin(i) \\
& m_i \leq T_{spec} && \forall\, gates\ i\ at\ PO's \\
& d_i \geq \tilde{D}(w_i, w_{i,1}, \cdots, w_{i,fo(i)}, \varepsilon_i, \omega_i, \eta_i, \sigma_i) && (5.11) \\
& w_i \geq Minsize(i) \\
& w_i \leq Maxsize(i) \\
& \varepsilon_i \geq x_{ij} && \forall\, 1 \leq j \leq p_i \\
& \omega_i \leq x_{ij} && \forall\, 1 \leq j \leq p_i \\
& \eta_i \geq y_{ij} && \forall\, 1 \leq j \leq p_i \\
& \sigma_i \leq y_{ij} && \forall\, 1 \leq j \leq p_i
\end{aligned}$$

The above is a linear program in the variables $w_i, m_i, d_i, x_i, y_i, \varepsilon_i, \omega_i, \eta_i$, and σ_i.

5.4 A Unified Algorithm for Adjusting Placement and Gate Sizing

Although it is possible to solve (5.11) directly, the execution time may be excessively large due to the large number of variables and constraints, In this section, we present an algorithm which tackles this problem indirectly, and thus reduces execution time.

Notice that timing-driven placement is needed because, in general, gate sizes are selected *before* the placement procedure, and gate sizes are fixed during placement. This imposes a restriction on the placement tool in the search for a good placement with minimum wire length. On the other hand, although placement tools such as those in [59, 64] can obtain a placement with minimal wire length, the delay of the circuit based on that placement may exceed timing constraints. Recently, it has been suggested that a compact placement which violates the timing constraint could be made to satisfy the delay bound by adjusting the sizes of some gates, without altering the placement topology (Chapter 16, [65]). In the following, we propose an algorithm which combines gate resizing and relocation to satisfy timing constraints, and at the same time the total circuit area (including cell area and wire length) is kept to a minimum, for a given compact placement.

First, all of the gates in the circuit are set to their minimum size. A compact placement is obtained with the objective of minimizing the total wire length. This can be done by using existing placement packages (e.g., Timberwolf [59] or Gordian [64]). After that, the wiring capacitance associated with the output of each gate is calculated. Based on this information, together with the circuit structure, optimal gate sizes are selected using the gate size optimization algorithm described in Sections 2.4 and 2.5. In general, some gates will be selected to have a larger size. This may cause overlap among cells. This problem

```
ALGORITHM Resizing and Relocation()
 1.  do initial placement;
 2.  do initial gate sizing for all cells in the circuit;
 3.  while ( timing constraints are not satisfied ) {
 4.    select gates belonging to type 1, 2, and 3;
 5.    formulate LP (Eq. (5.11)) for these gates;
       (remaining cells serve as boundary conditions)
 6.    solve the LP;
 7.    use mapping algorithm (Sec. 2.4) to obtain permissible size
       for each gate;
 8.    adjust cell locations to avoid overlap;
 9.  }
10.  report final placement;
```

Figure 5.5 An outline of the Resizing and Relocation algorithm.

can be solved by shifting cells to avoid overlap. In general, however, the perturbation on the delays of individual gates may cause the circuit delay to exceed the delay constraints. If that does happen, a conventional approach would repeat the gate-sizing procedure to guarantee that the circuit delay of that specific layout is below the delay constraint. Usually, the gate sizing and placement procedures have to be repeated a few times before a final solution is reached.

Our algorithm, in contrast, does not repeat the gate-sizing procedure all over again. Rather, once the algorithm detects that the delay of the circuit is violated, a number of gates, as described below, are selected. These gates will be resized and/or moved to different locations to satisfy time constrains as well as to minimize total circuit area (including cell area and wire length). The outline of our algorithm is shown is Figure 5.5.

In the following, we described how we select only a small portion of gates in the circuit for resizing and relocation, and how cell resizing and relocation can be combined into one formulation.

