DELAY MODELING AND OPTIMIZATION IN VLSI CIRCUIT SYNTHESIS

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

MAHESH KETKAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Sachin S. Sapatnekar, Advisor

JULY 2002

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral thesis by

MAHESH KETKAR

and have found that it is complete and satisfactory in all aspects,
and that any and all revisions required by the final
examining committee have been made.

Professor Sachin S. Sapatnekar

Name of Faculty Advisor

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Abstract

The evolution of deep submicron technologies has placed a high importance on the fidelity of simulation and modeling techniques as compared to the final chip realization. In addition, ever tightening delay, area, and power constraints together with increasing design complexities demand that the optimization techniques used in design process be fast and accurate. The purpose of this thesis is to develop accurate delay models that are amenable to optimization and to develop efficient yet accurate synthesis and optimization strategies at logic, circuit, and physical levels of design process. The thesis consists of three parts that are described below.

A large number of the optimizations that arise in electronic design automation can be modeled as nonlinear optimization problems, which can be solved more easily if the objective and constraint functions are convex. Particularly for delay optimization, this is becoming increasingly problematic since the existing models that possess such properties, such as the Elmore delay model, have become highly inaccurate in deep submicron technologies. Therefore, the first part of this thesis involves development of accurate and convex models for gate delay. The efficacy of the model is shown by its incorporation into various optimizers and performing area-delay and dynamic power-delay trade-off optimizations.

Another important issue that has arisen in the recent technology generations is that, gate leakage has become a significant contributor to the total power. The reason for this is that the subthreshold leakage current of a transistor has an exponential dependence on its threshold voltage (V_t), which no longer scales in proportion to the supply voltage. Hence, the second part of this thesis targets leakage-delay optimization in contrast to the dynamic power-delay optimization formulation used in the first problem. A novel enumerative approach, along with efficient pruning techniques, is proposed for V_t assignment to minimize power under delay constraints. This method is then used in a unified approach employing simultaneous transistor sizing and threshold voltage assignment to achieve a low leakage circuit configuration under given area and delay constraints.

The third part of this thesis develops a methodology for efficient timing closure for dat-

apath dominated circuits, and involves logic synthesis guided by structural properties of the circuit. Datapaths are typically characterized by a high degree of regularity, and it is beneficial to preserve this regularity during synthesis so that layout synthesis tools can generate regular, and hence predictable layouts. However, the rigid preservation of regularity is likely to result in circuit configurations with high delay values. The approach proposed involves controlled destruction of regularity during the synthesis process, thus leading to delay-optimized, yet highly regular, circuits.

Acknowledgement

First, I would like to thank my advisor, Prof. Sachin Sapatnekar, for his guidance and support throughout my doctoral studies. He provided very valuable help whenever I needed assistance of any kind. His emphasis on reasoning out everything, on clarity in presentation of ideas, and on looking at the holistic picture of a problem always guided me in the right direction, and will continue to guide me throughout my career.

I would like to thank Prof. Bazargan, Prof. Sobelman, and Prof. Du, for reviewing this thesis and for providing valuable suggestions. I would like to thank Dr. Priyadarsan Patra from Intel Corporation, for his valuable guidance throughout my doctoral work.

I would also like to thank members of the VEDA Lab - Brent, Cheng, Haihua, Haitian, Rupesh, Suresh, Tianpei, and Venkat - with whom I have had several meaningful discussions. I would like to thank Dr. Jiang Hu and Dr. Min Zhao for their help during early period of my doctoral studies, and to Arvind Karandikar for his help in various aspects of software development. I would like to thank Kishore Kasamsetty for his contributions in the project on gate delay modeling.

In four years I met a lot of interesting people who made my stay in Minneapolis a very enjoyable experience, and to that effect, I would like to thank Girish, Parag, Ameeta, Sirisha, Rupa, Kedar, and Cristi. Special thanks are due to Shantanu Rane, a great roommate and a close friend, for his support, and for various stimulating conversations.

Finally, I would like to thank my parents for their continuous support and encouragement during this work, and for their teachings throughout my life. This thesis would not have taken place without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

For the past four decades, the semiconductor industry has distinguished itself by the rapid pace of improvements in its products. The propelling force behind this magnificent journey has been an ambition to adhere to Moore's Law [3], which originally stated that the number of devices on a chip would double every year (the time frame was later modified to every two years, and subsequently averaged to every eighteen months).

Thus each new generation of chips has more and more devices and hence interconnects as compared to previous generations. At the same time, market competition and widespread use of mobile devices, has resulted in ever tightening delay, area, and power constraints. These together have led to a drastic increase in design complexity, which means that the automatic synthesis tools must do a better job of optimization; specifically, the tools should realize good circuit configurations in as little time as possible. This requirement translates down to various issues in development of CAD tools, and some of these are mentioned below.

- It dictates that the CAD tools should be scalable to handle larger and more complex designs as compared to previous generations.
- It warrants better fidelity of area, delay, and power models to actual silicon realiza-

tions, and that the models should be amenable to optimization so that accurate global solutions can be found.

- The advances in process technology typically come with a host of new design variables and hence a new set of challenges for the designers. One example of this is the increased relative importance of newer design considerations such as leakage power as compared to previous generations. CAD tools are required to handle such new design parameters.

- It prompts the exploration of radical shifts in design methodologies to minimize synthesis times and to obtain good solutions.

The research in this thesis tries to address timing optimization of VLSI circuits in light of some of the issues mentioned above. The three problems solved in this thesis are listed below, and their relation to the issues mentioned above is examined following the enumeration.

- Development of convex and accurate models for the gate delay. The models are incorporated into optimization engines to solve area-delay and power-delay tradeoffs.

- Optimization of standby leakage power under area and delay constraints utilizing transistor sizing and dual threshold voltage assignment.

- Regularity-driven synthesis of datapath dominated circuits.

The underlying theme in the choice of the problems solved is delay optimization. The first problem is targeted towards development of delay models that will lead to fast and accurate solution of a transistor sizing problem. The problem tries to address the issue of modeling mentioned above, and at the same time has repercussions on scalability, because the existence of optimization-amenable models can enable fast and accurate optimizations with proper choice of optimization engine. The second problem provides an algorithm for the optimization of leakage-delay trade-offs and thus addresses an important emerging issue. The third part presents a methodology that improves upon the traditional synthesis flow in that it uses physical information to drive the logic synthesis process.

The material presented below provides an introductory treatment of these problems and gives an insight into the choice of these particular problems in the arena of timing optimization. It also provides numerical figures to highlight the contributions of this thesis.

Before embarking on the introduction of individual problems it is in order to see where these problems fit in a typical design flow, which consists of various stages [4]. Such a flow ranges from behavioral and architectural synthesis at the topmost level, which determines an assignment of circuit functions to various operators, their interconnection, and execution timing, to layout synthesis at the bottommost level, which converts a netlist of transistors into a geometric representation to be realized in silicon. This thesis concentrates on circuit synthesis problems arising at lower levels of abstraction, namely, the logic, circuit, and physical levels.

Let us consider the motivation for the first problem. A large number of the optimizations that arise at the circuit and physical levels can be modeled as nonlinear optimization problems, which can be solved more easily if the objective and constraint functions have the property of convexity, which is defined precisely in Section 2.6.1, and numerous optimizations have been facilitated by the fact that the Elmore delay model [5] can be proved to be a convex function in some design variables. Unfortunately, the value of this property has been diminished as the Elmore has become highly inaccurate in deep submicron technologies. Therefore, the first issue addressed in this thesis involves the development of accurate and convex models for gate delay. A new class of convex functions called *generalized posynomials* is proposed and subsequently used to accurately model the gate delay. Experimental results showed that the delay values obtained using the proposed models were within 5% of SPICE [6] on average, and within 2% of SPICE for most of the gates. The efficacy of the model is shown by incorporating it into various optimizers and performing area-delay and dynamic power-delay trade-off optimizations.

It is well known that in the past, trends in performance, density and power have followed the scaling theory. For these trends to continue, two important issues that need special attention are power delivery and power dissipation [7]. Typically, each generation of chips

has shown about 30% reduction in gate delay and about the same or greater reduction in energy dissipation per transition. The total power dissipation can be scaled down by reducing either the supply voltage, frequency or die size, each of which results in reduced performance. Therefore, the problem of optimization for low power to meet the stringent performance constraints continues to be a very important problem.

Recognizing this, power-delay trade-off optimizations are addressed in two of the problems solved in this thesis. The power dissipation of a gate results from three sources, namely, dynamic power, leakage power, and short circuit power, described in detail in Section 2.1.3. Power optimization is addressed in part in the first problem, by incorporating the new delay model into an optimization engine to solve the problem of trading-off dynamic power with delay.

In recent generations, however, the gate leakage has become a significant contributor to the total power. The reason for this is that the subthreshold leakage current of a transistor has an exponential dependence on its threshold voltage (V_t), which no longer scales in proportion to the supply voltage from one generation to the next. Hence, the second problem targets the trade-off optimization between leakage power and delay, in contrast to the dynamic power-delay optimization formulation used in the first problem. A novel enumerative approach, along with efficient pruning techniques, is proposed for V_t assignment to minimize power under delay constraints. This method is then incorporated into an unified approach employing simultaneous transistor sizing and threshold voltage assignment to achieve a low leakage circuit configuration under given area and delay constraints. Results show that the V_t assignment algorithm on average results in 50% reduction in leakage power at fixed transistor sizes as compared to an existing approach [1]. The unified approach resulted in 9% lower leakage on average, and barring two small ISCAS 85 benchmark circuits, performed about 1.4x faster as compared to an existing unified optimization approach [2].

The third part of this thesis is related to the interaction between logic synthesis and layout. It develops a methodology for efficient timing closure of datapath-dominated circuits, and involves logic synthesis guided by structural properties of the circuit. Datapaths

are characterized by *regularity* (explained in Section 5.2). It is beneficial to preserve this regularity during synthesis so that layout synthesis tools can generate regular and hence predictable layouts. However, the rigid preservation of regularity typically results in circuit configurations with high delay values. The approach proposed involves controlled destruction of regularity during the synthesis process, thus leading to delay-optimized, yet highly regular circuits. The experimental results show that the proposed methodology realizes circuits with about the same delays and much higher regularity, as compared to the circuits obtained by traditional synthesis.

1.2 Thesis organization

The organization of this thesis is as follows. Chapter 2 presents background material relevant to this thesis, including a brief treatment of mathematical optimization. Chapter 3 describes a solution to the first problem, introducing a new class of convex functions and demonstrating its use in gate delay modeling. The delay model is incorporated into various optimization engines and the results of area-delay and dynamic power-delay tradeoff optimizations are provided. Next, an algorithm for leakage power optimization via transistor sizing and V_t assignment is presented in Chapter 4, corresponding to a solution to the second problem in the thesis. The third problem is tackled in Chapter 5 which develops a method for regularity-driven synthesis of datapath-dominated circuits. Finally, Chapter 6 provides concluding remarks and future research directions.

Chapter 2

Preliminaries

Two of the problems tackled in this thesis involve optimization of design variables, and hence it is in order to begin by presenting some introductory material on mathematical optimization. Most of the circuit optimization problems pertain to one specific category of optimization problems: *nonlinear optimization*. Transistor sizing, a problem from circuit synthesis, is used repeatedly in this thesis, and is also used as a running example throughout explanation of optimization categories. To give an idea of the transistor sizing problem, various circuit parameters are first introduced in Section 2.1. The explanation also serves the purpose of introducing the models used in this thesis. In particular, the Elmore model is explained in detail in Section 2.2, and following that the transistor sizing problem is introduced in Section 2.3. Nonlinear optimization is then introduced in Section 2.4, while three special types of nonlinear optimization problems, namely, geometric programs, convex programs, and mixed integer nonlinear programs, are introduced in Sections 2.5, 2.6, and 2.7, respectively.

2.1 Circuit parameters

An integrated circuit can be represented at various levels of abstractions such as the circuit level, the logic level, or the architectural level. At each level of abstraction, circuit parameters are estimated using different models, and the accuracy of these models typically

decreases as one moves to a higher level of abstraction. The first two problems tackled in this thesis involve circuit-level optimizations, and this section provides the models used at this level.

2.1.1 Area

The area of a transistor netlist is typically measured as summation of transistor widths:

$$Area = \sum_{i=1}^{n} x_i \tag{2.1}$$

where n is the number of transistors in the circuit, and x_i represents the size of the i^{th} transistor in the circuit. This area model does not consider the routing area, but this is acceptable in case of transistor sizing, which makes but little perturbations to circuit placement, and in case of threshold voltage (V_t) assignment, which does not necessitate any alterations in placement.

2.1.2 Delay

The delay characteristics of the output waveform at a gate may be represented by two numbers:

(1) the *delay*, i.e., the difference in the time when the output waveform crosses 50% of its final value, and the corresponding time for the input waveform.

(2) the *output transition time*, i.e., the time required for the waveform to go from 10% to 90% of its final value.

The Elmore delay model, described in the next section, models the gate delay.

2.1.3 Power

The power dissipated in a gate consists of three components.

(1) Dynamic power: This refers to the power dissipated in a gate when it switches from one logic state to another. This component of power dissipation is due to the charging and

discharging of the output capacitance, and is given by

$$P_{i_{sw}} = \frac{1}{2} \cdot {V_{dd}}^2 \cdot C_i \cdot TD_i \tag{2.2}$$

where $P_{i_{sw}}$ is the average switching power dissipation, V_{dd} is power supply voltage, C_i is the output capacitance, and TD_i (explained later in Section 3.7.3) is the transition density, all corresponding to gate n_i.

(2) Leakage power: This component corresponds to the power dissipated by the current that flows through a transistor that is nominally off and is caused by two mechanisms. The first is the subthreshold leakage, which forms the dominant source of leakage power, and the second part is due to the tunneling current through the gate oxide. The BSIM [8] model for the subthreshold leakage current for a single transistor is listed below.

$$\begin{aligned} I_{leak} &= I_o e^{(V_{gs}-V_t)/nV_{therm}}(1 - e^{-V_{ds}/V_{therm}}), \\ \text{where} \quad I_o &= \mu_0 C_{ox}(W/L){V_{therm}}^2 e^{1.8} \end{aligned} \tag{2.3}$$

Here μ_0 is the zero bias electron mobility, n is the subthreshold slope coefficient, V_{gs} and V_{ds} are the gate-to-source voltage and the drain-to-source voltage, respectively, V_{therm} is the thermal voltage, and W and L are the transistor width and length, respectively.

(3) Short circuit power: When the output of a gate switches, a low resistance path exists between V_{dd} and ground during the period of the input transition, and this leads to short circuit power dissipation. Short circuit power forms a very small component of the total power provided the transition times are all controlled, and hence is not considered as an optimization parameter in this thesis. Interested reader can find further treatment of short circuit power in [9].

2.2 Elmore delay model

An RC network is a collection of resistances and capacitances. Elmore [5] proposed that the delay of an RC network under a step input can be reasonably estimated as the time

coordinate of the center-of-area of the impulse response curve. To use this model to estimate the delay of a gate, the gate is converted into a network of resistances and capacitances, as shown in Figure 2.1. Figure 2.1(a) shows a switching gate with transistors of sizes x_1, x_2 and x_3, and its equivalent Elmore model is shown in Figure 2.1(b).

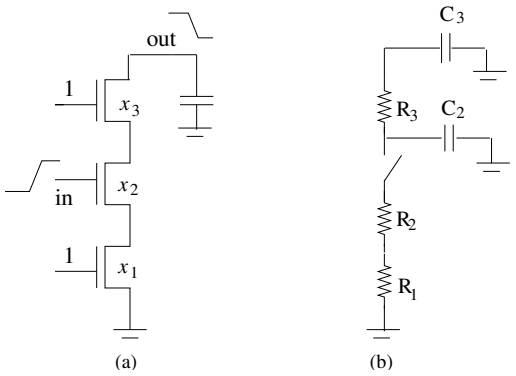

Figure 2.1: Elmore delay model

The delay of the gate is then evaluated as the Elmore time constant of this RC network, given as

$$(R_1 + R_2)C_2 + (R_1 + R_2 + R_3)C_3 \tag{2.4}$$

Each R_i is inversely proportional to the transistor width x_i, and each C_j is directly proportional to transistor width x_j. Hence Equation (2.4) can be rewritten as

$$(A/x_1 + A/x_2)(Bx_2 + Cx_3 + D) + (A/x_1 + A/x_2 + A/x_3)(Bx_3 + E) \tag{2.5}$$

where A, B, and C are constant coefficients for the resistance, drain capacitance, and source capacitance, respectively, of a unit size transistor, and D and E are wire capacitances.

Thus the general form of the Elmore delay can be written as

$$D(\mathbf{x}) = \sum_{i,j=1}^{n} a_{ij}\frac{x_i}{x_j} + \sum_{i=1}^{n}\frac{b_i}{x_i} + K \tag{2.6}$$

where $a_{ij}, b_i, K \in \mathbf{R}^+$ are constants and, $\mathbf{x} = [x_1, \cdots, x_n]$ is the vector of transistor sizes.

This form is used in various published works on transistor sizing, for example, in TILOS [10] and iCONTRAST [11]. A detailed treatment of timing analysis using the Elmore model can be found in [12].

2.3 Transistor sizing problem

Increasing the width of a transistor results in an increase in the circuit area and power dissipation. Moreover, if the updated transistor is an nmos transistor and lies on the resistive path between input a_i and the output of the gate, then it has two effects on the gate delay:
(1) it results in reduced delay from a_i to output for the falling output transition, and
(2) it leads to increased delay from a_i to output for the rising output transition.
Hence determining the widths of transistors in a circuit involves carefully solving this trade-off. Transistor sizing has been traditionally been formally defined as [10]:

$$\begin{aligned} \text{minimize} \quad & Area \; or Power \\ \text{subject to} \quad & Delay \; \leq T_{spec}. \end{aligned} \tag{2.7}$$

Previous works on the use of transistor sizing for area-delay trade-off optimization and power-delay trade-off optimization are outlined in Sections 3.7.2 and 3.7.3, respectively.

2.4 Nonlinear optimization

A Nonlinear Program (NLP) is a problem that can be formulated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i \; = \; 0, \quad 1 \leq i \leq k \\ & h_j \; \geq \; 0, \quad k+1 \leq j \leq m \end{aligned} \tag{2.8}$$

where $f(x)$, g_i and h_j are all nonlinear functions, and k is the number of equality constraints and m is the number of total constraints. Since, the functions are nonlinear, finding a global minimum of the optimization problem is a very difficult task, and there is always the danger

that a solution technique may be trapped in a local minimum.

However, not all nonlinear programs are difficult to solve. The objective functions and constraints may possess certain properties that can be exploited to efficiently obtain a global solution. Two of such problems are the *geometric programming* and the *convex programming* problems, which are explained in the next two sections.

2.5 Geometric optimization

2.5.1 Posynomial functions

A posynomial is a function p of a positive variable $\mathbf{x} \in R^n$ that has the form

$$p(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} \tag{2.9}$$

where the exponents $\alpha_{ij} \in \mathbf{R}$ and the coefficients $\gamma_j \in \mathbf{R}^+$. In the positive orthant in the $\mathbf{x}$ space, posynomial functions have the useful property that they can be mapped onto a convex function (explained in next section) through an elementary variable transformation, $(x_i) = (e^{z_i})$. Notice that the Elmore delay expressions provided in Equations (2.5) and 2.6 are a subset of the set of posynomials; specifically they are posynomials whose exponents belong to the set {-1,0,1}.

