1998

Concurrent optimization strategies for high-performance VLSI circuits

Yanbin Jiang
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Concurrent optimization strategies for high-performance VLSI circuits

by

Yanbin Jiang

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering (Microelectronics)

Major Professor: Sachin Sapatnekar

Iowa State University

Ames, Iowa

1998

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Yanbin Jiang

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

DEDICATION

Dedicated to my Parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my heartfelt thanks to my advisor, Professor Sachin S. Sapatnekar, for leading and encouraging me through this research. Without his guidance and detailed advice, it would have been impossible for me to finish this research. I also thank to the help and valuable suggestions I received from the other committee members: Professors Randall Geiger, Marwan Hassoun, Akhilesh Tyagi and David Fernandez-Baca.

I would like to express my sincere appreciation to Professor LiangFang Chao at Iowa State University, Dr. Cyrus Bamji and Dr. Juho Kim at Cadence Design Systems, Inc., Dr. Leon Stok, Dr. Ruchir Puri, Dr. Daniel Brand and Dr. Chandu Visweswariah at IBM.

I would like to thank my colleagues for their enthusiasm and friendship: Naresh Maheshwari, Jian-Feng Shi, Jatan C. Shah, Huibo Hou, Kishore V. Kasamsetty, Min Zhao, Sabita Pilli, Raghuram Madabushi, Tapas Ray, Sridharan Sucheendran, Chung Yen Chang, Vara Varavithya, Jiang Hu, Kaushik Gala and Junsoo Lee.

I am grateful to National Science Foundation, Cadence Design Systerm Inc. Lucent Technologies and Design Automation Conference for funding parts of my research; and IBM for providing me with the opportunity to work for a summer.

I thank my parents and my family for their support.

I thank my wife Xiumei Pu for her understanding.

I thank all the people who gave me encouragement and help during my growing up.

ABSTRACT

In the next generation of VLSI circuits, concurrent optimizations will be essential to achieve the performance challenges. In this dissertation, we present techniques for combining traditional timing optimization techniques to achieve a superior performance.

The method of buffer insertion is used in timing optimization to either increase the driving power of a path in a circuit, or to isolate large capacitive loads that lie on noncritical or less critical paths. The procedure of transistor sizing selects the sizes of transistors within a circuit to achieve a given timing specification. Traditional design techniques perform these two optimizations as independent steps during synthesis, even though they are intimately linked and performing them in alternating steps is liable to lead to suboptimal solutions. The first part of this thesis presents a new approach for unifying transistor sizing with buffer insertion. Our algorithm achieve from 5% to 49% area reduction compared with the results of a standard transistor sizing algorithm.

The next part of the thesis deals with the problem of collapsing gates for technology mapping. Two new techniques are proposed. The first method, the odd-level transistor replacement (OTR) method, performs technology mapping without the restriction of a fixed library size, and maps a circuit to a virtual library of complex static CMOS gates. The second technique, the Static CMOS/PTL method, uses a mix of static CMOS and pass transistor logic (PTL) to realize the circuit, using the relation between PTL and binary decision diagrams. The methods are very efficient and can handle all ISCAS'85 benchmark circuits in minutes. On average, it was found that the OTR method gave 40%, and the Static/PTL gave 50% delay reductions over SIS, with substantial area savings.

Finally, we extend the technology mapping work to interleave it with placement in a single optimization. Conventional methods that perform these steps separately will not be adequate

for next-generation circuits. Our approach presents an integerated solution to this problem, and shows an average of 28.19%, and a maximum of 78.42% improvement in the delay over a method that performs the two optimizations in separate steps.

1 INTRODUCTION

With the growth of the VLSI technology, the VLSI chip feature size has been constantly reduced. From $2um$ in 1985, advances in process technology have reduced feature sizes was decreased to $1um$ in 1990, and to 0.35-0.5um in 1996-1998. This is project to be reduced to about 0.18um in 2001 [88]. Such a continual reduction of the VLSI device feature size has strong impact on the VLSI circuits performance in several ways. First, the device density on a chip grows quadratically with the rate of the feature size reduction. The total number of transistors on a chip has increased from less than 500,000 in 1985 to over 10 millions today. It will reach 64 millions in 2001 [23, 88]. Secondly, the devices are required better performance than before. With the complexity of VLSI circuits increasing and the feature sizes shrinking, it is impossible for designers to design large digital circuits manually, or using designer's intuitional. Therefore, electronic design automation (EDA) tools must be used during the chip design. Due to the requirements of high performance and reduced time to market, the need for more sophisticated computer-aided design (CAD) tools is becoming indispensable.

1.1 Our Research Goal

This dissertation studies issues in both design methodology and in design techniques in VLSI CAD.

In terms of design methodology, a VLSI design is typically developed, analyzed, and refined by a sequence of several different design tools used across a diverse design team in the shortest possible time and at the lowest possible cost. Until recently, the methodology of using separate optimizations in a sequential manner has been adequate. Rapid increases in the complexity of designs and a collection of deep submicron effects have changed the nature of the design process. These changes will soon require new approaches to the design process. This will result

in profound changes in overall design methodology. Specifically, it will be essential to perform concurrent design optimizations to achieve design goals that are increasingly aggressive.

In terms of design techniques, with the advent of the VLSI technology and high-performance designs, the delay of VLSI circuits is a key factor, and timing optimization techniques, i.e., techniques for reducing delay of VLSI circuits, have received much attention in recent years, for example, [32, 41, 65, 93, 96]. The purpose of this research is to develop more sophisticated techniques for timing optimization for VLSI circuits. Here, we only address the timing optimization techniques for the combinational circuits. Sequential circuits may be handled by treating them as many combinational sub-circuits isolated by registers with each combinational sub-circuits being optimized individually.

There are several timing optimization techniques at gate/transistor level circuit design, and we will consider the following methods: (1) Transistor Sizing (2) Buffer Insertion (3) Technology Mapping (4) Placement.

Table 1.1 shows the levels of abstraction in chip design of traditional design flow. This traditional design methodology will not be suitable to the advanced technology.

Semiconductor manufacturers have come to realize that layout-level and logic-level design must be unified, and this has been stated in the National Technology Roadmap for Semiconductors [88]. In particular, system timing management, logic optimization, placement and routing must coexist in a single environment. This dissertation investigates a new design methodology that performs concurrent optimization for improved performance. The techniques developed here use integrated sizing with buffer insertion, dynamic libraries, and combined logic synthesis with physical layout design. Such an integrated approach can achieve better overall results than those achieved by separating these optimizations.

1.2 Terminology

Transistor Sizing:

The usual statement of the problem is as follows:

Assign sizes to the transistors of circuit X so that all its outputs are produced by time T.

Table 1.1 Traditional Abstract Levels of VLSI Design

Design Level	Concerns Addressed
Behavior	Functionality
Functional blocks Linked module abstractions	Resource allocation sequencing, causality
Register-transfer level Clocked register and logic	Testability Timing, synchronization
Gate Level Clocked primitive switches	Implementation with proper digital behavior
Circuit Level	Performance, noise margins
Sticks Level	Layout topology
Mask Geometry	Implementation, yield

The parameter T is called the maximum delay or timing constraint, and the entire process is referred to as transistor sizing of a circuit. The actual delay through the circuit is the delay given a particular assignment of sizes to its transistors, while the size or area of a circuit is the sum of the transistors areas.

The problem can be described in more detail as follows: Given a CMOS circuit C, choose the optimal size S_i of gate G_i, $G_i \in C$, such that the circuit has the minimum area with satisfying the timing constraints.

Buffer Insertion:

Buffer insertion is the procedure of finding the buffer position and buffer sizes for a given circuit to achieve the objective function. Buffer insertion is a common and effective technique to reduce the circuit delay. Buffer can be used to isolate the high critical paths from the noncritical paths, and can also be used as a stronger driver to drive the high critical paths.

Traditional methods have performed transistor sizing and buffer insertion as two independent optimization steps. However, this is suboptimal since it is impossible to know where to insert buffers optimally unless sizing has been performed, and it is impossible to perform optimal sizing unless buffer locations are specified. This thesis presents a unified approach to the problem.

Technology Mapping:

Technology mapping is an optimization phase at the logic synthesis stage that binds the gates in the circuit network to the cells of the specific technology library. Its goal is to achieve the high speed/low power and minimum area.

Gate Collapsing:

The result of traditional technology mapping is greatly impacted by the size of the library. i.e., the types of cells in the library etc.. Thus, traditional technology mapping may result in suboptimal solutions. In addition, the algorithms for covering and matching in the traditional mapping system are highly time consuming.

We propose the idea of global gate collapsing in this work, where the goal is similar to that of technology mapping but the list of permissible gates is not tied to any specific library. The procedure works on a virtual library that is assumed to have all types of cells so that the global gate collapsing technique can have the full flexibility of finding the optimum possible combinations of standard gates in a network, and to generate the optimum set of collapsed complex gates. The input to global gate collapsing comes from the output of technology independent optimization, and the result of the procedure is that it collapses the network into an optimal set of complex gates of the network. This technique can result in optimum solution of minimum circuit delay, minimum circuit area and minimum power dissipation etc. The essential idea of gate collapsing is to begin with a decomposition of the circuit and then to combine or collapse these simple gates into more complex gates. Although there are several delay models for the complex gates [1, 29, 38, 60, 61, 70], in order to get more accurate delay information, we use look-up table method for the delay calculation. In previous techniques, the collapsed gates were constrained to belong to the available cell library. To the best of our best knowledge, there has been no published work on global gate collapsing. Recent work [39] selects the parts of the circuit after technology mapping and remaps them based on virtual library.

1.3 Contributions

The major contributions of this dissertation are:

The idea of combined transistor sizing and buffer insertion into a single optimization has been proposed. We have developed a strategy to realize the idea and obtained performance improvements using this method.

The idea of gate collapsing was proposed and realized using the Odd-level transistor replacement and mixed static CMOS/pass-transistor logic design. The results show the effectiveness, usefulness and promise of the idea.

Several important steps in the VLSI design flow, namely, technology mapping to a dynamic library, merged logic synthesis and physical layout design, and interconnect-conscious physical design, have been combined together and implemented into a new algorithm.

1.4 Organization of this Dissertation

This thesis focuses on the timing optimization techniques of VLSI circuits and the crucial issues of the next generation VLSI CAD tools. Parts of this research have been published in [50, 51, 52, 56]. Parts of this research will be submitted for publication. The remainder of the thesis is organized as follows:

Background: Transistor sizing, buffer insertion and technology mapping (gate collapsing) are the three techniques for timing optimization in our research. In Chapter 2, we present these three techniques and a static timing analyzer (PERT) which is the key part of the timing analysis.

Combined Transistor Sizing and Buffer Insertion: In Chapter 3, we introduce the limitations of performing transistor sizing and buffer insertion separately. We propose our idea of combined transistor sizing and buffer insertion. We also introduce the area and delay modeling used in our work, the type A buffer insertion which is used to drive the critical paths and type B buffer insertion which is used to isolate the highly critical paths from the noncritical

paths and the outline of algorithms for combining the transistor sizing and buffer insertion. The encouraging results verified our original idea.

Gate Collapsing and Mixed Static CMOS and Pass-transistor Design: The traditional procedure of technology mapping has its limitation. i.e., the mapped result entirely depends on the library size. This limitation makes it impossible for the circuit designers to explore the entire design space, and therefore, it results in a suboptimal solution of the circuit design. In order to allow the circuit designers to explore the whole design space, we propose the idea of gate collapsing. Gate collapsing has the same goal as that of technology mapping, but without the library restriction. It assumes that it works on a virtual library. It can generate any types of complex gate. This property allows the circuit designers to explore the whole design space and results in the optimal solution of the design. In Chapter 4. we present two methods to realize the gate collapsing idea, one is odd-level transistor replacement which is based on the topology of the given circuit, the other is mixed static CMOS and pass-transistor design which is based on Boolean function of the circuit. For the mixed static CMOS and pass-transistor design method, we use BDD to represent the Boolean functions of the circuit and then to use pass transistor to implement the circuit. Gate collapsing by using fully static CMOS and mixed static CMOS/pass-transistor design makes it easier to achieve the high performance of the circuits.

A New Idea of Combining Placement with Technology Mapping: A new idea of combining placement with technology mapping is presented in the Chapter 5. With the semiconductor technology shrinks down to the deep sub-micron region, many new problems need to be solved, such as dynamic library, interconnect wire issue and integrated EDA tools of different design level etc.. Parts of our work address the key issues for the next generatiton VLSI CAD tools. We proposed our ideas to deal with the problems for tomorrow's CAD.

Concusion and Future Work: We conclude our work and point out the future research directions in Chapter 6.

2 BACKGROUND

2.1 Transistor Sizing

The designer of a VLSI circuit must consider not only functional correctness but timing behavior. Usually, there is some specification of how quickly the circuit must produce its output. Once a schematic, transistor-level description of the circuit is produced, it must be forced to meet the delay constraint. This is done by assigning sizes to the transistors. Increasing the size of transistors in a VLSI circuit tends to decrease the delay through the circuit, but at the cost of increasing its area. While transistor area is usually only a small component of total chip area, this is only because transistor sizes are usually "reasonable". Minimizing delay can result in huge transistors beyond a certain point, however, larger transistors actually increase delay. Actual minimization of the circuit's delay is usually not required. Instead, the delay must be reduced to meet the specified constraint. Given a delay model, some expression for maximum delay through the circuit can be derived. It is thus possible to view the problem as one of constrained minimization:

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

The property of the transistor sizing problem is: the objective function is quite simple, but the constraint is both highly non-linear and expensive to compute – even finding a feasible solution is very difficult. The major difficulty is that circuit delay is the maximum path delay, and there are a combinatoric number of paths through the circuit [76]. Circuit designers avoid considering all these paths by using intuition and heuristics. After some initial configuration is chosen, simulation and timing analyses are run on the circuit to find its critical paths - the

paths through the circuit whose delay exceed the constraints - and the designers reduce these paths' delay sufficiently. Now some other paths may be critical, so the process iterates until the maximum delay through the circuit is satisfactory.

Transistor sizing is also called repowering, i.e., changing the sizes of various transistors. It is a powerful technique that has the minimum impact on the layout. The size of a transistor is measured in terms of its channel width, since the channel lengths of MOS transistors in a digital circuit are generally uniform. While a combinational CMOS circuit with minimum-sized transistors has a small area, its 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. Hence making the circuit faster usually entails the penalty of increased circuit area. The well-studied optimization problem that deals with this area-delay tradeoff is known as the transistor sizing problem.

Figure 2.1 shows a chain of three CMOS inverters. For simplicity, we assume that the size of G2 is $w2$ and the sizes of other gates are fixed. Let D be the total delay through the three gates and $d2$ be the delay of G2. Consider the effect of increasing $w2$. This causes the magnitude of the output current of G2 to increase and $d2$ to decrease monotonically. However, increasing $w2$ also increases the capacitive load on the output of G1, thus slowing down the output transition of G1. Beyond a certain point, $w2 = A$, the total delay D begins to increase with respect to $w2$, which shows the nonmonotonicity of the delay-area relationship for the circuit.

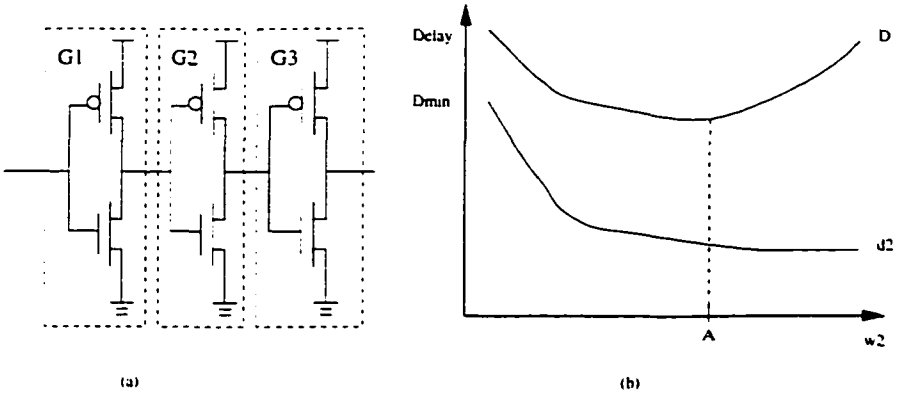


Figure 2.1 A transistor sizing example

Transistor sizing is a conventional technique for reducing the circuit delay [34, 47, 53, 54, 67, 68, 72]. A larger driver at the source has a stronger driving capability (or equivalently, smaller effective driver resistance), reducing the circuit delay. But a larger driver also means a heavier load (large sink capacitance) to the the previous stage and thus increases its delay. The transistor sizing problem is to determine the optimal size of each driver to minimize the circuit delay.

The transistor sizing problem has been approached using both sensitivity based methods and mathematical optimization based methods [76].

2.1.1 Sensitivity Based Transistor Sizing

Fishburn and Dunlop [37] studied the transistor sizing problems for synchronous MOS circuits. Let $x_1, \cdots x_i \cdots x_n$ be the transistor sizes, A the total area of transistors and T the clock period. If K is a positive constant, there are three forms for the transistor sizing problem as follows:

1 Minimize A subject to the constraint $T \leq K$

2 Minimize T subject to the constraint $A \leq K$

3 Minimize AT^K

Let a transistor be modeled by the switch level model, the gate, source and drain capacitance are all proportional to the transistor size, and the effective resistance is inversely proportional to it. A CMOS gate will be modeled by a distributed RC network. The Elmore delay is used to compute the worst-case delay of the gate, which is the delay through the highest resistive path in the RC network. The delay of a PI-PO is the sum of delays through all gates in the path. It is not difficult to verify that the delay of a PI-PO path can be written into this form:

$$\sum a_{ij}\frac{x_i}{x_j} + \sum \frac{b_i}{x_i} \tag{2.2}$$

where the a_{ij} and b_i are nonnegative constants. The above equation and the area $A = \sum x_i$ are posynomials and the transistor sizing problem of the three forms are all posynomials programs.

Even though posynomials programming methods can be used to optimally solve the three forms of the transistor sizing problem, it is computationally expensive to be used for an entire circuit. Thus the transistor sizing tool TILOS (TImed Logic Synthesizer) was developed to minimize A subject to $T \leq K$ based on the following scheme: First, the minimal size is assigned to all transistors. Then timing analysis is performed to find the critical delay T. If T is larger than K, the sensitivities of all transistors related to the critical path will be computed. The sensitivity is defined as the delay reduction due to per transistor size increment. The size of the transistor with the largest sensitivity will be multiplied by a user defined factor (BUMPSIZE) and the algorithms goes to the timing analysis again. This procedure will be terminated when the timing specification is satisfied or there is no improvement.

2.1.2 Mathematical Programming Based Transistor Sizing

Studies have been done to formulate the transistor sizing problem as mathematical programming problems to obtain an optimal solution. Method in [5] solves the transistor sizing problem with linear programming. [6] solves the sizing problem based on the piecewise linear simulation. Methods in [20, 46, 69] formulate the transistor sizing problem as nonlinear programs and solve them by the method of Lagrangian multipliers. Methods in [28, 48, 18] apply the following two-step iterations. First, the delay budget is distributed to each gate: Then, the transistor in each gate are sized optimally to satisfy the time budget. Later, a two-phase algorithm is presented in [90] to minimize the circuit area under timing constraints: first, TILOS is used to generate an initial solution: then, a mathematic optimization is formulated and solved by using feasible directions to find the optimal solution. Sapatnekar [85, 83, 84] develops a transistor sizing tool iCONTRAST to minimize the circuit area under timing constraints. It employs the analytical delay model developed in [45] which can consider the waveform slope of input signals to transistors. Under the delay model, the transistor sizing problem is a posynomials program that can be transformed into a convex program and the convex programming method [97] is implemented to solve the transformed problem.

2.2 Buffer Insertion

The process of inserting buffers at the output of some gates in the circuit can be used for timing optimization. Buffer insertion is a technique that is used either to increase the driving power of a path in a circuit, or to isolate the most critical paths from the noncritical paths so that buffer insertion can achieve fanout optimization of those gates. Buffer positions and buffer sizes should be considered in the buffer insertion process for optimal solutions.

The buffer insertion problem can be stated as: Given a CMOS circuit C, find the positions and sizes of buffers such that the circuit has the minimum area with satisfying the timing constraints.

Existing procedures for buffer optimization consider the following issues:

Buffer Position: In [92], the input to the algorithm is a combinational circuit consisting of gates implemented in a target technology. If the circuit does not meet the timing constraints, the algorithm iteratively reduces the violation in the constraints by applying fanout correction at multiple fanout gates. The fanout correction procedure uses buffers to isolate the high critical paths from noncritical paths. For the purpose of applying fanout correction on the circuit, gates with a large number of fanouts on the critical path are good candidates.

Buffer Sizing: The work in [59] builds a fanout chain of cascaded buffers which are in the library. The procedure enumerates all the possibilities for selecting the optimal buffers to get the optimal solution. The algorithm in [45] derives a timing model for CMOS combinational logic based on an analytical solution for the CMOS inverter output responsing to an input ramp. This model is then used to calculate the optimal sizes of each buffer in a chain.

2.3 Technology Mapping

Technology mapping as a cornerstone in logic synthesis has been well studied in the past. Existing technology mapping techniques can be classified into four categories: rule-based mapping [43], graph matching [55], direct mapping [63] and functional mapping [66].

Rule-based mapping [43] incorporates logic and circuit level manipulation and optimization techniques. The method uses library which describes alternative circuit implementations

in a rule (if - then form). Competing alternatives are evaluated by implementing each in turn.

Graph mapping [55] treats a circuit as a DAG (directed acyclic graph) in which the labels of the vertices in the graph correspond to the Boolean operators NAND. NOT etc., and the edges are directed from outputs to inputs. The program takes technology-independent description of a combinational circuit (G_c) and a list of patterns describing both the cells in the technology and local transformations. The algorithm creates a technology bound circuit (G_t) by partitioning the circuit into a forest of trees and then uses a tree pattern matching algorithm to match the individual tree.

Direct mapping [63] uses an approach that directly translates each node or function of a multilevel network into a gate of a target library. Direct mapping requires that all the nodes of the optimized Boolean network represent implementable gates. After all nodes are made implementable, each of the sum of products/products of sums nodes of the networks is translated into a gate from the library.

Functional mapping [66] provides an approach where matchings are recognized by means of Boolean operations. A Boolean match can be determined by verifying that there exists a matching of the input variables such that the target function f and the library function g are a tautology.

