University of California

Los Angeles

Implications of Modern Semiconductor Technologies on Gate Sizing

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

John Hyung Lee

2012

Abstract of the Dissertation

Implications of Modern Semiconductor Technologies on Gate Sizing

by

John Hyung Lee

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2012

Professor Puneet Gupta, Chair

Gate sizing is one of the most flexible and powerful methods available for the timing and power optimization of digital circuits. As such, it has been a very well-studied topic over the past few decades. However, developments in modern semiconductor technologies have changed the context in which gate sizing is performed. The focus has shifted from custom design methods to standard cell based designs, which has been an enabler in the design of modern, large-scale designs. We start by providing benchmarking efforts to show where the state-of-the-art is in standard cell based gate sizing. Next, we develop a framework to assess the impact of the limited precision and range available in the standard cell library on the power-delay tradeoffs.

In addition, shrinking dimensions and decreased manufacturing process control has led to variations in the performance and power of the resulting designs. We provide methods for gate sizing with statistical delay, and compute bounds to show that full statistical power optimization is not essential. Lastly, to address the complexities of doing in design in a yet immature process, we provide a method to perform incremental discrete gate sizing to account for both anticipated and unanticipated changes in the manufacturing process parameters.

The dissertation of John Hyung Lee is approved.

University of California, Los Angeles

2012

To my teachers.

Table of Contents

1 Introduction **1**

1.1 The physical design process 6

1.2 Outline 9

2 Gate Sizing and Threshold Voltage Assignment Overview **11**

2.1 Static Timing Analysis 12

2.1.1 Concepts and terminology 12

2.1.2 Computing arrival times and slacks 16

2.1.3 Interconnect delay and slew propagation 18

2.1.4 Additional topics 20

2.2 Continuous sizing methods for gate sizing and v_t assignment: 22

2.3 Discrete gate sizing and v_t assignment methods 25

2.3.1 Preliminaries 25

2.3.2 Score and Rank algorithms 30

2.3.3 Greedy methods 33

2.3.4 Global sizing 35

2.3.5 Slack and Delay budgeting methods 36

2.3.6 Methods based on Continuous Sizing 39

2.3.7 Dynamic programming based algorithms 41

2.3.8 Lagrangian relaxation 46

2.3.9 Slew targeting methods 49

2.3.10 Linear programming based assignment methods 52

2.3.11 Summary of methods 55

3 Benchmarking Discrete Gate Sizing and Threshold Voltage Assignment Methods 57

3.1 The standard cell library 60

3.1.1 Delay 61

3.1.2 Delay models 63

3.2 Power tradeoffs 69

3.2.1 Power models 73

3.2.2 Benchmarks and synthesis options 75

3.3 Post-layout considerations 76

3.3.1 Post-layout timing considerations 78

3.3.2 Incremental placement and routing considerations 80

3.4 Comparisons 80

3.4.1 Comparisons using UCLA Timer 82

3.4.2 Comparisons using eyecharts 83

3.4.3 Comparisons using commercial tools 86

4 Parametric Variation and Gate Sizing 91

4.1 Motivating examples 92

4.1.1 The slack wall 92

4.1.2 Worst-case corners and over-design 94

4.2 Statistical power optimization 96

4.2.1 Preliminaries 98

4.2.2 Circuit examples 107

4.2.3 Analysis via rank correlations 107

4.2.4 Suboptimality bounds 109

4.2.5 Experiments 115

4.2.6 Bridging the suboptimality gap 128

4.2.7 Summary 132

4.3 Statistical delay optimization 132

4.3.1 Statistical delay models 134

4.3.2 The Mean Excess Delay 136

4.3.3 Minimization algorithm 140

4.3.4 Results 141

4.3.5 Summary 144

5 Manufacturing Processes, Gate Sizing and Threshold Assignment . . . 146

5.1 Impact of precision and range in standard cell libraries 146

5.1.1 Suboptimality of v_t Assignment 149

5.1.2 Suboptimality in gate sizing 157

5.1.3 Post-layout experiments 163

5.1.4 Dynamic range and precision selection 166

5.1.5 Multi-gate library considerations 169

5.1.6 Summary 171

5.2 ECO Gate sizing for Manufacturing Process Changes 171

5.2.1 ECO Cost 173

5.2.2 Solving the redesign problem 177

5.2.3 Creating initial designs 184

5.2.4 Experimental results 186

5.2.5 Summary 187

6 Conclusions 188

References 191

List of Figures

1.1 Electronic Design Automation flow. 8

2.1 Plot of the transition and delay times for an size 1 inverter driving a .05pF load. 14

2.2 Output transition (slew) dilemma. The input waveforms to a two input gate are shown. 19

2.3 Mesh example from [GKK10]. The optimum sizes for minimum delay are shown. 21

2.4 Forward topological order dynamic programming example. A simple delay vs. power model is used for the gates, where either Delay=1 and Power=2, or Delay=2 and Power=1. 42

2.5 reverse topological order dynamic programming example. A simple delay vs. power model is used for the gates, where either Delay=1 and Power=2, or Delay=2 and Power=1. The required arrival time at the primary output is assumed to be 4. 43

2.6 Design with reconvergent fanout. This structure is problematic in Dynamic Programming formulations. 45

3.1 Normalized inverter gate delays for a commercial 45nm library. 62

3.2 Normalized inverter *input* pin capacitances in a commercial 45nm library. . 63

3.3 Example of effective capacitance: an inverter driving an RC tree. The effective capacitance is less than the total capacitance of the tree: $C_{\text{eff}} < \sum_{i=1}^{4} C_i$. Adapted from [Sap04]. 67

3.4 Normalized inverter gate *internal* powers a commercial 45nm library. . . . 70

3.5 Sources of leakage power. 70

3.6 Normalized inverter gate *leakage* powers in a commercial 45nm library. . . 72

3.7 Plot of (a) the gate-length bias vs. power and (b) gate-length bias vs. delay. The data is for a nominal sized inverter in a commercial 45nm process, with a nominal gate length of 40nm. 72

3.8 Mismatch between the timing estimates at place and route and at sign-off. The gray line indicates when the two estimates match. Data adapted from, and courtesy of [JK10]. 79

3.9 Histogram of the change in the timing after incremental placement and routing. There are 64 different examples. The majority of the cases have a zero change in timing, and most of the changes increase the timing (the change is positive). The maximum decrease and increase are -.007ns and .057ns, respectively. 82

3.10 The effects of disabling slew in sizing experiments; the performance improvement is relative to the timing-feasible greedy method. Disabling slew increases the difference between sizing methods. 83

3.11 The basic structures used in the eyechart benchmarks. 84

3.12 Eyechart comparisons of different gate sizing and threshold voltage assignment methods. (a) represents a gate-sizing comparison, (b) represents a v_t assignment comparison with 3 threshold voltages, (c) represents a gate length assignment with 3 gate lengths. 85

3.13 The effect of the number of gate sizes available in the library on the suboptimality of the gate sizing method. 86

3.14 Comparison between sizing methods using the Nangate 45nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution. 87

3.15 Comparison between sizing methods using a Commercial 65nm Library=. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution. 89

3.16 Comparison between v_{t} assignment methods using a Commercial 65nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution. 90

4.1 Projected variability in key design parameters (adapted from [ITR]). . . . 92

4.2 Effects of a "slack wall" on statistical slack. The PDF for the worst-case slack is shown for 1, 10, 50 and 100 paths. The delay of each path is Gaussian with variance 100ps. 93

4.3 Power sensitivities for the different gates in the Nangate Library. The power vs. size sensitivities of the deterministic power measure ($\partial p_d / \partial z$), the mean power measure ($\partial p_m / \partial z$) and the quantile measure ($\partial p_q / \partial z$) for $\sigma_L = 2$nm (dtd)/0nm (wid) and $\sigma_{V_{\mathrm{t}}} = 4.7\%$ (dtd)/0% (wid) are shown. The sensitivities are sorted by the p_d. Notice that a sorting by p_m would be very similar, while a sort by p_q would be significantly different. 102

4.4 The percentage increase in the quantile power vs. size sensitivity ($\partial p_q/\partial z$) as $\sigma_{\mathrm{L}} = 1$nm increases to 2nm is shown. The increase in the sensitivity is not seen equally for all the gates. 103

4.5 Visual representation of the power vector for deterministic power and statistical power. Each axis represents the power for a different gate type. The vectors have different magnitude and direction, but the angle between them Θ is very small. 104

4.6 Plot of the deterministic power (x-axis) vs. statistical power (y-axis) for the c1355 v2 circuit for width sizing only. (a) Shows the plots for the p_d vs. the p_m measure, whose relation is nearly linear. In contrast, the p_d vs. $p_{m3\sigma}$ measure in (b) has more noise. In (c), a region of the p_d vs. $p_{m3\sigma}$ plot is chosen around $p_d = 4.2 \cdot 10^{-6}$. The difference in $p_{m3\sigma}$ between the upper and lower bounding lines (approx $2.6 \cdot 10^{-7}$ or about 3.5%) is an estimate of the suboptimality bound. Compare this with the bounds in Table 4.8. . 110

4.7 Suboptimality examples for the two gates in (a). The examples in (b) and (c) have the same optimal solutions for both the statistical and deterministic measures, which means that the actual suboptimality is zero. However, the two examples have different suboptimality *bounds*. 116

4.8 The worst-case suboptimality for the example c432 v4 as a function of the assumed deterministic power value p'_d. 119

4.9 The change in the suboptimality as a function of the maximum gate size in c1355 v4. Changing the maximum gate size from 2 to 128 increases the suboptimality by 8.6 percentage points. 121

4.10 The change in the worst-case suboptimality (δ_{wc}) as a function of the maximum gate size in c1355 v4 for the case in Section 4.2.5.2 with the same range of $p_d^\star$ as in Table 4.9. The worst-case suboptimality δ_{wc} reaches its limit of 20.58% for large maximum gate sizes. 122

4.11 The difference between the sizes that are synthesized for deterministic power (w_d), and the sizes that are used to compute the lower bound (w_{lb}). The c1355 v2 circuit is shown. 126

4.12 Suboptimality plot for a 10 gate circuit (computed by enumeration). The x-axis is the delay constraint, while the y axis is the statistical power. The 'x' denote statistical minima and the 'o' denote deterministic optima. When the black and gray lines diverge, there is a suboptimality gap, which is the difference between the two lines. The maximum gap in this plot is 10.1%. Note that for the majority of the delay constraints, the deterministic and statistical optima are the same. 129

4.13 Histogram of the suboptimalities for 500 randomly generated circuits. The majority of the examples have suboptimality of 0%. 130

4.14 The pdf of the delay $T(x, v)$ for a Gaussian distribution is plotted above. $q_{0.95}(x)$ is the 95% quantile, and $m_{0.95}(x)$ is the 95% mean excess delay. The function $g_{0.95}(t)$ is plotted to illustrate the properties in Section IIIA. The minimizer of $g_{0.95}(t)$ is $q_{0.95}$, and the minimum value, which is reflected across the diagonal line, is $m_{0.95}$. 139

4.15 The delay pdfs of benchmark c1355 are plotted for each of the three sizing methods. The MED sizing gives the best pdf, followed by the padded delay sizing and the nominal sizing, which are nearly identical. 142

4.16 The delay pdfs of benchmark c2670 with size dependent variations are plotted for each of the three sizing methods. The MED gives the best delay distribution, followed by the padded delay sizings. These methods give very good results with significant gains over the nominal. 143

5.1 An inverter chain example. 149

5.2 Suboptimality for a single gate with two threshold voltages. 151

5.3 Histogram of the errors in predicting the discrete optimum, from a given optimal continuous solution. The errors are given as a percentage of the discrete optimum. 154

5.4 Error prediction on the c7552 benchmark, in the v_{t}-L1 case. The so_{c} and the $\mathrm{so}_{\mathcal{Q}}$ predictions are made using the data from the dense library result. . 157

5.5 Suboptimality estimates with library v_{t}-L1 for the benchmarks c6288, c7552, s35932, and s38584. The $\mathrm{so}_{\mathcal{Q}}$ method is on the left, and the so_{c} is on the right. The benchmark s38417 omitted as the errors for this benchmark were small. 158

5.6 Arrival time at the output of gate 2 (from Figure 5.1), as a function of the gate sizes s_1, s_2, s_3. The 'o' marks the minimum delay value on each curve. 159

5.7 Power vs delay curve for gate 2 in Figure 5.1 where the size of gate 1 is 1 ($s_1 = 1$) and the size of gate 3 is equal to 2 ($s_3 = 2$). Delay values < 3 are unachievable with the sizes $\{1, 2, 3, 4\}$. 162

5.8 Power delay curve for gate 2 in Figure 5.2 where $s_1 = 2$ and $s_3 = 4$. Using these gate sizes, the gray piecewise linear tradeoff curve can be approximated. 162

5.9 Estimation results for c7552. The estimate is predicted using from the dense sizing using the suboptimality results in this section, and the actual power refers to the actual power achieved by the TILOS method. 165

5.10 Effect of adding an additional minimum size of .5x on the s35932 benchmark in a 32nm library. The overall power decreases significantly, without affecting the delay range. 166

5.11 Effects of library cell granularity on the achievable delay range for v_{t} assignment (a) and gate sizing (b). The optimization results for the s35932 benchmark in a 32nm library are shown. 168

5.12 Achievable power delay range using two different types of devices. By mixing the device types, a better power vs. delay tradeoff can be achieved than by either of the device types alone. 170

5.13 Comparison of the 2008 and 2010 process specifications for a commercial 45nm process. The graph plots the percentage increase or decrease for several key parameters. 172

5.14 Changed area caused by an ECO; $c_{\text{area}} = 27\mu\text{m}^2$, benchmark s38417 (left); and $c_{\text{area}} = 277\mu\text{m}^2$, benchmark mult (right). 172

5.15 ECO example to estimate c_{area}. Gate G4 changes from INV size 1 to INV size 2, dislocating cells G2 and G3. There are 6 pins that are moved by the change, but the number of dislocated pins, $m_1 = 5$, because pin G4/Z still overlaps with its old location. 176

5.16 Error histogram of the difference between the estimated ECO area values ($\hat{c}_{\text{area}}$) and the actual ECO area values (c_{area}) for 644 data points over the benchmark s35932. 176

5.17 Results comparing the feasibility of the original design, a modified design using a β-guardband, and the Initial Design Method (ID). The Initial design method improves the feasibility substantially. 187

List of Tables

2.1 Summary of Score and Rank methods for gate sizing and threshold voltage assignment. 33

2.2 Summary of discrete gate sizing and threshold voltage assignment methods. 56

3.1 Improvements reported in literature. 58

3.2 Rise times (delays) and transitions (slews) in ns for a nominal inverter gate as a function of the input transition time and the output capacitance. . . . 65

3.3 Setup times for a rising input to a nominal D-Flip Flop as a function of the input transition and the clock transition. 66

3.4 Leakage power for a NAND3 cell as a function of its inputs. 71

3.5 Leakage powers for the NAND2_X1 gate. A1 and A2 are the two input pins. 73

3.6 Effect of synthesis on the ISCAS '85 benchmarks [HYH99] (adapted from [LG10]). All times are in ns. 75

3.7 Effect of different timing constraints used during synthesis on the final layout (adapted from [JK10]) for an AES encryption circuit from [OPE]. After synthesis, a timing constraint of 2.0ns is used to perform placement, routing and incremental timing optimizations. 77

3.8 Comparisons using UCLA timer. Percentages denote improvements over a feasible-start greedy algorithm (see Section 2.3.3). 81

4.1 Corners in the Nangate 45nm library [NANb]. 94

4.2 Notations 98

4.3 Measures of statistical leakage power 99

4.4 Number of gates in test circuits and the corresponding delay target in parenthesis (in ps) 106

4.5 Gate sizes in the Nangate Library 106

4.6 Rank correlation τ 108

4.7 Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations . . . 117

4.8 Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations . . . 118

4.9 Worst-case Suboptimality for $p_{m3\sigma}$ 120

4.10 Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations and no size upper bounds 123

4.11 Suboptimality ratios $\delta_{\text{full,max}}/\delta_{w,\text{max}}$ 125

4.12 Worst-case suboptimality (δ_{wc}) of p_{approx} as an estimate of $p_{m3\sigma}$ for a maximum size of 32, and w, l, v_{th} assignment 131

4.13 Sizing Results - Size independent variations 141

4.14 Sizing Results - Size Dependent Variations 143

5.1 Errors in v_{t} suboptimality estimation (as a percentage of the total power). Statistics are over the range of delays. 156

5.2 Errors in gate sizing suboptimality estimation (as a percentage of the total power). Statistics are over the range of delays. 164

5.3 Benchmark Information for the nominal process 181

5.4 Experimental Results comparing LPECO-S with the commercial tool . . . 182

Acknowledgments

I am indebted to Professors Lieven Vandenberghe and Jason Cong, who supervised and supported my first three years of research, spending many, many, hours providing guidance. I am fortunate to have benefited from their vision, patience, and dedication to research, and I developed immensely as a researcher under their guidance.

I am also indebted to Professor Puneet Gupta with whom I spent the latter three years of my graduate studies. He was the source of many interesting research topics, and under his supervision, I had to fortune to write a survey of gate sizing, advise high school, college, and graduate students, and travel to both Europe and Asia. He provided the guidance and support to finish this dissertation, and it would not have been possible without him.

I have been very fortunate to have ample support during my graduate studies. I am grateful to Professor Dejan Markovic and his service in my dissertation committee. I am grateful for my friends at the VLSICAD lab, in Professor Vandenberghe's research group, and in the NanoCad Lab, with whom I have spent many coffee breaks, and lunch breaks. I am grateful for my family and my girlfriend Susan, who have been nothing but supportive. I am also grateful for the funding support of the Semiconductor Research Consortium (SRC tasks 1460 and 1816), and the National Science Foundation Award 0811832. Lastly, I am grateful to all of my teachers who are responsible for my love of learning.

Research in Section 3.4.1 was performed in conjunction with Santiago Mok, and research in Section 4.3 was performed in conjunction with Professors Lieven Vandenberghe and Jason Cong, and appeared as [CLV08]. Research in Section 4.2 was performed in conjunction with Professors Jason Cong and Puneet Gupta, and appeared as [CGL09, CGL10]. A significant amount of background material on Gate Sizing in this dissertation was drawn from, and appeared in [LG12].

Vita

2003	B.S., Electrical Engineering and Computer Sciences
	B.A., Applied Mathematics and Philosophy
	University of California at Berkeley
2004–2005	Systems Engineer
	Raytheon Company, Bedford, Massachusetts.
2006	M.S., Electrical Engineering
	University of California at Los Angeles
2006–2007	Teaching Assistant
2006–present	Research Assistant
	University of California at Los Angeles

Publications

Jason Cong, John Lee, and Lieven Vandenberghe, "Robust gate sizing via mean excess delay minimization," Proc. Int. Symposium on Physical design, April 2008, pp.10-14, Apr. 2008.

Jason Cong, Puneet Gupta, and John Lee, "On the futility of statistical power optimization," Proc. Asia and South Pacific Design Automation Conference, pp.167-172, Jan. 2009.

John Lee and Puneet Gupta, "Incremental gate sizing for late process changes," IEEE International Conference on Computer Design (ICCD), pp.215-221, 3-6 Oct. 2010.

Jason Cong, Puneet Gupta, and John Lee, “Evaluating Statistical Power Optimization,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.29, no.11, pp.1750-1762, Nov. 2010.

Jason Cong, John Lee, and Guojie Luo, “A unified optimization framework for simultaneous gate sizing and placement under density constraints,” IEEE International Symposium on Circuits and Systems (ISCAS), pp.1207-1210, May 2011.

John Lee and Puneet Gupta, “Discrete Circuit Optimization: Library Based Gate Sizing and Threshold Voltage Assignment,” Foundations and Trends in Electronic Design Automation: Vol. 6: No 1, pp 1-120.

John Lee and Puneet Gupta, “Parametric Hierarchy Recovery for Layout Extracted Netlists,” to appear in the International Symposium on VLSI, August 2012.

John Lee and Puneet Gupta, “ECO Cost Measurement and Incremental Gate Sizing for Late Process Changes,” submitted to the ACM Transactions on Design Automation of Electronic Systems.

Santiago Mok, John Lee, and Puneet Gupta, “Discrete Sizing for Leakage Power Optimization in Physical Design: A Comparative Study,” submitted to the ACM Transactions on Design Automation of Electronic Systems.

John Lee and Puneet Gupta, “Impact of Range and Precision in Technology on Cell-Based Design,” submitted to the International Conference on Computer-Aided Design.

CHAPTER 1

Introduction

Gate sizing and threshold voltage assignment are widely used to optimize digital circuits, as tools to explore the tradeoffs in power, timing, area, yield, crosstalk, statistical power, statistical delay, and soft-errors. They can also be used incrementally as a method for optimizing post-layout designs after placement and interconnect routing. After over three decades [RWG77], research is still active in the area.

Of the many variations on the gate sizing and threshold voltage assignment problem, this work will consider the case of gate sizing and threshold voltage assignment for *digital designs*, with following metrics:

- Leakage power
- Dynamic power
- Clock period.

The application most commonly found in literature today is to minimize some combination of the leakage power and the dynamic power, with a constraint on the clock period. In the case that timing closure is important, the objective may be to minimize the clock period.

Dynamic power is composed of two components: the power that is used to charge and discharge capacitances, and the power that is lost due to the short circuit current that flows when both PMOS and NMOS are conducting during a transition. Thus, if the total capacitance of a gate is C_g, and the gate switches at a frequency f_{switch}, then the power dissipation due to charging and discharging gates ($p_{\text{dynamic(cap)}}$) is approximately:

$$p_{\text{dynamic(cap)}} = f_{\text{switch}} \cdot (\frac{1}{2} C_g V_{\text{dd}}^2). \tag{1.1}$$

As the capacitance of a gate is linearly related to the area of the gate, source, and drain:

$$C_g \propto w \cdot l, \tag{1.2}$$

increasing the gate widths will have a linear effect on the dynamic power due to capacitance charging.

The other component of dynamic power is the *short-circuit* power ($p_{\text{dynamic(sc)}}$). This is the power that is lost when both the PMOS and NMOS are conducting while the gate is in transition. When the input transition is fast, then the short-circuit power is minimized. However, a slow input transition can cause a 5X to 10X increase in the short-circuit power, and using minimum sized devices may hurt overall power [SC95]. From [SN90], a short-circuit current expression based on the α-power law is:

$$p_{\text{dynamic(sc)}} \propto \tau \cdot V_{\text{DD}} \frac{w}{l} \cdot \frac{(1 - 2(\frac{v_{\text{t}}}{V_{\text{DD}}}))^{\alpha+1}}{(1 - (\frac{v_{\text{t}}}{V_{\text{DD}}})^{\alpha}}, \tag{1.3}$$

where τ is the input transition time, and α is a fitting coefficient for the technology library[1]. This indicates that the short-circuit current is roughly linear in the input transition time and the gate length, and is inversely proportional to the gate lengths[2].

The leakage power is dominated by the sub-threshold conduction from the source to the drain. This happens because the transistor is not fully "OFF", and has a current proportional to [SSK87]:

$$\frac{w}{l} \cdot e^{\frac{V_{\text{gs}} - v_{\text{t}}}{v_T}} \left(1 - e^{\frac{-V_{\text{ds}}}{v_T}}\right), \tag{1.4}$$

where $v_T = 26mV$ is the thermal voltage and V_{gs} is the gate-to-source voltage. This expression shows that the sub-threshold is a linear function of gate-width, but is an *exponential* function of the threshold voltage v_{t}.

The relationship between the gate size, length, and v_{t} as a function of the delay can be approximated to a first-order using an *α-power law model* [SN90]:

$$\text{Delay} \propto \frac{1}{\frac{w}{l}(V_{\text{gs}} - v_{\text{t}})^{\alpha}}, \tag{1.5}$$

[1]For the 45nm Nangate Library, $\alpha \approx 1.4$.

[2]This is just an approximation; the threshold voltages are also dependent on the gate lengths.

where w is the gate width, l is the gate length, V_{gs} is the gate-to-source voltage, and v_{t} is the threshold voltage. This expression indicates that the gate delay is roughly linear in the gate length, and inversely proportional to gate widths.

Increasing the gate widths also increases the capacitances of its input pins proportional to $w \cdot l$, as in (1.2). Increasing the size will therefore increase the delay of the fanin gates, and may have a cumulative negative effect on the delay. This is why gate sizing requires a *balancing* of the loads in the design, as increasing the sizes may actually hurt efforts to meet the delay constraints. In contrast, changing the threshold voltages has a milder effect on the input capacitances, and is generally between $\pm 5\%$ (see Section 3.1.1).

Formally, the gate sizing problem can be expressed as an optimization problem. For example, with the vector of cell options $\vec{\omega}$, the delay-constrained power optimization problem is:

$$\begin{aligned} \text{minimize} \quad & \text{Power}(\vec{\omega}) \\ \text{subject to} \quad & \text{Delay}(\vec{\omega}) \le T_{\max} \\ & \omega \in \Omega, \end{aligned} \tag{1.6}$$

where $T_{\max}$ is the clock period. Similarly, the power-constrained delay optimization problem is:

$$\begin{aligned} \text{minimize} \quad & \text{Delay}(\vec{\omega}) \\ \text{subject to} \quad & \text{Power}(\vec{\omega}) \le P_{\max} \\ & \omega \in \Omega. \end{aligned} \tag{1.7}$$

This form helps summarize the objectives and constraints in the optimization process. The variables ω in the optimization process are the cells used to implement the gates, and is used as a generalization of gate size w, threshold voltages v_{t}, and other potential design variables in the optimization problem. The notation $\vec{\omega}$ is used to represent a vector of cells, and the cell option for gate g is denoted ω_g.

There are two distinct versions of the gate sizing problem: the continuous gate sizing problem, when Ω is a continuous, connected set, and the discrete gate sizing problem, when Ω is a discrete set. Continuous sizing problems were the first to appear in literature [RWG77] and formed the bulk of the research in the 1980's and early 1990's. Early research focused on heuristics for the sizing problem [FD85], however with the increase in computing power, nonlinear programming methods for continuous gate sizing became dominant in literature [BJ90, CCW99, BKP05]. Continuous gate sizing is still used today in the context of custom designs [BAB07], and are used for the high-performance circuitry in these designs. In addition, it is sometimes used as the starting point for discrete sizing methods [CSH95].

The discrete gate sizing problem gained importance after standard cell library design became widespread in the 90's[3], and is the most widely used form of gate sizing today. The popularity of discrete gate sizing comes from the widespread use of electronic design automation for standard cell designs at the synthesis, placement and routing stages. The automation of the physical design process provides designers with a method to tackle the increasing complexities of circuit design [BAB07].

In contrast, the threshold voltage assignment problem is always treated as a discrete one, as the cost to creating fine-grained v_{t} is prohibitive– each additional threshold voltage is costly in terms of mask costs, fabrication costs, and production time. In practice, a limit of three or four threshold voltages is used.

There is extensive literature on each of the gate sizing and threshold voltage assignment methodologies above. Examples of continuous gate sizing research can be found in [RWG77, FD85, SRV93, CHV96, CCH98, CH99, CCW99, KKS00, KYT02, BVS02, SMN05, BKP05]. Discrete gate sizing research will be covered in Section 2.3 and can be found in [Cha90, LH95, CSH95, Cou96, CS96, PDC98, WV98, CS00, WRK00, PRC01, HO01, CS02, GKS04, SSS05, CK05, HKH07, RHC08, Hel09, WD09, LH09, HKH09, LG10, ALQ11, LH11, HHS11, OBH11]. Threshold voltage assignment also has a large literature surrounding it, [WCR99, WRK00, KC00, PRC01, KS02, KYT02, SEO02, NDO03], and will also be covered in Section 2.3. To summarize the achievements of prior research, there

[3]See, for example, [TKT84].

are several topics that can be considered well understood:

1. Methods for modeling continuous sizing problems and v_t assignment [FD85, BJ90, KKS00, BKP05].
2. Optimal methods for the posynomial gate sizing and v_t assignment problem [SRV93, CHV96, CCH98, CH99, KKS00, KYT02, BVS02, SMN05, BKP05].
3. Greedy methods for discrete gate sizing and v_t assignment [FD85, CHM96, Cou97, SEO99, WRK00, GKS04, SEO02, SSB04a].

In contrast, there are still open questions in gate sizing that are related to developments over the last decade, which caused a shift in the way that physical design is done. The first set of challenges was motivated by the continuing descent of lithography towards the diffraction limit, and the increasing effects of threshold voltage variations [Vis03, War11]. Control over transistor specifications has been decreasing, prompting the need for statistical design methodologies to manage the spread in transistor parameters, and flexible redesign methodologies to account for the uncertainty in the target manufacturing processes. The next development is the rising proportion of leakage power as a portion of total power, putting increasing pressure on the design and algorithms to provide (1) a methodology to cope with uncertainty, and (2) methods to provide performance benefits through design methodologies and algorithms that are closer to *optimal.* This dissertation tackles a few of these open questions in gate sizing and threshold assignment:

1. Which discrete sizing heuristic is the best?
2. Are there sizing methods that can handle statistical delay directly?
3. What are the potential benefits of optimizing statistical power?
4. What is the effect of novel technologies on the gate sizing problem?
5. What is the impact of the range and precision of standard library cells on the optimal power?
6. What methods can be used to perform incremental gate sizing for post-layout designs?

These topics are not only unified by their application, which is gate sizing, but they are also unified by the approach that is used to tackle these topics. Underlying the field of gate sizing is optimization– nonlinear programming, convex programming, combinatorial optimization, dynamic programming– that is used to approach the problem. In addition, there are fundamental concepts– bounding hyperplanes, convex analysis– that are used to construct bounds and approximations. These tools are used to answer these open questions, and to progress the state of a problem over three decades old.

1.1 The physical design process

Gate sizing and threshold voltage assignment are a part of the larger Electronic Design Automation (EDA) ecosystem that transforms Register Transfer Language (RTL) descriptions into a physical layout. The process of creating the physical layout is called the *physical design* process, and consists of six steps:

1. **Logic synthesis**.
    - Input: RTL/HDL design description, library cell information, timing constraint information.
    - Output: Netlist mapped to the library.

    Transform the RTL/HDL design description into a gate level netlist, using a given cell library. Convert state machines, map arithmetic blocks, etc.
2. **Floorplanning**.
    - Input: Synthesized netlist, macro information, library information, standard cell row information, information on chip inputs and outputs.
    - Output: Floorplan with rows for standard cell placement, pads for input and output, locations for macros.

    Create a die for the design, and allocate space for input output ports, macros, and library gates.

3. **Placement**.

    - Input: Synthesized netlist, floorplan.
    - Output: Locations for each of the cells in the design in a standard cell row.

    Places, flips and rotates cells into the rows created by the floorplanning algorithm. Minimizes the wirelength in the resulting layout, while meeting timing constraints (timing-driven placement).

4. **Clock-Tree synthesis**.

    - Input: Placed design, clock nets.
    - Output: Clock tree to distribute the clock signal to the sequential elements (flip-flops, latches, etc.).

    Creates a clock tree to distribute the clock signal across the design. The goals are to minimize the size of the clock tree (to minimize power), and the skew at each of the outputs of the clock tree. Buffers may be added to help distribute the clock signal.

5. **Routing**.

    - Input: Placed design, connection information, metal and via information from the library.
    - Output: Design with cells connected with wires.

    Connects cells in the library using metal layers and vias. The objective is to minimize the amount of interconnect needed, while observing design rules and meeting timing.

6. **Physical Verification and Yield Enhancement**.

    - Input: Placed, routed design.
    - Output: Verified design with improved yield.

    Improves the yield of the design. Corrects design rules violations, inserts metal fill, doubles vias, checks connectivity and topology.

A flowchart for the EDA process is shown in Figure 1.1.

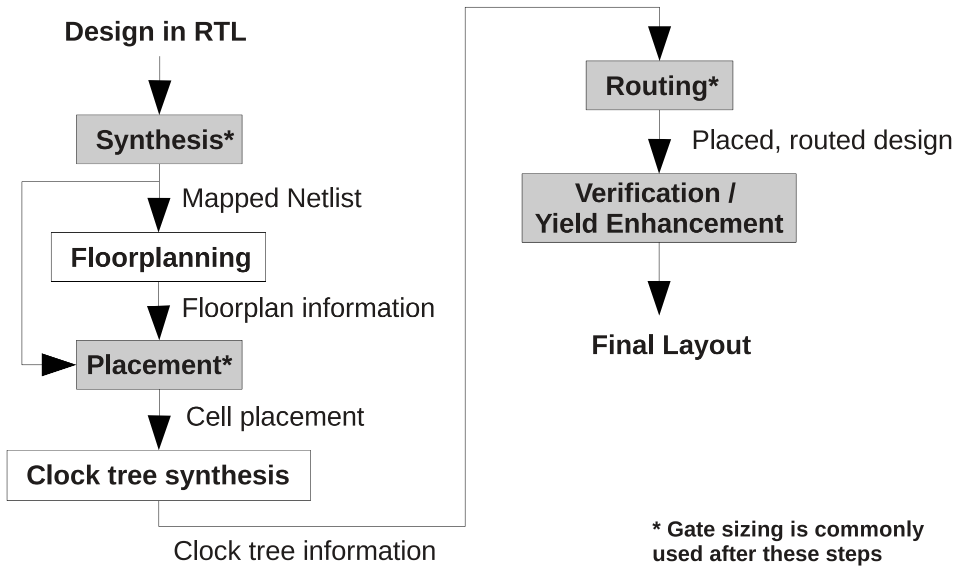


Figure 1.1: Electronic Design Automation flow.

Although gate sizing and threshold voltage assignment are not explicitly in the design flow above, they are used throughout the design flow to correct timing errors, and to optimize the design. For example, they are commonly used after placement to resize gates that violate maximum fanout rules or flip-flop setup time requirements (see, for example [Cad]). After clock tree synthesis, they are used to further fix rule violations and setup and hold violations, using the clock information from the clock-tree synthesis. After routing, they are again used to fix setup and hold violations, along with design rule violations. At this step, an incremental routing may be needed to account for changes in the cell sizes, and their corresponding pin locations.

They are also used at different points in the design flow to reduce the power consumption [ALQ11]. While this may reduce the power *using the same timing constraint*, in other instances the timing constraints may need to be relaxed to achieve a power reduction.

Gate sizing and threshold voltage assignment are powerful tools for optimization, and are the most widely used incremental optimization tools. These methods are more powerful and less intrusive than adjusting the placement or routing of the design. For example, fixing setup time violations using placement would require the gates on the violating path to be moved, and would require rerouting the interconnects. Similarly, fixing setup time violations using routing would also require the connections between the cells to be rerouted. In both cases, a significant portion of the design will need to be rerouted to provide benefits.

On the other hand, gate sizing and threshold voltage assignment are less disruptive than re-placing the cells or re-routing them. For example, the timing of a buffer driving a large wire load could be improved by increasing its size. This may result in a local rerouting to accommodate the different pin locations of the larger cell, and if there is no space around the surrounding cell, then an incremental placement will be needed to create space for the cell. However, this is preferable to rerouting large sections of the design, or adjusting the placements of tens or hundreds of cells.

In some cases, when only the gate lengths or threshold voltages change, the disruption is very minimal. In these cases, the cell dimensions and pin locations are the same, and these cell alternatives can be swapped without any change in the routing or the placement. The only verification needed is to ensure that the crosstalk noise, power, and timing constraints are satisfied.

Another advantage of gate sizing and threshold voltage assignment is that they are *versatile*, as they can be targeted to different optimization objectives. They have been used for power and timing optimization [Cou96], to fix noise constraints due to crosstalk [LMK90, XM01], to harden soft-errors due to radiation [ZM06], to improve yield [DS06, CSS05], and to minimize statistical power [SSB04b, CGL10].

1.2 Outline

The dissertation is structured as follows. Chapter 2 gives an overview of the principles needed to understand the gate sizing and threshold assignment problem, and a survey of gate sizing and threshold assignment methods. Chapter 3 describes the need and efforts towards a benchmarking system for standard cell library based discrete gate sizing. Chapter 4 considers the increasing variations in manufactured circuits, and its impacts on delay and power optimization. Chapter 5 addresses the impact of manufacturing technology on gate sizing. This dissertation is concluded in Chapter 6.

CHAPTER 2

Gate Sizing and Threshold Voltage Assignment Overview

Gate sizing, along with gate-length biasing and v_{t} assignment, works by matching the drive-strengths of the transistors to the capacitive loads in the design. Faster gates with larger drive currents are used to charge the larger capacitive loads due to long interconnect wires, I/O pads, or large fanouts. However, improvements in the delay must be weighed against increases in the power, and this tradeoff is the central mechanism behind gate sizing and threshold voltage assignment.

In this chapter, we provide an overview of gate sizing and threshold voltage assignment. First, the concept of Static Timing Analysis is explained, as it is the de-facto method for computing the delay. This is followed by a discussion on the tradeoffs between power and delay, which are explored quantitatively and qualitatively.

Lastly, an overview of the methods for gate sizing and v_{t} assignment is provided. These methods can be broadly divided as continuous methods and discrete methods. The distinction is whether the gate sizes and threshold voltages are considered to be a finite set of discrete values, or are chosen from a continuous range. The continuous methods draw heavily from mathematical programming methods, while the discrete methods are inherently combinatorial and rely on heuristics. These methods will be described in detail in this chapter.

2.1 Static Timing Analysis

Gate sizing and threshold voltage assignment methods that optimize timing or use timing constraints rely on Static Timing Analysis (STA) based timers. Static timing analysis computes the delays and arrival times in the graph without requiring data inputs or simulations. This is in contrast to dynamic timing methods which require simulation vectors to find critical paths. In this section, a broad overview of STA will be reviewed. For readers new to the subject of STA, we refer the reader to [Sap04] for a comprehensive treatment of the subject, and [BC09] for a practical introduction.

The principle behind STA is to propagate the best- and worst- case delays through the circuit. The arrival times (t_a) at each node are computed by traversing the circuit in topological order, starting with the primary inputs, and propagating the delays through to the primary outputs. The slacks (s) are computed in reverse-topological order, starting with the required arrival times (t_r) at each of the outputs and propagating these times through to the inputs. In sequential circuits, the inputs of the sequential elements are also considered to be primary outputs and the outputs are considered to be primary inputs.

When *setup* or long-path violations are of interest, the STA propagates the worst-case times through the graph to find the critical, or max path. This is to find whether the signal will arrive at its destination by the time it is required to arrive.

Similarly, when *hold* or short-path violations are of interest, the STA propagates the best-case times through the graph to find the shortest, or min path. This is used to find whether the signal arrives too quickly at the input of a flip-flop or latch, causing the state of the flip-flop to be set improperly, potentially causing metastability at the output.

2.1.1 Concepts and terminology

The terminology used to describe the topology and the signal and delay characteristics of a circuit is presented in this section. The first set of terms is used to describe the topology of the circuit. These terms are used to identify the inputs and the outputs of the circuit, the neighboring gates, and an ordering of the gates that can be used for performing the timing analysis.

- **Primary inputs** (PI): input ports in the design that are driven by external sources.
- **Primary outputs** (PO): output ports in the design.
- **Fanins of a gate** (fi(g)): the set of gates that drive the inputs of gate g.
- **Fanouts of a gate** (fo(g)): the set of gates that are driven by gate g.
- **Fanin cone of a gate**: the set of gates whose outputs have a path to the input of the given gate. This includes the fanins of the gate, the fanins of the fanins, and so on.
- **Fanout cone of a gate**: the set of gates whose inputs have a path from the output of the given gate. This includes the fanouts of the gate, the fanouts of the fanouts, and so on.
- **Topological order**: a way of ordering or numbering the gates where the fanins of a gate have a number smaller than the fanouts of a gate. When sequential elements are present, the sequential elements are placed at the top of the list. This ordering is generally not unique.

The second set of terms is used to describe the output waveform and delay characteristics. These terms are used to describe the time the signal arrives at a given point, and the characteristics of the waveform that arrives.

- **Cell rise time or rise delay**: time from when the input crosses 50% voltage to when the rising output crosses 50% voltage.
- **Cell fall times or fall delay**: time from when the input crosses 50% voltage to when the falling output crosses 50% voltage.
- **Cell rise transition or slew**: the time elapsed from when the output signal crosses the 30% voltage to when it crosses 70%.

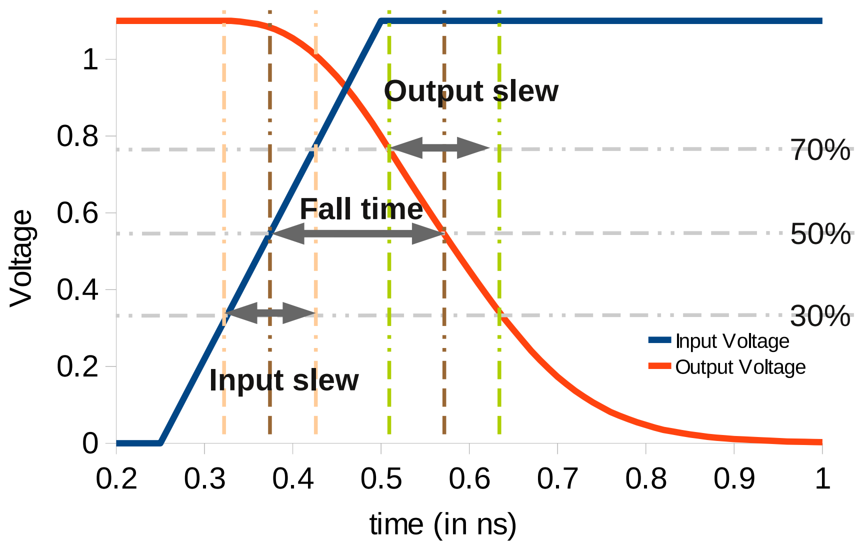


Figure 2.1: Plot of the transition and delay times for an size 1 inverter driving a .05pF load.

- **Cell fall transition or slew**: the time elapsed from when the output signal crosses the 70% voltage to when it crosses 30%.
- **Arrival time** (t_a): the time that the signal crosses the 50% voltage threshold at a given point in the circuit. The time associated with a rising and falling signal is called the rise arrival time and the fall arrival time, respectively.
- **Required arrival time** (t_r): the time a signal needs to cross the 50% voltage threshold at a given point in the circuit to be timing feasible. The required arrival time associated with a rising and falling signal are called the rise and fall required arrival time, respectively.
- **Slack** (s): the difference of the required arrival time and the arrival time, represents the amount of timing *slack* available at that point in the circuit.

An example of these concepts is shown in Figure 2.1. This figure shows a falling transition for an inverter in a 45nm [NANb] process. The input is assumed to be an ideal ramp, however, the output transition is not ideal. The input slew is .1ns, the fall time is .2ns, and the output slew is .12ns.

The 50% threshold for the delay, and the 30% to 70% thresholds for the slew are parameters that are set by the given timing library. The 50% threshold is generally standard across libraries, but libraries may differ in the slew thresholds. In libraries today, a 20% to 80%, or a 30% to 70% threshold are the most common.

At a given point in the circuit, the *arrival time* (t_a) is the time that a signal crosses the delay threshold. This is the time that the signal is said to *arrive* at the given point. Arrival times are a widely used to convert the arriving waveform into a single time. There are two types of arrival times:

- **Minimum arrival time**: the fastest time that a signal can arrive. Used to check for **hold-time** violations.
- **Maximum arrival time**: the slowest time that a signal can arrive. Used to check for **setup-time** violations and primary output arrival time requirements.

The *hold-time requirements* ensure that the signal does not arrive too quickly to the input of a sequential element. The setup-time requirements ensure that the signal does not arrive too late to be stored in the sequential element.

At a given point in the circuit, the *required arrival time* (t_r) is the time that the signal is *required* to arrive to meet the setup time and the output arrival time requirements. This is computed by subtracting the clock period with the delays downstream from the gate. Thus, if the signal arrives by the required arrival time, the delay conditions at the primary output or flip-flop inputs should be met.

The main use of the required arrival time is in computing the *slack* (s). The slack is defined as the difference between the required arrival time and the arrival times:

$$s = t_r - t_a \tag{2.1}$$

Thus, it measures the amount of timing "slack" that is available at the point in the circuit and how much the timing may be increased or must be decreased at that point in the design. Timing is infeasible wherever the slack is negative ($s < 0$), and timing is feasible wherever the slack is non-negative ($s \geq 0$).

The last set of terms is used to relate the signal waveforms and the arrival times between different adjacent parts of the design.

- **Timing arc**: a concept used to relate the delay between two adjacent points in the circuit.
- **Positive unate**: when a *rising* input to a gate causes a *rising* output, or a *falling* input causes a *falling* output.
- **Negative unate**: when a *rising* input to a gate causes a *falling* output, or a *falling* input causes a *rising* output.

The *timing arcs* are useful abstractions to connect the delays of adjacent points in the circuit. For example, the delay from an input A to output ZN is a timing arc, and the delay between an output of a gate to an input of another gate is a timing arc. This helps to conceptualize the circuit timing in terms of a graph with the delays represented by timing arcs along the edges.

The terms *positive* and *negative* unate are used to describe timing arcs in a cell. They connect the appropriate rising or falling arrival times with the corresponding rising or falling arrival time. For example, the timing arcs in an inverter cell are *negative unate*: a rising input will always cause a falling output.

2.1.2 Computing arrival times and slacks

Signals are propagated through the design using a series of *timing arcs* or *timing paths*. These timing arcs connect:

1. Primary inputs to gate input pins.
2. Gate input pins to gate output pins.
3. Gate output pins to other gate inputs or primary outputs using interconnects (nets).

4. Clock signals to input hold and setup conditions.
5. Clock signals to output “clock-to-Q” (C2Q) delays.

STA works through the following method.

1. Compute the delays and arrival times at the primary inputs (PI) and the flip-flop outputs.
2. In *topological order*, compute the delays and arrival times for the remainder of the gates.
3. Compute the arrival times at the primary outputs (PO) and the flip-flop inputs.
4. In *reverse-topological order*, compute the required arrival times and the slacks.

The arrival times along a path are found by adding the delays of the timing arcs.

However, there is a complication when the multiple inputs combine to form a single output. This happens in the case of multiple input gates, such as a 3-input AND, a multiplexer, or a 2-input exclusive-OR. In this case, the maximum of the arrival times at the output are propagated for a setup time or late-mode analysis, and the minimum of the arrival times at the output are propagated for a hold-time or early-mode analysis. When the *shortest path* is needed to check for hold-time violations is similar, the minimum arrival times are propagated.

The slack is computed using an extra backwards (reverse topological) pass through the circuit. In the backwards pass, the *required arrival times* (t_r) are backwards propagated through the circuit. It starts with the primary outputs and the inputs to the flip-flops, setting the required arrival time to the latest arrival time that the signal can arrive; this is generally equal to the clock period, or the clock period minus the setup time. The required arrival times are propagated backwards by subtracting the delays at each arc until it reaches the primary inputs to the flip-flops and latches. Once the required arrival times are set, the slacks are computed as the difference between the required arrival times, and the actual arrival times:

$$s = t_r - t_a \tag{2.2}$$

2.1.3 Interconnect delay and slew propagation

As scaling continues, interconnect delay becomes an increasingly large percentage of the total delay. This has increased the importance of computing accurate wire delays, and has motivated fast and accurate interconnect delay metrics [CPO02][1].

Interconnects are generally extracted from layouts in the form of parasitic networks. These parasitic networks model the capacitance and resistance properties of the interconnects and are used to analyze the propagation of the voltages down the wires.

The simplest and most common of these networks is the *RC tree*. If the restriction to a tree is dropped and loops are allowed, then the resulting network is called an *RC mesh*. When inductors are allowed, the resulting network is called an *RLC tree* or *RLC mesh*. Lastly, a pair of RC trees that are connected together using capacitors is called a *coupled RC tree*. There are many variations on this that emerge by mixing the type of network topology (line, tree, mesh), parasitic elements (resistors, capacitors, inductors), and coupling (coupled or not coupled).

The challenge in computing interconnect delay is to find the propagation delay and the transition time at the outputs. The most accurate method is to solve the differential equations related to the currents, charge, and voltages in the parasitic network. However, this is computationally prohibitive in large designs, and especially in the context of gate sizing and threshold voltage assignment, where fast delay estimates are needed for optimization.

The simplest method is the Elmore delay [Elm48]. This method approximates the delay as the expected value of the unit step response applied to the network. This is also equal to the first-moment of the impulse response, hence the Elmore delay is also called the *dominant-time constant*, or *first-order approximation* of the delay. This correlates well with

[1]For a detailed analysis of the subject, please see [CPO02] or [Sap04].

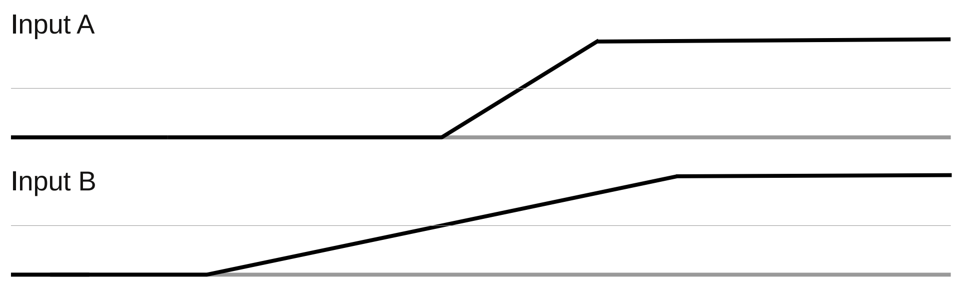


Figure 2.2: Output transition (slew) dilemma. The input waveforms to a two input gate are shown.

the actual delay [BKM93] (also see [ANC04]) and can be used for rough delay estimates or for short interconnect. In [GTP97], the Elmore delay is shown to be an upper bound on the actual delay, which justifies its use as a conservative estimate.

When accuracy is important, modern timers may employ more accurate methods that utilize higher order moments to improve the accuracy. Examples of this are the Asymptotic Waveform Evaluation Method [PR90] and the Krylov subspace methods [FF95, SKW96]. We refer the reader to [CPO02, Sap04] for more details on these methods.

Slew propagation. An important part of static timing analysis is the propagation of the input transitions, or slews [BC09]. The output waveforms play a major role in determining the output delay. For example, Table 3.2 shows that the difference in delay between the minimum and maximum transition is 2 to 7X. Thus, ignoring slew effects will cause inaccuracies in the delay. Furthermore, ignoring the slew in gate sizing and threshold voltage assignment eliminates the potential in increasing gate sizes or decreasing the threshold voltage to improve the downstream slew, and the corresponding downstream delays.

Computing the output slew is a difficult process when there are multiple inputs. This is because there may be a case where the input signals overlap. For example, Figure 2.2 shows the case of a two-input gate. The input A arrives at the gate later (in terms of the 50% delay threshold), but finishes transitioning faster. The input B arrives at the input earlier, but has a much slower transition.

When computing the output slews for max path analysis, the slower slew is usually propagated. Thus, in Figure 2.2, the arrival time of input A will be used to compute the output arrival time, and the slew of input B will be used to compute the output slew. However, most timers can also propagate the slew associated with the greatest arrival time if specified.

Similarly, for min path analysis, the fastest slew is propagated, as this is the worst case. Thus, in Figure 2.2, the arrival time of input B will be used to compute the output arrival time, and the slew of input A is used to compute the output slew.

2.1.4 Additional topics

Incremental Timing Analysis. Gate sizing and other optimization methods rely heavily on incremental timing analysis methods ([LT95, Sap04]). These methods are fast ways to find the updated timing information after changes in the design, from sources such as gate sizing, threshold voltage assignment, and routing. The idea is to update only the arrival times that may change; thus when the changes are relatively small, the speedup and be substantial. As an example of incremental timing analysis, consider Figure 2.3. If the gate g5 changes, then the arrival times of its fanout cone, g7, g8, and g9 may be affected, and must be recomputed, or at least checked to make sure the times are still valid. However, the change in g5 may also change the arrival times at the output of g2, and g3. Thus the fanout cone of its inputs should also be recomputed. Furthermore, when slack estimates are required, the required arrival times must be recomputed, and in this case, the required arrival times for the fanout cone (the gates whose delays change), and the fanin cone may need to be updated.

Differences between STA timers. While the basic methodologies that the timers use are all similar, there is still a difference between the outputs of different timers. Timers

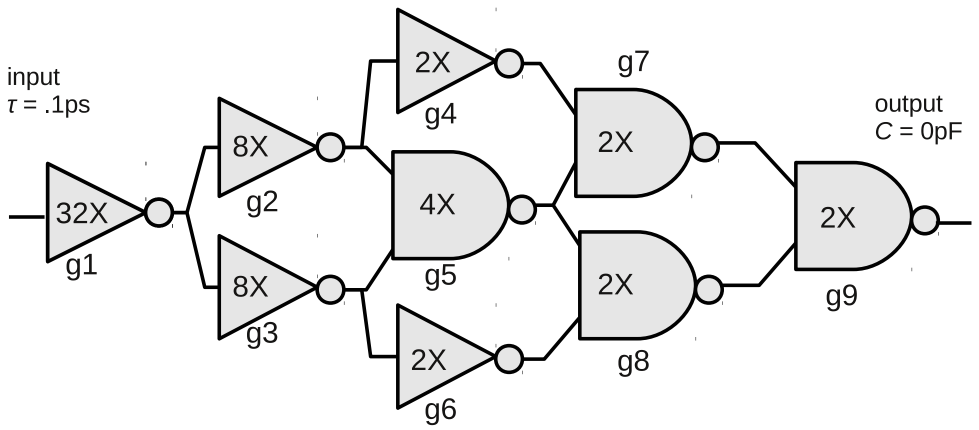


Figure 2.3: Mesh example from [GKK10]. The optimum sizes for minimum delay are shown.

used in place and route tools will generally use approximations for variation modeling, crosstalk and interconnect delay calculations. On the other hand, sign-off quality timers will use methods that are more accurate, but will require a longer runtime.

