Foundations and Trends® in
Electronic Design Automation
Vol. 6, No. 1 (2012) 1–120
© 2012 J. Lee and P. Gupta
DOI: 10.1561/1000000019

Discrete Circuit Optimization: Library Based Gate Sizing and Threshold Voltage Assignment

By John Lee and Puneet Gupta

Contents

Foundations and Trends® in
Electronic Design Automation
Vol. 6, No. 1 (2012) 1–120
© 2012 J. Lee and P. Gupta
DOI: 10.1561/1000000019

Discrete Circuit Optimization: Library Based Gate Sizing and Threshold Voltage Assignment

John Lee[1] **and Puneet Gupta**[2]

[1] *UCLA, Los Angeles, CA 90095, USA, lee@ee.ucla.edu*
[2] *UCLA, Los Angeles, CA 90095, USA, puneet@ee.ucla.edu*

Abstract

Discrete gate sizing and threshold assignment are commonly used tools for optimizing digital circuits, and ideal methods for incremental optimization. The gate widths and threshold voltages, along with the gate lengths, can be adjusted to optimize power and delay. This monograph surveys this field, providing the background needed to perform research in the field. Concepts such as standard cell libraries, static timing analysis, and analytical delay and power models are explained, along with examples and data to help understand the tradeoffs involved. Comparative results are also provided to show the current state of the field.

1

Introduction

Gate sizing and threshold voltage assignment[1] are widely used to optimize digital circuits. They can be used to manage trade-offs in power, timing, area, yield, crosstalk, statistical power, statistical delay and soft-errors. They can also be used incrementally and as a method for optimizing post-layout designs after placement and interconnect routing. After over three decades, research is still active in the area.

This work will consider the case of gate sizing and threshold voltage assignment for standard library cell designs. In this context, gates are chosen from a library of pre-characterized gates that act as the fundamental building blocks. Cell-based designs compose the majority of the digital designs today.

1.1 Other Types of Cell Optimization Problems

There are other variants of gate sizing and threshold voltage assignment that are not covered in this work:

(1) Transistor sizing for analog design
(2) Transistor sizing for custom digital design

[1] While the title explicitly states the gate sizing and V_t assignment, the material is relevant to other cell optimization methods, such as gate-length biasing problems.

These variants are a minority of IC designs. Custom digital design is mainly limited to high-performance designs. However, analog designs are becoming increasingly important with the increase in systems on a chip (SoC) methodologies that integrate entire TVs or radios on a single die [63].

1.1.1 Transistor Sizing for Analog Designs

In analog design, there are many different constraints and performance specifications: gain, accuracy, linearity, signal-to-noise, and impedance matching. For example, the pair of transistors forming a current mirror or differential pair must be *matched*, or have very similar electrical characteristics. Also, transistors that function as voltage-controlled resistors must be operating in the linear region, and an amplifier must have the proper signal-to-noise ratio for the system to work properly. While standard cells mask much of the underlying electrical waveforms using logic states, analog designs utilize these underlying characteristics to produce amplifiers, digital-to-analog converters, current sources, etc. However, many more facets of the design must be controlled for a proper function.

The main challenge in automated analog sizing is to input the design specifications and models into a form that can be used by the sizing method. This is challenging because the range of analog designs is large. For instance, while the analytical performance models for a given op-amp topology might be well known, it is difficult to write down the equations that govern the sizing of an *arbitrary* design.

Early methods for automated sizing were *knowledge-based*, where templates [52, 55, 72] were used to synthesize designs. These pre-characterized templates would carry information on a good initial sizing and on how to optimize the given template. Sizing these designs was therefore equivalent to executing the design plans. For example, in [52], the sizing proceeds by determining the bias current, then the W/L ratios, followed by the $1/f$ noise consideration, and finally the W and L of each device. In [72], the values are chosen using a fixed point method, where the parameters are determined serially. Each parameter is chosen to best satisfy the design considerations, assuming the other parameters to be fixed.

The time required to construct these templates, however, was often much greater than the time needed to design the circuit directly [63]. The accumulation of knowledge bases, and the codification of the expert knowledge was not practical. This, coupled with the limited range of circuits that the method could handle, led to the decline of these types of methods. However, there is recent interest in automating the knowledge-extraction process [108], and in identifying substructures in a design automatically [107].

Another branch of analog sizing is the *optimization-based* methods. These methods use optimization procedures, rather than codified design rules, to size the design. The first subclass of optimization methods consists of *equation based methods*[2] [64, 75, 87] that rely on the designer to provide the equations, but in contrast to the knowledge-based methods, the sizing process is automated using optimization methods, rather than rules. The optimization process may use simulated annealing [64], steepest-descent [87], or convex optimization [75]. This sub-class works well in certain contexts (such as in [75]), but they may be limited by their accuracy.