2.5.2 Geometric programming

An optimization problem of the type,

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{such that} \quad & g_i(\mathbf{x}) \leq C, 1 \leq i \leq m \\ & \mathbf{x} \in \mathbf{R}^+ \end{aligned} \tag{2.10}$$

is a geometric programming problem if $f(\mathbf{x})$ and $g_i(\mathbf{x})$ are posynomials.

A geometric optimizer converts the above minimization problem, called the primal prob-

lem into a maximization problem called the dual problem. It uses the arithmetic-geometric inequality, which states that the weighted arithmetic mean of a set of positive numbers is at least as great as their geometric mean, to prove that the maximum of the dual problem is indeed the minimum of the primal problem.

The degree of difficulty of a geometric program is given as

$$\text{degree} = \text{Number of terms} - \text{No. of variables} - 1 \tag{2.11}$$

When the degree of a geometric programming problem is zero, then the solution of the dual problem can be obtained by solving a system of linear equations. When the degree is greater than zero, the dual problem takes the form of a simple optimization problem, where all of the constraints are linear. Thus in either case, the dual problem is much simpler than the original or primal problem, and leads to fast solutions. A more in-depth treatment on geometric programming can be found in [13].

It can be noted that the area model mentioned in Section 2.1.1, is in the posynomial form. Moreover, if the Elmore model is used to evaluate the gate delay, then the path delay can be expressed as a summation of posynomials, and is hence a posynomial in itself. Thus the transistor sizing problem under these models is one of minimizing a posynomial function under posynomial constraints of the form mentioned in Equation (2.11), and is therefore a geometric programming problem.

2.6 Convex optimization

2.6.1 Convex sets and convex functions

A set C in R^n is said to be *convex* if, for every $x_1,\ x_2 \in C$, and every real number α, $0 \leq \alpha \leq 1$, the point $\alpha\ x_1 + (1 - \alpha)\ x_2 \in C$. This definition can be interpreted geometrically as stating that a set is convex, if given any two points in the set, every point on the line segment joining these two points is also a member of the set.

A function f defined on a convex set ω is said to be *convex* if, for every $x_1,\ x_2 \in \omega$, and

every α, $0 \leq \alpha \leq 1$,

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha\, f(x_1) + (1-\alpha)\, f(x_2) \tag{2.12}$$

This can be geometrically interpreted as stating that a function is convex if the line joining any two points on it, lies above the function. A few results are listed below.

(1) If $f(x)$ is a convex function, the $f(x) \leq c$ corresponds to a convex set.

(2) The intersection of two convex sets is a convex set. However, the union of convex sets, in general, is not convex.

For further details on convexity, the reader is referred to [14]. Figure 2.2 shows examples of a convex set and a convex function.

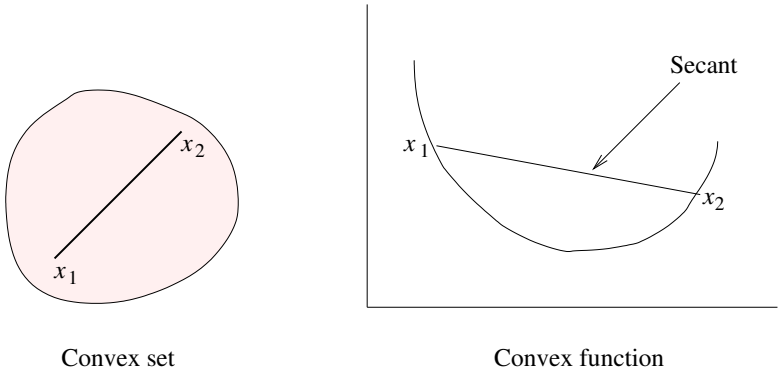

Figure 2.2: Convex set and convex function

2.6.2 Convex programming problem

A convex programming problem, also referred to as a convex optimization problem, involves the minimization of a convex function over a convex set. A problem of the type

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{such that} && g_i(\mathbf{x}) \leq 0, 1 \leq i \leq m \\ & && \mathbf{x} \in \mathbf{R}^n \end{aligned} \tag{2.13}$$

is a convex programming problem if $f(\mathbf{x})$ and $g_i(\mathbf{x}), 1 \leq i \leq m$, are convex functions.

The advantage of a convex programming formulation is that the problem is known to have the property that any local minimum is also a global minimum, and efficient optimization algorithms for solving such a problem are available. Therefore, it is desirable to attempt to express any optimization problem using a convex formulation, as far as possible, under the caveat that the accuracy of the modeling functions for the objective and the constraints must be preserved.

Section 2.5 showed that transistor sizing under the area and delay models from Sections 2.1 and 2.2 is a geometric programming problem. Under the variable transformation mentioned in Section 2.5.1 the transistor sizing problem takes the form of a problem of minimization of a convex function under convex constraints and hence is a convex programming problem. This indicates that under such models one can find a global optimum of the transistor sizing problem. Since the posynomial to convex function mapping mentioned in Section 2.5.1 is one-to-one, it can be used to claim that the minimum in original space corresponds to minimum in the transformed space.

In the next chapter a new delay gate model is proposed that retains the convexity property of the Elmore delay model.

2.7 Mixed integer nonlinear programs

A mixed integer nonlinear program (MINLP) is a nonlinear programming problem, where some of the variables x_i's are required to have integer values. This problem is much more computationally expensive than continuous optimization problems. The reason is that in case of continuous optimization, gradient-based methods can be used to arrive at a minimum, either local or global. But in the case of MINLP, due to the presence of integer variables gradients are not available, since the function exists only at discrete points.

Obtaining computationally efficient heuristics that can provide reasonably good solutions to this problem is an area of active research. One typical approach involves creation of a branch-and-bound tree. Each node in the tree corresponds to a nonlinear program, and the

tree is bounded according to the solution of these nonlinear programs. One particular type of MINLP is the 0-1 integer nonlinear program, where the integer variables are constrained to have one of the two values. The application of the branch-and-bound method for this case results in a binary tree, where the left child of a node fixes its corresponding integer variable to 0, and the right child sets its to 1. This approach can result in reasonable computation times for smaller programs, but for general purpose optimizations involving thousands of variables and constraints, the approach is computationally prohibitive and unrealistic.

The second part of this thesis solves the V_t assignment problem, where two fixed levels of V_t are used. It will be shown later in Section 4.3 that this problem is in the MINLP form. Hence, Chapter 4 develops an efficient heuristic to solve this problem.

Chapter 3

Convex Delay Modeling

3.1 Introduction

While building a timing optimization engine, it is essential for the underlying delay models to be accurate and efficient. Moreover, the task of obtaining globally optimal solutions is greatly eased if these models should possess convexity properties, as explained in Section 2.6. This chapter proposes an accurate circuit delay modeling procedure that results in provably convex delay functions.

The chapter is organized as follows. A brief summary of existing models is provided in Section 3.2. The idea of a generalized posynomial is presented in Section 3.3, and its application to gate delay modeling is discussed in Section 3.4. Section 3.5 presents the results demonstrating the accuracy of the new model, and a technique for converting generalized posynomial programs to posynomial programs is presented in Section 3.6. Section 3.7 presents a proof of convexity of path delays under this gate delay model and is followed by experimental results on circuit optimization in Section 3.7 that demonstrate the computational efficacy of the new model.

3.2 Existing delay modeling approaches

The Elmore delay model presented in Section 2.2 has been used traditionally to perform transistor-level delay analysis. This model, however, is known to be very inaccurate for

modern designs. This inaccuracy can be attributed to its failure to accurately consider important factors such as input transition times, the position of the switching transistor, the sizes of fighting complementary transistors, and the temporal relationship between inputs and transistor nonlinearities. As a result, exact optimization under this model may lead to an incorrect solution to the sizing problem since the timing model has a bad correlation with reality. More precisely, the solution may be either suboptimal in that it meets the timing specification without minimizing the cost function, or entirely inaccurate, in the sense that it may not meet the timing constraints at all. Therefore, it is immensely important to use more accurate timing models in sizing.

Several other approaches for accurate timing modeling have been proposed in the past. For example, one could model gate delays by developing closed form expressions [15]. Much work has been carried out in the development of closed form models for inverters and then mapping other gates to an equivalent inverter [16, 17]. An alternative approach uses a look-up table constructed using experimentally-derived delay data for various configurations, with intermediate data points being derived by interpolation methods, as in the delay model in [18]. However, this approach requires storage of large number of data points to guarantee accuracy and hence is very expensive in terms of memory requirements. Neither the closed-form modeling approach nor the table-look-up modeling method is particularly well suited for optimization since the modeling functions typically do not possess any convexity properties and cannot be used in the context of a formal optimization algorithm that is guaranteed to find the global minimum in a reasonable time. Moreover, it is not necessarily true that these models will have continuous derivatives, or, in the case of look-up tables, any derivative at all. Therefore, there is a need for new models that permit accurate delay computations, while maintaining convexity properties suited for optimization. This work derives a methodology for developing such models.

The theoretical underpinning of this approach is a result that defines a new class of functions that are shown to work well for modeling circuit delays. These functions are provably convex under a variable transformation that is explained in the next section. The

set of functions from which these functions are chosen includes the set of posynomials as a proper subset, and therefore, we refer to these functions as *generalized posynomials*. This work uses a curve-fitting approach to find a least-squares fit from the delay function, computed by SPICE over a grid, to a generalized posynomial in order to provide guarantees on accuracy of the delay model.

3.3 Generalized posynomials

3.3.1 Definitions

Posynomials and convex functions, described in Sections 2.5 and 2.6, are rich classes of functions and the basic motivation for this work is that better delay estimates can be obtained by fully exploiting this richness.

A generalized posynomial function $G_k(\mathbf{x}), \mathbf{x} \in \mathbf{R}^n$, where $k \geq 0$ is called the order of the function, is defined recursively as follows:

1. A generalized posynomial of order 0, G_0, is the posynomial defined earlier:

$$G_0(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}}, \tag{3.1}$$

   where the exponents $\alpha_{ij} \in \mathbf{R}$ and the coefficients $\gamma_j \in \mathbf{R}^+$.

2. A generalized posynomial of order $k \geq 1$ is defined as

$$G_k(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} \left[G_{k-1,i}(\mathbf{x})\right]^{\alpha_{ij}}, \tag{3.2}$$

   where the exponents $\alpha_{ij} \in \mathbf{R}^+$ and the coefficients $\gamma_j \in \mathbf{R}^+$, and $G_{k-1,i}(\mathbf{x})$ is a generalized posynomial of order $k-1$.

Specifically, the generalized posynomial of first order, is given by

$$f(\mathbf{x}) = \sum_i \gamma_i \prod_{j=1}^{m} \left(\sum_{l=1}^{p_i} \omega_{ijl} \prod_{s=1}^{n} x_s^{a_{ijls}} \right)^{\beta_{ij}} \tag{3.3}$$

where each $\beta_{ij} \in \mathbf{R}^+$, each $a_{ijls} \in \mathbf{R}$, each $\gamma_i \in \mathbf{R}^+$, and each $\omega_{ijl} \in \mathbf{R}^+$. Stripping Equation (3.3) of its complicated notation, one may observe that the term in the innermost bracket represents a posynomial function. Therefore, a generalized posynomial of first order is similar to a posynomial, except that the place of the x_i variables in Equation (2.9) is taken by a posynomial. Similarly, a generalized posynomial of order k uses a generalized posynomial of order $k-1$ in place of the x_i variables in Equation (2.9). Appendix A shows that if the range of interest of $\mathbf{x}$ is restricted to the positive orthant where each $x_i > 0$, then under the variable transformation from the space $\mathbf{x} \in \mathbf{R}^n$ to the space $\mathbf{z} \in \mathbf{R}^n$ given by $x_i = e^{z_i}$, the generalized posynomial function f of Equation (3.2) is mapped to a convex function in the $\mathbf{z}$ domain.

3.3.2 Proof of convexity

The following theorem parallels the relationship between posynomials and convex functions.

Theorem 1: If the range of interest of $\mathbf{x}$ is restricted to the positive orthant where each $x_i > 0$, then under the variable transformation from the space $\mathbf{x} \in \mathbf{R}^n$ to the space $\mathbf{z} \in \mathbf{R}^n$ given by $x_i = e^{z_i}$, the generalized posynomial function f of Equation (3.2) is mapped to a convex function in the $\mathbf{z}$ domain.

Proof: It is well known that a generalized posynomial of order 0, $G_0(\mathbf{x})$, is transformed to a convex function, $G_0(\mathbf{z})$ in the $\mathbf{z}$ domain [19]. Since the functional form of the functions $G_k(\mathbf{x}), k > 0$, is different from that of $G_0(\mathbf{x})$ due to the additional nonnegativity constraint on the α_{ij} variables, they are treated separately.

The proof of Theorem 1 proceeds by considering $G_k(\mathbf{z})$ for $k \geq 1$; to prove its convexity, it is enough to prove the convexity of

$$L = P \prod_{i=1}^{m} (G_{k-1,i})^{\beta_i}, \beta_i \geq 0, \tag{3.4}$$

since a sum of convex functions is convex. The gradient and Hessian of this function are,

respectively, given by

$$\begin{aligned}
\bigtriangledown L &= P\sum_{i=1}^{m}\left\{\left(\prod_{j=1,i\neq j}^{m}(G_{k-1,j})^{\beta_j}\right)\beta_i(G_{k-1,i})^{\beta_i-1}\bigtriangledown G_{k-1,i}\right\} \\
&= L\sum_{i=1}^{m}\frac{\beta_i\bigtriangledown G_{k-1,i}}{G_{k-1,i}} \qquad (3.5)\\
\bigtriangledown^2 L &= L\left\{\left(\sum_{i=1}^{m}\frac{\beta_i\bigtriangledown G_{k-1,i}}{G_{k-1,i}}\right)\left(\sum_{i=1}^{m}\frac{\beta_i\bigtriangledown G_{k-1,i}^T}{G_{k-1,i}}\right)+ \qquad (3.6)\right. \\
&\left.\sum_{i=1}^{m}\frac{\beta_i}{G_{k-1,i}^2}\left(G_{k-1,i}\bigtriangledown^2 G_{k-1,i}-\bigtriangledown G_{k-1,i}\bigtriangledown G_{k-1,i}^T\right)\right\}
\end{aligned}$$

We will prove that L is a convex function by showing that the matrix $\bigtriangledown^2 L$ is positive semidefinite. Since the first term is easily seen to be positive semidefinite, the function L is convex if $(G_{k-1,i}\bigtriangledown^2 G_{k-1,i}-\bigtriangledown G_{k-1,i}\bigtriangledown G_{k-1,i}^T)$ is positive semidefinite. We will now show this by proving the following result, by induction and the proof of Theorem 1 follows as an immediate consequence. The matrix $\left(G_k\bigtriangledown^2 G_k-\bigtriangledown G_k\bigtriangledown G_k^T\right)$ is positive semidefinite for all $k\geq 0$.

Basis case Consider a zeroth order generalized posynomial given by

$$G_0=\sum_{i=1}^{p}\omega_i\prod_{j=1}^{n}e^{a_{ij}z_j}=\sum_{i=1}^{p}h_i,$$

where $h_i=\omega_i\prod_{j=1}^{n}e^{a_{ij}z_j}$. It is easy to see that the value of each h_i is positive for all $\mathbf{z}$; this observation is used later in the proof.

Now consider the matrix $H=\left(G_0\bigtriangledown^2 G_0-\bigtriangledown G_0\bigtriangledown G_0^T\right)$. The $(q,l)^{\text{th}}$ term of this matrix is given by

$$\begin{aligned}
H_{ql} &= \left(\sum_{i=1}^{p}h_i\right)\left(\sum_{i=1}^{p}h_i a_{iq}a_{il}\right)-\left(\sum_{i=1}^{p}h_i a_{iq}\right)\left(\sum_{i=1}^{p}h_i a_{il}\right)\\
&= \sum_{i=1}^{p}\sum_{j=1,j\neq i}^{p}\left[h_i h_j\left(a_{iq}-a_{jq}\right)\cdot a_l\right]\\
&= \sum_{i=1}^{p}\sum_{j=i+1}^{p}\left[h_i h_j\left(a_{iq}-a_{jq}\right)\cdot\left(a_{il}-a_{jl}\right)\right]
\end{aligned}$$

Therefore, we can write

$$H = \sum_{i=1}^{p} \sum_{j=i+1}^{p} h_i h_j \, (\vec{a}_i - \vec{a}_j) \cdot (\vec{a}_i - \vec{a}_j)^T$$

where $\vec{a}_i = [a_{i1},\ a_{i2},\ \cdots a_{in}]^T$. Therefore, H is positive definite since each $h_i > 0$.

Induction hypothesis: For a generalized posynomial $G_{k-1}(\mathbf{z})$ of order $k-1$, where $k \geq 1$,

$$G_{k-1}(\mathbf{z}) \bigtriangledown^2 G_{k-1}(\mathbf{z}) - \bigtriangledown G_{k-1}(\mathbf{z}) \bigtriangledown G_{k-1}(\mathbf{z})^T$$

is positive semidefinite.

For the inductive step, we write

$$G_k = \sum_{i=1}^{r} L_{k,i} = \sum_{i=1}^{k} P_i \prod_{j=1}^{m_i} (G_{k-1,i,j})^{\beta_{i,j}}, \tag{3.7}$$

so that each $L_{k,l}$ is of the form of the function L defined in Equation (3.4). We may use the expressions for the gradient and Hessian of L in Equations (3.5) and (3.6) to write

$$\begin{aligned}
& G_k \bigtriangledown^2 G_k - \bigtriangledown G_k \bigtriangledown G_k^T \\
= & \left(\sum_{l=1}^{r} L_{k,l} \right) \left(\sum_{l=1}^{r} \bigtriangledown^2 L_{k,l} \right) - \left(\sum_{l=1}^{r} \bigtriangledown L_{k,l} \right) \left(\sum_{l=1}^{r} \bigtriangledown L_{k,l} \right)^T \\
= & \sum_{l=1}^{r} \sum_{q=1}^{r} \left(L_{k,l} \bigtriangledown^2 L_{k,q} - \bigtriangledown L_{k,l} \bigtriangledown L_{k,q}^T \right)
\end{aligned}$$

If we set

$$\vec{u}_j = \sum_{j=1}^{m} \frac{\beta_j \bigtriangledown G_{k-1,i,j}}{G_{k-1,i,j}}, \tag{3.8}$$

this may be rewritten as

$$\begin{aligned}
& \sum_{l=1}^{r} \sum_{q=1}^{r} L_{k,l} (L_{k,q} \{ \vec{u}_q \vec{u}_q^T + \sum_{i=1}^{m} \frac{\beta_i}{G_{k-1,q,i}^2} (G_{k-1,q,i} \bigtriangledown^2 G_{k-1,q,i} - \\
& \bigtriangledown G_{k-1,q,i} \bigtriangledown G_{k-1,q,i}^T) \}) - L_{k,l} L_{k,q} \vec{u}_l \vec{u}_q^T
\end{aligned}$$

$$= \sum_{l=1}^{r}\sum_{q=1}^{r} L_{k,l}L_{k,q}\sum_{i=1}^{m}\frac{\beta_i}{G_{k-1,q,i}^2}(G_{k-1,q,i}\bigtriangledown^2 G_{k-1,q,i} -$$

$$\bigtriangledown G_{k-1,q,i}\bigtriangledown G_{k-1,q,i}^T) + \sum_{l=1}^{r}\sum_{q=l+1}^{r} L_{k,l}L_{k,q}\sum_{i=1}^{m}(\vec{u}_q - \vec{u}_l)(\vec{u}_q - \vec{u}_l)^T,$$

which is positive semidefinite by the induction hypothesis. **QED**.