The main benefit to using better timing methods is to reduce the pessimism. STA is designed to be conservative, as the resulting design must meet the timing. However, a large amount of pessimism may cause over-design, while a timing method that is not conservative enough may result in catastrophic results.

The difference between the timers used in place and route, and the timers used in sign-off will be discussed in more detail in Section 3.3.1.

There are several additional concepts that are used in Static Timing Analysis [BC09] that will not be covered here:

- **On-Chip Variations**: modeling the variations in operating conditions (temperature and voltage) and manufacturing process (transistor dimension and threshold variations, interconnect dimension variations, etc.).
- **Crosstalk**: the effect of neighboring interconnects due to coupling capacitances.

- **Timing borrowing**: utilizing transparent latches to improve performance.

Overviews of these topics can be found in [BC09].

2.2 Continuous sizing methods for gate sizing and v_t assignment:

Continuous gate sizing has been well studied [RWG77, FD85, SRV93, CHV96, CCH98, CH99, CCW99, KKS00, KYT02, BVS02, SMN05, BKP05], and have been applied as heuristics for discrete gate sizing and threshold voltage assignment. These methods model the delay and power as continuous functions of the design parameters, and then use the models to formulate an optimization problem, which returns a set of continuous sizes. This section will survey methods for continuous sizing.

Modeling for continuous gate sizing and threshold voltage assignment problems has been well studied [KKS00, BKP05, RCC07]. Common delay models that are used are the linear models, posynomial models, and general models. The distinction between these types of problems is the speed in which they can be solved and guarantees of optimality. Linear and posynomial models have reasonably fast methods that can be used to solve them (see for example [CCW99, RS08]), and the global optimality of a solution can be verified. While linear and posynomial methods are the major classes of continuous sizing methods and models, other models such as device-level models with piecewise constant current vs. voltage characteristics with general nonlinear programming solvers have been used [CCH96]. However, these methods generally do not scale well, and are used for smaller designs. Furthermore, these general models may suffer from converging to local optima that are different from the global optimum.

Linear models and linear programming The linear model is arguably the fastest to optimize. Linear models for gate sizing were studied in [BJ90, TMF94]. Power models are roughly linear (see Section 3.2), and can be approximated as:

$$\text{Power} = \sum_{g \in \mathcal{G}} p_{g,\text{nom}} \cdot w_g, \tag{2.3}$$

where w_g is the width of the gate or the size of the standard library cell, and $p_{g,\text{nom}}$ is the power of a minimum sized gate or library cell.

Gate delay is not linear in the gate size (see Section 3.1.1), thus linear models for the gate delay require fitting and approximations. In [BJ90], the gate delay is linearized by:

$$d_g = c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'}, \tag{2.4}$$

where c_1, c_2, and c_3 are modeling coefficients, w_g is the width of gate g, and C_g is the input capacitance of g. These models can be improved to be any convex piecewise-linear function by using a series of inequalities:

$$\begin{aligned} d_g &\geq c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ d_g &\geq c_4 + c_5 w_g - c_6 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ d_g &\geq c_7 + c_8 w_g - c_9 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ &\ldots \end{aligned} \tag{2.5}$$

These inequalities make it possible to use arbitrary convex, piecewise linear functions to model the delay that may be preferable to geometric programming formulations (see, for example [RCC07]). The resulting linear program, using the block-based delay formulation in Section 2.3.1, is:

$$\begin{array}{ll} \text{minimize} & \sum_{g \in \mathcal{G}} p_g w_g \\ \text{subject to} & t_{a(g)} + d_g \leq t_{a(g')}, \quad \forall g' \in \text{fanout}(g) \\ & c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \leq d_g \\ & 0 \leq t_{a(g)} \leq T_{\max}, \quad \forall g \in \mathcal{G} \\ & w_{\min} \leq w_g \leq w_{\max}, \end{array} \tag{2.6}$$

whose optimum, $w^\star$, are the optimal continuous sizes.

2.2.0.1 Posynomial models and geometric programming

An improvement on the linear model is to use posynomial models [FD85] or generalized posynomials [KKS00]. Posynomials are equations of the form:

$$\sum_k \prod_j x_j^{\alpha_{jk}}, \tag{2.7}$$

where x_j is restricted to be positive, but $\alpha_{jk} \in \Re$ can be positive or negative. This form can handle the basic transistor models[2] for delay:

$$\begin{aligned} \text{Delay}_g &= R_g \frac{\sum_{g' \in \text{fo}(g)} C_{g'} w_{g'}}{w_g} && (2.8) \\ &= R_g w_g^{-1} \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'}. && (2.9) \end{aligned}$$

This results in the geometric programming problem:

$$\begin{array}{ll} \text{minimize} & \sum_{g \in \mathcal{G}} p_g w_g \\ \text{subject to} & t_{a(g)} + R_g w_g^{-1} \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \le t_{a(g')}, \quad \forall g' \in \text{fanout}(g) \\ & 0 \le t_{a(g)} \le T_{\max}, \quad \forall g \in \mathcal{G} \\ & w_{\min} \le w_g \le w_{\max}. \end{array} \tag{2.10}$$

While this is not convex as written, it is convex when the change of variables $w_g = e^{x_g}$ is made[3]. Extensions to these models can be made for v_{t} [CWC05] and gate length assignment. Also, methods to improve the accuracy of these models can be made by including rise and fall times, wire delays, and estimating slew effects[4]. These models have the property that there exists a unique optimum that can be found in polynomial time [BV04, RS08]. Furthermore, there are methods that can solve large posynomial based gate sizing problems relatively quickly (see for example [CCW99, JB08]).

[2]For example, the alpha power law model in Equation (1.5).
[3]Note that the objective is no longer linear under the substitution.
[4]see [KKS00, BKP05] for a discussion on posynomial models for gate sizing.

2.3 Discrete gate sizing and v_{t} assignment methods

Discrete gate sizing and threshold voltage assignment methods choose their gates from a from standard cell library. As the range of gate options is discrete, the problem is inherently combinatorial, and has been shown to be NP-hard [Li93].

There are many methods that approach the problem of discrete sizing and threshold voltage assignment using continuous sizing[5]. The methods that use rounding generally have at least 8 gate sizes per cell, as in [RHC08]. Other methods use the continuous solution as a starting point, and then employ heuristics to map the gate to discrete sizes [CSH95, SSS05, HKH07].

Other methods attack the discrete problem directly. For example, [Cou96] uses an optimization-derived approach that uses gradient-like functions to minimize the delay or power. [LH09] uses a dynamic programming and Lagrangian relaxation based heuristics, and [NDO03] uses slack allocation via linear programming to improve power. This chapter will survey the field of discrete gate sizing and threshold voltage assignment, however, the coverage is not exhaustive. Methods that are not covered here include the randomized exhaustive search based methods [WD09], and graph based methods [CS96, WV98, CS00, CS02].

In this section, we cover methods for timing-constrained power optimization, although similar methods are also applicable for other objectives.

2.3.1 Preliminaries

During the optimization process, gate sizing and threshold voltage assignment methods must find candidate gates to consider for optimization. The "Rank-based" algorithms in Section 2.3.2 and the slack budgeting methods in Section 2.3.5 use the idea of a sensitivity

[5]See, for example [CSH95, SSS05, HKH07, RHC08]

between the power and the delay:

$$\rho_d = \Delta t_a(g, \omega; \omega_0)/\Delta \text{Power}(g, \omega; \omega_0), \tag{2.11}$$

as in [FD85], where

$$\Delta \text{Power}(g, \omega; \omega_0) = \text{Power}(g, \omega) - \text{Power}(g, \omega_0) \tag{2.12}$$

$$\Delta t_a(g, \omega; \omega_0) = t_a(g, \omega) - t_a(g, \omega_0) \tag{2.13}$$

and Power(g, ω) is the power of gate g using library cell ω, and $t_a(g, \omega)$ is the arrival time at the output of gate g using library cell ω. This sensitivity is used as a first-order type analysis to determine how efficiently the delay can be improved for a change in power. Thus, the algorithms can prefer gates with better tradeoffs to meet a timing constraint with a lower power design.

A variant of this method is to use a power vs. slack sensitivity [GKS06]:

$$\rho_s = \Delta \text{Power}(g, \omega; \omega_0)/\Delta \text{Slack}(g, \omega; \omega_0). \tag{2.14}$$

This does a better job in capturing the effects of the delay, because both downstream (fanout) and upstream (fanin) effects due to slew will also be accounted for. For example, downsizing a gate may decrease the arrival time of the input gates, but the decrease in slew may increase the rise and fall delays in the fanout gates and in the *fanout cone* – all gates that downstream from the gate. Thus, this is an improvement on the power vs. delay sensitivity above.

Computing this sensitivity requires computing both the power and delay for each of the cell-options ω. This requires multiple calls to the timer to evaluate how the delay changes over the different cell options. When slacks are involved, this requires recomputing the required arrival times as well.

Finding and evaluating candidate gate moves is the most time consuming step in gate sizing and threshold voltage assignment. The challenge is to search the design and find gates *and* alternative cell options that: (1) will not violate timing and (2) will improve the objective. Many algorithms and most commercial tools use incremental methods to improve the speed of this process.

Avoiding significantly suboptimal solutions. It is difficult for discrete sizing and threshold voltage assignment algorithms to measure how far it is from optimal. Due to this, algorithms terminate whenever the method is unable to optimize any further. This is a major concern for the quality of the result, as it becomes imperative that when the algorithm is unable to optimize further, it is at a solution that is relatively close to optimal. Thus, many algorithms will take a circuit-wide or global approach to gate sizing and threshold voltage assignment.

One reason that the algorithm terminates in a significantly suboptimal solution is because the evaluation metrics at the core of these methods are *local* (see 2.3.1). Candidate gates and cell options are evaluated by how they individually affect local delay, slacks, or power. However, the effect of simultaneously changing large groups of gates is what is needed, and this discrepancy may cause sub-optimality.

Many methods make an effort to incorporate a circuit-wide perspective. For example, the "global sizing" method from [Cou97] in Section 2.3.2 uses a gradual method, by making a series of smaller steps to perform the optimization gradually. The justification for this rests upon the assumption that local metrics are good guides for small changes but they are not good guides for large changes. Thus, a series of smaller changes can help to improve the outcome. The slack, slew and delay budgeting methods in Sections 2.3.5 and 2.3.9 incorporate a circuit-wide view by utilizing a pre-processing step to allocate the slacks and slews. These allocations may take into account the topology of the design, or the slacks, or the relationship between slacks of neighboring gates. These allocations are then used to guide optimization at a local level. Lagrange multiplier weighting methods in Section 2.3.8 also use circuit-wide metrics to guide local optimization. The Lagrange multipliers are updated according the slack needs across the design. They are iteratively refined to balance the needs across different parts of the design.

Another reason for significantly suboptimal solutions is the large space of solutions that must be searched. For example, a 100,000 gate design with 5 cell options per gate yields a search space of $5^{100,000}$. While this large search space can be reduced by pruning methods, the number of options does not reduce down to a manageable size. However, Dynamic Programming based-heuristics in Section 2.3.7 perform a smart enumeration to quantify the interactions between different gates in the design, and prune the options that are suboptimal.

When evaluating discrete sizing and threshold voltage assignment methods, it is helpful to keep these considerations in mind. Most methods were motivated as an improvement upon the greedy heuristic in [FD85], and they incorporate specific methods to avoid the pitfalls of greedy optimization.

Block-based delay formulation In the STA method, the delays are calculated by propagating the worst-case arrival times and slews at every gate output. This is called a *block-based* method, and is in contrast to the *path-based* method, where the delays and arrival times are computed independently for each path. While the path-based method is more accurate, as less pessimism is needed when the actual path is known, it has the drawback of being computationally expensive, as the number of paths is exponential in the number of gates.

The block-based delay is essential for incorporating timing into mathematical programming based sizing method. This is done by formulating the delays and arrival times in a series of inequalities. These inequalities express the arrival time at the output of a gate as the *maximum* of the input arrival times, plus the delay through the gate. Thus, at the output of any gate g:

$$t_{a(g')} + d_{g'} \leq t_{a(g)}, \forall g \in \text{fi}(g'), \tag{2.15}$$

or equivalently:

$$t_{a(g)} + d_{g} \leq t_{a(g')}, \forall g' \in \text{fo}(g). \tag{2.16}$$

For primary input gates, we have:

$$d_g \leq t_{a(g)}, \forall g \in \mathrm{PI}, \tag{2.17}$$

and for primary outputs, we can write

$$t_{a(g)} \leq T_{\max}, \forall g \in \mathrm{PO}, \tag{2.18}$$

where $T_{\max}$ is a constant representing the maximum delay constraint for timing constrained problems, or as

$$t_{a(g)} \leq t_{a(\max)}, \forall g \in \mathrm{PO}, \tag{2.19}$$

where $t_{a(\max)}$ is a variable that can be used to minimize the maximum delay. While these inequalities can be replaced by using the max functions, e.g.

$$t_{a(g)} = \max_{\forall g \in \mathrm{fi}(g')} \{t_{a(g')} + d_{g'}\}, \tag{2.20}$$

the expression in terms of inequalities is better suited for use in optimization routines, as the max operator is generally non-differentiable.

The model can be improved to account for rise and fall times, and the different timing arcs in each gate. For example, the model in (2.15) can be improved to

$$t_{a(g'(\mathrm{rise}))} + d_{g'(\mathrm{rise}) \to g(\mathrm{fall})} \leq t_{a(g(\mathrm{fall}))}, \forall g \in \mathrm{fi}(g'). \tag{2.21}$$

In addition, the delays can be written as functions of the gate type, ω, and the input slew:

$$t_{a(g'(\mathrm{rise}))} + d_{g'(\mathrm{rise}) \to g(\mathrm{fall})}(\omega, \tau_{g'(\mathrm{rise})}) \leq t_{a(g(\mathrm{fall}))}, \forall g \in \mathrm{fi}(g'). \tag{2.22}$$

The differences in the input pins can also be accommodated by adding new variables, such as $t_{a(g'(\mathrm{rise}),\mathrm{in1})}$, $t_{a(g'(\mathrm{rise}),\mathrm{in2})}$. As this makes the notation difficult to follow, it will be omitted in the remainder of the text. However, it should be understood that the block-based delay formulations can also account for these subtleties.

Hold-time constraints can also be handled using the block-based formulation. In this case, the minimum arrival time is propagated, and the greater-than operator is used instead:

$$t_{a(g')} + d_{g'} \geq t_{a(g)}, \forall g \in \mathrm{fi}(g'). \tag{2.23}$$

At the primary outputs, the constraint is on the minimum delay:

$$t_{a(g)} \geq t_{a(\min)}, \forall g \in \mathrm{PO}. \tag{2.24}$$

Extensions to the hold-time model to handle rise and fall times, input pin dependencies, and other subtleties can also be made.

2.3.2 Score and Rank algorithms

Score and Rank algorithms (SR)[6] for gate sizing and threshold voltage assignment were developed as fast heuristics for gate sizing. These methods include TILOS [FD85], other greedy heuristics [LH95, WRK00, PRC01, HO01, GKS04], and its variants [Cou96, DBN97, SEO99, SEO02, SSB04a]. The defining characteristic of these methods is a scoring method for each gate that measures its ability to provide a power per timing tradeoff, and a ranking step to select the best gates for optimization. In this method, the gates are first evaluated using the scoring method; then the highest ranking gates are evaluated and changed. This process is iterated until no improvement is possible.

The steps in SR methods are summarized in Algorithms 1 and 2. The two algorithms are nearly identical but are different in their input design. The Algorithm 1 starts with a timing feasible design, and trades the extra slack for power savings, choosing the cells that provide the best power vs. delay tradeoff. On the other hand, Algorithm 2 starts with a timing infeasible design, and the gates are changed to reduce the negative slack.

While all algorithms in this category share these fundamental steps, they are distinguished by their choice in scoring functions:

- TILOS [FD85][7] and [WRK00] use the power vs. delay metric

$$\rho_d(g, \omega; \omega_0) = \Delta t_a(g, \omega; \omega_0)/\Delta \mathrm{p}(g, \omega; \omega_0), \tag{2.25}$$

[6]Note that the term "Score and Rank" is ours. We believe that it is a useful category that covers a large and important class of methods.

[7]In their paper, the delay vs. size sensitivity is used; however, as we are mainly interested in power, we present it as the delay vs. power sensitivity.

```
Input: Timing Feasible Design
while Slack($\mathcal{G}$) > 0 do
    Step 1: Score the gates and cell options according to a given score function $\rho$.
    Choose the best option for each gate;
    Step 2: Choose candidate gate moves by ranking the gates according to the
    score. If no power improving options are available, break;
    Step 3: Perform optimizations in decreasing order of ranking;
end
```

Algorithm 1: The general Feasible-Start Score and Rank Algorithm.

```
Input: Timing Infeasible Design
while Slack($\mathcal{G}$) < 0 do
    Step 1: Score the gates and cell options according to a given score function $\rho$.
    Choose the best option for each gate;
    Step 2: Choose candidate gate moves by ranking the gates according to the
    score. If no slack improving options are available, break;
    Step 3: Perform optimizations in decreasing order of ranking;
end
```

Algorithm 2: The general Infeasible-Start Score and Rank Algorithm.

where $\Delta \mathrm{p}(g, \omega; \omega_0)$ is the change in the power $(\mathrm{p}(g, \omega) - \mathrm{p}(g, \omega_0))$, and $\Delta t_a(g, \omega; \omega_0)$ is the change in the arrival times at the output of the gate $(t_a(g, \omega) - t_a(g, \omega_0))$. Note that this formulation accounts for changes in the fanin gate delays due to the changing input capacitances.

- For gate-length biasing, [GKS06] uses the score:

$$\rho_s(g, \omega; \omega_0) = \Delta \mathrm{p}(g, \omega; \omega_0) / \Delta s(g, \omega; \omega_0), \tag{2.26}$$

  where $\Delta \mathrm{p}(g, \omega; \omega_0)$ is the change in the power, and $\Delta s(g, \omega; \omega_0)$ is the change in the slack. The change in the slack accounts for timing changes in the fanout cone due to slew effects. This provides a more complete picture of the timing effects than (2.25). [PRC01] approximates slew effects using sensitivities instead of using the timer.

- The Duet method [SEO99, SEO02], along with [SSB04b], adapts the scoring function from [DBN97]:

$$\rho_{\mathrm{duet}}(g, \omega; \omega_0) = \tag{2.27}$$

$$-\frac{1}{\Delta \mathrm{p}(g, \omega; \omega_0)} \sum_{\forall \alpha \in \mathcal{G}} \Delta d_\alpha(g, \omega; \omega_0) \cdot \frac{1}{s_\alpha - s_{\min} + \epsilon} \tag{2.28}$$

  where $\alpha \in \mathcal{G}$ are the timing arcs in the design (see Section 2.1.1), $\Delta d_\alpha(g, \omega; \omega_0)$ is the change in the delay, $s_{\min}$ is the minimum slack in the entire design, and ϵ is a small number used for the case $s_\alpha = s_{\min}$. Note that in practice, the $\Delta d_\alpha(g, \omega; \omega_0)$ values are computed using analytical models and Elmore delay approximations to improve runtime.

- Coudert [Cou97] uses a relax function to score the gates:

$$\rho_{\mathrm{Relax}}(g, \omega; \omega_0) = (\alpha(\Delta \mathrm{p}(g, \omega; \omega_0)) - \epsilon) \cdot \phi\left(\frac{\Delta s(g, \omega; \omega_0)}{|s(g, \omega_0)| + \epsilon}\right) \tag{2.29}$$

  where

$$\phi(x) = \begin{cases} 1 + x & \text{if } x \geq 0 \\ \frac{1}{1-x} & \text{otherwise.} \end{cases} \tag{2.30}$$

method	input design	score	application
[FD85]	min area	(2.25)	transistor sizing
[Cou97]	min delay	(2.29)	gate sizing
[WRK00]	min area	(2.25)	v_t, gate sizing
[GKS04]	min delay	(2.26)	gate length biasing
[SEO99, SEO02]	all gates at high-v_t	(2.27)	v_t

Table 2.1: Summary of Score and Rank methods for gate sizing and threshold voltage assignment.

These methods are summarized in Table 2.1. The variables vary from transistor sizing to gate length biasing and v_t assignment. Also the starting point for each of these methods varies. Methods [FD85, WRK00, GKS04, SEO99, SEO02] start with a design with timing violations, with the gates at their minimum area and/or high-v_t configurations. These methods improve the timing until the design is timing feasible, trading power increases for delay reductions. In contrast, [Cou97, GKS04] start with a timing feasible design that meets the timing constraints. In these cases, positive slacks are traded for power reductions.

These methods will be discussed in greater detail below.

2.3.3 Greedy methods

Although Greedy methods were first applied to gate sizing over two decades [FD85] ago, they are still widely in use [CK07]. These methods have the benefit of being scalable, easy to implement, and they perform reasonably well (see Chapter 3). They use a *locally optimal* choice at each step in the algorithm, which gives these methods the title *greedy*.

The well-known TILOS method [FD85] was motivated as a fast heuristic to solve the continuous gate-sizing method. Their paper notes that the problem has a convex formulation and can therefore be solved optimally; however, due to runtime reasons, they provide a heuristic for approximate sizing.

The TILOS method for timing constrained area minimization starts with a design at minimum size (all gates sized down to their lowest power sizes), and then scores each gate on the critical paths using (2.25). The gate with the greatest score is chosen for optimization and is set to the cell option with minimum delay. This continues until the timing constraint is met. As this algorithm starts with a timing-infeasible design, this is an instance of a timing-infeasible start greedy algorithm.

This method can also be used for delay minimization. In this case, the method terminates when the delay cannot be further decreased.

TILOS has been adapted to other contexts. [WRK00] applies this method to the simultaneous gate sizing and threshold voltage assignment problem, by adding multiple v_t options into the optimization process. The method is identical to the TILOS algorithm, except that the cell options include v_t variants along with the different gate width choices.

[GKS04] applies the greedy heuristic to gate-length biasing. This method starts with a timing feasible design at minimum delay and all gates with nominal gate-lengths. The gate widths are fixed, and the method scores each of the gates and the cell options with the different gate length variants. The top ranked gates and options have their gate lengths increased, and this continues until no slack remains. As a variation on the original delay vs. power metric however, this method uses a slack vs. power metric to capture delay changes in the fanout cone due to slew effects.

[SEO99, SEO02, SSB04a] apply a modified score that accounts for the criticality of each gate– gates with less slack are given greater weights than gates with larger slack (see Equation (2.27)). Each timing arc and its delay vs. power sensitivity are weighted as:

$$\frac{1}{s_\alpha - \min(\mathrm{slacks}) + \epsilon}. \tag{2.31}$$

where ϵ is a small positive constant. For the most critical timing arcs, the weight is $1/\epsilon$, and it decreases as ($s_\alpha - \min(\text{slacks})$) increases. Thus, delays in timing critical paths can be improved even when other delays in non-critical paths are increased.

2.3.4 Global sizing

[Cou97] attempts to improve upon the greedy method. A score function "Relax" (2.29) is proposed to gradually tradeoff the delay for the power. The rationale for this approach is given in [CHM96]:

> "Optimizing the delay gives plenty of alternatives for power optimization, i.e., going far away from the infeasible region makes power optimization less likely to be trapped in a local minimum."
>
> "[Timing feasible start optimization works by] relaxing the delay constraints using a penalty/benefit function, as opposed ... to a greedy method that ... can give low quality results ... [because] resizing a few nodes to their local minimal power too "quickly" creates critical paths that can prevent most of the other nodes from being resized and saving more power."

```
GLOBAL SIZING: Relax the timing gradually;
Input: Design with gates optimized for minimum delay
while Power improves and Slack($\mathcal{G}$) > 0 do
    Step 1: Score the gates and cell options according to $\rho_{\text{relax}}$ (Equation 2.29);
    Step 2: Compute a set of moves: find a maximal ordered subset that minimizes
    the Power (rank the gates according to score, and choose the top $k$ gates);
    Step 3: Update the scores for changed gates and gates in neighborhood of
    changed gates;
end
```

Algorithm 3: The Global Sizing method [Cou97]

In effect, the relax function is designed to prevent the algorithm from converging too quickly to a poor solution, as may be the case with greedy methods. This method gradually trades the delay for power reductions, until it settles on a good solution.

In the Global Sizing Algorithm (Algorithm 3) there are two main differences from the TILOS algorithm. Firstly, the method requires the computation of a “maximal ordered subset of moves that minimizes the cost.” They suggest that this be done using a gradient-type approach, or conjugate gradient method.

The second difference is the incremental updates to the sensitivities. Unlike the TILOS algorithm, which only scores the critical gates, the global sizing requires that all gates are scored. Thus, an incremental method is proposed where after the initial iteration, a neighborhood around each changed gate, which is defined as the gates a given number of levels away from it, have their scores updated.

2.3.5 Slack and Delay budgeting methods

Delay budgeting methods [CK91, PDC98, NDO03] were developed as fast, scalable methods for gate sizing and threshold voltage assignment [CK91][8]. These methods are an improvement over the greedy methods in Section 2.3.2 because they can manage the slack budget between gates– they understand that the slack used on one gate in the design reduces the amount of slack left for other gates. Furthermore, these methods account for the topology of the design by prioritizing low sensitivity gates that are bottle-necks for many critical paths.

In this section, we will focus on the recent approach by [NDO03]. This method works in two phases:

- **Phase I**: Allocate a slack budget to each gate in the design.
- **Phase II**: For each gate, use the slack budget to reduce the power used by the gate.

[8][DA89] also provides a method for budgeting-based sizing, applied to a single logic gate.

In Phase I, there is an important consideration. When the design is not timing feasible, then the algorithm will need to distribute negative slacks, along with positive slacks. When a gate is allocated a negative slack, the gate must *improve* its timing. To circumvent this difficulty, a minimum delay design is used as a starting point, where the slacks are positive and have been maximized.

In Phase I, the slacks are allocated to the gates. This is done by first computing the power-delay sensitivities using (2.11). These sensitivities are then used in the objective of a linear programming method (Algorithm 4). These sensitivities *weight* the slacks allocated to the gates, and the solution to the linear program "allocates the optimal delays to gates depending on their ability to convert slack to power" [NDO03].

In Phase II, the slacks are converted into reductions in power (see Algorithm 4). This step is done at a local level, whereby each gate uses their slack allocation to reduce their power. For example, at a gate g, the options with smaller power consumption are evaluated to see if the change in delay is within the allocated slack budget. After the options are evaluated, the gate is changed to the least power option that meets the budget.

The biggest limitation to this method is that changing a gate will affect the slacks of neighboring gates; the sensitivities for each gate may change if any of their fanin or fanout gates change. An increase in the size of an input gate may provide a greater improvement in slack, while a decrease may cause an opposite effect. Similarly, an increase in the size of an output gate will improve the slack improvement of the current gate, and a decrease will reduce the slack improvement. These interactions are not considered by this method and are accounted for by performing multiple iterations of this method.

Moreover, due to slew effects, the delay sensitivities may also change if any gates in its fanin cone had changed. For example, if a change in the fanin cone from a gate improves the slew dramatically, then this may dampen the improvement seen at the current gate. Similarly, a decrease in the slew may increase the improvement that comes from increasing a gate's size.

```
Input: Design with gates optimized for minimum delay

$P_{\text{best}} \leftarrow \infty$;
while $\text{Power}(\mathcal{G}) < P_{\text{best}}$ do
    $P_{\text{best}} \leftarrow \text{Power}(\mathcal{G})$;
    Setup: Compute Power vs. Delay sensitivities;
    foreach $g \in \mathcal{G}$ do
        Compute $\rho_g =$
            $$\max_{\omega \in \{\text{CellOptions}(g)\}} \{\rho_d(g,\omega;\omega_0) = \Delta t_a(g,\omega;\omega_0)/\Delta \text{p}(g,\omega;\omega_0)\} \quad (2.32)$$
    end
    Phase I: Allocate slacks to gates using LP;
    Solve:
        $$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{g \in \mathcal{G}} \rho_g s_g \\ \text{subject to} \quad & t_{a(g)} + s_g \le t_{a(g')} \quad \forall g' \in \text{fanout}(g) \\ & 0 \le t_{a(g)} \le T_{\max} \\ & 0 \le s_g \le D_{g,\max} \end{aligned} \quad (2.33)$$
     with variables $s_g$, $t_{a(g)}$ to get slack allocations $s_g$.;
    Phase II: Use the slacks to reduce the power;
    foreach $g \in \mathcal{G}$ do
        Set library cell of gate $g$ to;
        $\text{argmin}_{\omega \in \{\text{CellOptions}(g) \mid s(g,\omega) - s(g,\omega_0) \le s_g\}} \{\text{Power}(\omega)\}$;
    end
end
```
Algorithm 4: Slack budgeting method in [NDO03]

Thus, this algorithm is sensitive to the order in which the slacks are distributed to the gates. Furthermore, due to these interactions, positive slack may remain after the first iteration. Thus, the method is iterated until the entire budget is used.

2.3.6 Methods based on Continuous Sizing

Continuous sizing methods are also used in conjunction with discrete sizing. The continuous sizing problem is first solved using the methods in Section 2.2, followed by a process whereby the continuous solution is "snapped" to a discrete one.

Continuous sizing methods are attractive as they have fast algorithms, and may have a global optimum that can be found efficiently. However, there two significant downsides, as noted above: the modeling errors, and the issues with snapping. Thus, these methods are used when modeling and snapping errors are tolerable [Cou97, HKH09], but they are severely limited in post-layout contexts.

2.3.6.1 "Snapping" continuous sizes back to library cells

After the continuous model is optimized, the continuous sizes need to be mapped, or "snapped", back to the available library cells. Intuitively, this is challenging when the number of library cells is sparse[9], however, [HKH09] reports that even in dense libraries, with cell sizes of $\{1, 2, 3, 4, 6, 8, 12, 16, 24, 32\}$, snapping to the nearest library cell results in slack violations and infeasible designs. [WD09] shows the importance of a "snapping"– the difference between an optimal snapping method and a greedy snapping method can be up to 51% of the power objective.

Moving from a continuous solution to the discrete sizes is difficult, and not guaranteed to be optimal. For example, suppose that the continuous solution is used to limit each gate to two cell choices: the library cell that is larger, and the library cell that is smaller than the continuous size. This still leaves $2^{|\mathcal{G}|}$ choices, and furthermore *the optimal solution may*

[9]See [HGS97, BKK98] for a discussion on selecting library cells

not be one of these choices [CSH95]; however results in [WD09] show it is within 1% of optimal for a set of 20 commonly used benchmarks.

Heuristics for snapping continuous sizes are studied in [CSH95, SSS05, MDO05, ZBS08, RHC08]. Simple snapping to the closest library cell is used in [MDO05, RHC08]; in [SSS05], the gate sizing is re-performed after the v_t is snapped, followed by the snapping of the gate sizes. In [CSH95] the continuous solution via linear programming is used to limit the size options. Gates that are set at the minimum size by the continuous sizer are fixed to be at the minimum; the options for the remainder of the gates are two sizes that are smaller than, and greater than the continuous size. The algorithm then creates a *state-space tree* that is used to enumerate the different possibilities. This process is tractable as the number of gates that are not at minimum size is “relatively small”.

[HKH09] uses a dynamic-programming like enumeration approach. This approach traverses the graph *breadth-first* from primary input nodes to output nodes, enumerating all the possible solution possibilities. When the size of the enumerated space becomes too large, it is pruned using a locality-sensitive hash that diversifies the set of solutions.

2.3.6.2 Modeling errors

Modeling errors are a large limitation for continuous-sizing based approaches. Even with posynomial models, errors of up to 10% were reported in [KKS00] for a 0.25μm technology. Furthermore, these errors were reported for custom transistors. Using transistors in standard cells are more difficult to approximate using standard cells. The standard height of each cell requires the transistors to be packed using transistor folding, and diffusion sharing, and requires metal pins to connect the pins to the routing layers. This alters the behavior of the transistor from the properties of an ideal transistor. In [RC05] errors of 5% to 23% were reported for a 0.13μm standard cell library.

While these errors can be reduced by applying corrections to the model, to increase the accuracy around the current set of sizes, the problem is still significant. Even modeling errors of 5% in the delay may introduce a significant suboptimality in the design (see, for example, Figure 3.14 for an example of the power vs. delay tradeoffs). Furthermore, modeling errors associated with ignoring slew effect affect gate sizing methods heavily; for example, Figure 3.10 shows the performance of sizing methods with and without slews. The performance is very different between the two options.

2.3.7 Dynamic programming based algorithms

Dynamic programming (DP) [Bel57] is an optimization method for problems with *stages*[10]. At each stage, a decision is made with the assumption that all *prior* decisions were optimal. For problems where optimal prior decisions can be modeled efficiently, this presents an effective method for optimization.

In the context of gate sizing and threshold voltage assignment, dynamic programming methods provide a structure to break apart the problem using decision stages and cost-to-go functions. At each stage, *cost-to-go functions* recursively define the cost for the objective for all prior stages of the design. After all the stages have been traversed, the final cost-to-go function is used to find the minimum cost solution.

Dynamic programming for gate sizing and threshold voltage assignment has been applied in both forward topological order [Cha90, HKH09, LH11][11], and reverse topological order [LH09]. In the forward topological order version, the power and the arrival times are propagated in the form of *cost-to-go functions*:

$$\begin{aligned} J_g(t_a) = & \\ & \min_{\omega \in \mathrm{CellOptions}(g)} \left\{ \mathrm{Power}(g,\omega) + \sum_{g' \in \mathrm{fi}(g_k)} J_{g'}(t_a - d(g,\omega)) \right\}. \end{aligned} \tag{2.34}$$

The cost function represents the minimum power, as a function of the arrival time at the gate's output. It is defined recursively as a function of the cost functions for its fanin gates,

[10]see [Ber05] for an introduction to the subject.

[11]Note that [Cha90] does not explicitly refer to itself as Dynamic Programming, but has the characteristics of DP.

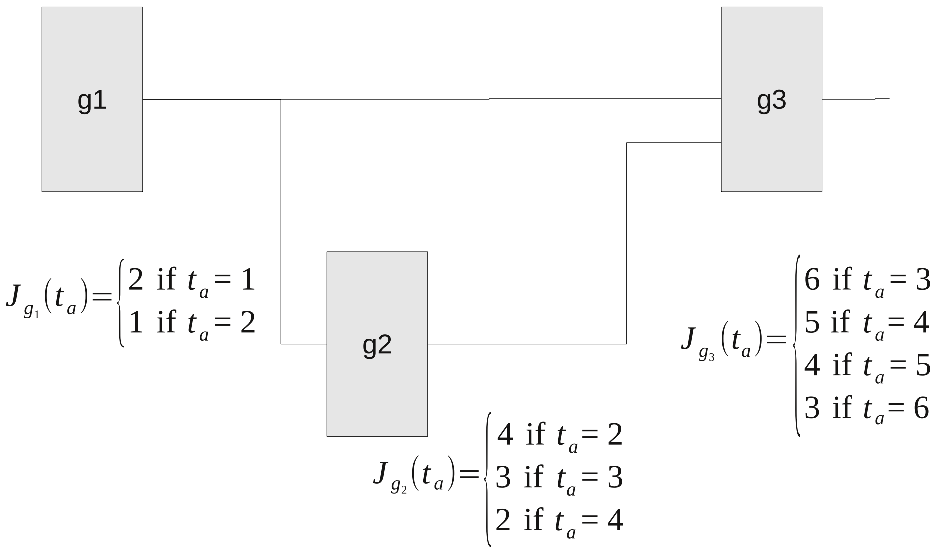
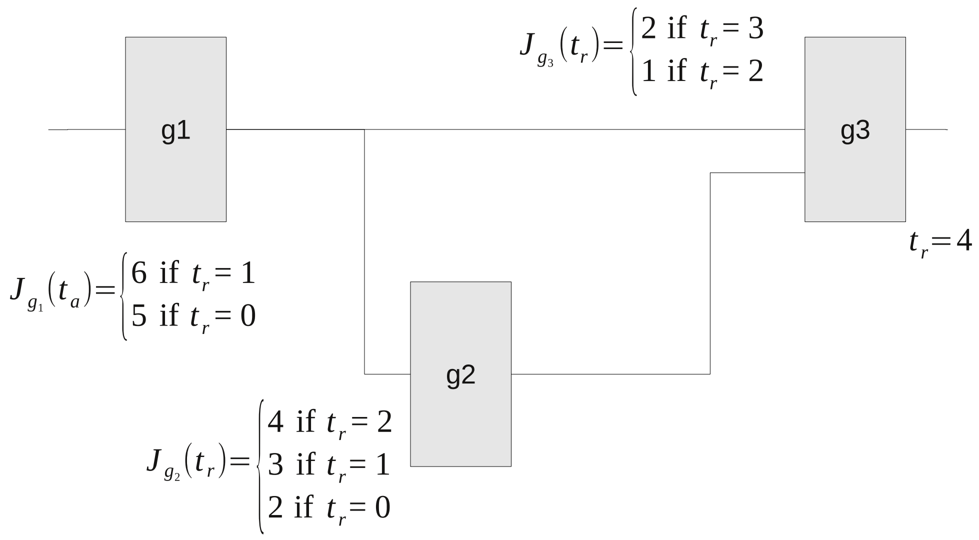

g1
g2
g3
$J_{g_1}(t_a)=\begin{cases}2 \text{ if } t_a=1\\1 \text{ if } t_a=2\end{cases}$
$J_{g_2}(t_a)=\begin{cases}4 \text{ if } t_a=2\\3 \text{ if } t_a=3\\2 \text{ if } t_a=4\end{cases}$
$J_{g_3}(t_a)=\begin{cases}6 \text{ if } t_a=3\\5 \text{ if } t_a=4\\4 \text{ if } t_a=5\\3 \text{ if } t_a=6\end{cases}$

Figure 2.4: Forward topological order dynamic programming example. A simple delay vs. power model is used for the gates, where either Delay=1 and Power=2, or Delay=2 and Power=1.

and the term $J_{g'}(t_a - d(g, \omega))$ is the power needed for the signal to arrive at the input by the time $t_a - d(g, \omega)$, which is the given arrival time minus the delay at the current gate. To simplify notation, we can assume that $J_g(t_a) = \infty$ when $t_a < 0$, and thus any infeasible arrival time will have a cost ∞.

An example of the forward version is shown in Figure 2.4. In this example, a simple delay model is used with two cell options for each gate: either Delay=1 and Power=2, or Delay=2 and Power=1. At g_1, this means that the arrival time of the output (t_a) can be 1 with a power cost of 2 ($J_{g1}(1) = 2$), or the arrival time can be 2 with a power cost of 1 ($J_{g1}(2) = 1$). At g_2 the computation becomes more complex: J_{g_2} is computed as a function of its fanin, J_{g_1}. $J_{g_2}(2)$ can be achieved only using the delay 1 cell option for g_2, and using $J_{g_2}(1)$, for a cumulative power cost of 4. Similar computations can be made for $J_{g_2}(3)$; however, in this case the minimizer is not unique – either of the two cells, g_1 or g_2 can be at the minimum delay option. At the primary output, the cost-function is evaluated with the



$J_{g_3}(t_r) = \begin{cases} 2 \text{ if } t_r = 3 \\ 1 \text{ if } t_r = 2 \end{cases}$
g1
g3
$t_r = 4$
$J_{g_1}(t_a) = \begin{cases} 6 \text{ if } t_r = 1 \\ 5 \text{ if } t_r = 0 \end{cases}$
g2
$J_{g_2}(t_r) = \begin{cases} 4 \text{ if } t_r = 2 \\ 3 \text{ if } t_r = 1 \\ 2 \text{ if } t_r = 0 \end{cases}$

Figure 2.5: reverse topological order dynamic programming example. A simple delay vs. power model is used for the gates, where either Delay=1 and Power=2, or Delay=2 and Power=1. The required arrival time at the primary output is assumed to be 4.

delay constraint to yield the optimal power, and the circuit can be traversed backwards to find the cell options that produce the minimum power design. Note that the computation of J_{g_3} was adjusted to avoid double counting the power of g_1.

In *reverse-topological order*, the procedure works from the primary outputs towards the primary inputs. The power and the *required* arrival times are propagated in this case, and the cost-to-go functions are defined recursively, as:

$$J_g(t_r) = \tag{2.35}$$

$$\min_{\omega \in \text{CellOptions}(g)} \left\{ \text{p}(g,\omega) + \sum_{g' \in \text{fo}(g)} J_{g'}(t_r - d(g,\omega)) \right\}. \tag{2.36}$$

An example of the reverse topological version is shown in Figure 2.5. In this case, the cost functions are simpler than the forward topological case, as the options for negative required arrival times ($t_r < 0$) can be eliminated[12]. Once the cost-to-go functions are

[12]However, this can also be done in the forward topological case – options with delay greater than clock period can be pruned.

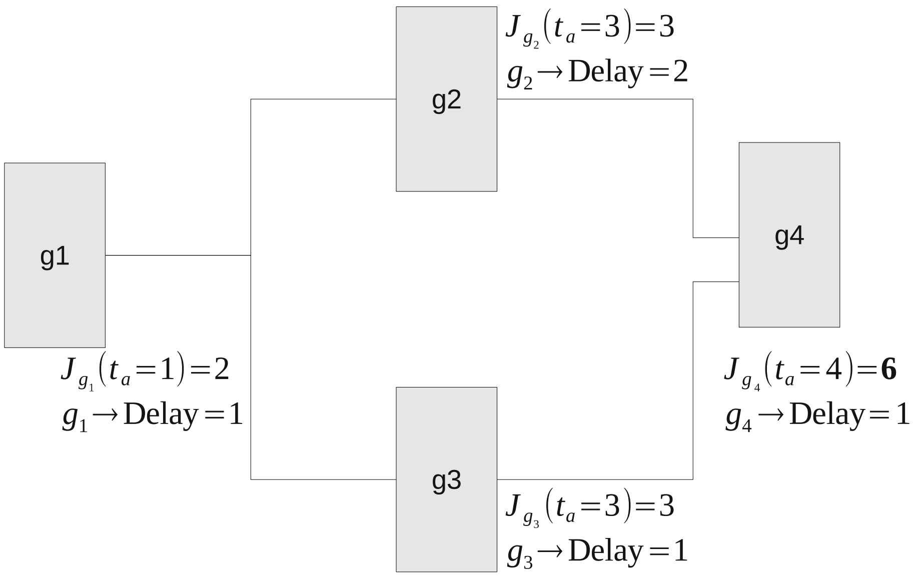
propagated to the primary input, the cost-to-go function is evaluated with $t_r = 0$ to find the optimal configuration. Again, note that the computation of J_{g_1} was adjusted to avoid double counting the power of g_3.

Note that in both cost-to-go functions, (2.34) and (2.35), the costs are computed by enumerating the different options at each gate. The information needed to store the cost-to-go functions is usually stored in the form of a table. The size of this table can be reduced by removing options that are not Pareto-optimal, and this is implicitly done in the cost-to-go functions. For example, in the forward topological case, $t_a = 4$ and $p = 5$ are certainly superior to $t_a = 5$ and $p = 5$ and the latter entry would be removed. This helps to reduce the complexity of the cost-to-go functions.

Dynamic programming based algorithms suffer from two limitations. The first limitation is the size of the solution space that must be propagated. Even after the non Pareto-optimal elements are pruned, the number of elements in the table may be very large. To combat this, [HKH07] uses locality-sensitive hash functions to reduce the number of entries to the table. In [Cha90, LH09], the slews are ignored, reducing the number of entries to a manageable size. Furthermore, while [LH09] uses the dynamic programming methods for slack minimization, they rely on Lagrangian methods to perform the timing constrained power optimization (see Section 2.3.8).

Secondly, they have difficulty with non-tree structures, e.g. when a gate has multiple paths that lead to it. This is present in the forward topological case in Figure 2.6; to account for this, the cost-to-go function must be adjusted. As written in (2.34), $J_{g_4}(t_a = 4) = 8$ because the cost of g_1 is double counted – it is present in J_{g_2} and J_{g_3}. However, the actual minimum cost is $J_{g_4}(t_a = 4) = 6$.

There is also an inconsistency in the cell options that are assigned by the DP. In the example in Figure 2.6, for a delay constraint of 4, the cell options are set by traversing the graph in reverse topological order. g_4 is set to the cell option that minimizes $J_{g_4}(t_a = 4)$, which is Delay=1 / Power=2 option. Next, suppose that g_2 is also set to Delay=2 /



g1
g2
g3
g4
$J_{g_1}(t_a=1)=2$
$g_1 \rightarrow \text{Delay}=1$
$J_{g_2}(t_a=3)=3$
$g_2 \rightarrow \text{Delay}=2$
$J_{g_3}(t_a=3)=3$
$g_3 \rightarrow \text{Delay}=1$
$J_{g_4}(t_a=4)=\mathbf{6}$
$g_4 \rightarrow \text{Delay}=1$

Figure 2.6: Design with reconvergent fanout. This structure is problematic in Dynamic Programming formulations.

Power=1, but g_3 is set to Delay=1 / Power=2 – this is valid in the DP algorithm. When assigning the size at g_3, it does not coordinate with the parallel reconvergent path through g_2, forcing g_1 to be Delay=1 / Power=2, and resulting in a total power of 7, which is greater than the optimal power 6.

[LH09] uses a heuristic to fix this problem by considering the problem in two steps. In the first step, when the cell options are being assigned, and multiple cell options are possible (when the minimizer in (2.34) is not unique), all of the cell options are set as viable candidates for that cell. After this step is over, each gate may have multiple candidate cell options that it can be assigned to. This is reconciled in the next step, when the algorithm traverses the graph in topological order, and assigns each gate to the cell option that is locally optimal.

2.3.8 Lagrangian relaxation

Lagrangian relaxation is widely used in gate sizing and threshold voltage assignment to perform continuous sizing with posynomial models (see, for example, [CCW99, ST02, CWC05, WDZ09]), and it has also been applied directly to discrete gate sizing [LH09, HHS11, OBH11]. It provides scalable methods for large-scale gate sizing, and they provide a useful way to convert constrained optimization problems into unconstrained optimization problems.

Lagrangian relaxation methods convert a constrained optimization problem into an unconstrained problem by using by using *Lagrange multipliers*[13], also known as dual variables, to add the constraints to the objective. For the optimization problem:

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \le 0, & i = 1, ..., m \\ & g_i(x) = 0, & i = 1, ..., p \\ & x_{\min} \le x \le x_{\max}, & \end{array} \tag{2.37}$$

the associated *Lagrangian* is:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i g_i(x), \tag{2.38}$$

where λ and ν are the Lagrangian multipliers. Note that the constraints $x_{\min} \le x \le x_{\max}$ do not appear in the Lagrangian (2.38). In gate sizing and threshold voltage assignment problems, it is more efficient to handle these constraints *implicitly* by restricting the domain of the function. Thus, the domain of x in (2.38) is restricted to $x_{\min} \le x \le x_{\max}$.

The *Lagrange dual problem* associated with the Lagrangian is:

$$\max_{\lambda \ge 0, \nu \in \Re} \left\{ \min_{x_{\min} \le x \le x_{\max}} \{L(x, \lambda, \nu)\} \right\}. \tag{2.39}$$

This is the result of minimizing the Lagrangian over the primal variables, x, and maximizing over the dual variables, λ and ν. This formulation is interesting when the minimization $\min_{x_{\min} \le x \le x_{\max}} \{L(x, \lambda, \nu)\}$ can be performed faster than solving (2.37) directly.

[13]See [BV04] for a good reference on the subject

The main motivation for solving the Lagrange dual problem (2.39) comes from two theoretical results.

- The solution to the dual problem (2.39) is always *less than* the primal (2.37).
- The solution to the dual problem (2.39) is equal to the solution of the primal (2.37) for a wide class of convex continuous problems[14].

Using the Lagrange dual problem is said to be using *Lagrangian relaxation*[15], and it is a useful method to find lower bounds or optimize certain types of problems. Generally speaking, these methods perform well when there is an efficient way to minimize $L(x, \lambda, \nu)$ over x for fixed λ and ν.

Gate sizing and threshold voltage assignment benefit from Lagrangian relaxation. In this case, the associated Lagrangian for (2.10) is:

$$L(w, t_a, \lambda) = \sum_{g \in \mathcal{G}} p_g w_g + \tag{2.40}$$

$$\sum_{g} \sum_{g' \in \text{fo}(g)} \lambda_{g,g'}(t_{a(g)} + d(g, w) - t_{a(g')}) + \tag{2.41}$$

$$\sum_{g \in \text{PO}} \lambda_g (t_{a(g)} - T_{\max}). \tag{2.42}$$

The associated Lagrangian dual problem is:

$$\max_{\lambda \geq 0} \left\{ \min_{t_a, w_{\min} \leq w \leq w_{\max}} \{L(w, t_a, \lambda)\} \right\}. \tag{2.43}$$

In this expression, the minimization over t_a is equal to $-\infty$ for certain values of λ. This

[14]A sufficient, but not necessary, condition for equality comes from Slater's condition:

1. x is continuous, and the domain of x is a convex, connected set.
2. f_i is convex $\forall i$.
3. g_j is linear $\forall j$.
4. There is an x such that $f_i < 0 \; \forall i$, and $x_{\min} < x < x_{\max}$.

[15]See [Lem01] for a good reference on Lagrangian relaxation.

can be seen by reorganizing the terms in (2.46) as:

$$\begin{aligned} L(w, t_a, \lambda) = &\sum_{g \in \mathcal{G}} \left(\sum_{g" \in \text{fo}(g)} \lambda_{g,g"} - \sum_{g' \in \text{fi}(g)} \lambda_{g',g} \right) \cdot t_{a(g)} + \\ &\sum_{g \in \mathcal{G}} p_g w_g + (\sum_{g \in \text{PO}} \lambda_g) \cdot T_{\max} + \\ &\sum_{g} \left(\sum_{g' \in \text{fo}(g)} \lambda_{g,g'} \right) d(g, w) \end{aligned} \tag{2.44}$$

If any of the coefficients of t_a are non-zero, then minimizing $L(w, t_a, \lambda)$ over w and t_a would yield $-\infty$, as the t_a are *free variables* and are *unrestricted* (the λ are fixed through this minimization. t_a with positive coefficients would have $t \to -\infty$ and negative coefficients would have $t \to \infty$. Thus, to avoid these cases, the λ is restricted to the *dual feasible* set that satisfies:

$$\sum_{g" \in \text{fo}(g)} \lambda_{g,g"} - \sum_{g' \in \text{fi}(g)} \lambda_{g',g} = 0, \tag{2.45}$$

thereby making all the coefficients of t_a in (2.44) equal to zero. For these dual feasible λ, the Lagrangian reduces into a weighted delay-power minimization:

$$\begin{aligned} L(w, t_a, \lambda) = &\sum_{g \in \mathcal{G}} p_g w_g + \\ &\sum_{g} \left(\sum_{g' \in \text{fo}(g)} \lambda_{g,g'} \right) d(g, w) + \\ &(\sum_{g \in \text{PO}} \lambda_g) \cdot T_{\max}. \end{aligned} \tag{2.46}$$

For continuous gate sizes and v_t, there are fast methods to solve this problem (see [RS08]) in linear time by cycling through each gate and choosing the gate width that minimizes the power and weighted delay combination while keeping all the other gate sizes fixed.

Lagrangian relaxation was also applied to discrete problems in [LH09, HHS11, OBH11]. In this case, idea is to solve the discrete version of the Lagrangian dual problem:

$$\max_{\lambda \geq 0} \left\{ \min_{t_a, \omega \in \text{CellOptions}} \{ L(\omega, t_a, \lambda) \} \right\}, \tag{2.47}$$

where the discrete cell options ω replace the continuous variables. This is done with two loops. An inner loop is used to minimize L over the ω, e.g. the cell options are chosen to minimize L. The outer loop adjusts λ to maximize L.

There are two challenges to this discrete Lagrangian relaxation. The first challenge is in the minimization over ω. While the problem is now unconstrained, the minimization is still combinatorial. [LH09] uses a reverse-topological dynamic programming method to perform this minimization, but this method is limited in that it does not account for slews, and the reconvergent fanout (see Section 2.3.7) limit the ability to apply dynamic programming. [OBH11] extracts a "Critical Tree" from the circuit before the dynamic programming is applied, thus avoiding the problems related to reconvergent fanout.

The other issue associated with discrete Lagrangian relaxation is updating of the multipliers λ. Traditionally, the multipliers are updated according to the sub-gradient (a generalized gradient).

$$\Delta\lambda = \frac{\partial}{\partial\lambda}\left(L(\omega, t_a, \lambda)\right). \tag{2.48}$$

λ next updated using two steps as

1. $\lambda = \lambda + \alpha\Delta\lambda$
2. λ is projected to satisfy (2.45) and $\lambda \geq 0$.