Technology mapping techniques have been employed over the past decade for the circuit optimization. It allows translation of a technology-independent logic description into a library-specific (technology-dependent) implementation. In the matching/covering step, each subject graph is implemented using elements from the technology library. All possible matches to library elements are found for each subnetwork within each subject graph. An optimal set of matching library elements is then selected from sets of possible matches to realize the network from elements of the library. Of all aspects of algorithmic technology mapping, the matching/covering step has received the most attention over the years [91, 96], since it has the biggest impact on the quality of the results and is potentially very expensive computationally. Matching algorithm can be classified as either structural or Boolean. Structural matching was first proposed by Keutzer, and has since been used in subsequent technology mappers,

such as MIS. With structural matching, each library is decomposed into a graph composed of base functions, called pattern graphs. A library element may have more than one pattern graph associated with it, representing the multiple possible decompositions of that element to allow the covering routine to find the best matching element later on. A tree-based matching routine can then recursively determine which pattern graphs are isomorphic to a subgraph of the subject graph. The subgraph is then annotated with the set of possible matches along with their costs. Dynamic programming techniques can then be used to find the optimum cover of the subject graph with elements from the library in linear time. Although structural techniques are computationally feasible, they do not exploit all properties of Boolean functions - such as *don'tcare* information-during library binding, leading to solutions of inferior quality. In contrast, by using Boolean techniques in the matching/covering phase. *Don'tcare* information can be exploited to find a better cover for the network at a cost of increased computational complexity. With Boolean matching, each library element's function is represented by an ordered binary decision diagram(OBDD), called a pattern function. Each subnetwork of the subject graph is also represented by its OBDD, called a cluster function. During the matching phase, each cluster function is tested for equivalence against the pattern function in the library by fixing the variable order in one OBDD and then permuting the order of the variables in the other OBDD until a match is found.

2.4 Placement

In our work, we use GORDIAN [58] as our placement engine. Figure 2.2 shows the overview placement precedure. It formulates the placement as $min \sum (x_i - x_j)^2 + (y_i - y_j)^2$: x_i and y_i are the coordinates of cell i. The objective function can be written in matrix form

$$\Phi(\mathbf{x}, \mathbf{y}) = \frac{1}{2} x^T C x - d_x^T x + \frac{1}{2} y^T C y + d_y^T y; \tag{2.3}$$

The basic idea of it is that at the top optimization level ($l = 0$), all m modules to be placed belong to the root region which covers the whole placement area available to the modules. Then it solves the quadratic programming problem. According to the module coordinates, if there is overlap, then it partitions the region into two parts. The sums of the module areas of

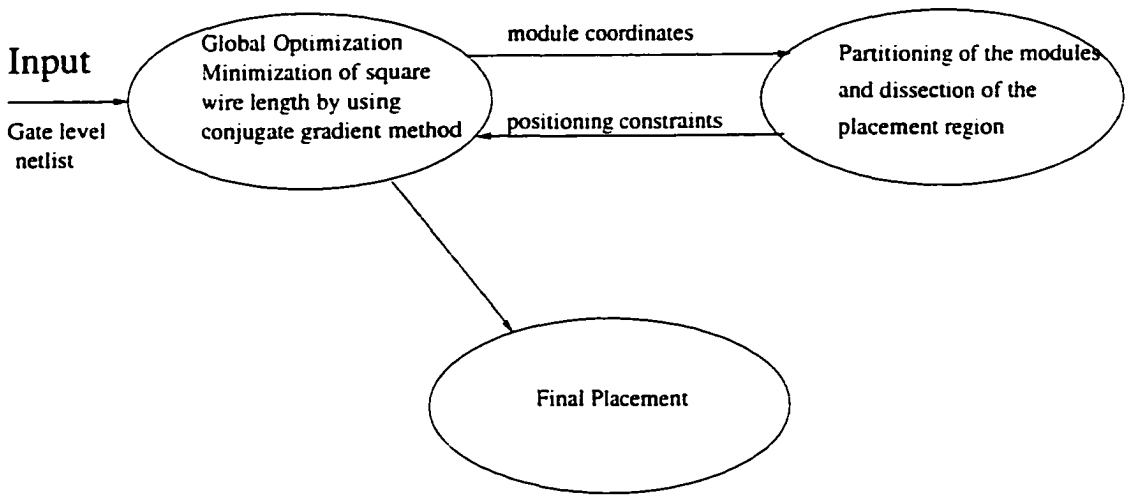


Figure 2.2 Data flow in the placement procedure

both subregions determine the dissection of the original region. Then it puts the constraints of the region followed by solving the quadratic programming problem again. At the lth level of optimization, the placement area is divided into $q \leq 2$ regions $\rho \in R^{(l)}$, where $R^{(l)}$ is the index set of the regions on level l. The centers (u_ρ, v_ρ) of these regions impose constraints on the global placement of the modules: $\mathbf{A}^{(l)}\mathbf{x} = \mathbf{u}^{(l)}$; such that the area weighted mean value of the coordinates of modules $\mathbf{u}$, i.e., the center of gravity, corresponds to the center of region ρ. The matrix $\mathbf{A}^{(l)}$ is determined by the following way. The entries $a_{\rho u}$ of the $< q \times m > -matrix\mathbf{A}^{(l)}$ depend on which module belongs to which region ρ.

$a_{\rho u} = \begin{cases} \frac{F_u}{\sum F_u} & \text{if } u \in \text{region } \rho \\ 0 & \text{otherwise} \end{cases}$ F_u is the area of the module u. Figure 2.3 shows the build-up of matrix $\mathbf{A}$.

The placement problem now becomes solving

$$\Phi(\mathbf{x},\mathbf{y}) = \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{C}\mathbf{x} - \mathbf{d}_{\mathbf{x}}^{\mathbf{T}}\mathbf{x} + \frac{1}{2}\mathbf{y}^{\mathbf{T}}\mathbf{C}\mathbf{y} + \mathbf{d}_{\mathbf{y}}^{\mathbf{T}}\mathbf{y}; \mathbf{subject\ \ to}\mathbf{A}^{(l)}\mathbf{x} = \mathbf{u}^{(l)}; \tag{2.4}$$

It divides $\mathbf{A}$ into two parts, $\mathbf{A} < \mathbf{q} \times \mathbf{m} >= \begin{bmatrix} \mathbf{D}_{<\mathbf{q}\times\mathbf{q}>} \\ \mathbf{B}_{<\mathbf{q}\times\mathbf{m}-\mathbf{q}>} \end{bmatrix}$. $\mathbf{D}$ is a diagonal matrix made of non zero entries of $\mathbf{A}$. Correspondingly, it also divides $\mathbf{x}$ into $\mathbf{x} = \begin{bmatrix} \mathbf{x}_{\mathbf{d}<\mathbf{q}>} \\ \mathbf{x}_{\mathbf{i}<\mathbf{m}-\mathbf{q}>} \end{bmatrix}$, m is the total number of the modules need to be placed, q is the region number, so

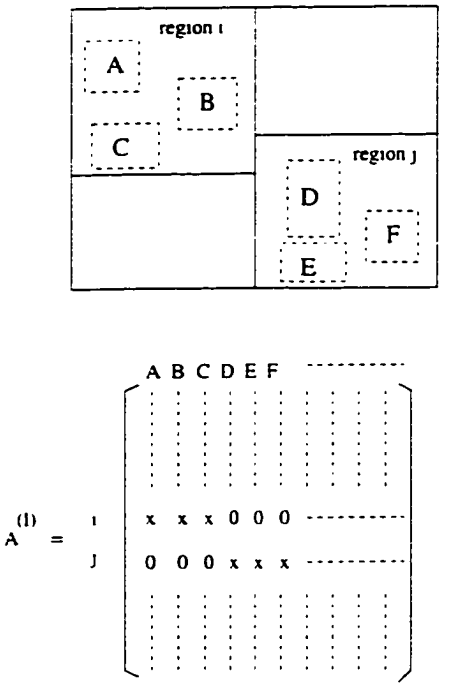


Figure 2.3 The constraints for global placement

$$\mathbf{Ax} = \mathbf{u} \Rightarrow$$
$$[\mathbf{DB}]\begin{bmatrix} \mathbf{x_d} \\ \mathbf{x_i} \end{bmatrix} = \mathbf{u} \Rightarrow$$
$$\mathbf{x_d} = -\mathbf{D^{-1}Bx_i} + \mathbf{D^{-1}u} \Rightarrow$$
$$\mathbf{x} = \begin{bmatrix} \mathbf{x_d}, \\ \mathbf{x_i} \end{bmatrix} = \begin{bmatrix} -\mathbf{D^{-1}B} \\ \mathbf{I} \end{bmatrix} \mathbf{x_i} + \begin{bmatrix} \mathbf{D^{-1}u} \\ \mathbf{0} \end{bmatrix};$$

it denotes

$$\mathbf{Z} = \begin{bmatrix} -\mathbf{D^{-1}B} \\ \mathbf{I} \end{bmatrix} \text{and } \mathbf{x_0} = \begin{bmatrix} \mathbf{D^{-1}u} \\ \mathbf{0} \end{bmatrix}.$$

it needs to solve

$$\Phi(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\mathbf{x^T Cx} + \mathbf{d_x^T x} + \frac{1}{2}\mathbf{y^T Cy} + \mathbf{d_y^T y}; \tag{2.5}$$

as $\Phi(\mathbf{x}, \mathbf{y}) = \Psi(\mathbf{x}) + \Omega(\mathbf{y})$. It present of solving $\Psi(\mathbf{x})$ for an example. So the linearly constrained quadratic programming LQP

$$\mathbf{min}\Psi(\mathbf{x}) = \frac{1}{2}\mathbf{x^T Cx} + \mathbf{d_x^T} | (\mathbf{A^{(l)}x} = \mathbf{u^{(l)}} \tag{2.6}$$

is transfered to unconstrained quadratic programming UQP:

$$\mathbf{min}\Psi(\mathbf{x_i}) = \frac{1}{2}\mathbf{x_i^T Z^T C x_i} + \mathbf{c^T x_i} \tag{2.7}$$

with $\mathbf{c^T} = (\mathbf{Cx_0} + \mathbf{d})^\mathbf{T}\mathbf{Z}$. it sets $\Psi(\mathbf{x}) = \mathbf{Z^T CZ x_i} + \mathbf{c^T} = \mathbf{0}$; so

$$\mathbf{Z^T CZ x_i} = -\mathbf{c^T}: \tag{2.8}$$

Thus to determine the global minimum solution of Equation 2.4. simply means to solve the Equation 2.8 system. After getting $\mathbf{x_i}$. $\mathbf{x}$ can be calculate as $\mathbf{x} = \mathbf{Zx_i} + \mathbf{x_0}$.

It is difficult to solve Equation 2.8 by using direct solver and iterative methods. For this class of problems, the well-suited solution is the conjugate-gradient method [78].

The attractiveness of solving $\mathbf{Z^T CZ x_i} = -\mathbf{c^T}$ by conjugate-gradient-method is that we do not need to multiply $\mathbf{Z^T CZ}$ explicitly. The solution is carried out by generating a succession of search directions $\mathbf{p_k}$ and improved minimizers $\overline{\mathbf{x_k}}$. At each stage quantity α_k is found that minimizes $f(\mathbf{x_k} + \alpha_\mathbf{k}\mathbf{p_k})$. (where $f(\mathbf{x}) = \frac{1}{2}\mathbf{x^T Cx} + \mathbf{d_x^T x}$) and $\mathbf{x_{k+1}}$ is set equal to the new point $\mathbf{x_k} + \alpha\mathbf{p_k}$. The $\mathbf{P_k}$ and $\mathbf{X_k}$ are built up in such a way that $\mathbf{x_{k+1}}$ is also the minimizer of function f over the whole vector space of directions already taken, $(\mathbf{p_1}, \mathbf{p_2}, ...\mathbf{p_k})$. After some iterations, you will arrive at the minimizer over the entire vector space.

The method builds the following recurrence.

$\alpha_k = \frac{\overline{\mathbf{r_k}}\mathbf{r_k}}{\overline{\mathbf{p_k}}\mathbf{Z^T CZ p_k}}$

$\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_\mathbf{k}\mathbf{Z^T CZ p_k}$

$\overline{\mathbf{r_{k+1}}} = \overline{\mathbf{r_k}} - \alpha_\mathbf{k}(\mathbf{Z^T CZ})^\mathbf{T}\overline{\mathbf{p_k}}$

$\beta_k = \frac{\overline{\mathbf{r_{k+1}}}\mathbf{r_{k+1}}}{\overline{\mathbf{r_k}}\mathbf{r_k}}$

$\mathbf{p_{k+1}} = \mathbf{r_k} + \beta_\mathbf{k}\mathbf{p_k}$

$\overline{\mathbf{p_{k+1}}} = \overline{\mathbf{r_k}} + \beta_\mathbf{k}\overline{\mathbf{p_k}}$

$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_\mathbf{k}\mathbf{p_k}$

If $\frac{|\mathbf{Z^T CZX} - \mathbf{c^T}|}{|\mathbf{c^T}|} < \epsilon$, the method stops.

The pseudo code is as followings.

k=0, $x_1 = 0$

while $\frac{|\mathbf{Z^T CZX} - \mathbf{c^T}|}{|\mathbf{c^T}|} > \epsilon$

```
k = k + 1
if k = 1
    r_1 = -c^T - Z^T C Z x_1
    r̄_1 = r_1
    p_1 = r_1 / (Z^T C Z)
    p̄_1 = r̄_1 / (Z^T C Z)
else
    β_k = (r̄_{k+1} r_{k+1}) / (r̄_k r_k)
    p_k = r_{k-1} + β_{k-1} p_{k-1}
    α_k = (r̄_k r_k) / (p_k Z^T C Z p_k)
    r_k = r_{k-1} - α_{k-1} Z^T C Z p_{k-1}
    r̄_k = r̄_{k-1} - α_{k_1} (Z^T C Z)^T p̄_{k_1}
    x_k = x_{k-1} + α_{k-1} p_{k-1}
end
x = x_k
end
```

2.5 Static Timing Analyzer

The purpose of Static Timing Analyzer is to analyze the circuit delay and to find the critical path. We develop a Static Timing Analyzer according to PERT (Program Evaluation and Review Technique) [57]. PERT is a method to determine the delay of a circuit, given the individual gate delays. The procedure is illustrated by means of a simple example. Consider Figure 2.4, where each box represents a gate. The number within the box represents the delay associated with it. We assume that the worst case arrival time for transition at any primary input, i.e., at the inputs A,B,C, and D is 0.

A component is ready for evaluation when the signal arrival time information is arrived for all of its inputs. Initially, since signal arrival times are known only at the primary inputs, only those components that are fed solely by primary inputs are ready for processing. In the example, components A,B,C and D are ready. These are placed in a queue and are scheduled to be processed.

In the iterative process, the component at the head of the queue is scheduled for processing. Each processing step consists of

- Finding the maximum of all worst case arrival times of inputs to the component.
- Adding the delay of the component to the latest arriving input time, to obtain the worst case output transition.
- Checking all of the components that are the current component fanouts, to find out whether they are ready for processing. If so, the component is added to the tail of the queue.

The iterations end when the queue is empty.

```
Outline of Algorithm
                put all primary input into a queue
                for each gate g in the circuit
                ready_for_queuing = 0;
                end for
                 while(queue is not empty)
                 {
                  qh = the head of the queue
                  calculate qh's delay
                  for each fanout gate g of qh
                   ready_for_queuing = ready_for_queuing + 1;
                   if(read_for_queuing eq to input numbers)
                   put g into the tail of the queue
                  end for
                 (remove the head of the queue.)
                 the head of the queue = the next entry of the the queue head
                 }
```

In the example shown in Figure 2.4, the algorithm is executed as follows:

- Step 1. A, B, C and D are placed in the queue.
- Step 2. A is scheduled. The latest input transition for A is 0, the delay of A is 1; hence, the latest output transition for A is at time (0+1) = 1. No additional components are ready for processing.

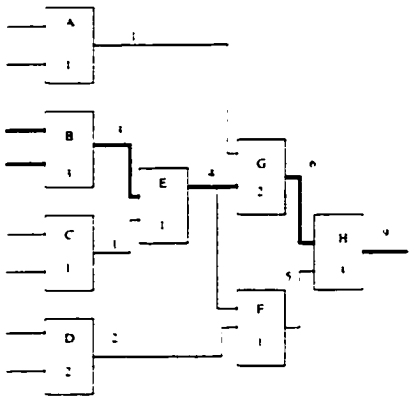


Figure 2.4 PERT

- Step 3. B is scheduled. The latest output transition is at time (0+3) = 3. No additional components are ready for processing.
- Step 4. C is scheduled. The latest output transition is at time(0+1) = 1. E is now ready for processing and is placed at the tail of the queue.
- Step 5. D is scheduled. Worst case output transition is at (0+2) = 2.
- Step 6. E is scheduled. Latest output transition is at (3+1)=4. F and G are ready to be processed and are added to the tail of the queue.
- Step 7. F is scheduled. Worst case output transition is at (4+1) = 5.
- Step 8. G is scheduled. Worst case output transition is at (4+2) = 6. H is now ready and is added to the tail of the queue.
- Step 9. H is scheduled. Worst case output transition is at (6+3) = 9. The queue is empty and the algorithm terminated.

The critical path, defined as the path between an input and output with the maximum delay, can now easily be found by using a traceback method. Beginning at a component whose output is a primary output with the latest transition time, the latest arriving input to this component is found. The process is repeated recursively until a primary input is reached. In the Figure 2.4, the critical path is B-E-G-H, shown in bold lines.

We use the Elmore delay model for our transistor sizing and buffer insertion. The motivation for the definition of Elmore delay is as follows:

Figure 2.5 illustrates the output waveform, $e(t)$, of an RC network, in response to a unit step excitation at its input. The delay time should be measured from t=0, when the input step transient voltage reaches the 50% point. Elmore suggested that the center of area of the region under the curve $e'(t)$, shown in Figure 2.5(b), would serve as a reasonable estimate to the delay, i.e.,

$$T_d = \int_0^\infty te'(t)dt \tag{2.9}$$

Moreover, the quantity T_d is also the area above the step response, as shown in Figure 2.5 (c). This can be seen by performing integration by parts, as shown below:

$$T_d = \int_0^\infty te'(t)dt = \int_0^\infty [1 - e(t)]dt - t[1 - e(t)]\Big|_0^\infty = \int_0^\infty [1 - e(t)]dt.$$

Here, we make use of the fact that $\lim_{t\to\infty} t[1 - e(t)] = 0$, since $e(t) \to 1$ exponentially as $t \to \infty$.

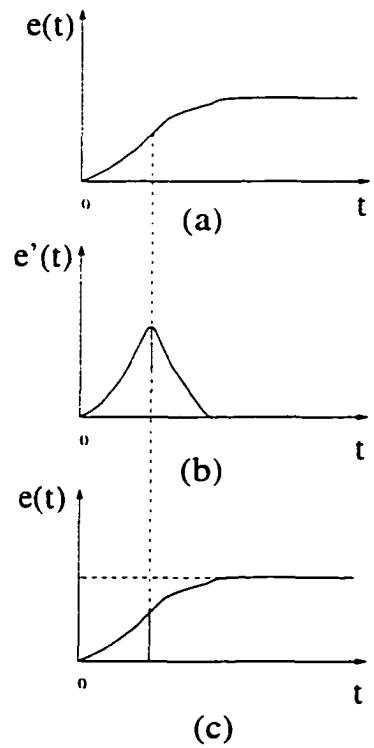


Figure 2.5 Elmore delay definition

For the special case of a RC tree, the Elmore delay from input to node i in the RC tree is given by the expression [80]:

$$T_{di} = \sum_{j=0}^{n} R_{ij} C_j. \tag{2.10}$$

where C_j is the grounded capacitor at node j, R_{ij} is the common resistance of the path from input to node i and the path from input to node j.

For example, for the RC tree shown in Figure 2.6, the delay T_{dn7} to node 7 is given by $T_{dn7} = R1 \cdot C1 + R1 \cdot C2 + R1 \cdot C3 + R1 \cdot C4 + R1 \cdot C5 + (R1 + R6) \cdot C6 + (R1 + R6 + R7) \cdot C7 + (R1 + R6 + R7) \cdot C8$.

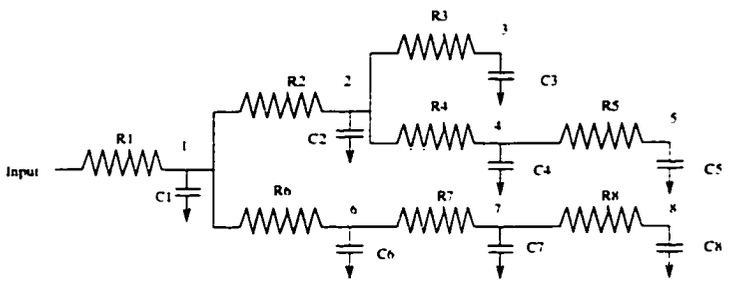


Figure 2.6 RC tree

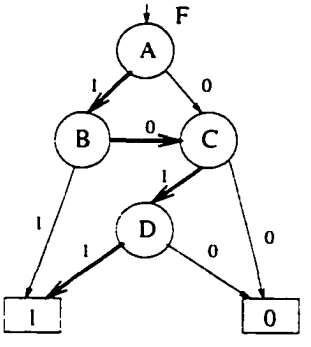
2.6 Binary Decision Diagram

The manipulation of Boolean function is a fundamental part of technology mapping and we use binary decision diagrams (BDDs) in our work on global gate collapsing. The performance of the technology mapping systems greatly depends on the efficiency with which Boolean functions are manipulated. A good data structure is key to efficient Boolean function manipulation. Because classical methods are impractical for large-scale problems, it has been necessary to develop an efficient method for representing practical Boolean functions. The basic concept of Binary Decision Diagrams (BDDs), which are graphic representations of Boolean functions, was introduced in [2], and efficient methods for manipulating BDDs were developed in [11]. BDDs have attracted the attention of many researchers because of their suitability for representing Boolean functions. One attractive feature of BDDs is known that many practical functions can be represented by BDDs of feasible sizes.

A Binary Decision Diagram (BDD) is a directed acyclic graph with two terminal nodes called the 0-terminal node and 1-terminal node, which represent the Boolean values 0 and 1, respectively. Each nonterminal node has an index to identify an input variable of the Boolean function and has two outgoing edges called the 0-edge and 1-edge, corresponding to an assignment of the variables to Boolean values 0 and 1, respectively.

Figure 2.7 shows the BDD representation of the Boolean function F = AB + CD. For the assignment (ABCD) = (1011), the Boolean value of the function can be evaluated by traversing the BDD along the darkened edges.

In many cases, BDD are generated as the results of logic operations and techniques for combining BDDs under various logic operations are described in [11, 13].



F = AB+CD

Figure 2.7 BDD

3 COMBINED TRANSISTOR SIZING AND BUFFER INSERTION

While a combinational CMOS circuit with minimum-sized transistors has a small area, its 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 well-studied optimization problem that deals with this area-delay tradeoff is known as the sizing problem [37, 85, 9] and it is often formulated as

$$\begin{aligned} &\text{minimize} && Area \\ &\text{subject to} && Delay \leq T_{spec} \end{aligned} \tag{3.1}$$

In some formulations, it is the power that is minimized instead of the area. For edge-triggered circuits, we need consider only one combinational subcircuit at a time, minimizing its area while meeting the timing requirements that state that the delay of each combinational segment should satisfy the clock period. Therefore, the problem of sizing even a very large circuit can be decomposed into individual problems of sizing individual combinational blocks to meet the clock period, and the problem complexity is considerably reduced. For the remainder of this chapter, we will therefore assume that the circuit is purely combinational.

For a given combinational circuit, the nature of the area-delay tradeoff curve for gate sizing is as shown in Figure 3.1. Typically, a small amount of sizing is adequate to reduce the delay corresponding to the unsized circuit, $d_{unsized}$, in order to meet a loose delay specification. However, as the specification is tightened, the circuit has to be sized by greater degrees, until we reach the knee of the curve where it must be sized tremendously to achieve further delay reduction. Further, it is impossible to reduce the delay of a circuit indefinitely through sizing, and there is a minimum achievable delay, d_{min}, that cannot be bettered through sizing.