The second subclass of optimization based methods are the *simulation based methods* [54, 58, 88, 118, 119, 126] that use numerical simulations to measure the performance. The simulations provide these methods with greater accuracy, however they create a large overhead that makes these methods much slower than their equation-based counterparts.

1.1.2 Transistor Sizing for Custom Digital Designs

In the custom digital setting, every transistor in the design is available for optimization [7, 44, 45, 82, 153]. The early papers on gate sizing were based on transistor-level sizing (see [60, 135]), until standard cells became widespread in the 1990s.[3] With the increasing complexity of designs, standard cell libraries are almost universally used.

Custom digital design techniques are primarily used for high-performance parts of high-volume designs which is needed to recover

[2] See [63] for the taxonomy of analog sizing methods and a comprehensive review of methods prior to 2000.
[3] See, for example, [159].

the increased cost of designing at a transistor level, as in the case of microprocessors [12]. Custom digital transistor sizing is still an active area of research today and most current research is directed toward the statistical design of custom circuits [8, 43, 146]. However, these methods account for a minority of the digital designs.

1.2 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, standard cell library information, timing constraint information.
   - Output: Netlist mapped to the standard cell 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, standard cell 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.

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 [22]). After clock tree synthesis, they are used to further fix

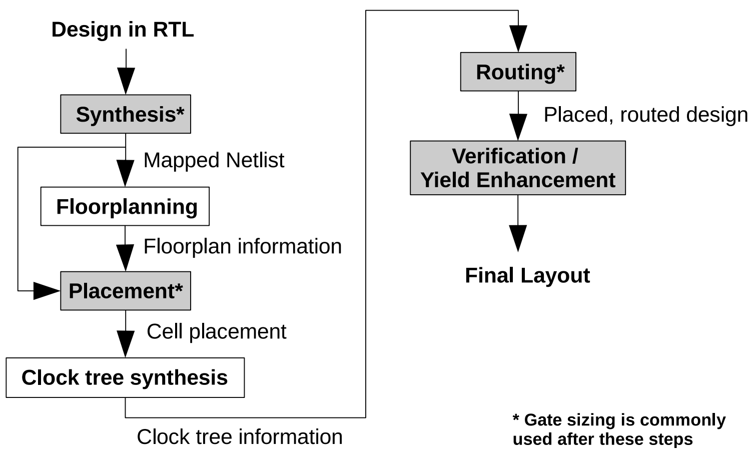


Fig. 1.1 Electronic Design Automation flow.

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 [2]. 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 the 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 [46], to fix noise constraints due to crosstalk [98, 170], to harden soft-errors due to radiation [174], to improve yield [37, 50], and to minimize statistical power [42, 151].

1.3 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, 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. This will be covered in more detail in Section 2.4.

1.4 The Gate Sizing and V_t Assignment Problem

Of the many variations on the gate sizing and threshold voltage assignment problem, this work will consider the following metrics:

- Leakage power
- Dynamic power
- Clock period

The application most commonly found in literature today is to minimize the power, or some combination of the leakage power and the dynamic power, with a constraint on the clock period. When timing closure is important, the objective is to minimize the clock period, and in post-layout situations, the noise and crosstalk violations are often optimized.

The variables in the optimization process are the cells used to implement the gates. These cells can be swapped to decrease the delay or power, or to modify the output signal waveform. The cells may be different sizes, and have different pin locations.

Functionally equivalent cells that can be interchanged may be identified by the designer, the design tool, or by the library by having the same *footprint*. For example, the inverter-type cells will have an "inverter" footprint, which identifies the family of cells that can be used to replace the gate. Each of these cells performs the same logic function, thus changing the cells does not affect the functionality of the design.

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

$$\begin{aligned} &\text{minimize} && \text{Power}(\vec{\omega}) \\ &\text{subject to} && \text{Delay}(\vec{\omega}) \leq T_{\max} \end{aligned} \tag{1.1}$$

where $T_{\max}$ is the clock period. This form helps summarize the objectives and constraints in the optimization process.

Table 1.1. Notation.

Symbol	Meaning
$\mathcal{G}$	Set of gates in the design
g	A gate in the design
ω, ω_0	A cell option, current cell option
CellOptions(g)	Set of alternative library cell options for g
w_g	The width for gate g
$v_{\mathrm{th}\,g}$	The threshold voltage for gate g
t_a	Arrival time
$t_{a(g)}$	Set of arrival times for the inputs of gate g
t_r	Required arrival time
$t_{r(g)}$	Set of required arrival times for the inputs of gate g
τ_g	Set of input transition slews for gate g
d	Delay
p	Power
PI	*Primary inputs* input ports in the design
PO	*Output ports* in the design
fi(g)	The set of gates that drive the inputs of gate g
fo(g)	The set of gates that are connected to the output of gate g
ϵ	A small positive number
ρ	Power/delay tradeoff sensitivity

The delay is estimated using a Static Timing Analysis method (see Section 2) and timing constraints are usually expressed in a Synopsys Design Constraint (SDC) format. These constraints can be very complex, with multicycle paths, multiple clocks, power domains, and false-path definitions. In addition, parasitic information in a Standard Parasitic Exchange (SPEF) format is used to approximate the interconnect delay.