As the inner minimization over ω is likely to be suboptimal, the choice of λ and the method that it is updated affects the resulting solution. Furthermore, large changes in λ will cause the solutions to oscillate. [HHS11] finds an improved method to update the multipliers by storing the results of prior values λ for each gate, and the related arrival times at the gate. The stored results are then used to prevent overshoot when updating λ.

2.3.9 Slew targeting methods

Slew targets [Hel09] are also used for gate sizing. This works on the idea that the delay is a monotonically increasing function of the slew; thus improving the slew will also improve the delay. This is an interesting perspective on the timing closure problem to use the slews targeting as a proxy for timing closure.

Using slew targets is a well-known method used by designers and some commercial tools to perform optimizations. The work in [Hel09] formalizes the slew-based optimization for large scale timing closure. This method is also one of the few discrete sizing methods that deal with slew directly. Most other methods ignore slew, because it creates long range interactions between gates that are difficult to model (see Section 2.1). However, as slew has a large effect on the delays, ignoring slews may introduce sub-optimality into the design.

[Hel09] works by alternating between assigning slew targets, and optimizing the gates for the given slew targets. These slew targets are assigned as a function of the slacks. Gates with negative slack will have their slack targets increased, while gates with positive slacks will have their slack targets decreased.

Central to the slack adjustment is the concept of *local criticality*, $\mathrm{lc}(g)$. This is defined as the difference between the slacks of the input pins, and the minimum of the slacks of its fanins:

$$\mathrm{lc}(g) = \max_{g' \in \mathrm{fi}(g)} \{s(g) - s(g'), 0\}. \tag{2.49}$$

A value $\mathrm{lc}(g) = 0$ means that the gate is at least as critical as any of its fanins. However, a value of $\mathrm{lc}(g) > 0$ implies that there is a fanin gate that is more critical and thus the current gate does not contribute to the worst case slack.

Following this rationale, in Step 1 of Algorithm 5, gates with negative slack and $\mathrm{lc}(g) = 0$ have their slew targets decreased, or tightened, to reduce timing violations. All other gates have their slew targets increased, to help recover power. These increases or decreases are changed as a function of the local criticality $\mathrm{lc}(g)$, slack $s(g)$, damping factor Θ_k, and a constant γ, which is related to $\frac{\partial \tau(g)}{\partial s(g)}$. Θ_k starts with value ≈ 1, indicating that target values and estimates are used in the algorithm. As the algorithm progresses, $\Theta_k \to 1$, prompting the algorithm to use the true slew values in place of the estimates for increased accuracy.

```
$\Theta_k$ = step size & approximation parameter; lc = local criticality;
foreach $k \in \{1, ..., k_{\max}\}$ do
    Step 1: Update the slack targets;
    $\Theta_k = \frac{1}{\log(k+1)}$;
    foreach $g \in \mathcal{G}$ do
        $\mathrm{lc}(g) = \max_{g' \in \mathrm{fi}(g)}\{s(g) - s(g'), 0\}$;
        if $(s(g) < 0)$ & $(\mathrm{lc}(g) == 0)$ then
            $\Delta\tau_{\mathrm{target}} = -\min\left\{(\Theta_k \cdot \gamma \cdot |s_g|), \Delta\tau_{\max}^{\mathrm{target}}\right\}$;
        else
            $\mathrm{sm} = \max\{s(g), \mathrm{lc}(g)\}$;
            $\Delta\tau^{\mathrm{target}} = \min\left\{(\Theta_k \cdot \gamma \cdot |\mathrm{sm}|), \Delta\tau_{\max}^{\mathrm{target}}\right\}$;
        end
        $\tau_g^{\mathrm{target}} = \tau_g^{\mathrm{target}} + \Delta\tau_g^{\mathrm{target}}$;
        Project the slew target to feasible range $\tau_g^{\mathrm{target}} = \max\{\tau_g^{\mathrm{target}}, \Delta\tau_g^{\min}\}$;
        $\tau_g^{\mathrm{target}} = \min\{\tau_g^{\mathrm{target}}, \Delta\tau_g^{\max}\}$;
    end
    Step 2: Apply the slack targets;
    foreach $g \in \mathcal{G}$ in reverse topological order do
        foreach $\forall g' \in \mathrm{fi}(g)$ do
            Create input slew estimates $\hat{\tau}(g') = \Theta_k \cdot \tau_{g'}^{\mathrm{target}} - (1 - \Theta_k) \cdot s_{g'}$
        end
        Using the estimated input slews $\hat{\tau}(g')$ set $\omega_g$ as:
        the minimum power cell option $\omega$ that satisfies $\tau(g, \omega) > \tau_g^{\mathrm{target}}$
    end
    Step 3: Refine slew targets;
    foreach $g \in \mathcal{G}$ in topological order do
        $\hat{\tau}(g') = \Theta_k \cdot \tau_{g'}^{\mathrm{target}} - (1 - \Theta_k) \cdot s_{g'}$
        $s^p = \max_{\{g' |\ s_{g'} = s_g^{\mathrm{fi}}\}} \{\hat{\tau}(g')\}$
        $\tau_g^{\mathrm{target}} = \tau_g^{\mathrm{target}} + (\lambda) \cdot (s^p - \tau_g^{\mathrm{target}})$
    end
end
```

Algorithm 5: Slew targeting method in [Hel09].

Next, in Step 2 of the algorithm, the slew targets are applied to each gate in reverse topological order. This is done by first estimating the slews at the input of the gate, as the output slew depends on the input slews. Next, the minimum power cell option that satisfies the output slew target is chosen, and this process repeats for each gate in the design.

In Step 3 of the algorithm, the slew targets are refined. This is to reduce slack targets that are unnecessarily high, when "cells cannot be enlarged further, or to locally non-critical cells that cannot be downsized sufficiently because of too large successors" [Hel09]. To fix this, the slew target is reduced by a factor of λ times the difference between the estimated slew of the most critical input pin:

$$\tau_g^{\text{target}} = \tau_g^{\text{target}} + (\lambda) \cdot (s^p - \tau_g^{\text{target}}). \tag{2.50}$$

The author reports that this method is very fast; 5.8 million cells are sized within 2.5 hours on a 3.0 GHz Xeon Server. Furthermore, they report good results for timing closure with an average worst negative slack of 6% reduced to 2% after applying their algorithm.

2.3.10 Linear programming based assignment methods

Linear programming can also be used to assign gates to specific cell options [CK05, LG10, ALQ11]. This is in contrast to the continuous sizing methods in Section 2.2, where the variables represent the continuous gate sizes. In linear programming based assignment, each variable in this context is a binary variable, $y_{g\omega}$, where:

$$y_{g\omega} = \begin{cases} 1 & \text{if gate } g \text{ is assigned to cell } \omega \\ 0 & \text{otherwise.} \end{cases} \tag{2.51}$$

A value of $y_{g\omega} = 1$ means that gate g is implemented by the library cell ω, and a value of $y_{g\omega} = 1$ means that the gate g is implemented by a cell other than ω. These options ω can vary in sizes, threshold voltages, gate lengths, etc. The variables $y_{g\omega}$ are referred to as *assignment variables* as they *assign* gates to library cells. The power objective is written in terms of these assignment variables as:

$$\sum_{\forall g \in \mathcal{G}} \sum_{\omega \in \Omega_g} \Delta p(g, \omega; \omega_0) \cdot y_{g\omega} \tag{2.52}$$

where

$$\Delta p(g, \omega; \omega_0) = p(g, \omega) - p(g, \omega_0). \tag{2.53}$$

The $p(g, \omega)$ refers to the power consumption of gate g when implemented by the library cell ω (ω_0 is the current implementation of the gate). This power can be the leakage power or the dynamic power, or a weighted combination of the two. As the short circuit portion of the dynamic power is dependent on the input slew and output load, the current slew and load values are used to evaluate the Δp terms. Although this may introduce modeling errors if the gate's inputs change, this is still a good proxy for the actual power.

The delay constraints are written in the linear program using the *block-based formulation*:[16]

$$t_{a(g)} + \sum_{\omega \in \Omega_g} \Delta d(g, \omega; \omega_0) \cdot y_{g\omega} \leq t_{a(g')}, \quad \forall g' \in \text{fanout}(g)$$

$$0 \leq t_{a(g)}, \quad \forall g \tag{2.54}$$

$$t_{a(g)} \leq T_{\max}, \quad \forall g \in \text{PO}. \tag{2.55}$$

The $t_{a(g)}$ are variables that are used to model the arrival times at the input of gate g and the $\Delta d(g, \omega; \omega_0)$ term is the change in delay when the gate is changed from cell option ω_0 to ω. This can be computed by trying the cell option and computing the change in the arrival times. To improve accuracy, the change in slack may also be used to model downstream effects on the delay [LG10].

[16]See Section 2.3.1 for more information on the block-based delay formulation. For simplicity, the difference between the rise and fall times is omitted, as well as the difference in delay for different input-output paths in a gate. However, they can are commonly incorporated by adding separate rise and fall arrival-time variables and additional delay models for different the input-output paths in a gate.

Combining the power and delay terms results in the linear programming problem:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{\forall g \in \mathcal{G}} \sum_{\omega \in \text{CellOptions}(g)} \Delta p(g,\omega;\omega_0) \cdot y_{g\omega} \\
\text{subject to} \quad & t_{a(g)} + \\
& \textstyle\sum_{\omega \in \text{CellOptions}(g)} \Delta d(g,\omega;\omega_0) \cdot y_{g\omega} \le t_{a(g')} \\
& \qquad\qquad\qquad\qquad \forall g' \in \text{fanout}(g) \qquad (2.56) \\
& 0 \le t_{a(g)} \le T_{\max}, \forall g \in \mathcal{G} \\
& 0 \le y_{g\omega} \le 1, \forall g, \omega \in \text{CellOptions}(g) \\
& \textstyle\sum_{\omega \in \text{CellOptions}(g)} y_{g\omega} \le 1, \ \forall g.
\end{aligned}$$

Note that the assignment variable was originally intended to be *binary* – the variable should only be 0 or 1. However, to improve the runtime of this problem, it is relaxed so that the assignment variable is continuous between 0 and 1. [CK07] notes that the incremental benefit of solving the $\{0,1\}$ integer linear program is negligible, and may sometimes provide worse results.

Once (2.56) is solved, then the next step is to map the $y_{g\omega}$ into gate cell changes. In [CK05], the authors apply any $y_{g\omega} > .99$. However, due to the delay interactions, this may cause timing violations. They correct these timing violations by running a linear programming assignment for minimizing timing:

$$\begin{aligned}
\text{minimize} \quad & \max\{t_{a(\text{output})}, T_{\max}\} + \ldots \\
& \epsilon \textstyle\sum_{\forall g \in \mathcal{G}} \sum_{\omega \in \Omega_g} \Delta p(g,\omega;\omega_0) \cdot y_{g\omega} \\
\text{subject to} \quad & t_{a(g)} + \textstyle\sum_{\omega \in \Omega_g} \Delta d(g,\omega;\omega_0) \cdot y_{g\omega} \quad \ldots \\
& \qquad\qquad\qquad\qquad \le t_{a(g')} \quad \forall g' \in \text{fanout}(g) \qquad (2.57) \\
& 0 \le t_{a(g)} \le t_{a(\text{output})}, \qquad \forall g \in \mathcal{G} \\
& 0 \le y_{g\omega} \le 1, \qquad \forall g, \omega \in \Omega_g \\
& \textstyle\sum_{\omega \in \Omega_g} y_{g\omega} \le 1, \qquad \forall g
\end{aligned}$$

where ϵ is a small constant used to give more weight to the timing objective. In the minimizing timing case, values of $y_{g\omega} > .01$ are applied.

The main limitations with this method concern the delay model. This model does not account for the interactions in the delays between gates in two ways. To understand the first limitation, suppose that an input gate to gate g has decreased in size, diminishing its ability to drive its output gates. If the size of gate g is then increased, then the resulting delay improvement will be worse than if the input gate had not changed. This is because the input gate is less able to handle the increase in the output capacitance from an increase in the gate size of g.

The other limitation is the interaction between the transition times (slews). Decreasing the slew of an input gate will improve the delays downstream, while increasing the slew will have the opposite effect on the delay. Thus, any gate change may affect the $\Delta d(g, \omega; \omega_0)$ downstream from it.

Due to these limitations, [LG10] uses this approach for incremental gate sizing. The model has a better accuracy when the number of changes is not too large, thus it is well suited for finding a small number of gate changes that can be used to implement an Engineering Change Order (ECO).

2.3.11 Summary of methods

There is a wide diversity of methods used to tackle the discrete gate sizing and threshold assignment problem. There are mathematical programming methods, dynamic programming methods, greedy methods, and heuristics that try to find the best tradeoff between power and delay. The methods presented in this chapter are summarized in Table 2.2. This chart indicates whether the method is well-suited (✓), applicable ($\star$), or not applicable (–) in a given context.

Pre-layout means that the method is well-suited for optimization before the layout is fixed, such as after synthesis. Generally, this means that method can make large changes to the design and inaccuracies in the delay modeling can be tolerated. In contrast, *post-layout* indicates that the method can be used incrementally when large changes to the design are to be avoided. Post-layout methods must be able to modify the current design, rather than start from scratch, and being close to tape-out, accurate delay modeling is

	pre layout	post layout	power optimization	timing closure optimization
Score and Rank	✓	✓	✓	✓
Slack and Delay Budgeting	✓	⋆	✓	–
Continuous sizing based	✓	–	✓	✓
Dynamic Programming	✓	–	✓	✓
Lagrangian relaxation	✓	–	✓	✓
Slew targeting	⋆	✓	–	✓
LP-based assignment	⋆	✓	✓	✓

Table 2.2: Summary of discrete gate sizing and threshold voltage assignment methods.

important. Generally, methods that are not incremental do not work well in post-layout settings.

Power optimization indicates that the method can be used to minimize power given a timing constraint. The *timing closure optimization* column indicates that the method can be used to improve timing, whether to find a minimum delay design, or to meet a timing constraint.

CHAPTER 3

Benchmarking Discrete Gate Sizing and Threshold Voltage Assignment Methods

The largest barrier in advancing the field of gate sizing and threshold voltage assignment methods is the difficulty in comparing competing methods. Recent publications in gate sizing do not use common benchmarks, setups, or technologies; thus the question of whether recent research (see, for example [LH09, ALQ11, OBH11]) provides an improvement over current methods is questionable. The essential questions such as, "What is the best discrete gate sizing method," and "What are the deficiencies of current methods," are not understood, making it impossible to quantify when a significant improvement to the field is made. This chapter will explain challenges behind comparing competing methods, and will explain attempts to benchmark methods.

Surveying the literature does not provide insight into the state-of-the-art. The comparisons reported in the gate sizing and threshold voltage assignment literature are difficult to navigate. Most v_t assignment papers ([SEO99, WRK00, SEO02, NDO03, SSB04a]) report improvements over no v_t assignment. Similarly, [GKS04] shows improvements over no gate-length biasing, and the improvements are those gained by adding additional threshold voltages and gate-length variants. A summary of the reported results are shown in Table 3.1. However, while this does show that a few methods may be better than others, it does not show what the best method is.

An interesting set of comparisons is found in [WD09]. A branch-and-bound solver is implemented, which provides the optimal sizing solutions for ISCAS '85 [HYH99] bench-

		improvement			
method	comparison	min	max	avg	
DP+LP [LH10]	Budgeting [NDO03]	1%	31%	21%	w, v_{t}
LSH [HKH09]	Greedy [Cou97]	9%	31%	18%	w
Continuous based [SSS05]	Greedy [SEO02]	6%	62%	31%	w, v_{t}
Global sizing [Cou97]	Greedy [SSF88]	5.3%	57.3%	11%	w
Continuous based [CSH95]	Greedy [LMK90]	0%	12%	5%	w
Continuous based [RHC08]	Com.[1]	1%	39%	11%	w
LP assignment [CK05]	Com.[1]	-3.6%	44.9%	15.3%	w

[1] These methods were compared against the results of a commercial synthesis tool.

Table 3.1: Improvements reported in literature.

marks and ISCAS '89 [BBK89] benchmarks. For these benchmarks, the greedy method achieves between 2% to 35% of the optimum, with an average suboptimality of 11%. A simulated-annealing method is also compared, which achieves between .5% to 28% of the optimum, with an average suboptimality of 7%. The large range between min and max indicates that the performance of an algorithm is highly benchmark dependent, and while the greedy method provides reasonable performance for a simple algorithm, the large maximum suboptimality shows that there is room for improved algorithms.

Comparisons between methods are also complicated by the importance of the experimental setup. In contrast to the placement benchmarks, which have well-defined experimental setups, the results in Table 3.1 vary heavily in their technology, wire models, synthesis options, range and granularity of their gate sizes, types of logic cells available, and timing constraints. For example, a small library granularity tests a method's ability to handle only a few cells that are available of each type. Benchmarks without sequential elements (combinatorial benchmarks) generally have different distributions of critical paths than sequential benchmarks. Tighter timing constraints may test a method's ability to create timing feasible designs. These make it unfair to compare the results between papers.

Compounding the complexity is the context in which these methods are used:

- **Post-synthesis**: after the design is mapped to logic blocks.
- **Post-placement**: after the locations of the cells are known, along with approximate information on the lengths of connecting wires.
- **Post-layout**: after routing and placement are completed. At this step, the layout, along with information on wire parasitics, is known.

Each context has different flexibilities and accuracy requirements. For example, post-synthesis gate sizing and threshold voltage assignment has the most flexibility, as the gates are not yet placed (given locations), and also have the loosest timing accuracy requirement. In contrast, post-layout gate sizing and threshold voltage assignment has the least flexibility, as each change in the cell may require moving and/or rerouting other cells. Furthermore, there is a high accuracy requirement; after this stage, the design must be timing feasible.

The recent gate sizing and threshold voltage assignment contest at the International Symposium of Physical Design [OAA12] attempts to answer this question by providing a common framework for gate sizing benchmarking. However, the results from the contest showed that there is no "best" method. Of the top three methods that were submitted, no method performs the best on all benchmark suites. The first place method did not perform consistently well– on one benchmark it performed 75% worse than the best result, with the other methods having a similar range. This is in light of the fact that these methods incorporated a wide range of concepts, from network flow, dynamic programming, simulated annealing, Lagrangian relaxation, and hybrid approaches. This suggests that there is more work needed to develop the state-of-the-art.

This chapter focuses on making progress in determining the state-of-the-art. This chapter considers the different factors involved in comparing gate sizing and threshold voltage assignment methods, and provides comparative results.

3.1 The standard cell library

The standard cell library contains logic cells such as inverters, ands, not-ands (nands), and x-ors that implement Boolean logic functions. There are also sets of sequential cells such as flip-flops, latches, and their variants with capabilities for setting, resetting and reading in scan-chains. These sequential cells provide *memory*, allowing pipe-lining, state machines, and a memory for computations. Lastly, there are utility cells, such as filler cells, antenna cells, and buffer cells, which are tools to help with the physical implementation of the design.

The library generally provides several gate options for each logic function. Each of the options are logically equivalent – they implement the same boolean function – but have varying electrical characteristics, due to differences in their gate lengths, widths, PMOS-NMOS width ratios and threshold voltages v_{t}. These alternative options can be used to optimize the design. For example, critical paths can be sped up by swapping high-v_{t} cells by low-v_{t} cells, and gates with fanout violations can be fixed using alternatives with larger-transistor widths and smaller effective resistances at the gate outputs.

Two library files that are used for sizing and threshold voltage assignment are:

1. **Physical library information.**
   (usually expressed in Library Exchange Format (LEF)):
   - Cell information: dimensions of the cell and locations of the pins.
   - Interconnect information: dimensions, pitch, capacitance, and resistances for each metal layer.
   - Via information: dimensions, resistance, and layers that are connected.

2. **Timing library information.**

   (usually expressed in Liberty Format):

   - Library characterization information: temperature, voltage and process.
   - Parameters used in the library: slew thresholds, input thresholds, output thresholds, and measurement units.
   - Cell information: delays, area, logic function, short-circuit power, switching power, and leakage power. For flip-flops the hold and setup time requirements are also given.
   - Cell pin information: capacitances, maximum loads.

The geometry information in the LEF file is used for placement and routing. This information tells the program how to create standard cell rows for floorplanning, and the dimensions of each cell for placement. Next, the pin locations, interconnect geometries, and via dimensions are used for routing, and once routing is complete, the capacitance and resistance information is used to extract information about the wire parasitics.

The timing and power information from the Liberty file is used for the timing and power analysis of the design. The timing information is used in conjunction with the wire parasitic information to create delay estimates and power estimates with interconnect loading modeling. Understanding the basics of these models is helpful in understanding the operation of the gate sizing methods, and will be described below.

3.1.1 Delay

The relationship between the gate size, length and v_{t} as a function of the delay can be approximated using an *α-power law model* [SN90] in 1.5. For a 45nm commercial library, a sample of the effects of the delays is given in Figure 3.1[1]. α for this library is approximately

[1]In this table, the rise and fall transition thresholds are 20% and 80%, and the delay threshold is 50%. For example, the rise transition time is the time measured from when the signal passes 20% of the supply voltage, to when it reaches 80% of the voltage. The pull-up delay is the time from when the signal reaches the input of the gate (at 50%) to the time that the output pin voltage reaches 50%.

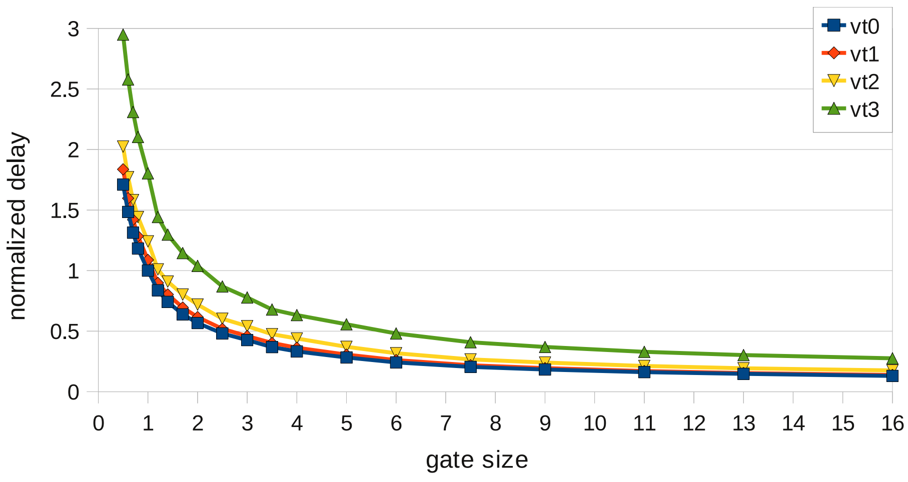


Figure 3.1: Normalized inverter gate delays for a commercial 45nm library.

1.4.

Notice that the gate widths in this library are geometrically spaced – there are 10 gates sized 2.5 and under, and there are 10 gates between 3 and 16. This is because of the inverse relationship between the gate delay and the gate width: reducing the delay by a factor of 2 requires the gate width to double in size.

The gates in the library are spaced to cover a wide range of delays. Increasing to the next gate size will lead to a 10 to 15% reduction in the output delay, and the difference between the delay of the minimum size and maximum size is 12X. In the case of threshold voltages, changing from $v_{\mathrm{t}0}$ to $v_{\mathrm{t}1}$ or $v_{\mathrm{t}1}$ to $v_{\mathrm{t}2}$ increases the delay by about 10%. However, increasing the threshold voltage from $v_{\mathrm{t}2}$ to $v_{\mathrm{t}3}$ increases the delay by the larger margin of 35%.

Increasing the gate widths also increases the capacitances of its input pins proportional to $W \cdot L$. For example, Figure 3.2 shows that this capacitance is roughly linear in the size of the gate. Increasing the size will therefore cause the delay of the fanin gates to increase, and may have a cumulative negative effect on the delay.

Changing the threshold voltages has a milder effect on the input capacitances. Figure 3.2 shows that the capacitance is a weak function of the threshold voltage. Moving

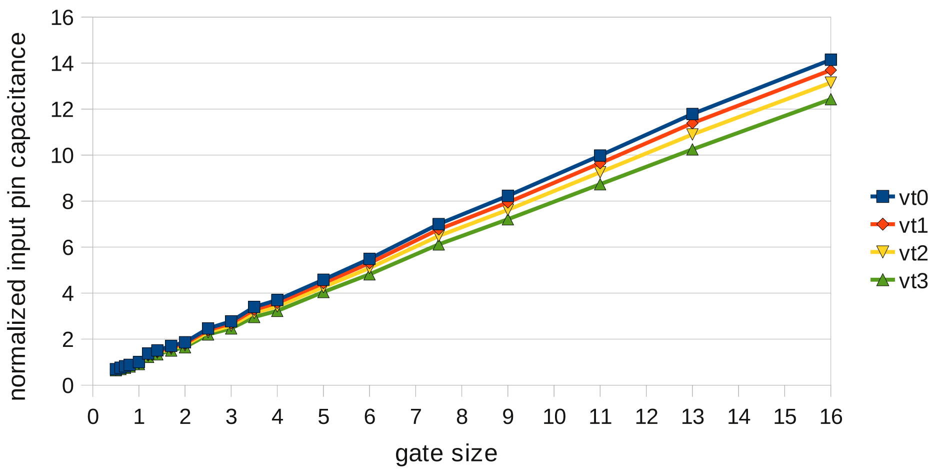


Figure 3.2: Normalized inverter *input* pin capacitances in a commercial 45nm library.

from $v_{\mathrm{t}0}$ to $v_{\mathrm{t}1}$ gives a 3% decrease in the capacitance, while moving from $v_{\mathrm{t}1}$ to $v_{\mathrm{t}2}$ gives a 4.5% decrease in the capacitance. Thus, although the input capacitance changes, the overall delay effect is very likely to be a decrease. In contrast, increasing the gate size may not decrease the overall delay – the increasing input capacitance may cause a larger increase at the fanins than the decrease in the output.

3.1.2 Delay models

The most accurate gate delay models are in the form of spice level netlists. These netlists contain transistor level information that can be used by transistor-level timing methods for static timing analysis[2]. These methods use fast Spice-like simulations without the use of input vectors to create timing estimates that are comparable to Spice simulations.

However, only a small percentage of designs are able to utilize transistor-level timing, as most designs are too large for transistor level simulation. Furthermore, these methods are too slow to be used in the process of optimizing large designs.

The library modeling standard that is used is the Synopsys Liberty modeling for-

[2]For example, the Synopsys NanoTime [Synb]

mat [Syna][3]. Currently, Liberty uses two different methods, the Nonlinear Delay Models (NLDM) and the current-source models, which include the Synopsys Composite Current Source (CCS) and the Cadence Effective Current Source Model (ECSM). The NLDM format is a legacy format introduced in 1992, and the CCS/ECSM methods are newer methods from 2004 that were introduced to improve accuracy.

3.1.2.1 The Nonlinear Delay Model (NLDM)

The NLDM has long been the de facto standard for static timing analysis methods, though it is slowly being replaced by the newer current-source methods. It utilizes tables that store rise and fall times, and output rise and fall transitions (slews), as a function of the input transition and the output capacitance load. The model also contains the capacitances for all of the input pins in the library.

The parameters that define the cell rise, fall, and transition times are given at the top of the Liberty file. The most common numbers in modern libraries are a 50% rise/fall threshold and a 20% to 80% or 30% to 70% slew rate.

A table for rise delay, fall delay, rise transition, and fall transition is provided for each cell in the library. Each of these tables is indexed by the input transition time, and the output capacitance. Table 3.2 provides a small example of the cell rise times for a nominal sized inverter gate. The delays are increasing functions of the input transition and the output load, and both the input transition and output capacitance have a large effect on the output delay.

There are also tables for the clock hold time and setup time of sequential elements. The hold and setup times are given for rise and fall transitions, as a function of the input transition (slew), and the clock slew. Table 3.3 shows a small example of the setup time for a nominal sized D-flip flop.

[3]See http://www.opensourceliberty.org for information on the format, technical papers, and an open source parser. Also, see [Tri08] for the motivation for current-source methods.

		output capacitance (fF)						
		0.4	0.8	1.6	3.2	6.4	12.8	25.6
output rise delay								
input slew	0.0075	0.018	0.022	0.032	0.051	0.089	0.164	0.315
input slew	0.0375	0.032	0.038	0.048	0.066	0.104	0.179	0.330
input slew	0.1500	0.059	0.069	0.087	0.117	0.164	0.239	0.388
input slew	0.6000	0.129	0.145	0.174	0.224	0.305	0.433	0.628

output rise transition								
input slew	0.0075	0.012	0.016	0.025	0.043	0.079	0.151	0.294
input slew	0.0375	0.018	0.021	0.027	0.043	0.079	0.151	0.294
input slew	0.1500	0.036	0.042	0.052	0.067	0.092	0.152	0.294
input slew	0.6000	0.095	0.100	0.114	0.139	0.183	0.252	0.353

Table 3.2: Rise times (delays) and transitions (slews) in ns for a nominal inverter gate as a function of the input transition time and the output capacitance.

		clock transition (ns)						
		0.005	0.0141	0.0281	0.056	0.113	0.225	0.450
setup time								
input slew	0.0075	0.038	0.033	0.028	0.022	0.017	0.017	0.028
	0.0375	0.050	0.046	0.041	0.035	0.030	0.031	0.041
	0.1500	0.087	0.083	0.078	0.071	0.067	0.067	0.075
	0.6000	0.181	0.176	0.170	0.162	0.155	0.153	0.159

hold time								
input slew	0.0075	0.009	0.013	0.018	0.022	0.023	0.018	0.002
	0.0375	0.045	0.048	0.051	0.051	0.047	0.031	-0.004
	0.1500	0.300	0.304	0.308	0.310	0.308	0.298	0.265
	0.6000	1.383	1.386	1.391	1.395	1.398	1.388	1.364

Table 3.3: Setup times for a rising input to a nominal D-Flip Flop as a function of the input transition and the clock transition.

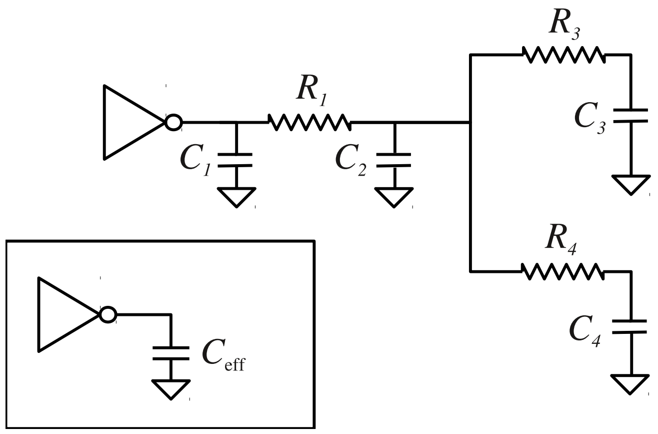


Figure 3.3: Example of effective capacitance: an inverter driving an RC tree. The effective capacitance is less than the total capacitance of the tree: $C_{\text{eff}} < \sum_{i=1}^{4} C_i$. Adapted from [Sap04].

One limitation of this model is that the load capacitance is not well-defined at the output of a gate. Gates usually drive other gates through a series of interconnect that have their own intrinsic resistances. However, this has the effect of "resistive shielding" as the resistance hides some of the capacitance from the gate. Essentially, this resistance will cause the capacitances closer to the gate to be charged faster than the ones downstream from the gate. Thus, the waveform at the output of the gate will be faster than having the total capacitance present at the output of the gate.

STA timers account for the resistive shielding by computing the *effective capacitance* of the gate (see [QPP94, MM98, BC09, FA08]) to predict accurate delays. Figure 3.3 shows an example of this resistive shielding effect. Due to the resistors R_1-R_4, the effective capacitance seen at the output of the inverter is strictly less than the total capacitance of the tree. As the resistances increase, the effective capacitance decreases.

Another limitation is the one-parameter model of the input and output waveforms [FA08]. The input and output waveforms are characterized only by their transition time. This limits the amount of flexibility and introduces modeling errors. However, many tools will re-model the NLDM library to approximate a current-source model (see, for example, [KLX06]). The NLDM tables are used to estimate the input-output current and voltage relationships. This helps to mitigate the limitations of the NLDM.

In the context of gate sizing and threshold voltage assignment, the NLDM has been the standard timing information for these algorithms for over a decade, and has heavily influenced these algorithms. Most algorithms estimate the gate delays to be functions of the input slew and output capacitance.

3.1.2.2 Current source models

Current source models were introduced to increase the accuracy of the gate models [Tri08, FA08, KTK08]. They use enhanced driver models, which model the output of the gates as function of time, and receiver models, which model both the loading and the Miller coupling effects. These libraries are generally much larger than the NLDM libraries, as they need to store voltage or current information over multiple time points, for each input load and slew value.

The driver models provide an output waveform for each input slew and output capacitance combination. In the CCS models, the current is given as a function of time, and in the ECSM model, the voltage is given as a function of time. This reduces the error of the timing simulation to approximately 2% for a single stage, when compared to HSPICE [Tri08].

The receiver models are used to capture the nonlinear characteristics of the receiver [Tri08, KTK08]. These models use two tables that give the receiver capacitance as a function of the input transition and the output load. The first table gives the input pin capacitance for the first half of the input waveform, and the second table gives the capacitance for the second half of the waveform. The capacitances vary greatly; for example the nominal sized inverter in the Nangate library has capacitances that range from .43fF to .78fF.

3.2 Power tradeoffs

Power is an important concern in gate sizing and threshold voltage assignment [CGL10, ITR]. Power dissipation has increased super-linearly with decrease of transistor dimensions, and the power consumption is increasing faster than the ability to remove the heat from the devices. Furthermore, there is an increasing focus on low power and mobile applications.

Power consumption is divided into static power and dynamic power. The static power is composed of the leakage power, while the dynamic power is composed of the power used to switch the voltages within the gates. Recently, the focus on power has been shifting from dynamic power to static power. At the 45nm technology node, static power is comparable to the dynamic power consumption, and the static power consumption will continue to increase at a much faster rate than the dynamic power [Nar05].

Dynamic power. Dynamic power is related to the energy used to switch logic states. This requires charging and discharging the interconnect and gates ($p_{\text{dynamic(cap)}}$ in (1.1)) and also includes the internal energy that is lost in the process of switching ($p_{\text{dynamic(sc)}}$ in (1.3)). The total dynamic power is thus

$$p_{\text{dynamic}} = p_{\text{dynamic(cap)}} + p_{\text{dynamic(sc)}}. \tag{3.1}$$

Figure 3.4 gives the relative internal power dissipation for the inverter gates. The internal power refers to the power that is dissipated inside the boundaries of the cell itself, and does not include the power dissipated by charging the output load. The figure shows that the switching power is indeed a linear function of the size. When comparing threshold voltages, the $v_{\text{t}0}$ gate has the highest internal power dissipation. Switching from $v_{\text{t}0}$ to $v_{\text{t}1}$ or from $v_{\text{t}1}$ to $v_{\text{t}2}$ gives an internal power savings of about 20%. The reduction from $v_{\text{t}2}$ to $v_{\text{t}3}$ is smaller, at 10%.

Leakage power. The leakage power comes from the current that flows when there is no change in the inputs. It is also referred to as the *static* power to contrast it with the

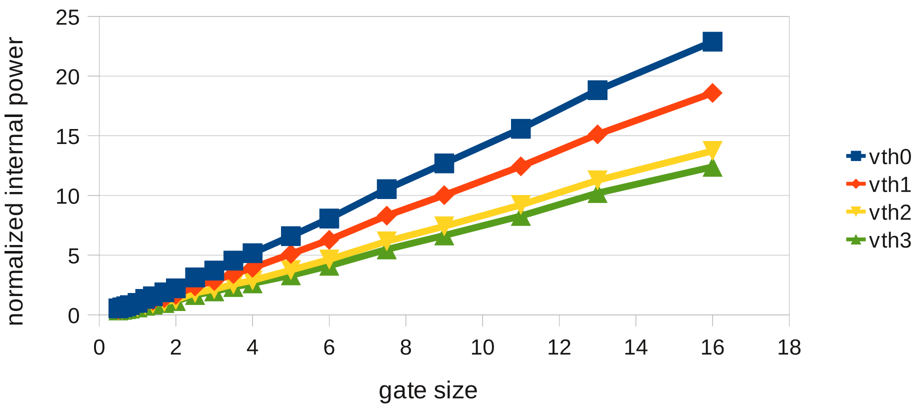


Figure 3.4: Normalized inverter gate *internal* powers a commercial 45nm library.

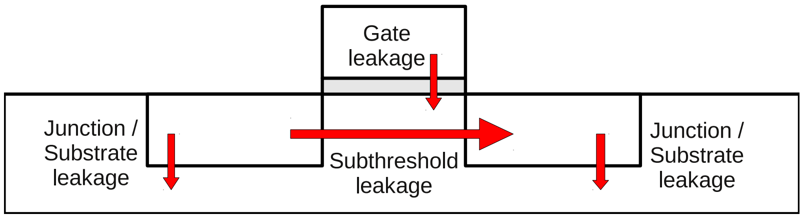


Figure 3.5: Sources of leakage power.

dynamic power. Leakage became a significant source of power dissipation from the 130nm technology node and at 45nm it is comparable to the dynamic power dissipation.

There are several sources of leakage power (see Figure 3.5). There is gate leakage from the gate to the substrate due to tunneling effects, leakage from the source to the drain due to sub-threshold conduction, and leakage from the wells to the substrate.

The main source of leakage is due to sub-threshold conduction from the source to the drain. This happens because the transistor is not fully "OFF", and is governed by (1.4) [SSK87].

The leakage power is a strong function of the gate inputs for multi-input gates. This is because the inputs affect the internal voltages of stacked transistors. For example,

state	leakage (nW)
A1 & A2 & A3	19.78
A1 & A2 & !A3	14.72
A1 & !A2 & A3	7.41
A1 & !A2 & !A3	.48
!A1 & A2 & A3	15.59
!A1 & A2 & !A3	1.63
!A1 & !A2 & A3	7.15
!A1 & !A2 & !A3	1.27

Table 3.4: Leakage power for a NAND3 cell as a function of its inputs.

Table 3.4 shows the leakages for the nominal sized NAND3 cell as a function of the input. The leakage values range from .48 nW to 19.78 nW– a range of over 40X. This range causes *significant* modeling error, as it is very difficult to predict the probabilities for each of the input states.

The exponential relationship between the leakage and the threshold voltage is the motivator for multiple-v_t cells. Figure 3.6 shows the relative change in the leakage power as a function of the size (note that the powers are plotted on a logarithmic scale). While the relationship between leakage and gate size are linear, the leakage is exponentially dependent on the threshold voltage. Thus, high v_t cells enable order of magnitude reductions that are not available using sizing alone.

Gate length biasing [GKS04] is another tool for optimization. The idea is to increase the lengths of gates with positive slack. The increase in gate length will have linear increases on the delay, but near exponential effects on the leakage power.

Figure 3.7 shows how the delay and leakage power change as a function of the gate-length bias. Increasing the gate-length by 5% gives an approximately 35% reduction in the

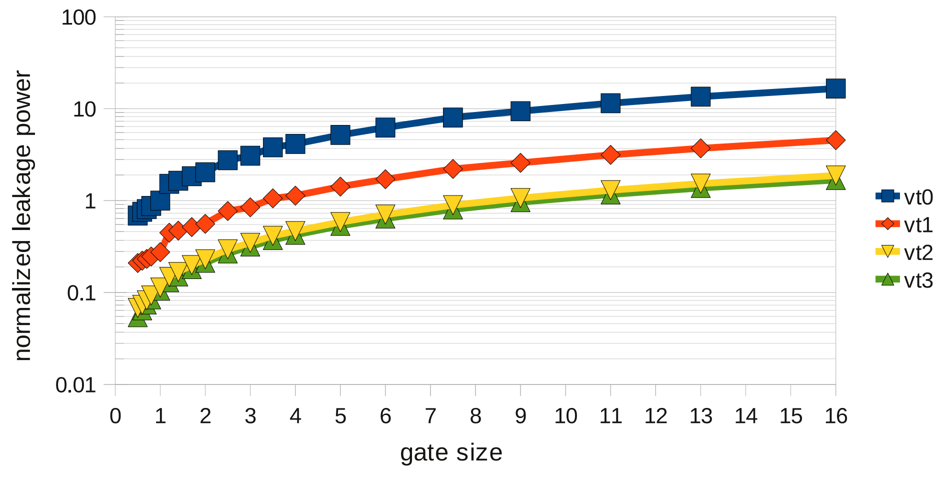


Figure 3.6: Normalized inverter gate *leakage* powers in a commercial 45nm library.

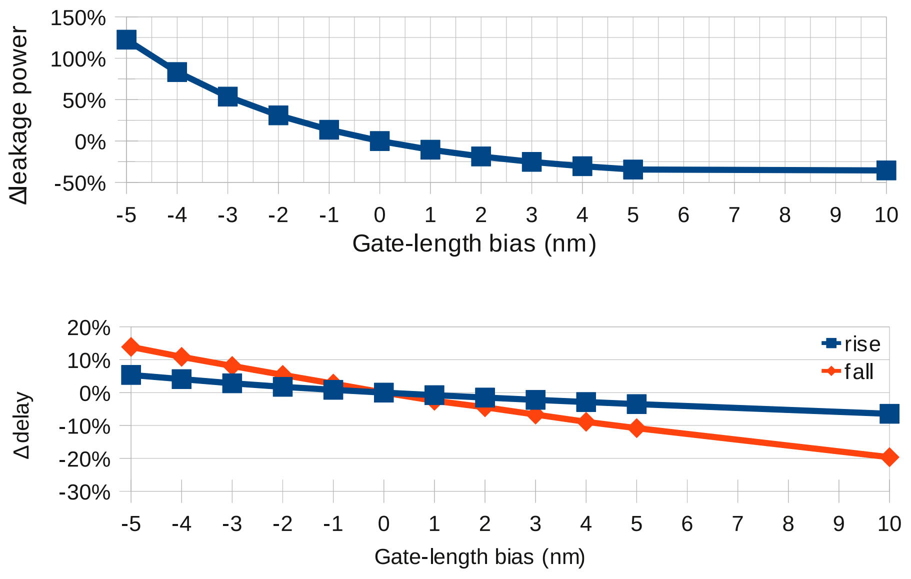


Figure 3.7: Plot of (a) the gate-length bias vs. power and (b) gate-length bias vs. delay. The data is for a nominal sized inverter in a commercial 45nm process, with a nominal gate length of 40nm.

when	value
!A1 & !A2	1017.82
!A1 & A2	7340.61
A1 & !A2	1204.57
A1 & A2	10286.09

Table 3.5: Leakage powers for the NAND2_X1 gate. A1 and A2 are the two input pins.

leakage power while creating a 3 to 11% increase in the delay. Adjusting the gate-lengths also provides a finer resolution than the v_t-assignment.

3.2.1 Power models

All Liberty models provide leakage power information in the maximum or average leakage power for the cell, and may additionally provide the leakage power for the different input combinations. For example, Table 3.5 shows the leakage information for the NAND2_X1 gate from the 45nm Nangate Library as a function of the inputs.

The Nonlinear Power Model (NLPM) is the power analog to the NLDM. This model provides a series of tables for the rise and fall *internal power* of the cell, as a function of the input transition and the output load. This internal power model gives the internal energy used at each transition, but *excludes* the power that is needed to charge the output load.

The most important factor in gate sizing and threshold voltage assignment experiments is the standard cell library. This is an even larger factor than the benchmark itself; the performance of an algorithm will vary on the same benchmark under two different standard cell libraries.

The two major qualities of a library that impact the gate sizing and threshold voltage assignment experiment are the range of the cells and the number of the cells. The range of the cells determines the range for the method. Gates that have a small range, such as those with cells of 1X to 2X of the minimum gate width, can provide only small impacts, as the sizing method can increase gate sizes to just 2X for large fanout loads. Similarly the range in the threshold voltages also plays a big role in the performance of the algorithm. Most cell libraries provide large ranges for popular gates and those used for buffering large fanouts (such as inverters and buffers). However, less popular gates (such as the AND-OR-INVERT) may have a small range, and the flexibility at these cells is very limited.

The granularity of a library impacts different algorithms differently. A library that is *dense*, with many gate sizes and threshold voltages for each gate footprint, will generally work better on methods that are derived from continuous sizing methods. This is because the increased granularity reduces the penalty for rounding and snapping – rounding from 5.2X to 5X is much better than rounding down to 4X.

In contrast, other methods may fare poorer on dense libraries. Dynamic programming based algorithms may suffer from significant slowdowns due to the increased number of cell options. Greedy methods, which are heuristics, may fare poorer than their continuous sizing based counterparts. The granularity question, whether the library is sparse or dense, may cause one method to perform better than another.

However, there is no *typical* library density or range. One 65nm commercial library provides 11 versions of the buffer and inverter cells, and 4 versions of other cells. The Nangate Library [NANb], which is used for testing and research, provides 5 versions of the buffer and inverter cells, and 3 versions of other cells. In contrast, another 45nm library provides 20 versions of the buffer and inverter cells (ranging from .5X to 16X), and between 5 to 11 versions of other cells (ranging from .5X to 6X). In addition, there are 3 threshold voltage options for each cell. Therefore, it is important to understand the library that will be targeted before developing a gate sizing and threshold voltage assignment method.

	$T_{\max}$	$\lvert\mathcal{G}\rvert$	$T_{\max}$	$\lvert\mathcal{G}\rvert$	$T_{\max}$	$\lvert\mathcal{G}\rvert$	$\%_{\Delta T_{\max}}$	$\%_{\Delta\lvert\mathcal{G}\rvert}$
c432	0.431	118	0.494	137	0.619	93	43.6%	47.3%
c499	0.477	586	0.612	322	0.881	189	84.7%	210.1%
c880	0.408	346	0.562	237	0.871	212	113.5%	63.2%
c1355	0.473	598	0.599	356	0.850	222	79.7%	169.4%
c1908	0.802	676	0.863	571	0.990	533	23.4%	26.8%
c2670	0.610	1041	0.787	791	1.140	748	86.9%	39.2%
c3540	1.096	1615	1.289	1359	1.675	1107	52.8%	45.9%
c5315	1.090	2019	1.226	1867	1.498	1766	37.4%	14.3%
c6288	1.845	3089	4.127	1656	8.692	1780	371.1%	86.5%
c7552	0.866	2894	0.977	2640	1.200	2481	38.6%	16.6%

Table 3.6: Effect of synthesis on the ISCAS '85 benchmarks [HYH99] (adapted from [LG10]). All times are in ns.

3.2.2 Benchmarks and synthesis options

The type of benchmark and the synthesis options are the second most important factor in comparing gate sizing and threshold voltage assignment methods. The benchmarks are usually designs that have interesting sizes and topologies. For example, the ISCAS '85 benchmarks were combinatorial designs representing controllers, arithmetic logic units, and error correcting circuits [HYH99]. The ISCAS '89 benchmarks are sequential designs[4] and are sections of real designs, traffic light controllers, and digital fractional multipliers [BBK89].

At a high level, these benchmarks differ by the number of gates, the depth of their logic, the delay distributions of their paths, and the average number of fanouts and fanins for each of the gates. These metrics, however, cannot predict what kind of method will work, or how difficult the design will be to size.

[4]e.g., these designs include flip-flops.

A complicating factor is the effect of the synthesis tool on the design. Table 3.6 shows the effect of different timing constraints on the resulting netlist. The leftmost columns represent synthesis for minimum delay, and the rightmost columns represent synthesis for minimum power. The delay spread between fastest and slowest synthesis outputs vary from 0.188ns (c432) to 6.847ns (c6288), depending on the benchmark. The size of the resulting netlists vary from 25 gates (c432) to 1309 gates (c6288), which means that the number of gates can double, depending on the design!

The difference between the synthesized netlists is the amount of buffering, and the type of gates that are used. When the timing constraint is generous, more complex gates are used, such as full adders, half-adders, or-and-inverts, and-or-inverts, etc. These gates are more power and area efficient than their lower level representations, but they do not offer the kind of flexibility that can be used to improve the delay. Conversely, tighter timing constraints will result in designs with many more inverters and buffers, and also more basic building blocks, such as nands.

An odd consequence of this effect is that synthesizing designs for minimum delay provides the largest examples for gate sizing and threshold voltage assignment. These designs not only have the most number of gates, but the most gates that have multiple cell options. For example, while there may only be 1 full-adder size, there might be several inverter and nand cell sizes to play with. However, the effect of the synthesis timing constraint on the end design after place and route and additional optimizations is not clear. Table 3.7 shows the effect of a variety timing constraints used during synthesis given in [JK10] on the final design. A tighter timing constraint at synthesis does not always lead to a faster design, nor is the relationship is not well-behaved.

3.3 Post-layout considerations

Post-layout discrete gate sizing and threshold voltage assignment are done after cell placement, interconnect routing, and clock tree synthesis (see [LG10]). They are performed

Timing Constraint used @ synthesis	**With a 2.0ns Clock**			
	After Place and Route Optimizations		@ sign-off	
	$T_{\max}$	slack	$T_{\max}$	slack
1.60	1.829	0.171	2.249	-0.249
1.80	1.912	0.088	2.196	-0.196
1.90	1.888	0.112	2.195	-0.195
1.95	1.926	0.074	2.449	-0.449
2.00	1.912	0.088	2.252	-0.252
2.10	1.912	0.088	2.214	-0.214
2.20	1.88	0.120	2.281	-0.281
2.40	1.838	0.162	2.081	-0.081

Table 3.7: Effect of different timing constraints used during synthesis on the final layout (adapted from [JK10]) for an AES encryption circuit from [OPE]. After synthesis, a timing constraint of 2.0ns is used to perform placement, routing and incremental timing optimizations.

to eliminate violations and modify the design to meet specifications. For example, they may be performed to fix timing violations, signal integrity issues, maximum capacitance violations, or hold time violations [HO01], as well as to recover power with accurate timing models.

3.3.1 Post-layout timing considerations

As placement and routing have been performed, post-layout timing estimates may utilize (see Section 2.1):

1. information about the interconnect, and interconnect parasitics;
2. the clock distribution tree with its associated skews;
3. the metal fills, effects of lithography on the gate length, and stress parameters;
4. complex timing models including variability and crosstalk effects.

These models generally do not have convexity properties, nor do they have simple analytical expressions. Thus, commercial timers use detailed approximations to ensure that the resulting delays are accurate.

Timing engines used at the place and route stage use a faster (albeit less accurate) timing engine than that used at sign-off. Furthermore, the parasitic extraction used at place and route generally is not as detailed as the sign-off parasitic extraction [JK10]. This mismatch may result in iterating between design modification and sign-off, as the timer used to modify the design is often different from the timer used at sign-off.

An example of this mismatch is shown in Figure 3.8. Though the timing estimates correlate well, the values differ by a substantial margin. The error is as large as .2ns, and in many cases the place and route timer underestimates the delay, which may lead to multiple cycles of validation. This deviation increases the difficulty of the timing closure problem, as the tool used for cell optimization, placing and routing does not match with the tool that is used for verification. Thus, iterations of design changes in the form of

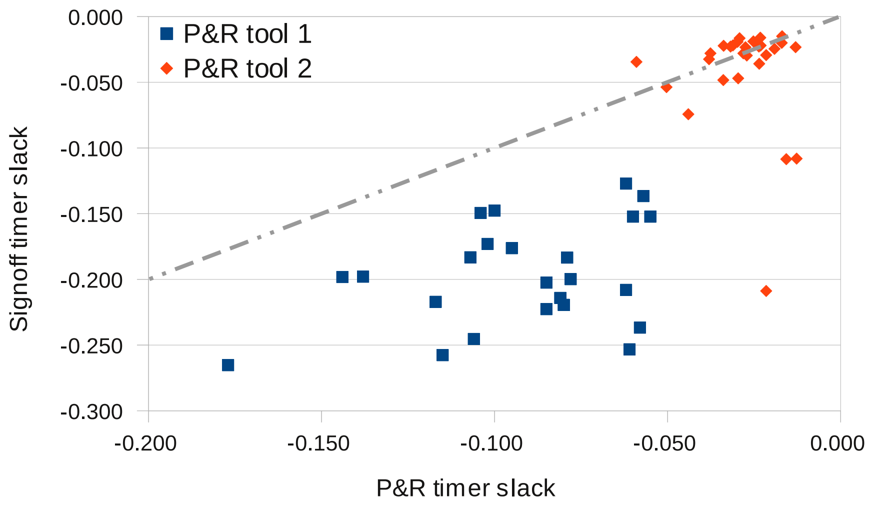


Figure 3.8: Mismatch between the timing estimates at place and route and at sign-off. The gray line indicates when the two estimates match. Data adapted from, and courtesy of [JK10].

an Engineering Change Order (ECO), and sign-off may need to be performed to create a timing feasible design.

Another problem that arises from this mismatch is the potential for suboptimality. While the sign-off and verification may be successful, the design tool may be overly pessimistic if the timing estimates of the place and route timer are significantly worse than the sign-off timer. There may be pessimism in the timing for each path, as well as the total worst-case delay, and minimum cycle time. This pessimism translates into suboptimality due to over-design, and as this discrepancy grows, the potential for this suboptimality also grows.