Note that gate sizing does not change the topology of the circuit, but merely changes the

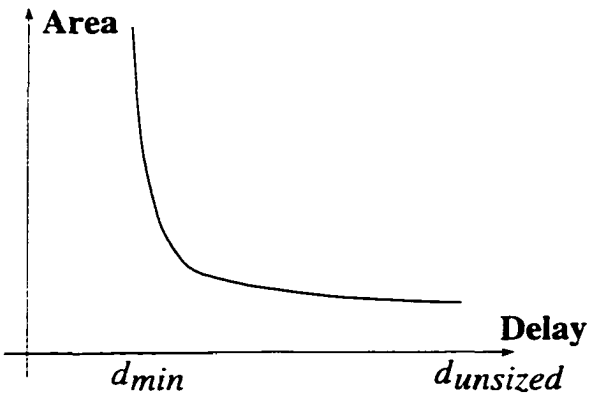


Figure 3.1 Area-delay curve for sizing

sizes of individual transistors within gates. We note that some gates in a circuit can be sized excessively because of the large loads that they drive. The appropriate insertion of buffers in a circuit can be used to prevent excessive sizing while meeting delay specifications. In fact, as we will see, buffer insertion in conjunction with sizing often permits greater circuit delay reductions than sizing alone.

Traditionally, gate sizing and buffer insertion (the "fanout problem") [92, 7] have been carried out separately and at different stages of the design process[1]. However, as sizing changes the capacitances driven by various gates, the locations of high-capacitance nodes are accurately established only during sizing, and any optimizations performed before sizing are necessarily based only on educated guesses. Therefore, it is useful to combine the two optimizations into a single step, and this is the objective of this research.

In this chapter, we first present the delay model used here, and then list the situations in which it is advantageous to insert buffers. Next, we present an algorithm to combine sizing with buffer insertion, and show that the application of these two transformations in unison can provide significant benefits.

[1]The fanout problem, however, only tackles what we will later refer to as Type B buffer insertion.

3.1 Delay and Area Modeling

3.1.1 Transistor Level Modeling

We first show how an n-transistor of width $w_{n,i}$ is modeled by a set of capacitances and resistors. A p-transistor of width $w_{p,i}$ is similarly modeled. Since all the transistors are set to minimum length, the capacitances can be modeled in terms of only the transistor widths. For an n-transistor, we can write the source/drain capacitance $C_{sdn_i} = C_{d,n_1} \cdot w_{n,i} + C_{d,n_2}$, and the gate capacitance as $C_{gn_i} = C_{g,n_1} \cdot w_{n,i} + C_{g,n_2}$, where $C_{d,n_1}, C_{d,n_2}, C_{g,n_1}$ and C_{g,n_2} are constants. The on-resistance, $R_{i,n}$, of an n transistor is given by $R_{i_n} = \frac{R_n}{w_{n,i}}$. As in previous work (for example, [37, 85]), the circuit area is modeled as the sum of all transistor sizes.

At the gate level, each gate G_i is modeled by an equivalent inverter, parameterized with all n-($p-$) transistor sizes set to $w_{n,i}$ ($w_{p,i}$). In this implementation, only static CMOS gates are considered. All transistors of the same type in a gate are assumed to have a uniform size. The ideas presented in this work are also applicable to the case where every transistor is allowed to have a different size. The pull-up (pull-down) structure is represented by an equivalent inverter with a p-transistor (n-transistor) size of $S_{p,i}$ ($S_{n,i}$) that corresponds to the worst-case situation: this number is referred to as the gate size. The relation between the gate sizes in the equivalent inverter and transistor widths in the gate can easily be computed for various type of gates. For example, for a k-input NAND gate, $S_{n,i} = w_{n,i}/k$, $S_{p,i} = w_{p,i}$ [2].

The capacitance loading, C_L, of gate G_i can be calculated from the transistor sizes of its fanouts as follows:

$$C_L = \sum_{j \in fanout_i} (C_{gn_j} + C_{gp_j}) + C_{intrinsic} + C_{wire} \tag{3.2}$$

where $C_{intrinsic}$ corresponds to the source and drain capacitance connected to the output node of G_i. The wire capacitance values are based on the placement.

[2] Notice that $S_{p,i}$ is $w_{p,i}$ (and not $k \cdot w_{p,i}$) since in the worst case, only one of the k transistors in parallel will be on.

3.1.2 Delay Computations

We first demonstrate the calculation of the step delay, i.e., the delay under the assumption that the input to each gate is a step transition with zero transition time. Next, we will show how this assumption is relaxed to allow for the realistic case where nonzero transition times are possible.

The Elmore fall step delay, t_{f_i}, of gate G_i can then be obtained from C_L and $S_{n,i}$ as [36, 80]

$$t_{f_i,\text{step}} = \frac{R_n \cdot C_L}{S_{n,i}}. \tag{3.3}$$

The rise delay is similarly obtained as $t_{r_i,\text{step}} = \frac{R_p \cdot C_L}{S_{p,i}}$.

To allow for the effect of nonstep input transitions, we use the inverter delay model presented in [49]. The effect of the input-to-output coupling capacitance and input slope effects are considered in this model. Consider the CMOS inverter structure driving a load C_L, as shown in Figure 3.2, where C_M is the coupling capacitance between the input and the output nodes. When the applied input is the ramp

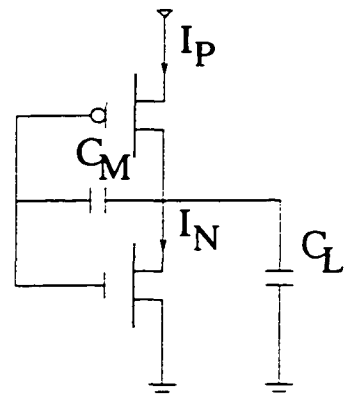


Figure 3.2 CMOS inverter structure

$$V_{in} = \begin{cases} 0 & t \le 0 \\ \frac{V_{DD}}{\tau} t & 0 \le t \le \tau \\ V_{DD} & t \ge \tau \end{cases}, \tag{3.4}$$

where τ is the slope of the input ramp, the delay is given by

$$t_{f_i,\text{ramp}} = v_{TN} \cdot \frac{\tau}{2} + \left(1 + 2\frac{C_M}{C_L}\right) t_{f_i,\text{step}} \tag{3.5}$$

Here, v_{TN} is $\frac{V_{TN}}{V_{DD}}$ where V_{TN} is the threshold voltage of the n-transistor and V_{DD} is the supply voltage. Typical values of v_{TN} and $\frac{C_M}{C_L}$, which we use in this work, are 0.2 and 0.1, respectively [49]. The term $t_{f,\text{step}}$ corresponds to the step input response corresponding to the fall transition. A similar expression is used for the rise transition. The value of τ is taken to be twice the Elmore delay of the preceding gate, as in [85].

Using this method to calculate the delays of individual gates, the PERT procedure is used to find the critical path in the circuit as in [37].

The proposed algorithm also requires the computation of the sensitivity of the gate delay with respect to a gate size. It is well-known [37] that the step delay sensitivity to a gate size can be computed by considering only that gate (whose resistance is affected by the gate size) and its fanin gates (whose load capacitances are affected by the size of that gate). Therefore, the delay sensitivity computation under step inputs is a very local computation.

Under the improved delay model above that considers input transition times, the size of a gate affects not only the delay of that gate and its fanin gates, but also the delay of all gates in the transitive fanout. The delays of the gates in the transitive fanout depend on the value of their input slew rates (τ values), which in turn, are dependent on the delay of the current gate. However, it can easily be shown from the application of Equation (3.5) that for real parameter values, the effect of changing a gate size is vastly diluted as one moves further and further away from the gate along its transitive fanout. For real circuits, we found that the size of a gate affects only the current gate, its fanin gates, its immediate fanout gates, and their fanouts. Therefore, for all practical purposes, the sensitivity computation remains an inexpensive local computation, even under the improved delay model that considers input transition times.

3.2 Buffer Insertion

The essential idea of buffer insertion is to reduce the delay at high capacitance nodes by reducing the load on the driving gate. To maintain signal polarities, we assume that each buffer consists of a pair of inverters that may be sized appropriately. Thus, the addition of each buffer implies the addition of four new transistors to the circuit.

3.2.1 Notions of Criticality

As a preliminary step, we define a *critical* path as any path that violates the timing specification. We also explain a nonquantitative and somewhat fuzzy term that we term as the criticality of a path. Roughly speaking, the criticality of a path is dependent on the magnitude of the violation, so that paths with large violations are identified as being highly critical, and those with small violations are only mildly critical. This notion is important since we observe that the greater the criticality of the path, the larger the amount of sizing required for the path to meet specifications. Later in this chapter, we will work towards developing measures to quantify the criticality of a path.

Generally speaking, it has been our experience that buffer insertion is useful only for highly critical paths. This experience is based on our experimental results which use a measure of criticality, developed later in this chapter, to quantify the criticality of a path. For mildly critical paths, it may be more advantageous to use sizing than buffer insertion. The intuition behind this is that mildly critical paths can be made to meet timing specifications through a small amount of sizing; inserting a buffer implies an increase in area corresponding to the four new transistors that constitute the buffer, which is likely to be larger. Moreover, the addition of an excessive number of buffers can actually increase the delay of some paths of the circuit, and therefore we add them only where we must, namely, to reduce the delays on the highly critical paths.

3.2.2 Types of Buffer Insertion Strategies

We identify two situations in which the insertion of buffers is advantageous, which we will refer to as Type A and Type B buffer insertion scenarios, respectively. As shorthand notation, we will refer to an output H of a gate G being highly critical if some highly critical path passes through gates G and H; similarly, we also refer to mildly critical and noncritical outputs.

Type A If a gate whose outputs are all *highly critical* drives a large capacitive fanout, buffer insertion can help in reducing the delays of these paths. Figure 3.3 shows the situation of

type A buffer insertion[3]. By choosing an appropriate size of buffer, the fanout capacitance of Gate G may become smaller, and sum of the delays of the buffer and Gate G may be smaller than the delay of Gate G in the unbuffered circuit.

Type B If a gate has some highly critical outputs and some mildly critical and noncritical outputs, then one may isolate the capacitance of the noncritical outputs from the highly critical path by inserting a buffer, as shown in Figure 3.4. The mildly critical paths constitute a gray area and must be assigned to be either critical or noncritical, based on measures that we will develop later in this chapter. Since the fanout capacitance of gate G becomes smaller, the RC delay of G is reduced, and therefore, the delay along the highly critical paths is reduced.

As a side-effect, the delay along the noncritical paths may be increased. The additional delay introduced along noncritical paths that became critical after buffer insertion, can be made to meet specifications through a small amount of sizing.

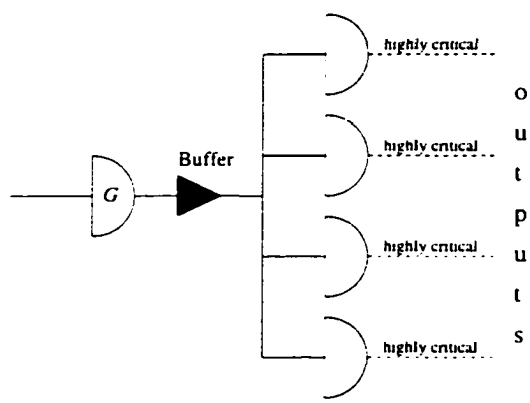


Figure 3.3 Type A buffer insertion

The challenge here is to quantify measures of criticality, and to use them to determine appropriate locations for buffer insertion.

Interestingly, the work in [87] that was performed independently also uses similar terminology for Type A and Type B buffers. However, that work concentrates on reducing wire delay

[3]The essential idea here is not dissimilar to the Mead-Conway idea of using chains of inverters to drive a large load, with a ratio of e minimizing the delay. However, we differ in the following ways: (a) our objective is not to minimize the delay but to meet a specification (b) if the circuit as a whole is anything other than a chain of inverters, it is not possible to use the constant-ratio idea to minimize the delay of the circuit.

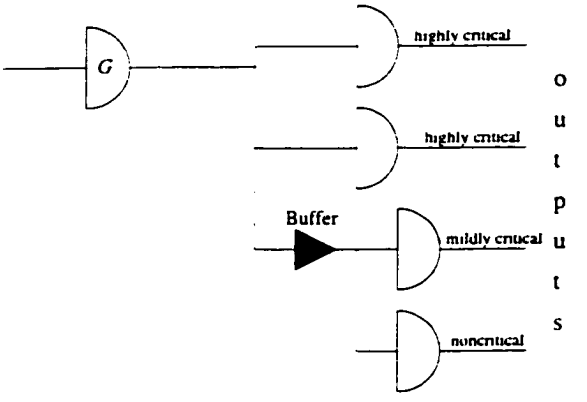


Figure 3.4 Type B buffer insertion

at the post-layout phase, an issue that we do not address here.

3.2.3 Examining the Effect of Buffer Insertion

We will examine the effect of buffer insertion through a simple example. Consider the gate G shown in Figure 3.5 driving gates $G_a, G_b, G_c, \cdots, G_f$. If all of the outputs are on highly critical paths, then we would insert a Type A buffer immediately after G, driving all fanouts. The insertion of this buffer would change the fanout capacitance of G and therefore, its delay would change from $D_{G,old}$ to $D_{G,new}$. If the delay of the buffer is D_{buf}, then the change in the delay to the most critical output would be $D_{G,new} + D_{buf} - D_{G,old}$, since all other gate delays in the circuit would be unaffected. For the buffer insertion to be advantageous, this value must be negative.

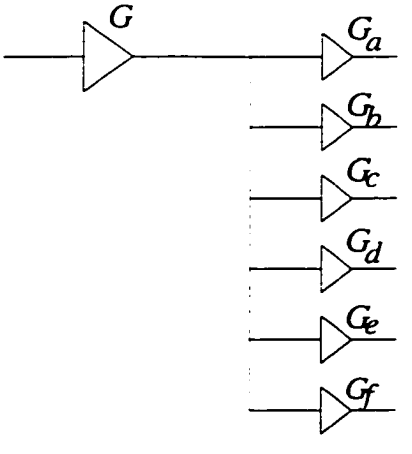


Figure 3.5 The effect of buffer insertion

Note also that this transformation would change the delay of *any* path passing through G by the same amount, and would therefore decrease it; for any path that does not pass through G, the delay remains unaffected. Therefore, this transformation either reduces the delay at each primary output, or leaves it unaffected.

If we consider Type B buffer insertion in Figure 3.5, and let us now assume that G_a and G_b are highly critical, G_e and G_f are noncritical, and G_c and G_d are critical, but not highly critical. We will refer to a gate G_i as being buffered if the Type B buffer is placed between the output of G and G_i, and we will consider it unbuffered otherwise. We can now place a single Type B buffer using the following ideas:

- G_a and G_b must certainly be unbuffered.
- G_e and G_f should be buffered, since they only add to the capacitance being driven by gate G. Although it is possible that G_e and G_f may become critical outputs after buffer insertion, they would, at worst, probably be very mildly critical since they were noncritical before buffer insertion.
- The key issue is the status of G_c and G_d. If they are buffered, then they may become highly critical after buffer insertion. On the other hand, if they are not buffered, the capacitance at G may be too high and the delay of gate G may not be reduced sufficiently by buffer insertion. Therefore, the best solution may either buffer off none, one, or two of the gates G_c and G_d, and a good criterion is required to determine which of these should be chosen.

3.2.4 On Complexity and Convexity Issues

The transistor sizing problem is well known to be equivalent to a convex programming problem [37, 85] when the topology of the circuit is fixed, since the area objective and the circuit path delays can be represented as posynomial [35] functions of the transistor sizes[4].

[4] A posynomial is a function g of a positive variable $\mathbf{w} \in \mathbf{R}^n$ that has the form $g(\mathbf{w}) = \sum_j \gamma_j \prod_{i=1}^{n} w_i^{\alpha_{i,j}}$, where the exponents $\alpha_{i,j} \in \mathbf{R}$ and the coefficients $\gamma_j > 0$. Roughly speaking, a posynomial is a function that is similar to a polynomial, except that (a) the coefficients γ_j must be positive, and (b) an exponent $\alpha_{i,j}$ could be any real number, and not necessarily a positive integer, unlike the case of polynomials. A posynomial has the

However, when the structure of the circuit is allowed to change, this is no longer true. If there are b possible buffer locations associated with a path, then there are b possible delay functions $f_1, f_2 \cdots f_b$ (each a posynomial), of which one (or at most a few) is optimal. Note that an optimal circuit is the circuit with the minimum area for the given delay specification. The path delay is thus f_1 or f_2 or $\cdots f_b$, which cannot be represented as a convex programming problem (it may, however, be written as an mixed integer nonlinear programming problem), and its solution is not easy to find. A second pointer to its difficulty is that even a special restriction of the problem, that of finding the optimal locations for Type B buffers in an unsized circuit, is NP-complete [7]. Therefore, we resort to heuristic methods for solving the problem.

3.3 Outline of the Algorithm

The procedure developed here enhances the TILOS algorithm [37] which operates iteratively, identifying the most critical path in every iteration. The sensitivity of the path delay, D, to the area, A, given by $\partial D/\partial A$, is computed for all of the transistors along the critical path, and the transistor with the most negative sensitivity is bumped up by a factor, `Bumpsize`. `Bumpsize` is typically set to a value that is just larger than one, and values between 1.1 and 1.5 have been seen to work well. The procedure continues until all timing specifications are met.

As in the TILOS algorithm, we begin with the unsized circuit as provided to us. We continue optimizing the circuit until all the delay constraints are met at every circuit output. Until that is achieved, in each iteration, we identify the most critical path, i.e., the path with the largest violation of the timing specification. We attempt to improve the delay along this path by one of several possible transformations

- bumping up the size of some transistor along the path
- inserting a Type A buffer along the critical path
- inserting a Type B buffer to isolate noncritical paths from critical paths

useful property that it can be mapped onto a convex function through an elementary variable transformation [35] $(w_i) = (e^{x_i})$

The general philosophy behind the algorithm is shown below. It should be stressed that although this is the general philosophy, the actual implementation is somewhat different and will be elaborated on in subsequent sections.

```
minimum_delay = minimum-sized circuit delay
Initialization (all gate sizes are set to minimum values)
While (delays at all primary outputs are not ≤ T_spec) {
      Compare path delays with T_spec and find the most critical path
      For all gates on the critical path {
          Estimate figure of merit of bumping up a transistor
          Estimate figure of merit for inserting a Type A buffer
          Estimate figure of merit for inserting a Type B buffer
      }
      If (bumping up a transistor has the best figure of merit)
          increase the size of a selected transistor
      if (inserting a Type A buffer has the best figure of merit)
          insert a Type A buffer
      if (inserting a Type B buffer has the best figure of merit)
          insert a Type B buffer
      Recompute circuit delays
      if (circuit_delay < minimum_delay) minimum_delay = circuit_delay
      if (circuit_delay > 1.1 × minimum_delay)
          /* failed to meet specifications */
          exit
}
```

The iterations end if futher sizing does not result in delay reduction, and in fact, increases the circuit delay by a significant amount.

In the following sections, we will consider the problems of developing figures of merit for bumping up a transistor and for inserting a Type A or a Type B buffer. Since we have followed the TILOS template, we will also use the most negative sensitivity $S_T = \partial D/\partial A$ of a transistor to compare the relative figures of merit. Note that S_T corresponds to the delay reduction caused by bumping up the transistor size; this fact will be used when we develop comparable figures of merit for inserting Type A and Type B buffers.

3.3.1 Type B Buffer Insertion

3.3.1.1 Rationale Behind the Procedure

The purpose of using Type B buffers is to insulate the noncritical paths from the highly critical paths, thereby enabling greater amounts of delay reduction for the circuit as a whole.

As stated in the outline of the algorithm, the objective is to determine a figure of merit that can reasonably be compared to the figure of merit for sizing, namely, the sensitivity of the most sensitive gate. Let us temporarily assume that we have developed a way of measuring the criticality of a gate output, and that we can recognize the highly critical outputs. We will later show the precise method by which this is achieved in Section 3.3.1.3.

While considering a candidate Type B buffer location at one of the outputs of a gate, we first consider the delay along its highly critical fanouts. By definition, a Type B buffer will always reduce the delay to a highly critical fanout, and this is achieved at the expense of an increase in the area: the area increase corresponds to the area of the inserted minimum-sized buffer.

Therefore, a reduction in the delay by an amount ΔD can be effected by an area increase of ΔA. We must now estimate the amount of area, ΔA_T, required by the sizing procedure to achieve the same delay reduction. If $\Delta A < \Delta A_T$, then we insert the Type B buffer.

To fairly compare the effects of sizing and Type B buffer insertion, let us consider the following problem:

> For the same reduction in delay, ΔD,
> what is the increase in the area required by the two procedures?

It is tempting to estimate ΔA_T as $-\frac{\Delta D}{S_T}$ (recalling that S_T, the figure of merit for transistor sizing, is the most negative sensitivity of a critical path transistor), as shown by curve (a) in Figure 3.6. However, the corresponding change in area, $\Delta A_{(a)}$, is only a lower bound on the value of ΔA_T and is typically not a very tight bound. This is because the sensitivity S_T corresponds to a *small* perturbation, whereas the change ΔD is large. A linear approximation would provide optimistic estimates of ΔA_T. Moreover, for a transistor with size x, it has been

shown that

$$S_T = K_1 - \frac{K_2}{x^2} \tag{3.6}$$

where K_1, K_2 are independent of x [37]. The above equation is accurate for the delay of the current critical path, but not for the delay of the entire circuit, which is the maximum path delays; note that the critical path may change when a transistor size is altered.

A second idea would be to use Equation(3.6) to estimate ΔA_T, as shown by curve (b) in Figure 3.6. However, this estimate, $\Delta A_{(b)}$, is accurate only if this same transistor is critical in every sizing step involved in reducing the delay by ΔD. This is typically not true, and therefore, such an expression would provide an upper bound on the area.

In most cases, the actual area-delay curve would lie between the two bounds as shown by curve (c) in Figure 3.6, and our problem is to determine the shape of this curve and the value of ΔA_T.