3.4 Delay modeling using generalized posynomials

Delay estimation is carried out by using precharacterization, which is performed once for every technology. For a given technology, the delay of a gate typically depends on the sizes of transistors in the gate, the input slope, and the output capacitance; and these are used as variables in the characterization procedure. These variables will be referred to as *characterization variables*. The choice of these circuit parameters as characterization variables can be better explained with the help of an example.

Let us consider the timing model for an inverter, such as the one shown in Figure 3.1; this model is generalized to complex gates in subsequent sections. The aim is to be able to estimate delay as a function of the pmos and nmos transistor widths, w_p and w_n, the input transition time τ, and the output load capacitance, C_L. Therefore, for an inverter, w_p, w_n, τ, and C_L form the set of characterization variables. These variables reflect the set of variables that are generally considered to be important in defining the delay of a gate in most models.

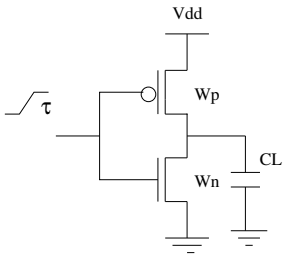

Figure 3.1: Inverter circuit

Several types of functions that are members of the generalized posynomial class, are tried out to achieve the desired levels of accuracy. The general form of expression that provided

consistently good results for different gate types is as follows

$$\text{Delay} = \sum_{j=1}^{m} P_j \cdot \prod_{i=1}^{n} (x_i^{\Delta} + c_{ij})^{\beta_{ij}} + C \tag{3.9}$$

Here, the x_i's are characterization variables, and the c_{ij}'s, β_{ij}'s, C, and P_j's are real constants, referred to collectively as *characterization constants*. The parameter Δ is set to either -1 or 1, depending on the variable, as will soon be explained. The problem of characterization is that of determining appropriate values for the characterization constants. It is shown in Section 3.7.1 that the use of this form of function implies that the circuit delay can be expressed as a generalized posynomial function of the transistor widths.

Due to the curve-fitting nature of the characterization procedure it is not possible to ascribe direct physical meanings to each of these terms. However, it can be seen that the fall delay increases as C_L, w_p and τ are increased, and decreases as w_n is increased, implying that an appropriate choice for the parameter Δ for the first three variables is 1, and that for w_n is -1. Note that this is not as restrictive as the Elmore form since, among other things, the β_{ij}'s and c_{ij}'s provide an additional degree of freedom that was not available for the Elmore delay form. A similar argument may be made for the rise delay case.

A two-step methodology is adopted to complete the characterization. In the first step, a number of circuit simulations are performed, using HSPICE to generate points on a grid. In the second, a least-squares procedure is used to fit the data to a function of the type in Equation (3.9).

A series of simulations is performed to collect the experimental data using the HSPICE circuit simulator. The total number of data points, N, increases exponentially with the number of characterization variables. For the inverter circuit with four characterization variables and d data points for each variable to cover the range of interest, the total number of data points, N would be d^4. Therefore, it is important to choose the data points carefully; in particular, it is not necessary to choose an even grid for the transistor widths and a smaller granularity of points can be chosen for larger w_n's in case of the fall transition.

The determination of the characterization constants was performed by solving the following nonlinear program that minimizes the sum of the squares of the percentage errors over all data points.

$$\text{minimize} \sum_{i=0}^{N} \left[\frac{D_{estim}(i) - D_{actual}(i)}{D_{actual}(i)} \right]^2 \tag{3.10}$$

where N is the number of data points, $D_{estim}(i)$ and $D_{actual}(i)$, respectively, represent the values given by Equation (3.9), and the corresponding measured value at the i^{th} data point. This minimization problem is solved using the MINOS optimization package [20] to determine the values of the characterization constants.

3.4.1 Characterization of a set of primitives

For a library-based design, a full characterization of all cells is a viable alternative and its complexity is comparable to that of characterizing the library using any other means. For general full custom designs, however, the number of SPICE data points to be generated for the curve fit increases exponentially with the number of characterization variables. It is computationally expensive to perform such a large number of simulations and hence an alternative strategy is suggested. An alternative strategy is to precharacterize a set of logic structures such that any gate can be mapped to one of the elements of this set with some acceptable loss of accuracy. It is important to note that even under this procedure, the transistor sizing approach will size each transistor individually, and this method is only used for delay estimation.

One straightforward technique that may be used is to map all of the gates to an "equivalent inverter" [16, 17], and use the inverter characterization to estimate delays; the sizes of the pull-down nmos transistor and the pull-up pmos transistor of this inverter reflect the real pull-down or pull-up path in the gate. The widths of these new transistors are referred to as the equivalent widths. The equivalent width calculation is based on modeling the "on" transistors as conductances, and the equivalent width corresponds to the effective conductance of the original structure. Accordingly, if two transistors of widths w_1 and w_2

are connected in parallel, the equivalent width is defined as $w_1 + w_2$ and if the transistors are connected in series, the equivalent width is defined as $\left[w_1^{-1} + w_2^{-1}\right]^{-1}$.

However, such a reduction has shortcomings. Consider the NAND gate in Figure 3.2(a), whose equivalent inverter approximation is illustrated in Figure 3.2(b). The node capacitances at nodes other than the output are not accounted for in this approximation. Also, the same mapping will be used irrespective of whether input A or B is switching, whereas in reality, these two cases correspond to different delay values. This issue is addressed in Section 3.4.1.

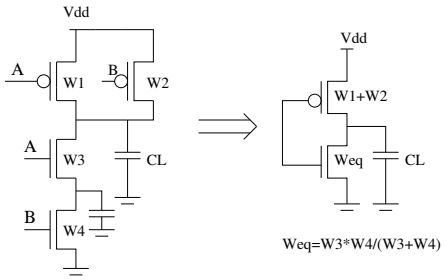

Figure 3.2: Inaccurate mapping of a NAND gate

The mapping procedure developed in this work attempts to reduce the errors caused because of such approximations. To this respect, a basic set of primitives is developed such that any arbitrary gate can be mapped to one of the primitives with acceptable loss of accuracy. Three types of primitives, namely, *simple primitives, complex primitives* and *sequential primitives* are developed to handle various logic structures, for both rise and fall transitions.

Simple primitives

One-input, two-input, and three-input primitives are developed as described below.

Single input primitive is basically an inverter; it is referred to as a primitive because of the fact that mapping procedure along with inverters also maps NOR gates for fall transition and NAND gates for rise transition on an inverter. For example, if we assume single input transitions, then in case of a NAND gate only one of the pmos transistors will be on during the rise output transition. The pmos transistor that is off contributes only as a loading

capacitance, and hence for rise delay calculation, the NAND gate is mapped to an inverter. Since this inverter primitive is identical to the inverter described in previous section it is not discussed any further.

It is necessary to emphasize that an n-input primitive does not mean that it is a primitive only for the n-input gates. Any gate having equal to or more than n inputs would be mapped to an n-input primitive depending upon the position of the switching transistor.

To illustrate this a set of two-input primitives for fall transition at the output is shown in Figure 3.3 (the presence of a load capacitance at the output is implicit and is not shown). We assume that the timing analysis procedure in our tool assumes only single input transitions, and hence there can only be one pair of pmos and nmos transistors switching at a time.

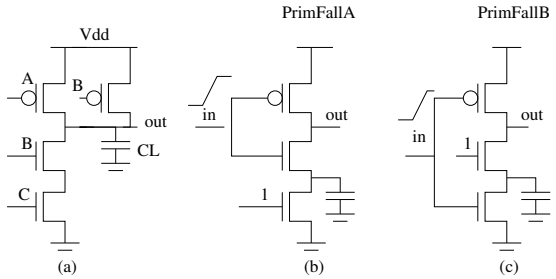

Figure 3.3: Two-input primitives for fall transition

A two-input NAND gate is shown in Figure 3.3(a). For the fall delay, if the input transition occurs at input A, then the gate is mapped to Figure 3.3(b). Note that since the output is being pulled down in the case of a fall delay calculation, the pull-down is retained while the pull-up is replaced by a single transistor, and the characterization equations of Figure 3.3(b) are used to estimate the delay. In a similar fashion, when the input transition occurs at input B of Figure 3.3(a), the gate is mapped to Figure 3.3(c). A similar procedure is applied for rise delays, i.e., the pull-up part is retained while the pull-down part is replaced by an equivalent nmos transistor. Similarly, 2-input primitives, containing two pmos transistors in series with an nmos transistor, namely PrimRiseA and PrimRiseB, are developed that can accurately model NOR gates and NOR gate-like structures.

For simple gates with more than two inputs and complex gates, an expanded set of

primitives is necessary. The set of primitives used to approximate such gates is shown in Figure 3.4. It should be noted that these are not the only primitives on which gates with more than three inputs will be mapped. For example, consider a three input NAND gate and the case where the latest arriving input is the one connected to the topmost transistor in the nmos chain. In this case, the NAND gate will be mapped to the two input primitive PrimFallA shown in Figure 3.3(a); the two nmos transistors at the bottom are collapsed into one transistor of equivalent width.

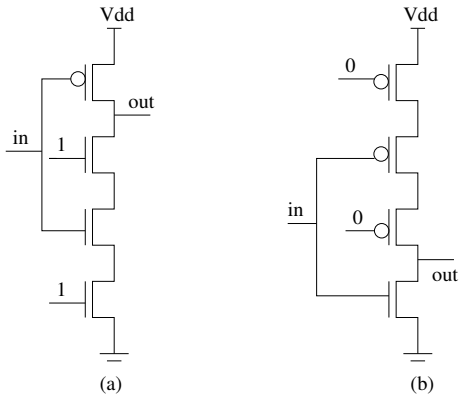

Figure 3.4: Three-input primitives

Complex primitives

One can see that above mentioned methodology can very efficiently handle various simple gates as well as complex gates under certain switching scenarios. However, for the gates with disjoints paths to power supply or ground, such as And-Or-Invert (AOI) gates, the methodology will not always give accurate delays. In case of simple gates with only one transistor chain, the internal node capacitances are inherently taken into account during the modeling phase. But in the case of AOI gates there are multiple parallel chains of transistors. Hence if AOI gates are mapped (except when all the transistors connected to the output and belonging to the nonconducting chains are off) on to the simple primitives developed earlier, then the internal node capacitances in non-conducting transistor chains would not be correctly accounted for, resulting in inaccurate delay values. When the internal node capacitances are charged then these capacitances need to be considered and hence a set of

complex (AOI) primitives is developed. The beauty of this approach is that these primitives can be developed as simple extensions of the primitives of simple gates. For AOI gates, one can observe that the worst case delay corresponds to one conducting chain of transistors between the output and supply, while all other chains are nonconducting. This shows that primitives for AOI gate can be developed by addition of a nonconducting transistor chain in parallel to the transistor chain in the simple gate primitive. A few example primitives for AOI gates are shown in the Figure 3.5.

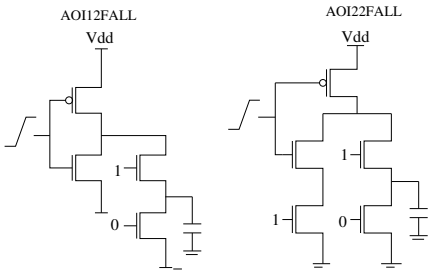

Figure 3.5: Examples of AOI Primitives

For a general purpose arbitrary gate the longest resistive path is obtained to estimate worst case delay. In addition, the side inputs are assumed to have values that will maximize the loading capacitance. Then a systematic reduction to primitives can be obtained by using following rules:

1. If there is a chain of on transistors, then those transistors can be collapsed in an equivalent transistor without loss of much accuracy.

2. The position of switching transistor should decide the primitive to be used for delay evaluation. For example, if there are four nmos transistors in a chain with the topmost transistor switching and all of the remaining transistors on, then the chain can be collapsed to an equivalent transistor. On the other hand if any of the middle transistors is switching, then the transistors on the top can be collapsed in one equivalent transistor, and those at the bottom can be collapsed into another transistor, and the primitive with this structure can be used. The two cases will require different primitives because in the first case, all the on transistors act only as resistances, while in the latter case, the transistors on top have

both resistive and capacitive contributions, while those on the bottom exhibit only resistive effects.

3. If a chain of transistors does not include the switching transistor, then after collapsing into an equivalent transistor, one of the primitives developed for complex gates, for example the primitives showed in Figure 3.5, can be used.

4. If a chain of transistors is on, but one of the ends is floating then the internal capacitances are not charged, and the chain can be safely omitted from consideration, since it cannot have any effect on delay.

If the reduction results into an error above acceptable limits and if there are considerable number of instances of this gate, then it might justity investing efforts in fitting a model for that particular gate, and the fitting procedure mentioned in Section 3.4 can be used.

Sequential primitives

A static sequential element normally consists of a set of pass transistors and inverters. An example sequential element is shown in Figure 3.6. Since an inverter that drives a transmission gate forms a single channel connected component, as shown in Figure 3.6, a separate model is developed for this component. Using this model in conjunction with the inverter model explained earlier, we are now able to model every channel connected component in this sequential element. An advantage of the ability to develop accurate models for the sequential elements is the simplicity in constraint formulation in the across-latch optimization.

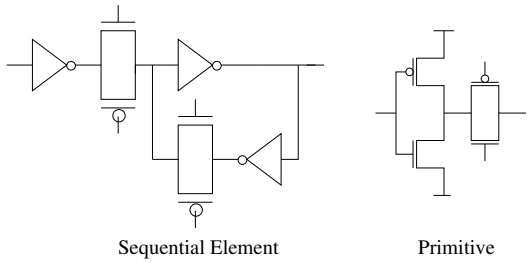

Figure 3.6: Sequential Element and Primitive

Primitive	Delay	
	Mean	Deviation
InvRise	0.31 %	2.84 %
InvFall	1.29 %	2.82 %
PrimFallA	-1.28 %	4.74 %
PrimFallB	1.07 %	2.95 %
PrimRiseA	-0.67 %	3.59 %
PrimRiseB	0.13 %	0.93 %
PrimCoFall	-0.68 %	2.96 %
PrimCoRise	-0.35 %	1.79 %
AOI12Fall	0.87 %	6.27 %
SeqFall	7.46 %	4.73 %

Table 3.1: Primitive Validation

3.5 Modeling Results

Table 3.1 shows the validation results of different primitives, proposed in Section 3.4.1, with respect to SPICE.

The purpose of listing these validation results on the primitives is to emphasize that transistor nonlinearities can be effectively modeled by convex functions and to test the validity of the basic idea of modeling delay as convex functions. Referring to Equation (3.9), a value of $j = 1$ was chosen, and it was observed that the use of higher values for j did not offer significant improvements in accuracy. The characterization was performed in a 0.25μm technology by varying transistor widths to up to 80μm, τ from 20 to 300 ps (10% to 90%) and C_L up to 800 fF. Figure 3.7 shows a typical histogram of the primitive validation.

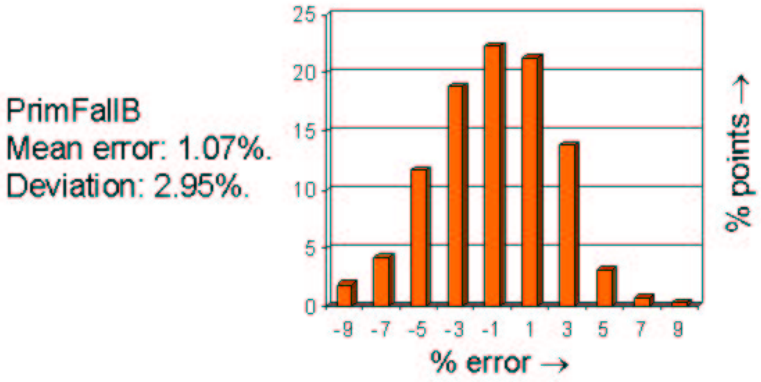

Figure 3.7: Primitive validation

Gate	Delay		
	Output Transition	Mean	Deviation
Inv	Rise	0.31%	2.83 %
	Fall	1.29 %	2.82 %
Nor2	Rise	1.82 %	2.56 %
	Fall	11.10 %	5.06 %
Nand2	Rise	5.18 %	6.17 %
	Fall	-0.46 %	3.58 %
Nor3	Rise	0.24 %	1.76 %
	Fall	24.2 %	7.64 %
Nand3	Rise	9.21 %	5.98 %
	Fall	1.26 %	2.29 %
AOI3	Rise	5.21 %	6.38 %
	Fall	0.86 %	6.27 %

Table 3.2: Gate Validation

It should be noted that accurate fits are required only in the region where sizing constraints are satisfied. For example, if output transition time violates the specification then the optimizer will ensure that its value is reduced to a point in the feasible region, and the convexity of the functions will force the optimization to move to this region after some iterations.

Table 3.2 shows the validation results of various gates with respect to SPICE. It is important to note that that all the possible mappings for a gate are considered and the worst case results are shown in the table. For example, the fall transition on gate Nand3 can map on to either primitive PrimFallA, PrimFallB or PrimCoFall. It was found that PrimCoFall provided the best results, while PrimFallA and PrimFallB provided a smaller degree of accuracy due to the fact that a three input gate was mapped to a two-input primitive using the concept of an equivalent width of two series transistors. Figure 3.8 shows a typical histogram of gate validation.

The results show that the primitive-based gate delay estimation gives highly accurate results. It is observed that all of the errors that are above 2% are obtained when an n-input gate is mapped to a k-input primitive where $k < n$. If some gate type gives unacceptable results for some input transition we can further enhance the accuracy by characterizing the

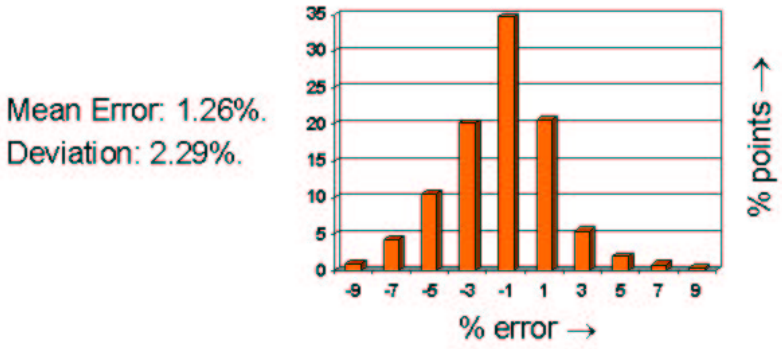

Figure 3.8: Validation of a 2-input NAND gate

model for that specific scenario over a range of parameter values. It can be seen from the table that two of the delay scenarios, namely, fall delay of NOR2 and NOR3, resulted in an error greater than 10%, and that rise delay of NAND3 is close to 10% as well. The reason for these high errors is that an inverter primitive is used to estimate fall delay of a NOR gate and rise delay of a NAND gate. One rise primitive can be developed for NAND gate, and similarly for NOR gate to reduce the errors.