At this point in the design, it is important to consider slew. Figure 3.10 shows how the sizing algorithms differ when slews are disabled; it is important that post-layout methods handle slew effects, and that post-layout sizing comparisons have slew effects enabled in the timer to be realistic.

3.3.2 Incremental placement and routing considerations

Another consideration is the need for incremental placement and routing. This may be needed if the cells increase in size or the connection pins change location. While this is not important in the v_t assignment or gate-length biasing contexts, it is important in the gate sizing context where the cell size and the pin locations may change.

Incremental placement and routing is usually done after a series of gate sizing changes[5], and may affect other aspects of the design. The changes made by gate sizing may change the locations of the pins and require rerouting. This may also require neighboring gates to be moved, which in turn may also require rerouting. After the incremental placement and routing, the design may again become timing-infeasible, and require an additional iteration of gate sizing. An example of the change in timing after incremental placement and routing is shown in Figure 3.9.

3.4 Comparisons

In this section, we provide three attempts to benchmark gate sizing methods. These efforts focus on methods for post-layout gate sizing that utilize standard cell libraries with a limited availability of gate sizes and threshold voltages. In addition, the effects of interconnect delay, and slew are considered. The first benchmarking attempt is based on a open-source static timing software. The next attempt utilizes artificially created circuits with known optima. The last attempt uses scripts built on top of a commercial tool, which provides realistic timing and power estimates.

	Nangate 45nm		Commercial 65nm			
	w		w		w, v_t	
	LP	LR	LP	LR	LP	LR
c1355	0.20%	-0.56%	1.21%	-1.15%	0.41%	0.19%
c1908	0.47%	1.41%	1.90%	-0.40%	2.71%	0.56%
c3540	-0.20%	0.40%	1.70%	-0.24%	0.63%	-0.21%
c432	-0.20%	0.07%	-0.85%	-0.68%	0.60%	-0.35%
c5315	0.11%	1.08%	0.57%	-0.23%	-0.07%	-0.27%
c7552	0.37%	0.27%	0.52%	-0.14%	0.20%	-0.13%
b15	8.55%	8.54%	0.44%	-0.07%	0.10%	-0.11%
b17	6.50%	6.50%	0.21%	-0.06%	0.00%	-0.01%
b18	10.46%	10.46%	0.05%	0.05%	0.01%	0.00%
b20	0.30%	0.23%	0.39%	-0.04%	0.08%	-0.02%
AVERAGE	2.66%	2.84%	0.61%	-0.30%	0.47%	-0.04%

Table 3.8: Comparisons using UCLA timer. Percentages denote improvements over a feasible-start greedy algorithm (see Section 2.3.3).

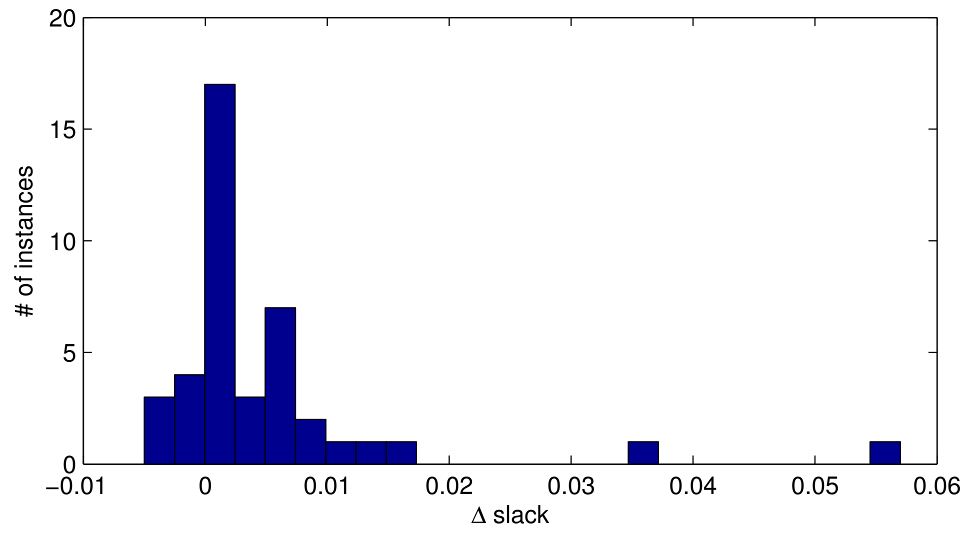


Figure 3.9: Histogram of the change in the timing after incremental placement and routing. There are 64 different examples. The majority of the cases have a zero change in timing, and most of the changes increase the timing (the change is positive). The maximum decrease and increase are -.007ns and .057ns, respectively.

3.4.1 Comparisons using UCLA Timer

Comparisons of gate sizing and threshold voltage assignment methods can be found in [Mok10]. In this comparison, an LP assignment [NDO03] and the LR+DP method [LH09] are compared against a feasible-start greedy algorithm (see Algorithm 1) that uses a ΔPower/ΔSlack sensitivity function to convert the slack in a minimum delay design into power savings. The experiments are run on the Nangate Open Cell Library [NANb] and a commercial 65nm library. The static timing analysis tool used in this work is the UCLA Timer [Mok11][6], which is an open source project based on Open Access[7] and the defunct OA Gear[8]. The open-source nature of this timer provides flexibility in implementing the sizing methods

[5]While it can be done after each gate is changed, it is more efficient to handle these changes at one time by performing an incremental placement, followed by an incremental routing after a series of changes are made.

[6]Available at http://www.nanocad.ee.ucla.edu/Main/DownloadForm.

[7]See http://www.si2.org.

[8]http://www.si2.org/openeda.si2.org/projects/oagear.

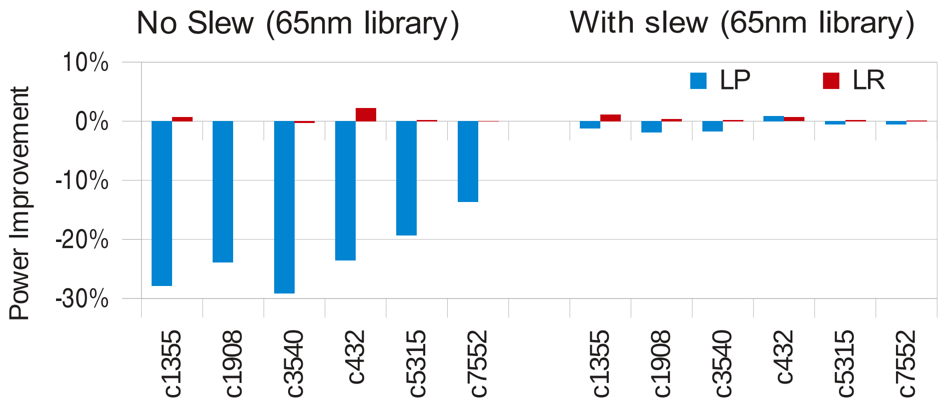


Figure 3.10: The effects of disabling slew in sizing experiments; the performance improvement is relative to the timing-feasible greedy method. Disabling slew increases the difference between sizing methods.

and also increased access to timing information.

The results are shown in Table 3.8. In this table, positive values denote improvement over a timing-feasible greedy method. The improvements are generally small, on the order of 1%, with a maximum improvement of 10.46% on the b18 benchmark using the Nangate Library. The small sizes of these improvements suggest that more sophisticated sizing and threshold voltage assignment methods provide only modest benefits. An interesting comparison is to see how disabling the slew affects the results. Figure 3.10 shows that disabling slew increases the difference between the sizing methods.

3.4.2 Comparisons using eyecharts

Eyecharts[9] [GKK10] were developed as a benchmarking tool for gate sizing and threshold voltage assignment, and are artificially constructed circuits that have known optimal solutions. These circuits are constructed using combinations of the basic structures in Figure 3.11, which have the property that the optimal solution can be computed using a dynamic-programming-like optimization.

[9]Eyecharts are available for download at http://www.nanocad.ee.ucla.edu/Main/DownloadForm.

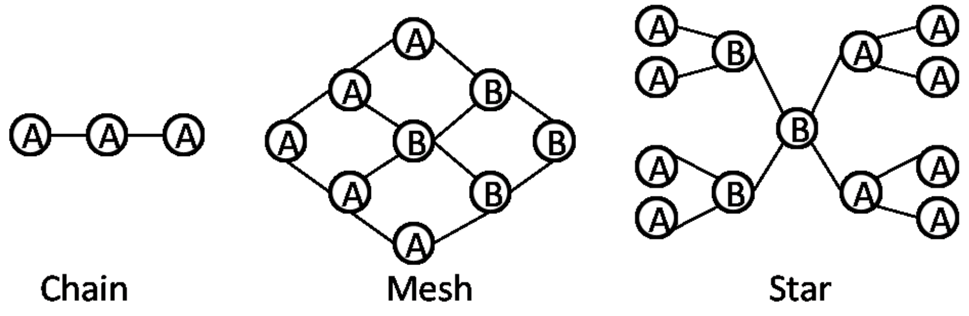


Figure 3.11: The basic structures used in the eyechart benchmarks.

The appeal of Eyecharts is that the optimal solutions are known. This makes it useful as a benchmarking tool for sizing methods. Their performance relative to the absolute minimum can be judged, as opposed to the majority of prior work where only the relative performance between algorithms could be compared.

In [GKK10], two commercial tools, (Comm1) and (Comm2), the LP slack allocation in [NDO03] (LP), a greedy sizing method using ρ_d from equation (2.25) (GS), and a greedy sizing using ρ_s from equation (2.26) (SBS) are compared.

Figure 3.12 show the results of the comparisons. The LP method performs the best in the gate sizing context (Figure 3.12(a)), while the commercial tool (Comm2) performs the best in the v_t (Figure 3.12(b)) and the gate-length contexts. In contrast, the LP method performs poorly in the v_t context.

The difference between the methods is significant, over 10% in some cases, indicating that there can be substantial improvements to be gained from good gate sizing heuristics. Furthermore, the results show a suboptimality gap; the sizing methods are approximately 5% to 15% from optimal.

Figure 3.13 shows the impact of the number of sizes on the resulting suboptimality. The suboptimality decreases overall as the number of sizes for each gate increase. However, this is not true for the commercial tools (Comm1) and (Comm2) – they perform as well on libraries with two cells as they do with libraries with 10 cells.

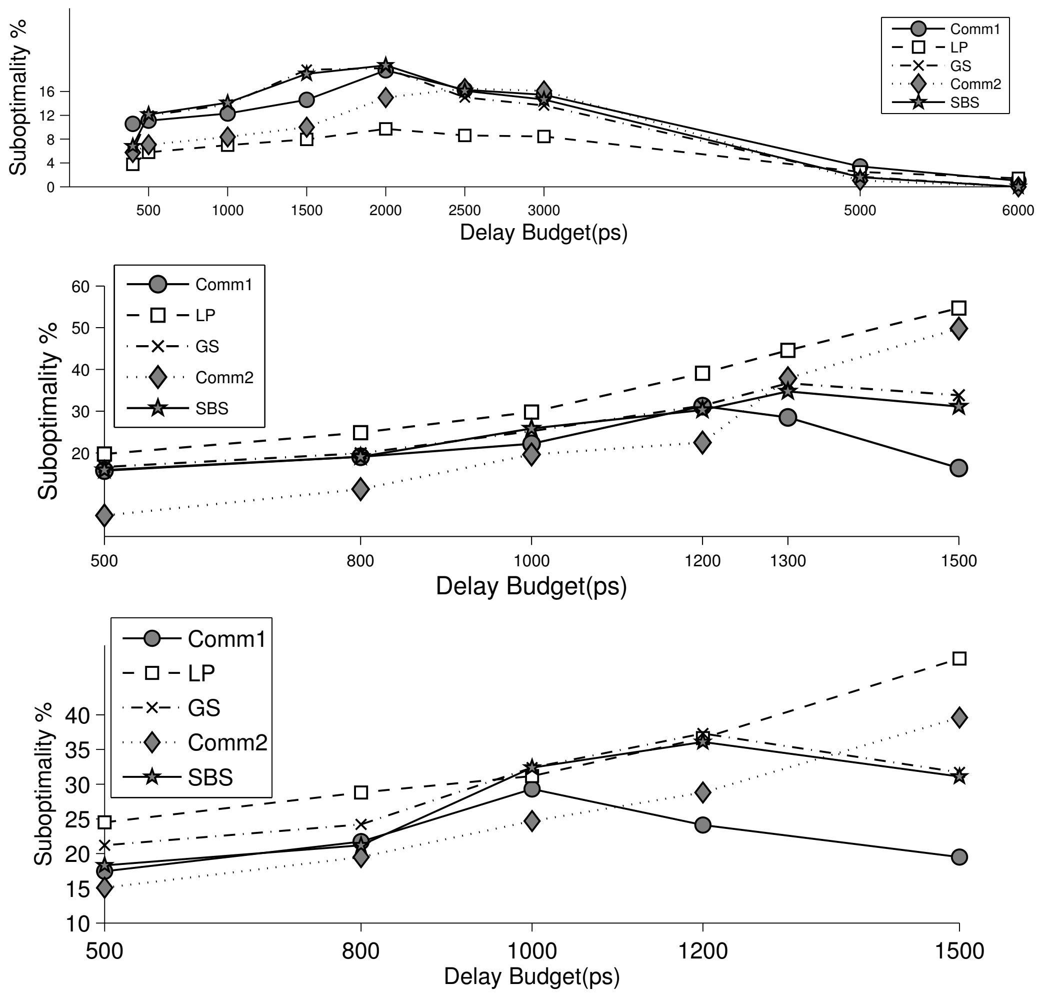


Figure 3.12: Eyechart comparisons of different gate sizing and threshold voltage assignment methods. (a) represents a gate-sizing comparison, (b) represents a v_{t} assignment comparison with 3 threshold voltages, (c) represents a gate length assignment with 3 gate lengths.

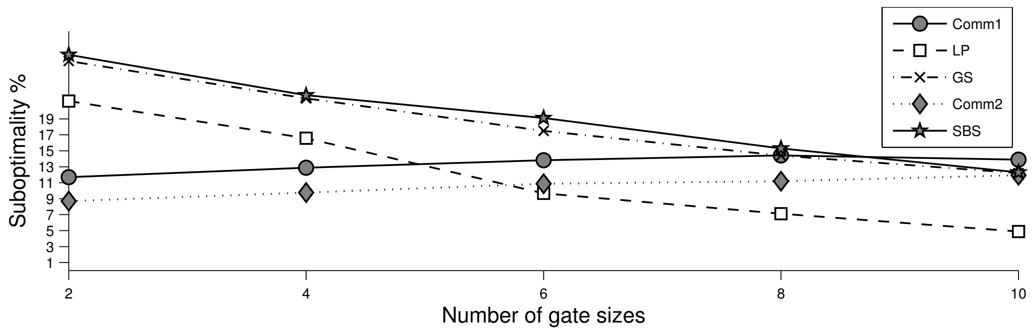


Figure 3.13: The effect of the number of gate sizes available in the library on the suboptimality of the gate sizing method.

3.4.3 Comparisons using commercial tools

Sizing and threshold voltage assignment methods were also compared using TCL scripts implemented in Cadence Encounter [Cad][10]. Implementations of four gate sizing and threshold voltage assignment methods were compared:

1. TILOS [FD85].
2. Feasible-start greedy method (Algorithm 1).
3. LP based slack assignment (Section 2.3.5).
4. LP assignment method (Section 2.3.10).

Experiments were run on the Nangate Open Cell Library [NANb] and a commercial 65nm library, and the benchmarks were originally synthesized, placed and routed for minimum delay, with all optimization effort flags set to high. In the implementation of [NDO03], ΔPower/ΔSlack was used in place of ΔPower/Δt_a, and in the implementation of TILOS and the greedy method, the ΔPower/ΔSlack sensitivity function was used.

Results are shown in Figures 3.14, 3.15, and 3.16 for the Nangate Library, the commercial 65nm library with gate sizing only, and the commercial 65nm with v_t assignment. The plots show the relative leakage power suboptimality of each of the method with the

[10]These scripts are available at http://www.nanocad.ee.ucla.edu/Main/DownloadForm.

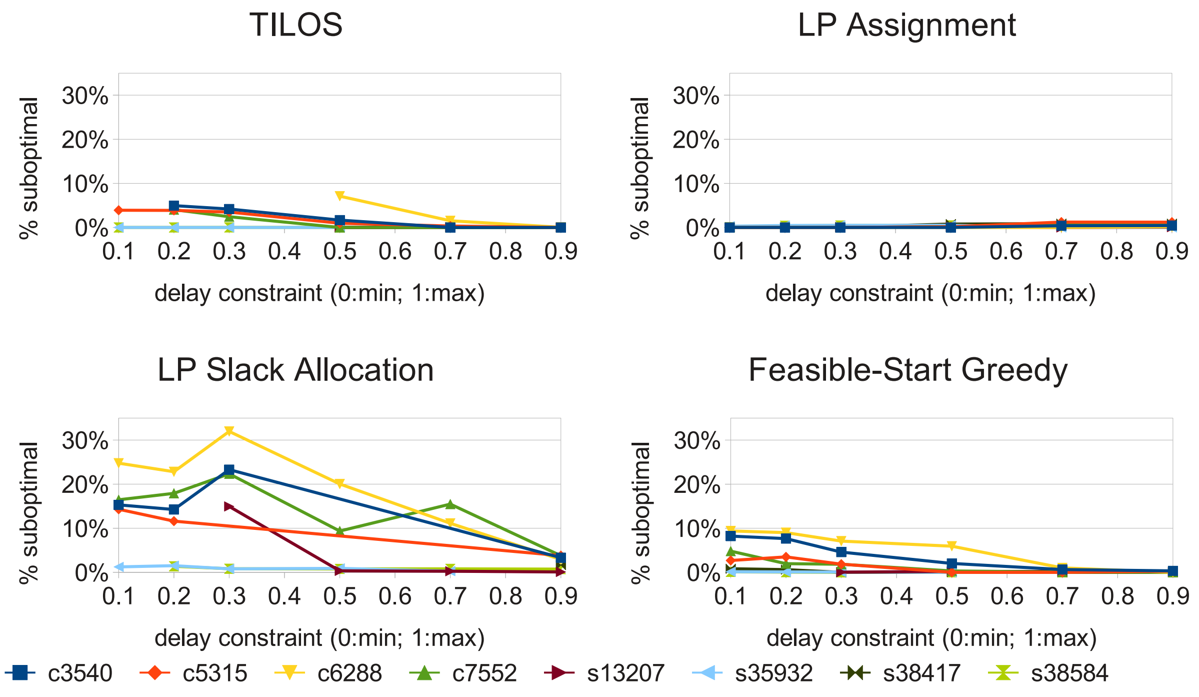


Figure 3.14: Comparison between sizing methods using the Nangate 45nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

best solution found for that benchmark and timing constraint. Overall, the LP assignment method performs the best, followed by the TILOS method, and then the feasible-start greedy method. The details of the results vary by library; in the Nangate Library results in Figure 3.14, the LP assignment provides results that are within 2% of the best solutions, followed by TILOS (within 7.1%), feasible-start greedy (within 10%), and LP Slack Allocation (within 32%). The average relative suboptimalities are .25%, 1.1%, 2%, and 11%, respectively.

The results for the commercial 65nm with gate sizing (Figure 3.15) are different from the Nangate Library results. This library has more available gate size options; for example, the nand cell has 10 options in this library, compared to 3 options in the Nangate Library. In this case, the LP assignment has a couple poor solutions (s35932 and s38584 with delay constraint .2). In this case, the TILOS method is the best, with a relative suboptimality of 1.3% on average, and a maximum of 4.4%. The suboptimalities for the remaining methods are 2.3% (avg) and 8% (max) for the LP assignment, 4% (avg) and 28% (max) for the feasible-start greedy, and 34% (avg) and 142% (max) for the LP slack allocations.

The commercial 65nm with v_{t} assignment has the most dramatic results (see Figure 3.16). This is because v_{t} assignment can provide order-of-magnitude type power reductions per gate, which exaggerates the difference between good and bad solutions. The LP slack assignment and the feasible-start greedy methods perform very poorly in this situation for the combinatorial circuits. However, the results for the sequential circuits (s13207, s35932, s38417, and s38584) are better; for these cases, the suboptimality is a maximum of 26% with the LP slack assignment and the feasible start greedy. The LP assignment method works very well in this case, with an average relative suboptimality of less than 1%, and maximum of 2%. The TILOS method was next in performance, with an average relative suboptimality of 5% and a maximum of 23%.

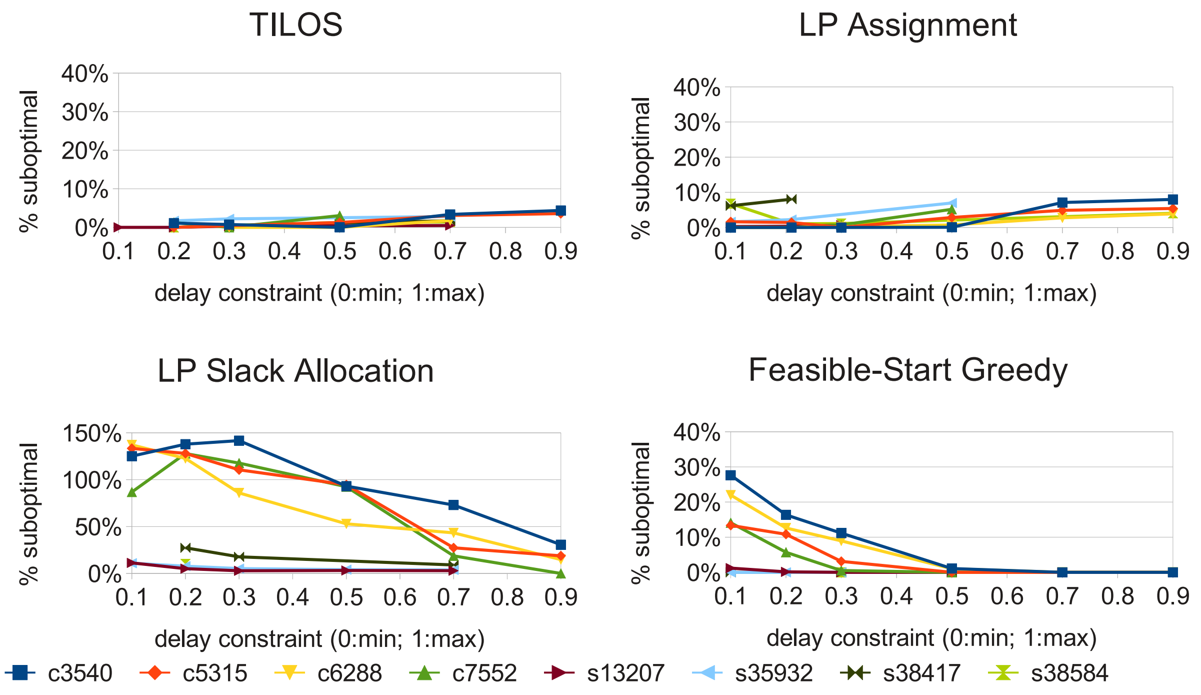


Figure 3.15: Comparison between sizing methods using a Commercial 65nm Library=. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

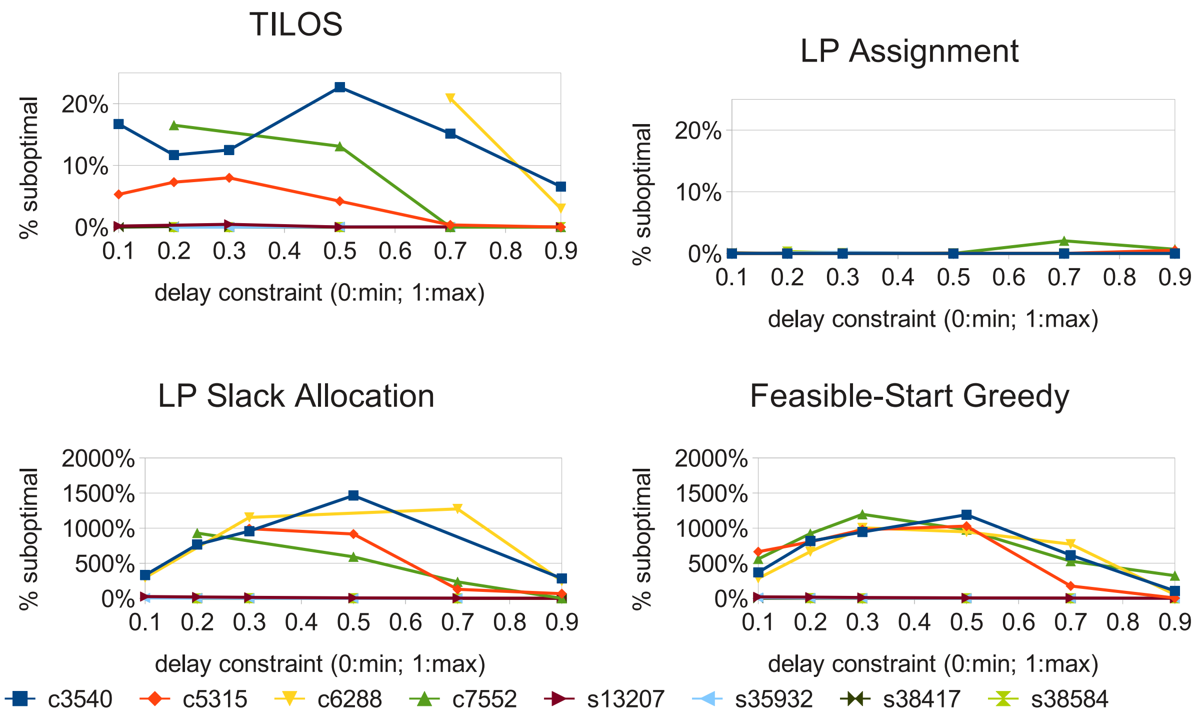


Figure 3.16: Comparison between v_t assignment methods using a Commercial 65nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint – 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

CHAPTER 4

Parametric Variation and Gate Sizing

In nanometer designs, the effects of variability are too large to ignore [Nar05]. This is best seen by examining the ITRS Roadmap 2009 [ITR], which is a report sponsored by semiconductor companies designed to identify future challenges. This report predicts that in 2011:

- 11%: Effects of parametric variation on sign-off delay.
- 20%: v_{t} variation.
- 60%: Performance variability.
- 88%: Total power variability.
- 255%: Leakage power variability.

Furthermore, this variability is projected to increase. Figure 4.1 shows the variability data through the year 2016. The increases are the greatest for the leakage power variability, which is projected to increase at a rate of 24% per year! This has motivated statistical gate sizing methods to mitigate the "soaring leakage variability" [ITR]. These methods use statistical models for delay and power to create designs that are robust against variation.[1].

Parametric variations are generally divided into two types [SSB05]: (1) within-die, or intra-die variations, where each device in a die will see different values of the variation, and (2) die-to-die, or inter-die variations, where each device on a die has the same value of the variation. These variations may come from many sources [LLP06], such as dishing, diffraction effects, line-edge roughness, lens aberrations, and dopant fluctuations.

[1]For an in-depth discussion of statistical variations and modeling, see [LLP07].

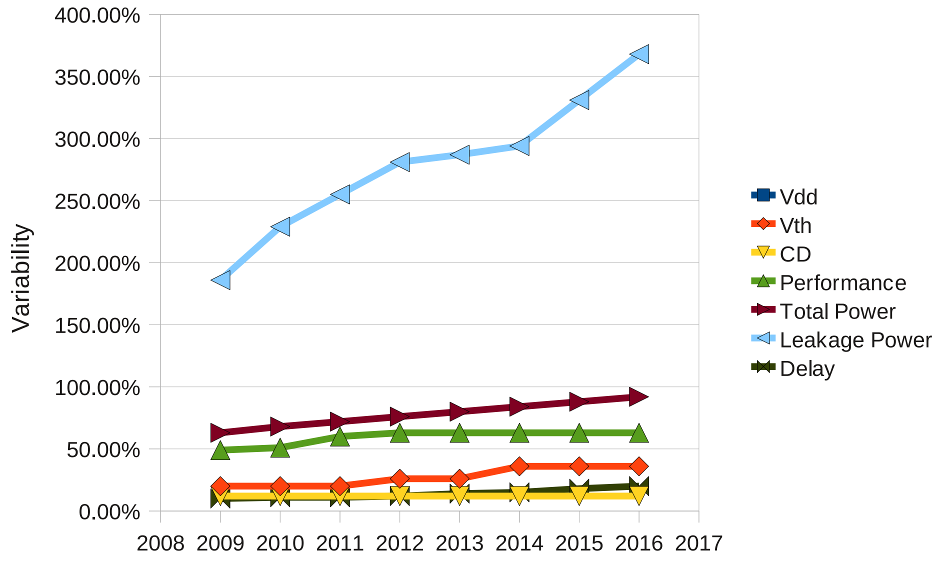


Figure 4.1: Projected variability in key design parameters (adapted from [ITR]).

This parametric variability has motivated statistical design paradigms in the new millennium, which have come with the optimism that these methodologies would provide improvements over deterministic methods [Vis03]. Methods to reduce the impact of these variations will be outlined in this chapter. First, motivating examples on the effects of statistical variation will be provided. Next, an analysis of the importance of statistical power optimization is given, and bounds provided to quantify the benefits of full statistical power optimization. Lastly, methods to optimize the statistical delay will be presented.

4.1 Motivating examples

4.1.1 The slack wall

An excellent motivational example for statistical design is the "slack wall" [BVS02]. In regular deterministic design, there is an incentive to make the delay for each path *equal to the maximum allowed delay* because:

1. Paths slower than the maximum delay violate the timing and are therefore not al-

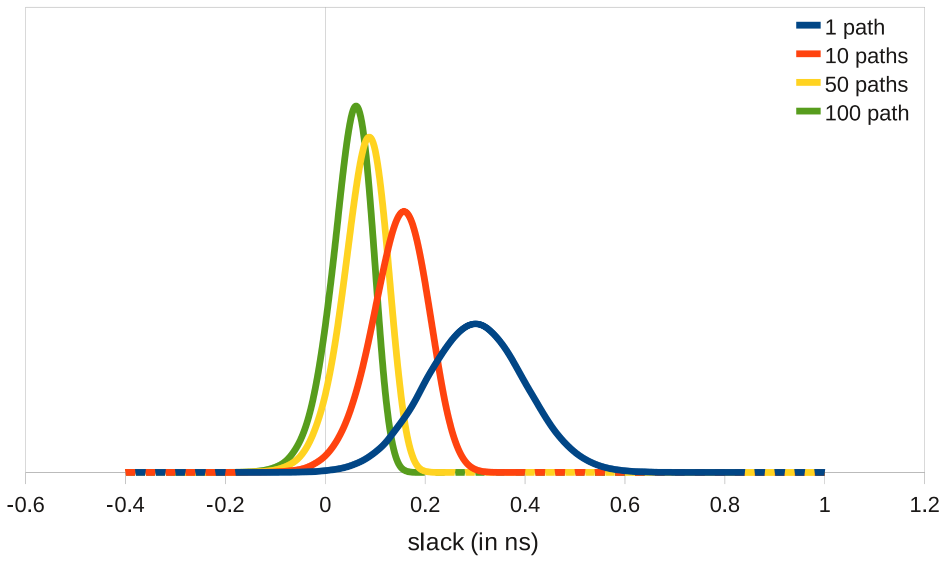


Figure 4.2: Effects of a "slack wall" on statistical slack. The PDF for the worst-case slack is shown for 1, 10, 50 and 100 paths. The delay of each path is Gaussian with variance 100ps.

lowed.

2. Paths faster than the maximum delay are not optimal– they can be slowed down to save power.

This has the effect of creating a "slack wall", a large number of paths having a small positive slack.

While these slack walls are optimal from a deterministic point of view, they are suboptimal from a statistical point of view because the minimum slack *over all the paths* will be *worse* than the slack of each one of its paths. For example, suppose that a 300ps guard-band is used to ensure a high yield. Also suppose that the delay variation of each path has a standard deviation of 100ps. Thus, the yield would be expected to be the 3-σ value 99.98%.

This is true when the paths are completely dependent, but it underestimates when the paths are not completely dependent. For example, Figure 4.2 plots the statistical slack

corner	temperature	supply voltage	process
fast	0C	1.25V	FastFast
typical	25C	1.1V	TypicalTypical
slow	125C	0.95V	SlowSlow

Table 4.1: Corners in the Nangate 45nm library [NANb].

probability density distribution (PDF) as a function of the number of independent paths. As the number of independent paths increases, the slack distribution shifts towards the negative slack direction. When there are 100 independent paths, the yield drops to 92% – a huge decrease from 99.98%.

This example shows that deterministic optimization may be overly optimistic and result in under-design. However, there are also cases where deterministic optimization may result in over-design [Vis03].

4.1.2 Worst-case corners and over-design

Most earlier design methods relied on corner based methods for design [NSD86, TGW87, Naj05]. The corners were process and operating condition parameters that were used for validation. For example, a fast corner might be at a low temperature, have a high supply voltage, and use a fast process. Conversely, a slow corner might operate at a high temperature, have a low supply voltage and use a slow process. An example of the corners in the Nangate 45nm Library is shown in Table 4.1.

When the variations become large, as in Figure 4.1, the parametric variability of the transistors cause variations in the timing and power that are on par with the fluctuations in the operating conditions. Due to this, the *fast* and *slow* process corners become inadequate for characterizing the design. One example of this inadequacy is the notion of a worst-case corner. Traditionally, this corner is taken as a 3-σ value of the variations. However, when there are n independent sources of variation, each with standard variations $\{\sigma_1, ..., \sigma_n\}$, the corner must be fitted to provide an overall $3-\sigma$ value. Using the 3-σ value for each of these parameters will result in a $\sqrt{n} \cdot 3\sigma$ value [GVV05], which is much more conservative than is needed.

Even if a correct $3-\sigma$ value for the delay can be found, it may still be conservative due to intra-die random variations. Intra-die variations may have a smaller impact on the size of the variation than inter-die variations. For example, consider an inverter chain with N gates, whose delays are random variables, $\mathbf{D}_1, ..., \mathbf{D}_N$ that can be generated from $N+1$ IID $\mathcal{N}(0,1)$ random variables $\mathbf{X}_1, ..., \mathbf{X}_N$ as:

$$\begin{bmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_N \end{bmatrix} = 1 + \begin{bmatrix} \alpha & 0 & \dots & 0 & (1-\alpha) \\ 0 & \alpha & \dots & 0 & (1-\alpha) \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \alpha & (1-\alpha) \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{N+1} \end{bmatrix}. \tag{4.1}$$

The parameter α is used to examine the effect of varying the proportion of the variation caused by inter-die and intra-die variations. The total delay of the inverter chain is the random variable defined by $\mathbf{D} = \sum_{i=1}^{N} \mathbf{D}_i$. When $\alpha = 1$ (no intra-die variations are present), $\mathbf{D}$ is mean N and standard deviation $\sqrt{N}$. On the other hand, when $\alpha = 0$ (only intra-die variations are present), $\mathbf{D}$ is still mean N but the standard deviation is N. For general α, the mean is always equal to N, but the standard deviation varies as $\sqrt{N^2(1-\alpha)^2 + N\alpha^2}$. The main idea here is that the correlation between gates is important, as it impacts the variance. However, the impact of the correlation is also dependent on the number of stages, N, as the difference between the correlated and uncorrelated cases is $N - \sqrt{N}$ and grows as the stages grow. However, this is difficult to incorporate into *corners*, as the number of stages is design dependent.

The last motivation for statistical methods is due to the difference in the sensitivities of gates to the variation sources. Different library cells may respond differently to variations in gate lengths, oxide thicknesses, doping, line-edge roughness, etc. This difference may make the parameters that make the worst-case for one cell different from another. In other words, it may be impossible for two cells to be at the worst-case scenario at the same time! Thus, characterizing them at their 3-σ points may be overly pessimistic.

These criticisms are not uncontested. In [Naj05] the author argues that the drawbacks of corner based analysis may be exaggerated. He argues that corners can account for the correlations between gates by *adjusting* the corner to be less conservative. Furthermore, he argues that corner based analysis may not be overly conservative for the reasons shown in Figure 4.2. A larger number of critical paths will cause the yield to decrease, and thus a conservative approach may be needed.

4.2 Statistical power optimization

While there are many papers that explore the benefits of adding statistical delay data into the optimization process [MO04, PYK05, SNL05, CSS05, GVV05, SSZ05, DS06], and there are also a number of papers that use a statistical power measure [SSB04b, CSS05, MDO05, BV05, YZL07, CLV08], there is no publication, to the best of our knowledge, that shows the benefits of using the statistical power measure alone [CGL09, CGL10]. This brings up an interesting question:

> How much of the improvement should be attributed to the use of a *statistical* delay model, and how much should be attributed to the use of a *statistical* power model?

This question is part of a growing skepticism over the benefits of statistical optimization, and whether they outweigh its costs. Adopting statistical analyses and optimization involve considerable overhead in terms of the engineering effort involved as well as the turn-around time. It requires an almost complete overhaul of process modeling, circuit simulation and also a modification to the algorithms for statistical optimization. It is therefore important to do a thorough cost-benefit analysis of statistical optimization compared to conventional deterministic optimization methods.

In this section we focus on the amount of improvement that can be made by using a statistical power measure as an objective for gate sizing, l_{eff} and v_{t} assignment, when compared to the deterministic power measure. The key contributions are as follows.

- We develop a mathematical programming based framework to estimate the suboptimality gap between different power measures.
- For the common case of discrete gate sizing, we give an intuitive explanation of the suboptimality using solution rank orders.
- We show that for certain sizes of variations, the deterministic power measure is a provably good approximation for the statistical power measures, which means that the deterministic power measure can be used in place of the statistical power measures with very similar optimization results.
- We present a simplified measure of statistical power that can be used as a proxy for full statistical power optimization.

It is important to mention that the analysis in this section is independent of the model for the delay. This research does not give judgments on the difference between statistical delay optimization and deterministic delay optimization, or the difference between static timing analysis and statistical static timing analysis. The delay is only used to generate examples for the suboptimality bounds.

The approach in this section intends to remove the dependence on timing feasibility, which is a consideration that makes the problem of finding bounds difficult. To remove this dependence, we rely on the structure of the power objective to create a relaxation of the timing feasible region. This can be done because a power optimum inherently contains information about the timing feasible region, as this optimum is the minimum power point in the region. Once this is done, the relaxation can be used to find bounds without any further dependence on the timing feasible region.

The rest of the section is organized as follows. The following section outlines the leakage power measures and models used in this section. Section 4.2.2 describes the test

Table 4.2: Notations

symbol	meaning
$w_i, \vec{w}$	width of gate i / vector of gate widths
$l_i, \vec{l}$	length of gate i / vector of gate lengths
$v_{\mathrm{t}_i}, \vec{v}_{\mathrm{t}}$	threshold voltage of gate i / vector of threshold voltages
$\vec{z}$	'adjusted gate widths', $z_i = w_i e^{\alpha l_i^2 + \beta l_i} e^{-\gamma v_{\mathrm{t}_i}}$
$\mathbf{L}_{\mathrm{w}}$	Within-die gate length variation random variable
$\mathbf{L}_{\mathrm{d}}$	Die-to-die gate length variation random variable
$\mathbf{V}_{\mathrm{t}_{\mathrm{w}}}$	Within-die v_{t} variation random variable
$\mathbf{V}_{\mathrm{t}_{\mathrm{d}}}$	Die-to-die v_{t} variation random variable

circuits that are used. Section 4.2.3 gives an intuitive explanation of the suboptimality gap using rank orders. Section 4.2.4 develops the mathematical framework to estimate the suboptimality gap incurred from using the deterministic power measure in place of the statistical power measures. Section 4.2.6 introduces a simplified measure of statistical power that can be used as a proxy for full statistical power optimization, followed by a summary in Section 4.2.7.

4.2.1 Preliminaries

In this dissertation, we follow the convention uppercase bold symbols represent random variables (e.g., $\mathbf{X}$), and uppercase non-bold symbols represent commonly used constants (e.g., V_{dd}). Vector quantities will have arrows above them (e.g., $\vec{x}$) to avoid confusion, and scalar quantities will be lowercase non-bold (e.g., p). The principal symbols are summarized in Table 4.2.

4.2.1.1 Variations

In this section, the gate length and the threshold voltage are assumed to be the sources of power variations. These variations are assumed to be Gaussian in both the length and the

measure	symbol	expression
deterministic	$p_d(\cdot)$	$\sum \kappa_i$
mean	$p_m(\cdot)$	$\sum \kappa_i \exp((\lambda^2_{\mathrm{w}(i)}\sigma^2_{\mathbf{L}_{\mathrm{w}(i)}} + \lambda^2_{\mathrm{d}(i)}\sigma^2_{\mathbf{L}_\mathrm{d}} + ...$ $\tau^2_{\mathrm{w}(i)}\sigma^2_{\mathbf{V}_{\mathrm{tw}(i)}} + \tau^2_{\mathrm{d}(i)}\sigma^2_{\mathbf{V}_{\mathrm{t_d}}})/2)$
mean+3σ	$p_{m3\sigma}(\cdot)$	$p_m(\cdot) + 3p_\sigma(\cdot)$
99.87%–quantile[1]	$p_q(\cdot)$	$\sum \kappa_i \exp(3(\lambda_{\mathrm{d}(i)}\sigma_{\mathbf{L}_\mathrm{d}} + \tau_{\mathrm{d}(i)}\sigma_{\mathbf{V}_{\mathrm{t_d}}}))$
standard deviation, σ	$p_\sigma(\cdot)$	(see eqn (4.4))

[1]For inter-die variation only

Table 4.3: Measures of statistical leakage power

threshold voltage. The standard deviations in v_t are 4.714% in both die-to-die (dtd) and within-die cases (wid); the within-die variations are uncorrelated and they affect each gate independently. Three different standard deviations for the length are used in this section:

- 1nm (dtd) and .5nm (wid)
- 1.6nm (dtd) and 1.15nm (wid)
- 2nm (dtd) and 0nm (wid)

These are representative of the variations for 45nm given in the ITRS Roadmap 2007.

The effects of the variation are simulated for the gates in the Nangate Open Cell Library v1.2 [NANa] using PTM 45nm and HSpice 2007[2]. We assumed that the input combinations to each gate are equi-probable and the measured leakage power is the average power over all the input combinations.

[2]For reference, the channel length used in the HSpice models was 50nm. Also, note that this simulation tool is different from the one in [CGL09]. It was changed because the version of the tool in [CGL09] could not simulate the effects of v_t variation.

4.2.1.2 Models

The leakage power is modeled as a log-normal random variable, as in [RSB04]:

$$\begin{aligned}\mathbf{P}_{\mathrm{L}} &= \sum \kappa_i e^{\lambda_{\mathrm{d}(i)}\Delta\mathbf{L}_{\mathrm{d}}+\lambda_{\mathrm{w}(i)}\Delta\mathbf{L}_{\mathrm{w}(i)}} e^{\tau_{\mathrm{d}(i)}\Delta\mathbf{V}_{\mathrm{td}}+\tau_{\mathrm{w}(i)}\Delta\mathbf{V}_{\mathrm{tw}(i)}} \\ &= \sum \mathbf{P}_{\mathrm{L}_i}.\end{aligned} \tag{4.2}$$

κ_i is the nominal power for the gate i; $\Delta\mathbf{L}_{\mathrm{w}(i)}$, $\Delta\mathbf{L}_{\mathrm{d}}$, $\Delta\mathbf{V}_{\mathrm{tw}(i)}$, $\Delta\mathbf{V}_{\mathrm{td}}$ are independent Gaussian random variables; and $\lambda_{\mathrm{d}(i)}$, $\lambda_{\mathrm{w}(i)}$, $\tau_{\mathrm{d}(i)}$ and $\tau_{\mathrm{w}(i)}$ are coefficients that are used to fit the mean and the standard deviation of the Spice simulated data. In equation (4.2) above, the variations in the threshold voltage and the gate length are assumed to be Gaussian, and the relationship between these parameters to the leakage power is exponential.

To model the change in the leakage as a function of the size, threshold voltage and the length (see Section 4.2.4), we use the approximation:

$$\begin{aligned}\kappa_i &\approx \kappa_i' w_i e^{\alpha_i l_i^2+\beta_i l_i} e^{-\gamma_i v_{\mathrm{t}_i}} \\ &= \kappa_i' z_i\end{aligned} \tag{4.3}$$

where κ_i', α_i, β_i and γ_i are parameters that are fitted to the data. The variable z_i denotes the "adjusted gate width", which incorporates the effect of w, l and v_{t} on the power into an equivalent gate width. The z variables cannot be re-mapped to a unique w, l and v_{t}, but this is not a problem, as z is used only to compute lower bounds and is not used for design purposes. The nonlinear mapping from w, l, and v_{t} to z has the effect of shifting the nonlinear relation of p_d on v_{t} to a linear one in z (which is in turn nonlinear in v_{t}), creating a useful abstraction for computing bounds.

4.2.1.3 Measures of statistical leakage power

In this section we will cover the statistical measures summarized in Table 4.3: the deterministic, mean, mean + 3σ, and the quantile power measures. The mean + 3σ measure refers to the mean + 3σ of the total leakage power of a design. When there is no within-die variation, the 99.87% quantile measure, which corresponds to the 3 sigma quantile of a Gaussian distribution, will be used as well.

In the table, $p_\sigma(\cdot)$ is the measure of the standard deviation of the power, which can be expressed as:

$$p_\sigma^2 = \sum_i \sum_j \mathbf{E}\mathbf{P}_{\mathrm{L}_i}\mathbf{P}_{\mathrm{L}_j} - \mathbf{E}\mathbf{P}_{\mathrm{L}_i}\mathbf{E}\mathbf{P}_{\mathrm{L}_j} \tag{4.4}$$

The quantile measure p_q is used when only inter-die variation is present. When intra-die, or within-die, variation is present, there is no closed form expression for p_q.

The power measures above have useful mathematical properties. p_d, p_m and p_q are all linear in κ_i, and are thus concave *and* convex in z_i. $p_{m3\sigma}$ is convex in $\vec{z}$.

4.2.1.4 Increasing variations

It is also interesting to examine how the sensitivities change as the variations increase. We can use expression (4.2) for a first-order analysis of how the sensitivities will change as the variations change.

We reconsider the case with inter-die length variation only. Doubling the variations results in:

$$\begin{aligned} p_q &= \sum \kappa_i e^{3\lambda_{\mathrm{d}(i)}(2\sigma_{\Delta \mathbf{L}_\mathrm{d}})} \\ &= \sum (e^{3\lambda_{\mathrm{d}(i)}\sigma_{\Delta \mathbf{L}_\mathrm{d}}})^2 \kappa_i. \end{aligned} \tag{4.5}$$

This will increase the difference in the scaling factors between gates. This is seen in Figure 4.4, where the increase in the variations changes the sensitivities for each gate. Notice that the sensitivities do not increase uniformly for each gate. The growth of the sensitivities depends on the topology of the design, and how much effect the variations will have on the gate.

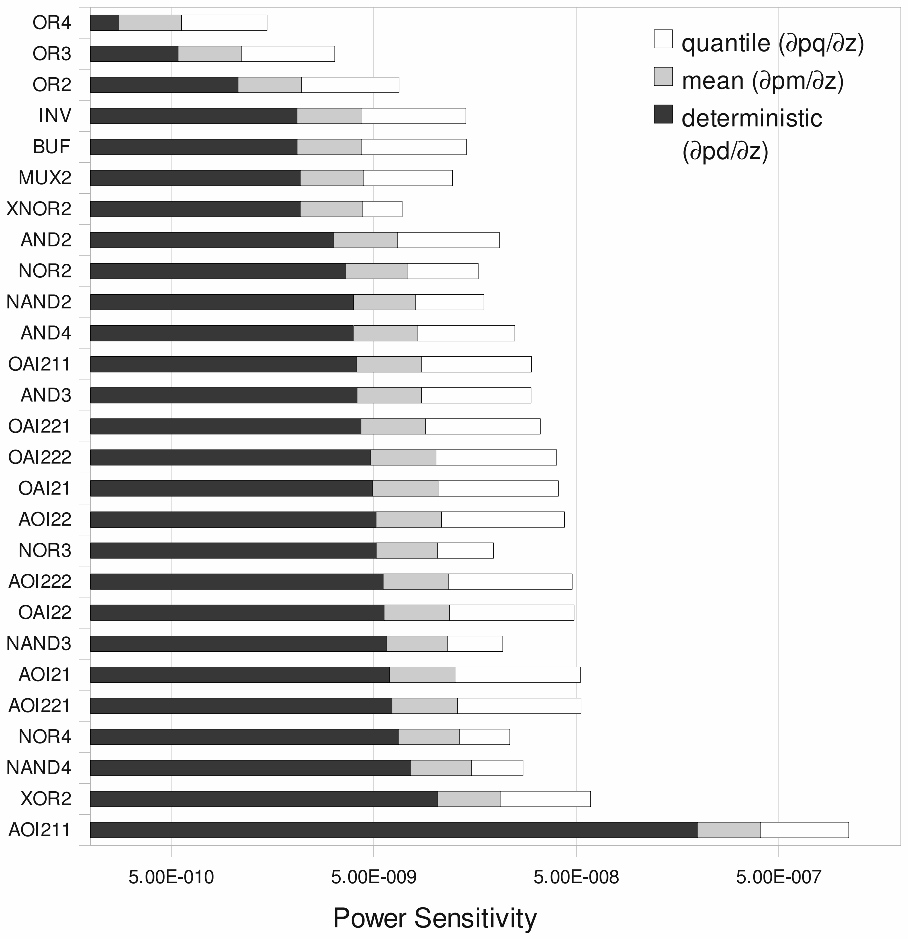


Figure 4.3: Power sensitivities for the different gates in the Nangate Library. The power vs. size sensitivities of the deterministic power measure ($\partial p_d/\partial z$), the mean power measure ($\partial p_m/\partial z$) and the quantile measure ($\partial p_q/\partial z$) for $\sigma_L = 2$nm (dtd)/0nm (wid) and $\sigma_{V_t} = 4.7\%$ (dtd)/0% (wid) are shown. The sensitivities are sorted by the p_d. Notice that a sorting by p_m would be very similar, while a sort by p_q would be significantly different.

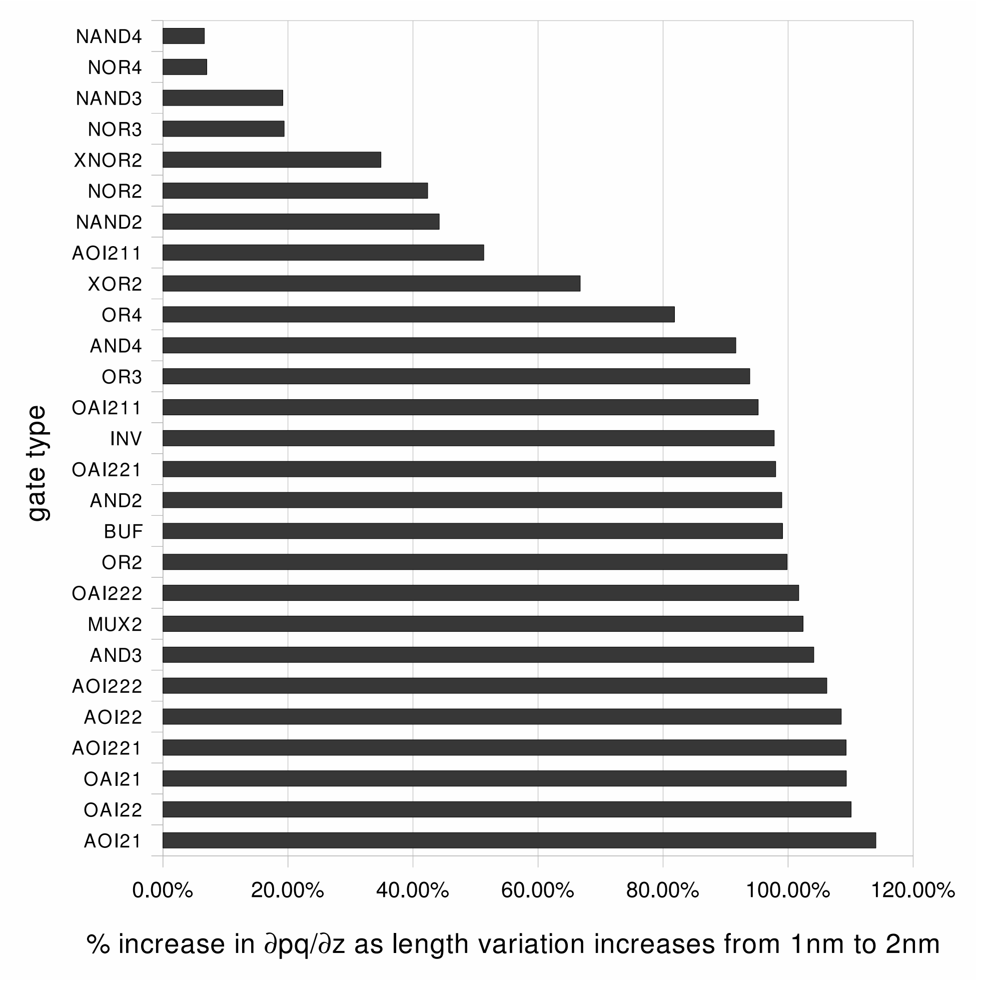


Figure 4.4: The percentage increase in the quantile power vs. size sensitivity ($\partial p_q/\partial z$) as $\sigma_{\rm L}$ = 1nm increases to 2nm is shown. The increase in the sensitivity is not seen equally for all the gates.