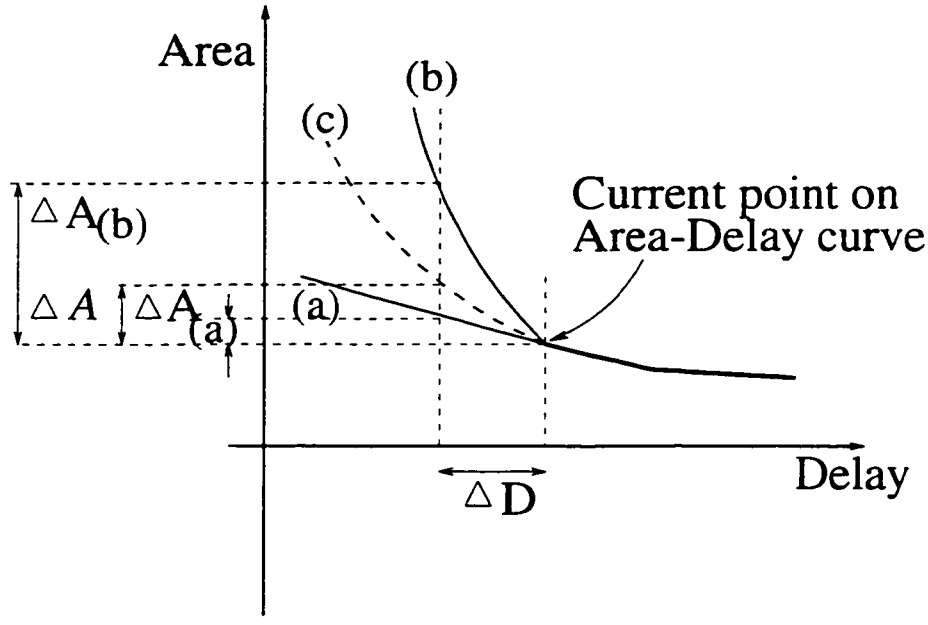


Figure 3.6 Estimating ΔA_T

An additional complication is as follows. Consider, for a moment, the TILOS algorithm for transistor sizing, and let D be the delay of the circuit during the current iteration. Then, bumping up the size of an individual transistor causes a delay reduction of δD on the current most critical path and an area increase of δA. However, the *circuit* delay is not necessarily

reduced by δD since the bumping operation could cause a different path to become the most critical path. This approximation is justifiable in TILOS because the area and delay change in each iteration are very small. Note that if we wanted to be exact, we should have considered the area increase required to constrain *all* path delays to $D - \delta D$.

However, if the delay is changed from D by a *large* amount, ΔD, as is the case in our situation, such an approximation is invalid, and we must find the area increase required by sizing to ensure that the delays of *all* paths (and not just the current most critical path) are less than $D - \Delta D$. In other words, all of these paths must be sized appropriately, and ΔA must be computed by considering the effect of all of these paths, and not just the most critical path as in TILOS.

To take care of this problem, we consider all primary outputs, and find the area increase required to ensure that the maximum delay over *all* outputs (and not just at the critical output) is no larger than $D - \delta D$.

3.3.1.2 Details of the Procedure

As indicated above, calculating ΔA_T for a gate with size w as $\left[\Delta D / \frac{\partial D}{\partial A}\right]$ is very inaccurate since

(1) the sensitivity $\frac{\partial D}{\partial A}$ varies nonlinearly with the gate size.

(2) a large delay reduction of ΔD is probably best achieved by sizing not just this single gate, but several other gates too.

To estimate the value of ΔA_T, given a specific buffer insertion point, we first calculate the change in the delay of the circuit due to the insertion of a minimum sized Type B buffer. At each such primary output i, we use an extrapolation method to estimate the area increase, Δa_i, required to match the circuit delay reduction. We then calculate the figure of merit for sizing as

$$\Delta A_T = \sum_{i \in po} \Delta a_i \tag{3.7}$$

The extrapolation procedure is implemented as follows. For each primary output, we store the effect of the most recent sizing steps as a delay vs. circuit area table. We use those data to extrapolate the change in area corresponding to ΔD at every output: these values are summed up to give ΔA_T, as described in the last section. Specifically, we use Lagrangian extrapolation [79] to estimate ΔA_T for ΔD. We found that a fourth order polynomial approximation was adequate.

If ΔA_T, the estimated area required to achieve the delay reduction through sizing alone, is lower than ΔA_B, the area of a minimum sized Type B buffer to be inserted at the chosen point, then the buffer is inserted. If not, the algorithm abandons the Type B buffer insertion in the current iteration, and then chooses either a Type A buffer insertion or a sizing step.

3.3.1.3 Finding an Appropriate Location for Buffer Insertion

The criterion for Type B buffer insertion is to isolate the less critical paths from the more critical ones: if the total capacitance of the less critical paths is substantial, then significant delay improvements are possible. Therefore, our first challenge is to develop a measure for criticality, which is key to the success of this algorithm and is required to partition the fanouts of a gate into a "critical" and "noncritical" set, as shown in Figure 3.7. We will also quantify the criticality of the mildly critical path, which constitute a gray area, and develop measures to decide whether they should be considered critical or noncritical.

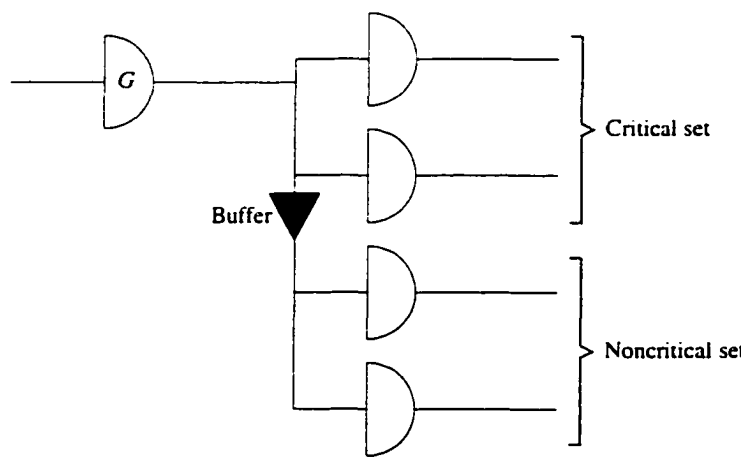


Figure 3.7 Partitioning the fanout gates for type B buffer insertion

Consider the sensitivity, $\frac{\partial d}{\partial x_i}$ for each gate i, where x_i is the size of the gate, and d is the delay of the most critical path through the gate. We consider the possibility of bumping up the size of gate i, and recognize that it is pointless to size a gate with positive sensitivity as that would increase the circuit delay. We maintain the number

$$\sigma_i = \min(0, \frac{\partial d}{\partial x_i} \cdot \Delta x_i) \tag{3.8}$$

for each gate i, where Δx_i is the amount by which the gate size would be increased if it were to be bumped up. Therefore, σ_i estimates the reduction in the gate delay through a possible bumping up operation. Note that gates with a positive sensitivity are assigned a σ_i of zero since the gate size would be left unchanged if the bumping operation were to increase the delay.

We define a measure for the criticality that we call χ, associated with each gate fanout. This measure is related to the amount by which the delay of a circuit can be reduced and to the delay along a path. Fanouts with larger χ values are less critical than those with smaller χ values.

A backward PERT traversal[5] is performed from the primary outputs towards the primary inputs (PI's) to calculate the value of χ for each gate. The χ value at each primary output is set to be the difference between the maximum delay at the primary output and the actual delay to that point. Therefore, increasing the path delay to that primary output by χ will leave the circuit delay unchanged.

If we know the χ value for all the fanouts of a given gate i, its own χ value is calculated as

$$\chi_i = \min_{j \in \text{fanouts}(i)} [\chi_j + \text{slack}_j] + |\sigma_i| \tag{3.9}$$

where slack_j represents the slack at fanout j. The slack is defined as the amount by which the delay along this path may be increased before it becomes the longest delay path in the circuit. Note that all elements in this equation have dimensions of delay. Therefore χ_i is a measure of the amount of delay increase along a path from the gate to any primary output that can be absorbed "easily," either by the slack or by a small amount of sizing. This is consistent with

[5] As pointed out in [15], the method referred to as PERT in the CAD literature is actually the critical path method (CPM). However, we persist with the prevailing incorrect usage to avoid confusion.

the idea that a high value of χ_i at the fanout j of a gate i implies that the maximum delay path from i to a primary output through j is not very critical.

The next challenge is to use this measure of criticality to determine the best location at which a Type B buffer should be inserted to improve the current most critical path. The steps involved in determining the buffer location can now be summarized as follows:

1. Find the gate i with the maximum fanout capacitance along the most critical path of the circuit. We will consider inserting a Type B buffer at the output of this gate to partition the critical and noncritical fanouts[6].

2. Find the maximum value of χ_j of all fanouts of gate i; let χ_{max} be the maximum value of χ_j. All fanouts j whose χ_j is $\geq c_1 \cdot \chi_{max}$ (where $c_1 < 1$ is an empirically tuned number) are placed in the noncritical set[7] This has the effect of placing all fanout gates with high χ values in the noncritical set.

   However, if too many gates are placed in the noncritical set, the capacitance to be driven by the type B buffer may become too high. This could cause its delay to be very large, so that path through i and some of the gates in the noncritical set may become very critical. Therefore, the above classification may be too optimistic, and we apply the next criterion described below.

3. Having determined the noncritical set, we next estimate the effect of inserting a minimum-sized buffer. When a buffer is inserted, the delay of gate i is reduced by an amount ΔD_{dec}, which is the delay reduction along the critical paths. Along a noncritical fanout j, the delay is increased by $\Delta D_{inc} - \Delta D_{dec}$, where ΔD_{inc} is the increased delay due to the insertion of a buffer.

   Recall that χ_j is an estimate of the amount by which the delay from j to the primary outputs may be increased before j lies on the most critical path of the circuit. Of this amount, $\Delta D_{inc} - \Delta D_{dec}$ is consumed by the insertion of a buffer.

[6]Several other criteria for selecting gate i were tried out; however, this criterion was seen to be the most successful across all circuits.

[7]We reiterate that some of the mildly critical paths may also be quantized as being "noncritical."

Therefore, with the insertion of the buffer, we may say that the delay from j to the primary outputs may be increased by $\chi_j - (\Delta D_{inc} - \Delta D_{dec})$ before j would lie on the most critical path. The larger this amount, the less critical the path would be after buffer insertion. Therefore, we calculate this quantity for each fanout and if its value is small, then we remove the fanout j from the noncritical set.

4. For any fanout j, if $\chi_j - (\Delta D_{inc} - \Delta D_{dec}) < \beta$ for some empirically determined β, then the gate is moved from the noncritical set to the critical set.

   The value of β was chosen as $c_2 \cdot d_{minsize}$, where $d_{minsize}$ is the delay of a minimum size inverter driving a minimum size load. The use of $d_{minsize}$ is purely for normalization purposes to ensure that the value of β is of the correct order of magnitude. The value of c_2 is then determined empirically.

It was experimentally found that approximate values of $c_1 = 0.8$ and $c_2 = -0.5$ work well on the circuits that we tested. Note that a negative value of c_2 causes a negative value for β.

Recall that a mildly critical path (using the terminology of Figure 3.4) may also be buffered off. The negative value for β corresponds to a mildly critical path where the insertion of a buffer may actually increase the path delay enough to cause the value of $\chi_j - (\Delta D_{inc} - \Delta D_{dec})$ to be negative. When this value is negative, it might seem that the path delay would increase after buffer insertion. However, in our calculations, we assumed a minimum sized buffer, and if the value of $\chi_j - (\Delta D_{inc} - \Delta D_{dec})$ is very slightly negative, we may recover from this easily by sizing the buffer by a small amount. Therefore, in practice, we found that a small negative value for c_2 gave good results.

3.3.2 Type A Buffer Insertion

During the iterative procedure, we observed that inserting a minimum-sized Type A buffer almost always caused the path delay to increase. However, by appropriately choosing the size of the Type A buffer, delay reductions can be effected. The following procedure is used to estimate the potential delay reduction through Type A buffer insertion at each gate output:

1. Find the minimum (most negative) sensitivity among the gates along the most critical path, denoted as $\left.\frac{\partial D}{\partial x}\right|_{best}$.

2. For each gate on the most critical path, we calculate the values of ΔD_{rise} and ΔD_{fall}, the changes in the rise and fall delays, respectively, if a Type A buffer were to be inserted at that gate output. For the buffer insertion step to achieve a useful purpose, it must be ensured that both $\Delta D_{rise} < 0$ and $\Delta D_{fall} < 0$. Keeping the topology of the rest of the circuit constant (in particular, keeping the sizes of the gates fanning into and out of the proposed buffer constant), the sizes of transistors in the buffer are estimated[8].

   Only those gates at which both the rise and fall delays can be reduced are considered as candidates for buffer insertion. For these gates, the sensitivity of the buffer, $\left.\frac{\partial D}{\partial x}\right|_{buffer}$, is determined for the calculated size.

   If $\left.\frac{\partial D}{\partial x}\right|_{buffer} < \left.\frac{\partial D}{\partial x}\right|_{best}$, then this location is designated as a permitted buffer insertion location. The rationale behind this is that at this point, after buffer insertion,

   (a) the delays of all the paths driven by the buffer are smaller than those of prior to buffer insertion, and

   (b) the most negative sensitivity value on that critical path is made even more negative than before, implying that the increase in area due to buffer insertion could be recouped in future steps through sizing. In other words, the potential for reducing the path delays with buffer insertion is better than the ability without buffer insertion.

3. Among the permitted buffer insertion points in Step 2, the output of gate k with the best delay reduction is chosen to be the best Type A buffer insertion location. The value of $(\Delta D_{rise} + \Delta D_{fall})/2$ is used to estimate the effect of buffer insertion on the delay.

4. Having performed a Type A buffer insertion, the buffer and its predecessor gate k are now reset to the minimum size to correct for any over-sizing in k in the past. The sizing procedure is permitted to size these gates back up again in subsequent iterations to their

[8]For purposes of this estimation only, we make a simplification where we consider that the size of the p-transistor is K times that of the n-transistor, and use a simple iterative loop to solve for the transistor sizes in the buffer that minimize the average of the rise and fall delays.

optimal sizes, so that the solution is not unduly bound by any incorrect sizing choices that were made before the buffer was added. During this process, we prohibit further Type A buffer insertion until the iterations reach the point where the circuit delay becomes smaller than that before the insertion of this Type A buffer.

3.3.3 The Final Algorithm

The pseudocode shown in Section 3.3 was only a general outline of our procedure, and we may now describe the pseudocode of the algorithm more accurately as follows:

```
minimum_delay = minimum-sized-circuit-delay
while (current_delay > T_spec)
{
  if the criteria of type B buffer insertion are satisfied
        then perform type B buffer insertion.
  else if the criteria of type A buffer insertion are satisfied
        then perform type A buffer insertion.
  else
        bump up the most sensitive transistor.
  recalculate current_delay
  if (current_delay < minimum_delay) minimum_delay = delay
  if (current_delay > 1.1 × minimum_delay) exit.
}
```

The algorithm chooses to consider the option of inserting a Type B buffer first, and then considers the Type A buffer, finally defaulting to transistor sizing if neither is viable. It is possible to consider these in any order, but it was found that this ordering worked best for the circuit examples that we tried.

We now attempt to provide an estimate of the amount of computation involved in each iteration. While a detailed complexity analysis is unrealistic due to the unpredictability of the number of iterations, it is useful to count the number of computations involved in each iteration of this algorithm.

We assume that the number of gate fanins and fanouts are bounded by a constant, which implies that delay and sensitivity calculation for each gate can be carried out in constant

time. The timing analysis required to calculate the circuit delay is $O(|V| + |E|)$, where $|V|$ is the number of vertices in the circuit graph, corresponding to the number of gates in the circuit, and $|E|$ is the number of edges in the circuit graph, where each edge corresponds to an interconnection from one gate to one of its fanouts. After the first time, however, the computation is significantly reduced since incremental techniques are used. In the worst case, we only process all edges in the fanout cone of the predecessor of the gate that is sized or the buffer that is inserted. The worst-case complexity of this step is also $O(|V| + |E|)$, but the typical update is empirically seen to occur in much less time. During delay calculation, the slack at each node is also calculated at no additional increase in the computational complexity.

The next step involves the calculation of gate sensitivities along the critical path. If D_c is the depth of the circuit (largest number of gates on any path) then the number of gates on the critical path is bounded above by D_c, and the amount of time required to compute the sensitivities and to find the maximum sensitivity is $O(D_c)$. This step is the only computation required for the sizing operation, and is also required by the criteria of Type A and Type B insertion.

For type B buffer insertion, the calculation of σ values is required in the fanout cone of gates in the critical path. This can be carried out in $O|V|$ time. This is followed by a PERT procedure that uses the slacks to compute the χ values in $O(|V| + |E|)$ time. The gate on the critical path with the highest capacitance is found in $O(D_c)$ time. Since the number of fanouts is bounded, the use of the χ values to partition the fanouts into critical and noncritical fanouts is completed in constant time. Note that the comparison with sizing, illustrated in Figure 3.6 is performed in constant time.

For type A buffer insertion, the amount of time required for steps 1 through 4 in Section 3.3.2 is $O(D_c)$, assuming (as is seen in practice), that the iterations of step 2 take constant time.

Therefore, in summary, each iteration requires $O(|V| + |E|)$ time for timing analysis and slack calculation, $O(D_c)$ time for sensitivity calculation, $O(|V| + |E|)$ time to evaluate type B buffer insertion, and $O(D_c)$ time to evaluate Type A buffer insertion, and since $D_c < |V|$,

the overall complexity of each step is $O(|V| + |E|)$. We emphasize that due to the incremental techniques used, this is a pessimistic estimate of the complexity.

3.4 A Brief Note on Unsuccessful Strategies

For purposes of completeness (and since a negative result is also sometimes a worthwhile result), we believe that it is also worthwhile to point out a few strategies that seem sound on the surface, but were found to be unsuccessful in our experiments.

3.4.1 Incorporating Rollback

Since the insertion of a buffer is a drastic step, we considered including rollback, where after each buffer insertion, a certain number of prior sizing steps were nullified. The idea behind rollback is that any sizing steps performed immediately prior to the buffer insertion step may have been suboptimal since they were performed under the assumption that no buffer would be inserted. Therefore, it was thought to be a good idea to consider rolling back to an earlier iteration and resuming the process from there.

However, in practice, we tried several criteria for incorporating rollback and found that the results using rollback were seldom better, while the memory requirements and execution times were phenomenally large. Therefore, we abandoned the idea of incorporating rollback into the optimization process.

3.4.2 Gate Cloning

An alternative to buffer insertion would be to clone gates to perform Type B buffer insertion. The primary idea is that instead of creating a new buffer, the use of cloned versions of a gate could be useful. For example, if a Type B buffer is to be inserted at the output of an inverter, then cloning the inverter amounts to an additional expense of two transistors, while inserting a buffer (which consists of two inverters) amounts to an expense of four transistors. Secondly, buffer insertion increases the number of levels in a circuit and therefore may cause unnecessary delay increases along paths that lead to outputs of moderate criticality. The use of cloning

may resolve this problem by avoiding the insertion of an additional level of logic in the form of a buffer.

In our implementations we found that when we added gate cloning to the list of strategies used here, we never obtained better results on any of the benchmark circuits. This can be attributed to the fact that situations such as the above do not occur sufficiently often in these circuits. However, one could certainly generate an artificial example where gate cloning could be useful.

3.5 Experimental Results

The algorithms described above have been implemented in C on an HP 735 workstation. In Table 3.1, we present the results on some circuits from the ISCAS85 [4] and LgSynth91 [99] benchmark suites.

For each circuit, the number of gates $|G|$, the unsized delay D_u, and the unsized area A_u are shown. For a given (moderate) timing specification T_{spec}, the area of our approach is compared with the area from our implementation of TILOS, which is a direct implementation from [37]. The differences between our implementation of TILOS and [37] is that we replace each gate by an equivalent inverter, characterized by gate sizes W_n and W_p, and solve the circuit to find the optimal W_n and W_p and, in case of our algorithm, the optimum buffer locations too. Moreover, we use a timing model that takes input slew times into consideration. Next to the area numbers the table are also shown (in brackets) the number of Type A and Type B buffers. It is seen here that most of the buffers in this table are of Type B. Although not shown in this table, it was observed that for tighter specifications, a larger number of Type A buffers were added. The CPU times for both methods are very similar. The area ratio shown in the last column shows the ratio of the area required by sizing alone as compared to the area required by our method. Therefore, this number should be at least equal to 1 (as it always is here) and a larger magnitude implies a better improvement over sizing alone. Significant improvements are possible in most cases. We point out that as the delay specification is tightened further, larger area savings are possible for each circuit for tightened constraints.

Table 3.1 Comparison of Sizing vs Sizing+Buffer Insertion

Circuit	$\|G\|$	D_u	A_u	T_{spec}	*Sizing*		*Sizing+Buffer Insertion*		Area Ratio
					Area	CPU (s)	Area	CPU (s)	
cc	58	61.4	248	23	900	6.7	706 (A:1:B:6)	4.9	1.27
cm163	43	43.4	160	14	692	3.0	428 (A:1:B:5)	2.6	1.62
f51m	136	82.5	548	50	2627	17.6	1627 (A:1:B:4)	13.3	1.62
i135	269	121.1	1252	36	4307	29.9	2183 (A:0:B:13)	32.8	1.97
c499	202	177.3	816	51	3004	30.2	2571 (A:1:B:15)	62.4	1.17
c1355	546	324.5	2128	100	5001	145.9	4279 (A:1:B:38)	192.3	1.17
c2670	1193	456.0	4152	88	9000	481.3	8586 (A:1:B:94)	595.7	1.05
c5315	2307	831.2	8772	190	15000	987.2	13619 (A:1:B:125)	1013.3	1.10

For example, for circuit c5315 with a timing specification of 190ns, our approach provided an area of 13619, while the area of the circuit using sizing alone was 15000. This corresponds to a savings of about 10%. Our approach requires the insertion of 1 Type A buffer and 125 Type B buffers to meet this specification.

The entire area-delay tradeoff for this algorithm for three different benchmark circuits is shown in Figures 3.8 through 3.10. In each case, it is seen that significant improvements are possible from the use of our approach, particularly for tighter specifications. The reader is cautioned that although some curves, such as the one in Figure 3.9, seem to be close to each other, in the steep region of the curves, even small differences are greatly magnified on the y-axis, and our approach gives significant cost savings in that region.

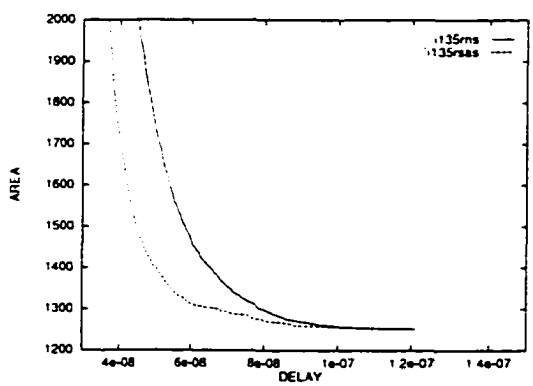


Figure 3.8 Area-delay tradeoff for circuit i135

For some circuits, such as c499 (Figure 3.9) and c1355 (Figure 3.10) the area from our approach for loose delay specifications is very slightly worse than that from sizing alone. The explanation for this can be seen by examining our approach in a different light. In each step, the approach attempts to reduce the delay of the circuit, going along an area-delay tradeoff curve that is similar in nature to that shown in Figure 3.1, with a smaller value of d_{min}. As the algorithm progresses, each iteration represents a motion to the left along this tradeoff curve. The method typically "looks ahead" to determine if a buffer will be required to meet a delay specification (with the best area) several iterations in the future. Therefore, for a few iterations after the buffer is inserted, the results are likely to be slightly suboptimal, and some of this is manifested in the results. Since we are primarily interested in sizing circuits to meet tight delay specifications, which is a region where our algorithm works well, we have not taken any steps to remedy the occasional minor problem with loose delay specifications.