To verify the accuracy of the proposed model on whole circuit, the C17 benchmark from ISCAS85 benchmark suite, is optimized with an accurate convex optimizer [11]. The circuit is optimized for the target delays that vary from 60% to 90% of the unsized delay, where the unsized delay corresponds to the circuit with all transistor sizes set to minimum. The optimized circuit is then timing analyzed using SPICE. The results of the comparison are shown in Table 3.3, where a row represents a circuit optimized by putting the delay constraint provided in the first column. Columns two and three show the SPICE delay of the optimized circuit and the error in the output delay measured by the proposed model as compared to SPICE. The area of the circuit is shown in the last column.

3.6 Conversion of generalized posynomials to posynomials

The generalized posynomial model appears very complex at first and there are no accurate mathematical optimizers available which can operate directly on generalized posynomials. Therefore, it is in order that we examine how the model can be incorporated into conven-

Output Capcitance = 30fF
Delay without any constraint = 934ps

Model Delay (ps)	SPICE Delay (ps)	Error	Area
840	835	0.59 %	6.67
745	752	-0.94 %	7.75
655	670	-2.30 %	10.01
560	594	-6.07 %	14.05

Table 3.3: Comparison of Model Delay for C17 with SPICE

tional mathematical optimizers such as geometric optimizers which are traditionally used to solve posynomial programs. This section shows the detailed mapping of generalized posynomials to posynomials.

As will be shown shortly, the transformation is carried out by the introduction of additional variables. Consider the constrained generalized posynomial below:

$$\sum_i \gamma_i \prod_{j=1}^{m} \left(\sum_{l=1}^{p_i} \omega_{ijl} \prod_{s=1}^{n} x_s^{a_{ijls}} \right)^{\beta_{ij}} \leq D_{req}, \tag{3.11}$$

where D_{req} is the required delay time, and $\beta_{ij} > 0 \ \forall \ i, j$. Note that the term in the parenthesis is a regular posynomial. If we substitute that posynomial by the variable y_j, the result is the constraint

$$\sum_i \gamma_i \prod_{j=1}^{m} (y_j)^{\beta_{ij}} \leq D_{req}. \tag{3.12}$$

It is well known that any constraint where a posynomial function is required to be less than or equal to a constant, is equivalent to a convex set under the variable transformation; and such a constraint is referred here as a posynomial constraint. However, the above substitution also requires that the following equality must be satisfied:

$$\sum_{l=1}^{p_i} \omega_{ijl} \prod_{s=1}^{n} x_s^{a_{ijls}} = y_j. \tag{3.13}$$

This equality can be represented by a pair of inequalities, only one of which is a posynomial

constraint. This implies that the constraint set is no longer transformable to a convex set in the x and y variables.

If we relax the equality in (3.13) into a "$\leq$" inequality, then for any variable assignment that satisfies the relaxed set of constraints, since $\gamma_i > 0$ and $\beta_{ij} \geq 0$, it must be true that

$$\sum_i \gamma_i \prod_{j=1}^{m} \left(\sum_{l=1}^{p_i} \omega_{ijl} \prod_{s=1}^{n} x_s^{a_{ijls}} \right)^{\beta_{ij}} \leq \sum_i \gamma_i \prod_{j=1}^{m} (y_j)^{\beta_{ij}} . \tag{3.14}$$

This subtle observation in conjunction with constraint (3.12) implies that the constraint (3.11) must be satisfied.

Therefore, we may replace each such generalized posynomial constraint of order 1 by the set of inequalities given by

$$\begin{aligned} \sum_i \gamma_i \prod_{j=1}^{m} y_j^{\beta_{ij}} &\leq D_{req} \\ \sum_{l=1}^{p_i} \omega_{ijl} \prod_{s=1}^{n} x_s^{a_{ijls}} &\leq y_j \end{aligned} \tag{3.15}$$

It is instructive to note that this substitution technique may be used, in general, for generalized posynomials of order k. This is easily seen, since the above procedure reduces an order k generalized posynomial constraint to an order $k-1$ generalized posynomial constraint; this process may be carried out recursively until posynomial constraints are obtained.

This is a powerful result, since we now have a large new class of functions that can accurately capture the delay behavior of a circuit, but may use conventional fast methods, which exploit the structure of the problem, for solving them.

3.7 Circuit optimization incorporating the new delay model

It is important to note that the proposed delay model is targeted towards transistor-level optimization, although one can use them in gate-level optimizations as well. The material presented below examines two problems in the domain of circuit optimization, namely, area-delay optimization and dynamic power-delay optimization. The first problem is solved using

a heuristic approach, while the latter is solved using an accurate mathematical optimizer and exploiting the conversion outlined in the previous section. The formulations for both problems are similar and are provided in Section 2.3.

3.7.1 Proof of convexity of path delays

The ensuing discussion shows that the delays of individual paths under the generalized posynomial gate delay model satisfy the property of convexity. It is to be emphasized that this discussion is purely for expository purposes; the optimizers used in this work for sizing does not require the enumeration of all paths, and perform the optimization efficiently by checking, through a timing analysis, whether the constraints are satisfied or not. For details, the reader is referred to [10].

Let the critical path of the circuit be represented by a set of stages, where each stage represents a gate. Let us first consider a scenario with fully characterized gates where no primitives are used, but the delay is characterized in terms of the size of each transistor. Then, substituting the characterization variables explicitly into Equation (3.9), we see that the fall delay of the gate corresponding to stage l has the following form:

$$\begin{aligned}\text{Delay}_l = \sum_i P_i \cdot (w_{n1}^{-1} + c_{n1})^{\beta_{n1}} \cdots (w_{nm_n}^{-1} + c_{nm_n})^{\beta_{nm_n}} \\ (w_{p1} + c_{p1})^{\beta_{p1}} \cdots (w_{pm_p} + c_{pm_p})^{\beta_{pm_p}} (\tau_{i-1} + c_\tau)^{\beta_\tau} \Pi_j (C_j + c_{C_j})^{\beta_{C_j}}\end{aligned} \tag{3.16}$$

and the output fall transition time of the gate in stage l has the form[1]

$$\begin{aligned}\tau_l = Q \cdot (w_{n1}^{-1} + k_{n1})^{\gamma_{n1}} \cdots (w_{nm_n}^{-1} + k_{nm_n})^{\gamma_{nm_n}} (w_{p1} + k_{p1})^{\gamma_{p1}} \cdots \\ (w_{pm_p} + k_{pm_p})^{\gamma_{pm_p}} (\tau_{i-1} + k_\tau)^{\gamma_\tau} \Pi_j (C_j + k_{C_j})^{\gamma_{C_j}}\end{aligned} \tag{3.17}$$

where $P_i > 0$, $Q > 0$, c_{ni},c_{pi},k_{ni},k_{pi},β_{ni},β_{pi}, γ_{ni},γ_{pi} $\forall i, k_{C_j}$,c_{C_j} $\forall j$,k_τ,c_τ, β_{C_j},γ_{C_j},β_τ,γ_τ are real constants. The w_{ni} and w_{pi} values, as usual, refer to the nmos and pmos transistor

[1] The rise delay and rise transition time expressions are similar, with the roles of w_n and w_p interchanged.

sizes, τ refers to the transition time, and the C_j's correspond to the capacitances at the gate output and at internal nodes.

The capacitance at each internal or gate output node i, C_i is modeled by

$$C_i = \sum_j k'_j w_j + k'' \tag{3.18}$$

where the k'_j and k'' values are real constants, and w_j's represent the equivalent transistor widths in the circuit.

From the Equation (3.18) we can see that output transition time is represented by a generalized posynomial. Additionally, the loading capacitance given by Equation (3.18) has the form of a generalized posynomial. Using Theorem 1 from Section 3.3.2, it can be seen that when the input transition time and loading capacitance expressions are substituted in Equation (3.17), the resulting expression is also a generalized posynomial.

For area-delay trade-off problem, the objective function is the sum of the transistor sizes, which is clearly a generalized posynomial form. For the power-delay trade-off problem the dynamic power is expressed as Equation (2.2). It can be easily verified that this equation conforms to the genearalized posynomial form.

Using identical arguments to [10, 11], since the maximum of convex functions is convex, the problem of area or power minimization under delay constraints for "template" gates can be shown to be a convex programming problem. For gates that do not adhere to the template, the mapping techniques described in Section 3.4.1 may be used to model the delay function. We will now show that in such a case, the delay function continues to remain in the generalized posynomial form. Let $w'_1, \cdots, w'_m$ represent transistor widths in the primitives the gates are mapped to. In the process of mapping the gates, the transistor widths in the primitives can be expressed in terms of the actual transistor widths in the circuit. Let $w_1, \cdots, w_n$ represent the actual transistor widths in the circuit. Then w''s can be expressed as

$$w'^{-1}_i = \sum_{q \in \{1 \cdots n\}} w_q^{-1}, 1 \leq i \leq m \tag{3.19}$$

All occurrences of value of $w_i'^{-1}$, which is a basic variable in the characterization equation, can be substituted as above in Equation (3.17), maintaining the generalized posynomial property of the delay equation.

3.7.2 Area – delay trade-off optimization

There have been many significant attempts to solve this problem, for example, [10, 11, 21]. Most published approaches use the Elmore delay model [5] for timing calculations, and a breakthrough observation in [10] was that the circuit delay under this model is a posynomial function of the transistor sizes. Recently an accurate technique for circuit optimization has been presented in [22], using a nonlinear optimization technique. Simulation followed by time domain sensitivity computation is used to provide gradients to a nonlinear optimizer. However the drawback, as they have noted, is the possible destruction of convexity due to transient-simulation-based modeling. Other drawbacks are the involvement of circuit-level simulation which tend to make the optimization process a bit cumbersome, and the pattern-dependent nature of the timing analysis which requires the user to supply the simulation vectors.

The approach in this thesis tries to eliminate these drawbacks by incorporating the proposed accurate and convex delay models. A sensitivity-based heuristic is used to solve the optimization efficiently and with reasonable accuracy. The delay models were incorporated into the TILOS algorithm described in [10] in a C program. The results of running the algorithm on various test circuits are shown in Table 3.4. The cost function is set to be the area of the circuit, estimated as the sum of the transistor sizes. We first measured unsized delays using our model. The circuits are then optimized for target delays of 70% to 95% of the unsized delay. The results show that the proposed convex model, in addition to being very accurate is also computationally efficient when used in the inner loop of a TILOS-like iterative transistor sizing algorithm.

Circuit	Unsized Delay (ns)	Unsized Area (μm)	T_{spec} (ns)	Sized Area (μm)	Execution Time (s)
C432	2.517	403.5	2.391	458.09	69
			2.265	499.39	130
			2.175	595.06	173
			2.136	603.08	179
			2.013	787.57	270
C880	2.295	721	2.180	722.63	4
			2.066	727.74	11
			1.950	735.34	21
			1.836	752.94	42
			1.721	775.31	65
C499	3.644	1023	3.462	1023.35	3
			3.279	1025.68	9
			3.097	1031.27	18
			2.915	1048.99	51
			2.733	1104.73	166

Table 3.4: Results of area-delay tradeoff optimization

3.7.3 Dynamic power – delay trade-off optimization

Need for power-delay trade-off optimizations has accentuated in the recent generations due to increasing heat dissipation which can be attributed in turn to increasing number of transistors per chip. Several approaches that perform circuit level optimization for area or power have been published. A linear programming based method for gate sizing for power-delay tradeoffs, using a piecewise linear gate delay model, is presented in [23]. Such a model is simplistic and inaccurate in the present deep submicron regime. In [24], the power optimization problem is solved by transistor sizing and ordering. Power dissipation is modeled accurately by incorporating fanout capacitances and gate transition measure. A "pin-delay" model is developed based on the delay model used in SIS. However, in the absence of any convexity properties, they use heuristic techniques to solve the problem. The evident drawback of employing a heuristic approach is that there is no guarantee that the solution will be optimal, even after application of refinement techniques. In [25], only transistor reordering is used for the power-performance optimization, but the important aspect of transistor sizing is not considered. However, drawbacks include the destruction

of convexity properties due to transient simulation-based modeling, the need for circuit-level simulation which tends to make the optimization computationally intensive, and the pattern dependent nature of the timing analysis which requires the user to supply the simulation vectors. Thus, although several attempts have been made to solve the problem, the lack of convexity based techniques tends to keep the results away from optimality. The solution is to use the models that posses accuracy as well as convexity properties and hence lend themselves to accurate optimization techniques, and hence an optimization approach that can derive benefits from the convex delay models presented earlier is developed in the ensuing discussion.

In this problem we consider dynamic power dissipation, and the model used is introduced in Section 2.1 and is given below for ease of explanation.

$$P_{i_{sw}} = \frac{1}{2} \cdot V^2 \cdot C_i \cdot TD_i \tag{3.20}$$

where $P_{i_{sw}}$ is the average switching power dissipation, V is power supply voltage, C_i is the output capacitance, and TD_i is the transition density, all corresponding to gate n_i. The transition density is defined [26] as $\lim_{T\to\infty} n(T)/T$, where $n(T)$ represents the number of transitions the gate performs in time T. The use of transition density allows a gate with lower switching probabilities to be sized larger. The short circuit power is known to be dependent on the input transition time to a great extent, and hence can be controlled by placing constraints on the input transition time for each gate. Equation (3.20) also has the generalized posynomial form and the proof is similar to the one that shows convexity of the delay model.

Having shown that both power and delay model have generalized posynomial form, we can use the conversion outlined in previous section to formulate the problem as a posynomial program. A geometric optimizer is then used to solve this problem, and unlike the convex optimizer in [7], it requires all constraints to be explicitly listed. This may be carried out efficiently while preserving the posynomial nature of all constraints.

3.7.4 Introduction of intermediate variables

As mentioned in Section 3.7.1, the path delay can be proved to be a convex function. However, the number of input to output paths increases exponentially with an increase in the number of gates in the combinational logic. In this work path enumeration is avoided by the introduction of intermediate variables. Following variables are introduced for each gate, n_i:

D_{i_r} : Arrival time at the output of gate n_i for the rise transition at the output.

D_{i_f} : Arrival time at the output of gate n_i for the falling transition at the output.

τ_{i_r} : Rise transition time at the output of gate n_i.

τ_{i_f} : Fall transition time at the output of gate n_i.

In addition, variables are introduced to model the pin to pin delay. D_{ij_r} represents the delay from gate n_i to gate n_j for the rise transition at the output of gate n_j. Similarly, D_{ij_f} represents delay from gate n_i to gate n_j for the fall transition at the output of gate n_j. To illustrate the constraint formation, consider a subcircuit shown below.

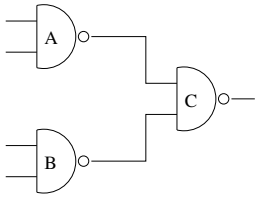

Figure 3.9: An example circuit

Then the constraints related to gate C are,

$$\begin{aligned}
(D_{a_f} + D_{ac_r})/D_{c_r} &\leq 1 \\
(D_{a_r} + D_{ac_f})/D_{c_f} &\leq 1 \\
(D_{b_f} + D_{bc_r})/D_{c_r} &\leq 1 \\
(D_{b_r} + D_{bc_f})/D_{c_f} &\leq 1
\end{aligned} \tag{3.21}$$

The above set of equations represents the constraints on the arrival time at the output of the gate C. All the gates are assumed to be inverting, but similar equations can be derived if noninverting gates are used. These equations along with the equations resulting from the conversion of generalized posynomials corresponding to $D_{ac_r}, D_{ac_f}, D_{bc_r}, D_{bc_f}$, to regular posynomials, and the constraints imposed on the output delay, form the complete set of posynomial constraints to be fed to the geometric optimizer.

Circuit	Tran Count	Dout_{spec} (ps)	τout_{spec} (ps)	Power (mw)
inv10	20	720	400	0.191
		648	360	0.214
		576	320	0.261
		500	280	0.443
c17	24	600	400	0.634
		540	360	0.705
		480	320	0.804
		420	280	0.959
s27	42	900	400	0.648
		810	360	0.739
		720	320	0.891
		630	280	1.314
comb1	70	900	450	0.448
		800	400	0.546
		720	360	0.757
		640	320	1.574
comb2	200	1669	360	0.634
		1502	324	0.717
		1335	288	0.922
		1168	251	2.062
s298	582	1027	350	3.705
		926	315	4.141
		720	245	5.877

Table 3.5: Results of power-delay tradeoff optimization

A transistor sizing optimization tool which integrates constraint generation program and a generalized posynomial solver is implemented in C and C++. The constraint generation program writes the objective function and constraints in the generalized posynomial form. The constraints are then converted into posynomial constraints, using the procedure for

conversion of a generalized posynomial to a regular posynomial as described in Section 3.6. The solver is extremely efficient since it utilizes the properties of geometric programs to arrive at a solution. Several circuits have been optimized for power, placing constraints on the input-to-output delay. The circuits are optimized for varying target delay and output transition time constraints. The results are tabulated in Table 3.5. The first two columns show the test circuit names and transistor counts. The third column titled Dout_{spec} represents the input-to-output delay constraint, while the fourth column titled τout_{spec} represents the transition time constraint placed on the output. The last column represents the power obtained after optimization. The execution times of optimization, using geometric program solver, vary from about a second for very small circuits, to about 8 minutes for moderately sized circuits. As expected, the power dissipation increases as the constraints are made tighter.

3.8 Conclusion

A new delay model for CMOS gates is introduced in this chapter. The model is better suited for modern technologies than the Elmore model, and yet maintains the convexity properties. A new class of functions called generalized posynomials is proposed and its members are shown to have the same relation to convex functions as posynomials. It is shown that members of this new class of functions can accurately model gate delay. A new mapping technique has been presented that can be used to convert generalized posynomials to regular posynomials enabling the use of conventional geometric optimizers. The model is incorporated in various optimizers and its use in the context of optimization for area-delay trade-off and power-delay trade-off is demonstrated on several circuits. Results show that in addition to possessing the useful property of convexity, the model is also computationally efficient.

Chapter 4

Leakage – Delay Trade-off Optimization

4.1 Introduction

The scaling of supply voltage and the accompanying reduction of transistor threshold voltage (V_t) has helped in achieving about a 30% reduction in circuit delay from one technology generation to another [7]. However, in future technologies, V_t will not scale in proportion to the supply voltage, and this will result in a lower reduction in circuit delay between generations. A strategy to improve circuit timing has taken shape in the form of dual V_t technology, where a transistor can have either a fixed low V_t or a fixed high V_t. The choice of V_t involves a tradeoff since the high V_t leads to higher gate delay and lower standby leakage power dissipation while the low V_t leads to faster gates but with drastically higher standby leakage power dissipation. However, unlike transistor sizing which achieves delay reduction at the cost of increased area, V_t selection has the advantage of allowing timing improvements with no area overhead.

Section 4.2 presents a brief summary of previously proposed approaches for V_t assignment and for simultaneous sizing and V_t assignment. The problem of leakage-delay trade-offs in formulated in Section 4.3, and the techniques for estimating leakage and delays are

presented in in Section 4.4. The V_t assignment approach is presented in Section 4.5, and the simultaneous optimization is presented in Section 4.6. This is followed by implementation details in Section 4.7 and concluding remarks in Section 4.8.

4.2 Previous work

Static power-delay tradeoff optimization by dual V_t assignment has been presented in many works. A circuit graph enumeration approach presented in [27] involves successive alteration of gate V_t's from low to high, while the work in [1] discusses different V_t assignments that are suitable for manufacturing, and presents a heuristic method involving a backward traversal- or priority-based method. In [28], an approach similar to retiming is used to assign V_t's.