In addition to the worst-case signal arrival time, m_i, for each gate i in the circuit we introduce the *required signal arrival time*, r_i. The required signal arrival time is the latest time by which a signal has to arrive at the output of gate i to make the delay at the POs less than the specified delay. The required signal arrival time is defined to be

$$r_i = \begin{cases} T_{spec}, & \text{if gate } i \text{ at PO} \\ \max\{r_j - d_j \mid \forall\, j \in Fanout(i)\}, & \text{otherwise} \end{cases} \tag{5.12}$$

For each gate i, we also define a slack s_i, where.

$$s_i = r_i - m_i \tag{5.13}$$

Definition 5.1 *An active gate i is a gate with $s_i < 0$. The set of all active gates is denoted by C.*

Definition 5.2 *The timing of a circuit layout is said to be satisfied if and only if C is empty, i.e., $s_i \geq 0$, for every gate i in the circuit.*

Definition 5.3 *A critical path is a path in which all of the gates along the path have slack values less than or equal to zero.*

Our objective is to satisfy specified delay bounds and to keep the total circuit area to a minimum. This can be achieved in two ways. The first one is to resize gates. For example, for those gates lying on critical paths, we may replace a gate with a template with a higher driving capacity to reduce the delay of that gate. Alternatively, we may

replace a gate with one with smaller input capacitance, thus reducing the delay of its driving gates. The second one is to move some cells to new locations, so as to reduce the interconnect wiring capacitance attached to those gates lying on critical paths. This will also reduce the delays of these gates.

The unified optimization algorithm begins by calculating the slack of each gate. Then three types of gates are selected for improvement.

(1) The first type is active gates, which are gates with negative slack. These gates will be allowed to change their sizes as well as be free to move to new locations. Since active gates are those with worst-case signal arrival times later than the required signal arrival time, it is most important to adjust the delays of active gates such that the circuit delay is less than the specified timing constraints. However, in addition to adjusting the size and location of an active gate, the following two types of cells should also be included in the linear program.

(2) The second type involves those gates with nonnegative slacks less than a small specified value, δ. During this phase some gates will change their size, and some others will be moved to new locations; as a result, output load capacitances of certain gates will increase (while those of others will decrease). For those gates with large slacks, it is likely that such changes, although they will increase their delay, will not make their slack negative. Therefore, those gates with larger slacks are likely to remain nonactive. However, for those gates with small nonnegative slacks, it is possible that such a delay increase will make their slacks become negative. Therefore, it is more advantageous to include those gates with small nonnegative slacks in the formulation to avoid additional iterations.

(3) The third type of gate includes those that are directly connected to the outputs of active gates. Remember that active gates are those which violate timing constraints. Therefore, reducing the delays of these gates, besides changing their sizes, can also be accomplished by reducing their output load capacitances. This can be done by either moving or reducing the sizes of the active gates' fan-out cells.

Gates belonging to types 1, 2, and 3 are put into the linear program, (5.11), and a new solution is obtained by solving it. In principle, to obtain a better solution, it is necessary to include all three types of gates in the linear program. In practice, however, to maintain the efficiency of the program, it is necessary to limit the number of gates to be included. In addition, since many gates' locations are fixed, they can serve as boundary conditions for physical constraints. Therefore, the gravity centering constraints, (5.10), are not needed. Furthermore, to avoid drastically changing the solution, each selected gate is allowed to change to its nearest larger or smaller size only.

The solution of such a formulated linear program gives a new size and a new position for each selected cell. The mapping algorithm described in Section 2.4 is used to obtain the permissible size for each gate. Since many cells are moved to new locations, and some of them are replaced with templates of different sizes, there may be overlap among some cells. Therefore, it is necessary to move cells into (slightly) different locations to avoid overlap.

If necessary, the above procedure is repeated until the delay constraints are all satisfied. However, according to our experience, only one iteration is needed in most cases. Also, since only a relatively small number of gates are selected to be resized and relocated, the execution time for each iteration is relatively small (compared to the time needed to resize all gates).

Table 5.1 Experimental results of PRECISE.

Circuit	T_{spec}	Conventional approach			PRECISE			$\frac{A_P}{A_M}$	$\frac{L_P}{L_M}$
		cell area (A_M)	wire length (L_M)	run time	cell area (A_P)	wire length (L_P)	run time		
c432	14.0	3111	785	31.22s	3061	732	45.77s	0.984	0.922
	13.0	3726	912	31.58s	3484	796	59.26s	0.935	0.873
	12.0	-	-	-	4344	913	1min 52s	-	-
c1355	20.0	8096	2612	2min 2s	7997	2356	1min 53s	0.988	0.902
	19.0	8998	2794	3min 18s	8911	2619	2min 51s	0.990	0.937
	18.0	-	-	-	9912	2811	5min 33s	-	-
c2670	25.0	18015	11243	12min 31s	17680	10710	7min 26s	0.981	0.953
	23.0	18648	11462	14min 14s	18408	10840	8min 11s	0.987	0.946
	21.0	-	-	-	19692	11788	8min 57s	-	-
c5315	26.0	37650	23810	47min 31s	37310	23173	25min 7s	0.991	0.973
	24.0	38432	24231	59min 46s	38055	23534	31min 11s	0.990	0.971
	22.5	-	-	-	39351	24344	41min 51s	-	-
c7552	27.0	50968	43625	2h 3min	51046	42524	1h 0min	1.007	0.975
	24.0	-	-	-	52699	43613	1h 14min	-	-
	23.0	-	-	-	54088	43673	2h 2min	-	-
Average Ratio								0.983	0.939

5.5 Experimental Results

The above algorithms have been implemented in C in the program PRECISE (PeRformancE-driven plaCement with automatIc gate SizE optimization) on a Sun Sparc 10 Station.