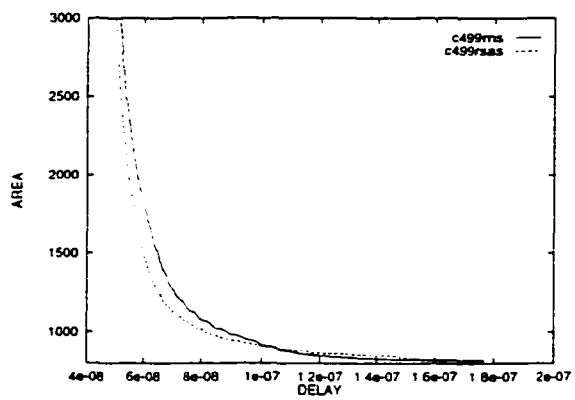


Figure 3.9 Area-delay tradeoff for circuit c499

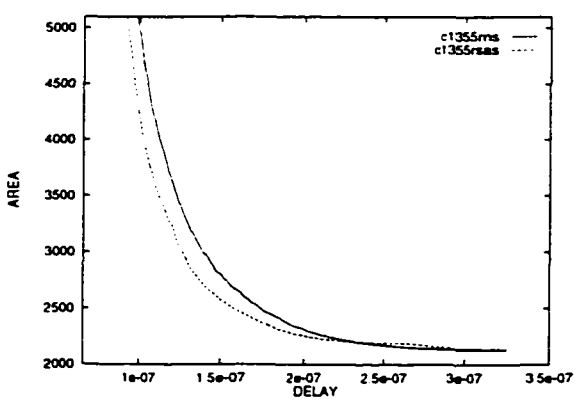


Figure 3.10 Area-delay tradeoff for circuit c1355

3.6 Conclusion

In this work, we have aimed to support the basic idea that buffer insertion can help to improve the area-delay tradeoff curve and have presented heuristic algorithms for the purpose. The gate sizing procedure is known to be power-conscious since it sizes gates only when necessary and reduces the dynamic power: in this work, the efficacy of this is further improved by considering buffer insertion to achieve the delay goal for the circuit with a smaller area/power cost. Additionally, it is ensured that buffers are added only as needed so as to minimize the area and the power dissipation, and the process of buffer insertion is targeted towards meeting a given specification, rather than towards minimizing the circuit delay. The techniques developed herein are supported by experimental results that demonstrate that improvements can be achieved both in the area and the minimum achievable delay in comparison with an algorithm that performs sizing alone.

4 GATE COLLAPSING AND MIXED STATIC CMOS AND PASS TRANSISTOR DESIGN

Technology mapping has been a cornerstone in the logic synthesis process and this area has been well studied in the past. Existing technology mapping techniques can be classified into four categories: rule-based mapping [43], graph matching [55], direct mapping [63] and functional mapping [66]. Traditional methods for technology mapping are directed towards a *specific* library and are targeted towards objectives such as minimizing the circuit delay, minimizing the area and reducing the power dissipation. Using a pre-characterized library methodology has the inherent major disadvantage that the quality of the results is dependent on the richness of the library: a library with a larger number of cells is likely to lead to better results than a sparsely populated library.

For deep-submicron technologies, it has been shown in [81] that the ratio of the delay of NAND/NOR gates to the inverter delay is becoming smaller than in older technologies, which encourages the use of longer chains of pull-up/pull-down logic in circuits and therefore will lead to an increased usage of complex gates in deep-submicron circuits. While this leads to better circuit performance, it also complicates the problem of traditional library-based technology mapping. With the increasing use of complex gates in the design, the number of possible gate types increases exponentially [31]. Therefore, the number of gates in any library of a reasonable size can only capture a small fraction of the total number of possibilities and traditional technology mapping is too restrictive.

To take full advantage of the availability of complex gates, the idea of using a dynamic "virtual library" is becoming attractive. In this approach, individual complex gates are generated off the fly, instead of using a pre-characterized library. The translation of this new gate to a layout can be performed using a module generator. We note that as layout synthesis systems

have become more mature, module generators can synthesize layouts for arbitrary complex gates accurately and efficiently.

The design methodology is as shown in Figure 4.1. This methodology can be extended to work with the traditional design methodology, so that in case a library is available, the complex gates may be implemented using either the cells in the given library or the virtual library.

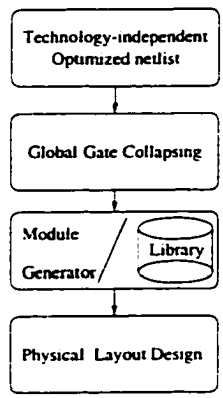


Figure 4.1 New design methodology

In this work, we propose a new design methodology that uses a virtual library, such as the one described above, with an unrestricted variety of cells. The procedure automatically generates a complex gate of an appropriate size during the technology mapping phase. The focus of this chapter is on the *gate collapsing* phase which is the basis for this new design methodology; this is the phase where the complex gates are generated. The discussion about the module generator for layout of the complex gates is beyond the scope of this chapter and the reader is referred to the wide body of literature on this topic.

The essential idea of gate collapsing is to begin with a decomposition of the circuit and then to combine or collapse these simple gates into more complex gates. This basic idea has been used by traditional techniques for technology mapping that used local gate collapsing through pattern matching as a supplementary way to improve the circuit performance has been used in technology mapping. The word "local" refers to the fact that the collapsed gates are constrained to belong to the available cell library in these approaches. In contrast, our approach of global gate collapsing does not tie the list of permissible gates to any specific

library. Our procedure works on a virtual library that is assumed to have all types of cells so that the global gate collapsing technique can have the full flexibility of finding the optimum possible combination of standard gates in a network.

The input to global gate collapsing comes from the output of technology-independent optimization, and the result of the procedure is a network where the input netlist is collapsed into an optimal set of complex gates corresponding to that decomposition. This technique can result in a solution that can be optimized for various objectives such as minimizing the circuit delay or the circuit area, or the power dissipation, etc. In this work, we consider gate collapsing on the circuit using a combination of two forms of logic styles:

- Complex static CMOS gates
- Pass transistor logic (PTL)

To the best of our knowledge, there has been no prior work that solves this problem. A related piece of work on library-less mapping was published in [39] which traverses the circuit and chooses a succession of windows within which to perform local resynthesis operations with sizing. Our technique, in contrast, uses dynamic programming to consider the entire circuit at the same time. Moreover, a unique feature of our work is that we incorporate the use of PTL with the virtual library.

The matching method of traditional technology mappers such as MIS [12] cannot be applied on the virtual library since the number of possible templates is far too large. Therefore, this chapter develops techniques for generating the complex gates for the virtual library. We propose two methods for this purpose:

- Topological mapping: Odd-level transistor replacement (OTR) for mapping to complex static CMOS gates
- Boolean functional mapping: Using binary decision diagrams (BDD's) to map logic to pass transistor logic.

Recently, pass transistor logic has been actively considered as a significant alternative to full static CMOS since it requires a smaller number of transistors. Several publications on

PTL research have been published in the recent past (for example, [62, 73, 86, 94, 95, 100]) showing its viability.

There has been relatively little work on design automation for PTL. We believe that this is among the first published techniques to incorporate pass transistor logic with technology mapping issues. A related body of work work was a recent heuristic approach to logic synthesis for PTL [13, 14]. This procedure is more focussed on the logic synthesis aspects and uses a simple delay model that estimates the delay using the number of transistors on a PTL chain. In contrast, our work targets the technology mapping issue and uses a more sophisticated SPICE-calibrated delay model.

Our approach uses dynamic programming techniques to partition the circuit into PTL segments separated by static CMOS segments. An exhaustive set of SPICE simulations was performed to characterize complex gates and PTL and an accurate look up table was constructed where the gate delay is listed as a function of parameters such as the input signal transition time, the interconnect load, the transistor sizes and position of the switching positions within the gate. We also propose a new technique that is employed to reduce the size of the look-up table and the corresponding memory overhead.

The organization of the chapter is as follows. The first technique for gate collapsing, called the OTR method, is described in Section 1. The method is based on an observation that uses the topological properties of the circuit in collapsing complex gates in a computationally efficient manner. Next, we consider the problem of mixed static/PTL mapping in Section 2 and extend the algorithm from Section 1 to perform gate collapsing in conjunction with static/PTL mapping using the relationship between PTL and BDD's. Experimental results are presented in Section 4, followed by concluding remarks in Section 5.

4.1 Odd-level Transistor Replacement (OTR) Method

4.1.1 An Example

We will now present a method for building complex gates, based on a simple topological technique that permits subcircuits with an odd number of gate levels to be collapsed into a

single complex gate.

The basic idea of the OTR method is to use the pull-down (pull-up) transistor structure from the gates at the previous level gates to replace the pull-up (pull-down) transistors of the gates at the next level. To illustrate this, consider the circuit in Figure 4.2(a) consisting of gates G1 through G7. This structure has 20 transistors in all, and a transistor-level version is shown in Figure 4.2(b).

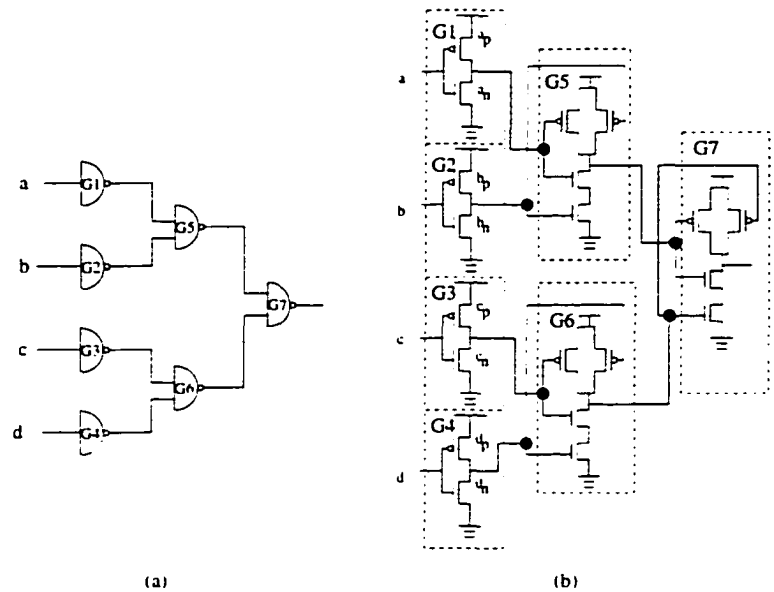


Figure 4.2 A circuit for gate collapsing

We use the pull-down (pull-up) transistors in G1 and G2 to replace the pull-up (pull-down) transistors in G5 to obtain the gate G5', a nontraditional static CMOS gate, shown in Figure 4.3(a). Similarly, the transistors in G3 and G4 are inserted into G6 to get another nontraditional static CMOS gate, G6'. We treat these nontraditional gates as intermediate synthesis stages and we will eliminate them in the next step by performing the same operation, replacing the pull-down (pull-up) block of G7 by the pull-up (pull-down) blocks of the intermediate gates G5' and G6', respectively. The detailed illustration of the final collapsed gate is shown in Figure 4.3(b). Note that the final implementation has only 8 transistors, a transistor count reduction of 60%.

From the principle illustrated in this example, it is easy to see that if we collapse an even number of levels of gates, we will be left with a nontraditional static CMOS gate, whereas if

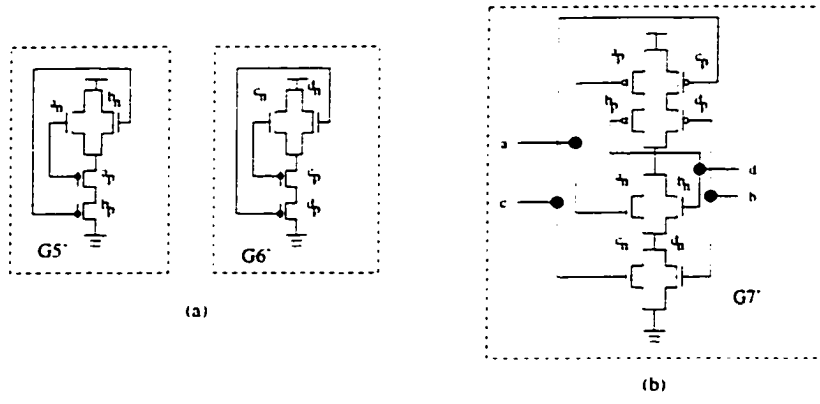


Figure 4.3 The procedure of OTR gate collapsing

we collapse an odd number of levels, we will return to the traditional CMOS complex gate structure, and therefore we call this technique the *odd-level transistor replacement (OTR) method.*

4.1.2 Proof of Logic Correctness

Before beginning this proof, it is important to state that the OTR technique works when the network is entirely specified in terms of inverting gates, as is the case in any CMOS implementation. When the circuit is specified in terms of noninverting gates, the first step would be to convert all noninverting gates into an inverting gate followed by an inverter, and then apply the OTR method.

Theorem:

(1) On completion of the OTR procedure, all gates are transformed into (complex) traditional static CMOS gates.

(2) The OTR method preserves the logic function $f(x_1, x_2, \cdots, x_n)$ *of the original circuit.*

Proof: The first part of this proof is easy to see, since by construction, the pull-up of the final gate consists purely of pMOS transistors and the n part consists purely of nMOS transistors, and each pMOS structure in the pull-up will have a dual nMOS structure in the pull-down. Therefore, the final result will be a traditional static CMOS gate.

We prove the second result on the reduction of a three-level subcircuit to one level. The proof for other odd numbers of levels l can be deduced from this proof in a constructive manner

by applying this procedure to reduce the number of levels in steps to $l-2, l-4, \cdots, 1$.

For the circuit configuration shown in Figure 4.4, we label the levels from 1 to 3 as shown. We will consider the situation when the output of level 3 is at logic 1: the proof for the logic 0 case is analogous.

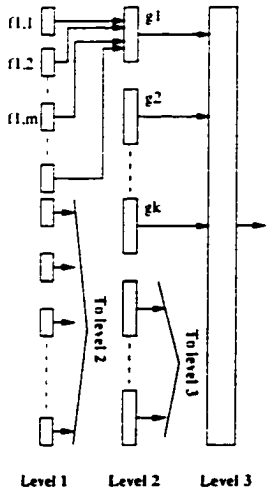


Figure 4.4 Circuit configuration considered in the proof

Since the level 3 output is at logic 1, it implies that there is a set of pMOS transistors that provides at least one pull-up path between V_{dd} and the output node. We will show that under the OTR scheme, the new gate will also have a pull-up path corresponding to each of these pull-up paths.

Without loss of generality, we may consider any one of these paths, P. Before proceeding, we note that in the original circuit, each of the inputs $g_1, g_2, \cdots, g_k$ of P is connected to a pull-down path in the previous level that is connected to ground, i.e.,

$$g_1 \wedge g_2 \wedge \cdots \wedge g_k = 0$$

Each transistor on P is replaced by transistor segments from the nontraditional CMOS gate by applying the OTR procedure, which replaces that transistor with the pull-down of the preceding nontraditional gate.

To show that after modifications, the path P continues to provide a pull-up path between V_{dd} and the output node, it suffices to show that each such preceding nontraditional gate has a path from ground to its output node. We will call this Requirement (Φ).

Consider any such level 2 gate g_i in the original circuit that excites a transistor on path P. If the level 3 output is high, then it must be true that there is a one or more path in gate g_i that connects the output node to ground in the original circuit. Without loss of generality, we consider any one such path Q_i. Then it must be true that if the level 1 gates driving this path are $f_{i,1}, f_{i,2}, \cdots, f_{i,m}$, then

$$f_{i,1} \wedge f_{i,2} \wedge \cdots \wedge f_{i,m} = 1$$

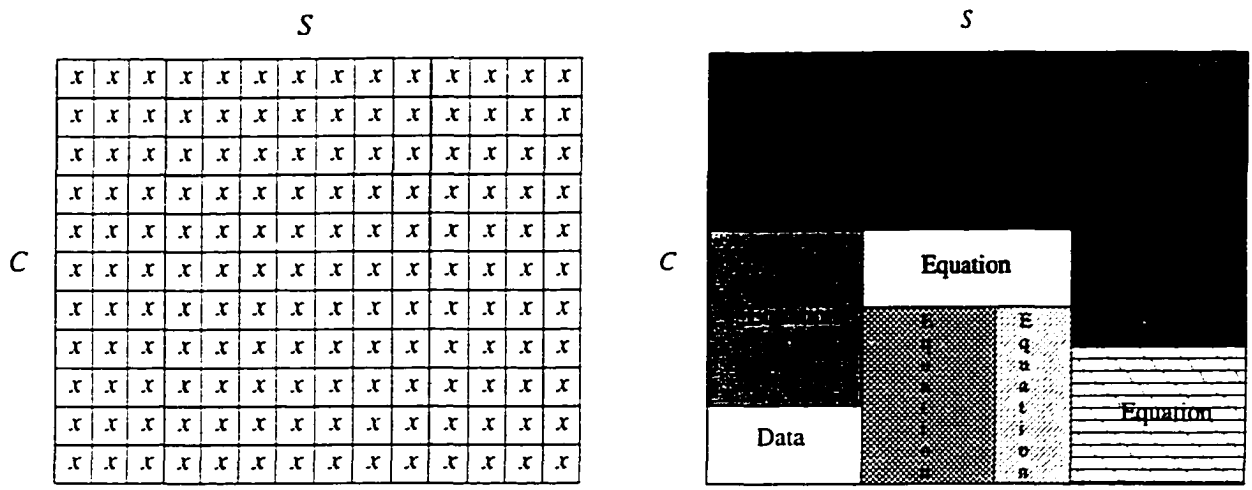
The OTR procedure constructed the nontraditional gate by inserting pull-up stages from level 1 to replace the pull-down transistors on Q_i. Therefore, any conducting pull-up path in a level 1 gate will be conducting pull-down subpath in the nontraditional gate. Therefore, all the subpaths corresponding to $f_{i,1}, \cdots, f_{i,m}$ will be conducting pull-down paths in the nontraditional gate at level 2, and when these are placed in series, we have a conducting pull-down path between output and ground for the nontraditional version of gate g_i at level 2. Thus we have shown Requirement (Φ) and we are done.

4.1.3 Delay Estimation

In this section, we describe the technique used in this work for delay calculation for complex gates, including a new method used to reduce the amount of storage for the look-up table while maintaining accuracy.

The delay is characterized in the look-up table as a function of the switching position, transistor size, input slope S, and loading capacitance C. We assume in our implementation that each transistor in a gate has the same size. This makes layout easy and compacts the size of the look-up table. Some further improvements are possible by allowing transistors to be sized individually. Our experimental results show that even under our implementational assumption, substantial area/performance improvements are possible. Moreover, the theoretical framework presented here can be extended to the case of nonuniform sizes.

Given a switching position and a transistor size, a traditional look-up table is a two-dimensional array of values parameterized by S and C, as shown in Figure 4.5(a). This table requires a large amount of memory which can make the look-up speed slow since it is impossible



(a) Traditional Table (b) Refined Table

Figure 4.5 Look-up tables

to store all such tables for all possible switching positions and transistor sizes in the cache or in the RAM. In order to refine this look-up table method, we compact the information in this table into a delay characteristic equation for each such two-dimensional array. For purposes of characterization, we find a least-squares fit to the characteristic delay equation from [98] which is of the type used by Synopsys:

$$D = \alpha * S + \beta * C + \gamma * S * C + \omega$$

where α, β, γ, ω are constants. However, if we attempt to find a single delay equation for the entire table, the accuracy of the characterization may be poor. Therefore, we use a set of equations that capture the information embedded in a subset of the data, ensuring that the accuracy of each such fit is within a prescribed range, ϵ. The entire data can be fitted accurately to a small set of delay equations, and any data points that have an error larger than ϵ from the set of equations are stored as pure data. The overall structure of the storage is as shown in Figure 4.5(b).

Our experimental results show that we can use the delay equation to represent about 75% of the delay data points by using 4-10 different sets of coefficient for each two-dimensional table within an ϵ error of 5%.

The procedure for finding the values of α, β, γ and ω requires a least-squares minimization

of the following form:

$$\text{minimize } F = \sum_i [(\alpha * S_i + \beta * C_i + \gamma * S_i * C_i + \omega) - D_i]^2$$

where the summation is performed over all SPICE-measured data points i, and C_i, S_i and D_i, respectively, denote the load capacitance, slope and delay corresponding to the i^{th} data point. This unconstrained minimization can be performed by setting the partial derivatives of F with respect to each of the parameters to zero, i.e., $\frac{\partial F}{\partial \alpha} = 0$; $\frac{\partial F}{\partial \beta} = 0$; $\frac{\partial F}{\partial \gamma} = 0$; $\frac{\partial F}{\partial \omega} = 0$.

This leads to a set of linear equations of the following form:

$$a_{11} * \alpha + a_{12} * \beta + a_{13} * \gamma + a_{14} * \omega - b_1 = 0$$

$$a_{21} * \alpha + a_{22} * \beta + a_{23} * \gamma + a_{24} * \omega - b_2 = 0$$

$$a_{31} * \alpha + a_{32} * \beta + a_{33} * \gamma + a_{34} * \omega - b_3 = 0$$

$$a_{41} * \alpha + a_{42} * \beta + a_{43} * \gamma + a_{44} * \omega - b_4 = 0$$

where the values of a_{11} through a_{44} and b_1 through b_4 can be calculated from the function F. We solve the above system of linear equations to find the values of α, β, γ and ω.

4.1.4 Outline of the Algorithm

We now present a dynamic programming based approach to solve the problem of area/power minimization under delay constraints. In Section 5, we show the results of applying this technique to find the minimum delay, but the method can equally well be used to solve the constrained optimization problem.

To understand the difficulty of this problem, we observe that technology mapping, a special case of global gate-collapsing, is known to be NP-complete for directed acyclic graph structures [55]. A technique that has been routinely and successfully used in technology mapping is to decompose a DAG into a set of trees and to perform mapping on those trees (for example, in [17, 31, 55]), with the trees being selected in such a way that they are all rooted at gates with multiple fanouts or at gates at the primary output. We persist with this approach in our work.

The algorithm is based on dynamic programming and uses OTR combinations to generate possible complex gates. As in [17], we begin with a 2-input NAND gate decomposition of the

circuit, though we emphasize that any other initial circuit can also be used. The pseudocode for the algorithm is shown below:

Algorithm Outline

```
Input:Initial circuit decomposed into inverters and 2-input NAND gates.
Output:  Optimum network of complex gates.
{
  levelize the circuit
  find_roots
  sort_roots
  from primary inputs to primary outputs
   for each root generate tree
   apply dynamic programming
   for each node in the tree from leaf nodes to the root
       find_all_possible_collapsing_solutions
       store non_inferior_solutions [Area, Delay]
  find optimum solution based on all generated noninferior states
}
```

An explanation of the pseudocode is as follows. The circuit is first levelized to find the level number for each gate, which is the maximum number of gates between the primary inputs and the gate output. Next, the procedure find_roots is invoked to split the DAG circuit structure into a forest of trees. The function sort_roots then arranges the roots of these trees according their level number. The trees are processed in order of the level number of their roots, thereby ensuring that before each tree is considered, all of its fanin nodes have been processed.

The dynamic programming procedure [27] proceeds by associating a set of states with each node, where a node corresponds to a gate output. A state corresponds to a partial solution that corresponds to a possible configuration of collapsed gates for the subtree rooted at that node. The state information for each node is a pair [Area,Delay], calculated from the primary inputs up to that node. The method can easily be extended to consider measures such as power in this framework. The Area at a node g is given by the sum of the Area of a candidate complex gate with output g, and the node Area for all possible states at the fanin nodes of the current complex gate. The complex gates are chosen so that the number of series-connected

mosfets on a path to V_{dd} or ground does not exceed a user-specified number k.