1.5 Notations and Acronyms

While new notation is avoided when possible, notation for certain key concepts are unavoidable. The notation for key symbols used in this monograph are summarized in Table 1.1. These symbols, and concepts, will be elaborated in the remainder of the monograph; for example, the slew and other timing concepts will be covered in the following section.

2
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 [137] for a comprehensive treatment of the subject, and [14] 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 as 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 fast at the input of a flip-flop or latch, causing the input state of the flip-flop to be set improperly, potentially causing metastability at the output.

2.1 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 are 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.

For example, in Figure 2.2, the primary inputs are {`IN[0]`, `IN[1]`}. The primary outputs are {`OUT[0]`}. The fanin of gate G2 is {G1} and the fanout of G3 is {G2}. In this example, a topological ordering is {G1,G3,G2}.

The second set of terms are 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%.
- **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 are 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 45 nm [114] process. The input is assumed to be an ideal ramp, however, the output transition

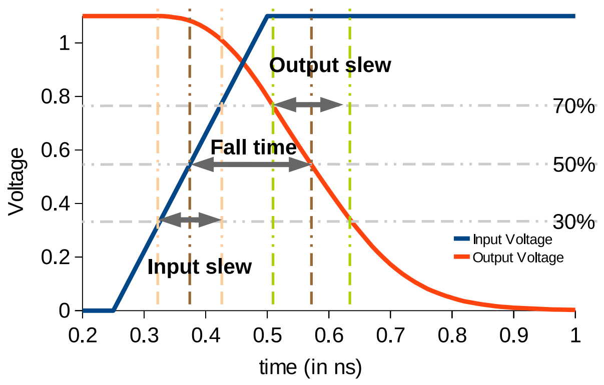


Fig. 2.1 Plot of the transition and delay times for a size 1 inverter driving a 0.05 pF load.

is not ideal. The input slew is 0.1 ns, the fall time is 0.2 ns, and the output slew is 0.12 ns.

The 50% threshold for the delays, and the 30–70% thresholds for the slews 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 20–80%, or a 30–70% 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 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 or be read from the primary output.

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 slacks 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 are 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. In contrast, the buffer cell has only *positive unate* timing arcs — a rising input will always lead to a rising output. In the exclusive-OR gate, there are both positive unate and negative unate arcs, as a rising input can lead to a rising output (if both inputs are initially 0), or a falling output (if the other input is 1).

2.2 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.

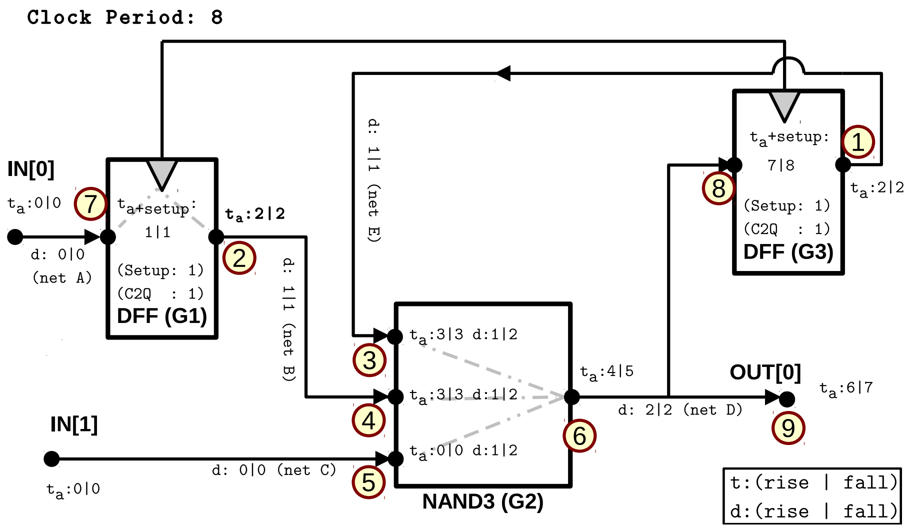


Fig. 2.2 Static timing analysis example–finding the maximum path. The notation t_a: [rise] | [fall] is used to denote the rise arrival times and the fall arrival times, and d: [rise]|[fall] is used to denote the rise and fall delays.