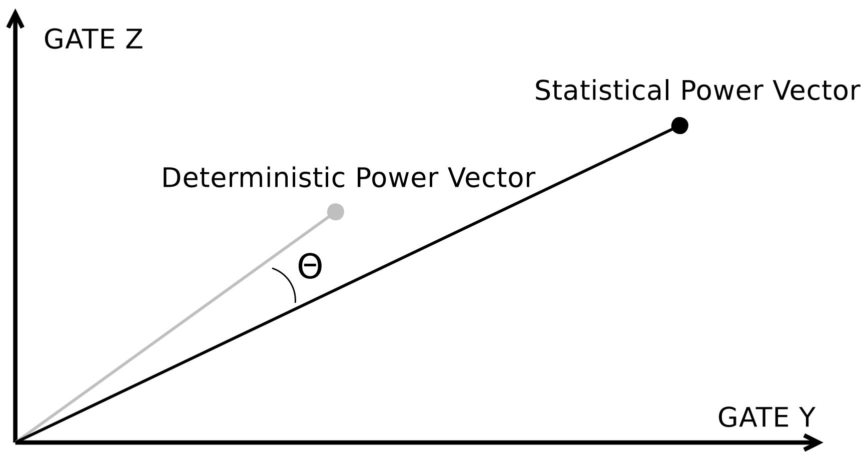


Figure 4.5: Visual representation of the power vector for deterministic power and statistical power. Each axis represents the power for a different gate type. The vectors have different magnitude and direction, but the angle between them Θ is very small.

4.2.1.5 Why do we expect the optimizations to be similar?

The statistical power and deterministic power are not similar. For example, the statistical leakage power can be larger than the deterministic leakage by 10% to 500%. It is natural to expect that the influence of these measures will also be different, and that optimizing statistical power will yield different results compared to deterministic power.

In optimization however, it is not the magnitude of the power, but the *relative* magnitude which matters. If the statistical power is a scaled version of the deterministic power, then the optima will be the same. To see this mathematically, we examine the optimality condition for an optimum $x^\star$: $x^\star$ is optimal if for any *feasible* $x^\star + \Delta x$,

$$f(x^\star) \leq f(x^\star + \Delta x). \tag{4.6}$$

This condition will also hold for any *positive* scaling of $f(x)$.

Although the values of the statistical power and the deterministic powers may be quite different, the trends are similar– the mean power is larger than the deterministic power for all of the gates, as is the quantile power, etc. For example consider Figure 4.3, which shows the power vs. size sensitivities. The sensitivities for the different gates follow the same trend. This suggests that the optimizations will be similar as well.

To see why this happens for statistical power, we take as an example the quantile power expression with inter-die length variation:

$$p_q = \sum \kappa_i e^{3\lambda_{\mathrm{d}(i)}\sigma_{\Delta\mathbf{L}_{\mathrm{d}}}}. \tag{4.7}$$

If the $\lambda_{\mathrm{d}(i)}$ are all equal, then the effect of variation will be seen equally for each gate and the objective will be a scaled version of the deterministic power:

$$\begin{aligned} p_q &= \sum \kappa_i e^{3\lambda_{\mathrm{d}(i)}\sigma_{\Delta\mathbf{L}_{\mathrm{d}}}} \\ &= e^{3\lambda_{\mathrm{d}}\sigma_{\Delta\mathbf{L}_{\mathrm{d}}}} \sum \kappa_i \\ &= e^{3\lambda_{\mathrm{d}}\sigma_{\Delta\mathbf{L}_{\mathrm{d}}}} p_d \end{aligned} \tag{4.8}$$

In actuality, the values of $e^{3\lambda_{\mathrm{d}(i)}\sigma_{\Delta\mathbf{L}_{\mathrm{dtd}}}}$ are different for different gates. Figure 4.4 plots the percentage increase in the power vs. size sensitivities, $\partial p_q/\partial z$, from $\sigma_{\mathrm{L}} = 1$nm to 2nm. The percentage increase is not uniform across the different gates, indicating that the $\lambda_{\mathrm{d}(i)}$ are not equal. The larger variations thus magnify the discrepancies between the different gates.

Surprisingly, the effect of the scaling leaves the orientation of the power sensitivities mostly intact. The magnitude changes significantly, but the direction of the vector remains similar. We can plot a power vector where each gate is a separate dimension, and the magnitude along the dimension is the leakage power for that gate. We can compute the difference in the angle of the corresponding statistical power vector and the deterministic power vector, θ, as shown in Figure 4.5. For $\sigma_{\Delta\mathbf{L}} = 2$nm (dtd) / 0nm (wid) and $\sigma_{\Delta\mathbf{V}_{\mathrm{t}}} = 4.7\%$ (dtd & wid), the difference between the vectors is very small, at $\Theta \approx 1^\circ$. In Section 4.2.4 we will see that the effects may not be large enough to make a significant difference in the optimized powers.

	v1	v2	v3	v4
c432	118 (431)	137 (494)	93 (557)	81 (619)
c499	586 (477)	322 (612)	189 (746)	174 (881)
c880	346 (408)	237 (562)	212 (716)	176 (871)
c1355	598 (473)	356 (599)	222 (724)	174 (850)
c1908	676 (802)	571 (863)	533 (923)	528 (990)
c2670	1041 (610)	791 (787)	748 (964)	723 (1140)
c3540	1615 (1096)	1359 (1289)	1107 (1482)	1046 (1675)
c5315	2019 (1090)	1867 (1226)	1766 (1362)	1717 (1498)
c6288	3089 (1845)	1656 (4127)	1780 (6410)	1189 (8692)
c7552	2894 (866)	2640 (977)	2481 (1089)	2425 (1200)
alu	12598 (896)	5190 (8460)	5271 (16025)	3237 (23652)

Table 4.4: Number of gates in test circuits and the corresponding delay target in parenthesis (in ps)

Gates	Sizes
HA, FA, OAI33	1x
MUX2, XNOR2, XOR2	1x-2x
AND2/3/4, AOI211/21/221/222/22, NAND2/3/4, NOR2/3/4, OAI211/21/221/222/22, OR2/3/4	1x-4x
BUF, INV	1x-32x

Table 4.5: Gate sizes in the Nangate Library

4.2.2 Circuit examples

In this section, we use the ISCAS '85 benchmarks and a 128-bit Arithmetic Logic Unit (ALU) [OPE] as examples. The Verilog RTL of these benchmarks are synthesized to four different target speeds using the Encounter RTL compiler [Cad07] to the Nangate Open Cell Library v1.2 [NANa], which uses the 45nm technology node and has the sizes given in Table 4.5. The synthesized speeds are the maximum speed, the minimum speed, and two speeds in-between. The fastest speed is labeled v1, and the slower synthesized speeds have higher cardinality (e.g., the slowest speed is v4). A table listing the number of gates in each design and the target delays is given in Table 4.4. The tighter delay constraints result in larger designs, as these designs utilize more buffering to meet the delay constraint. These larger synthesized designs are also more interesting from a sizing point of view, as the space of possible solutions is larger.

All the optimization routines in this section are solved using the Matlab Optimization Toolbox [The08].

4.2.3 Analysis via rank correlations

The major difficultly in estimating the difference between statistical optimization and deterministic optimization is due to the difficulty in gate sizing itself. With discrete sizes (e.g., $\vec{w} \in \{1, 2, 4, 8\}^n$ and $\vec{l} \in \{1, 2, 3\}^n$), the problem is NP-complete [Li93] and it is tremendously difficult to find an optimal solution. However, there is some intuition that can be gained by examining the statistical and deterministic powers of points even if timing feasibility is ignored. Namely, we can consider (1) how does a sorting of configurations (w, l and v_{th} assignments) change when ordered statistically, instead of deterministically, and (2) how does the statistical power vary for configurations with the same deterministic power.

To circumvent this difficulty, the issue of timing feasibility is ignored in this section. Instead, we generate random gate size, v_{t} and l assignments and compare the rankings (e.g. orderings) of the deterministic power and the statistical power, and show that ranking /

	p_m				$p_{m3\sigma}$			
	v1	v2	v3	v4	v1	v2	v3	v4
c432	.989	.986	.985	.985	.653	.591	.603	.604
c499	.990	.988	.990	.990	.685	.605	.516	.524
c880	.989	.989	.989	.987	.658	.651	.649	.585
c1355	.995	.987	.987	.991	.684	.603	.504	.537
c1908	.988	.987	.987	.987	.671	.624	.608	.606
c2670	.991	.991	.991	.992	.712	.647	.646	.651
c3540	.991	.990	.989	.989	.702	.668	.619	.602
c5315	.991	.991	.991	.991	.675	.659	.641	.634
c6288	.989	.999	.999	.999	.680	.894	.913	.922
c7552	.990	.990	.990	.990	.686	.660	.637	.642
alu	.997	.999	.999	.999	.894	.899	.885	.905

Table 4.6: Rank correlation τ

ordering by the deterministic power is a good ranking / ordering by the statistical power. In these examples, we assumed that $\sigma_L = 0$nm (wid)/2nm (dtd) and $\sigma_{V_t} = 4.7\%$ (wid/dtd).

As an experiment, the deterministic and statistical powers were computed for 10,000 randomly generated widths, lengths and v_t, for each design. Three different v_t values were used (low, high and normal), and four different gate lengths were used ($\{+0\text{nm}, +1\text{nm}, +2\text{nm}, +3\text{nm}\}$). In these examples, we assumed that $\sigma_L = 2$nm(dtd) / 0nm(wid) and $\sigma_{V_t} = 4.7\%$ (dtd & wid).

Figure 4.6 plots the deterministic power vs. the statistical power for the c1355 v2 circuit. From the plot, it is clear that the p_d and the p_m measures correlate more than the p_d and the $p_{m3\sigma}$. We can measure the correlation between the ranks using Kendall's τ [Ken38]. Sequences that are perfectly correlated (e.g. have the same ranking) have a $\tau = 1$ and sequences that are perfectly anti-correlated (e.g. are reverses of each other) have a $\tau = -1$. The values for the different designs are given in Table 4.6.

From this table, we can see that the p_m rankings are near perfect (≈ 1). This indicates that the optimizations will be very similar. However, the $p_{m3\sigma}$ rankings range from $.5 - .9$. In this case it is not clear from the rankings whether the statistical power is a good surrogate for the deterministic power.

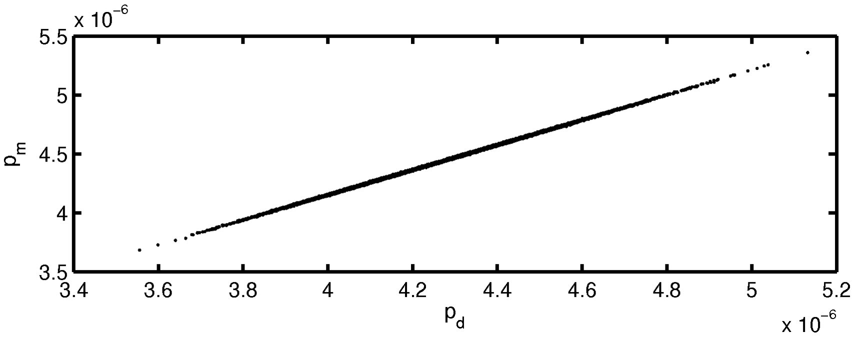
Why the p_m rankings correlate better than the $p_{m3\sigma}$ measures can be seen from Figure 4.3, which plots the power vs. size sensitivities for the p_d, p_m and p_q measures (the $p_{m3\sigma}$ can be thought of as a rough approximation of the p_q measure). Notice that a sorting of the gates by sensitivity would be very similar for the p_d and the p_m measures. However, the sorting by p_q, and hence the relations of the sensitivities of different gates, is very different.

In the analysis above, it is difficult to tell exactly how large the gap will be between the deterministic and statistical sizing solutions. To get a direct idea of the difference between the deterministic power compared to the statistical power, we can make a useful assumption. Suppose we know the value of the minimum deterministic power ($p_d^\star$) that is timing feasible. Then we can estimate the difference between the best statistical sizing and the worst sizing for the given deterministic power as in Figure 4.6(c).

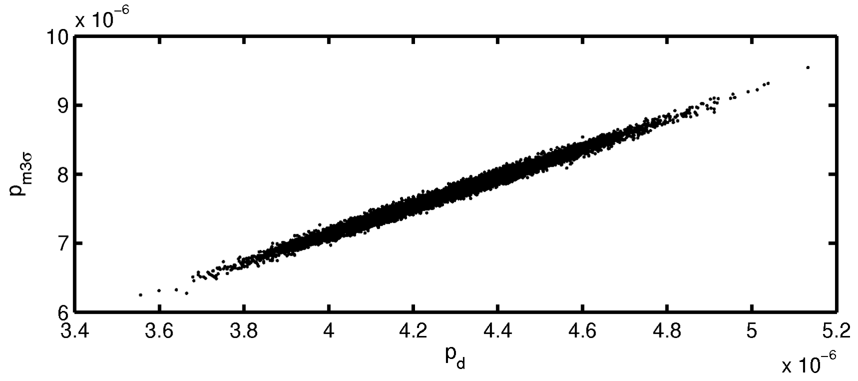
There is inherent error in this process because it depends on the number of samples. As the number of samples grows, the gap will increase as more extreme points are sampled. In the following section, we will compute these bounds without needing to sample the distribution.

4.2.4 Suboptimality bounds

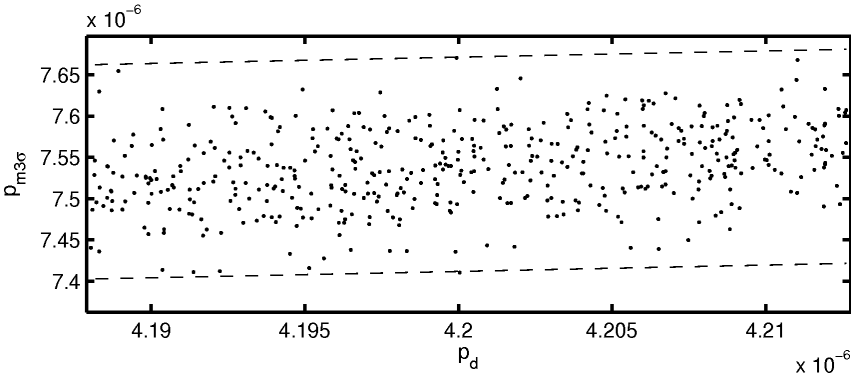
The central question in this section is whether the deterministic power solution is a good approximation for the statistical power optimum. This generally requires information about the space of timing-feasible solutions ($\mathcal{T}$), which is difficult to describe, and is highly problem dependent, making it hard to give an exact answer. However, there is a way to solve a simpler problem with no assumption on the structure of $\mathcal{T}$, instead, relying on the



(a)



(b)



(c)

Figure 4.6: Plot of the deterministic power (x-axis) vs. statistical power (y-axis) for the c1355 v2 circuit for width sizing only. (a) Shows the plots for the p_d vs. the p_m measure, whose relation is nearly linear. In contrast, the p_d vs. $p_{m3\sigma}$ measure in (b) has more noise. In (c), a region of the p_d vs. $p_{m3\sigma}$ plot is chosen around $p_d = 4.2 \cdot 10^{-6}$. The difference in $p_{m3\sigma}$ between the upper and lower bounding lines (approx $2.6 \cdot 10^{-7}$ or about 3.5%) is an estimate of the suboptimality bound. Compare this with the bounds in Table 4.8.

structure of the deterministic power objective.

In this section we consider the following question: suppose we approximate the solution to the statistical power optimization problem:

$$\begin{array}{lll} \text{minimize} & p_s(\vec{w}) & \text{(statistical power)} \\ \text{subject to} & \vec{w} \in \mathcal{T} & \text{(timing constraint} \\ & & \text{ \& discreteness)} \\ & \vec{w} \in \mathcal{B} & \text{(upper and lower)} \\ & & \text{(size bounds on } \vec{w}\text{)} \end{array} \tag{S}$$

using the deterministic power optimum, $\vec{w}_d^\star$, which is the solution to the problem:

$$\begin{array}{lll} \text{minimize} & p_d(\vec{w}) & \text{(deterministic power)} \\ \text{subject to} & \vec{w} \in \mathcal{T} & \text{(timing constraint} \\ & & \text{ \& discreteness)} \\ & \vec{w} \in \mathcal{B} & \text{(upper and lower).} \\ & & \text{(size bounds on } \vec{w}\text{).} \end{array} \tag{D}$$

How good of an approximation will this be, and what is a bound for the suboptimality of this solution?

In the following, we will describe a method for creating suboptimality bounds. First a simple set $\mathcal{T}'$ is constructed that contains $\mathcal{T}$. Optimizing the statistical power over this simpler set will return a lower bound on the statistical power optimum. This lower bound is then compared with the statistical power of the approximate solution, $p_s(\vec{w}_d^\star)$, to bound the accuracy of the approximation. This is described in detail below.

4.2.4.1 Relaxed constraints, enclosing sets and lower bounds

The difficult part of w, l and v_t optimization is the timing constraint and the discreteness constraint. Thus, to find a quick lower bound, we must first relax the timing constraint with a looser constraint. In other words, we would like to *relax* the constraints, by enclosing the timing feasibility and discreteness condition in a simple, convex set.

Relaxing the constraints of a problem turns the resulting solution into a lower bound for the true solution. For example, consider the sets $\mathcal{T}_0 \subseteq \mathcal{T}_1 \subseteq ... \subseteq \mathcal{T}_k$ and the sequence of problems:

$$\begin{array}{lll} (\mathrm{P}_i) & \text{minimize} & p_s(\vec{w}) \\ & \text{subject to} & \vec{w} \in \mathcal{T}_i \\ & & \vec{w} \in \mathcal{B} \subseteq \mathbb{R}^n. \end{array} \tag{4.9}$$

If the optimal solution of problem (P_i) is $\vec{w}_i^\star$, then we have the property that:

$$p_s(\vec{w}_0^\star) \geq p_s(\vec{w}_1^\star) \geq ... \geq p_s(\vec{w}_k^\star). \tag{4.10}$$

In other words the optimal value for the relaxed problem is a lower bound for the original problem.

The intuition for this is the fact that the constraints in the relaxed problem enclose the constraints on the original problem. Thus, the optimal solution in the original problem is also *feasible* for the relaxed problem. In the process of solving the relaxed problem, the solver is free to choose a better point in the larger space, making the resulting optimum a *lower bound* for the original problem.

4.2.4.2 Linear functions, optimum solutions and enclosing sets

For certain classes of functions, it is easy to find a simple set that encloses the optimum. The following analysis will derive a set using the properties of linear functions, but the results also hold for more general functions.[3]

The key to finding an enclosing set for the constraints is to start with an optimal solution and leverage the fact that any other feasible point cannot be better. For example,

[3]Specifically, equation (4.15) holds whenever $p_d(\vec{w})$ satisfies:

$$\{\vec{w} \mid p_d(\vec{w}^\star) \leq p_d(\vec{w})\} \subseteq \{\vec{w} \mid 0 \leq \nabla p_d(\vec{w}^\star)^T(\vec{w} - \vec{w}^\star)\}. \tag{4.11}$$

Mathematically, this is equivalent to saying that $0 \leq \nabla p_d(\vec{w}^\star)^T(\vec{w}-\vec{w}^\star)$ defines a supporting hyperplane for the super-level sets $p_d(\vec{w}^\star) \leq p_d(\vec{w})$. This includes functions that are linear, concave and quasi-concave. This does not necessarily hold for convex functions or posynomial functions, and in these cases, other properties of the timing-feasible region must be assumed for the results in this section to hold.

if $\vec{w}_d^\star$ is optimal for problem (D), then

$$\forall \vec{w} \in (\mathcal{T} \cap \mathcal{B}) : \; p_d(\vec{w}_d^\star) \leq p_d(\vec{w}). \tag{4.12}$$

For linear functions, the inequality on the right side can be rewritten in a simple form. This is because any linear function $f(x)$ can be expressed in the form $f(x) = f(x_0) + s^T(x - x_0)$ (where $s = \nabla f(x)$)[4]. Thus expanding about the minimum $x^\star$ of f gives:

$$f(x^\star) \leq f(x^\star) + s^T(x - x^\star) \tag{4.13}$$

$$0 \leq s^T(x - x^\star) \tag{4.14}$$

Applying this to (4.12) with $s = \nabla p_d(\vec{w}_d^\star)$ shows that

$$(\mathcal{T} \cap \mathcal{B}) \subseteq \mathcal{T}' = \{\vec{w} \mid 0 \leq s^T(\vec{w} - \vec{w}_d^\star)\}. \tag{4.15}$$

Note that $\mathcal{T}'$ is a continuous, connected set. This gives the relaxed problem:

$$\begin{array}{ll} \text{minimize} & p_d(\vec{w}) \\ \text{subject to} & \vec{w} \in \mathcal{T}' \\ & \vec{w} \in \mathcal{B} \subseteq \mathbb{R}^n \end{array} \tag{4.16}$$

4.2.4.3 Creating lower bounds for related problems

The above analysis seems a little circular – the optimum is required to create a lower bound for the optimum. However, the utility emerges when we use the same enclosing sets to find lower bounds for related problems.

Suppose the solution for problem (D) is known, and we would now like to find the lower bound for problem (S), which has a different objective function, but identical constraints. This can be done by leveraging the solution $\vec{w}_d^\star$ for problem (D) to compute a simple, enclosing set for the constraints, as in the subsection above. The relaxed problem is then

[4]Note that s is constant over x for linear functions.

solved:

$$\begin{aligned} \text{minimize} \quad & p_s(\vec{w}) \\ \text{subject to} \quad & \vec{w} \in \mathcal{T}' \\ & \vec{w} \in \mathcal{B} \subseteq \mathbb{R}^n. \end{aligned} \tag{4.17}$$

The problem is solved over the continuous set $\mathcal{T}'$, and the continuous upper and lower bound constraints in $\mathcal{B}$. The corresponding solution $\vec{w}'$ can be used as a lower bound on the true optimum $w_s^\star$ ($p_s(\vec{w}') \leq p_s(\vec{w}_s^\star)$). In the case of $p_s = p_d$ and $p_s = p_q$, this is a linear programming problem, and for $p_s = p_{m3\sigma}$, this is a nonlinear convex optimization problem.

Using this lower bound, we can now find a bound for how well the deterministic solution approximates the solution for the statistical problem. The suboptimality gap between this approximation $\vec{w}_d^\star$, and the true optimum, $\vec{w}_s^\star$ is bounded by:

$$\delta_{\text{so}} = 100 \cdot \frac{p_s(\vec{w}_d^\star) - p_s(\vec{w}')}{p_s(\vec{w}_d^\star)} \tag{4.18}$$

Smaller values indicate that $\vec{w}_d^\star$ is a good approximate solution for $\vec{w}_s^\star$, and larger values indicate that it is a bad approximation. For example, if

$$\delta_{\text{so}} \leq 5\% \tag{4.19}$$

then $\vec{w}_d^\star$ is a 5% approximate solution for $\vec{w}_s^\star$. In other words, using $\vec{w}_d^\star$ in place of the real optimum $\vec{w}_s^\star$, would cost at most 5% (it is *suboptimal* by at most 5%).

This process also works with optimization over w, l and v_{t}. This is exactly similar to the above example, with w replaced by z, the adjusted gate widths.[5]

[5]This gives the problem:

$$\begin{aligned} \text{minimize} \quad & p_s(\vec{z}) \\ \text{subject to} \quad & 0 \leq \nabla p_d(\vec{z}_d^\star)^T(\vec{z} - \vec{z}_d^\star) \\ & z_{min} \leq \vec{z} \leq z_{max} \end{aligned} \tag{4.20}$$

Here, z is a continuous variable, which represents the effect of the widths, lengths and v_{t} on the power. z_{min} and z_{max} are the minimum and maximum values of $\vec{z}$. For example, for the i^{th} entry, they are:
$z_{i,\text{min}} = w_{i,\text{min}} e^{\alpha l_{i,\text{max}}^2 + \beta l_{i,\text{max}}} e^{-\gamma v_{\text{t}_i,\text{max}}}$ and
$z_{i,\text{max}} = w_{i,\text{max}} e^{\alpha l_{i,\text{min}}^2 + \beta l_{i,\text{min}}} e^{-\gamma v_{\text{t}_i,\text{min}}}$.

Interestingly, the actual values of $\vec{l}$, $\vec{w}$ and v_{th} do not play a direct role in the optimization above. They affect the optimization by determining a range for the values of z_i. In fact, the corresponding values of w_i, l_i and v_{t} may not be unique; it is only important that there is a least one combination of w_i, l_i and v_{t} that satisfies $z_i = w_i e^{\alpha l_i^2 + \beta l_i} e^{-\gamma v_{\text{t}_i}}$. Thus, z acts as a proxy for w, l and v_{th} and a corresponding value is not important, as z is used solely to find a lower bound on the power, and not a minimum power configuration.

A surprising fact is that no properties of $p_s(\vec{w})$ are assumed. It may be non-convex or it may be non-linear, as in the case of the $p_{\mathrm{m}3\sigma}$ measure. The only assumption is that the problem (4.17) is solvable.

Another interesting fact is that the deterministic optimum is only used to determine (1) the minimum deterministic power that is timing feasible, and (2) its corresponding statistical power. The delay of the design is thus important in determining the deterministic optimum, but does not play any other role in the bounding process. This highlights the independence of this method on the timing feasible region.

A visual example of the lower bounding process is shown in Figure 4.7. The figure shows that the suboptimality bound is more related to the geometry of the problem than the actual difference between the statistical and deterministic optima.

4.2.5 Experiments

4.2.5.1 Width-sizing experiment

In this subsection we compute suboptimality bounds for width sizing. The bounds (δ_{so}) are computed for each circuit in Section 4.2.2 and the results are presented in Tables 4.7 and 4.8.

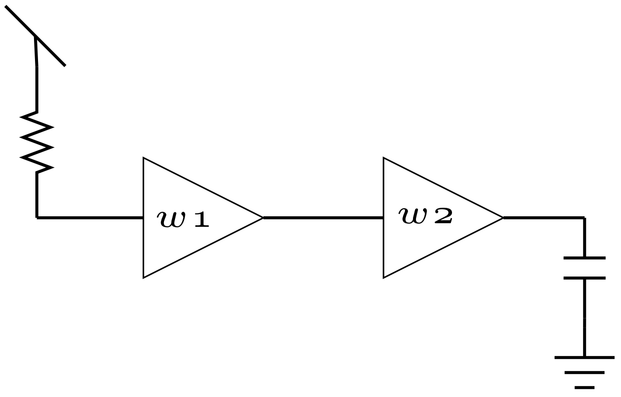
The Nangate Library is modeled using the expression in (4.3). The synthesized widths from the Cadence RTL compiler are assumed to be the optimal deterministic widths[6], $\vec{w}_d^\star$.

These tables show that the bounds grow larger as the proportion of die-to-die variation increases. Note that this is in spite of the fact that the total σ_L per gate is roughly the same, e.g.

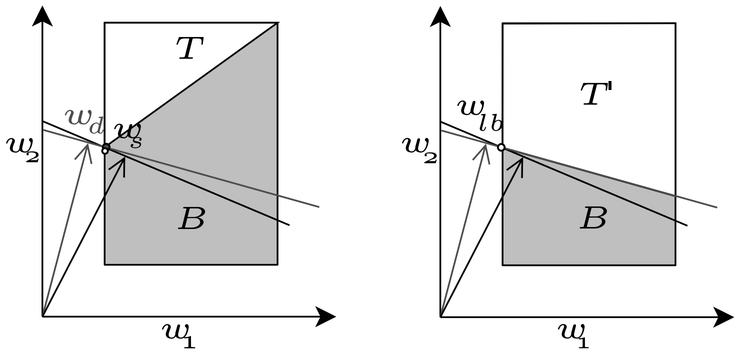
$$\sigma_{L,\mathrm{total}}^2 \approx 1.15^2 + 1.6^2 \approx 2^2. \tag{4.21}$$

This is because the die-to-die variations affect the power more efficiently than within-die variations, which may cancel each other out (e.g. see the expression in (4.4)). The case

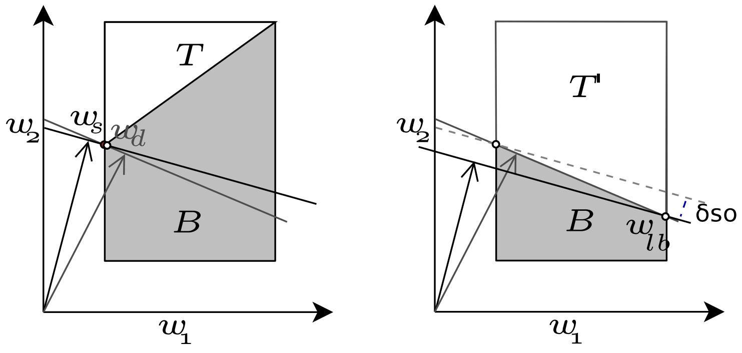
[6]We will drop the assumption that the synthesized widths are optimal in Section 4.2.5.2



(a)



(b)



(c)

Figure 4.7: Suboptimality examples for the two gates in (a). The examples in (b) and (c) have the same optimal solutions for both the statistical and deterministic measures, which means that the actual suboptimality is zero. However, the two examples have different suboptimality *bounds*.

$\sigma_L = 1$nm (dtd)/.5nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)

	p_m					$p_{m3\sigma}$				
	v1	v2	v3	v4	avg	v1	v2	v3	v4	avg
c432	.34%	.31%	.39%	.42%	.37%	3.7%	3.5%	4.5%	4.9%	4.1%
c499	.33%	.33%	.25%	.2%	.28%	3.6%	3.6%	2.7%	2.2%	3%
c880	.28%	.28%	.29%	.29%	.29%	3.1%	3.3%	3.3%	3.2%	3.2%
c1355	.34%	.25%	.29%	.15%	.26%	3.7%	2.8%	3.3%	1.6%	2.8%
c1908	.37%	.36%	.38%	.38%	.37%	4.1%	4.1%	4.2%	4.2%	4.1%
c2670	.43%	.4%	.4%	.39%	.4%	4.6%	4.3%	4.3%	4.1%	4.3%
c3540	.42%	.46%	.42%	.4%	.43%	4.5%	5%	4.6%	4.3%	4.6%
c5315	.34%	.36%	.37%	.36%	.36%	3.7%	3.8%	4%	3.9%	3.8%
c6288	.25%	.26%	.31%	.34%	.29%	2.8%	2.9%	3.5%	3.8%	3.3%
c7552	.39%	.38%	.37%	.37%	.38%	4.2%	4.1%	4%	4%	4.1%
alu	.3%	.27%	.29%	.21%	.27%	3.3%	3%	3.2%	2.4%	3%

$\sigma_L = 1.6$nm (dtd) / 1.15nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)

c432	.64%	.6%	.76%	.83%	.71%	5.9%	6.2%	8.1%	8.7%	7.2%
c499	.63%	.58%	.42%	.35%	.5%	5.7%	5.4%	4%	3.4%	4.6%
c880	.52%	.5%	.52%	.53%	.52%	4.9%	5%	5.2%	5.1%	5.1%
c1355	.65%	.45%	.49%	.25%	.46%	5.9%	4.3%	4.8%	2.4%	4.3%
c1908	.71%	.69%	.71%	.71%	.71%	6.8%	6.9%	7%	7.1%	6.9%
c2670	.82%	.74%	.73%	.72%	.75%	7.4%	6.6%	6.5%	6.4%	6.7%
c3540	.82%	.87%	.79%	.76%	.81%	7.4%	7.9%	7.3%	6.9%	7.4%
c5315	.65%	.67%	.67%	.66%	.67%	5.9%	6.1%	6.1%	6%	6%
c6288	.46%	.47%	.56%	.6%	.52%	4.5%	4.4%	5.3%	5.5%	4.9%
c7552	.72%	.69%	.66%	.67%	.68%	6.5%	6.2%	6%	6%	6.2%
alu	.55%	.5%	.53%	.4%	.49%	5.2%	5.1%	5.2%	4%	4.9%

Table 4.7: Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations

$\sigma_L = 2$nm (dtd)/0nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)										
c432	1.1%	1.1%	1.4%	1.5%	1.2%	8.3%	9.7%	13%	14%	11%
c499	1.1%	.91%	.64%	.54%	.79%	8.1%	7.1%	5.1%	4.5%	6.2%
c880	.85%	.82%	.86%	.88%	.85%	6.8%	7.2%	7.8%	7.4%	7.3%
c1355	1.1%	.73%	.73%	.39%	.74%	8.3%	5.9%	6.1%	3.1%	5.9%
c1908	1.2%	1.2%	1.2%	1.2%	1.2%	9.8%	10%	10%	10%	10%
c2670	1.4%	1.2%	1.2%	1.2%	1.2%	10%	8.8%	8.7%	8.6%	9.1%
c3540	1.4%	1.5%	1.3%	1.2%	1.3%	10%	11%	10%	9.7%	10%
c5315	1.1%	1.1%	1.1%	1.1%	1.1%	8.2%	8.5%	8.3%	8.1%	8.3%
c6288	.76%	.75%	.88%	.91%	.83%	6.4%	6%	7%	7.1%	6.6%
c7552	1.2%	1.1%	1.1%	1.1%	1.1%	8.8%	8.3%	7.9%	8%	8.3%
alu	.9%	.86%	.88%	.7%	.84%	7.3%	7.6%	7.5%	6.4%	7.2%

Table 4.8: Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations

$\sigma_{\mathrm{L}} = 1$nm (dtd) /.5nm (wid) is given for comparison with [CGL09].

Another interesting thing is that adding v_{t} variations does not necessarily increase the suboptimality bound. When the length variations are small, the bounds do tend to increase. However, for large σ_L, the effect of v_{t} variations is unpredictable and small.

4.2.5.2 Assumption free bounds

The biggest weakness of the analyses above is that the deterministic optimum is not available, as the problem is too difficult to solve exactly for real circuits. In this part, we derive bounds that do not rely on an initial deterministic solution. This works by exploiting the geometry of the deterministic power measure.

We begin by assuming that only the value of the deterministic power measure p'_d is known. Thus, the actual deterministic optimum is one of the configurations that has a corresponding power p'_d. Out of these possible configurations, the worst case statistical

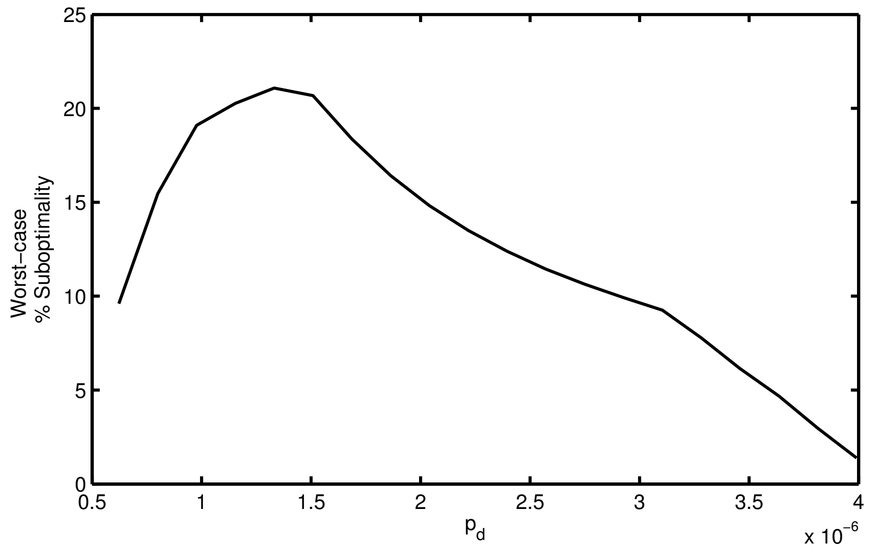


Figure 4.8: The worst-case suboptimality for the example c432 v4 as a function of the assumed deterministic power value p'_d.

power can be found by solving the problem:

$$\begin{aligned} &\text{maximize} && p_s(\vec{w}) \\ &\text{subject to} && p'_d = p_d(\vec{w}) \\ & && w_{\min} \le \vec{w} \le w_{\max}. \end{aligned} \tag{4.22}$$

Denoting the optimal solution of the above as $\vec{w}_{\text{ub}}$, the worst-case suboptimality bounds can be found as:

$$\delta_{\text{wc}} = 100 \cdot \frac{p_s(\vec{w}_{\text{ub}}) - p_s(\vec{w}_{\text{lb}})}{p_s(\vec{w}_{\text{lb}})} \tag{4.23}$$

where $\vec{w}_{\text{lb}}$ is found using equation (4.17). Note that in (4.17), only the value p'_d and not the actual sizes of the deterministic optimum are used. As an experiment, we chose 20 equally spaced values for p'_d (between the minimum power and the maximum power), and computed the worst case suboptimalities using (4.23). Running these examples for the mean + 3σ measure gives the values in Table 4.9, which are approximately 3x-4x the values in Tables 4.7 and 4.8. Note that these numbers are for width sizing only.

It is interesting to note how the suboptimality changes as a function of p'_d. This is shown for the c432 v4 circuit in Figure 4.8. At the minimum value of p'_d, there is only one

	$\sigma_L = 1.6$nm (dtd) / 1.15nm (wid) $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)			
	v1	v2	v3	v4
c432	19.1%	19.5%	21.3%	21.1%
c499	18.1%	15.1%	10.2%	8.9%
c880	17.7%	16.5%	16.4%	15.2%
c1355	18.0%	16.4%	11.6%	6.6%
c1908	20.1%	18.5%	17.4%	17.7%
c2670	15.3%	13.1%	12.3%	12.2%
c3540	16.3%	14.5%	13.7%	13.3%
c5315	13.9%	12.7%	12.1%	11.8%
c6288	18.2%	13.6%	13.5%	9.4%
c7552	15.5%	14.8%	14.3%	14.0%
alu	17.8%	15.4%	14.6%	13.6%

Table 4.9: Worst-case Suboptimality for $p_{m3\sigma}$

possible sizing (e.g. all gates at minimum size), so the suboptimality is zero. Similarly, for the maximum value of p'_d, there is also only one size, so the suboptimality is also zero. For the remainder of the values, the suboptimality grows to a peak at a third of the way, and decreases for the remainder of the values. This implies that the suboptimality is larger for more aggressive designs, and smaller for low power designs.

4.2.5.3 Relating maximum gate size to suboptimality bounds

In this subsection we use expressions (4.2) and (4.3), which model the power as a function of the gate size, to see how the bounds would change if the maximum gate sizes are increased *for all gates.*

Table 4.10 shows how the upper bounds in Tables 4.7 and 4.8 (Section 4.2.5.1) increase

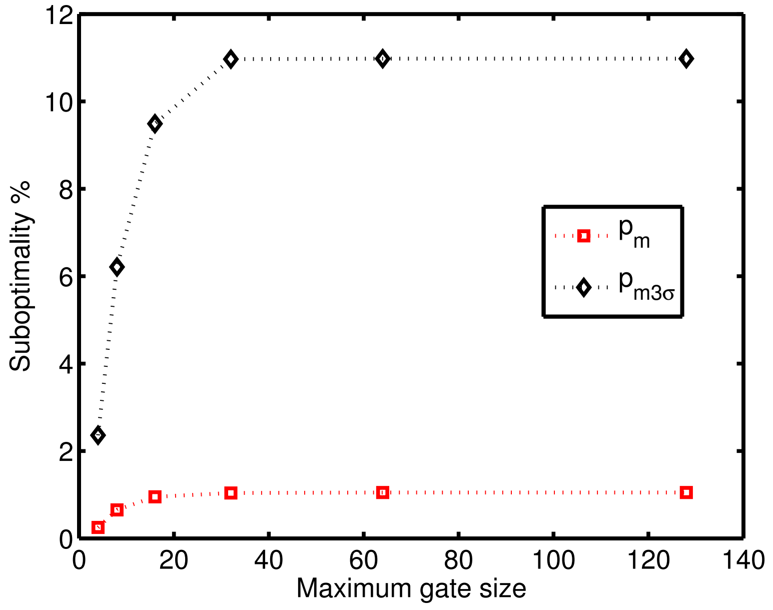


Figure 4.9: The change in the suboptimality as a function of the maximum gate size in c1355 v4. Changing the maximum gate size from 2 to 128 increases the suboptimality by 8.6 percentage points.

when the gate size upper bounds are removed. In this case, the optimal deterministic power value is fixed. The values are given for the case $\sigma_L = 1.6$nm (dtd)/1.15nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid). When compared to Tables 4.7 and 4.8, the suboptimalities for p_m increase by .05 to .75 percentage points, and the suboptimalities for $p_{\mathrm{m}3\sigma}$ increase by .5 to 8.6 percentage points.

The suboptimalities increase because a better lower bound is found when larger gates available in (4.17). This is because the maximum size bounds limit the sizes of the gates with the best deterministic power vs. statistical power tradeoff. As the maximum sizes increase, more of the power is used on the gates with the best tradeoffs. However, once the maximum gate size is large enough to use the entire power budget on the gates with the best tradeoff, the suboptimality will not increase any further.

The relation between suboptimality and the maximum size can be seen in Figure 4.9.

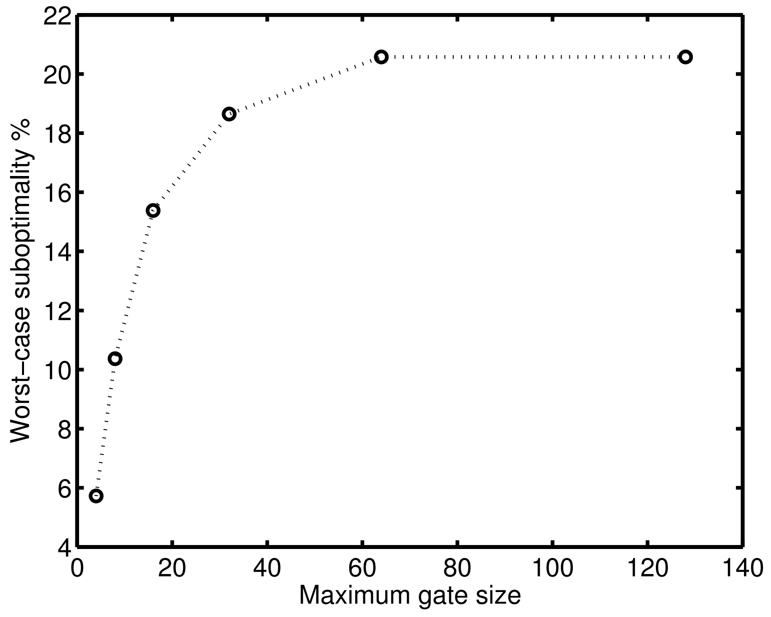


Figure 4.10: The change in the worst-case suboptimality (δ_{wc}) as a function of the maximum gate size in c1355 v4 for the case in Section 4.2.5.2 with the same range of $p_d^\star$ as in Table 4.9. The worst-case suboptimality δ_{wc} reaches its limit of 20.58% for large maximum gate sizes.

The increase in suboptimality from a maximum size of 4 to 8, and from 8 to 16, is approximately 3.5 percentage points. However the increase tapers off – the increase from 16 to 32 is 1.48 percentage points and the increase from 32 to 64 is 0.01 percentage points. The suboptimalities for 64 and 128 are identical. This happens because the maximum size bound no longer plays a role in determining the lower bound[7].

Results for the "assumption free bounds" in Section 4.2.5.2 can also be analyzed to see how the maximum gate size affects the worst-case suboptimality δ_{wc}. Recall that in this case, the bounds are computed for a range of values for $p_d^\star$ and not just one value. Using the same range of values as used in Table 4.9, the suboptimality also reaches a limit as the maximum gate size increases, as in Figure 4.10. The values for a maximum gate size of 64

[7]In the terminology of Section 4.2.4.3, increasing the maximum sizes ceases to change the optimum in (4.17). This is because the optimum has sizes strictly less than the maximum gate size, and therefore the upper bounds do not affect the solution.

$\sigma_L = 1.6$nm (dtd)/1.15nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)										
	p_m					$p_{m3\sigma}$				
	v1	v2	v3	v4	avg	v1	v2	v3	v4	avg
c432	.93%	.74%	.81%	.89%	.84%	9.9%	8.5%	8.8%	9.6%	9.2%
c499	.92%	.88%	.93%	.92%	.91%	9.8%	9.8%	10%	10%	10%
c880	.77%	.7%	.75%	.82%	.76%	8.8%	8.2%	8.6%	9.1%	8.7%
c1355	.94%	.74%	.77%	1%	.88%	10%	8.4%	8.7%	11%	9.5%
c1908	.96%	.9%	.91%	.9%	.92%	11%	10%	10%	10%	10%
c2670	1.1%	1%	1%	.78%	.97%	12%	11%	10%	7.8%	10%
c3540	1.1%	1.2%	1.2%	1.2%	1.2%	12%	13%	13%	13%	13%
c5315	1.1%	1%	1%	1%	1%	11%	11%	11%	11%	11%
c6288	.68%	.72%	.76%	.72%	.72%	8.2%	8%	8.7%	6.6%	7.9%
c7552	1%	.99%	.96%	.97%	.99%	11%	11%	11%	11%	11%
alu	.84%	.79%	.82%	.63%	.77%	10%	9.4%	9.5%	7.4%	9.1%

Table 4.10: Suboptimality (δ_{so}) for Leakage Optimization with l and V_{t} variations and no size upper bounds

and 128 are identical.

The explanation for this trend is similar to the case in Figure 4.9. However, in this case, an upper bound is also computed in (4.22). This upper bound can also take advantage of the increasing maximum gate size by enabling it to use more of the power on the gates with the *worst* deterministic power vs. statistical power tradeoff.

It is important to note that in realistic libraries, gate sizes will not go beyond 64X, with most cells being limited to 16X. As a result, numbers presented in this subsection are somewhat exaggerated.

4.2.5.4 Suboptimalities for w, v_t and l sizing

In this section, we extend these results to w, v_t and l assignment. Because we do not have designs with optimal w, v_t and l assignment (and hence an optimal $z^\star$), we follow a methodology that is similar to Section 4.2.5.2. 10 equally spaced values of the deterministic power are chosen ($\{p_{d_1}, ..., p_{d_{10}}\}$), where p_{d_1} is the power with all gates at the minimum power cell, and $p_{d_{10}}$, is the power with all gates at the maximum power cell. For each deterministic power, a corresponding lower bound is found (z_lb, as in the prior sections) and a corresponding upper bound is found (z_ub). The lower bound is computed by using a variation of (4.20) that uses a given deterministic power value p_{d_i}:

$$\begin{array}{ll} \text{minimize} & p_s(\vec{z}) \\ \text{subject to} & p_{d_i} \leq p_d(\vec{z}) \\ & z_\mathrm{min} \leq \vec{z} \leq z_\mathrm{max} \end{array} \tag{4.24}$$

The upper bound is computed by using a variation of (4.24) as a *maximization* problem:

$$\begin{array}{ll} \text{maximize} & p_s(\vec{z}) \\ \text{subject to} & p_{d_i} = p_d(\vec{z}) \\ & z_\mathrm{min} \leq \vec{z} \leq z_\mathrm{max} \end{array} \tag{4.25}$$

The latter problem finds the maximum statistical power configuration that has a deterministic power equal to p_{d_i}. Thus, we find the maximum statistical power that can also be an optimal deterministic power solution. Combining the two results, the max suboptimality is computed as:

$$\delta_\mathrm{max} = 100 \cdot \frac{p_s(\vec{z}^\star_\mathrm{ub}) - p_s(\vec{z}_\mathrm{lb})}{p_s(\vec{z}_\mathrm{lb})} \tag{4.26}$$

This results in a more conservative bound than in Section 4.2.5.1, because a worst case lower bound *and* a worst case upper bound are computed. These values are computed for width sizing ($\delta_{w,\mathrm{max}}$), and full w, v_t and l assignment ($\delta_\mathrm{full,max}$). The ratios $\delta_\mathrm{full,max}/\delta_{w,\mathrm{max}}$ are shown in Table 4.11.

The ratios in Table 4.11 can be used to relate the suboptimality bounds from Section 4.2.5.1 to the case of full w, v_t and l assignment. This table indicates that the values

	$\sigma_L = 1.6$nm (dtd) /1.15nm (wid) $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)				
	v1	v2	v3	v4	avg
c432	2.1	1.9	1.7	1.7	1.8
c499	1.7	1.8	2.9	3.6	2.5
c880	2	1.9	2	1.8	1.9
c1355	1.7	1.6	2.5	4.3	2.5
c1908	2	1.9	1.8	1.8	1.9
c2670	1.8	1.9	2	2	2
c3540	2.1	2	2.1	2.2	2.1
c5315	2.3	2.1	2.2	2.2	2.2
c6288	2	1.9	2.5	3	2.4
c7552	2	1.8	2	1.9	1.9
alu	1.8	1.7	2.1	1.4	1.7

Table 4.11: Suboptimality ratios $\delta_{\text{full,max}}/\delta_{w,\text{max}}$

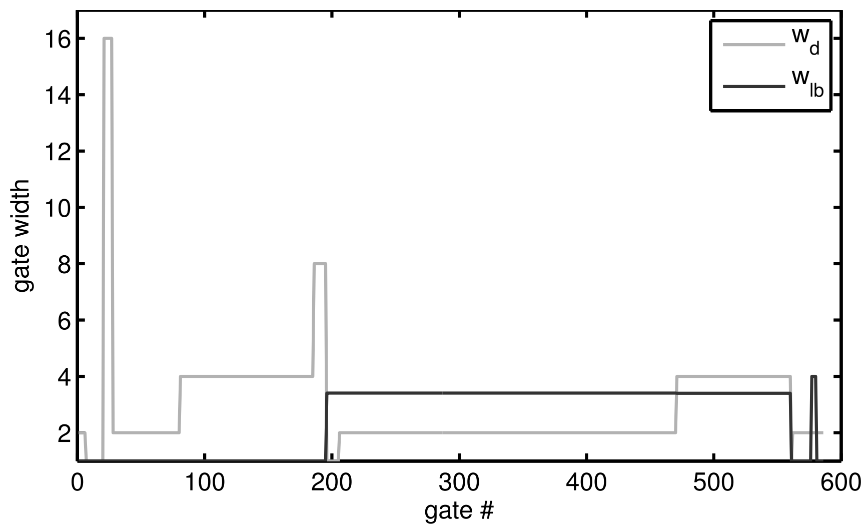


Figure 4.11: The difference between the sizes that are synthesized for deterministic power (w_d), and the sizes that are used to compute the lower bound (w_{lb}). The c1355 v2 circuit is shown.

will roughly double in most of the cases, but may increase up to 6.6x.

Note that in the process above, w, l and v_{th} are not used. This is because the effect on the power is central to the bounding process, and the effect of w, l and v_{th} on the power can be summarized by the variable z. Using the actual w, l and v_{th} cannot be done as p_d is not quasi-concave in w, l and v_{th}.

4.2.5.5 How conservative are these bounds?

A natural question to ask about the bounds above is “How conservative are they?” This is because the bounds for the larger length variations are significant, and it may be useful to perform *statistical optimization* to see what the actual suboptimality is for those cases.

One indication of how conservative these bounds are comes from looking at the sizes that are used to compute the lower bound. In Figure 4.11, the sizes of the deterministic optimum are plotted against the sizes that are used to compute the lower bound. The difference in the sizings is large – many of the minimum sized gates become large gates, and vice-versa.

The effect of flipping the gates will generally violate timing. Gates are usually sized larger because they need to drive larger fanouts, and smaller gates are smaller because they have smaller fanouts. Thus, by flipping the sizes of these gates, small gates will need to drive larger fanouts, and there will be large gates with small fanout load.

Another intuition can be gained by re-examining Figure 4.7. In both cases (b) and (c), the actual suboptimality is 0%. However, due to the geometry of the sets and the sensitivities, the suboptimality *bounds* are very different. This indicates that the bounds may be very loose in some cases.

A final comment is related to the mathematical procedure used to create the bounds. The space was relaxed to be continuous, and the lower bound is found by finding a continuous sizing that is a bound. Adding in the discreteness constraints can only make the suboptimalities smaller.

To test the bounds, we made several small circuit examples:

- 7 gate fanout tree: 1 primary input gate, 4 primary output gates
- 7 gate fanin tree: 4 primary input gates, 1 primary output gate
- 9 gate diamond: 1 primary input gate, 1 primary output gate, with 2, 3 and 2 gates at logic depth 2, 3, 4, respectively
- 13 gate star: (there is one gate in the middle at depth 3, and the fanout and fanin cones are both trees with root at this node)
- 14 gate, 2-bit adder

We randomly assigned the gates to these examples, and used enumeration to compute the suboptimality. A variation of $\sigma_L = 1.6$nm (dtd)/1.15nm (wid) and $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid) is used. After running an optimization however, we found that all of these designs have suboptimality of 0%.