Dynamic programming proceeds by enumerating the possible states at a node, and eliminating all partial solutions at each step that are provably suboptimal. For example, a state [Area,Delay] is provably inferior if there exists another state [Area',Delay'] such that Area $\geq$ Area' and Delay $\geq$ Delay'. The pruned list of possible states at each node are used as candidate states at the next node, and so on. Under this basic framework, the dynamic programming procedure stores only the noninferior [Area,Delay] combinations and proceeds in a manner that is fundamentally similar to that in [17]. Due to limitations of space, we do not describe any further details here.

Finally, when all noninferior states have been enumerated, the optimal state is chosen and the corresponding circuit configuration is determined. An outline of the computational complexity is provided after the pseudocode for the Static/PTL method described in Section 2.

4.2 Combined Static CMOS/Pass Transistor Logic Design

4.2.1 Fundamentals

Although static CMOS has been a mainstay of circuit design for decades, with increasing performance requirements on circuit in terms of speed and power, there is a conscious attempt to seek design styles with better performance. Several techniques such as dynamic logic and PTL have been proposed in the recent past. In this section, we develop techniques for the synthesis of circuits with a combination of static CMOS and PTL and present a procedure that partitions a circuit into static CMOS and PTL to achieve the minimum delay.

PTL is widely considered to be a promising design style since it can implement most functions using fewer transistors than a static CMOS implementation. This reduces the overall capacitance, resulting in circuits with faster speed and lower power dissipation. The logic style is illustrated in Figure 4.6 which shows a PTL logic segment that realizes the two-input AND function. Only recently has PTL become noticed as a viable design style in its own right, and consequently and there are no mature synthesis tools to realize the advantages of this logic style.

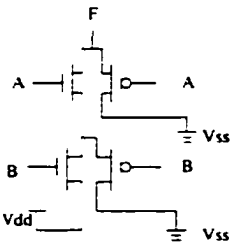


Figure 4.6 Realization of an AND function using PTL logic

In dealing with PTL, a designer must be aware of the following limitations:

(1) For an nMOS (pMOS) transistor, the low-to-high (high-to-low) transition is imperfect and therefore PTL cannot achieve full voltage swings, resulting in reduced noise margins.

(2) It is possible for sneak paths between V_{dd} and ground to exist unless the circuit is designed carefully. An example of sneak paths is shown in Figure 4.7: if $X = 0$ and $Y = 1$ at the same time, then F is connected to connect both power supply and ground simultaneously.

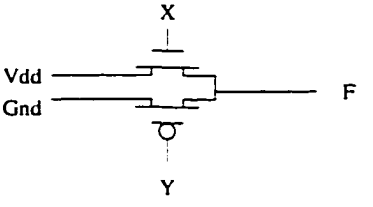


Figure 4.7 An example of a sneak path

Pass transistors can be used to build a 2-input multiplexer, leading to a one-to-one correspondence between BDD's and their PTL implementations. Since a BDD can represent any logic function, we can use the BDD representation to directly arrive at a PTL implementation of a complex gate. In Figure 4.8, we show the correspondence between a BDD node and a pass transistor, build the BDD representation for the 2-input AND gate, and arrive at the pass transistor implementation of the BDD. Figure 4.8(a) shows a BDD node whose PTL implementation is shown in Figure 4.8(b). Using this as a basis for design, we take the BDD in Figure 4.8(c), representing a two-input AND gate, and build the the corresponding PTL implementation as shown in (d). A second example of more complex logic is shown in Figure 4.9.

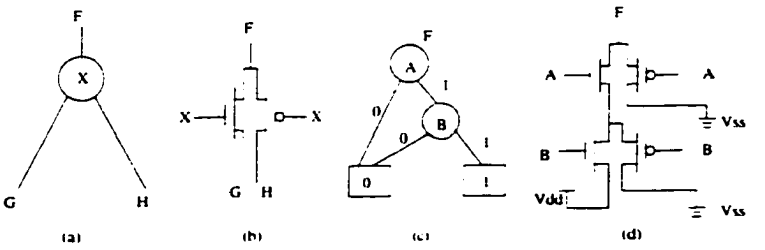


Figure 4.8 Circuit example 1

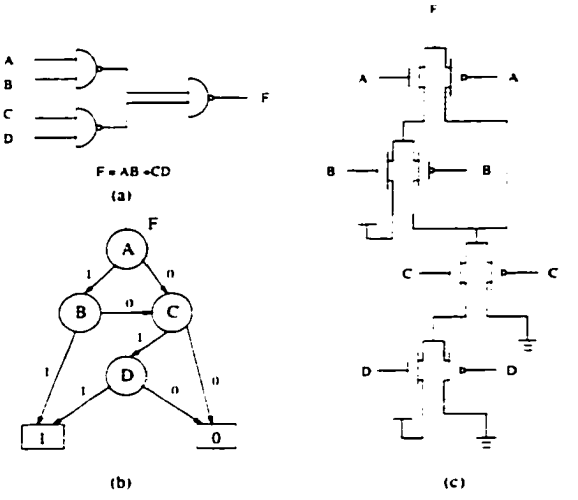


Figure 4.9 Circuit example 2

4.2.2 Fundamental Pass Transistor Cell Selection

In order to implement a BDD node using a 2-input multiplexer-like pass transistor. a suitable choice of the fundamental pass transistor cell must be made. As pointed out in [13], there are two possible types of fundamental pass-transistor units, as shown in Figure 4.10(a) and (b). The first uses a pair of nMOS pass transistors. while the other utilizes an nMOS transistor and a pMOS transistor. While the worst case noise immunity of the first configuration is better than that of the second, it requires the generation of complementary signals at the gate

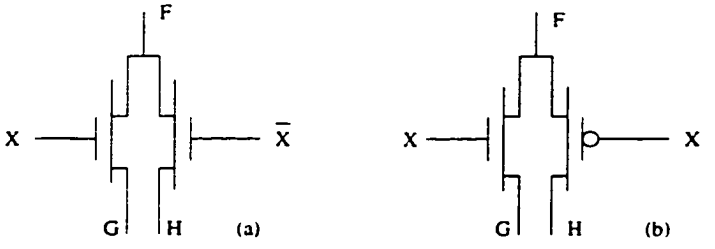


Figure 4.10 Two types of PTL units

inputs. which results in an extra area overhead. Moreover, the extra delay in generating the complement could lead to a sneak path. which could result in a larger power dissipation. In this work, we choose a fundamental cell with one nMOS and one pMOS transistor: however. the work can be extended to handle the other configuration too.

4.2.3 Outline of the Algorithm

The dynamic programming approach is also used to give us a technique for building mixed static/PTL circuits. The basic idea is to use BDD's to represent a candidate logic function that can be implemented in PTL during dynamic programming. The implementation uses the BDD package described in [10].

In using PTL, as in the case of complex static gates. we must ensure that the number of pass transistors in series should be no more than a predetermined number p. In other words. while generating BDD's. we do not permit the depth of the BDD to become larger than p. and at that point, we force the use of a static gate at the fanout. The final circuit is likely to contain pieces of pass transistor logic that are isolated from each other by static CMOS gates The purpose of the static CMOS gates is to isolate the pass-transistor parts and to restore the level of degraded signals.

The dynamic programming approach here is used to determine how the circuit should be partitioned between static CMOS and PTL implementations. In our current implementation. we use a PTL look-up table delay model that is similar to that used for the static CMOS logic. with the coefficients of the delay characteristic equations being altered.

As in Section 1.4, and for the same reasons, the algorithm begins by decomposing the circuit into a forest of trees. For each such tree, we perform mapping by dynamic programming in a manner similar to that described in Section 1.4 to implement the design using mixed static/PTL logic. Since the threshold value p a small number, it is computationally inexpensive to generate BDD's in the fanin cone up to a BDD depth of p. Moreover, the number of possibilities for mapping a node either into a complex gate with a bounded number, k, of series-connected mosfets. or as PTL with a bounded p value, is finite and small. Therefore, the computation is fast. The dynamic programming approach is guaranteed to find the optimal solution of mixed

static/PTL circuits for tree structures. For DAG structures, since the approach uses techniques that have worked well for technology mapping, we expect the results to be near-optimal for this problem too, and this as our experimental results show, the procedure leads to sensible designs.

The dynamic programming procedure is similar to the OTR algorithm described earlier and is represented by the following pseudocode:

Algorithm Outline

```
Input:Initial circuit decomposed into inverters and 2-input NAND gates.
Output:  Optimum mixed PTL/static CMOS gate network.
{
  levelize the circuit
  find_roots
  sort_roots
  from primary inputs to primary outputs
   for each root generate tree
       for each node in the tree from leaves to the root
           apply dynamic programming procedure
           find maximum fanin cone
           generate all possible BDDs inside the maximum fanin cone
                to generate PTL solutions
           find_all_possible_collapsing_solutions
           store_non_inferior_solutions [Area,delay]
  find optimum solution of the primary outputs
}
```

The chief difference between the OTR approach and this approach is that we maintain BDD representations for all possible candidate PTL implementations: as mentioned earlier, due to the limitation on the number of series-connected PTL transistors, these BDD's operate within a maximum fanin cone and are typically small. We compute the possible states of a node g, allowing the possibility of a static CMOS gate, or a PTL implementation using the BDD representation, and calculate the Area and Delay for every candidate state. As before, each state corresponds to an [Area,Delay] combination, and only the noninferior states are stored. Finally, when all noninferior states have been enumerated, the optimal state is chosen and the corresponding circuit configuration is determined.

While the nature of dynamic programming makes it inherently difficult to arrive at an accurate measure of the computational complexity, it is worthwhile to attempt an estimate of the complexity. For both the OTR and the static/PTL methods, we need to build complex gates, either in static form or as PTL. Suppose that for each node, it is possible to build C possible complex gates, that a complex gate can have a maximum of I inputs, and that each node can have up to M [Area, Delay] pairs stored during dynamic programming. Therefore, for each node, the amount of computation for calculating the [Area, Delay] pairs is $O(C \cdot I \cdot M)$. In general, C and I are bounded, and so the computation complexity can be written as $O(M)$. Since the dynamic programming technique handles each of the N gates in the circuit, the computation complexity of our algorithm is $O(N \cdot M)$.

4.3 Experimental Results

The methods described in this chapter were both implemented in C on a SUN Sparc 1/170 workstation. For purposes of comparison, results were generated using SIS [89]. OTR (Section 1) and mixed-static CMOS/PTL methods (Section 2) on the ISCAS'85 benchmark circuits. The circuits were first decomposed into inverters and 2-input nand gates network using SIS. Next, we performed a minimum circuit delay technology mapping in SIS for the circuits using the libraries nand-nor.genlib, mcnc.genlib and lib2.genlib. We set the the values of the parameters k and p (described in Sections 1.4 and 2.3, respectively) to 4 in our work.

Our OTR results and SIS results on these three libraries are shown together in Table 4.1 for various circuits, and a comparison of the Static CMOS/PTL results and SIS results on these three libraries is shown in Table 4.2 for the same circuits. In each table, column 1 shows the circuit name; columns 2-4 show, respectively, the minimum delay, the corresponding area, and the CPU time of the SIS mapping results for each circuit on the nan-nor.genlib library. The same information is then shown for the SIS mapping results for each circuit on the mcnc.genlib library are shown in columns 5-7, and for the lib2.genlib library in columns 8-10, and finally, for our OTR (Static CMOS/PTL) method results in columns 11-13. The last line shows the average improvements of delay and area.

A comparison of the results of SIS and OTR. shows that that OTR provides better results than SIS results. with average delay reductions of over 40%. and area reductions of around 10%. A comparison of Static CMOS/PTL results with the results of SIS and OTR show much greater performance enhancements from the use of PTL in the circuits. The average delay reduction is above 50% and the area reduction above 60% over the results of SIS.

The reason for these improvements is that in the above modestly sized SIS libraries. none of the complex gates that have four transistors in series in the pull-up/pull-down path. while in our work, we have the flexibility of choosing *any* complex gate with up to four series transistors. In contrast, a library that contained all gates with up to four parallel chains. each with up to four series transistors. would require the characterizatien of 3503 gates [31]. A second major improvement is brought about in our work by the use of PTL in conjunction with complex static CMOS gates from the virtual library.

These results are indicative of the the power of our technique. and it is important to note that SIS simply cannot work in our new design methodology because as it cannot work on a virtual library. and requires all allowable gates to be listed and characterized in the library, which could be a prohibitive overhead. Our methods are fast and the largest ISCAS85 circuit can be handled in minutes.

4.4 Conclusion

We have presented the idea of global gate collapsing for pure static CMOS designs. and of using BDD's to realize mixed technology design using a combination of static CMOS and PTL. Our goal has been to present a general technique for performing overall circuit optimization using purely topological and Boolean functional techniques for static CMOS and small BDD's for PTL. The OTR method is fast and simple and avoids the intractable problems in technology mapping, such as matching and covering. The use of PTL is a powerful technique to reduce the area and power dissipation of the circuit. The results obtained show that both techniques are very fast and show significant improvements over existing approaches.

Table 4.1 Experimental Results of SIS and OTR Methods

Circuit	nand-nor.genlib			mcnc.genlib			lib2.genlib			OTR method		
	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)
C432	55.15	3960	5.0	52.35	3768	9.5	47.17	3880	7.8	24.53	3354	4.86
C499	44.23	8488	9.5	40.10	7472	15.6	36.47	7784	13.9	24.15	6912	7.93
C880	37.79	5992	8.0	37.15	5668	12.4	34.58	5104	11.2	24.08	5573	6.99
C1355	46.91	9584	12.5	44.90	9600	20.1	42.97	9952	17.9	24.99	7392	7.87
C1908	61.71	11232	14.4	56.49	10736	25.7	54.38	12260	22.4	30.07	11004	13.49
C2670	60.59	18650	20.2	53.28	16720	35.4	46.84	16480	35.5	30.41	16098	22.29
C3540	79.36	21816	27.2	74.44	24180	55.8	67.56	24800	44.7	52.36	21600	29.30
C5315	74.03	41870	43.3	68.40	39900	86.7	64.84	36790	74.6	35.50	35670	60.31
C6288	208.78	38416	58.6	206.92	38544	90.6	205.79	38528	80.2	100.66	28992	23.83
C7552	72.04	46048	68.7	61.94	43904	242.5	58.95	41176	159.7	28.96	43896	70.02
Avg. Imp.	48.3%	12.4%		44.4%	9.05%		40.3%	8.79%				

Table 4.2 Experimental Results of SIS and PTL Methods

Circuit	nand-nor.genlib			mcnc.genlib			lib2.genlib			Static CMOS/PTL method		
	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)	Min. Delay (ns)	Area (unit)	CPU Time (s)
C432	55.15	3960	5.0	52.35	3768	9.5	47.17	3880	7.8	19.24	1372	190.90
C499	44.23	8488	9.5	40.10	7472	15.6	36.47	7784	13.9	19.50	3098	196.66
C880	37.79	5992	8.0	37.15	5668	12.4	34.58	5104	11.2	19.18	2432	168.50
C1355	46.91	9584	12.5	44.90	9600	20.1	42.97	9952	17.9	20.77	3450	376.49
C1908	61.71	11232	14.4	56.49	10736	25.7	54.38	12260	22.4	25.28	4753	296.05
C2670	60.59	18650	20.2	53.28	16720	35.4	46.84	16480	35.5	26.75	5980	621.61
C3540	79.36	21816	27.2	74.44	24180	55.8	67.56	24800	44.7	36.20	9059	887.46
C5315	74.03	41870	43.3	68.40	39900	86.7	64.84	36790	74.6	25.92	11601	990.18
C6288	208.78	38416	58.6	206.92	38544	90.6	205.79	38528	80.2	88.33	14403	1306.02
C7552	72.04	46048	68.7	61.94	43904	242.5	58.95	41176	159.7	21.74	18350	1093.61
Avg. Imp.	58.8%	63.1%		55.7%	61.8%		52.5%	61.7%				

5 A NEW IDEA OF COMBINING PLACEMENT WITH TECHNOLOGY MAPPING

CMOS processes have shrunk, and more automated design tools have become commonplace, leading to far more complex chips operating at much higher speed than decades ago. Design technologies must be advanced for design flows, methodologies, tools. Several things in the next generation VLSI CAD tools need to be emphasized.

- Dynamic library, which allows the circuit designer to explore the whole design space.
- Interconnect wire, which should be considered at the technology mapping level.
- Merging technology mapping with placement design.

Dynamic Library: For decades, all the automatic designs have been based on the libraries. The libraries indeed make the automatic design successfully. With the requirement of the high performance increasing, the limitation of the library based design becomes apparent. The library limits the circuit designer to explore the design space. The circuit designers hope that there are dynamic libraries which are generated on the fly can make them to explore the whole design space. The mature techniques of the layout synthesis make the dynamic libraries possible. The prerequisite of the application of the dynamic libraries is to build the complex gates. In our work, we used global gate collapsing technique to generate the complex gates. Our approach of global gate collapsing does not tie the list of permissible gates to any specific library. Our procedure works on a virtual library that is assumed to have all types of cells so that the global gate collapsing technique can have the full flexibility of finding the optimum possible combination of standard gates in a network. The input to global gate collapsing comes from the output of technology-independent optimization, and the result of the procedure is

a network where the input net list is collapsed into an optimal set of complex gates corresponding to that decomposition. This technique can result in a solution that can be optimized for various objectives such as minimizing the circuit delay or the circuit area, or the power dissipation, etc..

Interconnect Wire: As the percentage of interconnect wires delay to the total circuit delay increasing, the incorporation of the interconnect wire into every level of VLSI design is becoming more and more important. We calculated the interconnect wire information at the technology mapping level when we did gate collapsing. In our work, we used π model to capture the delay of the interconnect wire. By considering the interconnect wire effect at the technology mapping level and using π model, we can accurately capture the interconnect wire issue during the circuit design.

Merging Technology Mapping and Placement Design: The traditional "divide-and-conquer" strategy has been the main design methodology for decades. In this strategy, technology mapping and physical layout design are separated. The move to deep sub-micron technologies where interconnect delays dominate makes this methodology not suitable for the high performance design. With the increasing dominating delay of the interconnect wire, the value of stand alone technology mapping and place and route is undermined until it is of zero value. The technology mapping integrating the estimation and physical layout information will be crucial to the success of tomorrow's CAD flows. Without the physical layout information, the optimal solution at the technology mapping stage will not remain optimal after the physical layout design. In physical layout design stage, the gate level net list is fixed. It has no flexibility to do logic optimization. It is the trend to merge the technology mapping and physical layout stages. Therefore, the merging technology mapping and physical layout design is one of the key parts of the next generation VLSI CAD tools.

There are several research work on the merging technology mapping and physical layout design procedure. One trend is to do the technology mapping procedure based on the estimated the placement information [17, 74]. This kind of method is called incremental placement. Other trend is to do the logic resynthesis based on the placement information [39]. This kind

of method is called re-mapping.

In our work, we truly merge the gate collapsing and placement procedures. It does the logic optimization considering the placement information. The nice property of this procedure is that the optimal result at the technology mapping level is the optimal result at the physical level.

Figure 5.1 is a diagram of the evolution of the merging technology mapping and physical layout design and our method. The first column is the traditional "divide-and-conquer" strategy. The second column are the two trends of incremental placement and re-mapping. The third column is our work.

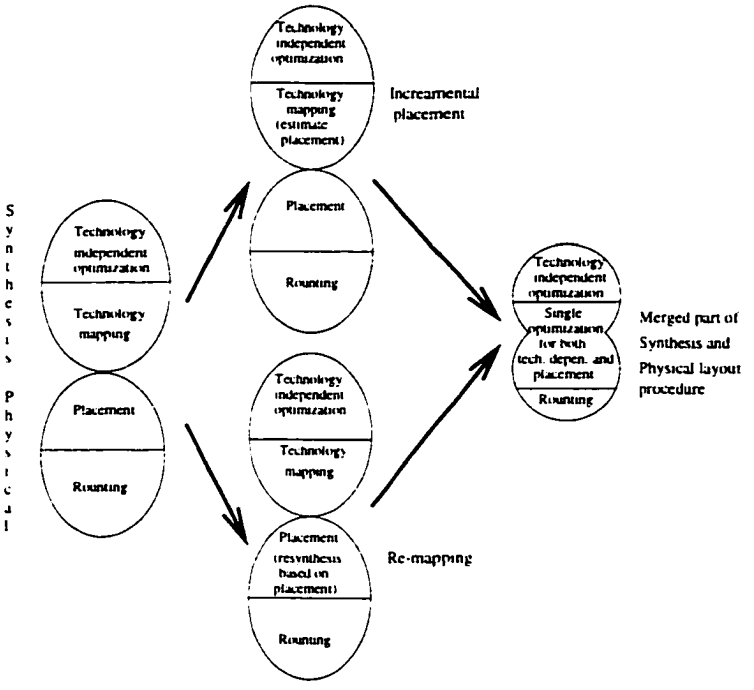


Figure 5.1 The cases of merging technology mapping and placement

Incremental placement is faster but not accurate. Therefore, we proposed our idea of merging technology mapping and placement procedure by doing gate collapsing and placement simultaneously. The final result will be the optimal results of placement with considering the technology optimization optimization.

The method of re-mapping based on placement is also the separated procedure of the technology mapping and placement design. The basic idea of integrating the placement is to

find wire information to calculate the delay.

Our method is better than the "re-mapping" based on placement data, and the "incremental placement" based on technology mapping result.

5.1 Dynamic Library

Our global gate collapsing technique [51] is used to break through the library limitation at technology mapping stage. It can dynamically produce the complex cells on the fly. There are several ways for building the complex gates, i.e., functionally or structurally. We use gate collapsing for building complex gates, based on a simple topological technique that permits subcircuits with an odd number of gate levels to be collapsed into a single complex gate. Our method is a simple and effective way to build a dynamic library.

The detailed information of the gate collapsing technique was presented in Chapter 4.

The work [51] did not consider the interconnect wire information. If considering the interconnect wire information, some gates may not be collapsed together. Those gates are preferred to be separately to act as repeaters than collapsing them together. Figure 5.2 shows an example. Suppose we assume the delay is RC delay and R is proportional to the length of the interconnect wire, and C is also proportional to the length of the interconnect wire. If the length of the interconnect wire is L, the delay is proportional to L^2. In other case, if we did not collapsing gates B and C, gate B divides the wire into two parts with length l_1 and l_2, respectively. The delay from A to C is proportional to $l_1^2 + l_2^2$. In general, the delay (L^2) of the wire with the length L is smaller than the summation delays ($l_1^2 + l_2^2 + \cdots + l_n^2$) of the segments ($l_1, l_2, \cdots, l_n$) of L. Because if $L = l_1 + l_2 + \cdots + l_n$, then $L^2 < l_1^2 + l_2^2 + \cdots + l_n^2$.