An important drawback in [27, 28] is that V_t assignment is performed at the gate-level, i.e., all of the transistors in a gate are forced to have the same V_t, as opposed to considering each transistor separately for assignment, and this eliminates a significant number of good circuit configurations.

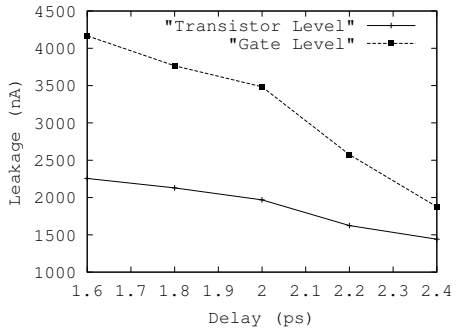

Figure 4.1: Transistor-level versus gate-level V_t assignment for C2670

As an example, let us consider the V_t assignment results on the C2670 benchmark circuit for the circuit configuration where all of the transistor widths are set to minimum size. A comparison of the leakage current-delay tradeoff curve under gate-level V_t assignment (where all of the transistors in a gate must be assigned the same V_t value) and under transistor-level V_t assignment (where V_t assignments for transistors in a gate are carried out independently)

is shown in Figure 4.1. The former yields a solution that is provably inferior to (or at best, equal to) the latter, and experimental results reinforce the point that substantial gains are obtainable from transistor-level V_t assignment. This acts as a motivation for presenting an algorithm that works at the transistor-level. However, in practice, process constraints may place restrictions on the assignment of V_t's to individual transistors (for example, it may be required that transistors in a pull-up or a pull-down have the same V_t). The method proposed in this thesis is general enough that it can tackle additional constraints such as assignment of the same V_t to a group of transistors. In fact, such a restriction could work in favor of the proposed approach as it reduces the number of variables in the optimization problem and further simplifies it. Experimental results with both individual V_t assignment and collective assignment are provided in the result section. The proposed approach is based on explicit enumeration of all the design solutions, made efficient by the use of provable pruning techniques. For each gate, it considers all possible V_t settings for the transistors within that gate, and stores all nonsuboptimal solutions at its output. The only heuristic part of our approach involves breaking down the circuit graph into fanoutfree regions, a technique traditionally employed in technology mapping algorithms.

Although V_t assignment permits great reductions in circuit delay, it may not prove to be the best approach for timing optimization if used alone, since it could entail a huge penalty in leakage power. Nevertheless, the V_t assignment algorithm finds its use in scenarios where postplacement perturbations are heavily constrained and the only way to go for delay reduction is either dual V_t assignment or the use of other recently presented techniques such as rewiring.

As mentioned in previous chapters, transistor sizing is another powerful method that has been used traditionally to contain delays. However, in recent generations, the effectiveness of transistor sizing has been limited because of severe constraints that restrict the area, either directly or through indirect considerations such as dynamic power dissipation. This, together with a need to contain leakage power, implies that a meaningful optimization must carry out sizing simultaneously with V_t assignment. The gate delay as well as the

leakage current depend on the V_t and the transistor size, and hence optimizing V_t without considering transistor sizing leaves a significant part of the design space unexplored. The major drawback of [1, 27, 28] is that the dual V_t assignment is carried out without any consideration of transistor sizing.

This problem of circuit optimization by V_t assignment and transistor sizing was first introduced in [29], which used a simplistic binary search technique. The work in [2] considerably enhances prior work in removing several of the limitations. It employs a a sensitivity-based optimizer that begins with an initial circuit configuration in which all transistors are set to a high V_t and are sized to obtain the best delay under a specified area. In every iteration, the optimizer evaluates each high V_t transistor on the critical path to obtain the ratio of its delay improvement to leakage power increase if it were to be changed to low V_t. The transistor with the largest such ratio is set to low V_t, and the circuit area is then redistributed to recover performance. However, this approach is purely heuristic and lacks an accurate mathematical formulation, due to which this may be far from optimal; in particular, the area redistribution may not work well in the presence of stringent constraints or interacting critical paths.

This chapter examines the circuit optimization problem of minimizing standby leakage power under area and timing constraints. During the design process, a circuit is expected to meet a set of delay constraints and to occupy an area that is no more than that allotted to it during floorplanning, and hence this formulation has a closer correlation to reality as compared to all previously proposed approaches. The variation in the dependence of the transistor drain current on V_t in the ohmic and subthreshold regions and the nature of a typical area-delay tradeoff curve are used to draw a set of conclusions that form the theoretical underpinnings of our approach. A solution approach based on the above mentioned conclusions is then presented; it involves sizing as the first step and the proposed V_t assignment approach as the second step. Since the proposed V_t assignment approach works at transistor level, it fits in well with the transistor sizing, and thus leads to efficient exploration of the design space at transistor level. Sizing, a continuous optimization, is

performed by using a fast sensitivity-based optimization engine.

Sizing is a continuous optimization problem, and is solved using a sensitivity-based optimization procedure. V_t assignment, on the other hand, is a discrete optimization problem, and hence leaves room for fine tuning of continuous variables, which we perform using a sensitivity-based sizing in postprocessing step.

4.3 Problem formulation

If we represent a high V_t as 0 and a low V_t as 1, then the V_t assignment, X, of a gate, can be encapsulated as a binary number whose q^{th} bit corresponds to the V_t of the q^{th} transistor in the gate. For example, for an m-input static CMOS gate, j, the binary number $X_j = X_{j1}X_{j2}\cdots X_{j2m}$, where X_{ji} is the V_t of the i^{th} transistor of gate j. A gate is then defined to be in a V_t-state (not to be confused with an input state, to be defined later) k if k is the decimal equivalent of the binary number, X_j, representing the V_t settings of the gate j.

The V_t assignment and transistor sizing problem can then be formulated as follows using Equation (4.1), which together with leakage current equations introduced in the next section and the delay model equation presented in Appendix A, states the above problem in terms of optimization variables X_{ji} and w_{ji}. For easy readability, constraints are shown without listing rise and fall times separately, although these considerations are incorporated in our implementation. Likewise, constraints for the transition time are not shown but can be similarly stated, and are included in the implementation.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{all\ gates\ j} I_{leak_j} \\
\text{subject to} \quad AR_j &\leq D_{target} \ \forall Primary\ outputs j \\
AR_i &= 0 \ \forall Primary\ inputs i \\
AR_i + f(X_j, k) \times D_{ij}^k &\leq AR_j \ \forall \{i, k\} | \\
i \in fanin(j), \quad k &\in V_t - states\ of\ j \\
X_{ji} &= \{0, 1\} \ \forall i |
\end{aligned}
$$

$$\begin{aligned} & & & & & i \in trans(j), \forall gates\ j \\ & \sum_{all\ gates\ j} \sum_{all\ i \in trans(j)} & w_{ji} & \leq & A_{target} \end{aligned} \tag{4.1}$$

where I_{leak_i} = leakage current of gate i

AR_i = arrival time at the output of gate i

X_{ji} = binary variable associated with the V_t of transistor i of gate j

D_{ij}^k = delay from gate i to gate j, under the V_t-state k,

w_{ji} = width of the transistor connected to input gate i of gate j

D_{target} = target delay at the output

A_{target} = target area

$f(X_j, k)$ is a function of binary V_t variables that correspond to transistors on the paths to the supply nodes from the transistors connected to switching input. The gate delay D_{ij}^k depends on the widths of transistors in gate j, widths of transistors in fanout of gate j, the assignment corresponding to V_t-state k, and the transition time at the output of gate i, which depends on widths of transistors in the gate i and its fanin cone. The first constraint ensures that the arrival time at the circuit outputs should be less than the target delay, D_{target}, while the second constraint sets the input arrival times to zero. The third constraint states that for each V_t-state k and for each input of the gate, the arrival time at its output should be greater than or equal to the arrival time at the input plus the delay from that input to the output under the V_t-state k. The fourth and the fifth constraints, respectively, enforce the conditions that the X_{ji}'s are binary variables, and that the sum of all transistor widths should be less than a target area.

It can be seen that the problem represented by Equation (4.1) is in the form of a mixed integer nonlinear program (MINLP) introduced in Chapter 2. This is one of the most

computationally expensive problems in the realm of mathematical programming, and our experiments showed that the use of an accurate and exact MINLP solver [30] required large CPU times even on small circuits. Therefore, a computationally efficient, yet accurate, heuristic is essential.

4.4 Leakage and delay estimation

4.4.1 Leakage estimation

The effect of V_t on leakage current can be seen from the BSIM leakage current model equations for a single transistor provided in Section 2.1.3. Equation (2.3) shows that the single transistor leakage current is exponentially related to the transistor V_t. This model provides a closed form expression for leakage current and thus allows static optimization.

Now, for the purpose of dual V_t optimization, the leakage model of Equation (2.3) can be multiplied with the probability that the transistor is leaking, and can be represented using the terminology introduced in previous section, as

$$I_{leak_j} = \sum_{all\ gates\ j} \sum_{all\ i \in input(j)} K'' \times P_{ji} \times w_{ji} \times e^{X_{ji} K'} \tag{4.2}$$

where K'' and K' are constants, and P_{ji} is the probability that the transistor i of gate j is off and the input state (to be defined shortly) is dominant [2]. We obtain the constants K' and K'' for a given technology, separately determined for nmos and pmos transistors, using curve fitting over a set of SPICE-generated data points. It should be noted that K' and K'' can be obtained directly from Equation (2.3), and have a physical significance, which is often not the case for fitted equations. Since it is complicated to extract I_o and V_{therm} from SPICE technology files and to get accurate value of V_{ds}, the fitted approach is used instead to determine K' and K''.

The evaluation of the leakage current of a gate can be made efficient by understanding its leakage behavior. Let the input state of a gate be a possible combination of the logical states of all of the inputs of that gate. In static CMOS, in each input state, a set of transistors is on and the set of complementary transistors is leaking. To see why Equation (2.3) is

sufficient to estimate leakage even for a *chain of transistors*, let us look at the concept of dominant leakage states that are formally defined in [2]. Stated in plain English, dominant leakage states for simple gates refer to the leakage states with only one transistor off in the pull-up or pull-down chain. It has been shown that the dominant leakage states account for 95% of the total leakage for equal input state probabilities. Therefore, if an input state is not dominant, then its subthreshold leakage current is negligible. This observation allows us of the single transistor BSIM leakage current model for leakage power estimation, since for a dominant state, each stack contains only one transistor in the subthreshold region.

4.4.2 Delay estimation

The dependence of transistor switching current on V_t is examined in detail in section 4.6.1. For the present treatment of the delay estimation, it will suffice to note that the gate delay increases as V_t increases. The gate delay model proposed in Chapter 3 is used for delay estimation. In addition to being accurate in deep submicron and nanometer technologies, the model also possesses convexity properties for a fixed value of V_t. It is desirable to preserve these convexity properties so that they can be exploited by the area-delay tradeoff optimization, which is the first step in our solution approach, to arrive at a good solution.

However, when V_t is varied, even under a simple model for inverter delay [31], it is easily verified that the delay is not a convex function of the transistor V_t (even assuming that V_t could be changed continuously). Since the objective is to capture the effect of V_t on delay while maintaining accuracy and convexity properties, V_t is not used as a characterization variable in the convex delay model, but instead, multiple pin-to-pin models are developed for each switching scenario. The V_t's of transistors on the pull-up and pull-down resistive path in a switching scenario have a varying, albeit visible, effect on the delay, and hence we develop multiple equations for each switching scenario, with each equation corresponding to a different V_t-states. For example, in case of an inverter there are total four different threshold voltage combinations, corresponding to low and high V_t assignments on the nmos and pmos transistors, and hence four different model equations are developed for each switching

scenario. Transition time models are developed similarly. The models are characterized once for each technology. The precharacterization procedure involves development of a set of primitives such that every gate can be mapped to one of those primitives with acceptable loss of accuracy [32]. A primitive typically involves all of the transistors that lie on the paths from switching transistors to output and supply nodes. Primitive development not only avoids development of separate models for each new gate type but also obviates the need for generating 2^m models for a general m-input gate, for large m.

Although, Chapter 3 provides details of using primitives in single V_t configurations, the usage can be easily extended to dual V_t configurations. The extension to more than two V_t configurations could require development of many more primitives. To avoid this, for the dual V_t case, accurate equations are developed to collapse two transistors of different V_t's into only one "equivalent" transistor of either high or low V_t. The collapsing procedure is validated on an extensive set of data, and delay values were seen to be within 6% of the actual delay. However, if the accuracy of the mapping is not acceptable, one can characterize all of the available gate types. Since this is a one-time procedure for a given technology, the effort involved is comparable to that of characterizing a library.

The experiments performed showed that the effect of the V_t's of leaking transistors on the switching delay of the previous stage, which is primarily manifested in the form of a change in loading capacitance, is extremely small, and can safely be neglected. Moreover, in nanometer technologies this change is nearly overshadowed by the interconnect capacitance. On a 0.1μm technology [33], this delay change was found to be under 2% in most cases.

4.5 Dual V_t assignment

The proposed method employs an enumeration-based approach to explore the design space at the transistor level, and the method is made efficient by the use of effective pruning techniques enumerated following the presentation of the algorithm.

4.5.1 Algorithm

A level is assigned to each gate using the topological sort algorithm [34]. In other words, primary inputs of the fanout-free region correspond to level 0, while a gate is at level i if the maximum level among all its inputs is $i-1$. The enumeration, which starts from the primary inputs and proceeds in the order of increasing level, processes every gate for its all possible V_t-states, and calculates the corresponding arrival times and leakage.

A combinational circuit is represented as a directed acyclic graph where each node corresponds to a gate and each edge corresponds to an interconnection between gates. The graph is decomposed into fanout-free regions (as in technology mapping algorithms such as DAGON [35]) and the enumerative techniques are used to select V_t's within each such region.

At any particular gate, the enumeration stores all solutions that are not suboptimal. This is achieved by maintaining a set of tuples of the form $\{X, L, AR, AF, IS\}$, where X is the V_t-state of the gate under consideration, L is the total leakage of the tree rooted at the node representing current gate. IS represents the set of input tuples (where one tuple corresponds to each input), and AR and AF are the rise and fall arrival time at the output of the gate just processed. The accurate gate delay model also uses the input transition time as a variable. Hence the transition times are included in the tuples as well, but are omitted from the discussion for simplicity.

Figure 4.2 provides the outline of the V_t assignment algorithm. The routine get_new_tuple generates a tuple for the gate under consideration based on its input tuple set and the current V_t-state. The output tuple set is generated by setting the arrival times at the gate inputs to those corresponding to the input tuples, and then calculating the arrival times, AR and AF, at the gate output and the corresponding leakage under allowable V_t-settings for the gate. If a generated tuple is not provably suboptimal (explained in Section 4.5.2), it is inserted in the tuple set of the gate j.

When a multifanout node is reached, algorithm process_tree is used to assign V_t's to the nodes belonging to the fanout free tree rooted at the multifanout node. At multiple

```
1. Algorithm : assign_vt
2.     for each level i
3.         for each gate j in level i
4.             for each V_t-state k
5.                 for each set of tuple combinations
                   at the input nodes of j
6.                     get_new_tuple();
7.                 prune_tuple_set();
8.                 if multifanout gate
9.                     process_tree();

11. Algorithm : get_new_tuple
12.         AR_j = AF_j = L_j = 0;
13.         L = leakage(j);
14.         for each input k of gate j
15.             evaluate AR_kj = AF_k + DR_kj;
16.             evaluate AF_kj = AR_k + DF_kj;
17.             L_j = L_j + L_k;
18.             check for suboptimality;
19.             if provably suboptimal then continue;
20.         insert current tuple in the set of tuples,
            tuple_set(j)

21. Algorithm : prune_tuple_set
22.     for each tuple j in the set
23.         for each tuple i|i > j, in the set of tuples
24.           if AR_i ≥ PF × AR_j and AF_i ≥ PF × AF_j
25.             if L_i ≥ PF × L_j then remove
26.             tuple i from the set;
27.           elseif AR_j ≥ PF × AR_i and AF_j ≥ PF × AF_i
28.               if L_j ≥ PF × L_i then remove
29.               tuple j from the set;
30.               proceed to next tuple;

31. Algorithm : process_tree
32.     for each tuple at the node
33.         if AR_j < RR_j and AF_j < RF_j
34.           if L < L(mintuple)
35.               mintuple = current tuple;
36.     assign X(mintuple) to current gate;
37.     input_queue insert IS of mintuple;
38.     while input_queue is not empty
39.         current_tuple = pop input_queue;
40.         current_gate = gate of current_tuple;
41.         if V_t assigned to current_gate, next;
42.         assign X(current tuple) to current_gate;
43.         if IS not empty insert IS in input_queue;
```

Figure 4.2: Pseudocode for V_t assignment

fanout node, the assignment of V_t is chosen based on the required time constraints that are calculated from the required times at the outputs. At root node, the tuple that has the lowest leakage among all of the tuples that meet the arrival time constraints at that node is selected. If none of the tuples can meet the constraints, the tuple that has minimum offset from the required arrival time is selected. Based on the ID's of the tuples at each of its input nodes, the inputs are assigned the corresponding V_t settings, and the backward traversal is performed until the tree has been exhausted.

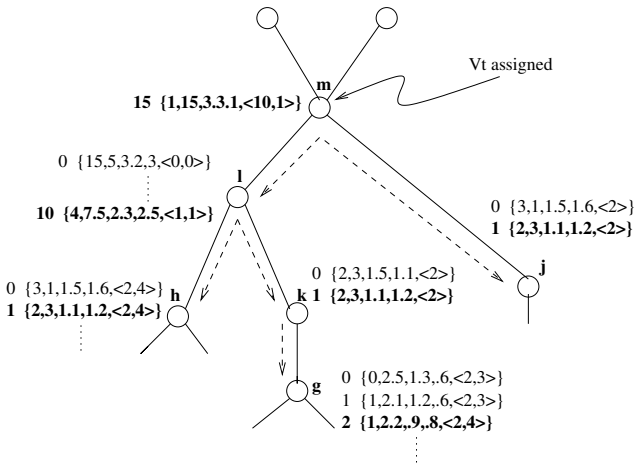

Figure 4.3: Enumeration for V_t assignment

Figure 4.3 shows an example of the enumeration phase, where tuples are shown at the internal nodes of the tree. Consider a sample tuple, shown in bold face, at node j, which corresponds to an inverter, where the number outside the brace represents the ID of the tuple. The first value in the tuple is 2 which means that the V_t-state is 2, corresponding to a V_t assignment of {1,0}, for the transistors of the gate j. The second number represents the total local leakage, i.e., the leakage of node j plus the leakage corresponding to its transitive fanins, the third and fourth values are AF and AR, respectively, and the last field is a vector that holds the ID of the input tuple. Now, when this tuple propagation reaches node m, which is a multifanout node, the best tuple that meets the timing constraints is selected and the corresponding V_t is assigned to m. The algorithm then checks the ID's of input tuples, $< 10, 1 >$, for the left and right child, respectively, and assigns V_t's corresponding to tuple 10 to node l and corresponding to tuple 1 to node j, and so on.

4.5.2 Pruning techniques

Since the algorithm employs explicit enumeration, use of pruning techniques is necessary to make it efficient. Due to the dependence of the gate delay and transition time on the fanout loading, there is no straightforward optimal substructure property that can enable the use of dynamic programming.