The experimental results of the program PRECISE, which implements the unified placement improvement and gate resizing algorithms, are summarized in Table 5.1.

To show the effectiveness of our algorithm, we intentionally adjust the value of the interconnect wiring capacitance per unit length, such that interconnect delay accounts for about 30% of the total delay of each circuit. At present, we use Fiduccia's min-cut partitioning algorithm [47] to obtain a compact placement. The partitioning algorithm recursively divides cells into two partitions so that the number of nets that cross the partition boundaries is minimized, until a small number of cells are left in each partition and then cells are placed to their final location. It has been observed that partitioning-based placement tends to spread the wiring across the layout surface and thus produces very routable placement (Chapter 4, [3]). More compact placement can be obtained by using other algorithms (e.g., [59, 64]). For comparison, we also perform placement and gate sizing based on the purely iterative approach. That is, the two procedures of placement adjustment and gate resizing are executed separately and are included in an iteration loop. The experimental results show that PRECISE is able to obtain better solutions than the conventional iterative approach. Moreover, for very tight timing bounds, the conventional approach fails to obtain solutions at all. This is because cell locations are fixed in the conventional approach, and excessively large capacitances may have been introduced at the output of some gates on critical paths. On the other hand, in addition to resizing cells, PRECISE also moves cells to different locations to reduce large wiring capacitance. Therefore, it is able to obtain solutions even for tight delay bounds. Furthermore, since instead of trying to resize all cells, PRECISE resizes only a small portion of cells when timing bounds are violated; as a result, its execution time is faster in general.

5.6 Conclusion

To date, gate sizing and placement are treated separately in different steps during the circuit design process. Such an approach has not caused much trouble because interconnect delay takes up only a small amount of the circuit delay in a chip fabricated using today's VLSI technology. However, as the devices are scaled down in deep submicron technology, the delay of a circuit becomes dominated by interconnect delay. Therefore, it becomes more and more important to combine gate sizing and placement into one procedure.

In this chapter, for the first time, the gate-sizing problem is combined with placement in one formulation. Although the execution time for the combined problem may be excessively large, we propose an indirect approach to fully utilize some special properties of the formulation to develop a novel algorithm which performs delay and area optimization for a given compact placement, by resizing and relocating cells in the circuit layout. The experimental results are very encouraging.

CHAPTER 6

CONCLUSIONS

In this thesis, an efficient algorithm is presented to minimize the area taken by cells in standard-cell designs under timing constraints. Experimental results show that our approach can obtain a near-optimal solution (compared to simulated annealing) in a reasonable amount of time, even for very tight delay constraints.

For synchronous sequential circuits, a unified approach to minimizing circuit area and optimizing clock skews is presented. Traditionally, the circuit area of a sequential circuit is minimized one combinational subcircuit at a time. Our experiments have shown that this may lead to very suboptimal solutions 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 not only reduces total circuit area, but also gives much faster operational clock speed. For large sequential circuits, we also present a partitioning procedure. Experiments shows that our partitioning procedure is very effective. Using our partitioning procedure, our optimization algorithm is able to run at a much faster speed, with no significant degradation in the quality of the solution.

To date, most research on performance-driven placement assumes that gate sizes are selected before the placement stage. This imposes a restriction on the placement tool

in searching for a good placement with minimum wire length. Recently, it has been suggested that a compact placement which violates the timing constraint could be made to satisfy the delay bound by adjusting the sizes of some gates, without altering the placement topology [65]. In this thesis, we have shown that such an approach may lead to solutions of inferior quality. Instead, by considering resizing and moving the locations of some gates in a unified optimization procedure, we are able to obtain better solutions, with smaller execution times than the conventional iterative method. For the first time, the gate-sizing problem is combined with placement in one formulation. Although the execution time for the combined problem may be excessively large, we propose an indirect approach to fully utilize some special properties of the formulation to develop a novel algorithm which performs delay and area optimization for a given compact placement, by resizing and relocating cells in the circuit layout.