5.2 Interconnect Wire Model

For devices, we found the worst case delay, i.e., the max number of transistors in serial. The device is modeled as R and $C_{intrinsic}$. For interconnect wire, we use π model to formulate them. Figure 5.3 illustrates the π model. For an interconnect wire with the length l, it can be modeled as a resistance $R_{unit} * l$ connected by two grounded capacitance $\frac{C_{unit} * l}{2}$. R_{unit}

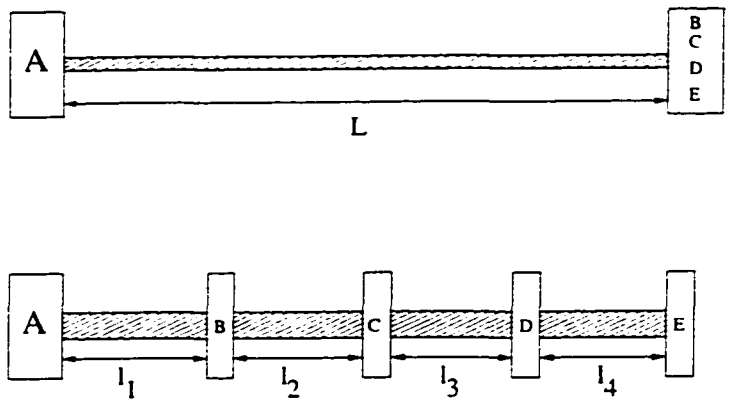


Figure 5.2 An example of preferred not collapsing

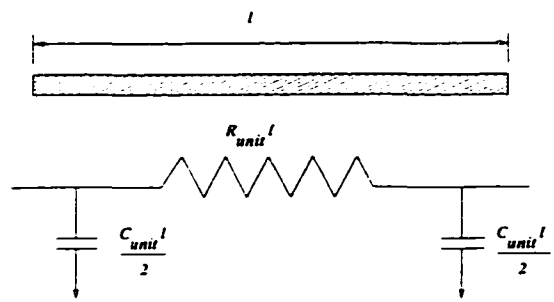


Figure 5.3 π model for interconnect wire

and C_{unit} are interconnect wire unit length resistance and capacitance, respectively. It has be shown that π model is an accurate delay model [30].

5.3 Merging Technology Mapping and Placement Design

It is the trend for merging technology mapping and physical layout design. Unfortunately, merging technology mapping, placement and routing together is a difficult task in engineering and theory. In our work, we try to merge technology mapping and placement into one step design, not considering routing problem for the time being.

In order to merge technology mapping and placement, we need consider the methods for the technology mapping and placement. We use gate collapsing technique to do technology mapping optimization. We introduce the placement method in detail in the following.

5.3.1 Placement Formulation

Our placement tool is based on [58]. The detailed information is presented in Chapter 2. Here, we will not repeat it.

5.3.2 Outline of the Algorithm

In our work, gate collapsing technique is used for logic optimization, therefore, we can find all the possible collapsed complex gates for a given circuit. For every possible collapsed complex gate, there are two choices for them, i.e., to be collapsed or not. We denote the collapse as 1, no collapse as 0. For N possibilities, in theory, there are total 2^N configurations. The computation is expensive. Therefore, we resort to a heuristic method to solve this problem. We start at $\underbrace{(0,0,0,\cdots,0)}_{N}$ configuration, then we perform N iterations. In ith iteration, we collapse the ith possible complex gate, i.e., changing the ith item value in the configuration from 0 to 1. We get a new configuration, then we compare the new configuration with the best configuration so far. If the new configuration is better, then we update the best configuration, otherwise, we keep the best configuration and continue to the $(i+1)$th iteration. The design flow chart is shown in Figure 5.4.

The pseudo code is like the following:

```
Input:  gate-level netlist
do placement
best_placement = placement (0000000 ··· 0000)
find all possible collapse of the gates
foreach possible collapse
{
        collapse it (0000 ··· 10 ··· 000)
        do placement
        compare new placement with best_placement
        if new placement is better than best_placement
        then best_placement = new placement
        else keep best_placement
}
```

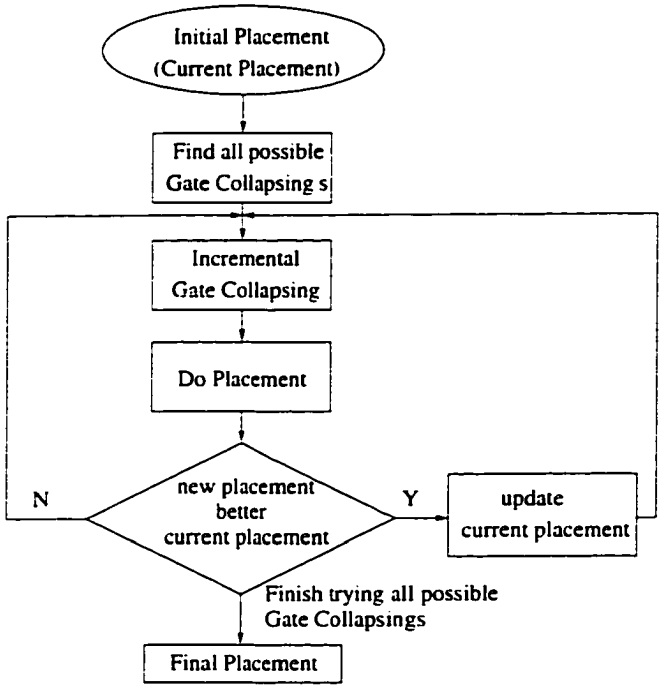


Figure 5.4 Algorithm flow chart

The final pattern of the configuration will be (0/1.0/1.....0/1). The above procedure only takes N iterations.

The complexity of the algorithm is as followings: For placement, the solution of each iteration by using conjugate gradient method is $O(n)$, n is the number of the gates in the circuit, and m is the number of iterations. Because of the partition, the partition at most can go $\lg n$ levels. Therefore the complexity of the placement engine is $O(mn \lg n)$. There are at most n possible gate collapsing, therefore, the total complexity is $O(mn^2 \lg n)$.

5.4 Other Heuristics

For the above design flow (heuristic 1), we first find all possible gate collapsing, then inside the loop, we collapse the possible gates one by one. In our work, we also try another heuristic method (heuristic 2). The design flow of heuristic 2 is shown in Figure 5.5. In this heuristic method, we re-run gate collapsing procedure with fixed previous collapsed gates and also we collapse one gate at a time.

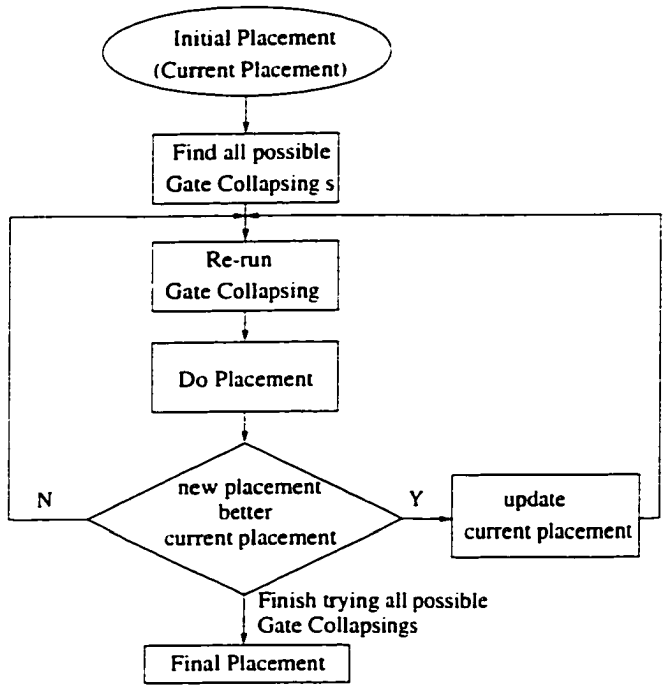


Figure 5.5 Another flow chart

5.5 Experimental Results

We implemented our algorithm in C on a SUN Sparc 1/170 workstation. We run our algorithm on the ISCAS'85 benchmark circuits. For the comparison, we run all the three heuristics and the method of performing gate collapsing and placement separately. The method of performing gate collapsing and placement separately is that we first collapse all the possible gates (which is equivalent to $111\cdots111$ case.) then we do placement.

Table 5.1 and Table 5.2 show the results of the heuristics and the separated gate collapsing and placement method. The first column of the table is the circuits' name, the $2nd$ and $3rd$ columns are the minimum delay and CPU time for a heuristic method, the $4th$ and $5th$ column are the minimum delay and CPU time for the separated gate collapsing and placement design method, the 6th column are the delay improvement of the heuristic method over the separated method. We used the $0.5u$ technology parameters from MCNC as our R_{unit} and C_{unit} etc.. In the future, we will select a more efficient gate collapsing order for the possible collapsing gates and hope it will give us better results.

From the results, we can see the delay improvement by using the heuristic 1 method is

Table 5.1 Results of Heuristic 1 and Separated Methods

Circuit	Heuristic 1		Separated Method		Delay Improvement
	Min. Delay (ns)	CPU (s)	Min. Delay (ns)	CPU (s)	
C432	8.03	14.50	18.99	6.33	57.71%
C499	8.05	100.30	9.92	7.45	18.85%
C880	3.38	80.60	3.67	9.68	7.90%
C1355	6.11	255.13	24.87	11.02	75.43%
C1908	94.55	426.72	438.11	20.14	78.42%
C2670	239.04	1271.40	335.44	34.12	28.74%
C3540	347.17	2015.50	627.15	48.45	44.64%
C5315	2110.20	8451.96	3551.37	124.24	40.58%

Table 5.2 Results of Heuristic 2 and Separated Methods

Circuit	Heuristic 2		Separated Method		Delay Improvement
	Min. Delay (ns)	CPU (s)	Min. Delay (ns)	CPU (s)	
C432	7.56	16.71	18.99	6.33	60.19%
C499	8.65	138.08	9.92	7.45	12.80%
C880	4.00	76.57	3.67	9.68	4.90%
C1355	6.24	260.34	24.87	11.02	74.91%
C1908	100.81	424.04	438.11	20.14	76.99%
C2670	281.93	1260.97	335.44	34.12	15.95%
C3540	421.98	2007.82	627.15	48.45	32.71%
C5315	3017.76	8383.42	3551.37	124.24	15.03%

28.19% on average, 78.42% on maximum. by using heuristic 2 method is 23.48% on average. 76.99% on maximum. We should notice that although heuristic 3 does not get better results for two small circuits, it work well for most circuits and very fast. The nonimproved results of the two small circuits may be caused by selecting M or K. The sophisticated strategy of selecting M and K will take care of it in the future.

Figure 5.6 and Figure 5.7 show the delay vs. iteration number curves for the circuits CC2670 and CC3540 respectively.

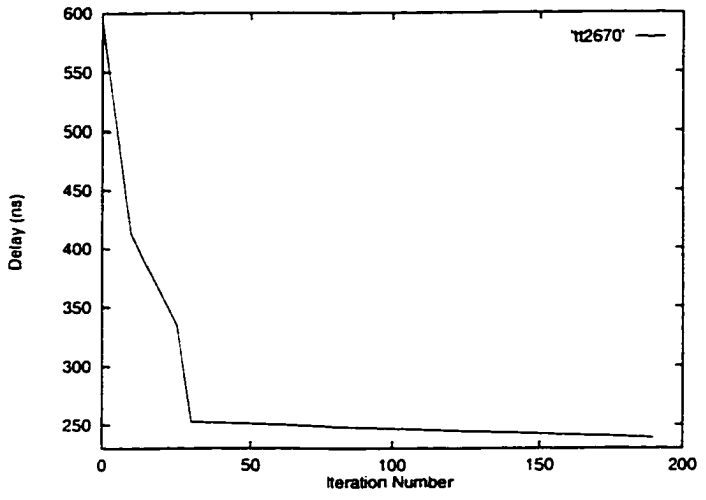


Figure 5.6 CC2670 results

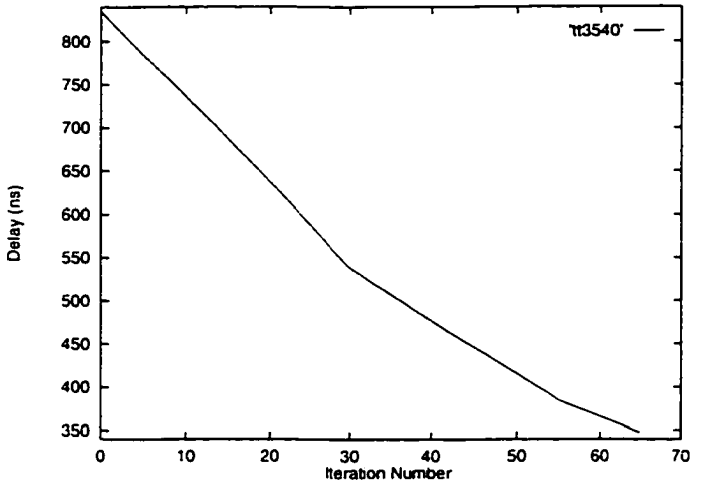


Figure 5.7 CC3540 results

5.6 Conclusion

We proposed our strategies to deal with the three crucial issues for the next generation VLSI CAD tools: dynamic library, interconnect wire and merging technology mapping and placement design. The results are encouraging. To the best of our knowledge, this is the first step to consider the three crucial issues into a single algorithm. We firmly believe that more and more research work will be emphasized in this direction.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

In this thesis, we presented three timing optimization techniques at gate level and transistor level and a new design methodology for the next generation VLSI CAD tools which takes care of the tree crucial issues for tomorrow's CAD tools: dynamic library, interconnect wire and merged logic synthesis and physical layout design.

In Chapter 3, we presented strategies to insert buffers in a circuit, combined with gate sizing, to achieve better power-delay and area-delay tradeoffs, because performing transistor sizing without buffer insertion results in some transistors in the circuit extremely large or performing buffer insertion without transistor sizing results in suboptimal results. Until sizing is performed, any information on capacitive loads is incomplete and therefore a buffer insertion algorithm must operate with incomplete information, leading to suboptimal results. Moreover, the insertion of buffers can change the structure of the circuit sufficiently so that it may lead to a different sizing solution from the unbuffered circuit. Therefore, these techniques of buffer insertion and sizing are intimately linked and it makes a lot of sense to integrate them into a single optimization.

The purpose of this work is to examine how combining sizing algorithm with buffer insertion will help us achieve better area-delay or power-delay tradeoffs, and to determine where and when to insert buffers in a circuit. The delay model incorporates placement-based information and the effect of input slew rates on gate delays. The results obtained by using the new method are significantly better than the results given by merely using a TILOS-like gate sizing algorithm alone, as is illustrated by several area-delay tradeoff curves shown in this paper.

In Chapter 4, a new design methodology of mapping circuits is discussed. It proposes

two new techniques for mapping circuits. The first method, known as the odd-level transistor replacement (OTR) method, has a goal that is similar to that of technology mapping, but without the restriction of a fixed library size, and maps a circuit to a virtual library of complex static CMOS gates. The second technique, the Static/PTL method, uses a mix of static CMOS and pass transistor logic (PTL) to realize the circuit, extending the OTR method and using the relation between PTL and binary decision diagrams. The methods are very efficient and can handle all of the ISCAS85 benchmark circuits in minutes. A comparison of the results with traditional technology mapping using SIS on different libraries shows an average delay reduction above 40% for OTR, and an average delay reduction above 50% for the Static/PTL method.

In Chapter 5, we emphasized three crucial issues in the next generation VLSI CAD tools: dynamic library, interconnect wire and combining placement with technology mapping. Global gate collapsing technique was used for the dynamic library issue. For the interconnect wire, we used π model to calculate the delay. We proposed a new idea of combining placement with technology mapping. The merged procedure does the technology mapping and placement level optimization simultaneously. The final results of the merged algorithm is the optimized results of the placement level.

This thesis involves heuristic method and mathematical methods such as: extrapolation technique, dynamic programming, quadratic program and conjugate gradient method. This thesis is also the application of electrical engineering and computer science knowledge.

6.2 Future Work

Timing optimization has raised more and more people's attention and a significant amount research has been done on it. However, many promising areas for research are still almost untouched or need to be addressed better and better. We now present some of these issues.

6.2.1 Combined Transistor Sizing and Buffer Insertion

Device Delay Model: In order to get more accurate delay model than Elmore delay model, we need to refine the device model in the future. Although we may use the device

models [38, 61, 70] to improve the delay model, however, they are not accurate in the deep sub-micron design. We need to define the new device model for the more accuracy. The new device delay model should consider the parasitic capacitance inside the gate and the transition order of the inputs of the gates. We may use the look-up table method with the analytical equations of the device to deal with this topic.

Repeater Insertion: Now in our work, we assume the buffer is inserted at the beginning of the interconnect wire. To get better results, the buffer insertion should be repeater insertion, i.e., it should be inserted in the optimal position of the interconnect wire. Some initial works have been addressed in [3, 33, 40, 64] without considering transistor sizing.

Simultaneous Transistor Sizing, Buffer Insertion and Wire Sizing: Transistor sizing, buffer insertion and interconnect wire sizing are the effective optimization techniques for tuning the circuits. The global optimal solution of the circuit should be the results of simultaneously performing transistor sizing, buffer insertion and interconnect wire sizing. The work [25] is about the wiresizing under Elmore delay. [64] does the buffer insertion and wire sizing. [19, 26] consider simultaneous buffer and wire sizing. [21, 71] consider simultaneous transistor and interconnect sizing. So far, no published work about simultaneous transistor sizing, buffer insertion and wire sizing.

6.2.2 Gate Collapsing and Mixed Static CMOS and Pass-transistor Design

Decomposition Technique: In technology mapping, how decomposition is done can have a significant impact on the design quality of the final implementation. For the same circuit, different decompositions will result in different results. Figure 6.1 shows two ways of 4-input $NAND$ gates decomposition.

[101] does gate decomposition technique for low power design. [24] considers the gate decomposition for the FPGA design. None of the above techniques can be directly applied to our decomposition for gate collapsing because those techniques do not target at gate collapsing based on odd-level transistor replacement. Now, our odd-level transistor replacement method depends on the input circuit decomposed into inverters and 2-input *nand* gates. In the future, we will find a decomposition which is the best for the odd-level transistor replacement method.

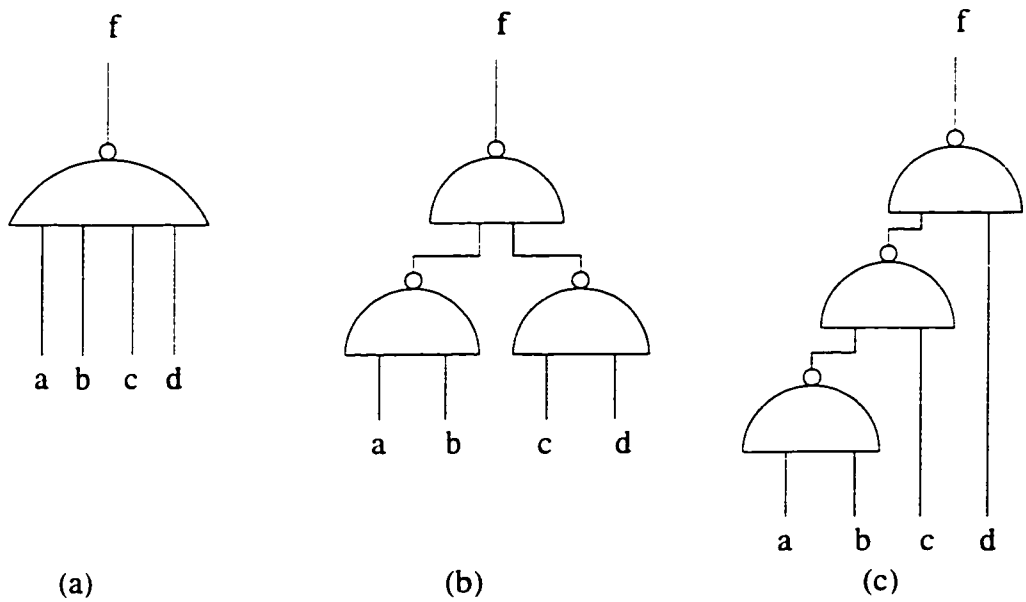


Figure 6.1 Decomposition example

Pass-transistor Reordering: Traditionally, there are two ways for detemining best transistor ordering at each gate: exhausitive search, i.e., enumerate all possible permutations [42, 77] and heuristic search [16]. Although those ideas can be applied to our pass-transistor reordering, however, they don't take the advantage of the one-to-one correspondence between a BDD node and a pass-transistor in our work. In our mixed static CMOS and pass-transistor design, we used BDD to represent the Boolean functions of the circuit and mapped each BDD node by a pass-transistor unit. Therefore, manipulating the BDD variables is equivalent to manipulating the pass-transistors. In our work, we can do the BDD and transistor reorderings by single step at the BDD generating phase for the representation of the functions. BDD variables reordering, i.e. pass-transistor reordering, can achieve better results. Figure 6.2 shows an example of reordering. If transistor A has low transition activity and transistors B and C have high transition activities, so, the configuration in the bottom of the figure has lower power consumption than that of the top configuration.

6.2.3 A New Idea of Combining Placement with Technology Mapping

Interconnect Wire: With the technology advent, interconnect wire has significant impact on performance, functionality and reliability. Interconnect effects must be accurately modeled.

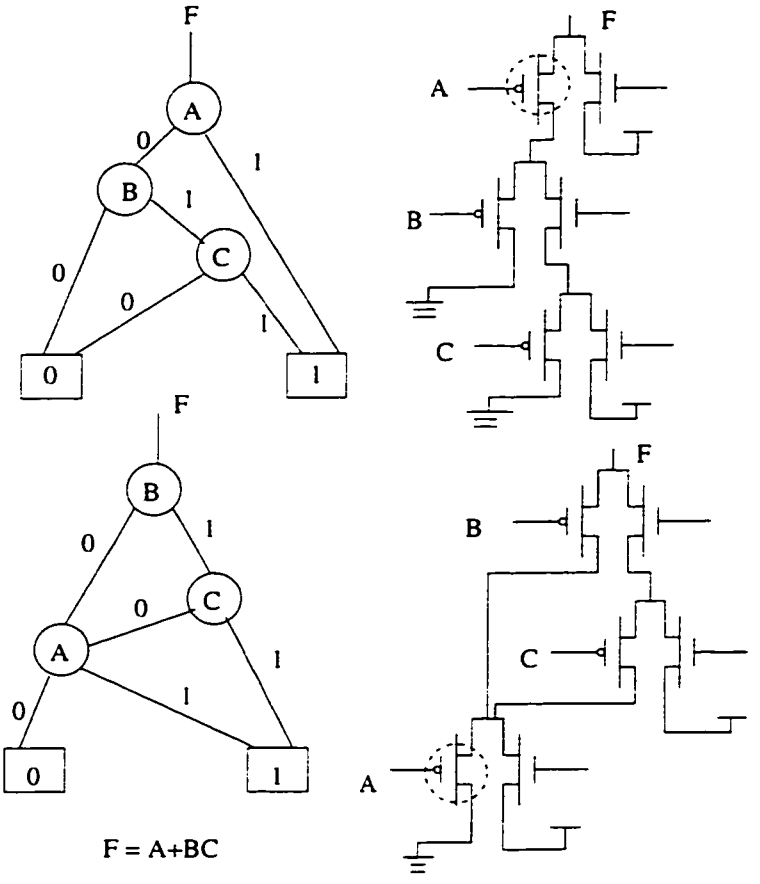


Figure 6.2 Reordering example

Currently, we use Elmore delay model for interconnects. The advantages of Elmore delay model for interconnect are that: it has simple closed form expression; it is the upper bound of 50% delay [44]; it is high fidelity [8, 22], i.e., good solution under Elmore delay are good solution under actural (SPICE) delay. The disadvantages are that: It is low accuracy; it inherently can not handle inductance effect. We should consider the interconnect wire as a transmission line and should consider the inductance of the interconnect wire. We can use Asymptotic Waveform Evaluation (AWE) [75] technique to model the interconnect wires. Voltage drop (IR drop), electronimmgration and signal integrity on the interconnect wire also should be addressed.