Provably suboptimal tuples

A provably suboptimal tuple can be a tuple that satisfies one of the following three properties:

- There exists another tuple whose fall and rise arrival times and leakage current are all less than those in the current tuple.
- The rise time for the tuple, when it is added to the low-V_t delay from current gate to the root of the fanout-free tree, results in a delay that exceeds the required rise time at that multi-fanout gate. A similar condition applies for the fall time.
- The tuple has rise and fall times such that when they are added to the high-V_t delay from the output of the gate under consideration to the root of the tree, the resulting delays both satisfy the required times at the multifanout gate, but with a higher leakage than a previously computed tuple that also satisfies those required times.

To illustrate first condition let us consider two tuples with $\{AR, AF, L\}$ values of $\{1, 1, 1\}$, and $\{1.2, 1.3, 1.1\}$. The latter tuple has both delay and leakage worse than first tuple, and hence is clearly suboptimal. The second criterion can be explained using a chain of three inverters where the last inverter fanouts to more than one gate. Here, we have a tree, in this case just a series connection of three nodes, rooted at the node corresponding to last inverter. There are two inversions between output of first inverter and last inverter, or in other words, a rising output at first inverter, gives rise to a rising output at last inverter. Now, if the first inverter had a tuple $\{1, 1, 1\}$ at its output and the delay corresponding to low-V_t setting between these two points is 3, and the required time at the output of

last inverter is 3.5, then this tuple cannot clearly lead to a solution, and hence is provably suboptimal. Since the assignment of required times at the output of multifanout nodes is heuristic in nature, in the implementation low-V_t delay is multiplied by a factor less than 1, before testing for suboptimality resulting from condition 2. The last criterion can be explained using the tuples $\{1.4, 1.5, 2\}$, and $\{1.3, 1.7, 3.5\}$ at the output of first inverter, in the inverter chain mentioned above. Let us assume that the rise and fall delays between first inverter output and third inverter output are both 1 units, under high V_t setting. Now if the required times at the output of third inverter are $\{5, 5\}$ then both of the tuples clearly can lead to a feasible solution at the output of third inverter. Out of these solutions, the one corresponding to the prior tuple will have lower leakage and hence the latter tuple is provably suboptimal.

Quasi-suboptimal tuples

More aggressive pruning techniques are required to control the number of tuples, especially when the fanout-free regions are large. For this purpose, the function prune_tuple_set removes quasi-suboptimal tuples. A tuple is quasi-suboptimal if there exists another tuple such that multiplication of the arrival times and leakage of that tuple by a small prune factor makes the current tuple provably suboptimal. Intuitively, this is equivalent to saying that the tuple is "nearly suboptimal," but not provably so. For example, consider tuples A and B with $\{AF, AR, L\}$ values of $\{1, 1, 1\}$ and $\{1.01, 1.01, 0.5\}$, respectively. With a pruning factor of 0.99, tuple B is transformed to $\{0.9999, 0.9999, 0.495\}$, which is provably suboptimal to tuple A. To prevent the better of the two tuples from being removed, the tuples can be sorted in order of nondecreasing leakage before pruning, and only the tuple under consideration and all the tuples after it are considered for pruning using this condition. This quasi-suboptimal pruning helps in reducing the size of tuple set, and speeds up optimization, with very minute, if any, loss in optimality. In practice, different pruning factors can be used for arrival times and leakage.

It is important to explain why the use of these pruning techniques leads to the removal of a significant number of tuples. While the number of tuples can theoretically increase exponentially, this does not happen in practice.

The first reason for existence of suboptimal tuples is the nature of enumeration technique. The extensive spread of variables such as input arrival times, input transition times, and V_t-settings can produce very close tuples. For example, consider two scenarios for an inverter:
Case 1: The V_t-setting for the inverter is such that the nmos transistor is set to high V_t, and the input tuple has low arrival times and high leakage.
Case 2: The V_t-setting for the inverter is such that the nmos transistor is set to low V_t, and the input tuple has high arrival times and low leakage.
In the first case, the inverter delay is high but the input arrival times and the inverter leakage are low, and in the second case inverter delay is low, due to low V_t of switching nmos, but the arrival times and input leakage are low. Hence the two tuples produced at the output can lie in close proximity. In case of gates with higher number of inputs several combinations of arrival times and V_t-settings can lead to generation of very close tuples.

Another reason for tuple pruning is the phenomenon of *input dominance*. This refers to the fact that if one of the inputs of a gate has a tuple with rise and fall arrival times of AR_1 and AF_1, respectively, then for any tuple at the other input with rise and fall arrival times $AR_2 \leq AR_1$ and $AF_2 \leq AF_1$, respectively, the output rise and fall time will be decided by the tuple at the first input (assuming equal pin-to-pin delays, although a similar argument can be made if these are unequal). We say that the first input *dominates* the second input for that particular tuple-combination.

The concept can be further explained using a noninverting two-input gate, C, shown in Figure 4.4, where all tuples have the form $\{X, L, AR, AF, IS\}$, as described earlier. The rise and fall times in Tuple 1 at input A of the gate are both higher than those in all of the tuples at input B. Assuming a unit pin-to-pin delay from both input pins, it can be seen that the input tuple combinations will create three tuples at the output with the same arrival times. It is important to note that these arrival times are all dictated by tuple at

the input A. The leakage values in the output tuples are obtained by adding leakage values of the input tuples and to that adding leakage of the gate C, which is assumed to be 1 unit, in the 0 V_t-state. Out of the three tuples at gate C, the highlighted tuple is the only nonsuboptimal tuple, since it has the least leakage among three tuples and only that tuple will be preserved. The other tuples will be pruned, and this is indicated in the figure by striking them out.

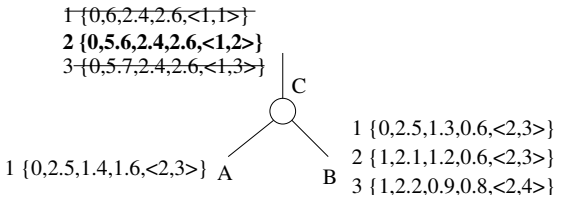

Figure 4.4: Input dominance

In general, if a tuple at one input dominates k tuples at the other input, then all of the resulting k tuples will have the same rise times and the same fall times. Out of these k tuples only the tuple having lowest leakage is nonsuboptimal and all others will be pruned. In the implementation there is in fact no need to generate the suboptimal tuples that fall under this category. A simple condition before generating the tuple can test for provable suboptimality due to input dominance. This obviates generation of output tuple altogether, resulting in reduction in CPU time.

4.6 Sizing and V_t assignment

4.6.1 Theoretical development

The heuristic approach presented in this section separates the optimization problem into two distinct steps, namely, transistor sizing and dual V_t assignment. The important consideration in this two step procedure is to decide whether to carry out sizing prior to V_t assignment, or vice versa, and to analyze whether such a separation can be logically justified. To assist in this analysis, we will now consider the effect of sizing and V_t assignment on both the delay and leakage current. Consider the first order equations for inverter delay [31], given below:

$$t_f = \frac{2C(V_t - 0.1V_{DD})}{\beta_n(V_{DD} - V_t)^2} + \frac{2C}{\beta_n V_{DD}(1-r)}\left[\frac{r-0.1}{1-r} + \frac{1}{2}ln(19-20r)\right] \quad (4.3)$$

where r is V_t/V_{DD}, and the two terms on the right hand side represent two distinct intervals corresponding to the inverter being in saturation region and linear region, respectively, of the total fall duration. All other symbols have their standard meanings. Although these equations may be numerically inaccurate in nanometer technologies, they have a good fidelity with accurate models and are good predictors of trends. Following trends are evident form Equation (4.3): *Observation 1*: Delay is inversely proportional to transistor width, and decreases at a super-quadratic rate with decrease in V_t.

From the transistor leakage current expression in Equation (2.3), one can see that:

Observation 2: The leakage current varies exponentially with V_t, but linearly with the transistor width.

Let us examine a typical delay-area trade-off curve for applying only transistor sizing on a combinational circuit; an example is shown in Figure 4.5.

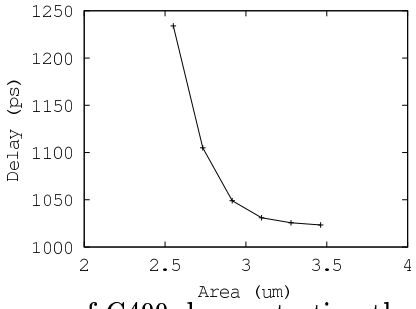

Figure 4.5: Delay-area curve of C499 demonstrating the knee region

The curve shows a distinctive knee region, beyond which the improvement in the delay per unit increase in the area decreases significantly, and this leads to the next observation: *Observation 3*: Up to the knee point, sizing results in a linear increase in the area, with a very small rate of increase. Beyond this region, the area increases exponentially. Although this is not visible from the figure, our experiments show that individual transistor sizes

also increase at a slow rate until the knee of the curve, and rapidly beyond it. Since the leakage current varies linearly with the transistor width (see Observation 2), it shows the same trends as the area.

From these three observations, we can conclude that:

- Up to the knee of the curve, both sizing and V_t optimization cause the delay to reduce, but from Observation 3, their effects on the leakage power are different: while V_t alteration causes an exponential jump, a sizing step causes a much more sedate and linear change.

- Beyond the knee of the curve, both sizing and V_t alteration show an exponential increase in the leakage power, while the latter is more effective at delay reduction, as noted in Observation 1.

Therefore, it makes engineering sense to use a two-stage optimization as follows:

- Perform delay reduction using sizing until the knee of the area-delay curve is reached, so that leakage current increases linearly in this region.

- Beyond this point, since the leakage current increases exponentially regardless of whether the sizing is used or the V_t assignment is used, V_t assignment alone can be used since it is more effective in delay reduction, while maintaining constant area.

Thus even though the problem is divided into two steps, the advantage of this methodology stems from the fact that the required time constraints, used in the second step correspond to optimal sizes from sizing problem.

4.6.2 Algorithm

The two steps of the heuristic are given below.

Step 1: All transistors to high V_t and the transistor sizing problem is formulated as one of delay optimization under area constraints. A sensitivity-based heuristic, similar to TILOS [10] is used for optimization in this step. The optimization is stopped at the knee of the

area-delay curve.

Step 2: In the second step dual V_t assignment is performed using the transistor sizes obtained in the first step. Based on the delay values, which correspond to transistor sizes under high V_t assignments, and on the required times at the output, rise and fall required times are assigned at the output of every multifanout gate. The required times are propagated from outputs to inputs using Critical Path Method [12], with modifications to incorporate separate rise and fall times.

4.6.3 Postprocessing

Although V_t assignment is carried out based on the optimal transistor sizes calculated in Step 1, Step 2 was a discrete optimization step, and there is room for fine tuning the transistor sizes by continuous optimization after this step. This is achieved by employing the fast sensitivity-based sizing engine again to meet delay constraints, this time under the V_t assignments obtained above. Since the result of Step 2 is a feasible solution for this problem, the result is guaranteed to be no worse than that after Step 2.

Typically this step leads to a circuit configuration where the area is less than that after step 1 of the heuristic, and this motivates further processing to recover leakage. For each low V_t transistor we examine if the transistor can be converted to high V_t while maintaining circuit delay and area constraints. An incremental timing analysis engine makes the postprocessing very fast.

4.7 Implementation and results

The algorithms are implemented as a CAD tool called MinSATVA (**Min**nesota **S**izing **And** **T**hreshold **V**oltage **A**ssignment). The experiments are carried out using 0.1μm technology parameters provided by Predictive Technology Model developed at Berkeley [33]. The probability values used in Equation (4.2) are obtained using a probability propagation routine written in PERL. In the implementation, all of the inputs are assigned a probability, $P_1(i) = 0.5$, where $P_1(i)$ represents the probability that the signal i is at logic 1. The probability calculation is carried out only once, since input state probabilities are not dependent

on the delay value; the delay can only change the input state probabilities if glitching is considered, which does not arise in the stand-by mode. The delay analysis routine, the sizing algorithm, the enumeration based heuristic algorithms and postprocessing algorithms are implemented in C. The circuit is first sized up to the knee point of the area-delay curve in Step 1, after which the enumerative V_t assignment technique is applied in Step 2 followed by a final postprocessing step. Before application of Step 2, delays from every gate to root of the fanout free tree to which it belongs are calculated and stored, for both all high-V_t and all low-V_t settings, and are used in provable pruning. The experiments are carried out on a Sun Ultra 10 workstation. ISCAS85 benchmark circuits, mapped to simple gates, are used as inputs to our optimization algorithm.

Circuit	Low V_t	PB		Our Tool	
	L (nA)	L (nA)	%	L (nA)	%
C432	1937	1422	26.6	417	78.5
C499	4990	4266	14.5	1944	61.1
C1908	7295	6116	16.6	1593	78.2
C2670	11173	8704	22.1	2450	78.1
C3540	15520	12874	17.1	3300	78.7
C5315	21749	17625	19.0	4414	79.7
C6288	18099	15646	13.6	5536	69.4
C7552	32019	26247	18.0	6923	78.4

Table 4.1: Dual V_t assignment using PB algorithm [1] and the proposed algorithm

A priority-based backtracking (PB) approach similar to one presented in [1] is implemented for the purpose of demonstrating the efficacy of the V_t assignment algorithm under constant transistor sizes. Both algorithms are run at transistor-level, and the comparison is shown in Table 4.1. The first column lists the circuit name, while the second column lists its leakage (L) when all the transistors are set to low V_t. Columns 4 and 6, provide leakage numbers when the PB algorithm and the proposed V_t assignment algorithm are applied, while maintaining the delay corresponding to all low-V_t setting. It can be seen that the proposed algorithm performs better than the PB algorithm on all the circuits. The benefits in results can be attributed to the fact that our algorithm evaluates all the possible

solutions for a gate for all the solutions at its input gates. Percentage reductions obtained using PB algorithm do not match to those indicated in [1]; this might be due to difference in the delay model. The highly accurate delay model used here also considers transition time effects, and V_t setting of the transistor that opposes the output transition.

For purposes of comparing MinSATVA algorithm with an existing simultaneous sizing and V_t assignment algorithm, a sensitivity-based approach (SBA) similar to [2] is implemented, and the results of comparison are shown in Table 4.2.

Circuit	Unsized delay (ps)	Target area	Target delay (ps)	SBA		MinSATVA	
				Leakage (nA)	CPU (s)	Leakage (nA)	CPU (s)
C432	1775	106	1240	302	64	259	448
			1150	363	77	360	545
			1060	537	98	488	1762
C499	1570	262	1100	631	332	610	460
			1020	698	340	651	465
			940	761	386	797	463
C1908	2240	391	1570	923	519	939	1770
			1460	1025	568	1054	1864
			1340	1365	824	1411	175
C2670	2615	564	1830	1374	407	1351	270
			1700	1482	560	1467	257
			1570	1795	879	1665	283
C3540	3155	788	2210	1965	1167	1906	473
			2050	2336	1726	2149	2000
			1890	2905	5556	2488	898
C5315	2786	1163	1950	2669	1233	2629	757
			1810	2856	1714	2706	717
			1670	3365	4900	2832	776
C6288	7760	1080	5430	5275	5400	3726	2700
			5040	6657	9400	4853	3250
			4650	8903	9436	5960	3000
C7552	2280	1646	1600	4197	10600	3999	5929
			1480	4891	11700	4299	6007
			1370	6632	11125	4802	6202

Table 4.2: Optimization results, comparing the performance of SBA [2] and MinSATVA.

The first column lists the circuit name, and the second column provides the circuit delay when all of the transistors are set to minimum size. The third and fourth columns represent target area and target delay, respectively. The circuit configuration obtained after the sizing step of MinSATVA is used as the input to SBA. The remaining columns list the

leakage current and CPU times for the two methods. It can be seen that MinSATVA delivers performance improvements over SBA, and also results in significant improvement in quality, especially as the constraints are tightened.

These observations are in line with the observation that a general sensitivity-based algorithm fails to perform well on tight constraints on large circuits. Moreover, in this case, where the problem contains discrete variables, a sensitivity-based heuristic is further handicapped. Our algorithm, on the other hand, only uses a sensitivity-based approach to solve a continuous optimization problem, and this has been well documented as providing good solutions until the knee of the sizing curve [36]. The enumeration-based algorithm then exploits the good quality of the starting circuit configuration to solve the discrete problem.

The improvement in processing time compared to SBA stems from the fact that the proposed enumerative algorithm is not altered by the constraints, while in the case of sensitivity-based algorithm, not only it is hard to converge with tightening constraints, but the tightened constraints also themselves require further processing. It was found that the dominant factor in the CPU times of MinSATVA comes from the sizing step, while the V_t assignment step is very fast.

Finally, the postprocessing step, of Section 4.6.3 was seen to improve the result typically by about 3 to 5%.

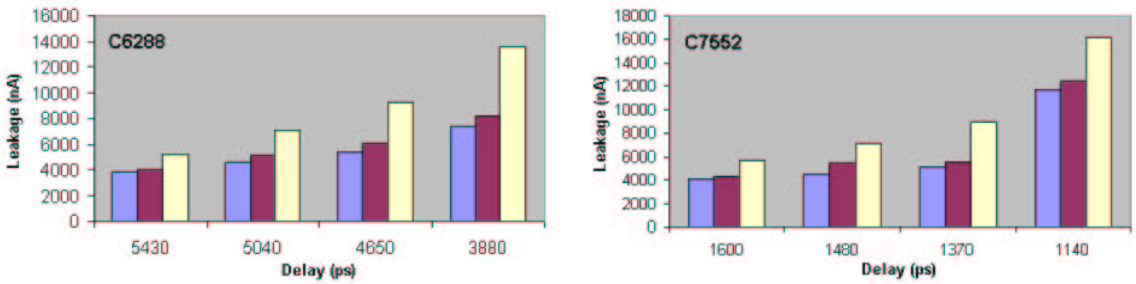

Figure 4.6: Comparison of V_t assignment at transistor-level, stack-level, and gate-level

To test the efficacy of MinSATVA when a set of transistors is constrained to have the same V_t, stack-level and gate-level assignments are performed. Stack-level assignment means all the transistors in a stack are constrained to have the same V_t, while in case of gate-level,

all the transistors within a gate are constrained to have the same V_t. Note that this type of assignment allows more freedom than gate-level assignment, since although V_t assignment is not performed at transistor-level, the sizing is still performed at the transistor-level. The enumeration-based algorithm makes it very easy to incorporate stack-level and gate-level assignments; we only need to consider those V_t-states where the stack-level and the gate-level assignment constraints are satisfied, respectively. The sized circuit obtained after Step 1 is used as input configuration. Figure 4.6 shows optimization results on only two of the circuits at different delay constraints. The bars from left to right correspond to transistor-level, stack-level, and gate-level, respectively. It can be seen that the gate-level assignment results in significant increase in leakage in comparison with transistor-level or stack-level assignment, especially as the constraints are tightened, underscoring the relevance of a approach that can work at these finer levels. The CPU times for assignment at the stack-level and gate-level varied from about half to about the same as that of the transistor-level assignment.

4.8 Conclusion

A fast approach to solve the problem of stand-by power versus delay tradeoff optimization is presented. The problem is formulated as stand-by leakage optimization problem under area and delay constraints. The proposed solution approach involves transistor sizing, which is carried out till the knee point on the delay-area curve is reached, followed by an enumeration algorithm for V_t assignment. Effective pruning is used to enhance the speed of enumeration algorithm. The results show that the algorithm performs much better compared to sensitivity-based algorithm, both in terms of quality of solution and optimization time, on a wide range of combinational circuits.