6.1 Future Work

In a combinational circuit, there are some paths that can never be excited by any combination at the primary inputs. Hence, these paths can never be critical [66–68]. The presence of false paths in a circuit causes some gates to be sized unnecessarily, since the optimizer tries to reduce the delay along a path that can never be critical. This may lead to a suboptimal solution to the gate-sizing problem. In other words, although a physical level performance optimizer must certify that the delay of the longest sensitizable paths after optimization is not longer than the specified delay, long paths are allowed to exist in the optimized circuit if they are not sensitizable. As demonstrated in [69], most long paths in a complex circuits are actually false. As a result, to optimize the performance

of a large circuit at the physical level, it is important to consider the false path problem in conjunction with gate size optimization.

Retiming [70] has been shown to be an effective technique to optimize the performance of synchronous sequential circuits. Retiming is an operation on a network whereby registers move across logic blocks in order to minimize the clock cycle or the number of registers while maintaining the behavior of the circuits.

The retiming technique considers only the sequential elements of the circuit; it assumes that the combinational logic structure is fixed. In [71], a set of logic synthesis operations has been combined with retiming to optimize sequential circuits for the area and clock period. Peripheral retiming has been used to optimize the performance of pipelined circuits using combinational delay optimization techniques [72]. While these formulations do exist, they are not directly relevant to our work since we assume that we begin our optimization at the end of the logic synthesis stage. It has been shown that a retiming algorithm can be formulated as a mixed integer linear program [70]. This approach, however, may not be applied directly to our problem, since the computational complexity involved would be prohibitive. We propose to seek methods of carrying out retiming through a series of inexpensive local optimization. For example, as shown in Figure 6.1 [9], it can be seen that changing the clock arrival time at a flip-flop is equivalent to changing the delay specifications on the combinational subcircuits to which that flip-flop is connected. The net effect of this is similar to the moving of the flip-flop across combinational logic module boundaries. Therefore the solution to the clock skew optimization problem could also be interpreted to be a new set of timing specifications for each combinational subcircuit, which may be enforced either by permitting a clock skew, or through retiming.

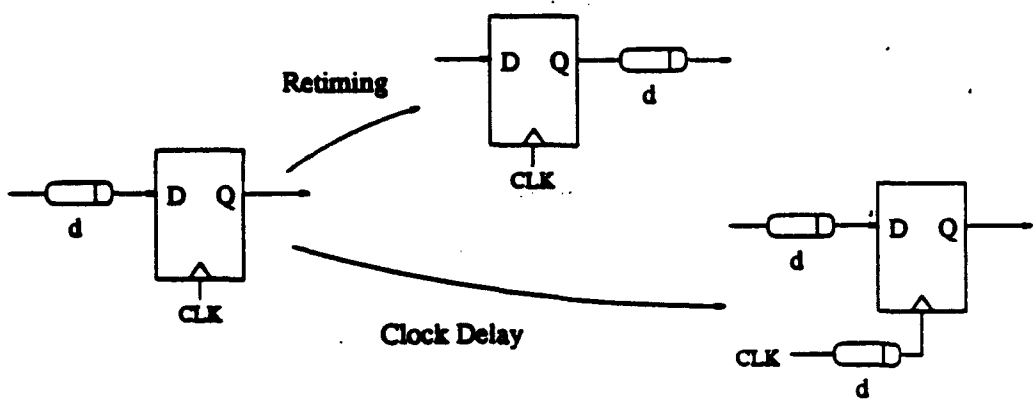

Figure 6.1 Retiming and clock delay transformation.

Methods for using retiming in combination with clock skew for achieving this change in the timing specification should be explored, thus obtaining better solutions for gate-size optimization.

In a real chip, the delay between two successive logic gates is composed of three elements: (1) intrinsic delay due to switching a gate on/off, (2) delays due to charging fanout and load capacitance, and (3) delay due to distributed RC interconnection. The scaling rule suggests that the interconnect delay will be dominant for the circuits with larger chip size and smaller geometry. The effect can be quite significant for submicron circuits since the interconnect delay grows superlinearly with the scaling factor and the chip dimension. As the VLSI fabrication technology reaches submicron device dimensions and gigahertz frequencies, it is necessary to consider such interconnect delays. The timing-driven placement improvement algorithm we propose in Chapter 5 uses a lumped RC model. However, wires on scaled-down ICs have significant resistance and should be analyzed as distributed RC lines. In summary, timing-driven placement and routing remain the challenges of today's submicron device technology.