To find examples with suboptimalities we randomly generated circuits according to the following rule:

1. With an N gate circuit
2. j is a fanout of i with probability p if $j > i$, probability 0 otherwise.

We used $N = 10$ and $p = .5$. All the possible width combinations were enumerated, and the best designs for a series of delay targets were recorded. Figure 4.12 shows one example case. The statistical and deterministic optima are the same for the majority of the cases, but they depart for a few different delay targets. In this case, the maximum suboptimality is 10.1%. This example circuit has logic depth 5, and the non-primary output gates have fan-outs of $\{6, 5, 5, 4, 3, 2, 1\}$, respectively.

We generated 500 of these random circuits, and computed the worst-case suboptimality for each circuit across all delay targets. Most of the circuits that were generated have 0% suboptimality (see the histogram in Figure 4.13), however the worst-case suboptimality was 16.6% which correlates with the numbers in Table 4.9. This shows that the suboptimality is likely to be 0%, although the worst-case numbers may be significant.

4.2.6 Bridging the suboptimality gap

The suboptimality bounds for large l variations cause some reason for concern. Some of the bounds are over 10%, which is a significant amount that is too large to ignore. In this section we present ways to bridge the suboptimality gap using simpler measures.

There is a significant cost to using the $p_{m3\sigma}(\cdot)$ measure. It is significantly more complicated than the other power measures. While the deterministic ($p_d(\cdot)$) or the mean ($p_m(\cdot)$) power measures are linear in z, the $p_{m3\sigma}(\cdot)$ is nonlinear in z (although it is convex in z). Linear measures have the advantage that the total power is the sum of the individual powers. Thus, these measures can be used in existing optimization methods by replacing the power values in the library files with the statistical power values.

Fortunately, there is a simple linear approximation that we can use to bridge the sub-

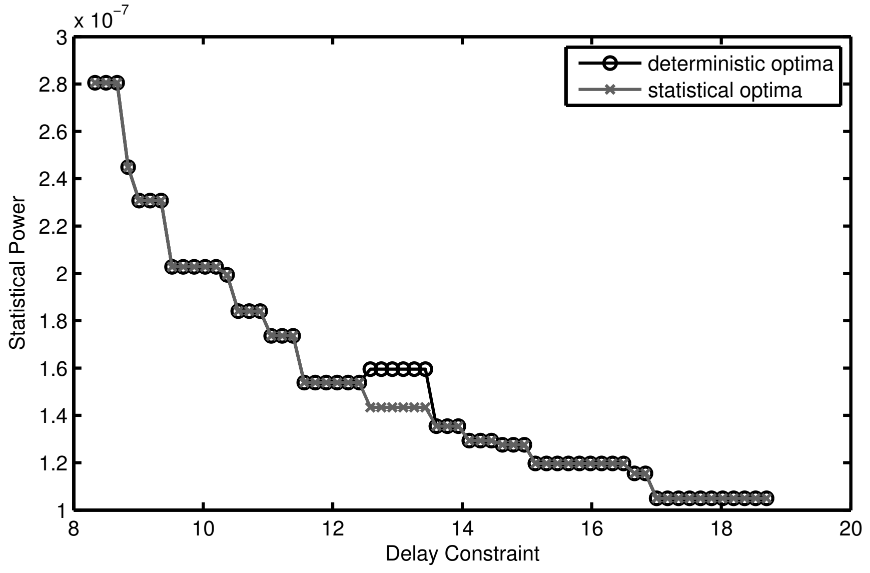


Figure 4.12: Suboptimality plot for a 10 gate circuit (computed by enumeration). The x-axis is the delay constraint, while the y axis is the statistical power. The 'x' denote statistical minima and the 'o' denote deterministic optima. When the black and gray lines diverge, there is a suboptimality gap, which is the difference between the two lines. The maximum gap in this plot is 10.1%. Note that for the majority of the delay constraints, the deterministic and statistical optima are the same.

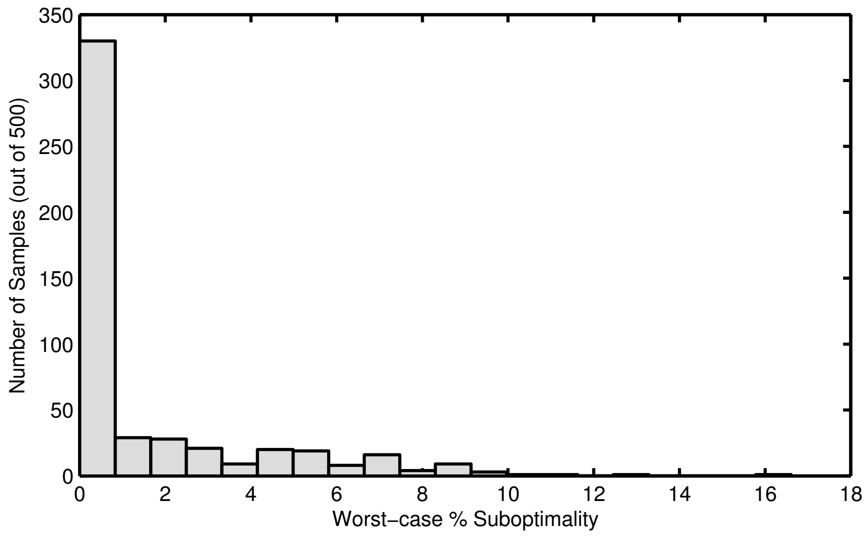


Figure 4.13: Histogram of the suboptimalities for 500 randomly generated circuits. The majority of the examples have suboptimality of 0%.

optimality gap. The idea is to use the variation information to construct a linear measure that has a low suboptimality gap. This would result in a measure that is a provably good approximation to the full statistical optimization, and can be used by replacing the deterministic power values by these approximate values. We define this approximation to the $p_{m3\sigma}$ measure as:

$$p_{\text{approx}} = \sum \kappa_i \exp\Big(\lambda_{\text{w}(i)}\sigma_{\mathbf{L}_{\text{w}(i)}} + \lambda_{\text{d}(i)}\sigma_{\mathbf{L}_{\text{d}}} + ... \quad \sqrt{2}(\tau_{\text{w}(i)}\sigma_{\mathbf{v}_{\text{t}_{\text{w}(i)}}} + \tau_{\text{d}(i)}\sigma_{\mathbf{v}_{\text{t}_{\text{d}}}})/2\Big) \tag{4.27}$$

The derivation of this approximation is purely empirical– this measure was found to give very small suboptimality bounds. It can be interpreted as using the 1-σ value of the random variables $\mathbf{v}_{\text{t}_{\text{w}(i)}}$, $\mathbf{v}_{\text{t}_{\text{d}}}$, $\mathbf{L}_{\text{w}(i)}$, and $\mathbf{L}_{\text{d}}$. The $\sqrt{2}/2$ factor used to account for intra-die cancellation. This measure performs better than the p_m measure and it is also better than using the 3-σ value of the random variables.

Worst-case suboptimality bounds for this approximation are given in Table 4.12 for full w, v_{th}, and l assignment (as in Section 4.2.5.4), 1-32x sized gates (as in Section 4.2.5.3), and the same range for $p_d^\star$ as used in Table 4.9. This measure is very effective, as it

$\sigma_L = 1.6$nm (dtd) / 1.15nm (wid) $\sigma_{\mathrm{V_t}} = 4.7\%$ (dtd & wid)					
	v1	v2	v3	v4	avg
c432	2.2%	1.9%	1.6%	1.4%	1.8%
c499	2.0%	2.1%	2.0%	1.4%	1.9%
c880	2.1%	2%	2%	1.9%	2.0%
c1355	2.1%	2.2%	1.7%	1.6%	1.9%
c1908	2.2%	2.2%	2.2%	2.2%	2.2%
c2670	1.9%	1.7%	1.6%	1.6%	1.7%
c3540	2.1%	2.2%	2.1%	2.1%	2.1%
c5315	1.8%	1.7%	1.7%	1.6%	1.7%
c6288	2.1%	3.1%	2.7%	1.4%	2.3%
c7552	1.7%	1.6%	1.6%	1.6%	1.6%
alu	2.2%	3.5%	3.0%	3.5%	3.0%

Table 4.12: Worst-case suboptimality (δ_{wc}) of p_{approx} as an estimate of $p_{m3\sigma}$ for a maximum size of 32, and w, l, v_{th} assignment

reduces the suboptimality gap to 3.5% and less. This is around an 80% reduction in the suboptimalities. This indicates that although the $p_{m3\sigma}$ measure may be different from the deterministic measure, the sensitivities can be well approximated by a linear measure.

In contrast p_m is not as good of a linear approximation to $p_{m3\sigma}$. The resulting suboptimality bounds give about a 10% reduction in the suboptimalities.

4.2.7 Summary

In this section we compared deterministic solutions of sizing problems with statistical solutions of sizing problems. The rankings of the solutions coincide very well for the mean power measure, indicating that good deterministic solutions will be equally good mean power solutions. However, the other measures have significant discrepancy in their rank. These findings were substantiated by computing the worst-case bounds on the suboptimality gap. The bounds are always insignificant for the mean-power measure p_m, but they may become significant for the $p_{m3\sigma}$ measure when the inter-die component of the length variations is large. As a way to bridge the suboptimality gap, we present a proxy measure for the $p_{m3\sigma}$ measure which is an excellent approximation, and has an insignificant suboptimality.

4.3 Statistical delay optimization

The related question of statistical delay optimization vs. deterministic delay optimization has been studied in [Naj05, GVZ05, BKM07]. In [BKM07] the authors quantify the difference between corner based methodologies and full statistical optimization methods. They find that with a 5% variation in stage delay, the full-blown statistical analysis and optimization gives a mere 2% improvement, and a 12% variation gives a 6% improvement over a statistical worst-case corner that employs a guardband. In [GVZ05], the tradeoff between yield and circuit delay, and the improvements in slack are examined. Significant improvements are shown for a set of benchmark circuits. This shows that methods that use statistical delay provide significant benefits over deterministic methods.

In this section we use circuit sizing to improve the timing robustness of a design. This is not a new area, and several approaches to this problem already exist. In [DS06] the *bin-yield loss function*, defined as the expected loss or penalty for circuits that exceed a given delay threshold, is minimized using stochastic optimization. Other groups have used slack redistribution to reallocate the statistical slack of each of the paths [SSZ05]. The most popular method in literature estimates a worst-case scenario for each gate, and then sizes the circuit according to this estimate [MO04, SNL05, PYK05]. These methods add a "padding" that is proportional to the standard deviation of the gate delay. For the sake of simplicity, these methods will be referred to as "padded delay methods." The deficiency of the padded delay methods is that it uses a conservative estimate and it cannot distinguish between correlated and uncorrelated variations.

In this chapter we propose the *Mean Excess Delay* (MED) as a statistical measure of the delay in the presence of parameter variations. This measure is used in the finance industry to minimize the risk associated with the value of an investment portfolio [RU00]. In the context of circuits, we show that MED is a convex function of the logarithm of the gate sizes, and therefore well-suited for minimization. We also discuss a numerical algorithm for mean excess delay gate sizing, and present some encouraging numerical results.

In summary there are two main contributions in this section:

1. The introduction of the Mean Excess Delay as a statistical measure of circuit delay. The mean excess delay preserves the convexity of the underlying delay model, and is therefore well-suited for minimization.
2. Numerical results that compare the padded delay method with the mean-excess delay method for gate sizing.

The remainder of the section is organized as follows. Section 4.3.1 gives a background on the circuit sizing problem in its nominal and statistical forms. In Section 4.3.2 we introduce the Mean Excess Delay function, and explain its mathematical properties. Section 4.3.3 briefly outlines the minimization algorithm we use. Results are shown in Section 4.3.4.

4.3.1 Statistical delay models

In the circuit sizing problem, the sizes of each gate are selected to minimize the delay of a circuit with constraints on the power and area.

4.3.1.1 The nominal case

In the nominal case, the variations in delay are ignored, resulting in the following problem

$$\begin{array}{ll} \text{minimize} & T^{\text{nom}}(x) \\ \text{subject to} & A(x) \leq A_{\max} \\ & P(x) \leq P_{\max} \\ & x \geq 0. \end{array} \tag{4.28}$$

Here, the optimization variable x is a vector of the log-gate sizes (or more accurately, the log of the normalized gate scaling factors). The functions $T^{\text{nom}}(x)$, $A(x)$ and $P(x)$ represent the nominal delay, area and power of the circuit as a function of the log-gate sizes x. We assume the functions $A(x)$, $P(x)$ and $T^{\text{nom}}(x)$ are posynomials in convex form [BV04]. This problem has been studied extensively, and can be solved efficiently via geometric programming [CCW99, JB08]. For a good tutorial on circuit optimization via geometric programming, see [BKP05].

4.3.1.2 The statistical delay

The effects of the process variations on the delay are often modeled as follows [DS06]:

$$d_k(x, v) = (1 + v_k)\, d_k^{\text{nom}}(x). \tag{4.29}$$

In the above, $d_k^{\text{nom}}(x)$ is the nominal delay, $d_k(x, v)$ is the gate delay with the process variations, and v is a vector of zero-mean random variables with the same dimension as x

(v_k denotes the k^{th} element of the vector v). The random variables v_i are not restricted to be independent or Gaussian, and may have correlations that are "gate by gate", "die-to-die", or by location. This model preserves convexity of $d_k^{\mathrm{nom}}(x)$. That is, if $d_k^{\mathrm{nom}}(x)$ is convex, then $d_k(x, v)$ will be convex as well for fixed v.

Note that the distribution and the correlations of the random variable are unrestricted, making this model general enough to handle a large class of variations. For example, correlations between the gate length or doping can be included in the random variable v_i. The only restriction is that the effect on the delay is multiplicative, and the standard deviation of the variations is independent of the size of the gate. This last restriction is discussed below.

A modification of this model makes the variance of the random variables v_i a function of the corresponding gate size:

$$d_k(x, v) = (1 + v_k e^{-\alpha x_k})\, d_k^{\mathrm{nom}}(x) \tag{4.30}$$

where v_k is a random variable that is scaled by the function $e^{-\alpha x_k}$. Here α is a modeling parameter that is 0.5 in Pelgrom's model [PDW89] or 0.3 according to recent work in [WAN07]. The intuition behind this model is that the saturation current I_{sat} is better controlled as gate is made larger. Note that if $v_k \geq 0$ then this $d_k(x, v)$ is convex for fixed v. However, if $v_k < 0$, the expression is not convex, and does not have a posynomial representation.

To distinguish between the two models, we will call model (4.29) the size independent variation model, and model (4.30) the size dependent variation model. The net effect of both models above is to make the total delay of the circuit a function of the gate-scaling factors *and* the random variables. This turns the total delay itself into a random variable.

4.3.1.3 The statistical sizing problem

To convert the nominal problem (4.28) into one that minimizes the statistical delay, we need to decide what it means to minimize a random variable. One obvious definition is to focus on the β-quantile $q_\beta(x)$, defined as:

$$q_\beta(x) = \inf\{t \mid \mathbf{Prob}\,(T(x,v) \le t) \ge \beta\}.$$

In the context of circuits, the β-quantile is the delay specification that will give a yield of β-percent.

Reformulating the deterministic problem with a β-percent yield gives the following problem

$$\begin{array}{ll} \text{minimize} & q_\beta(x) \\ \text{subject to} & A(x) \le A_{\max} \\ & P(x) \le P_{\max} \\ & x \ge 0. \end{array} \tag{4.31}$$

This problem, however, is difficult to solve. First, the cost function q_β is difficult to evaluate, because in practice there is no closed-form expression for the distribution of $T(x,v)$, although there are many ways of approximating it [VRK04, CZN05, WKO06, ZSL05]. A further difficulty is that the problem (4.31) is generally not convex, even when the nominal problem (4.28) is convex.

4.3.2 The Mean Excess Delay

The β-Mean Excess Delay (β-MED) is a measure that is closely related to the β-quantile. For continuous distributions, it is the expected value of the tail of the delay, when the tail is measured past the β-quantile.

$$m_\beta(x) = \mathbf{E}\left[T(x,v) \,|\, T(x,v) \ge q_\beta(x)\right]. \tag{4.32}$$

A graphical interpretation of the mean excess delay is shown in Fig. 4.14. In this graph, we plot the probability density function (pdf) of $T(x, v)$. $q_{0.95}(x)$ is the 95% quantile, and the shaded area to the right is the tail of $T(x, v)$. The center of mass of this shaded region is the mean excess delay of the distribution, $m_{0.95}(x)$.

This measure is used in the finance industry to manage risk. In this context, $T(x, v)$ is the loss of a portfolio, q_β is the value-at-risk, and m_β is the conditional value-at-risk [RU00].

4.3.2.1 Properties of the Mean-Excess Delay

By definition, $m_\beta(x) \geq q_\beta(x)$, *i.*e. , the β-mean excess delay is an upper bound on the β-quantile. Thus, minimizing mean excess delay will indirectly minimize the β-quantile, and it makes sense to consider the problem

$$\begin{array}{ll} \text{minimize} & m_\beta(x) \\ \text{subject to} & A(x) \leq A_{\max} \\ & P(x) \leq P_{\max} \\ & x \geq 0. \end{array} \tag{4.33}$$

The important feature of this measure is that it can be minimized via convex optimization, if $T(x, v)$ is convex in x for fixed v [RU00].

To show this, we define the function

$$g_\beta(x, t) = t + \frac{1}{1-\beta}\mathbf{E}_v[T(x, v) - t]^+ \tag{4.34}$$

where

$$[u]^+ = \begin{cases} u & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and $\mathbf{E}_v$ is expectation with respect to v, *i.*e. , if v is a continuous random vector in $\mathbb{R}^n$ with pdf $p(v)$,

$$\mathbf{E}_v[T(x, v) - t]^+ = \int_{v \in \mathbb{R}^n} [T(x, v) - t]^+ p(v) dv. \tag{4.35}$$

It follows from standard properties of convex functions that $g_\beta(x,t)$ is convex in x and t if $T(x,v)$ is convex in x for fixed v [BV04]. We can also note that for fixed t, the function (4.35) is the *binning yield-loss* discussed in [DS06], *i.e.* , a linear penalty on delays that exceed a timing budget. However, although the mean-excess delay and the binning yield-loss achieve different goals, they can be implemented using similar algorithms.

The following theorem, due to Rockafellar and Uryasev [RU00], relates (4.34) to the mean-excess delay.

Theorem 4.3.1. *If v is a continuous random vector, then the minimum value of $g_\beta(x,t)$ over t is equal to the mean-excess delay:*

$$m_\beta(x) = \inf_{t\in\mathbb{R}} g_\beta(x,t).$$

In effect, this theorem states that the mean-excess delay can be minimized by minimizing the function $g_\beta(x,t)$ over the design variables x and the additional variable t.

Another interesting fact is that the minimizer of (4.34) over t is the β-quantile of the distribution $T(x,v)$:

$$q_\beta(x) = \inf\{t \mid g_\beta(x,t) = m_\beta(x)\}$$

This gives us the resulting quantile for free as a by-product of the optimization.

In summary, for continuous distributions, we have the following properties:

1. $q_\beta(x) \le m_\beta(x)$
2. $m_\beta(x) = \inf\limits_t g_\beta(x,t)$
3. $q_\beta(x) = \inf\{t \mid g_\beta(x,t) = m_\beta(x)\}$
4. $g_\beta(x,t)$ is jointly convex in t and x if $T(x,v)$ is convex in x.

These properties are illustrated in Fig. 4.14. Here, the minimum of $g_{0.95}(x,t)$ is $m_{0.95}(x)$ with a minimizer $q_{0.95}(x)$.

In the case of a discrete distribution, properties 1, 3 and 4 hold. However in this case, the mean-excess delay is defined to be $\inf_t \; g_\beta(x,t)$, and it can still be used as an upper bound on the quantile.

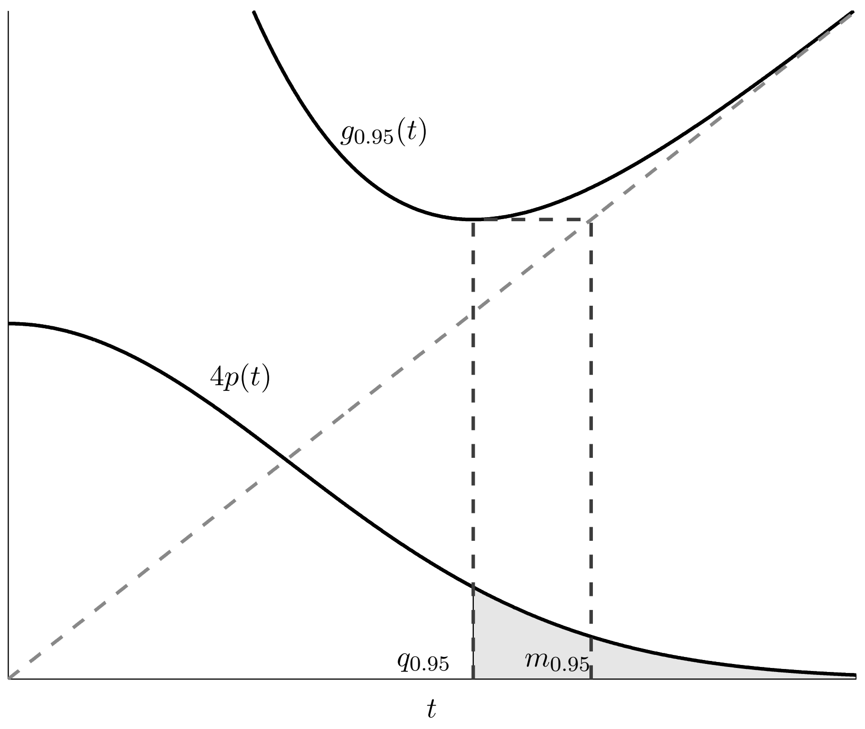


Figure 4.14: The pdf of the delay $T(x, v)$ for a Gaussian distribution is plotted above. $q_{0.95}(x)$ is the 95% quantile, and $m_{0.95}(x)$ is the 95% mean excess delay. The function $g_{0.95}(t)$ is plotted to illustrate the properties in Section IIIA. The minimizer of $g_{0.95}(t)$ is $q_{0.95}$, and the minimum value, which is reflected across the diagonal line, is $m_{0.95}$.

Using the properties above, we can now reformulate (4.33) as:

$$\begin{array}{ll} \text{minimize} & g_\beta(x,t) \\ \text{subject to} & A(x) \le A_{\max} \\ & P(x) \le P_{\max} \\ & x \ge 0 \end{array} \tag{4.36}$$

with variables x and t.

Because the preceding discussion is not unique to circuit sizing, the mean-excess delay can be applied to other problems where the delay is a convex function of the design variables, for a fixed variation. Furthermore, the same type of analysis can be used to combine power and delay minimization.

4.3.3 Minimization algorithm

In the case of a discrete probability distribution and model (4.29), problem (4.36) can be expressed as a geometric program. This can be solved using a general purpose solver such as Mosek [MOS], but for large problems and a large set of discrete variations, a specialized algorithm must be used. The case of continuous distributions can be approximated as a discrete case by taking a fixed number of samples. This is referred to as a *Sample Average Approximation* (SAA) of the problem.

We solve the sample average approximation of the problem (4.36) using the Analytic-Centering Cutting Plane Method (ACCPM) [AV95]. In this method, we start with a polyhedral set S_0 that contains the optimum gate size. At each step of this method, the approximate "center" of the polyhedron is found, and the set of possible solutions S_k is reduced by adding a *cutting plane* through this center. The cutting plane *cuts away* the part of the region where the optimum cannot lie, reducing the set of possible solutions. In the ACCPM, this is achieved as follows.

1. Find the analytic center x_k^c of the current set of possible solutions S_k using Newton's method

	c432		c499		c1355		c1908	
Method	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}
Nominal	3.9ns	2.9ns	.73ns	.50ns	.79ns	.57ns	1.9ns	1.3ns
Padded	3.9ns	2.9ns	.73ns	.50ns	.79ns	.57ns	1.9ns	1.3ns
MED	3.9ns	2.9ns	.68ns	.51ns	.72ns	.58ns	1.8ns	1.3ns

	c2670		c3540		c5315		c7552	
Method	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}
Nominal	.82ns	.63ns	1.5ns	1.0ns	1.1ns	.89ns	.96ns	.77ns
Padded	1.3ns	1.1ns	1.5ns	1.0ns	1.1ns	.89ns	.95ns	.77ns
MED	.79ns	.69ns	1.4ns	1.0ns	1.0ns	.93ns	.93ns	.83ns

Table 4.13: Sizing Results - Size independent variations

2. Evaluate the gradient $\nabla g_\beta(x,t)$ of the Mean-Excess Delay at the point x_k^c
3. Use the cutting plane to reduce the set of possible solutions:

$$S_{k+1} = S_k \cap \{x \mid \nabla g_\beta(x,t)(x - x_k^c) \leq 0\}.$$

The analytic center of a set of linear inequalities is defined as the minimizer of the log-barrier function:

$$x_c = \operatorname*{argmin}_x \left\{ -\sum_{i=1}^{m} \log(b_i - a_i^T x) \right\}$$

where the vectors a_i and scalars b_i define the cutting planes. This centering process is done in an effort to maximize the region that will be cut away by the cutting plane.

4.3.4 Results

We sized the ISCAS '85 benchmark circuits using 3 different sizing methods:

1. Nominal Sizing: variations are ignored
2. Padded Delay Sizing: a conservative robust sizing method [PYK05]
3. Mean Excess Delay Sizing.

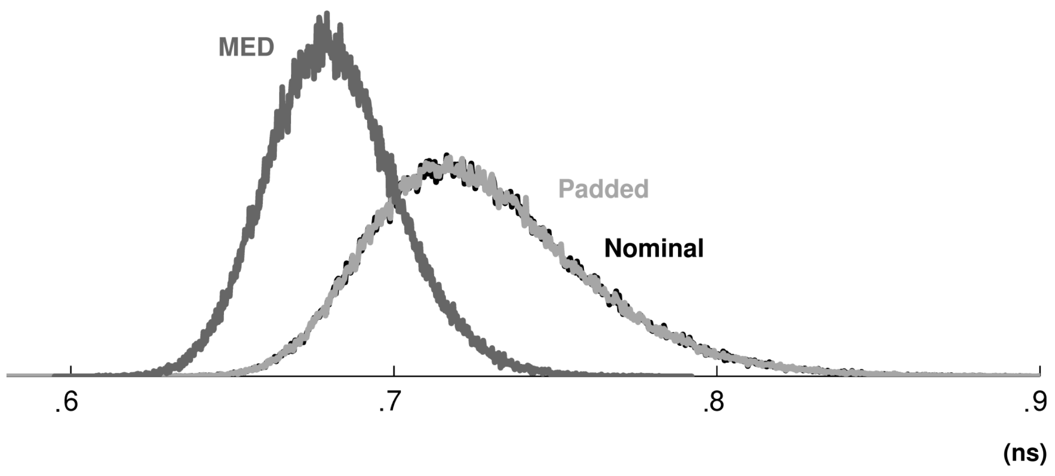


Figure 4.15: The delay pdfs of benchmark c1355 are plotted for each of the three sizing methods. The MED sizing gives the best pdf, followed by the padded delay sizing and the nominal sizing, which are nearly identical.

The standard deviations of each random variable were set to 0.25 and are split evenly between the "die-to-die" and "gate-by-gate" variations. These variations are made to be independent of each other, and they are assumed to be Gaussian. The nominal and padded delay sizing problems were solved using the general purpose commercial solver MOSEK [MOS], and the mean-excess delay sizing was done using the ACCPM with 2000 samples of the distribution.

With the size independent variations in (4.29), the Mean-Excess Delay Sizing always results in a better quantile than the nominal or padded delay sizings, but the magnitude of this difference depends on the structure of the circuit. This is summarized in Table 4.13. Here, $q_{0.95}$ denotes the 95% quantile and d_{nom} denotes the delay associated with the nominal case, where the variations are ignored. In the case of c432, the numbers were nearly identical for each of the three sizings, however in the case of the larger c1355, the difference was approximately 10%.

With the gate-size dependent variations in (4.30) the problem is no longer convex, and it loses the tractability that is associated with convex problems. However, a few runs

	c880		c1355		c2670		c3540	
Method	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}	$q_{0.95}$	d_{nom}
Nominal	.86ns	.49ns	.73ns	.56ns	.70ns	.64ns	1.2ns	1.1ns
Padded	.73ns	.50ns	.62ns	.58ns	.67ns	.66ns	1.1ns	1.1ns
MED	.73ns	.50ns	.61ns	.58ns	.66ns	.64ns	1.1ns	1.1ns

Table 4.14: Sizing Results - Size Dependent Variations

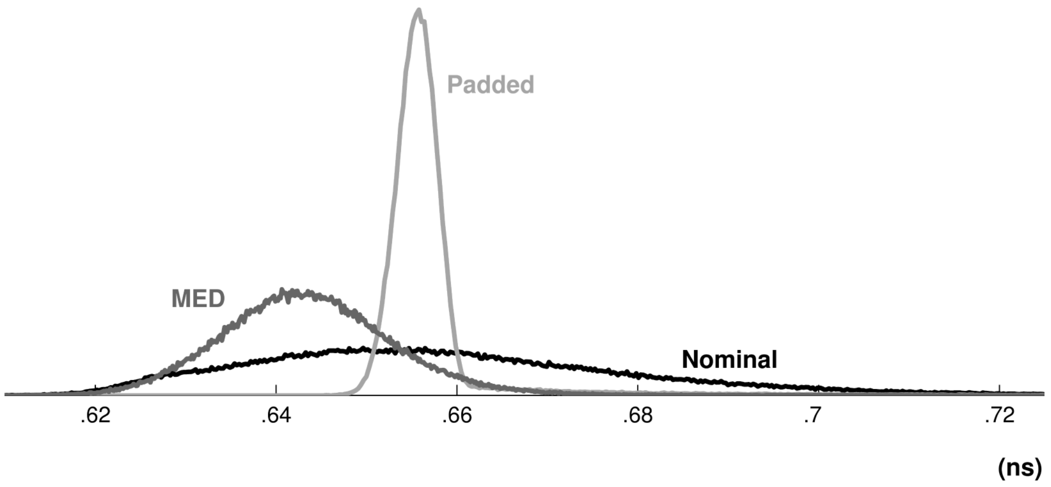


Figure 4.16: The delay pdfs of benchmark c2670 with size dependent variations are plotted for each of the three sizing methods. The MED gives the best delay distribution, followed by the padded delay sizings. These methods give very good results with significant gains over the nominal.

were made to compare these methods under this model. We use the setting $\alpha = .3$ and the variations were set to be the same as above. In this case, the MED sizing can still give better results (see Table 4.14), however the improvements on the quantile are not as large, dropping to 2%. A closer look at the pdfs in Fig. 4.16 shows that although the 95% quantiles are competitive, the pdf of the MED sizing has a significantly faster average delay than the padded delay sizing.

There are three conclusions that can be drawn from these results. First, there is a significant gain when the statistical problem is solved over the nominal problem. With size independent variations, this can be up to 10% and with size dependent variations, the difference grows to 15%. Secondly, the padded delays usually give results that are similar to the nominal sizing, and in our experience, we find this approximation to be useful when there is a large spread in path delays, such as large adder circuits. In contrast, the padded delay method does an excellent job when a size-dependent variation model is used, but there is still some improvement that can be made by using the MED sizing.

The runtime of this method scales similarly to the interior point solver used to perform the nominal and padded delay sizings when the number of samples is fixed. This is because both methods use Newton's method at each iteration to solve the subproblems. Furthermore, the same techniques that are used to improve scalability in the nominal sizing, as in [JB08], might apply to MED sizing with SAA sampling.

4.3.5 Summary

In this section we introduced the mean excess delay as a statistical measure for circuit delay in the presence of random parameter variations. The β-mean excess delay is defined as the expected delay of the circuits whose delay exceed the β-quantile. Thus, minimizing mean excess delay indirectly reduces the β-quantile. This technique is known in finance as conditional value-at-risk optimization [RU00]. Compared with other popular methods (such as the padded delay sizing method [PYK05]), MED sizing has the important advantage that it takes into account the type of correlation in the gate delay variations. The results show a significant improvement can be made by using the mean-excess delay over the padded and nominal methods.

CHAPTER 5

Manufacturing Processes, Gate Sizing and Threshold Assignment

The introduction of multi-gate devices for the 22nm technology node introduces new challenges in creating standard cell libraries. The gate sizes can no longer be arbitrary widths, and instead must come in multiples of the fins, e.g. as 1x, 2x, 3x, 4x, etc. [PAS06]. To compound the problem, a wide array of threshold voltages may not be possible [War11]. Thus, to understand the impact of these restrictions on the power and delay tradeoffs for gate sizing and threshold voltage assignment, we examine the impact of the range and precision on standard cell-based designs in this chapter.

In addition, with the aggressive production schedules in the semiconductor industry, integrated circuits are often designed before the manufacturing process is mature. As a result, substantial changes in the specifications may cause timing infeasibility. These timing errors are corrected using Engineering Change Orders, commonly referred to as ECOs. The use of incremental gate sizing to correct for changes in the manufacturing process specifications is also developed in this chapter.

5.1 Impact of precision and range in standard cell libraries

Standard cell libraries offer a wide variety of both logical functions and sequential elements. In addition to the variety of functions, a range and selection of cell parameters are provided, including changes in the threshold voltages, transistor sizes, and P/N ratios. This provides flexibility for the designer, allowing them to fine tune power, delay, robustness and noise characteristics during circuit design.

Choosing the threshold voltages, transistor sizes, transistor lengths, and P/N ("beta") ratios from the values allowable by the technology is a tradeoff between maintaining a manageable and supportable library, and providing flexibility to the circuit designer. Large libraries require substantial time to design, tune, and characterize. Commercial design teams often use libraries with hundreds of cells, and these cells must be re-characterized periodically to match silicon measurements, and changes in manufacturing process parameters.

This section examines the suboptimality associated with any given selection of gate sizes and threshold voltages. The suboptimality is relative to the case where a continuous range gate sizes and threshold voltages are given. More formally, the problem is: *given a design, delay target (or target clock period), and standard cell library, estimate the suboptimality in power, relative to a case where a continuous set of sizes are available.*

Note that this question is a stronger question than the library cell selection problem. That is, if the suboptimality is known, then this information can be used to select a set of library cells, or even the technology that should be used to implement the design. In contrast, research on library cell selection [BKK98, SG06, ART09] has proceeded without a means to quantify this suboptimality.

Prior work on this subject uses quantization error, and experimental results to guide library cell selection. In [BKK98], the authors take an error based approach that minimizes the error incurred from snapping continuous gate sizes. The gate sizes are chosen to minimize the expected error in cell area ("size-match"), delay ("delay-match") or power ("power-match"). More specifically, let s represent the gate size, c_L represent the capacitive load of a gate, $\mathbf{Prob}\,(s)$ represent the probability distribution for the optimal continuous sizes, and $\mathbf{Prob}\,(s, c_\mathrm{L})$ represent the joint probability distribution for the optimal continuous sizes and the capacitive loads. If the set of gate sizes is given as $\{s_1, ..., s_n\}$, then the errors are:

$$\mathrm{so}_{\mathcal{Q}(\mathrm{delay})} = \min_{s_i} |d(s_i, c_\mathrm{L}) - d(s, c_\mathrm{L})| \tag{5.1}$$

$$\mathrm{so}_{\mathcal{Q}(\mathrm{size})} = \min_{s_i} |s_i - s| \tag{5.2}$$

$$\mathrm{so}_{\mathcal{Q}} = \mathrm{so}_{\mathcal{Q}(\mathrm{power})} = \min_{s_i} |p(s_i) - p(s)| \tag{5.3}$$

where $\mathrm{so}_{\mathcal{Q}(\mathrm{delay})}$ is the delay-match, $\mathrm{so}_{\mathcal{Q}(\mathrm{size})}$ is the size match, $\mathrm{so}_{\mathcal{Q}(\mathrm{power})}$ is the power match, d is the delay of the gate, and p is the power of the gate. The work in [BKK98] focuses on minimizing the expected value of these errors over the distribution of s and c_L. These metrics accurately measure the quantization, or the errors associated with snapping to the discrete sizes. However, these metrics are not related to design metrics, such as power or delay suboptimality, and it does not explain why matching the sizes ("size-match") produces the best results.

[SG06] examines which geometric size progression provides the best tradeoffs. They find that a ratio of 1.3 with cells of size $\{1\times, 1.3\times, 1.3^2\times, ...\}$ provides results very close to that with an infinite set of gate sizes. [ART09] provide experimental results that evaluate different types of library selection. The experiments are performed using their continuous sizing plus branch-and-bound sizing method to evaluate different sets of gate sizes. They conclude that a compact library with sizes $\{0.5\times, 1\times, 2\times, 3\times, 4\times\}$ and 3-4 beta values work the best.

However, while prior work provides some insight into creating libraries, they lack an analytical framework to consider the power vs. delay tradeoff. The question of how the library cell selection affects delay constrained power optimization is still an open question that is very relevant with the increasing constraints on standard cell libraries and the increasing importance of power minimization.

This work shows how the library cells in a design affect the suboptimality of a delay constrained, power optimized design. In this section, we focus on leakage power, but the

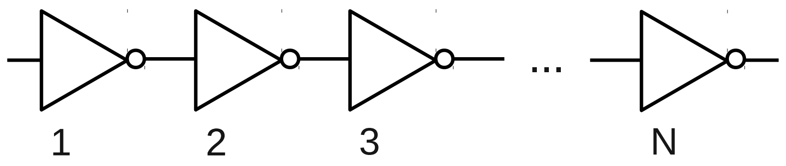


Figure 5.1: An inverter chain example.

framework can be used to estimate the dynamic power suboptimality as well. We also provide experimental results that show our estimates are useful in practice.

This section will cover the following topics:

- Framework for analyzing the suboptimality in power of a design due to the range and precision of v_{t} and sizes.
- Experimental results that show the usefulness and accuracy of these methods.
- Extensions to library cell selection and applications in multi-terminal gate based design.
- A study of the impact of the range in sizes or v_{t} on the achievable power and delay.

5.1.1 Suboptimality of v_{t} Assignment

Threshold voltage assignment is a very useful tool to optimize designs for power. Increasing the threshold voltage provides an exponential decrease in leakage, with an effect on the delay proportional to (1.5) [SN90]. For example, in the Nangate Open Cell Library [NANb], $V_{\mathrm{dd}} = 1.1$ and $\alpha \approx 1.4$.

The rationale in selecting threshold voltages is can be governed by Equations (5.1), (5.2), and (5.3). However, this is inadequate to estimate suboptimality. Consider the example in Figure 5.1, which has N inverters tied as a chain. Suppose a very simple model for the delay and power as a function of v_{t}, given by:

$$\text{Delay} = v_{\mathrm{t}}, \quad \text{Power} = 3 - v_{\mathrm{t}}, \quad 1 \leq v_{\mathrm{t}} \leq 2.$$

For a delay target of T (with $T > N$), an optimal set of continuous v_{t} is one with all gates at $v_{\mathrm{t}} = T/N$, and a corresponding power $3N - T$.

Equations (5.1), (5.2), and (5.3) suggest that the suboptimality associated with choosing a discrete set of sizes is related to the quantization error. In the above case, this is accurate for $N = 1$, where the optimal continuous value, $v_{\mathrm{t}} = T$, is rounded down to the nearest available discrete v_{t}. This gives a power suboptimality of $3 - (T - v_{\mathrm{t}})$.

However, the interaction between gates is overlooked in this case. As N grows larger, the suboptimality can decrease – even with just two values: $v_{\mathrm{t}} \in \{1, 2\}$. The idea is that the suboptimality incurred by snapping down a v_{t} can be mitigated by snapping up in the next stage. Surprisingly, with this model, the difference in power between the optimal discrete and continuous solutions is at most 1, and as a percentage it is at most $1/N$. Thus, as N increases, the difference between the quantized and continuous solutions, as a percentage of the total power, continues to decrease to zero. Also in the limit, as N increases, the effective delay vs power tradeoff is linear when this effect is taken into account.

In contrast, consider the case where the delay and power are given by:

$$\text{Delay} = v_{\mathrm{t}}, \quad \text{Power} = 4 \cdot \exp(-\ln(2) \cdot v_{\mathrm{t}}) = 2^{-v_{\mathrm{t}}+2}$$

where the power as a function of v_{t} is slightly different. In this case, the values of delay and power at $v_{\mathrm{t}} \in \{1, 2\}$ are identical to the earlier example, but the power at the points in between is different. This leads to the situation in Figure 5.2. In this case, by mixing the options $v_{\mathrm{t}} \in \{1, 2\}$, the points along the upper linear tradeoff can be achieved. However, using continuous v_{t} values can provide the tradeoff values along the lower curve. The difference between the two curves contributes to the suboptimality, and the v_{t} should be selected in a way to minimize this difference for a majority of designs.

Thus, in general, the suboptimality for a gate is proportional to the difference between the curves in Figure 5.2. For example, for a given gate, the required delay is first determined and the point on the continuous tradeoff curve is identified. Next, the vertical distance to the upper tradeoff line is computed, and this is the suboptimality for the gate. Note that

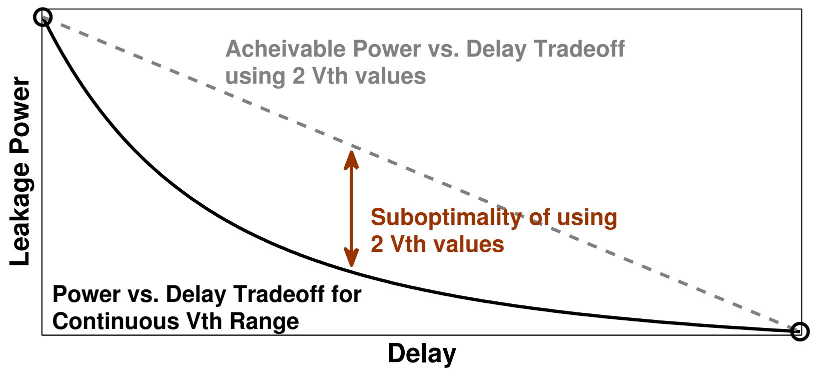


Figure 5.2: Suboptimality for a single gate with two threshold voltages.

the available v_{t} determines the upper tradeoff line. Mathematically, the difference between the upper tradeoff line $\hat{p}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}})$, and the continuous power vs. delay tradeoff curve, $p(v_{\mathrm{t}})$ is computed as:

$$\mathrm{so_c}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}}) = \hat{p}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}}) - p(v_{\mathrm{t}}). \tag{5.4}$$

Here, τ is the set of slews or input transition of the gate, and c_{L} are the capacitive loads for the gate. The upper tradeoff line $\hat{p}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}})$ is defined as the line between the two neighboring v_{t} that are available in the cell library:

$$\hat{p}(v_{\mathrm{t}}) = \lambda(v_{\mathrm{t}}) \cdot p(Q^{-}(v_{\mathrm{t}})) + (1 - \lambda(v_{\mathrm{t}})) \cdot p(Q^{+}(v_{\mathrm{t}})) \tag{5.5}$$

with

$$\lambda(v_{\mathrm{t}}) = \frac{d(v_{\mathrm{t}}) - d(Q^{-}(v_{\mathrm{t}}))}{d(Q^{+}(v_{\mathrm{t}})) - d(Q^{-}(v_{\mathrm{t}}))}, \tag{5.6}$$

and Q^{+} and Q^{-} are the round-up and round-down quantization functions, respectively, defined as:

$$Q^{+}(v_{\mathrm{t}}) = \min_{i}\{v_{\mathrm{t},i} \mid \; v_{\mathrm{t},i} \geq v_{\mathrm{t}}\} \tag{5.7}$$

$$Q^{-}(v_{\mathrm{t}}) = \max_{i}\{v_{\mathrm{t},i} \mid \; v_{\mathrm{t},i} \leq v_{\mathrm{t}}\}. \tag{5.8}$$

The "c" in $\mathrm{so_c}$ denotes that this suboptimality is related to the convexity of the delay vs. power curve. $\mathrm{so_c} > 0$ (e.g. there is a suboptimality) if the curve is strictly convex; as seen earlier, a linear delay vs. power tradeoff has a $\mathrm{so}_{\mathrm{c}} = 0$. An interesting consequence is that if the curve is *concave* then a better tradeoff can be achieved using the linear tradeoff curve, which will lie below the continuous tradeoff curve. Note that this estimate is less pessimistic than assuming that the v_{t} is snapped down.

This model can be improved to handle slew and other timing effects and interactions by using the slacks instead of the delays. In this case, we have:

$$\lambda(v_{\mathrm{t}}) = \frac{s(v_{\mathrm{t}}) - s(Q^{-}(v_{\mathrm{t}}))}{s(Q^{+}(v_{\mathrm{t}})) - s(Q^{-}(v_{\mathrm{t}}))}, \tag{5.9}$$

This improved expression will be used for the remainder of this section.

5.1.1.1 Suboptimality expressions

The total suboptimality of a design can be expressed in terms of so_{c}:

$$\mathrm{so}(\vec{v_{\mathrm{t}}}, \vec{\tau}, \vec{c_{\mathrm{L}}}) = \gamma_{\mathrm{c}} \sum_{\forall \text{gates } g} \mathrm{so}_{\mathrm{c}(g)}(v_{\mathrm{t}}(g), \tau(g), c_{\mathrm{L}}(g)) \tag{5.10}$$

where v_{t} above is the optimal continuous size, τ is the set of slews or input transitions in the design, and c_{L} are the capacitive loads for each gate in the design. $\gamma_{\mathrm{c}} \approx 1$ is a fitting term used to improve the fit of the bounds.

When the optimal continuous v_{t} are not available, but the design is available, the suboptimality for each gate can be taken to be the worst-case by taking the worst-case over the slews, τ, and v_{t}:

$$\mathrm{so}(c_{\mathrm{L}}) = \max_{v_{\mathrm{t}}, \tau} \left\{ \gamma_{\mathrm{c}} \mathrm{so}_{\mathrm{c}}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}}) \right\}. \tag{5.11}$$

If the bounds are relative to a given library, the expression for the suboptimality of each gate can be rewritten to provide the maximum over all the possible loads as well:

$$\mathrm{so} = \max_{v_{\mathrm{t}}, \tau, c_{\mathrm{L}}} \left\{ \gamma_{\mathrm{c}} \mathrm{so}_{\mathrm{c}}(v_{\mathrm{t}}, \tau, c_{\mathrm{L}}) \right\}. \tag{5.12}$$

5.1.1.2 Small circuit experiments

To better understand the effects of v_t selection on the suboptimality, 30 circuit examples were randomly generated using the method in [SVC00], each having 30 gates. The delay and power functions were modeled using a 65nm commercial library, and fit to posynomial models. These circuits were small enough to solve optimally, using a branch-and-bound method.

The optimal continuous power and discrete powers were computed for ten different delay targets between the minimum delay and maximum delay (the delay associated with the minimum power configuration). Next, the value γ_c in (5.10) is fit to its least-squares value, yielding $\gamma_c = .974$, which shows that the $\gamma_c \approx 1$ and that the preceding analysis is an accurate predictor of the suboptimality. The resulting fit, and errors, are shown in Figure 5.3.

The results are very good. The standard deviation in errors are 5.3%, 2.5%, 1.9% and .9%, for 1, 2, 3 and 4 v_t options, respectively (the errors are given as a percentage of the discrete optimum). In contrast, using the quantization errors in (5.1), (5.2), and (5.3) provide much larger errors, at 16%, 34%, 27% and 14%, respectively.

5.1.1.3 Effects on Post-layout Threshold Voltage Assignment

An experiment to measure the effect of threshold voltage assignment on post-layout designs (where the wire loads are known) was performed. Libraries with several different v_t choices were used (the percentages denote the deviation from the nominal v_t):

- v_t-L1: $\{-20\%, 20\%\}$
- v_t-L2: $\{-20\%, 0\%, 20\%\}$
- v_t-L3: $\{-20\%, 0\%, 10\%, 20\%\}$
- v_t-L4: $\{-20\%, 0\%, 10\%, 20\%\}$
- v_t-L5: $\{-20\%, 10\%, 0\%, 10\%, 20\%\}$

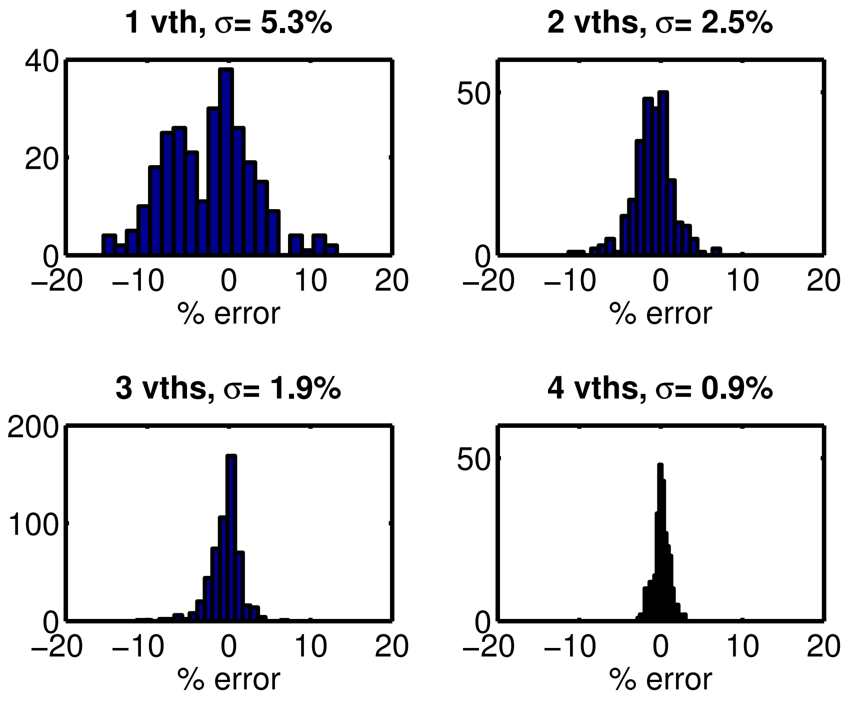


Figure 5.3: Histogram of the errors in predicting the discrete optimum, from a given optimal continuous solution. The errors are given as a percentage of the discrete optimum.

and they were compared to values for a dense library (e.g. a library with all v_t values between -20% and 20% of the nominal v_t, in increments of .1%. The library is a 32nm library from a leading EDA company where the $\{-20\%, 0\%, 20\%\}$ data points are included in the library. The remaining points are created by fitting each table entry in the Liberty file using exponential models to interpolate the power and (1.5) to fit the delay. The libraries v_t-L1, v_t-L2, v_t-L3, v_t-L4, v_t-L5 have 2, 3, 4, 4, and 5 v_t options, respectively.

All optimizations are performed using the widely-known and simple to implement circuit optimization method TILOS [FD85] (see Section 2.3.3). While this method is not optimal, even for the dense library, it is derived from an optimal algorithm, and therefore should provide reasonable results. In addition, [LG12] and Section 3.4.3 show that this method provides results that are robust – they perform well over a range of benchmarks. In this section, the dense library is used as a proxy for a continuous library.

The benchmarks used in this study are widely used gate sizing and v_t assignment benchmarks, from the ISCAS '85 and ISCAS '89 benchmark suites. They are initially synthesized, placed and routed with all optimization flags set to high-effort and minimum delay. Once the sizing is complete, the suboptimality estimate for each library choice can be computed in a matter of seconds. A common value of $\gamma_\text{c} = .87$ was used to estimate the suboptimality for all benchmarks.

Table 5.1 show the statistics for the suboptimality prediction errors (max, average, and standard deviation) for this work (so_c), and as a reference we provide estimates ($\text{so}_\mathcal{Q}$) derived from work in [BKK00]. The results show that using the so_c metric is a better predictor than the $\text{so}_\mathcal{Q}$, providing much smaller errors in the worst cases, such as the c6288 v_t-L1 library. In the c7552 v_t-L1 case, the errors seem worse, however, Figure 5.4 shows that these numbers are misleading – the fit using the so_c is qualitatively better than the fit using the $\text{so}_\mathcal{Q}$. Figure 5.5 shows that this is true in general– the fit using $\text{so}_\mathcal{Q}$ on the left does not follow the trend well, and the fit on the right using so_c is clearly superior. Overall, the so_c has an average error nearly half of the $\text{so}_\mathcal{Q}$ estimates – a mean error of

	ϵ-$\text{so}_{\mathcal{Q}(\text{power})}$			ϵ- so_{c}				ϵ-$\text{so}_{\mathcal{Q}(\text{power})}$			ϵ- so_{c}		
	max	avg	std	max	avg	std		max	avg	std	max	avg	std
c6288							s35932						
v_{t}-L1	712%	197%	249%	390%	100%	123%	v_{t}-L1	231%	69%	94%	144%	35%	48%
v_{t}-L2	33%	12%	14%	20%	5.5%	6.7%	v_{t}-L2	22%	6.8%	8.9%	17%	3.6%	5.4%
v_{t}-L3	42%	5.5%	9.2%	17%	2.6%	4.3%	v_{t}-L3	26%	3.6%	8.6%	16%	2.2%	4.7%
v_{t}-L4	47%	14%	19%	31%	6.9%	9.7%	v_{t}-L4	12%	5.4%	6.4%	9.7%	3.1%	4.0%
v_{t}-L5	8.5%	1.9%	2.4%	8.5%	1.6%	2.3%	v_{t}-L5	7.2%	1.4%	2.2%	10%	1.5%	2.9%
c7552							s38417						
v_{t}-L1	103%	39%	39%	142%	40%	37%	v_{t}-L1	48%	3%	7.2%	48%	1.5%	4.6%
v_{t}-L2	23%	7.2%	9.3%	14%	4.7%	6.2%	v_{t}-L2	6.3%	.3%	1.0%	5.1%	.2%	.8%
v_{t}-L3	41%	4.3%	9.4%	11%	1.6%	2.8%	v_{t}-L3	3.9%	.1%	.5%	3.6%	.1%	.5%
v_{t}-L4	17%	7.3%	8.7%	15%	5.1%	6.8%	v_{t}-L4	3.6%	.3%	.6%	2.7%	.1%	.4%
v_{t}-L5	7.0%	1.4%	2.1%	6.1%	1.6%	1.7%	v_{t}-L5	1.1%	.03%	.1%	1.1%	.04%	.1%
							s38584						
							v_{t}-L1	20%	2.9%	4.1%	13%	3.3%	4.2%
							v_{t}-L2	4.3%	.2	.59%	3.9%	.2%	.5%
							v_{t}-L3	3.9%	.1%	.44%	3.9%	.1%	.4%
							v_{t}-L4	1.4%	.1%	.27%	1.1%	.1%	.2%
							v_{t}-L5	.9%	.01%	.13%	1.1%	.1%	.1%

*minimum error for all designs is 0%

Table 5.1: Errors in v_{t} suboptimality estimation (as a percentage of the total power). Statistics are over the range of delays.