Chapter 5

Synthesis of Datapath-Dominated Circuits

5.1 Introduction

A majority of high-performance digital circuits tend to be datapath-dominated, and synthesis of these datapath circuits continues to be a very important problem. An important characteristic of datapath circuits is the presence of functional and structural regularity. This presence of structural regularity directly translates down to regularity in layout, and this is beneficial not only because it results in better area but also because it enables better prediction of layout parameters, especially that of layout parasitics during synthesis. Therefore, it is very important that the regularity that datapath circuits already possess be preserved during the synthesis process as much as possible.

Traditionally, dedicated datapath layout generators have been used to realize datapath circuits [37], and a lot of work has been carried out in exploiting regularity for layout purposes. Regular layouts generally benefit from smaller area and smaller complexity of layout synthesis process. In [38], the authors present an iterative methodology, consisting of logic optimization, layout, and back-annotation, to generate regular layouts. The original circuit is first laid out utilizing regularity information. The layout parameters are then used

in logic optimization, and following this functional correspondences between the optimized and the original netlists are obtained to realize a placement that is as far regular as possible. The method does not exploit regularity either to guide or to speed up the optimization process, and does not try to preserve any existing circuit regularity during optimization. It uses regular placement of the unoptimized circuit to back-annotate the layout parameters and hence results in early convergence.

Another approach used for realizing datapaths involves the use of specialized datapath compilers or datapath generators, which employ a fixed library of datapath operators to implement the circuits, and hence strictly maintain the circuit regularity. Due to the regular placement, the circuit realizations obtained by these tools are generally smaller in area as compared to traditional synthesis. However, it has been pointed out [39] that this enforced regularity often does not lead to the best possible performance since it hinders aggressive synthesis. Conventional synthesis, on the other hand, does not respect regularity, and applies transformations in a nonincremental and local fashion. Although such synthesis is good for control circuits, its complete disregard for regularity is detrimental for datapath circuits, and experimental results to support this claim have been well documented. It was shown in [39] that even though conventional synthesis results in faster circuits prior to layout, as compared to datapath generators, the gain in speed comes at the cost of increased layout area.

Regularity has also been exploited for speeding up synthesis of datapath circuits. The idea in [40] is to synthesize only one *slice* of a regular group and then to repeat that slice. This approach also preserves the regularity during the synthesis process. More recently, there has been some work on regularity-driven synthesis [41], where regularity information is used to selectively apply transformations to regions with similar regularity characteristics as captured by regularity signatures.

The approaches mentioned above suffer from the fact that they maintain all of the circuit regularity, and as has been pointed out earlier, this does not necessarily lead to area- and delay-optimized circuit configurations. Examining this critically, we observe that it is not

necessary to choose between the two extremes of complete regularity and no regularity. Instead, starting from a very regular layout we could identify gates that lie on a critical path and destroy the regularity on this path in a *controlled* manner. Consequently, we can trade off complete regularity for timing improvements. Moreover, since many gates in a circuit lie on non-critical paths, a substantial amount of regularity may be preserved. Such an approach would present a way of smoothly trading off the delay for the regularity, and the best solution could be characterized as one that satisfies the delay constraints with the maximum regularity.

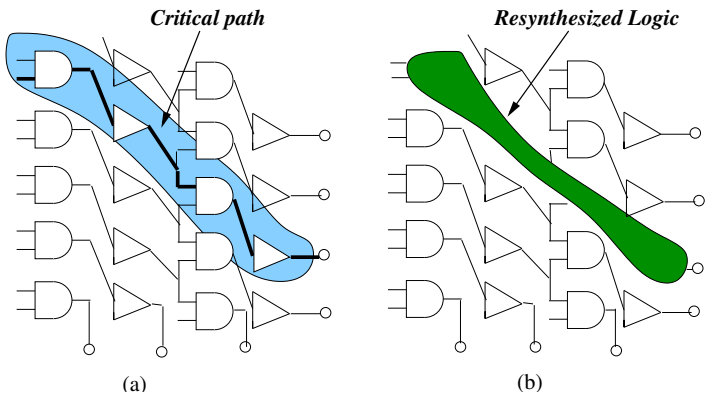

Figure 5.1: Trading off circuit regularity for delay reduction

The idea above can be further illustrated by an example. Figure 5.1(a) shows the original circuit, which clearly has a very high regularity, and the critical path is shown by the darkened edges. Let us assume, for simplicity, that the delays of all other paths in this example are significantly smaller than the critical delay. Preserving the regularity completely would imply living with this large delay. On the other hand, if the circuit is selectively resynthesized so that the critical path is mapped for reduced delay while the remainder is left untouched, we would obtain the circuit in Figure 5.1(b), which can be seen to have a slightly reduced, but still very high, regularity and perhaps a much smaller delay.

The "sacrifice" that is made in this example is to exclude the remainder of the circuit from a full synthesis step and to preserve its regularity instead. However, this is less of a compromise than is apparent, since the delay numbers computed by synthesis could be very far off from the actual delays for a complete synthesis step. If a large degree of regularity

is preserved, a good layout model is instantly available, which enables a more accurate prediction of circuit parasitics. Therefore, even though this circuit may not have as good a delay estimate from synthesis, the estimate will have a much better correlation with the final delay after layout, thereby greatly enhancing the predictability and hence the speed of design closure.

This forms the motivation for this work, and the method proposed in this thesis achieves high performance with very little sacrifice of regularity as compared to traditional logic synthesizer, and involves a two-step approach for synthesis of datapath-dominated circuits. In the first step, regularity is exploited to speed up synthesis by synthesizing only one slice and repeating it, thereby preserving maximum regularity in the synthesized circuit. In the second step, the regularity is iteratively destroyed in a controlled way to obtain timing improvement at minimum cost. Specifically, the regularity is destroyed on critical paths in an incremental fashion. Although logic synthesis transformations are employed in the second step, the approach of selective regularity destruction is general enough to accommodate other kinds of circuit transformations. For example, instead of applying logic-level operations, one can apply gate sizing transformations.

This chapter is organized as follows. Section 5.2 presents the motivation with the help of an example. Section 5.3 explains the details of regularity modeling, and presents various approaches of regularity extraction. Section 5.4 presents our methodology, while Section 5.5 lists our results and is followed by concluding remarks in Section 5.6.

5.2 Motivation

Synthesis typically involves two distinct steps, namely, technology-independent synthesis and technology mapping. Conventional technology-independent synthesis pays no regard to regularity, and in the succeeding technology mapping phase, the circuit is then mapped to available library cells. As stated above, this initial technology-independent synthesis can hinder the optimal synthesis of datapath circuits if it completely disregards the inherent circuit regularity. In case of control logic, where the initial circuit is likely to be irregular,

this kind of synthesis is acceptable, but this is not so for datapaths, where the performance constraints are very stringent.

As an example, we will now consider a four-bit ripple carry adder circuit, implemented in 0.13μ [42] technology. The adder was synthesized in three different ways as follows:

- In the first approach, the circuit was synthesized in SIS [43] by using `script.rugged` followed by the `map` command with the delay minimization option selected. The synthesized circuit had gate area of 363μ , and a delay of $1.69ns$. The value of the regularity index, to be defined in Section 5.3.1, for the synthesized circuit was 0.14 units in the resulting circuit.

- In the second approach, one slice of the circuit was synthesized and repeated to obtain a four-bit adder. This resulted in a configuration with a much reduced gate area of 86μ, a larger delay of 2.6 ns units, and a regularity index of 35 units.

- Finally, the regularity on the longest path was destroyed and the synthesized circuit had a gate area of 176μ, a delay of 1.72 ns, and a regularity index of 3.5 units. Thus the resulting circuit had better delay and regularity as compared to traditional synthesis.

A notable fact is that the circuit configuration obtained by strictly enforcing regularity would not have been able to match the delay obtained by traditional synthesis. The third method results in a higher gate area, but has much higher regularity as compared to the results of traditional synthesis, and lower delay as compared to that of circuit realized with strictly enforced regularity. Note that the area estimate in the first case is much more unreliable than that in the second and third cases. Therefore, the key to obtain both area- and delay- optimized circuit configurations is to use judicious regularity destruction.

To verify that the controlled regularity destruction indeed results in better postlayout parameters, the resultant circuits were laid out using DRAGON [44], a publicly available placer.

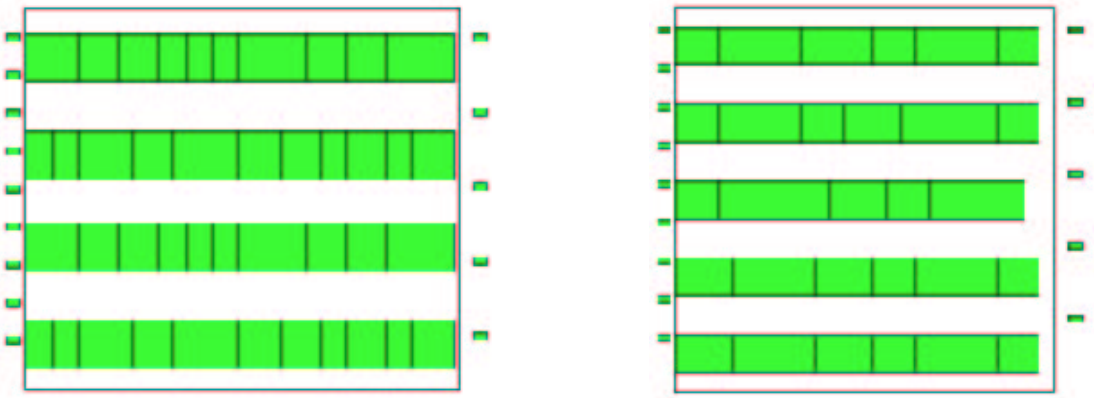

Figure 5.2: Layout of synthesized adders

Figure 5.2 shows the layout of the adder obtained by using judicious regularity destruction on the left and that obtained by conventional synthesis on the right. A quick inspection of the layouts can show that the one on left is very regular, while the one on right shows comparatively much less regularity. The postlayout delays were, for the traditionally synthesized circuit and the circuit obtained by controlled synthesis, were 1.73 ns and 1.75ns, respectively. Thus it can be seen that the controlled destruction of regularity leads to lower percentage increase in postlayout delay as compared to that of conventional synthesis. Therefore the judicious destruction of regularity can lead to better overall circuit configurations, and this forms the motivation of this work.

5.3 Circuit regularity

5.3.1 Regularity modeling

Regularity in a circuit can be of one of the two types: functional regularity or structural regularity. Functional regularity means that identical functional blocks exist within a circuit. The blocks can have different implementations. Structural regularity, on the other hand, indicates that structurally similar blocks exist in the circuit. Thus the presence of structural regularity indicates the presence of functional regularity, as well. Functional regularity does not necessarily result in regular layouts. In addition, identifying the functional regularity is a complex task, hence this thesis concentrates on exploitation of structural regularity.

Structural circuit regularity refers to the repetition of bit slices where each slice contains similar bit stages, and is illustrated in Figure 5.3.

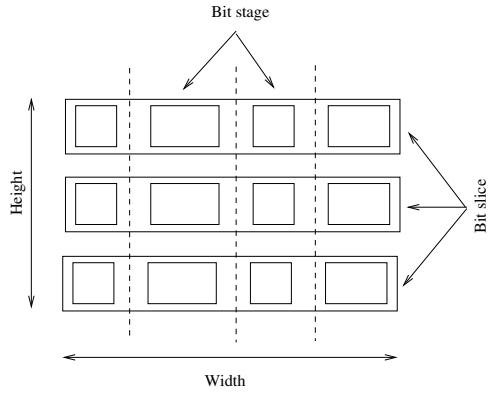

Figure 5.3: Circuit regularity

A *slice* or a *template* is a subcircuit of a given circuit that is repeated two or more times in that circuit. A set of slices with a similar structure is called a *regularity group*. Our purpose is to extract as many non-overlapping regular groups as possible in the circuit and use them to reduce synthesis effort. The amount of regularity can be represented numerically by the *regularity index*, RI, which is calculated as follows [41]:

$$RI = \frac{1}{n_{grp} + n_{nr}} \cdot \left(n_{nr} + \sum_{i=1}^{n} \left(S_{grp_i} \cdot \frac{2\sqrt{S}_{grp_i}}{h_i + w_i} \right) \right) - 1 \tag{5.1}$$

In the above equation, n_{grp} is the number of regular groups, n_{nr} is the number of gates that are not within a group, h_i and w_i are the height and the width of the regular group i, respectively, and S_{grp_i} is the product of h_i and w_i. The height of a group is the number of slices in that group and the width of the group is the number of gates in a slice. The term in the innermost parenthesis favors square structures over rectangular strip-like structures, and this is in line with the fact that the former is more amenable to chip layout, under the assumption of a square aspect ratio for the layout. It can be seen that higher regularity index represents higher regularity in the circuit and regularity index of 0 represents a circuit without any regularity. It should be noted that under this model, the presence of structural regularity necessarily indicates the presence of functional regularity. In other words, each

slice in a regular group represents the same function.

5.3.2 Regularity extraction

A number of approaches for extracting circuit regularity have been published in the past, and three of the recently employed approaches are presented in this section. The explanation uses following terminology. A digital circuit is represented as a directed graph G, composed of a vertex set V, where each gate in the circuit represents a vertex in the graph, and an edge set E, where each connection represents one edge in the graph.

A template-based approach was presented in [45], and can be summarized as follow. It involves generation of two different types of templates: tree templates and single principle output (PO) templates. A single PO template is a multi-output template where every output lies in transitive fanin cone of a single principle output. The problem of regularity extraction is tackled in two steps, namely, template generation and circuit covering by templates. In general, a given graph can lead to an exponential number of templates, and hence two simplifying assumptions are made to reduce the complexity of template generation and number of templates generated. The set of templates is restricted to contain only those subgraphs that are not in any other subgraph, and all of the incoming edges of a node are marked with unique indices. The latter assumption obviates checking for graph isomorphism, though this faster procedure may exclude otherwise regular structures. For covering the circuit graphs, two different methods are proposed. The, first method selects templates greedily according to size, while the second method makes greedy selections based on the number of occurrences. It is argued that the former method leads to better results, while the latter method requires a smaller optimization time due to a large number of repetitions of smaller blocks. The drawback of this approach is its very high time complexity; it is shown that the complexity for generation of tree templates is $O(V^2 logV)$, and for generation of single PO templates, it is $O(V^5)$. Experimental results show that CPU times are of the order of a few minutes to extract regularity from circuits up to 2000 nodes. Therefore, such a approach is clearly not very practical when larger circuits are under

consideration and synthesis is to be carried out after regularity extraction.

A clustering-based approach for regularity extraction was presented in [46]. This approach consists of two steps: the first step performs template generation while the second step involves graph covering. In the first step, the circuit graph is decomposed in a *hierarchical parse tree* using a clan-based decomposition algorithm. A *clan* is a group of nodes in the graph that have a natural affinity towards each other, and is modeled as a group of nodes with common transitive fanins and common transitive fanouts. The parse tree nodes are then classified into equivalence classes; the classes represent templates that are suitable for circuit covering. It is argued that the hierarchical nature of this method enables the fast recognition of templates. However, the complexity of the template generation is $O(V^4)$ and hence this approach has the same scaling problem as the previous approach.

To avoid the complexity of template generation, a very efficient approach based on regularity signature was first published in [47], and later used in [40, 41]. In [47] the signature of a random instance of a design cell is defined using its master cell and its connectivity. In general, the signature can be defined from the cell type and its connectivity.

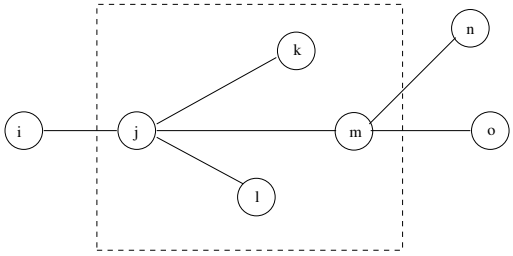

Figure 5.4: Regularity signature

A vertex v is called a *successor* of vertex u and vertex u is called a *predecessor* of vertex of v if there is a directed edge from vertex u to vertex v. A *regularity signature*, RS, is defined as a directed graph consisting of a vertex S and all of the vertices that are successors of the vertex v. Figure 5.4 shows a circuit graph; the subgraph enclosed in dotted rectangle represents a signature.

The extraction procedure used in this work is taken from [40] and is provided here in detail for completeness. A few terms are introduced before enumerating these steps. A gate

in a regular group is assigned a *group number* which is the identifier of the group to which that gate belongs, a *slice number* which is the identity of the slice in the group to which that gate belongs, and a *member number* that represents the identity of the gate within that slice. All of the gates that have the same member number within a group have the same regularity signature. The circuit is represented as a graph where each node in the graph corresponds to a gate and each edge corresponds to an interconnection. The extraction procedure can be now explained as follows:

Step 1 The extraction procedure starts by identifying sets of nodes that have the same regularity signature. Such a set is called a *reference set*, and the initial reference sets are obtained by considering input nodes and multifanout points. If a multifanout point has outputs that all have the same regularity signature, then the outputs form a valid reference set. Each set is taken as a starting point for a regular group. For example, in Figure 5.5 nodes i and i' form the initial reference set.

Step 2 All of the reference sets are then placed in a priority queue based on the number of nodes in the set.

Step 3 The reference sets are then popped from the queue, one at a time, and the regularity signatures of all of its outputs (inputs) are compared. If the regularity signatures are identical, then a forward (backward) expansion of all of the slices in the group is carried out by including those nodes in the regular groups. The newly added nodes with same signature will be assigned the same group number and member number, while their slice numbers will remain different. For example, in Figure 5.5 let us assume that nodes b and b' are in reference set. It can be seen that the node b along with its output nodes c, d, and e, form the regularity signature which is identical to that of node b'. Hence, the forward expansion is carried out by including nodes c, d, and e, in Slice 1, and nodes c', d', and e', in Slice 2.

Step 4 The newly added nodes constitute a reference set and are added to the reference

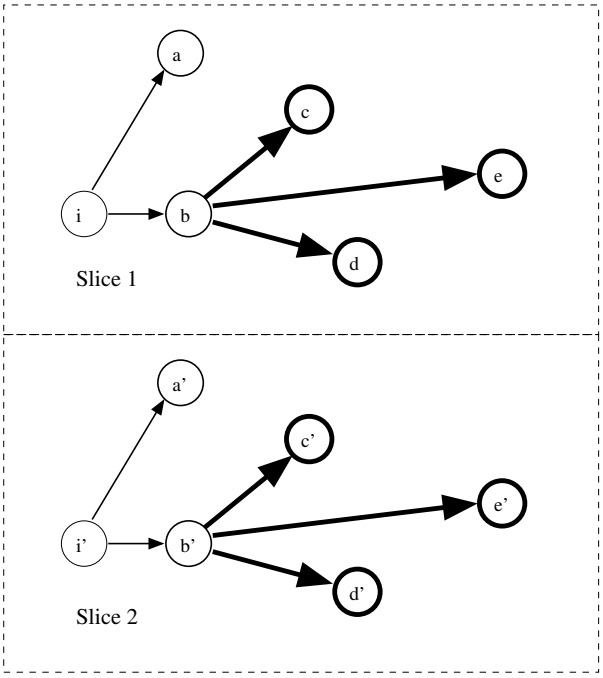

Figure 5.5: Regularity extraction

set queue.

Step 5 The above three steps are repeated until all of the regularity in the circuit is extracted.