Currently, there are growing demands for low power circuits for two main reasons. First, as the device size and chip density continue to increase rapidly, with a down scale to 0.6 μm at present (and 0.2 μm expected) and over 100 MHz clock cycles, it becomes too expensive to provide adequate cooling systems for powerful microprocessor chips. For example, using 0.75 μm technology and 3.3 V power supply, DEC's Alpha chip consumes 30 W at 200 MHz [73]. Second, with the increasing popularity of portable consumer products (e.g., laptop/notebook computers and cellular phones), low-power designs become a must, because conventional nickel-cadmium battery technology provides only 20 W·h of energy for each pound of weight [74]. For these reasons, designers now are willing to trade off area for low power consumption.

There has been active research related to low power designs [75–82]. At the architecture level, a parallel implementation can be used to maintain throughput while reducing the supply voltage, thus reducing the power consumption [75]. At the circuit level, a popular technique is to turn off the system clock for those parts of the circuit that are not active. In a CMOS design, the average power consumed by a gate is given by

$$P_{avg} = \frac{1}{2} \times C_{out} \times V_{dd}^2 \times D \tag{6.1}$$

where C_{out} is the output load capacitance, V_{dd} is the power supply voltage, and D is the transition density of the gate [79]. Hence, power consumption of a gate is determined by three factors, namely, C_{out}, V_{dd} and D. At the device level, work is being done to reduce the peak voltage needed for switching (reducing V_{dd}). Other than V_{dd}, we can reduce the power consumed by a single gate by reducing C_{out} and D. Some work has been done in the area of low power logic synthesis. However, as in the area and delay optimization in logic synthesis, the work is applied to technology-independent synthesis, where gate

models for power, delay, and area are not very accurate [76]. Therefore, it is important to perform low-power design at the gate-sizing and physical layout stages.

For the gate-sizing and physical layout problems, it is important to reduce the capacitance load of those gates with large switching activity. For example, if gate i has large switching activity, it is then desirable to reduce its capacitance load due to (1) its fan-out gates, and (2) interconnect wires. For case (1), we should choose a template with smaller input capacitance for its fan-out gates. Hence, the objective function of the optimization should be weighted by the transition density of each gate. To deal with case (2), during placement procedure, it is advantageous to put the fan-out gates of i closer. Similarly, a weight based on each gate's transition density can be included when calculating the total wire length. Therefore, a low-power driven placement algorithm can also be developed.

APPENDIX A

EXTRACTING PARAMETERS FROM A LIBRARY

In this appendix, we will show how various parameters needed in our formulation can be calculated from a given standard-cell library.

As an example, conside a three-input AOI (and-or-inverter) gate whose output logic value is $\overline{a_1 \cdot a_2 + b}$. This gate has three inputs, namely, a_1, a_2, and b. In the given library, this gate has three templates with different cell areas and driving capabilities. Usually, the library lists pin-to-pin delay information, as well as worst-case delay. In our application, we use the worst-case delay. Suppose the characteristics of the three templates are specified as follows.

- Template 1
  - cell area = 1856
  - worst-case delay = $R_{out} \times C_{out} + \tau = 3.64 \times C_{out} + 0.75$
  - pin capacitance of $a_1 = 0.123$
  - pin capacitance of $a_2 = 0.091$
  - pin capacitance of $b = 0.111$
- Template 2
  - cell area = 3401

- worst-case delay = $2.05 \times C_{out} + 1.40$
- pin capacitance of $a_1 = 0.185$
- pin capacitance of $a_2 = 0.175$
- pin capacitance of $b = 0.201$

• Template 3

- Cell area = 5797
- worst-case delay = $1.00 \times C_{out} + 2.30$
- pin capacitance of $a_1 = 0.405$
- pin capacitance of $a_2 = 0.285$
- pin capacitance of $b = 0.345$

Let $R_u = 5.0$. From the above information, we have

• $w_1 = R_u / R^1_{out} = 5.0/3.64 = 1.374$

• $w_2 = R_u / R^2_{out} = 5.0/2.05 = 2.439$

• $w_3 = R_u / R^3_{out} = 5.0/1.00 = 5.000$

where w_1, w_2, and w_3 are permissible gate sizes.