Futher Integrating Logic Synthesis and Physical Layout Design: All the work for integrating logic synthesis have focused on the integration of placement with technolody mapping [39, 74] so far. Recently, [82] integrates the floorplanning into the work of integration of placement with technology mapping. It is really a hard problem for entirely integrating logic

synthesis and physical layout design. There are two directions to do that. One is going up. i.e., based on the integration of placement with technology mapping. the integration goes up to floorplanning or technology independent optimization stage. The other is going down. i.e.. integrating routing procedure into the work of the integration of placement with technology mapping.

BIBLIOGRAPHY

[1] S. A. Aftab and M. A. Styblinski. "A New Efficient Approach to Statistical Delay Modeling of CMOS Digital Combinational Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 200–203, 1994.

[2] S. B. Akers, "Binary Decision Diagram," in *IEEE Transactions on Computers*, Vol. C-27, No. 6, pp. 509–516, June 1978.

[3] C. Alpert and A. Devgan, "Wire Segmenting for Improved Buffer Insertion," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*, pp. 588-593, 1997.

[4] F. Berglez and H. Fujiwara, "Accelerated ATPG and Fault Grading via Testability Analysis," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 695–698, 1985. (Available at http://www.cbl.ncstate.edu/CBL_Docs/iscas85.html).

[5] M. R. C. M. Berkelear and J. A. G. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming," in *Proceedings of the IEEE European Design Automation Conference*, pp. 217–221, 1990.

[6] M. R. C. M. Berkelear, P. H. W. Buurman and J. A. G. Jess, "Computing the Entire Active Area/Power Consumption Versus Delay Trade-off Curve for Gate Sizing with a Piecewise Linear Simulator," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 474–480, 1994.

[7] C. L. Berman, J. L. Carter and K. L. Day, "The Fanout Problem: From Theory to Practice," in *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pp. 69–99, 1989.

[8] K. P. Boese, A. B. Kahng, B. A. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 81–84, 1993.

[9] M. Borah, R. M. Owens and M. J. Irwin, "Transistor Sizing for Low Power CMOS circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 15, No. 6, pp. 665–671, June 1996.

[10] K. S. Brace, R. L. Rudell and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Proceedings of the ACM/IEEE 27th Design Automation Conference*, pp. 40–45, 1990.

[11] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677–691, August 1986.

[12] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung and A. Sangiovanni-Vincentelli, "Multiple-level Logic Optimization System," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 356–359, 1986.

[13] P. Buch, A. Narayan, A. R. Newton and A. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 663–670, 1997.

[14] P. Buch, A. Narayan, A. R. Newton and A. Sangiovanni-Vincentelli, "On Synthesizing Pass Transistor Networks," in *Proceedings of the International Workshop on Logic Synthesis*, pp. 1–8, 1997.

[15] T. M. Burks, K. A. Sakallah and T. N. Mudge, "Critical Paths in Circuits with Level-sensitive Latches," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 3, No. 2, pp. 273–291, June 1995.

[16] B. Carlson, S. Chen and C. Y. Roger, "Performance Enhancement of CMOS VLSI Circuits by Transistor Reordering," in *Proceedings of the ACM/IEEE 30th Design Automation Conference*, pp. 361–366, 1993.

[17] K. Chaudhary and M. Pedram, "A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints," in *Proceedings of the ACM/IEEE 29th Design Automation Conference*, pp. 492–498, 1992.

[18] H. Y. Chen and S. M. Kang, "iCOACH: A Circuit Optimization Aid for CMOS High-Performance Circuits," *Integration, the VLSI Journal*, Vol. 10, pp. 155-168, January, 1991.

[19] C. P. Chen, Y. W. Chang and D. F. Wong, "Optimal Wire-Sizing Formula under the Elmore Delay Model," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*, pp. 361–366, 1996.

[20] M. A. Cirit, "Transistor Sizing in CMOS Circuits," in *Proceedings of the ACM/IEEE 24th Design Automation Conference*, pp. 121–124, 1987.

[21] J. Cong and L. He, "Efficient Approach to Simultaneousl Transistor and Interconnect Sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 181–186, 1996.

[22] J. Cong and L. He, "Optimal Wiresizing for Interconnects with Multiple Sources," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 1, No. 4, pp. 478–511, April 1996.

[23] J. Cong, L. He, C. K. Koh and P. Madden, "Performance Optimization of VLSI Interconnect Layout" *Intergration, the VLSI Journal*, Vol. 21, pp. 1–94, 1996.

[24] J. Cong and Y. Y. Hwang, "Structural Gate Decomposition for Depth-Optimal Technology Mapping in LUT-Based FPGA Design," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*, pp. 726–729, 1996.

[25] J. Cong and K. S. Leung, "Optimal Wiresizing Under Elmore Delay Model," *IEEE Transactions on Computer Aided Design*, Vol. 14, No. 3, pp. 321–336, 1995.

[26] J. Cong and C. K. Koh. "Simultaneous Drive and Wire Sizing for Performance and Power Optimization." *IEEE Transactions on Very Large Scale Integration Systems*. Vol. 2. No. 4. pp. 408–425. December 1994.

[27] T. H. Cormen and C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill Book Company. New York. New York. 1990.

[28] Z-J. Dai and K. Asada. "MOSIZ: A Two-Step Transistor Sizing Algorithm Based on Optimal Timing Assignment Method for Multi-stage Complex Gates." in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 17.3.1–17.3.4. 1989.

[29] F. Dartu. N. Menezes. J. Qian and L. T. Pillage. "A Gate-Delay Model for High-Speed CMOS Circuits." in *Proceedings of the ACM/IEEE 31st Design Automation Conference*. pp. 576–580. 1994.

[30] F. Dartu and L. T. Pileggi. "Calculating Worst-Case Gate Delays Due to Dominant Capacitance Coupling," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*. pp. 46-51. 1997.

[31] E. Detjens. G. Gannot. R. Rudell. A. Sangiovanni-Vincentelli and A. Wang. "Technology Mapping in MIS." in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. pp. 116–119. 1987.

[32] S. Dey and F. Berglez and G. Kedem. "Corolla Based Circuit Partitioning and Resynthesis." in *Proceedings of the ACM/IEEE 27th Design Automation Conference*. pp. 607–612. 1990.

[33] S. Dhar and M. A. Franklin. "Optimum Buffer Circuits for Driving Long Uniform Lines," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 1. pp. 32–40, January 1991.

[34] S. Dutta, S. Nag and K. Rog, "ASAP: A Transistor Sizing Tool for Speed. Area and Power Optimization of Static CMOS Circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 61–64, 1994.

[35] J. Ecker. "Geometric Programming: Methods, Computations and Applications." *SIAM Review*, Vol. 22, pp. 338–362. July 1980.

[36] W. C. Elmore. "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers." *Journal of Applied Physics*, Vol. 19, No. 1, pp. 55–63. January 1948.

[37] J. Fishburn and A. Dunlop. "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[38] S. Gaiotti, M. R. Dagenais and N. C. Rumin. "Worst-case Delay Estimation of Transistor Groups," in *Proceedings of the ACM/IEEE 26th Design Automation Conference*, pp. 491–496, 1989.

[39] S. Gavrilov, A. Glebov, S. Pullela, S. C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan and D. T. Blaauw, "Library-Less Synthesis for Static CMOS Combinational Logic Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 658–662, 1997.

[40] L. P. P. P. V. Ginneken, "Buffer Placement in Distributed RC-tree Neteorks for Minimal Elmore Delay," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 865–868, 1990.

[41] L. A. Glasser and L. J. Hoyte. "Delay and Power Optimization in VLSI Circuits," in *Proceedings of the ACM/IEEE 21st Design Automation Conference*, pp. 529–535, 1984.

[42] A. L. Glebov, D. Blaauw and L. G. Jones, "Transistor Reordering for Low Power CMOS Gates Using an SP-BDD Representation," in *Proceedings of the International Symposium on Low Power Design*, pp. 161–166, 1995.

[43] D. Gragory, K. Bartlett, A. de Geus and G. Hachtel, "SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic," in *Proceedings of the ACM/IEEE 23rd Design Automation Conference*, pp. 79–85, 1986.

[44] R. Gupta, B. Krauter, B. Tutuianu, J. Willis and L. T. Pileggi, "Elmore Delay as a Bound for RC Trees with Generized Input Signals," in *Proceedings of the ACM/IEEE 33rd Design Automation Conference*, pp. 364–369, 1995.

[45] N. Hedenstierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization," *IEEE Transactions on Computer Aided Design*, Vol. 6, No. 2, pp. 270–287, March 1987.

[46] K. S. Hedlund, "AESOP: A Tool for Automated Transistor Sizing," in *Proceedings of the ACM/IEEE 24th Design Automation Conference*, pp. 114–120, 1987.

[47] K. S. Hedlund, "Models and Algorithms for Transistor Sizing in MOS Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 12–14, 1984.

[48] L. S. Heulser and W. Fichtner, "Transistor Sizing for Large Combinational Digital CMOS Circuits," *Integration, the VLSI Journal*, Vol. 10, pp. 270–281, January, 1991.

[49] K. O. Jeppson, "Modeling the Influence of the Transistor Gain Ratio and the Input-to-output Coupling Capacitance on the CMOS Inverter Delay," *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 6, pp. 646–654, June 1994.

[50] Y. Jiang, S. S. Sapatnekar, C. Bamji and J. Kim, "Interleaving Buffer Insertion and Transistor Sizing into a Single Optimization," to appear *IEEE Transactions on Very Large Scale Integration Systems*, 1998.

[51] Y. Jiang, S. S. Sapatnekar and C. Bamji, "A Fast Global Gate Collapsing Technique for High Performance Designs using Static CMOS and Pass Transistor Logic," to appear in *Proceedings of the IEEE International Conference on Computer Design*, 1998.

[52] Y. Jiang, S. S. Sapatnekar, C. Bamji and J. Kim, "Combined Transistor Sizing with Buffer Insertion for Timing Optimization," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 605–608, 1998.

[53] W. H. Kao, "ARIES. A Workstation Based. Schematic Driven System for Circuit Design." in *Proceedings of the ACM/IEEE 21st Design Automation Conference*. pp. 301-307. 1984.

[54] W. H. Kao. N. Fathi and C. M. Lee, "Algorithms for Automatic Transistor Sizing CMOS Circuits." in *Proceedings of the ACM/IEEE 22nd Design Automation Conference*. pp. 781-784. 1985.

[55] K. Keutzer. "DAGON: Technology Binding and Local Optimization by DAG Matching." in *Proceedings of the ACM/IEEE 24th Design Automation Conference*. pp. 341-347. 1987.

[56] J. Kim. C. Bamji. Y. Jiang and S. S. Sapatnekar. "Concurrent Transistor Sizing and Buffer Insertion by Considering Cost_Delay Tradeoffs." in *Proceedings the IEEE of International Symposium on Physical Design*. pp. 130–135. 1997.

[57] T. Kirkpatrick and N. Clark, "PERT as an Aid to Logic Design." *IBM Journal of Research and Development*. Vol. 10, pp. 135–141, March 1966.

[58] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich. "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization" *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 3, March 1991.

[59] K. Kodandapani, J. Grodstein, A. Domic and H. Touati. "A Simple Algorithm for Fanout Optimization using High-performance Buffer Libraries," in *Proceedings the IEEE/ACM of International Conference on Computer-Aided Design*, pp. 466–471, 1993.

[60] J-K. Kong and D. Overhauser, "Combining RC-interconnect Effects with Nonlinear MOS Macromodels," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 570–573, 1995.

[61] J-K. Kong, S. Z. Hussain and D. Overhauser, "Improving Digital MOS Macromodel Accuracy," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 578–581, 1995.

[62] T. Kuroda and T. Sakurai. "Overview of Low-power ULSI Circuit Techniques." *IEICE Transactions on Electronics*, Vol. E78-C, No. 4, pp. 334–344, April 1995.

[63] M. Lega, "Mapping Properties of Multi-level Logic Synthesis Operations," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 257–260, 1988.

[64] J. Lillis, C-K Cheng and T-T. Y. Lin. "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 138–143, 1995.

[65] S. Lin, M. M. Sadowska and E. S. Kuh, "Delay and Area Optimization in Standard-cell Design," in *Proceedings of the ACM/IEEE 27th Design Automation Conference*, pp. 349–352, 1990.

[66] F. Mailhot and G. DeMicheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," in *Proceddings of the IEEE European Design Automation Conference*, pp. 212–216, 1990.

[67] D. P. Marple and A. E. Gamal, "Optimal Selection of Transistor Sizes in Digital VLSI Circuits," in *Proceedings of the 1989 Conference Advanced Research in VLSI*, pp. 43–48, 1989.

[68] D. P. Marple, "Transistor Size Optimization in the Tailor Layout System," in *Proceedings of the ACM/IEEE 26th Design Automation Conference*, pp. 43–48, 1989.

[69] D. P. Marple, "Transistor Size Optimization of Digital VLSI Circuits," *Technical Report CSL-TR-86-308*, Stanford University, October, 1986.

[70] M. Matson and L. Glasser, "Macromodeling and Optimization of Digital MOS VLSI Circuits," *IEEE Transactions on Computer Aided Design*, Vol. 6, pp. 659–678, 1986.

[71] N. Menezes, R. Baldick and L. T. Pileggi, "Sequential Quadratic Programming Approach to Concurrent Gate and Wire Sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 144–151, 1995.

[72] F. W. Obermerier and R. H. Katz, "An Electrical Optimizer that Considers Physical Layout," in *Proceedings of the ACM/IEEE 25th Design Automation Conference*, pp. 453–459, 1988.

[73] N. Ohkubo, "A 4.4ns CMOS 54x54-b Multiplier Using Pass-transistor Multiplexer," *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 3, pp. 251–257, March 1995.

[74] M. Pedram and N. Bhat, "Layout Driven Technology Mapping," in *Proceedings of the ACM/IEEE 28th Design Automation Conference*, pp. 99–105, 1991.

[75] L. T. Pillage and R. A. Rohrer, "Asymptotic Waveform Evaluation for Timiing Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 4, pp. 352–366, April, 1990.

[76] J. Pincus, "Transistor Sizing," *Technical report UCB/CSD 86/285*, University of California, Berkeley, 1986.

[77] S. C. Prasad and K. Roy, "Circuit Optimization for Minimization of Power Consumtion Under Delay Constraints," in *Proceedings of the IEEE International Conference on VLSI Design*, pp. 305–309, 1995.

[78] W. H. Press, S. A. Teukolsky, W. T. Vettering and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.

[79] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, New York, NY: McGraw-Hill, 1978.

[80] J. Rubinstein, P. Penfield and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-2, No. 3, pp. 202–211, July 1983.

[81] T. Sakurai and A. R. Newton, "Delay Analysis of Series-connected MOSFET Circuits," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 2, pp. 122–131, February 1991.

[82] A. H. Salek, J. Lou and M. Pedram, "A DSM Design Flow: Putting Floorplanning, Technology-Mapping, and Gate-Placement Together," in *Proceedings of the ACM/IEEE 35th Design Automation Conference*, 1998.

[83] S. S. Sapatnekar, P. M. Vaidya and S-M. Kang, "Convexity-based Algorithms for Design Centering," *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 12, pp. 1536–1549, December 1994.

[84] S. S. Sapatnekar, V. B. Rao and P. M. Vaidya, "A Convex Optimization Approach to Transistor Sizing for CMOS circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 482–485, 1991.

[85] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya and S. M. Kang, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization," *IEEE Transactions on Computer-Aided Design*, Vol. 12, pp. 1621–1634, November 1993.

[86] Y. Sasaki, "Multi-level Pass-transistor Logic for Low-power ULSIs," in *Proceedings of the IEEE Symposium on Low Power Electronics*, pp. 14–15, 1995.

[87] K. Sato, M. Kawarabayashi, H. Emura and N. Maeda, "Post-layout Optimization for Deep Submicron Design," in *Proceedings of the ACM/IEEE 33rd Design Automation Conference*, pp. 740–745, 1996.

[88] Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors*, SEMATECH Inc., 1997.

[89] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K.Brayton and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," *Technical Report UCB/ERL M92/41*, Electronics Research Laboratory, University of California at Berkeley, Berkeley, May 1992.

[90] J. Shyu, A. S. Vincentelli, J. Fishburn and A. Dunlop, "Optimization-Based Transistor Sizing," *IEEE Journal of Solid-State Circuits*, SC-23, No. 2, pp. 400–409, April 1988.

[91] P. Siegel. "Automatic Technology Mapping for Asynchronous Designs." *Technical Report CSL-TR-95-663*, Stanford University. March 1995.

[92] K. J. Singh and A. Sangiovanni-Vincentelli. "A Heuristic Algorithm for the Fanout Problem." in *Proceedings of the ACM/IEEE 25th Design Automation Conference*. pp. 357–360. 1988.

[93] K. J. Singh. A. R. Wang. R. K. Brayton and A. S. Vincentelli. "Timing Optimization of Combinational Logic," in *Proceeding of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 282–285. 1988.

[94] D. Somasekhar and K. Roy, "Differential Current Switch Logic: A Low Power DCVS Logic Family," IEEE Journal of Solid-State Circuits, Vol. 31. No. 7. pp. 981–991. July 1996.

[95] M. Tachibana. "Synthesize Pass Transistor Logic Gate by Using Free Binary Decision Diagram," in *Proceeedings of the International ASIC Conference*. September 1997.

[96] H. J. Touati. H. Savoj and R. K. Brayton, "Delay Optimization of Combinational Logic Circuits by Clustering and Partial Collapsing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 188–191. 1991.

[97] P. M. Vaidya. "A New Algorithm for Minimizing Convex Functions over Convex Sets," in *Proceedings of IEEE Foundations of Computer Science*, pp. 338-343, October 1989.

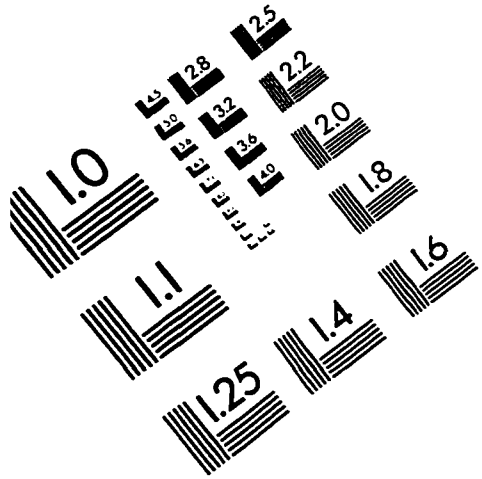
[98] D. F. Wong, "A Fast and Accurate Technique to Optimize Characterization Tables for Logic Synthesis," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*. pp. 337–340. 1997.

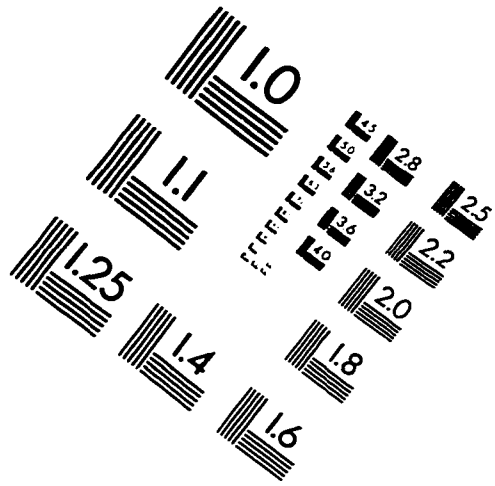
[99] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," *Technical Report*. Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991. (Available at http://www.cbl.ncsu.edu/CBL_Docs/lgs91.html).

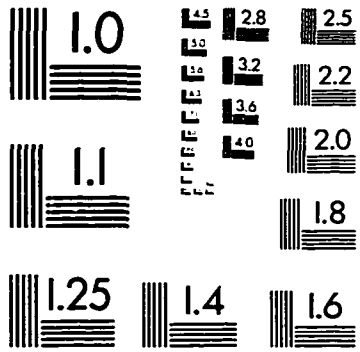
[100] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi and A. Shimizu, "A 3.8ns CMOS 16x16-b Multipliter Using Complementary Pass-transistor logic," *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 2, pp. 388–395, April 1990.

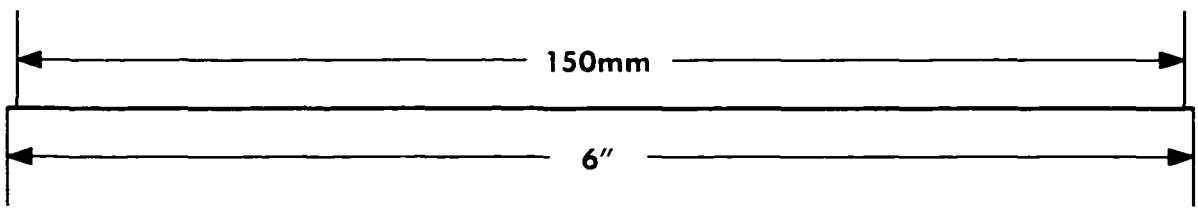
[101] H. Zhou and D. F. Wong, "Exact Gate Decomposition Algorithm for Low-Power Technology Mapping," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 575–580, 1997.

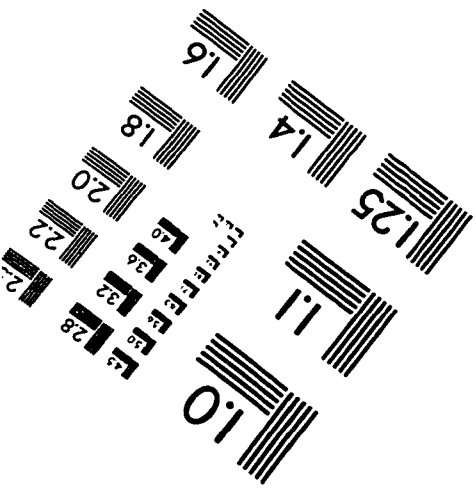
IMAGE EVALUATION
TEST TARGET (QA-3)

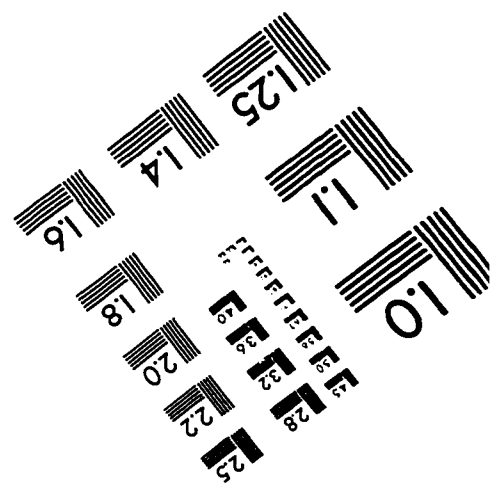












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989