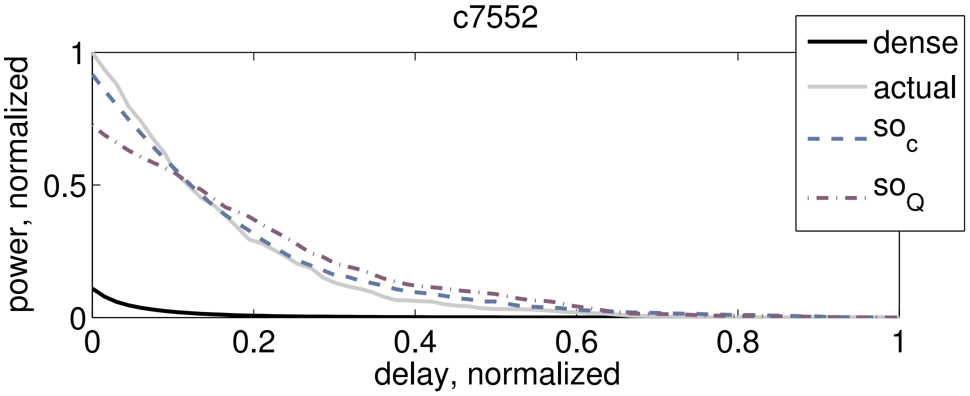


Figure 5.4: Error prediction on the c7552 benchmark, in the v_t-L1 case. The so$_c$ and the so$_Q$ predictions are made using the data from the dense library result.

8.8%, compared to 15%.

In viewing these results, it is important to remember two things. First, the suboptimality estimates are predicted from the dense v_t assignment. Figure 5.4 shows that the dense assignment is nearly an order of magnitude smaller than the discrete power, yet both the value and the trends are predicted well.

The next thing to remember is that these results are for practical optimization tools– TILOS method is not an *optimal* method, and the optimization results have some additional suboptimality involved. However, the estimates still predict the suboptimality well, showing that this method is applicable to practical optimization methods.

5.1.2 Suboptimality in gate sizing

Computing the suboptimality in gate sizing is substantially different than that of v_t. This is for two reasons:

- The delays between different gates are coupled – changing the size of a gate will affect the delays of its neighboring gates due to the changing capacitive loading.
- The power tradeoff between different gates is not as dramatic. The change in leakage

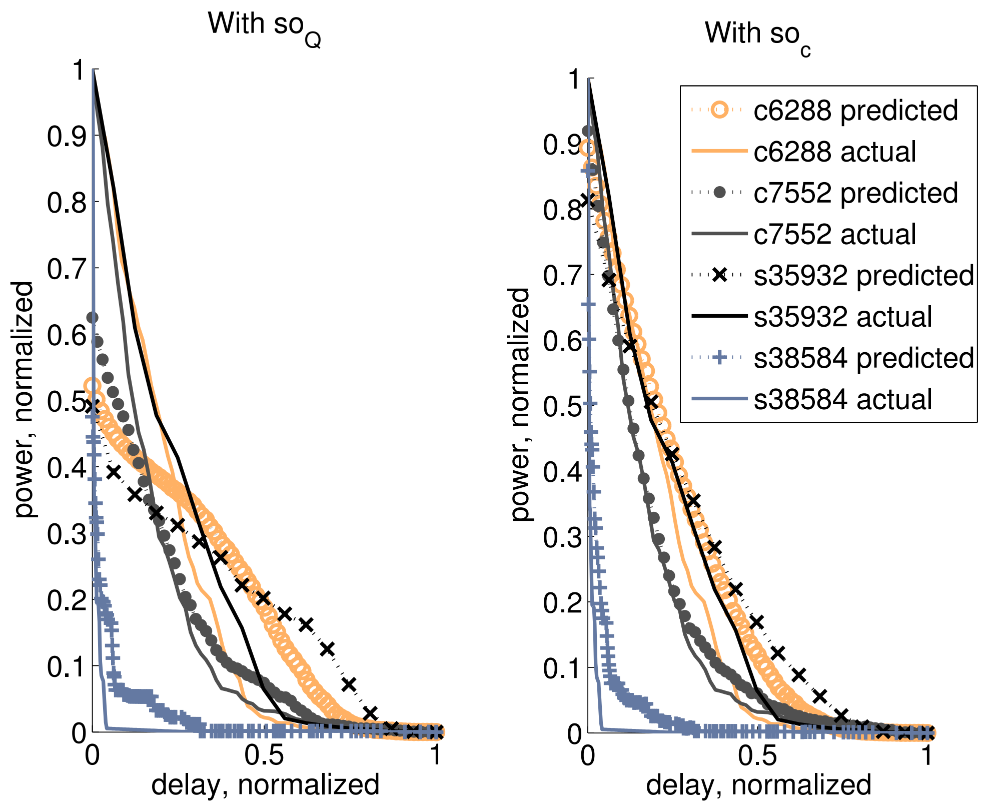


Figure 5.5: Suboptimality estimates with library v_t-L1 for the benchmarks c6288, c7552, s35932, and s38584. The so$_Q$ method is on the left, and the so$_c$ is on the right. The benchmark s38417 omitted as the errors for this benchmark were small.

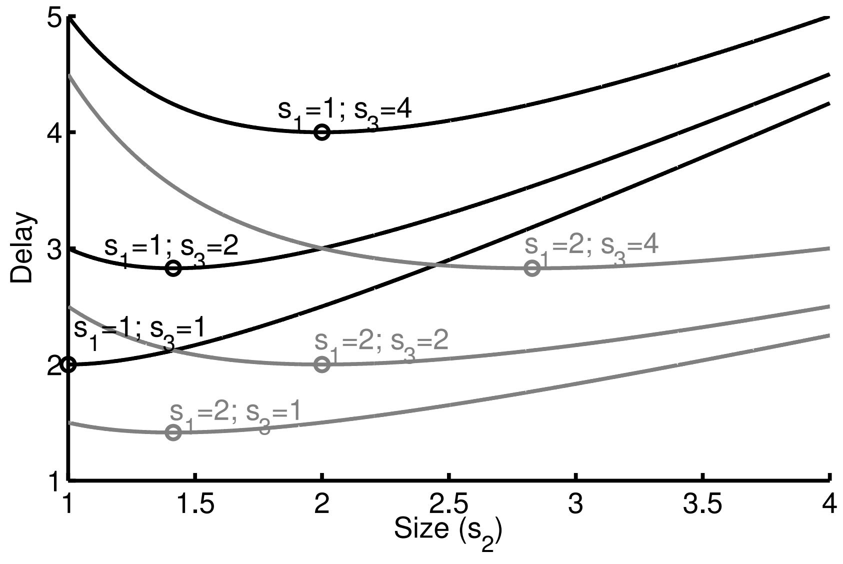


Figure 5.6: Arrival time at the output of gate 2 (from Figure 5.1), as a function of the gate sizes s_1, s_2, s_3. The 'o' marks the minimum delay value on each curve.

power as a function of v_t is exponential, while the change in leakage power as a function of size is linear.

The coupling of the delay between gates results in cases where increasing a gate's size *decreases* the delay at its output while *increasing* the delay at its inputs, because the gate's input pins increase in size and therefore in capacitance. For example, Figure 5.6 plots the cumulative delay at the output of gate 2 (from Figure 5.1), as a function of the input gate size (s_1), the size of gate 2 (s_2) and the size of the output gate 3 (s_3) with the model:

$$\text{Delay} = (\textstyle\sum_{j\in \text{fanout}(i)} s_j)/s_i, \quad \text{Power} = s_i$$

The tradeoff curves vary heavily as the input gate size changes. This is in contrast to the case in Figure 5.2, where the cumulative delay is not affected by the input gate size.

An important difference between Figures 5.2 and 5.6 is that the minimum cumulative delay does not always occur at the maximum or minimum sizes – the maximum size does not always result in the minimum delay. For example, suppose that in Figure 5.8 only gate sizes $s_2 = 1$ and $s_2 = 4$ are available. In this case, gate size $s_2 = 4$ may be used even though it is not Pareto-optimal on the continuous curve, and does not provide the best delay relative to other continuous sizes. As another example, consider the plot in Figure 5.7, which has the delay as the x-axis, and power as the y-axis. Delay values less than three require sizes between $(1, 2)$, and therefore the choice of intermediate sizes affect whether a particular delay is achievable. When the required delay cannot be achieved by a set of library sizes, it results in a delay penalty that must be corrected by sizing multiple gates.

This results in two cases:

1. A size within the library is able to provide the required delay.
2. No size in the library can provide the required delay.

In the first case, an analogue of (5.4) can be used to estimate the effect on suboptimality. This is done by finding the difference between the piecewise linear tradeoff function of the available sizes with the tradeoff curve of the continuous sizes. In Figure 5.8, this is the difference in power between the gray piecewise linear tradeoff function and the black curve. Mathematically, this is:

$$\mathrm{so_c}(s, \tau, c_\mathrm{L}) = \hat{p}(s, \tau, c_\mathrm{L}) - p(s) \tag{5.13}$$

where

$$\hat{p}(s, \tau, c_\mathrm{L}) = \lambda(s, \tau, c_\mathrm{L}) \cdot p(F^-(s, \tau, c_\mathrm{L})) + \tag{5.14}$$

$$(1 - \lambda(s, \tau, c_\mathrm{L})) \cdot p(F^+(s, \tau, c_\mathrm{L})) \tag{5.15}$$

$$\lambda(s, \tau, c_\mathrm{L}) = \frac{s(s, \tau, c_\mathrm{L}) - s(F^-(s, \tau, c_\mathrm{L}))}{s(F^+(s, \tau, c_\mathrm{L})) - s(F^-(s, \tau, c_\mathrm{L}))}. \tag{5.16}$$

where F^+ and F^- are generalizations of the quantization functions Q^+ and Q^- for sizing, that are used to handle the case where there is no available size that meets the required slack, and are defined as:

$$F^+(s,\tau,c_\mathrm{L}) = \min\{s_i \mid \; s(s_i,\tau,c_\mathrm{L}) \geq s(s,\tau,c_\mathrm{L})\} \tag{5.17}$$

$$F^-(s,\tau,c_\mathrm{L}) = \max\{s_i \mid \; s_i < F^+(s)\}. \tag{5.18}$$

When there are no sizes that provide the needed slack, the convention $F^+(s) = \infty$ is used. The slacks are used here to account for the cumulative effect on the delay, as sizing affects both the delay at the output of the gate, and the delay at the input of the gate.

In the next case, the effect on suboptimality is much more difficult to characterize. Not only is the current gate affected, but the resulting delay penalty (due to the gate's inability to achieve the required delay) requires the sizes of other gates to be adjusted to achieve the required delay. This is approximated as a function of the power penalty incurred when rounding up to the nearest size.

Combined, these two cases are:

$$\mathrm{so}_{\mathrm{c+}}(s,\tau,c_\mathrm{L}) = \begin{cases} \mathrm{so}_\mathrm{c}(s,\tau,c_\mathrm{L}) & \text{if } F^+(s) < \infty \\ \gamma_\mathcal{Q}(p(Q^+(s)) - p(s)) & \text{otherwise} \end{cases} \tag{5.19}$$

where $p(Q^+(s)) - p(s)$ is the power penalty for rounding up to the nearest size, with

$$Q^+(s) = \min_i\{s_i \mid \; s_i \geq s\}. \tag{5.20}$$

In the first case, the suboptimality is characterized by the convexity of the power vs. delay curve, and when this cannot be applied, it is characterized by the power penalty of snapping the next higher size.

The suboptimality over the whole design can then be estimated as:

$$\mathrm{so}_{\mathrm{c+}}(\vec{s},\vec{\tau},\vec{c_\mathrm{L}}) = \sum_{\forall \text{gates } g} \mathrm{so}_{\mathrm{c+}(g)}(s(g),\tau(g),c_\mathrm{L}(g)) \tag{5.21}$$

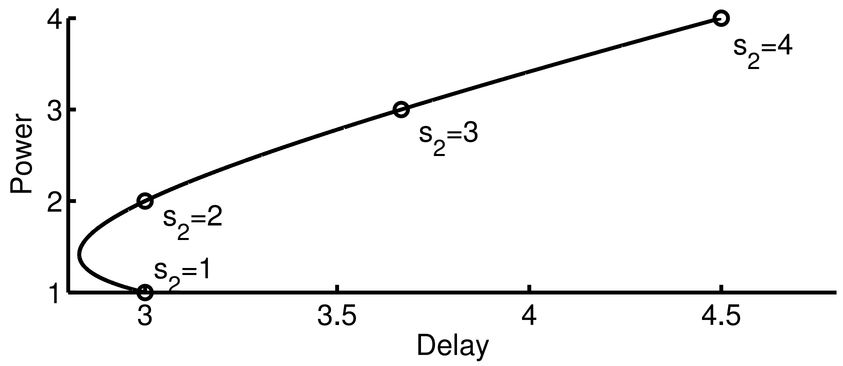


Figure 5.7: Power vs delay curve for gate 2 in Figure 5.1 where the size of gate 1 is 1 ($s_1 = 1$) and the size of gate 3 is equal to 2 ($s_3 = 2$). Delay values < 3 are unachievable with the sizes $\{1, 2, 3, 4\}$.

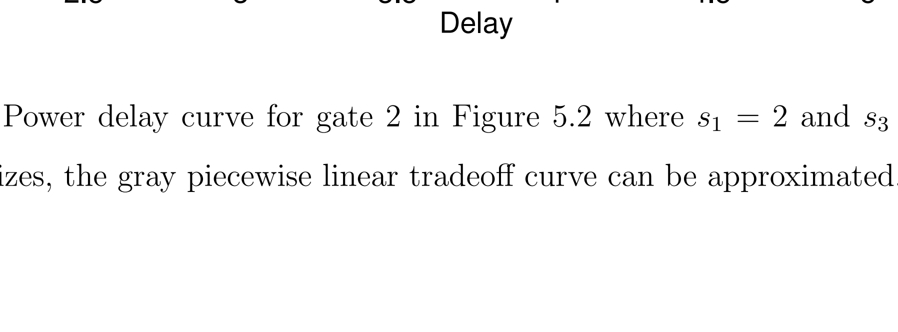


Figure 5.8: Power delay curve for gate 2 in Figure 5.2 where $s_1 = 2$ and $s_3 = 4$. Using these gate sizes, the gray piecewise linear tradeoff curve can be approximated.

5.1.3 Post-layout experiments

Gate sizing suboptimality estimates for post-layout designs (where the wire loads are known) were performed. Libraries with several different gate size choices were used:

- s-L1: 1x, 8x
- s-L2: 1x, 2x, 8x
- s-L3: 1x, 4x, 8x
- s-L4: 1x, 2x, 4x, 8x
- s-L5: 1x, 2x, 3x, 4x, 5x, 6x, 7x, 8x

and they were compared to values for a dense library (e.g. a library with all sizes between 1x and 8x, in increments of .1x. The library is derived from a commercial 45nm library. The sizes that are not in the library are created by fitting each table entry in the Liberty file using linear models to interpolate the power and posynomial models to fit the delay.

As in Section 5.1.1.3, all optimizations are performed using TILOS [FD85], and the same benchmarks are used. The designs are synthesized, placed and routed with all optimization flags set to high-effort.

Table 5.2 shows the errors in the suboptimality estimates for this work ($\mathrm{so}_{\mathrm{c+}}$), and as a reference we provide estimates ($\mathrm{so}_{\mathcal{Q}}$) derived from work in [BKK00]. The table shows that the $\mathrm{so}_{\mathrm{c+}}$ provides an excellent estimate of the suboptimality. Compared to predicting the error using $\mathrm{so}_{\mathcal{Q}(\mathrm{power})}$, the $\mathrm{so}_{\mathrm{c+}}$ is clearly better. Overall, the so_{c} has an average error less than a tenth of the $\mathrm{so}_{\mathcal{Q}}$ estimates – a mean error of .47%, compared to 7%.

In these results, $\gamma_{\mathcal{Q}}$ is fit separately for each of the benchmarks, unlike the case of v_{t} assignment, where the same value of γ_{c} is used for all benchmarks. This is because it is difficult to predict the power penalty when no gate size can provide the needed slack. While the same $\gamma_{\mathcal{Q}}$ can be used, the error rates jump significantly– for example, in the c7552 s-L2 case, the average error becomes 3.9% with a standard deviation of 2.9%. The optimal fitted values for $\gamma_{\mathcal{Q}}$ vary as well, with $\gamma_{\mathcal{Q}} = .71$ for the c6288 to $\gamma_{\mathcal{Q}} = .94$ for the

	ϵ-so$_{\mathcal{Q}(\text{power})}$			ϵ- so$_{\text{c+}}$				ϵ-so$_{\mathcal{Q}(\text{power})}$			ϵ- so$_{\text{c+}}$		
	max	avg	std	max	avg	std		max	avg	std	max	avg	std
c6288							s35932						
s-L1	0.6%	0.3%	0.4%	2.2%	0.8%	1.2%	s-L1	1.5%	0.9%	1.0%	2.1%	0.8%	1.1%
s-L2	78%	19%	30%	5.6%	1.1%	2.3%	s-L2	48%	15%	16%	1.0%	0.4%	0.5%
s-L3	58%	16%	24%	9.5%	2.2%	4.4%	s-L3	86%	24%	29%	4.7%	1.7%	1.6%
s-L4	90%	22%	35%	0.2%	0.1%	0.1%	s-L4	43%	15%	15%	0.7%	0.3%	0.4%
s-L5	86%	21%	34%	0.3%	0.1%	0.1%	s-L5	34%	14%	13%	0.5%	0.2%	0.3%
c7552							s38417						
s-L1	1.0%	0.4%	0.6%	1.1%	0.4%	0.6%	s-L1	0.1%	0.1%	0.1%	0.2%	0.0%	0.1%
s-L2	14%	4.1%	5.8%	2.8%	0.7%	1.2%	s-L2	0.8%	0.2%	0.3%	0.1%	0.0%	0.0%
s-L3	7.9%	3.1%	3.5%	6.1%	1.8%	3.2%	s-L3	1.2%	0.2%	0.4%	0.4%	0.0%	0.1%
s-L4	16%	4.8%	6.9%	0.6%	0.2%	0.4%	s-L4	0.8%	0.2%	0.3%	0.1%	0.0%	0.0%
s-L5	34%	13%	15%	3.1%	1.0%	1.4%	s-L5	0.9%	0.2%	0.3%	0.1%	0.0%	0.0%
							s38584						
							s-L1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
							s-L2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
							s-L3	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
							s-L4	3.2%	0.4%	1.0%	0.2%	0.0%	0.1%
							s-L5	2.6%	0.3%	0.9%	0.1%	0.0%	0.0%

*minimum error for all designs is 0%

Table 5.2: Errors in gate sizing suboptimality estimation (as a percentage of the total power). Statistics are over the range of delays.

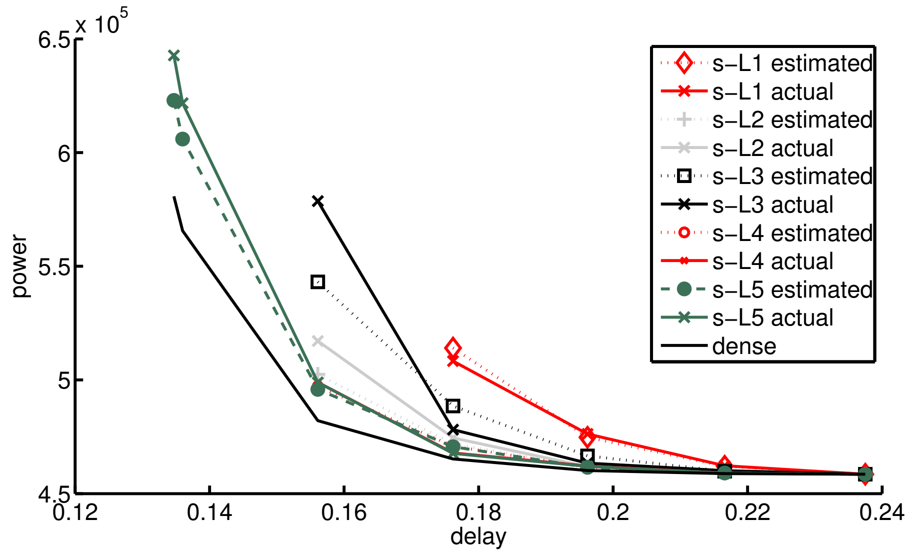


Figure 5.9: Estimation results for c7552. The estimate is predicted using from the dense sizing using the suboptimality results in this section, and the actual power refers to the actual power achieved by the TILOS method.

s35932 case. Determining the proper γ_Q for a given benchmark *a priori* is an interesting question for future research.

As in the v_t case, the estimates are also good qualitatively. Figure 5.9 shows the estimates with the actual values for the c7552 case. The estimates are very good and follow the trend well.

The analysis in this section shows why the size match library in [BKK00] performed the best. In the above analysis on sizing, the correct sizes are needed to provide the needed slacks in the circuit, otherwise the penalty related to $p(Q^+(s)) - p(s)$ is incurred, which is generally larger than the penalty when a size is available. Thus, it better done by matching the continuous sizes, which is a proxy for finding gates that provide the needed slacks, than in matching the delays at the output of the gate (note that in [BKK00], the delay match only matches the delay at the output of each gate, and ignores the impact on the delays at the inputs). This is why the size-match library in [BKK00] provides the best results.

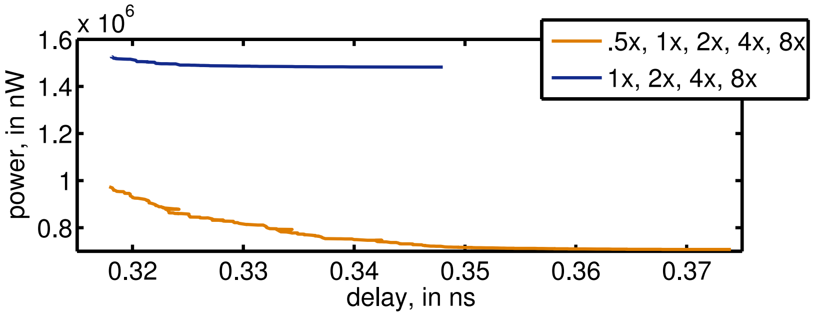


Figure 5.10: Effect of adding an additional minimum size of .5x on the s35932 benchmark in a 32nm library. The overall power decreases significantly, without affecting the delay range.

5.1.4 Dynamic range and precision selection

The theory developed in Sections 5.1.1 and 5.1.2 is useful in understanding the question of dynamic range – what should the maximum and minimum gate sizes and v_t values that should be used? – and the question of library precision selection – which v_t and sizes should be used?

The minimum size and maximum v_t should be set to the minimum values allowable by the technology. This is due to three reasons:

1. The power vs. delay tradeoff is convex and decreasing, thus lower power options improve the achievable power because they have a better power vs. delay tradeoff.
2. In most designs, there are a significant number of gates with positive slack and at their minimum power option. Thus, improving the power of these gates will create significant power savings, without a delay penalty. However, increasing the v_t and size choices can incur a technological as well as a library design cost.
3. In gate sizing, reducing a gate to a smaller gate size can reduce the capacitive load of the fanin gates.

For example, Figure 5.10 shows the effects of adding an additional gate with size 0.5x. The power drop is dramatic, and it is clear that smaller gate sizes make a large impact.

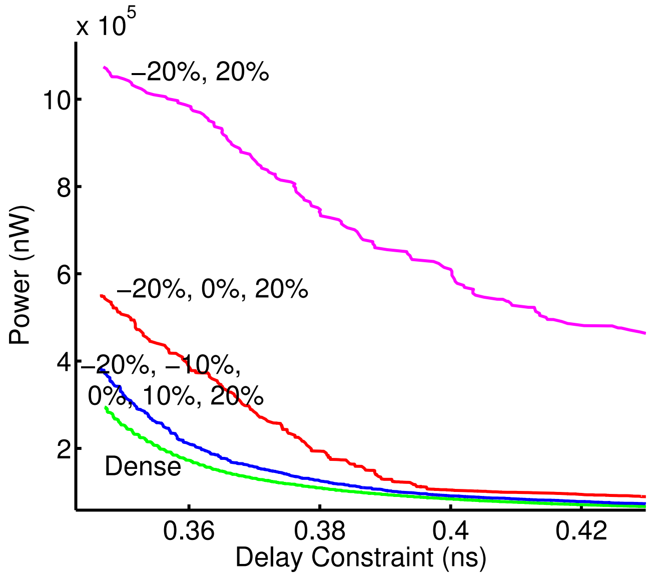
This power drop can be roughly estimated by assuming that the gates at minimum size (or maximum v_{t}) are switched to the newly available lower power option. A more accurate estimate can be found by performing the optimization itself, using a greedy method such as [GKS04].

The maximum size should be chosen to improve the delay range. The need for large gate sizes is largely dictated by the amount of capacitive loads in the design, in the form of wireloads and gate fanouts. This load dictates the power vs. delay tradeoff curve, as in Figure 5.6. When there are large load, the need for larger gate sizes increases (as in the $s_1 = 2$, $s_3 = 4$ case), but when larger loads are not present, then large gate sizes are not needed (as in the $s_1 = 1$, $s_3 = 1$ case).

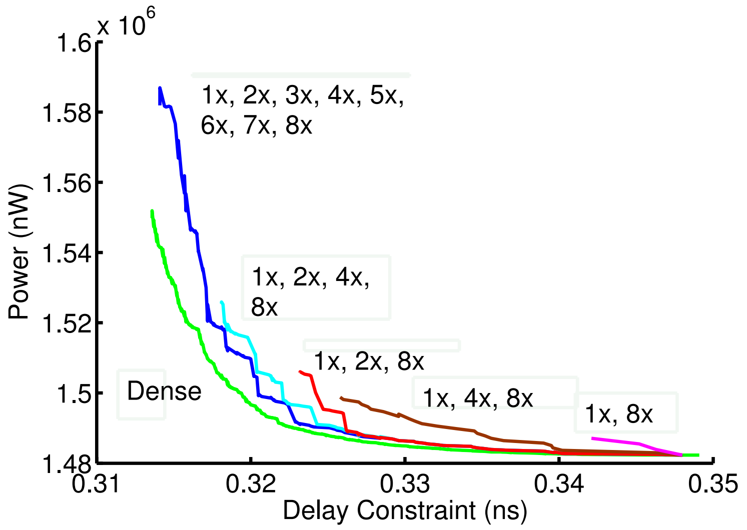
However, it is important to note that the maximum gate size does not itself dictate the delay range. This is because a wide selection of gate sizes is needed to achieve the minimum delay, as shown in Figures 5.7 and 5.8. The maximum size may not provide the minimum delay, and it is important to have the right size to achieve the delay that is necessary. The effects of different granularities on the delay range are shown in Figure 5.11(b). The achievable range using just 1x and 8x is quite poor.

The minimum v_{t} should be selected mainly for delay feasibility. Providing a lower v_{t} decreases the delay that is achievable in a design by providing faster gates. However, because the power tradeoff is convex and decreasing, adding a lower v_{t} is unlikely to improve the power, as the power vs. delay tradeoff is worse for lower v_{t} cells. However, the power may be improved in cases where there are cells that fanout to many other cells that are all critical.

Regarding library precision, the analysis in this section shows how to compute the suboptimality from a given design. This can be done using a continuous model of the library, or by creating a dense library. Once information from a dense solution is created, library choices can be analyze quickly – each library choice can be analyzed in a matter of seconds. Also, the dense library need not contain a large range of sizes and v_{t}; it must



(a)



(b)

Figure 5.11: Effects of library cell granularity on the achievable delay range for v_t assignment (a) and gate sizing (b). The optimization results for the s35932 benchmark in a 32nm library are shown.

only contain the candidate sizes and v_t that are under consideration.

If a design optimization is not possible, or not desired, then the library precision can be selected by using the expressions (5.12) and (5.19) for the expected input slews and output loads. The library designer can use the expressions to either minimize the worst-case suboptimalities or the average suboptimalities.

5.1.5 Multi-gate library considerations

FinFETs and other multi-gate devices are considered to be an alternative for conventional CMOS devices as they offer improved leakage power, reduced parasitic capacitances, and improved resistance to parametric variability [War11, PAS06]. These devices feature fully-depleted channels, and have gates that surround the channel on three sides, offering superior control of the channel.

However, these novel devices pose two challenges to standard library cell designers. Firstly, the gate widths must be quantized– different gate widths are created by placing multiple "fins" in parallel. This means that gate widths will only be available at multiples of the minimum gate width, and that fine granularities, such at 1.3x, may not be possible.

From the results above, this should not pose a large problem. As shown in Figure 5.11 the power penalty between a fully dense library and one with integer sized gates is not large. This is because the power vs. gate width tradeoff is linear, and therefore the power penalty of omitting fractional gate sizes will not be large. However, this may limit the achievable delay range, as in Figure 5.11, though this is minor.

On the other hand, it is not clear what kinds of v_t options will be available in FinFETs and multi-gate device. The v_t can be adjusted by changing the gate workfunction [War11], or by using the device as a three terminal device [MMJ10]. Researchers at the commercial foundry state that multi-v_t options can be provided by the FinFET family [YCL10], however the limited availability of v_t options to a designer will limit the design space very significantly, as in Figure 5.11(a).

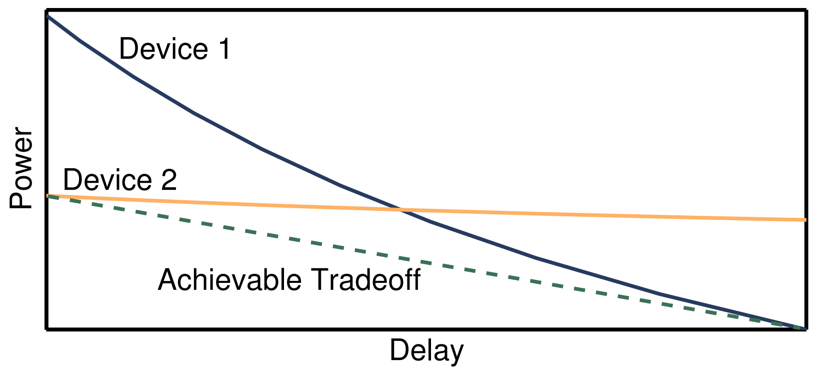


Figure 5.12: Achievable power delay range using two different types of devices. By mixing the device types, a better power vs. delay tradeoff can be achieved than by either of the device types alone.

5.1.5.1 Mixing technologies

The results in this section provide an interesting perspective on mixing technologies in a design. Consider the case in Figure 5.12, where there are two technologies with different power vs. delay curves. Device 1 provides a better tradeoff for large delays, and Device 2 provides better tradeoffs for smaller delays. The benefits of using both technologies can be analyzed in three cases: (1) when the technologies are used independently, (2) the technologies are mixed but optimized independently, and (3) the technologies are mixed and optimized interchangeably.

In the first case of using a single technology, either of the Device 1 or Device 2 tradeoff curves can be achieved, but not a mix of the two. In other words, once the device type is chosen, then the design is restricted to using that device alone. This can lead to a large suboptimality in when Device 1 is used near its minimum delay or Device 2 is used at its maximum delay. This suboptimality is avoided in case two, where the technologies are mixed. The mixture of technologies can achieve the pareto-frontier– the left-hand portion of Device 2 and right-hand portion of the Device 1's tradeoff curve.

However, there is a third case, where the devices are mixed and optimized interchangeably. In this case, gates in a single critical path can use devices from either technology, and this can provide a power savings that is greater than if each technology was used individually. For example, in Figure 5.12, this can provide the bottom dotted-tradeoff line. The tradeoff provided by this type of optimization is better than the best tradeoff of either technology, and there is an interesting synergy than can be exploited by mixture.

5.1.6 Summary

This section examined the impact of the range and precision in standard cell libraries by proposing methods for estimating the suboptimality in power. It is interesting that the suboptimality is primarily due to the convexity of the power vs. delay tradeoff. Compared to a method derived from literature [BKK00], our method provides a nearly 2x better estimate for v_{t} assignment and 10x improvement for gate sizing. This has interesting implications on selecting the range and precision of the standard cell libraries, especially with the new advances in multi-gate devices. For example, the minimum size and maximum v_{t} should be set to the least power option possible. Furthermore, the precision should be set with regards to the metrics (5.12) and (5.19), to minimize the potential suboptimality in the derivative design.

5.2 ECO Gate sizing for Manufacturing Process Changes

With the aggressive production schedules in the semiconductor industry, the design of integrated circuits runs concurrently with the development of the manufacturing process itself. As a result, the exact manufacturing specifications change over the design period. Substantial changes in the specification may cause timing infeasibility issues, which require Engineering Change Orders, commonly referred to as ECOs, to fix. As a tool for ECOs, gate sizing is commonly used to incrementally update designs, as it is generally less intrusive than adjusting the placement or performing buffer insertion on the design, and can be more powerful than rerouting the design.

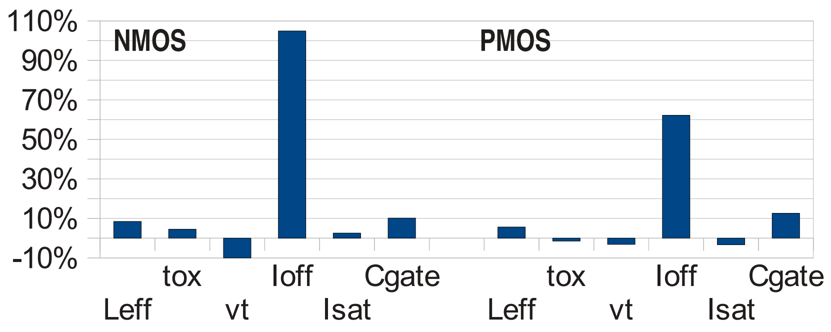


Figure 5.13: Comparison of the 2008 and 2010 process specifications for a commercial 45nm process. The graph plots the percentage increase or decrease for several key parameters.

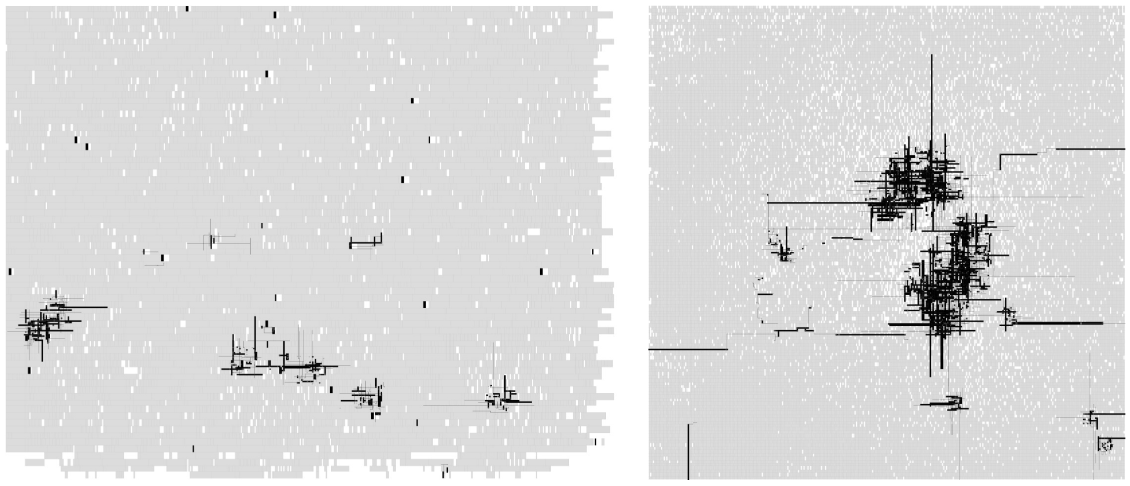


Figure 5.14: Changed area caused by an ECO; $c_{\text{area}} = 27\mu\text{m}^2$, benchmark s38417 (left); and $c_{\text{area}} = 277\mu\text{m}^2$, benchmark mult (right).

The nature of the ECO depends on when the updated information arrives in the product's development cycle. If the information arrives before substantial engineering time is spent, the product may simply be redesigned. In contrast, if significant time has been spent on the design, an ECO may be used that affects a minimal fraction of the design. When the violations are small, the design may be fixed manually; when the violations are large, they may be fixed using CAD tools in *incremental mode*, followed by manual tweaking to correct any remaining timing violations. The design is then verified using sign-off quality tools to verify the timing, power, crosstalk, and design rules, with more accuracy.

The change in the specifications can be substantial. For example, Figure 5.13 shows an example of process parameter change from April 2008 to March 2010, for a commercial 45nm process. The difference in these parameters is not negligible– the transistor off current (I_{off}) increases by over 80%, and the gate capacitance increases by approximately 10%. These two changes alone would have a large impact, by increasing the leakage power by over 80%, the dynamic power by approximately 10%, and the delay by approximately 10%. These are changes that may require substantial modifications in the design to correct the design according to its specifications.

In this section, we focus on late-design cycle ECOs when the changes arrive after the design has been placed and routed, but before it is sent for fabrication. The changes in parameters may also result from retargeting a design to a different, but design-rule compatible, process[1]. We would like to (1) minimize the impact of the ECO, while maintaining a solution that is reasonably optimal after the process change is introduced, and (2) provide a method to modify designs to be robust against late process changes. In this section, these goals are achieved by quantifying the ECO cost in terms of its area cost, and then approximating this relation as a function of layout parameters. The resulting model is fed into an optimization loop which minimizes the ECO cost and power while meeting the timing constraints. In comparison to the prior work in [LG10], an improved ECO area metric and a simplified version of the algorithm is presented in this section, which has improved performance, and faster runtimes.

5.2.1 ECO Cost

Research on ECO and incremental algorithms has focused on traditional costs such as wirelength, timing closure, and the number of changed nets (see for example [CFC07, DA06, RM07]); however, they are too general to be used to distinguish between timing-feasible solutions with very similar power, but very different implementation cost.

[1]Such multi-foundry sourcing is fairly common for large-volume designs.

In practice, the ECO cost is determined by the amount of time, in engineering work time and in tool hours, that is required to perform the ECO. This is the time is spent in checking and correcting: (1) timing errors, (2) problems with the layout, and (3) correcting design rule problems. Note that in modern designs and especially system-on-a-chip (SoC) designs, a large fraction of this verification may be manual.

As a measure of ECO that correlates to the costs in (1)-(3), we approximate these costs using an ECO area metric, c_{area}, that is the amount of layout area changed by the ECO. This area is computed over all layers of the design, and includes the amount of die area, in μm^2 that has been affected by:

- Cell resizing, movement or deletion
- Routing additions and deletions (interconnect and vias).

In this section, these changes are measured using a commercial tool that compares the layout before and after the ECO change, and generates a list of gate changes and movements, and routing modifications. Next, a map of the changed die area is created, and the regions that are affected by the ECO (as in Figure 5.14) are marked. After all ECO changes are considered, the marked regions are added to produce the ECO area cost.

The area ECO cost (c_{area}) is difficult to quantify without performing the ECO itself. These changes are the result of a chaotic interaction between the incremental design tool that is used and the current layout. However, there are intuitive rules that can be considered. The area cost is certainly related to the number of pins that are moved– each one of these pins require re-routing and reconnection. The difficulty in rerouting and reconnecting these pins is also related to the amount of free space in the routing layers above the cell. It is also important to consider the type of cell– some cells are tightly packed, which makes it difficult to access the pins. These ideas provide rules-of-thumb that designers can use to target low-ECO area designs.

For the purposes of guiding the optimization, we propose a method to estimate the effects of these rules-of-thumb on the ECO area cost as ($\hat{c}_{\text{area}}$) associated with changing a cell by performing a quick legalization-like placement check. This method first finds amount of free space around the current cell that is needed to accommodate the size change, and computes the required movements of the current cell and neighboring cells. This provides three pieces of information that are used to find the approximate ($\hat{c}_{\text{area}}$):[2]

- m_1: Number of *dislocated* pins
- m_2: Utilized area over pin bounding box (over all layers)
- m_3: The routing cost (from [TLA10]).

The information m_1 and m_2 are related to the effects of this change on routing. The m_1 are the pins that are moved by the placement check, whose new and old locations *do not overlap*. This measure is important because the change in location will require a rerouting of the connections to the pins, and ECO area cost. The utilized area over the pin bounding box (m_2) is the area above the pin bounding box, the box containing all of the dislocated pins, that is used by the metal layers for routing. Intuitively, larger values of m_2 indicate that it will be more difficult to reroute the m_1 dislocated pins, as the available space for routing is low, resulting in larger ECO costs.

The cost m_3 is a measure of the routability of a library cell called the *cell cost* [TLA10], and is defined as:

$$\begin{aligned}\text{[Cell Cost]} = \text{[\# of pins]} + \sum_{\forall \text{pins} i} 2^{\left(2 - \frac{\text{[Area of pin i]}}{\Theta}\right)} + \\ \frac{1}{2} \sum_{\forall \text{pins} i} \sum_{\forall \text{pins} j \neq i} 2^{\left(2 - \frac{\text{[Area of the Bounding Box of pins i, j]}}{3\Theta}\right)}.\end{aligned} \tag{5.22}$$

In the above, Θ is the minimum cell pin width. The total cost m_3 is then the sum of the cell costs for all moved or re-sized cells. These parameters are then used in the linear model, $\hat{c}_{\text{area}}$, that estimates the true area cost as $\hat{c}_{\text{area}} = \sum_{i=1}^{3} a_i m_i + b$.

[2]Other metrics such as congestion, net bounding boxes, number of changed cells, and the congestion on different metal layers, were also considered for estimating $\hat{c}_{\text{area}}$. The three measures used in this section, m_1, m_2, and m_3 provided the best performance in terms of intuitive appeal, and accuracy.
Also note that the metrics used to estimate ECO cost (m_1 to m_3) differ from [LG10]. These improvements reduce the average normalized error by 4%.

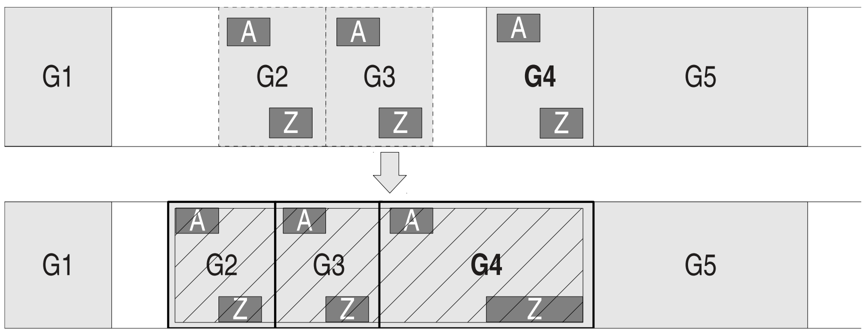


Figure 5.15: ECO example to estimate c_{area}. Gate G4 changes from INV size 1 to INV size 2, dislocating cells G2 and G3. There are 6 pins that are moved by the change, but the number of dislocated pins, $m_1 = 5$, because pin G4/Z still overlaps with its old location.

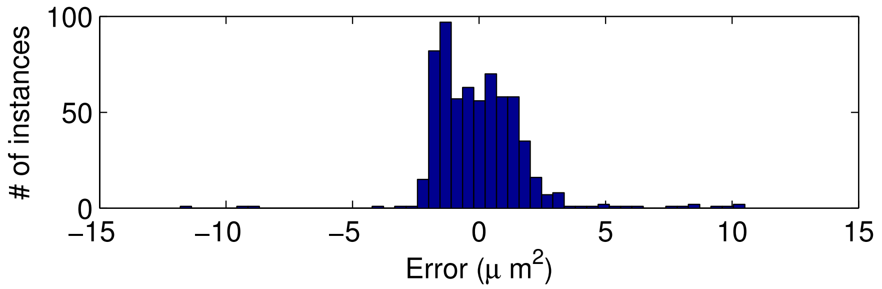


Figure 5.16: Error histogram of the difference between the estimated ECO area values ($\hat{c}_{\text{area}}$) and the actual ECO area values (c_{area}) for 644 data points over the benchmark s35932.

A sample of 644 ECO operations over the benchmark s35932 is used to fit the model, and a least-squares fit of the coefficients a_i is made. Each sample operation consists of changing the size of one gate, and recording the ECO cost, along with the values of m_i. The model parameters are:

$$a_1 = 0.183\ \mu\text{m}^2/\text{pin} \quad a_2 = 4.721 \quad a_3 = 0.123 \quad b = 0.835\ \mu\text{m}^2.$$

The quality of the fit is shown in Figure 5.16, which shows the errors between the estimate $\hat{c}_{\text{area}}$ and actual c_{area}. We shall see in Section 5.2.2.1 and in Tables 5.4 that the fidelity is high; minimizing the estimate $\hat{c}_{\text{area}}$ is effective in minimizing the actual ECO area cost.

We can use this information to estimate the cost of changing the size of a given cell.

For example, consider the case in Figure 5.15. A quick placement check is done to find the values of m_1 to m_3. With the value $m_1 = 5$ (and assuming $m_2 = 0.25$ and $m_3 = 0.5$) the expression gives the estimate of $2.99\mu\mathrm{m}^2$.

These estimates are used to guide the ECO process. Gates in congested areas will result in large estimated ECO costs, as changing the gate will move many neighboring cells (resulting in large values for m_1 and m_3), and require re-routing in a congested area (m_2). Relying on changes with small ECO cost will help to make changes where free space is high and congestion is low.

5.2.2 Solving the redesign problem

Incorporating the ECO cost into the Linear Programming gate sizing framework in [CK05] (see also Section 2.3.10) results in:

$$\begin{array}{ll} \text{minimize} & \sum_{i,k}(e_{ik} + \gamma p_{ik})y_{ik} \\ \text{subject to} & t_i + d_{i0} + \sum_k \delta_{ik} y_{ik} \le t_j, \quad \forall j \in \mathrm{fo}(i) \\ & t_i \le T_{\max}, \quad \forall i \in \mathrm{po} \\ & \sum_k y_{ik} \le 1, \quad \forall i \\ & 0 \le y_{ik} \le 1, \end{array} \tag{5.23}$$

which is applied iteratively. The variables are:

y_{ik}: Assignment variable of gate i to size k	
e_{ik}: ECO area cost estimate for y_{ik}	
t_i: Arrival time for gate i	d_{i0}: Current delay for gate i
δ_{ik}: Δ delay for y_{ik}	p_{ik}: Δ power for y_{ik}

We denote this algorithm LPECO-S, a *simplified* version of the LPECO from [LG10], that minimizes a weighted objective of power and ECO cost. The variables t_i, d_{i0} and δ_{ik} are related to the timing of the design, and they propagate the arrival times down the graph to enforce setup time constraints.[3] $\gamma = .05$, and is a factor used to consider the power, helping to break ties between gates with similar ECO costs. In contrast to [CK05], to account for the downstream delays due to slew effects, the negative change in the slack is used as δ_{ik} in place of the actual delay change. Also, in contrast to [LG10], the restriction preventing neighboring gates to change is dropped.

The variable y_{ik} is an assignment variable that is 1 when gate i is size k in the solution, and 0 otherwise; the sum $\sum_k y_{ik}$ (for each i) is restricted to be less than or equal to 1 to prevent to assignment of a gate to multiple sizes. Note that for a given i, if all $y_{ik} = 0$, the current gate size is kept and not changed. The e_{ik} is the estimated ECO cost related to y_{ik}, if it were performed one gate at a time. The entire ECO cost is estimated by using the assumption that the ECO costs are additive.

As the number of gate sizing candidates is very large, we restrict the search to the gates that have negative slack, and the moves that improve slack (e.g. $\delta_{ik} < 0$). This means that the size of the problem is dominated by the number of possible moves, and not the size of the circuit. Furthermore, to consider the effect of fan-out load, gates are also considered if they are a fan-out of a critical gate. Fan-ins can also be considered to account for slew effects but we ignore them in our current experiments as they have little effect on delay for our benchmarks.

Problem (5.23) may be infeasible when a large number of gate sizings is required to make the design timing-feasible. In these cases, the slack must be maximized iteratively, by solving (5.23) with $T_{\max}$ as the objective.

The solution to (5.23) may have indeterminate assignments, e.g. the y_{ik} may be greater than 0, but less than 1. In these cases, a decision must be made as to whether a gate should be changed, and if so, which size it should be assigned to. A guideline for the indeterminate

[3]This formulation can also consider hold time constraints by adding a second set of timing variables, denoting the earliest arrival time for each gate. Also that design rules such as max transition and max capacitance can be handled in this formulation, by removing the assignments that violate these rules.

assignments in the problem (5.23) can be derived from the value of the optimum:

$$\sum_{i,k}(e_{ik} + \gamma p_{ik})y^{\star}_{ik}. \tag{5.24}$$

In the worst case, each of the non-zero $y^{\star}_{ik}$ will be assigned by setting each non-zero y_{ik} to 1 ($= \lceil y^{\star}_{ik} \rceil$):[4]

$$\sum_{i,k}(e_{ik} + \gamma p_{ik})\lceil y^{\star}_{ik} \rceil \tag{5.25}$$

$$= \sum_{\forall y_{ik}>0} (e_{ik} + \gamma p_{ik}). \tag{5.26}$$

The difference between (5.25) and (5.24) provides an approximation for the suboptimality due to rounding. As this equation is linear, we can approximate the suboptimality for the case of $\delta_{ik} > 0$[5] as

$$\sum_{\forall y_{ik}>0} (e_{ik} + \gamma p_{ik})(1 - y^{\star}_{ik}) \tag{5.27}$$

This suboptimality comes from the difference between the continuous and the integer solutions to the problem.

We can reduce this gap by considering other sizes that may reduce the suboptimality in (5.27). Formally, if we are given an indeterminate assignment y_{ik}, the suboptimality is minimized over k by choosing the size s as:

$$s = \text{argmin}_{\{j|\ \delta_{ij} \geq y_{ik}\delta_{ik}\}} \left\{e_{ij} + \gamma p_{ij}\right\}, \tag{5.28}$$

where $y_{ik}\delta_{ik}$ is the amount of timing improvement that was allocated by the LPECO-S algorithm. The idea is to find an alternate size that provides the same timing improvement, with less cost. In LPECO-S, values of $y > 0.2$ are applied in (5.23).[6] As these approximations may prevent the algorithm from converging in one iteration, it is iterated until the timing is met.

[4]These expressions hold for $\delta_{ik} < 0$, when only delay-*improving* moves are considered. A similar expression can be derived for cases where the δ_{ik} are unrestricted.

[5]When all $\delta_{ik} < 0$, only delay-*improving* moves are considered. A similar expression can be derived for cases where the δ_{ik} are unrestricted.

[6]However in rare cases, a tolerance of $y > 0.2$ will lead to no changed cells. In these cases, the tolerance is gradually reduced until an ECO change is made. In the slack maximization analogue of the problem, values of $y > 0.1$ are applied.

5.2.2.1 Experimental Results

This algorithm is tested on the ISCAS '85 and '89 benchmarks, a 64-bit multiplier, and the Open Cores ALU [OPE]. These benchmarks are synthesized to the Nangate 45nm Library [NANb], and placed, routed and optimized[7] on different sized dies to provide 70% and 90% congestion and experiment on the effects of congestion and free space on the ECO. Table 5.3 gives information about these benchmarks for the nominal process parameters.

The library is then adjusted for the following parameter changes, using the Liberty NCX tool [Syn10]

v_t: nmos -10%, pmos -5%	t_ox: nmos +5%, pmos -5%
c_gate: nmos +10%, pmos +10%	l_eff: nmos +5%, pmos +5%.

These changes are derived from a two year change in a commercial 45nm process as in [LG10], and they create a negative-slack timing violation that is repaired using the algorithm LPECO-S. For comparison, the algorithm is run without the ECO costs (LP No Eco Cost), and the commercial design tool is also used to repair the timing violation in the *post-route* incremental mode with the optimization effort set to high. The commercial tool has the ability to add buffers, on top of sizing gates, and while this provides an advantage over LPECO-S, we show that LPECO-S still performs better. All timing and power data in this section is generated using this commercial design tool.