To obtain a linear relationship between the cell area and the gate size, we linearly approximate the set of data points of area vs. w, $\{(1.374, 1856), (2.439, 3401), (5.000, 5797)\}$. Then we have the following linear expression of the gate area in terms of the gate size, w.

$$area = \gamma \cdot w + \varepsilon = 1060.02 \cdot w + 571.346 \qquad \text{(A.1)}$$

Therefore, $\gamma = 1060.02$ and $\varepsilon = 571.346$. The data points and the affine function are plotted in Figure A.1

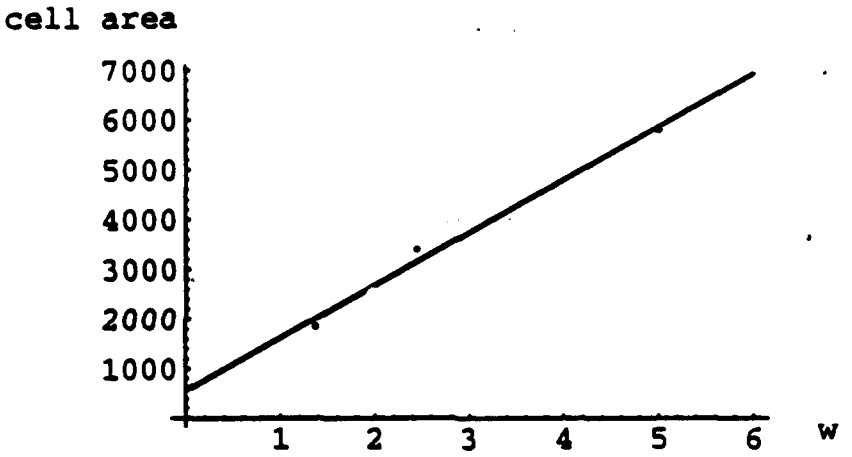

Figure A.1 Calculating γ and ϵ.

The capacitance at input pin a_1 to be used to calculate the loading capacitance, C_{out}, of this gate's fan-in gates can be obtained by linearly approximating the set of data points of a_1 pin capacitance vs. w, $\{(1.374, 0.1229), (2.439, 0.185), (5.000, 0.405)\}$. This gives

$$cap(a_1) = \alpha_{a_1} \cdot w + \beta_{a1} = 0.05885 \cdot w + 0.03007 \tag{A.2}$$

Therefore, $\alpha_{a1} = 0.05885$ and $\beta_{a1} = 0.03007$. The data points and the affine function are plotted in Figure A.2 The capacitances at pins a_1, a_2, and b contribute to the output capacitance of any fan-in gates.

Following the same procedure, we can find that $\alpha_{a2} = 0.06942$, $\beta_{a2} = -0.00033$, and $\alpha_b = 0.06371$, $\beta_b = 0.03336$.

To obtain the values of τ_1 and τ_2, we have to linearly approximate the following data set of τ vs. w, $\{(1.374, 0.75), (2.439, 1.40), (5.00, 2.30)\}$. This gives

$$\tau = \tau_1 \cdot w + \tau_2 = 0.41349 \cdot w + 0.268641 \tag{A.3}$$

Therefore, $\tau_1 = 0.41349$ and $\tau_2 = 0.268641$. The data points and the affine function are plotted in Figure A.3

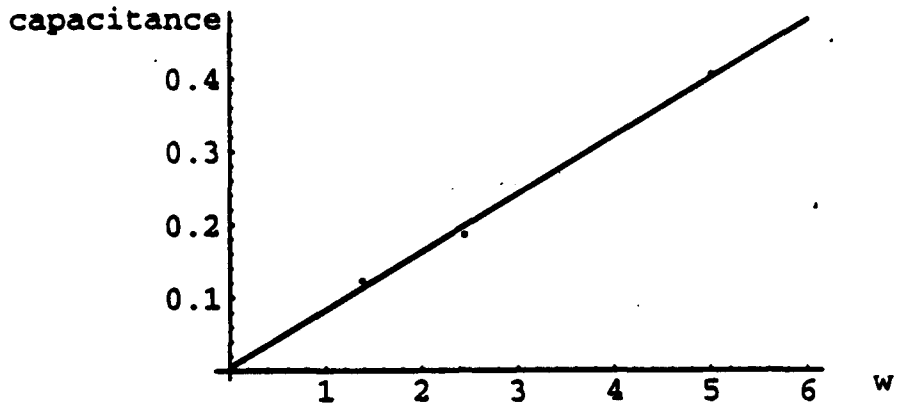

Figure A.2 Calculating α and β.