5.4 Synthesis approach

The synthesis approach is divided into two stages. In the first stage, regularity is extracted and a slice from each group is synthesized and repeated. In the second step the regularity is destroyed in a controlled manner if it results in the reduction in the circuit delay.

5.4.1 Slice synthesis

In the first step, regularity is extracted from the original circuit using the extraction approach described in Section 5.3.2. Each extracted group is a set of identical slices, and the synthesis can be carried out by synthesizing one slice from each group and then replacing all of the instances of the original slice with instances of the synthesized slice. The gates that do not belong to any regular group are then synthesized as a separate subcircuit, and the

whole circuit is then constructed by stitching the regular and irregular parts together. The synthesis here involves technology independent synthesis followed by technology mapping because we will need accurate timing information in the next step.

5.4.2 Controlled regularity destruction

The first synthesis stage rigidly maintains all of the regularity present in the circuit. In this stage we will destroy the regularity in a controlled manner to meet timing constraints using the following steps:

`Step 1:` Timing analysis is performed to identify the most timing critical path.

`Step 2:` The circuit is then synthesized along the critical path in one of the following ways.

- If all of the gates on the critical path lie within a single slice of a regular group then the delay can be optimized by resynthesis of that slice. In such a case it is often seen that parallel critical paths exist, each path typically belonging to a separate slice of the same group. A delay-oriented technology mapping step is carried out for the slice using cells from a given library, and the resynthesized slice then replaces all instances of the original slice. If the delay after this mapping is the same as the delay from the previous step, then no further improvement is possible and the synthesis procedure stops; otherwise it returns to Step 1. In this case, the delay could be reduced by adding cells of higher strengths into the library, but this is beyond the scope of our problem statement.

- If the critical path consists only of gates outside the regular region, then the subcircuit along the path is resynthesized to minimize delay.

- If the path crosses multiple slices, all of which belong to one regular group, then it is checked whether the network along the critical path within a slice is already resynthesized for delay. If it is not, then the part of critical path in one slice is synthesized for delay and is subsequently replaced in all the remaining slices of that regular group.

If the network along the path is already synthesized then nodes on the critical path from adjacent slices are synthesized together. A new regular group is formed, which has half the number of slices as the original regularity group.

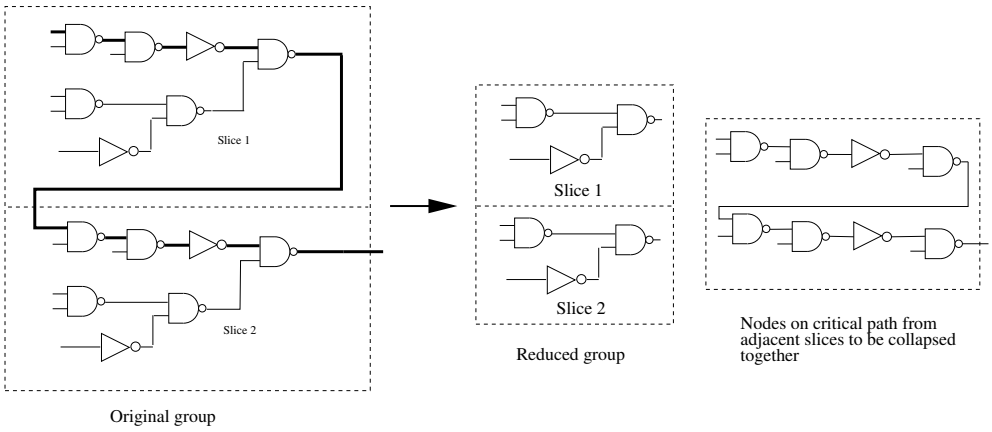

Figure 5.6: Collapsing nodes from adjacent slices

The latter case can be explained using Figure 5.6. The original regular group in Figure 5.6 has two slices, and the critical path through them is shown highlighted. We will assume that the slice has already been synthesized for delay. Now, to collapse nodes on the adjacent slices that lie on critical path, we separate out those nodes. That still leaves three nodes in each slice, and the original regular group is maintained. Our example shows only two slices, but let us consider an alternative scenario. If the original group had, say, four of these slices instead of two, with the same critical path going through the third and fourth slice, then the resynthesized circuit would have a critical path going through slices 1 and 2, and separately resynthesize the same critical path through slices 3 and 4. This would lead to the identification of new regularity, as the optimized critical path through Slices 1 and 2 would look identical to those through Slices 3 and 4. These would now form a new regular group with two slices.

The original regular group is removed only if all of the nodes in the group are used in destruction procedure. Otherwise it is stored and now contains nodes that are off the critical path.

- If some of the nodes on the critical path lie outside the regular groups, while some within regular group, then first nonregular part is mapped for delay. If that part is already resynthesized for delay, then nodes on the critical path that belong to regular group are removed from regular group. The network along the path is then resynthesized for delay. During the removal of the nodes from a particular slice of a regular group, the corresponding nodes from all the slices of that group are removed.

- If the critical path traverses multiple regular groups, then a check is made to see whether the path belongs to only one slice of each of those regular groups. If there is at least one regular group that has only one slice on the critical path, then that slice is mapped for timing and replicated throughout the group.

`Step 3:` Steps 1 and 2 are carried out iteratively till the delay cannot be further improved without significant reduction in regularity.

The details of the algorithm are now explained with the help of another example. Consider two adjacent bits of a fictitious datapath shown in Figure 5.7. Assuming a unit delay model for the sake of simplicity, it can be easily verified that the most timing-critical path consists of the highlighted edges.

All of the nodes in the first slice have a corresponding node in the second slice, and the circuit is totally regular. The critical path contains five nodes (i, j, l, p, and q) that are identical (have same regularity signature) in both slices. The second slice has one additional node, node o', on the critical path, and including it in the list for resynthesis would be beneficial. The algorithm, therefore, checks if there is a node corresponding to o' in Slice 1, and in this case node corresponds to node o. Thus the algorithm does not work only on the critical path, but also takes into account off-critical path nodes, which may correspond to some nodes on critical path in another slice.

However, in many cases, considering only the nodes on critical path may not yield much improvement. This is because the freedom for resynthesis is very limited since we must

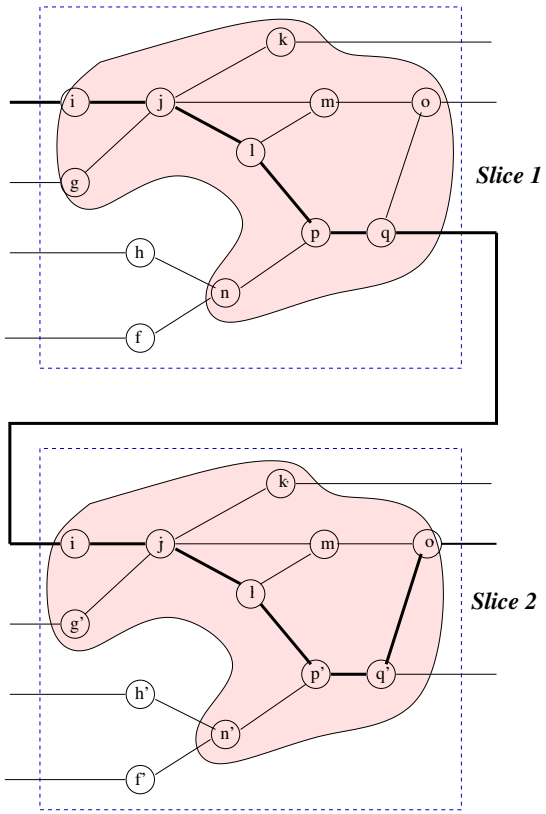

Figure 5.7: Node selection for resynthesis

preserve those nodes that fan out to nodes off the critical paths. Fox example in Figure 5.7, three nodes in Slice 1 (q, l, and j), are on the critical path, but also fanout to nodes off the critical path. Therefore, when only the critical path is synthesized, we must identify these three nodes as the outputs, as also the three corresponding nodes in Slice 2. Thus we can map those output nodes in a different way, but we cannot eliminate them using any aggressive transformations, and this is somewhat restrictive. Therefore, improved results are possible if we consider a subcircuit of depth k from the critical path, where k can be any positive integer. In practice, k can take values between 1 to 5, depending on the size of the circuit, and this provides ample freedom for resynthesis transformations. For example, if we use $k = 1$, then for the circuit shown in Figure 5.7, the algorithm will select the nodes shown by the shaded regions.

5.5 Implementation and results

The algorithms are implemented as a CAD tool called MinRDS (Minnesota Regularity Driven Synthesis). The regularity extraction procedure and the synthesis driver are implemented as C++ programs. The regularity signature is implemented as a properly-chosen hash function. In the implementation, a slice in a group is constrained to have atleast four gates. This is for two reasons: firstly, it is not useful, from the point of view of synthesis, to have very small slices, and secondly, if the limit is any smaller than this, then too many regular groups are extracted, and this may not be particularly useful in aiding layout.

Circuit	SIS				MinRDS			
	Area	Delay		Reg	Area	Delay		Reg
		synthesis	layout	Gates		synthesis	layout	Gates
	(μ)	(ns)	(ns)	(%)	(μ)	(ns)	(ns)	(%)
add16	915	4.76	4.65	88	705	5.02	4.98	100
logic5	1064	3.35	3.24	5	1219	3.48	3.49	33
div4	1027	7.47	7.52	24	1106	7.25	7.25	66
csa16x4	2726	5.14	5.47	16	2988	5.38	5.45	77
bmul8	2764	10.65	10.41	26	3823	10.26	10.20	39
cla64	4562	13.06	13.95	22	5211	8.49	8.66	100

Table 5.1: Comparison of synthesis results.

SIS [43] is used for performing synthesis, and is invoked from the synthesis driver in Step 2 of the methodology. The program writes the equations describing the slice to be synthesized in each step. SIS reads these equations and outputs another equation file which is read by the program. For original synthesis, `script.rugged` is used and is followed by the `map` command. The `map` command with timing optimization option is used during the resynthesis step. A small subset of a 0.13 μ library is used for conventional synthesis, as well as for regularity driven synthesis. First the delay values are obtained by synthesizing circuits using SIS, and then it is assumed that this represents the best speed that is possible using synthesis with pre-layout estimates. Since preserving the regularity involves an additional constraint on synthesis, it may not be possible to improve upon this speed, and hence our algorithm is run with a delay constraint set to be a little higher than the delay obtained by SIS. Because the library used in experimentation is not sufficiently fined-grained, the delay

obtained by MinRDS is seen to be higher than SIS delay for some circuits, while it beats SIS delay on others.

Table 5.1 shows a comparison of the area, delay, and regular gates as a percentage of total gates, obtained by our approach and by traditional synthesis. The synthesized circuits are laid out using DRAGON, the delay values after the layout step are provided, along with the prelayout delay values.

The important point is conveyed by prelayout delay and percentage of regular gates. It can be seen that MinRDS achieves delay in the vicinity of the SIS delay, while maintaining higher regularity in the synthesized circuit as compared to SIS synthesized circuit.

5.6 Conclusion

The chapter proposed an approach for synthesizing datapath-dominated circuits. The methodology employs controlled destruction of regularity to obtain circuits with higher regularity indices as compared to traditional synthesis. The experimental results show that the circuits synthesized in this way have resulted in delay close to that obtained by traditional synthesis.

Chapter 6

Conclusion and future research directions

This thesis has addressed three important problems in VLSI circuit synthesis. The first part of the thesis has proposed a convex and accurate model for gate delay that is better suited than the Elmore model for the current deep submicron technologies. The model was shown to be highly accurate and the computational efficacy of the model has been shown by incorporating it into various different optimization engines. In particular, the models have been incorporated in the solution of area-delay and power-delay trade-off optimizations.

The second part of the thesis has presented a methodology for leakage power minimization. A unified approach involving transistor sizing and V_t assignment has been proposed. A novel enumerative approach has been employed for the V_t assignment phase. The experimental results have shown that the proposed methodology outperforms an existing sensitivity-based methodology, both in quality of solution and in optimization time. The enumerative approach involves decomposing a circuit graph into fanout-free trees as in the case of technology mapping. This partitioning can, depending on the nature of the circuit, have adverse effect on the quality of solution. This prompts further explorations in the way in which the fanout free trees are obtained.

The third part of the thesis has addressed the issue of optimization considering multiple

levels by employing a synthesis approach directed by the structural properties of the circuit. In particular, the research has been targeted towards synthesis of datapath-dominated circuits, and is driven by the regularity presented in the unoptimized circuit. The approach involves the controlled destruction of regularity to obtain highly regular yet timing- and area-optimized circuit configurations. The research has considered synthesis of combinational circuits only. In future, it is necessary to evaluate such an approach on sequential circuits, perhaps by using regularity-driven synthesis along with retiming to obtain better realizations of sequential circuits. Another direction that may be explored is to use regularity-driven synthesis with lower levels of optimization such as transistor sizing or gate sizing. Sizing a gate can potentially avoid local destruction of regularity, and the cost of destroying regularity should be weighed against the improvements in performance.

Finally, the research in this thesis has been variously published [32, 48–50].

Bibliography

[1] L. Wei, Z. Chen, and K. Roy, "Mixed-Vth (MVT) CMOS circuit design methodology for low power applications," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 430–435, 1999.

[2] S. Sirichotiyakul, T. Edwards, C. Oh, J. Zuo, A. Dharchoudhary, R. Panda, and D. Blaauw, "Stand-by power minimization through simultaneous threshold voltage selection and ciruit sizing," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 436–441, 1999.

[3] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, pp. 114–117, apr 1965.

[4] S. Devadas, A. Ghosh, and K. Keutzer, *Logic synthesis*. McGraw-Hill, 1994.

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

[6] Meta-Software, Inc., *HSPICE User's Manual: Volume II: Elements and Device Models*, 1996.

[7] V. De and S. Borkar, "Technology and design challenges for low power and high performance," in *International Symposium on Low Power Electronics and Design*, pp. 163–168, 1999.

[8] B. Sheu, D. L. Scharfetter, P. K. Ko, and M. C. Jeng, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 558–566, Aug 1987.

[9] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-19, pp. 468–473, Aug. 1984.

[10] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[11] 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, Nov. 1993.

[12] S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-Driven Layout Synthesis.* Norwell, MA: Kluwer Academic Publishers, 1993.

[13] R. J. Duffin, E. L. Peterson, and C. Zener, *Geometric Programming - Theory and Application.* New York, NY: John Wiley and Sons, 1967.

[14] D. G. Luenberger, *Linear and Nonlinear Programming.* Reading, Massachusetts: Addison-Wesley, 2nd ed., 1984.

[15] S. Dutta, S. S. M. Shetti, and S. L. Lusky, "A comprehensive delay model for CMOS inverters," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 864–871, August 1995.

[16] A. Nabavi-Lishi and N. C. Rumin, "Inverter models for CMOS gates for supply current and delay evaluation," *IEEE Transactions on CAD*, vol. 13, pp. 1271–1279, October 1994.

[17] A. Chatzigeorgiou, S. Nikolaidis, and I. Tsoukalas, "A modeling technique for CMOS gates," *IEEE Transactions on CAD*, vol. 18, pp. 557–575, May 1999.

[18] V. B. Rao, T. N. Trick, and I. N. Hajj, "A table-driven delay-operator approach to timing simulation of MOS VLSI circuits," in *Proceedings of the 1983 International Conference on Computer Design*, pp. 445–448, 1983.

[19] J. Ecker, "Geometric programming: methods, computations and applications," *SIAM Review*, vol. 22, pp. 338–362, jul 1980.

[20] Department of Operations Research, Stanford University, *MINOS 5.4 USER'S GUIDE*, 1995.

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

[22] A. Conn, I. M. Elfadel, J. W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, "Gradient-based optimization of custom circuits using a static-timing formulation," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 452–459, 1999.

[23] M. R. C. M. Berkelaar, 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 piece-wise linear simulator," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 474–480, 1994.

[24] C. H. Tan and J. Allen, "Minimization of power in VLSI circuits using transistor sizing, input ordering, and statistical power estimation," in *Proceedings of the 1994 International Workshop on Low Power Design*, pp. 75–80, 1994.

[25] S. C. Prasad and K. Roy, "Circuit optimization for minimization of power consumption under delay constraint," in *Proceedings of the 1994 International Workshop on Low Power Design*, pp. 15–20, 1994.

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

[27] Q. Wang and S. B. K. Vrudhula, "Static power optimization of deep submicron CMOS circuits for dual V_t technology," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 490–496, 1998.

[28] V. Sundararajan and K. Parhi, "Low power synthesis of dual threshold voltage CMOS VLSI circuits," in *International Symposium on Low Power Electronics and Design*, pp. 139–144, 1999.

[29] P. Pant, V. De, and A. Chatterjee, "Simulataneous power supply, threshold voltage, and transistor size optimization for low-power operation of CMOS circuits," *IEEE Transactions on VLSI Systems*, vol. 6, pp. 538–545, Dec 1998.

[30] University of Dundee, *User Manual for MINLP_BB*, 1999.

[31] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design.* Reading, MA: Addison-Wesley, 2nd ed., 1993.

[32] M. Ketkar, K. Kasamsetty, and S. S. Sapatnekar, "Convex delay models for transistor sizing," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 655–660, 2000.

[33] "Predictive technology model." http://www-device.eecs.berkeley.edu/~ptm/.

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

[35] K. Keutzer, "DAGON: technology mapping and local optimization," in *Proceedings of the Design Automation Conference*, pp. 341–347, 1987.

[36] 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 International Conference on Computer-Aided Design*, pp. 482–485, 1991.

[37] G. Suzuki, T. Yayamoto, K. Yuyama, and K. Hirasawa, "MOSAIC: A tile-based datapath layout generator," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 166–170, 1992.

[38] R. Nijssen and C. van Eijk, "Regular layout generation of logically optimized datapaths," in *ACM International Symposium on Physical Design*, pp. 42–47, 1997.

[39] P. Ienne and A. Grießing, "Practical experiences with standard-cell based datapath design tools," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 396–401, 1998.

[40] T. Kutzschebauch, "Logic optimization using regularity extraction," in *Proceedings of the International Workshop of Logic Synthesis*, pp. 264–270, 1999.

[41] T. Kutzschebauch and L. Stok, "Regularity driven logic synthesis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 439–446, 2000.

[42] "High density standard cell library." http://www.virtual-silicon.com/.

[43] E. M. Sentovich, K. 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," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, University of California at Berkeley, May 1992.

[44] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 260–263, 2000.

[45] A. Chowdhary, S. Kale, P. Saripella, N. Sehgal, and R. Gupta, "A general approach for regularity extraction in datapath circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 332–339, 1998.

[46] S. Hassoun and C. McCreary, "Regularity extraction via clan-based structural circuit decomposition," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 414–418, 1999.

[47] S. R. Arikati and R. Varadarajan, "A signature based approach to regularity extraction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 542–545, 1997.

[48] K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar, "A new class of convex functions for delay modeling and their application to the transistor sizing problem," *IEEE Transactions on Computer-Aided Design*, vol. 19, pp. 779–788, july 2000.

[49] M. Ketkar, S. S. Sapatnekar, and P. Patra, "Convexity-based optimization for power-delay trade-off using transistor sizing," in *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 52–57, 2000.

[50] M. Ketkar and S. S. Sapatnekar, "Standby power optimization via transistor sizing and threshold voltage assignment," Accepted for publication in *ACM/IEEE International Conference on Computer Aided Design*, 2002.