The algorithm LPECO-S is implemented using C++ and the linear programming solver in MOSEK [MOS]. The ECO cost estimates are also programmed in C++, and the final ECO design is created using the commercial design tool.

Results are shown in Table 5.4. The c_area and p_l represent the actual ECO area cost and leakage power, respectively. The "iters" column gives the number of iterations that the LPECO-S algorithm needs to find a timing-feasible solution. The slacks in the table are computed after the parameter changes. In all of the cases, the algorithm LPECO-S is

[7]Note that this is a newer version of the tool used in [LG10]. In comparison to the benchmarks in [LG10], these benchmarks were more heavily optimized to produce a nominal design.

	70% Congestion				90% Congestion			
	cells	delay [ns]	power [μW]	die area [μm^2]	cells	delay [ns]	power [μW]	die area [μm^2]
c2670	912	0.589	8.0	1175	887	0.619	7.5	916
c3540	1538	1.118	10.1	1987	1423	1.053	12.6	1549
c5315	2038	1.046	14.5	2716	1899	0.973	16.4	2111
c6288	3451	2.290	23.7	3862	3128	2.226	22.8	2998
c7552	3029	0.925	25.7	3637	2773	0.957	23.1	2825
s13207	1183	0.612	22.8	3620	1083	0.618	22.6	2815
s35932	10570	3.054	144.5	23040	9842	4.899	136.9	17916
s38417	8820	1.793	133.0	21674	7744	1.740	129.7	16861
s38584	7908	4.366	103.7	16886	7131	2.946	98.8	13143
s5378	1286	0.923	14.2	2370	1052	0.881	13.5	1843
alu	13978	3.721	74.0	16242	12022	3.751	69.2	12640
mult	49141	6.095	558.2	54091	46701	7.324	401.3	42059

Table 5.3: Benchmark Information for the nominal process

able to find a timing feasible solution, while the commercial tool is unable to do so in 7 of the cases.

In the cases where both the LPECO-S and the commercial tool find a timing feasible solution, the LPECO-S provides significant reductions. On average, the area cost c_{area} improves by 93%; this performance is affected by the congestion; while the improvement is 99% for the 70% congestion benchmarks, it is 87% for the 90% congestion benchmarks.[8] This is due to the fact that it is more difficult to predict ECO area costs when the congestion is high, and the interactions between neighboring cells and interconnect increase. The difference in power between the commercial solution and the LPECO-S solution is very small (.17%), indicating that the ECO cost is needed to distinguish between solutions that are similar in power, but have different ECO implementation costs.

In the comparison with the ECO cost disabled (LP Without ECO Cost), LPECO-S

[8]Note that the difference in performance, compared to [LG10], is due to the improvements in the performance of the commercial tool.

70% Congestion																	
			LPECO-S				Commercial					LP (No ECO Cost)					
	slack$_{\text{init}}$ [ns]	p_{init} [μW]	slack [ns]	c_{area} [μm^2]	p_{l} [μW]	iter	slack [ns]	c_{area} [μm^2]	Δ	p_{l} [μW]	Δ	slack [ns]	c_{area} [μm^2]	Δ	p_{l} [μW]	Δ	iter
c2670	-0.028	8.0	0.000	0.028	7.97	2	0.000	1.51	98%	8.0	0.1%	0.001	12.55	100%	7.8	-2.2%	1
c3540	-0.053	10.1	0.002	5.249	10.26	3	-0.022	16.17	*	10.7	*	0.006	17.36	70%	9.96	-2.9%	3
c5315	-0.048	14.5	0.001	1.644	14.51	4	-0.022	7.17	*	14.5	*	0.001	7.84	79%	14.29	-1.5%	3
c6288	-0.113	23.7	0.000	4.596	23.87	2	-0.071	4.77	*	23.9	*	0.003	36.32	87%	22.69	-4.9%	3
c7552	-0.045	25.7	0.005	1.506	25.72	4	-0.002	16.53	*	25.9	*	0.002	43.41	97%	24.85	-3.4%	2
s13207	-0.020	22.8	0.095	0.014	22.84	1	0.095	1.65	99%	22.8	0.0%	0.095	0.01	0%	22.84	0.0%	1
s35932	-0.094	144.5	0.119	0.015	144.54	1	0.119	9.06	100%	144.6	0.0%	0.120	27.07	100%	144.31	-0.2%	1
s38417	-0.088	133.0	0.051	0.015	133.05	1	0.051	4.04	100%	133.1	0.0%	0.051	0.01	0%	133.05	0.0%	1
s38584	-0.084	103.7	0.344	0.029	103.69	1	0.344	19.93	100%	103.8	0.1%	0.004	31.42	100%	103.26	-0.4%	2
s5378	-0.038	14.2	0.050	0.013	14.21	1	0.050	1.15	99%	14.3	0.3%	0.050	0.01	0%	14.21	0.0%	1
alu	-0.139	73.9	0.015	0.013	73.95	1	0.015	6.13	100%	74.0	0.1%	0.015	0.01	0%	73.95	0.0%	1
mult	-0.316	558.2	0.154	0.013	558.20	1	0.154	14.82	100%	558.2	0.0%	0.149	37.85	100%	557.44	-0.1%	4
AVG	-0.089					1.8			99%		0.1%			61%		-1.3%	1.9

90% Congestion																	
c2670	-0.029	7.6	0.000	0.06	7.56	2	0.007	11.58	100%	7.7	1.4%	0.006	17.5	100%	7.25	-4.1%	6
c3540	-0.057	12.6	0.000	5.130	12.68	5	-0.016	31.07	*	13.1	*	0.002	47.19	89%	11.98	-5.5%	3
c5315	-0.047	16.4	0.001	0.070	16.44	2	-0.032	8.34	*	16.6	*	0.000	10.16	99%	16.21	-1.4%	1
c6288	-0.106	22.4	0.004	8.295	23.07	4	-0.086	3.15	*	22.9	*	0.005	47.28	82%	21.33	-7.6%	3
c7552	-0.040	23.2	0.013	2.636	23.15	2	0.010	9.82	73%	23.2	0.3%	0.003	10.83	76%	22.94	-0.9%	1
s13207	-0.018	22.6	0.023	0.013	22.62	1	0.088	1.10	99%	22.6	0.0%	0.023	0.01	0%	22.62	0.0%	1
s35932	-0.309	136.9	0.069	0.015	136.91	1	0.069	12.71	100%	136.9	0.0%	0.097	5.83	100%	136.84	-0.1%	2
s38417	-0.069	129.8	0.029	0.073	129.72	5	0.029	15.20	100%	129.8	0.1%	0.029	0.07	0%	129.72	0.0%	5
s38584	-0.128	98.8	0.495	0.014	98.80	1	0.778	9.78	100%	98.8	0.0%	0.579	12.99	100%	98.69	-0.1%	1
s5378	-0.025	13.6	0.048	0.079	13.54	1	0.049	3.52	98%	13.6	0.1%	0.031	0.01	-450%	13.54	0.0%	1
alu	-0.187	69.2	0.045	0.022	69.24	1	0.045	4.32	99%	69.3	0.1%	0.045	0.46	95%	69.24	0.0%	2
mult	-0.028	401.3	0.350	372.359	401.31	1	0.348	427.3	13%	401.4	0.0%	0.035	277.1	-34%	401.22	0.0%	1
AVG	-0.087					2.2			87%		0.2%			21%		-1.6%	2.3

*denotes infeasible designs

Table 5.4: Experimental Results comparing LPECO-S with the commercial tool

yields a significantly better ECO area cost in the majority of cases. In the 70% and 90% congestion cases, the area cost reduction was, on average, 61% and 21% respectively. There are a couple cases in the 90% where the LP Without ECO Cost performs better than the LPECO-S; however, these are not shortcomings of the algorithm, and have more to do with the difficultly in predicting the ECO cost at high congestion. In the s5378 case, the absolute difference is negligible (.06μm^2), and in the mult case, the algorithm is unable to predict the effects of incremental routing; the LPECO-S changes just one gate, from size 1 to size 2, while the LP Without ECO sizes 9 gates over an area with similar routing utilization (m_2). This difference is primarily due to routing changes, and is a comment on the difficulty of predicting routing changes.

The LP Without ECO Cost is able to improve the power of the design by an average of 1.5%. This is because the objective here is to fix the timing violation with the greatest power benefit. However, this is not ideal for the ECO case, as the focus is on minimal disturbance, and the greatest power savings may result in larger ECO costs (e.g. c6288 90% congestion). Furthermore, the power difference is negligible in the larger designs.

The runtime for this algorithm is dominated by the interface from the commercial tool to LPECO, which is needed to transfer timing information and gate sensitivity information. This sensitivity information is needed for any sizer, as the comparisons between competing gates must be made in the process of optimization. Each iteration of LPECO-S takes between 6 and 280 seconds, while solving the linear program in LPECO-S takes between .02 to 2.1 seconds for all benchmarks (excluding the time used by the commercial physical design tool). This is significantly faster than in [LG10], which required up to 103 seconds. In comparison, running the LP (without the ECO cost) takes between 1 and 71 seconds per iteration. The runtime of the commercial tool is comparable to the runtime needed to by the same commercial tool to perform the ECO, and ranges between 24 seconds and 23 minutes.

5.2.3 Creating initial designs

In some cases, there may be several target foundries that may be targeted for production, or there may be uncertainty in the manufacturing process parameters; there may be an idea of which parameters may fluctuate, and which parameters would be controlled well in future. These situations motivate the initial design problem, where an initial design is created that can tolerate future manufacturing process fluctuations.

We consider the following formulation of this problem. Suppose, as a starting point, we have an original, optimized design that has undergone placement and routing, and is timing-feasible in the nominal case, and the information on potential manufacturing-process changes is available in the form of corners, scenarios, or samples. As designing for all possible cases results in an overly conservative design with a large power, the goal of the initial design is: (1) the resulting design is timing feasible in the nominal corner; (2) the difference between the power of the original design, ($p_{y_{\text{orig}}}$), and the power of the new initial design is within a tolerance β; and (3) the need for a future ECO is reduced.

As a heuristic to meet these goals, we propose the following linear programming problem to solve the initial design problem:

$$\begin{array}{ll} \text{minimize} & t_{\max} \\ \text{subject to} & \sum_{i,k} p_{ik} y_{ik} \leq (\beta \cdot p_{y_{\text{orig}}}) \\ & t_i + \\ & \frac{1}{N+1} \sum_{n=0}^{N} (d_{i0}^{(n)} + \sum_k \delta_{ik}^{(n)} y_{ik}) \leq t_j, \ \forall i \in \text{fo}(j) \\ & t_i \leq t_{\max}, \quad \forall i \in \text{po} \\ & \sum_k y_{ik} \leq 1, \ \forall i, \quad 0 \leq y_{ik} \leq 1. \end{array} \tag{5.29}$$

The meanings of the variables y, δ and t are the same as in the LPECO-S formulation in (5.23). N is the total number of corners that are used, and n is the index used for the corners, with $n = 0$ denoting the nominal corner. The superscript $^{(n)}$ refers to the corner associated with the delay $d_{i0}^{(n)}$ or change in delay $\delta_{ik}^{(n)}$. Cell options that are delay improving in the nominal process ($\delta_{ik}^{(0)} < 0$) are considered as candidate cell changes. This formulation is similar in concept to [BKP05], where the delay for each gate is converted to a statistical delay. In this case, the manufacturing uncertainty is accounted for by adjusting the delay to be the average over the given scenarios. In contrast to (5.23) where the power plays a role in the objective, in this formulation it is used as a constraint.

Note that the input to the algorithm is a design that is timing-feasible in the nominal scenario. Also, as in Section 5.2.2, only moves that are delay-improving in the nominal process ($\delta_{ik}^{(0)} < 0$), are considered.

In the above, the variation information is assumed to be in the form of N corners, scenarios or samples. This flexible way to describe variations is useful when the information on the manufacturing parameters is scarce; there may not be accurate distributions available for modeling future variations. These corner-type specifications can then describe the kinds of variations that the designer would like to hedge against.

The algorithm (5.29) is similar to a statistical version of guardband. Given an amount of power β that the designer is willing to spend, the design maximizes the average slack over all of the scenarios using gate sizing. In effect, the *expected slack* is maximized to decrease the need for future ECO.[9] This algorithm is not applied iteratively and is run only once.

After (5.29) is solved, the y_{ik} are mapped to gate sizes by applying the methods in [LG10]. The indeterminate assignments are remapped if possible, and the candidates are sorted by sensitivity with values of $y_{ik} > 0.01$ as eligible for change. The changes are made until the power budget (e.g. the tolerance β) is met. Furthermore, each gate sizing is checked to ensure that it does not cause timing violations in the nominal process parameters, and is skipped if timing violations are created.

Note that this work is different from work in statistical gate sizing. In this situation, the manufacturing process changes may be impossible to predict using distributions, and the power and timing effects may be impossible to model statistically. This method provides

[9]While ECO area costs can be added to this formulation, we find that the improvements are not significant, as improving the slack and the future feasibility is the dominating effect.

a method to create initial designs with little statistical information, that are robust to manufacturing process changes, and is also simple enough to implement on top of current tools.

5.2.4 Experimental results

This algorithm is tested on the benchmarks in Table 5.3. The manufacturing process changes are assumed to be random variables with zero-mean Gaussian distributions, and the following standard deviations:

v_{th}: 5% t_{ox}: 2.5% C_{gate}: 2.5% l_{gate}: 2.5%.

The variations are the same across all gates (e.g. all transistors have the same increase in v_{th}, t_{ox}, C_{gate} and l_{gate}). However, the variations between the PMOS and NMOS transistors for the v_{th} and t_{ox} parameters are considered to be independent. This model may be pessimistic, as more information may be available, such as the direction of the variation. For example, the foundry might give the current and target PMOS v_{th}, implying that the final value would be between the current and the target values.

10 samples (set 1) are randomly generated according to the distribution above and are used in the LPECO-ID algorithm. These samples, along with the nominal process parameters, are used to create the initial design. A *separate* set of 10 different independent samples (set 2) is generated using the same distribution to evaluate the quality of the LPECO-ID algorithm. The two sets of samples are generated independently to simulate a realistic design condition. While a rough idea of the variations for the manufacturing process parameters may be known, the actual values are unavailable until after the initial design is set.

This initial design method (LPECO-ID) is implemented using C++ and the linear programming solver in MOSEK [MOS]. The ECO cost estimates are also programmed in C++, and the final ECO design is created using the commercial design tool for each of the manufacturing process variations. As a comparison, the same β budget is used to create a

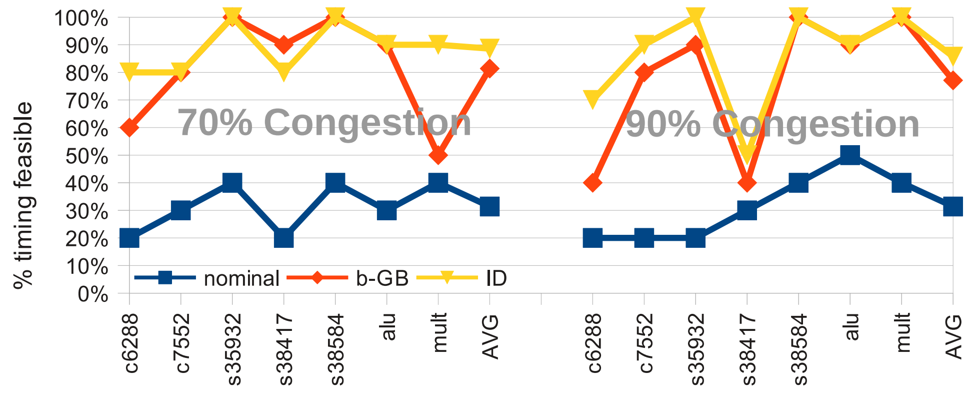


Figure 5.17: Results comparing the feasibility of the original design, a modified design using a β-guardband, and the Initial Design Method (ID). The Initial design method improves the feasibility substantially.

guardband β-GB by maximizing the slack in the nominal scenario.

The results in Figure 5.17 show that the LPECO-ID method drastically reduces the need to perform an ECO compared to the commercial tool. An ECO is needed just 13% of the time ID algorithm, while it is needed 21% of the time with β-GB, and 68% with the original design. In the s38417 70% congestion case, the β-GB performs slightly better, but this the only exception. This shows that this method is effective in hedging against future changes.

5.2.5 Summary

In this section, we present the idea of ECO cost to quantify the amount of time that is needed to validate an ECO operation. We then propose a novel method for performing ECO gate sizing, and give models for the ECO that can be incorporated into the optimization procedure. This leads to results that outperform a leading commercial design tool in reducing the amount of area that is changed by the ECO by an average of 89%. In addition, a novel method for creating initial designs is presented that drastically reduces the probability that a redesign is needed in the future, between 10% and 80%.

CHAPTER 6

Conclusions

In this dissertation, we have addressed the following questions:

1. Which discrete sizing heuristic is the best?
2. What are the potential benefits of optimizing statistical power?
3. Are there sizing methods that can handle statistical delay directly?
4. What is the effect of novel technologies on the gate sizing problem?
5. What is the impact of the range and precision of standard library cells on the optimal power?
6. What methods can be used to perform incremental gate sizing for post-layout designs?

The first question was addressed in Chapter 3. A survey of gate sizing methods was performed, and several discrete gate sizing methods were compared [FD85, NDO03, CK05, LH09], which collectively cover a variety of methods: dynamic programming [LH09], linear programming [CK05, NDO03], greedy heuristics [FD85], slack allocation [NDO03], and assignment methods [CK05]. The results show that one of the oldest algorithms [FD85] performs competitively with the best method [CK05] that was compared [LG12, MLG]. When taken in conjunction with the results from the recent sizing contest [OAA12], and the results from [WD09], this suggests that there should be research towards a discrete sizing method that performs *consistently* well. Furthermore, with the ease and simplicity of [FD85], the greedy TILOS algorithm should be used as a common point for comparison in gate sizing research.

The second question, whether full statistical power optimization has significant advantages over deterministic power optimization, was addressed in Section 4.2. The intuition is that while the values of statistical power and deterministic power are very different (nearly an order of magnitude apart), the power vs. gate size sensitivities of each are similar. Bounds were derived to quantify the benefits of using full statistical power optimization which show that with smaller variations ($\sigma_L = 1$nm (dtd)/ .5nm (wid) and $\sigma_{v_t} = 4.7\%$ (dtd & wid)), the benefits are less than 5% for a set of commonly used benchmarks [CGL10]. For larger variations, there may be a benefit in optimizing the mean plus three-sigma measure of the power, but there is a linear proxy measure that can be used in current tools to optimize within 4% of the true optimum [CGL10]. Thus, we have shown that full statistical optimization is not necessary.

The third question addresses the statistical delay. With the increased parametric variability from the decreasing control in gate dimensions, dopant concentrations, and oxide thicknesses, the variations in the delay are too large to treat deterministically [Vis03]. Prior art [BVS02, CPR04, MO04, SSZ05, PYK05] relied on heuristics to deal with the statistical delay quantities, but the question remained as to whether a rigorous formulation of the statistical delay could be used in mathematical-programming based gate sizing methods. In Section 4.3.2 we introduced a measure of statistical delay, the mean excess delay, which is an upper bound on the quantile delay, and also has a convex formulation [CLV08]. This was used to perform statistical delay minimization, and it performs well compared to the competing methods [CLV08]. This method can also be used to perform power optimization with statistical delay constraints, which is something that the competing "padded" methods [CPR04, MO04, SSZ05, PYK05] cannot handle directly without fitting.

The fourth and fifth questions were addressed in Section 5.1. Prior art dealt with methods for choosing library cell granularity [BKK00, BKK98, SG06, ART09], but could not relate the granularity of a cell library with the resulting power vs. delay tradeoff in a design. We developed a theory that explains the suboptimality associated with the precision of gate sizes and threshold voltages in a standard cell library [LGa]. The suboptimality is due to the convexity of the power vs. delay tradeoff and can be used to predict the suboptimalities relative to a given set of library cells; it can also be used to choose both the range and precision of the library cells [LGa]. An interesting result of this analysis is that the use of different technologies may be able to provide a better power vs. delay tradeoff than either of the technologies alone, and this helps to understand the impact of future technologies, such as multi-terminal devices [LGa].

The sixth and final question was addressed in Section 5.2. While there are many methods for correcting the timing violations in a design (see, for example, [CK05, Hel09]), there are no methods that also considered the impact of the changes– how intrusive is the resulting change and disturbance in the design? We introduced the ECO cost metric, an intuitive measure of how invasive an engineering change order is [LG10, LGb]. This measure is used to guide a linear-programming based incremental sizing method. Compared to a leading commercial design tool, it dramatically reduces the amount of area that is changed by the ECO by an average of 89% [LGb]. In addition, a novel method for creating initial designs is presented that drastically reduces the probability that a redesign is needed in the future, by between 10% and 80% [LGb].

References

[ALQ11] Hamed Abrishami, Jinan Lou, Jeff Qin, Juergen Froessl, and Massoud Pedram. "Post Sign-off Leakage Power Optimization." In *Proc. Design Automation Conference*, 2011.

[ANC04] A.I. Abou-Seido, B. Nowak, and C. Chu. "Fitted Elmore delay: a simple and accurate interconnect delay model." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **12**(7):691–696, 2004.

[ART09] R. Afonso, M. Rahman, H. Tennakoon, and C. Sechen. "Power efficient standard cell library design." In *IEEE Dallas Circuits and Systems Workshop,(DCAS)*, pp. 1–4, 2009.

[AV95] D. S. Atkinson and P. M. Vaidya. "A Cutting Plane Algorithm for Convex Programming that Uses Analytic Centers." *Mathematical Programming*, **69**(1):1–44, July 1995.

[BAB07] R. Berridge, RM Averill, AE Barish, MA Bowen, PJ Camporese, J. DiLullo, PE Dudley, J. Keinert, DW Lewis, RD Morel, et al. "IBM POWER6 microprocessor physical design and design methodology." *IBM Journal of Research and Development*, **51**(6):685–714, 2007.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. "Combinatorial Profiles of Sequential Benchmark Circuits." In *Proc. Int. Symp. Circuits and Systems*, pp. 1929–1934, May 1989.

[BC09] J. Bhasker and R. Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer Verlag, 2009.

[Bel57] R.E. Bellman. *Dynamic programming.* 1957.

[Ber05] D.P. Bertsekas. *Dynamic Programming and Optimal Control, Volume I.* Athena Scientific, 2005.

[BJ90] Michel R. C. M. Berkelaar and Jochen A. G. Jess. "Gate sizing in MOS digital circuits with linear programming." In *EURO-DAC '90: Proceedings of the conference on European design automation*, pp. 217–221, 1990.

[BKK98] F. Beeftink, P. Kudva, D. Kung, and L. Stok. "Gate-size selection for standard cell libraries." *Proc. Int. Conf. Computer-Aided Design*, pp. 545–550, Nov 1998.

[BKK00] F. Beeftink, P. Kudva, D.S. Kung, R. Puri, and L. Stok. "Combinatorial cell design for CMOS libraries." *Integration, the VLSI Journal*, **29**(1):67–93, 2000.

[BKM93] K.D. Boese, A.B. Kahng, B.A. Mccoy, and G. Robins. "Fidelity and near-optimality of Elmore-based routing constructions." In *Proc. Int. Conf. Computer Design*, pp. 81–84, 1993.

[BKM07] S. M. Burns, M. Ketkar, N. Menezes, K. A. Bowman, J. W. Tschanz, and V. De. "Comparative analysis of conventional and statistical design techniques." In *Proc. Design Automation Conference*, pp. 238–243, 2007.

[BKP05] S.P. Boyd, S.J. Kim, D.D. Patil, and M.A. Horowitz. "Digital Circuit Optimization via Geometric Programming." *Operations Research*, **53**(6):899, 2005.

[BV04] S.P. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[BV05] Sarvesh Bhardwaj and Sarma B. K. Vrudhula. "Leakage minimization of nano-scale circuits in the presence of systematic and random variations." In *Proc. Design Automation Conference*, pp. 541–546, 2005.

[BVS02] X. Bai, C. Visweswariah, P.N. Strenski, and D.J. Hathaway. "Uncertainty-aware circuit optimization." In *Proc. Design Automation Conference*, pp. 58–63, 2002.

[Cad] Cadence. "Encounter v. 10.11." http://www.cadence.com.

[Cad07] Cadence Design Systems Inc. "Encounter RTL Compiler 6.20." `http://www.cadence.com/`, 2007.

[CCH96] Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, and Chandu Visweswariah. "Optimization of custom MOS circuits by transistor sizing." In *Proc. Int. Conf. Computer-Aided Design*, pp. 174–180, 1996.

[CCH98] A.R. Conn, P.K. Coulman, R.A. Haring, G.L. Morrill, C. Visweswariah, and C.W. Wu. "JiffyTune: Circuit optimization using time-domain sensitivities." *IEEE Trans. on Computer-Aided Design*, **17**(12):1292–1309, 1998.

[CCW99] C.P. Chen, CCN Chu, and DF Wong. "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation." *IEEE Trans. on Computer-Aided Design*, **18**(7):1014–1025, 1999.

[CFC07] Yen-Pin Chen, Jia-Wei Fang, and Yao-Wen Chang. "ECO timing optimization using spare cells." In *Proc. Int. Conf. Computer-Aided Design*, pp. 530–535, Piscataway, NJ, USA, 2007. IEEE Press.

[CGL09] J. Cong, P. Gupta, and J. Lee. "On the futility of statistical power optimization." In *Proc. Asia and South Pacific Design Automation Conference*, pp. 167–172, 2009.

[CGL10] J. Cong, P. Gupta, and J. Lee. "Evaluating statistical power optimization." *IEEE Trans. on Computer-Aided Design*, **29**(11):1750–1762, 2010.

[CH99] J. Cong and L. He. "Theory and algorithm of local-refinement-based optimization with application to device and interconnect sizing." *IEEE Trans. on Computer-Aided Design*, **18**(4):406–420, 1999.

[Cha90] PK Chan. "Algorithms for library-specific sizing of combinational logic." *Proc. Design Automation Conference*, pp. 353–356, 1990.

[CHM96] O. Coudert, R. Haddad, and S. Manne. "New algorithms for gate sizing: a comparative study." In *Proc. Design Automation Conference*, pp. 734–739, 1996.

[CHV96] A.R. Conn, R.A. Haring, C. Visweswariah, P.K. Coulman, and G.L. Morrill. "Optimization of custom MOS circuits by transistor sizing." In *International Conference on Computer-Aided Design*, p. 174, 1996.

[CK91] HY Chen and SM Kang. "iCOACH: a circuit optimization aid for CMOS high-performance circuits." *Integration, the VLSI journal*, **10**(2):185–212, 1991.

[CK05] D. G. Chinnery and K. Keutzer. "Linear programming for sizing, Vth and Vdd assignment." In *Proc. Int. Conf. Low Power Electronics and Design*, pp. 149–154, 2005.

[CK07] D. Chinnery and K.W. Keutzer. *Closing the power gap between ASIC & custom: tools and techniques for low power design.* Springer-Verlag New York Inc, 2007.

[CLV08] Jason Cong, John Lee, and Lieven Vandenberghe. "Robust gate sizing via mean excess delay minimization." In *Proc. Int. Conf. Physical Design*, pp. 10–14, 2008.

[Cou96] O. Coudert. "Gate sizing: A general purpose optimization approach." In *Proc. Design, Automation and Test in Europe*, p. 214, 1996.

[Cou97] O. Coudert. "Gate sizing for constrained delay/power/area optimization." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **5**(4):465–472, Dec 1997.

[CPO02] M. Celik, L. Pileggi, and A. Odabasioglu. *IC interconnect analysis.* Springer Netherlands, 2002.

[CPR04] Seung Hoon Choi, Bipul C. Paul, and Kaushik Roy. "Novel sizing algorithm for yield improvement under process variation in nanometer technology." pp. 454–459, 2004.

[CS96] De-Sheng Chen and Majid Sarrafzadeh. "An exact algorithm for low power library-specific gate re-sizing." In *Proc. Design Automation Conference*, DAC '96, pp. 783–788, 1996.

[CS00] Chunhong Chen and Majid Sarrafzadeh. "Power reduction by simultaneous voltage scaling and gate sizing." In *Proc. Asia and South Pacific Design Automation Conference*, ASP-DAC '00, pp. 333–338, 2000.

[CS02] C. Chen and M. Sarrafzadeh. “Simultaneous voltage scaling and gate sizing for low-power design.” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, **49**(6):400–408, 2002.

[CSH95] W. Chuang, S.S. Sapatnekar, and I.N. Hajj. “Timing and Area Optimization for Standard-Cell VLSI Circuit Design.” *IEEE Trans. on Computer-Aided Design*, **14**:308, 1995.

[CSS05] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester. “Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation.” In *Proc. Int. Conf. Computer-Aided Design*, pp. 1023–1028, 2005.

[CWC05] H. Chou, Y.H. Wang, and C.C.P. Chen. “Fast and effective gate-sizing with multiple-Vt assignment using generalized Lagrangian Relaxation.” In *Proc. Asia and South Pacific Design Automation Conference*, pp. 381–386, 2005.

[CZN05] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. “Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay functions.” In *Proc. Design Automation Conference*, pp. 71–76, 2005.

[DA89] Z.J. Dai and K. Asada. “MOSIZ: A two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates.” In *Proc. Custom Integrated Circuits Conference*, pp. 17–3, 1989.

[DA06] Shantanu Dutt and Hasan Arslan. “Efficient timing-driven incremental routing for VLSI circuits using DFS and localized slack-satisfaction computations.” In *Proc. Design, Automation and Test in Europe*, pp. 768–773, 2006.

[DBN97] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela, and J. Dunning. “Transistor-level sizing and timing verification of domino circuits in the Power PCTM microprocessor.” In *Proc. Int. Conf. Computer Design*, pp. 143–148, oct 1997.

[DS06] A. Davoodi and A. Srivastava. “Variability driven gate sizing for binning yield optimization.” In *Proc. Design Automation Conference*, pp. 959–964, 2006.

[Elm48] WC Elmore. “The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers.” *Journal of Applied Physics*, **19**:55–63, 1948.

[FA08] P. Feldmann and S. Abbaspour. “Towards a more physical approach to gate modeling for timing, noise, and power.” In *Proc. Design Automation Conference*, pp. 453–455, 2008.

[FD85] JP Fishburn and AE Dunlop. “TILOS: A Posynomial Approach to Transistor Sizing.” *Proc. Int. Conf. Computer-Aided Design*, 1985.

[FF95] P. Feldmann and R.W. Freund. "Efficient linear circuit analysis by Pade approximation via the Lanczos process." *IEEE Trans. on Computer-Aided Design*, **14**(5):639–649, may 1995.

[GKK10] Puneet Gupta, Andrew Kahng, Amarnath Kasibhatla, and Puneet Sharma. "Eyecharts: Constructive Benchmarking of Gate Sizing Heuristics." In *Proc. Design Automation Conference*, 2010.

[GKS04] Puneet Gupta, Andrew B. Kahng, Puneet Sharma, and Dennis Sylvester. "Selective gate-length biasing for cost-effective runtime leakage control." In *Proc. Design Automation Conference*, pp. 327–330, 2004.

[GKS06] P. Gupta, A.B. Kahng, P. Sharma, and D. Sylvester. "Gate-length biasing for runtime-leakage control." *IEEE Trans. on Computer-Aided Design*, **25**(8):1475–1485, 2006.

[GTP97] R. Gupta, B. Tutuianu, and L.T. Pileggi. "The Elmore delay as a bound for RC trees with generalized input signals." *IEEE Trans. on Computer-Aided Design*, **16**(1):95–104, 1997.

[GVV05] MR Guthaus, N. Venkateswaran, C. Visweswariah, and V. Zolotov. "Gate sizing using incremental parameterized statistical timing analysis." *Proc. Int. Conf. Computer-Aided Design*, pp. 1029–1036, 2005.

[GVZ05] M.R. Guthaus, N. Venkateswaran, V. Zolotov, D. Sylvester, and R.B. Brown. "Optimization objectives and models of variation for statistical gate sizing." In *Proc. Great Lakes symposium on VLSI*, pp. 313–316, 2005.

[Hel09] Stephan Held. "Gate sizing for large cell-based designs." In *Proc. Conf. on Design, Automation and Test in Europe*, pp. 827–832, 2009.

[HGS97] R. Haddad, L.P.P.P. van Ginneken, and N. Shenoy. "Discrete drive selection for continuous sizing." In *Proc. Int. Conf. Computer Design*, pp. 110–115, oct 1997.

[HHS11] Y.L. Huang, J. Hu, and W. Shi. "Lagrangian relaxation for gate implementation selection." In *Proc. Int. Conf. Physical Design*, pp. 167–174, 2011.

[HKH07] Shiyan Hu, Mahesh Ketkar, and Jiang Hu. "Gate sizing for cell library-based designs." In *Proc. Design Automation Conference*, pp. 847–852, 2007.

[HKH09] S. Hu, M. Ketkar, and J. Hu. "Gate sizing for cell-library-based designs." *IEEE Trans. on Computer-Aided Design*, **28**(6):818–825, 2009.

[HO01] Masanori Hashimoto and Hidetoshi Onodera. "Post-layout transistor sizing for power reduction in cell-based design." In *Proc. Asia and South Pacific Design Automation Conference*, pp. 359–365, 2001.

[HYH99] M. Hansen, H. Yalcin, and J.P. Hayes. “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering.” *IEEE Design and Test*, **16**(3):72–80, July-Sept 1999.

[ITR] “International Technology Roadmap for Semiconductors – Design.” Available from http://www.itrs.net.

[JB08] S. Joshi and S. Boyd. “An efficient method for large-scale gate sizing.” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, **55**(9):2760–2773, 2008.

[JK10] K. Jeong and A.B. Kahng. “Methodology from chaos in IC implementation.” In *Proc. Int. Conf. Quality Electronic Design*, pp. 885–892, 2010.

[KC00] J.T. Kao and A.P. Chandrakasan. “Dual-threshold voltage techniques for low-power digital circuits.” *IEEE Journal of Solid-State Circuits*, **35**(7):1009–1018, 2000.

[Ken38] MG Kendall. “A new measure of rank correlation.” *Biometrika*, **30**(1-2):81, 1938.

[KKS00] K. Kasamsetty, M. Ketkar, and S.S. Sapatnekar. “A new class of convex functions for delay modeling and its application to the transistor sizing problem [CMOS gates].” *IEEE Trans. on Computer-Aided Design*, **19**(7):779–788, 2000.

[KLX06] A.B. Kahng, B. Liu, and X. Xu. “Constructing Current-Based Gate Models Based on Existing Timing Library.” In *Proc. Int. Conf. Quality Electronic Design*, pp. 37–42, 2006.

[KS02] M. Ketkar and S.S. Sapatnekar. “Standby power optimization via transistor sizing and dual threshold voltage assignment.” In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pp. 375–378, 2002.

[KTK08] I. Keller, K.H. Tam, and V. Kariat. “Challenges in gate level modeling for delay and SI at 65nm and below.” In *Proc. Design Automation Conference*, pp. 468–473, 2008.

[KYT02] T. Karnik, Yibin Ye, J. Tschanz, Liqiong Wei, S. Burns, V. Govindarajulu, V. De, and S. Borkar. “Total power optimization by simultaneous dual-Vt allocation and device sizing in high performance microprocessors.” In *Proc. Design Automation Conference*, pp. 486–491, 2002.

[Lem01] C. Lemaréchal. “Lagrangian relaxation.” *Computational Combinatorial Optimization*, pp. 112–156, 2001.

[LGa] J. Lee and P. Gupta. “Impact of Range and Precision in Technology on Cell-Based Design.” *Submitted to the ACM/IEEE International Conference on Computer-Aided Design.*

[LGb] J. Lee and P. Gupta. "Incremental gate sizing for late process changes." *ECO Cost Measurement and Incremental Gate Sizing for Late Process Changes.*

[LG10] J. Lee and P. Gupta. "Incremental gate sizing for late process changes." In *Proc. Int. Conf. Computer Design*, pp. 215–221, oct. 2010.

[LG12] J. Lee and P. Gupta. "Discrete Circuit Optimization: Library Based Gate Sizing and Threshold Voltage Assignment." *Foundations and Trends® in Electronic Design Automation*, **6**(1):1–120, 2012.

[LH95] How-Rern Lin and Ting-Ting Hwang. "Power reduction by gate sizing with path-oriented slack calculation." In *Proc. Asia and South Pacific Design Automation Conference*, 1995.

[LH09] Yifang Liu and Jiang Hu. "A new algorithm for simultaneous gate sizing and threshold voltage assignment." In *Proc. Int. Conf. Physical Design*, pp. 27–34, 2009.

[LH10] Yifang Liu and Jiang Hu. "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment." *IEEE Trans. on Computer-Aided Design*, **29**(2):223–234, feb. 2010.

[LH11] C. Liao and S. Hu. "Approximation scheme for restricted discrete gate sizing targeting delay minimization." *Journal of Combinatorial Optimization*, **21**(4):497–510, 2011.

[Li93] W.N. Li. "Strongly NP-hard discrete gate sizing problems." In *Proc. Int. Conf. Computer Design*, pp. 468–471, Oct 1993.

[LLP06] X. Li, J. Le, and L.T. Pileggi. "Statistical performance modeling and optimization." *Foundations and Trends® in Electronic Design Automation*, **1**(4):331–480, 2006.

[LLP07] Xin Li, Jiayong Le, and Lawrence T. Pileggi. *Statistical Performance Modeling and Optimization.* Now Publishers Inc., Hanover, MA, USA, 2007.

[LMK90] S. Lin, M. Marek-Sadowska, and E.S. Kuh. "Delay and area optimization in standard-cell design." *Proc. Design Automation Conference*, pp. 349–352, Jun 1990.

[LT95] Jin-fuw Lee and Donald T. Tang. "An algorithm for incremental timing analysis." In *Proc. Design Automation Conference*, pp. 696–701, 1995.

[MDO05] M. Mani, A. Devgan, and M. Orshansky. "An efficient algorithm for statistical minimization of total power under timing yield constraints." In *Proc. Design Automation Conference*, pp. 309–314, 2005.

[MLG] Santiago Mok, John Lee, and Puneet Gupta. "Discrete Sizing for Leakage Power Optimization in Physical Design: A Comparative Study." *Submitted to the ACM Transactions on Design Automation of Electronic Systems.*

[MM98] R. Macys and S. McCormick. "A new algorithm for computing the effective capacitance in deep sub-micron circuits." In *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*, pp. 313–316, 1998.

[MMJ10] P. Mishra, A. Muttreja, and N.K. Jha. "FinFET Circuit Design." *Nanoelectric Circuit Design*, p. 23, 2010.

[MO04] M. Mani and M. Orshansky. "A new statistical optimization algorithm for gate sizing." In *Proc. IEEE Int. Conf. on Computer Design*, pp. 272–277, 2004.

[Mok10] Santiago Mok. *"Post-layout sizing for leakage power optimization: A comparative study."*. Master's thesis, Department of Electrical Engineering, University of California at Los Angeles, 2010.

[Mok11] Santiago Mok. "Propagation Delay Approximation considering Effective Capacitance and Slew Degradation." Technical report, UCLA, 2011. Available at http://nanocad.ee.ucla.edu/pub/Main/Publications/MSTR4_paper.pdf.

[MOS] MOSEK ApS. "The MOSEK Optimization Tools Version 5.0." Available from http://www.mosek.com.

[Naj05] Farid N. Najm. "On the need for statistical timing analysis." In *Proc. Design Automation Conference*, pp. 764–765, 2005.

[NANa] "Nangate Open Cell Library v1.2." Available from http://www.si2.org/openeda.si2.org/projects/nangatelib.

[NANb] "Nangate Open Cell Library v1.3." Available from http://www.si2.org/openeda.si2.org/projects/nangatelib.

[Nar05] S.G. Narendra. "Challenges and design choices in nanoscale CMOS." *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, **1**(1):7–49, 2005.

[NDO03] David Nguyen, Abhijit Davare, Michael Orshansky, David Chinnery, Brandon Thompson, and Kurt Keutzer. "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization." In *Proc. Int. Conf. Low Power Electronics and Design*, pp. 158–163, 2003.

[NSD86] S.R. Nassif, A.J. Strojwas, and S.W. Director. "A Methodology for Worst-Case Analysis of Integrated Circuits." *IEEE Trans. on Computer-Aided Design*, **5**(1):104–113, January 1986.

[OAA12] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo. "ISPD 2012 Discrete Gate Sizing Contest and Benchmark Suite." In *Proc. Int. Conf. Physical Design*, pp. 161–164, 2012.

[OBH11] M.M. Ozdal, S. Burns, and Jiang Hu. "Gate sizing and device technology selection algorithms for high-performance industrial designs." In *Proc. Int. Conf. Computer-Aided Design*, pp. 724–731, nov. 2011.

[OPE] Available from http://www.opencores.org.

[PAS06] C. Pacha, K. von Arnim, T. Schulz, W. Xiong, M. Gostkowski, G. Knoblinger, A. Marshall, T. Nirschl, J. Berthold, C. Russ, et al. "Circuit design issues in multi-gate FET CMOS technologies." In *Proc. IEEE Int. Solid-State Circuits Conference (ISSCC)*, pp. 1656–1665, 2006.

[PDC98] P. Pant, V.K. De, and A. Chatterjee. "Simultaneous power supply, threshold voltage, and transistor size optimization for low-power operation of CMOS circuits." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **6**(4):538–545, 1998.

[PDW89] M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers. "Matching properties of MOS transistors." *IEEE Journal of Solid-State Circuits*, **24**(5):1433–1439, 1989.

[PR90] L.T. Pillage and R.A. Rohrer. "Asymptotic waveform evaluation for timing analysis." *IEEE Trans. on Computer-Aided Design*, **9**(4):352–366, 1990.

[PRC01] P. Pant, R.K. Roy, and A. Chattejee. "Dual-threshold voltage assignment with transistor sizing for low power CMOS circuits." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **9**(2):390–394, 2001.

[PYK05] D. Patil, S. Yun, S.J. Kim, A. Cheung, M. Horowitz, and S. Boyd. "A New Method for Design of Robust Digital Circuits." In *Proc. Int. Conf. Quality Electronic Design*, pp. 676–681, 2005.

[QPP94] J. Qian, S. Pullela, and L. Pillage. "Modeling the effective capacitance for the RC interconnect of CMOS gates." *IEEE Trans. on Computer-Aided Design*, **13**(12):1526–1535, 1994.

[RC05] Sanghamitra Roy and Weijen Chen. "ConvexFit: an optimal minimum-error convex fitting and smoothing algorithm with application to gate-sizing." In *Proc. Int. Conf. Computer-Aided Design*, pp. 196–203, 2005.

[RCC07] S. Roy, W. Chen, C.P. Chen, and Y.H. Hu. "Numerically Convex Forms and Their Application in Gate Sizing." *IEEE Trans. on Computer-Aided Design*, **26**(9):1637–1647, 2007.

[RHC08] Sanghamitra Roy, Yu Hen Hu, Charlie Chung-Ping Chen, Shih-Pin Hung, Tse-Yu Chiang, and Jiuan-Guei Tseng. "An optimal algorithm for sizing sequential circuits for industrial library based designs." In *Proc. Asia and South Pacific Design Automation Conference*, pp. 148–151, 2008.

[RM07] JA Roy and IL Markov. "ECO-system: Embracing the Change in Placement." *IEEE Trans. on Computer-Aided Design*, **26**(12):2173–2185, 2007.

[RS08] D. Rautenbach and C. Szegedy. "A class of problems for which cyclic relaxation converges linearly." *Computational Optimization and Applications*, **41**(1):53–60, 2008.

[RSB04] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester. "Statistical analysis of subthreshold leakage current for VLSI circuits." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **12**(2):131–139, 2004.

[RU00] R.T. Rockafellar and S. Uryasev. "Optimization of Conditional Value-at-Risk." *Journal of Risk*, **2**(3):21–41, 2000.

[RWG77] AE Ruehli, PK Wolff, and G. Goertzel. "Analytical power/timing optimization technique for digital system." In *Proc. Design Automation Conference*, pp. 142–146, 1977.

[Sap04] S.S. Sapatnekar. *Timing*. Springer Netherlands, 2004.

[SC95] S.S. Sapatnekar and Weitong Chuang. "Power vs. delay in gate sizing: conflicting objectives?" In *Proc. Int. Conf. Computer-Aided Design*, pp. 463–466, nov 1995.

[SEO99] S. Sirichotiyakul, T. Edwards, C. Oh, J. Zuo, A. Dharchoudhury, R. Panda, and D. Blaauw. "Stand-by power minimization through simultaneous threshold voltage selection and circuit sizing." In *Proc. Design Automation Conference*, pp. 436–441, 1999.

[SEO02] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw. "Duet: An accurate leakage estimation and optimization tool for dual-Vt circuits." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **10**(2):79–90, 2002.

[SG06] V. Singhal and G. Girishankar. "Optimal Gate Size Selection for Standard Cells in a Library." In *IEEE Workshop on Design, Applications, Integration and Software*, pp. 47–50, 2006.

[SKW96] L.M. Silveira, M. Kamon, J. White, and I. Elfadel. "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits." In *Proc. Int. Conf. Computer-Aided Design*, p. 288, 1996.

[SMN05] V. Stojanovic, D. Markovic, B. Nikolic, M.A. Horowitz, and R.W. Brodersen. "Energy–delay tradeoffs in combinational logic using gate sizing and supply voltage optimization." In *Proc. European Solid-State Circuits Conference (ESSCIRC)*, pp. 211–214, 2005.

[SN90] T. Sakurai and A.R. Newton. "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas." *Solid-State Circuits, IEEE Journal of*, **25**(2):584–594, apr 1990.

[SNL05] J. Singh, V. Nookala, Z.Q. Luo, and S. Sapatnekar. "Robust gate sizing by geometric programming." In *Proc. Design Automation Conference*, pp. 315–320, 2005.

[SRV93] 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 Trans. on Computer-Aided Design*, **12**(11):1621–1634, 1993.

[SSB04a] A. Srivastava, D. Sylvester, and D. Blaauw. "Power minimization using simultaneous gate sizing, dual-Vdd and dual-Vth assignment." In *Proc. Design Automation Conference*, pp. 783–787, 2004.

[SSB04b] A. Srivastava, D. Sylvester, and D. Blaauw. "Statistical optimization of leakage power considering process variations using dual-Vth and sizing." In *Proc. Design Automation Conference*, pp. 773–778, 2004.

[SSB05] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical analysis and optimization for VLSI: timing and power.* Springer Verlag, 2005.

[SSF88] J.M. Shyu, A. Sangiovanni-Vincentelli, J.P. Fishburn, and A.E. Dunlop. "Optimization-based transistor sizing." *Solid-State Circuits, IEEE Journal of*, **23**(2):400–409, 1988.

[SSK87] B.J. Sheu, D.L. Scharfetter, P.K. Ko, and M.C. Jeng. "BSIM: Berkeley short-channel IGFET model for MOS transistors." *Solid-State Circuits, IEEE Journal of*, **22**(4):558–566, 1987.

[SSS05] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw, and V. Zolotov. "Discrete Vt assignment and gate sizing using a self-snapping continuous formulation." In *Proc. Int. Conf. Computer-Aided Design*, pp. 705–712, 2005.

[SSZ05] D. Sinha, NV Shenoy, and H. Zhou. "Statistical gate sizing for timing yield optimization." In *Proc. Int. Conf. Computer-Aided Design*, pp. 1037–1041, 2005.

[ST02] C. Sechen and H. Tennakoon. "Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step." In *Proc. Int. Conf. Computer-Aided Design*, pp. 395–402, 2002.

[SVC00] D. Stroobandt, P. Verplaetse, and J. van Campenhout. "Generating synthetic benchmark circuits for evaluating CAD tools." *IEEE Trans. on Computer-Aided Design*, **19**(9):1011 –1022, sep 2000.

[Syna] Synopsys. "Liberty Open Source Library Modeling." Web resource, http://www.opensourceliberty.org.

[Synb] Synopsys. "NanoTime." http://www.synopsys.org.

[Syn10] Synopsys. "Liberty NCX D-2009.12-SP3." http://www.synopsys.com/, 2010.

[TGW87] P. Tuohy, A. Gribben, AJ Walton, and JM Robertson. "Realistic worst-case parameters for circuit simulation." In *Proc. Communications, Speech and Vision*, volume 134, pp. 137–140, 1987.

[The08] The MathWorks. "MATLAB R2008b." `http://www.mathworks.com/`, 2008.

[TKT84] T. Tokuda, J. Korematsu, O. Tomisawa, S. Asai, I. Ohkura, and T. Enomoto. "A hierarchical standard cell approach for custom VLSI design." *IEEE Trans. on Computer-Aided Design*, **3**(3):172–177, 1984.

[TLA10] T. Taghavi, Z. Li, C. Alpert, G.J. Nam, A. Huber, and S. Ramji. "New placement prediction and mitigation techniques for local routing congestion." In *Proc. Int. Conf. Computer-Aided Design*, pp. 621–624, 2010.

[TMF94] Yutaka Tamiya, Yusuke Matsunaga, and Masahiro Fujita. "LP based cell selection with constraints of timing, area, and power consumption." In *Proc. Int. Conf. Computer-Aided Design*, ICCAD '94, pp. 378–381, 1994.

[Tri08] R. Trihy. "Addressing library creation challenges from recent liberty extensions." In *Proc. Design Automation Conference*, pp. 474–479, 2008.

[Vis03] C. Visweswariah. "Death, taxes and failing chips." In *Proc. Design Automation Conference*, pp. 343–347, 2003.

[VRK04] C. Visweswariah, K. Ravindran, K. Kalafala, SG Walker, and S. Narayan. "First-order incremental block-based statistical timing analysis." In *Proc. Design Automation Conference*, pp. 331–336, 2004.

[WAN07] V. Wang, K. Agarwal, S. Nassif, K. Nowka, and D. Markovic. "A Design Model for Random Process Variability." In *Proc. Int. Conf. Quality Electronic Design*, 2007.

[War11] J. Warnock. "Circuit design challenges at the 14nm technology node." In *Proc. Design Automation Conference*, pp. 464 –467, June 2011.

[WCR99] L. Wei, Z. Chen, K. Roy, M.C. Johnson, Y. Ye, and V.K. De. "Design and optimization of dual-threshold circuits for low-voltage low-power applications." *IEEE Trans. on Very Large Scale Integration (VLSI)*, **7**(1):16–24, 1999.

[WD09] T.H. Wu and A. Davoodi. "PaRS: parallel and near-optimal grid-based cell sizing for library-based design." *IEEE Trans. on Computer-Aided Design*, **28**(11):1666–1678, 2009.

[WDZ09] J. Wang, D. Das, and H. Zhou. "Gate sizing by Lagrangian relaxation revisited." *IEEE Trans. on Computer-Aided Design*, **28**(7):1071–1084, 2009.

[WKO06] W.S. Wang, V. Kreinovich, and M. Orshansky. "Statistical timing based on incomplete probabilistic descriptions of parameter uncertainty." In *Proc. Design Automation Conference*, pp. 161–166, 2006.

[WRK00] L. Wei, K. Roy, and C.K. Koh. "Power minimization by simultaneous dual-Vth assignment and gate-sizing." In *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pp. 413–416, 2000.

[WV98] Q. Wang and S.B.K. Vrudhula. "Static power optimization of deep submicron CMOS circuits for dual VT technology." In *Proc. Int. Conf. Computer-Aided Design*, pp. 490–496, 1998.

[XM01] T. Xiao and M. Marek-Sadowska. "Gate sizing to eliminate crosstalk induced timing violation." In *Proc. Int. Conf. Computer Design*, p. 0186, 2001.

[YCL10] C.C. Yeh, C.S. Chang, H.N. Lin, W.H. Tseng, L.S. Lai, T.H. Perng, T.L. Lee, C.Y. Chang, L.G. Yao, C.C. Chen, et al. "A low operating power FinFET transistor module featuring scaled gate stack and strain engineering for 32/28nm SoC technology." In *IEEE International Electron Devices Meeting (IEDM)*, pp. 34–1, 2010.

[YZL07] Xiaoji Ye, Yaping Zhan, and Peng Li. "Statistical leakage power minimization using fast equi-slack shell based optimization." In *Proc. Design Automation Conference*, pp. 853–858, 2007.

[ZBS08] Cheng Zhuo, David Blaauw, and Dennis Sylvester. "Variation-aware gate sizing and clustering for post-silicon optimized circuits." In *Proc. Int. Conf. Low Power Electronics and Design*, ISLPED '08, pp. 105–110, 2008.

[ZM06] Q. Zhou and K. Mohanram. "Gate sizing to radiation harden combinational logic." *IEEE Trans. on Computer-Aided Design*, **25**(1):155–166, 2006.

[ZSL05] Y. Zhan, A.J. Strojwas, X. Li, L.T. Pileggi, D. Newmark, and M. Sharma. "Correlation-aware statistical timing analysis with non-gaussian delay distributions." In *Proc. Design Automation Conference*, pp. 77–82, 2005.