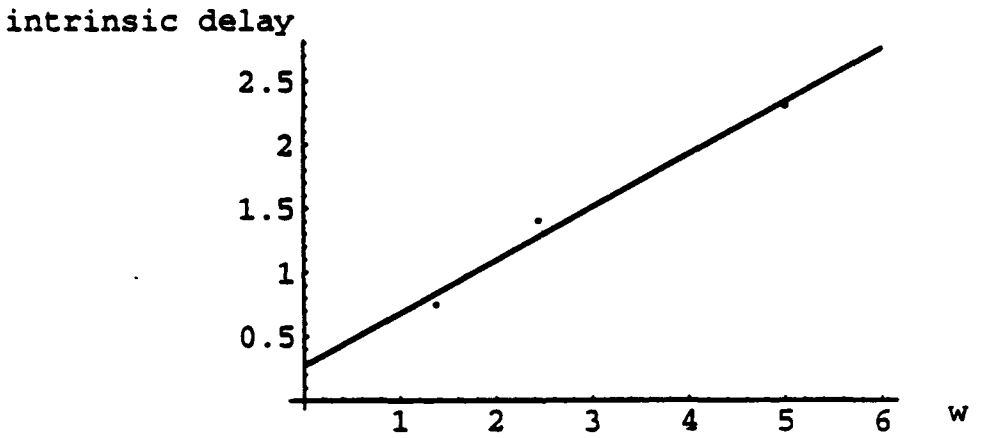

Figure A.3 Calculating τ_1 and τ_2.

Finally, the gate delay can be approximated by the following equation:

$$\begin{aligned} delay &= R_{out} \times C_{out} + \tau_1 \cdot w + \tau_2 \\ &= \frac{5.0}{w} \times C_{out} + 0.41349 \cdot w + 0.268641 \end{aligned} \tag{A.4}$$

REFERENCES

[1] G. Moore, "VLSI: Some fundamental challenges," *IEEE Spectrum*, pp. 30–37, Apr. 1979.

[2] N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design - A System Perspective*. Reading, Massachusetts: Addison-Wesley, 1993.

[3] B. T. Preas and M. J. Lorenzetti, eds., *Physical Design Automation of VLSI Systems*. Menlo Park, California: Benjamin/Cummings, 1988.

[4] A. S.-V. R.K. Brayton, G.D. Hachtel, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, pp. 264–300, Feb. 1990.

[5] T. E. Dillinger, *VLSI Engineering*. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[6] M. Kahrs, "Matching a parts library in a silicon compiler," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 169–172, 1987.

[7] K. Keutzer, K. Kolwicz, and M. Lega, "Impact of library size on the quality of automated synthesis," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 120–123, 1987.

[8] 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.

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

[10] M. A. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 370–375, 1989.

[11] A. Srinivasan, K. Chaudhary, and E. Kuh, "RITUAL: A performance-driven placement algorithm for small cell IC's," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 48–51, 1991.

[12] T. Gao, P. Vaidya, and C. Liu, "A new performance driven placement algorithm," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 44–47, 1991.

[13] T. Hamada, C.-K. Cheng, and P. M. Chau, "Prime: A timing-driven placement tool using a piecewise linear resistive network approach," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 531–536, 1993.

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

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

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

[28] 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.

[29] 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.

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

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

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

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

[35] S. Pullela, N. Menezes, and L. T. Pillage, "Reliable non-zero skew clock tree using wire width optimization," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 165–170, 1993.

[36] M. Shoji, "Elimination of process-dependent clock skew in CMOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 875–880, Oct. 1986.

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

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

[39] 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.

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

[41] M. Garey and D. Johnson, *Computers and Intractability.* San Francisco, California: Freeman, 1979.

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

[43] 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.

[44] L. R. Ford and D. Fulkerson, *Flows in Networks.* Princeton, New Jersey: Princeton University Press, 1965.

[45] C. Cheng and T. Hu, "Maximum concurrent flow and minimum ratio cut," Tech. Rep. CS88-141, University of California, San Diego, 1988.

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

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

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

[49] 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.

[50] 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.

[51] 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.

[52] W. Donath and A. Hoffman, "Lower bounds for the partitioning of graphs," *IBM J. of Res. Develop.*, vol. 17, pp. 420–425, Sept. 1973.

[53] K. Hall, "An r-dimensional quadratic placement algorithm," *Management Sci.*, vol. 17, pp. 219–229, Nov. 1970.

[54] C.-K. Cheng and Y.-C. A. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1502–1511, Dec. 1991.

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

[56] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI.* Reading, Massachussetts: Addison-Wesley, 1990.

[57] J. Benkoski and A. J. Strojwas, "The role of timing verification in layout synthesis," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 612–619, 1991.

[58] S. Kim, P. Banerjee, V. Chickermane, and J. H. Patel, "APT: An area-performance-testability driven placement algorithm," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 141–146, 1992.

[59] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510–522, Apr. 1985.

[60] S. Sutanthavibul and E. Shragowitz, "An adaptive timing-driven layout for high speed VLSI," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 90–95, 1990.

[61] W. E. Donath, R. J. Norman, B. K. Agrawal, S. E. Bello, S. Y. Han, J. M. Kurtzberg, P. Lowy, and R. I. McMillan, "Timing driven placement using complete path delay," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 84–89, 1990.

[62] G. Sigl, K. Doll, and F. M. Johannes, "Analytic placement: A linear or a quadratic objective function?," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 427–432, 1991.

[63] M. Hanan, "On Steiner's problem with rectilinear distance," *J. SIAM Appl. Math.*, vol. 14, pp. 255–265, Mar. 1966.

[64] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-10, pp. 356–365, Mar. 1991.

[65] T. Sasao, ed., *Logic Synthesis and Optimization*. Boston, Massachussetts: Kluwer, 1993.

[66] Y.-C. Ju and R. Saleh, "Incremental techniques for the identification of statically sensitizable critical paths," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 541–546, 1991.

[67] Y.-C. Ju and R. Saleh, "Identification of viable paths using binary decision diagrams," in *Proc. IEEE Int. Conf. Computer Design*, pp. 638–641, 1991.

[68] S. W. Cheng, H.-C. Chen, D. H. Du, and A. Lim, "The role of long and short paths in circuit performance optimization," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 543–548, 1992.

[69] H.-C. Chen and D. H.-C. Du, "Path sensitization in critical path problem," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 208–211, 1991.

[70] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[71] G. D. Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 63–73, Jan. 1991.

[72] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 74–84, Jan. 1991.

[73] R. Comerford, "How DEC developed Alpha," *IEEE Spectrum*, pp. 26–31, July 1992.

[74] T. Bell, "Incredible shrinking computers," *IEEE Spectrum*, pp. 37–43, May 1991.

[75] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Lower power CMOS digital design," *IEEE J. Solid-State Circuits*, pp. 473–484, Apr. 1992.

[76] V. Tiwari, P. Ashar, and S. Malik, "Technology mapping for low power," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 74–79, 1993.

[77] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Technology decomposition and mapping targeting low power dissipation," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 68–73, 1993.

[78] B. Lin and H. D. Man, "Lower-power driven technology mapping under timing constraints," in *Proc. IEEE Int. Conf. Computer Design*, pp. 421–427, 1993.

[79] F. N. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 310–323, Feb. 1993.

[80] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Extimation of average switching activity in combinational and sequential circuits," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 253–259, 1992.

[81] R. Burch, F. Najm, P. Yang, and T. N. Trick, "A Monte Carlo approach for power estimation," *IEEE Trans. VLSI*, vol. 1, pp. 63–71, Mar. 1993.

[82] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Efficient estimation of dynamic power consumption under a real delay model," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, pp. 224–228, 1993.

VITA

Wei-Tong Chuang was born in Taichung, Taiwan, in 1964. He received the Bachelor of Science degree from the National Taiwan University in 1986, and the Master of Science degree from the University of Maryland at College Park in December 1989, both in Electrical Engineering. From 1988 to 1989, he was a recepient of the Graduate School Fellowship of the University of Maryland. From January 1990 to January 1994, he was a research assistant at the Coordinated Science Laboratory at the University of Illinois. He is currently a member of technique staff of the AT&T Bell Laboratories at Murray Hill, New Jersey. His research interests include optimization algorithms for CAD, timing analysis, and physical design of VLSI circuits.

PUBLICATIONS

1. W. Chuang, S.S. Sapatnekar, and I.N. Hajj, "A unified algorithm for gate sizing and clock skew optimization to minimize sequential circuit area," in *Proc. IEEE/ACM 1993 International Conference on Computer-Aided Design.*
2. 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 Conference,* 1993.
3. W. Chuang and I.N. Hajj, "Fast mixed-mode simulation for accurate MOS bridging fault detection," in *Proc. IEEE International Symposium on Circuits and Systems,* 1993.
4. T. Lee, W. Chuang, I.N. Hajj, and W.K. Fuchs "Circuit-level dictionaries of CMOS bridging faults," to appear in *Proc. IEEE VLSI Test Symposium,* 1994.

5. W. Chuang, S.S. Sapatnekar, and I.N. Hajj, "Timing and area optimization for standard-cell VLSI circuit design," Tech. Rep. UIUC-ENG-93-2207 DAC-39, Coordinated Science Laboratory, University of Illinois. Submitted to *IEEE Trans. Computer-Aided Design.*

6. W. Chuang and T.S. Huang, "Estimating rotation speed from projections in SAR," in *Proc. IEEE International Conference on Acoustic, Speech, & Signal Processing,* 1992.

7. W. Chuang and T.S. Huang, "An algorithm for estimating rotation speed from projections," *IEEE Seventh Workshop on Multidimensional Signal Processing,* 1991.