Figure 2.2 gives a simplified example of the max path STA process. The STA computes the maximum path delay by computing the arrival times at:

(1) The output of G3 by using the C2Q delay.
(2) The output of G1 by using the C2Q delay.
(3) G2 from G3 by adding the output delay with the interconnect delay of net E.
(4) G2 from G1 by adding the output delay with the interconnect delay of net B.
(5) G2 from `IN[1]` by adding the arrival time at `IN[1]` to the interconnect delay of net C.
(6) The output of G2 by taking the *maximum* of the sum of the input arrival times and the gate delay.
(7) The input of G1 by adding the arrival time at `IN[0]` to the interconnect delay of net A.
(8) The input of G3 by adding the output arrival time at G2 with the interconnect delay of net D.

(9) The primary output `OUT[0]` by adding the output arrival time at G2 with the interconnect delay of net D.

In this example, the worst-case delay is a falling delay of 8 at flip flop G3. There are two equally critical paths, or the worst-delay paths with this delay. One is from G3 to G2 to G3, and another is from G1 to G2 to G3.

Finding the *shortest path* to check for hold-time violations is similar, except that the minimum arrival times are propagated. The difference in Figure 2.2 is that the minimum path, from the primary input `IN[1]` is propagated through gate G2. In this case, the hold requirement would be met: the minimum delay is 2, while the hold requirement is also 2.

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:

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

Figure 2.3 shows an example of this process. It starts at the flip-flop inputs (G3 and G1) and primary output (`OUT[0]`), and proceeds reverse-topologically.

The examples above shows how to compute the delay, the required arrival times and the slacks as a function of the gate delays and interconnect delays. The process of computing the gate and interconnect delays will be explained below.

2.3 Interconnect Delay

As scaling continues, interconnect delay becomes an increasingly large percentage of the total delay. This has increased the importance of

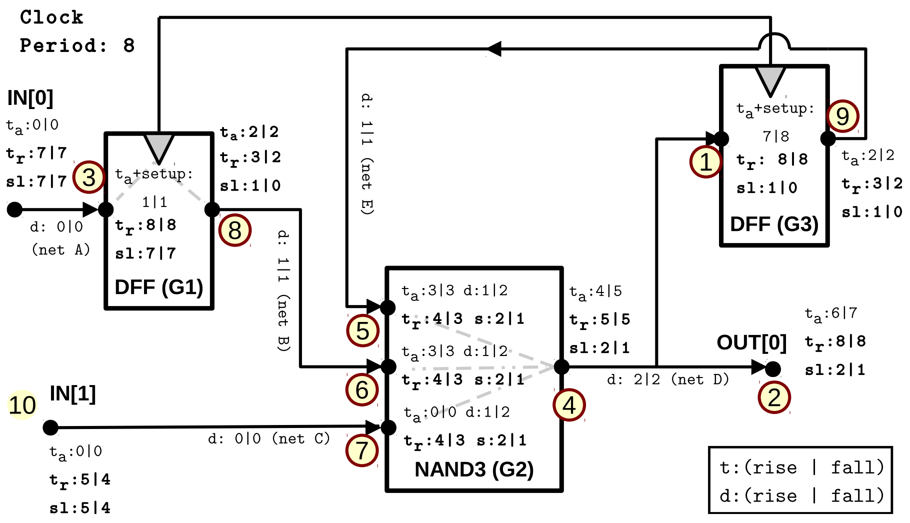


Fig. 2.3 Static timing analysis example to compute the slack. The required arrival times (t_r) are propagated from the outputs to the inputs, and the slack (s) is computed as $s = t_r - t_a$. The notation t_r: [rise] | [fall] is used to denote the rise required arrival times and the fall required arrival times.

computing accurate wire delays, and has motivated fast and accurate interconnect delay metrics [23].[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*. This is defined as a tree with:

- one voltage source,
- capacitances are connected to ground,
- resistances to other nodes (but never to ground).

There are more complex variations on the RC tree [137]. 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

[1] For a detailed analysis of the subject, please see [23] or [137].

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 [56]. 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 the actual delay [16] (also see [1]) and can be used for rough delay estimates or for short interconnect. In [68], 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 [127] and the Krylov subspace methods [59, 144]. We refer the reader to [23, 137] for more details on these methods.

2.4 Gate Delay Models

The most accurate gate 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.

[2] For example, the Synopsys NanoTime [157].

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 format [156].[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.

2.4.1 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 20–80% or 30–70% 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 2.1 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

[3] See http://www.opensourceliberty.org for information on the format, technical papers, and an open source parser. Also, see [160] for the motivation for current-source methods.

Table 2.1. 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.

	Output capacitance (fF)						
Input slew	0.4	0.8	1.6	3.2	6.4	12.8	25.6
Output rise delay							
0.0075	0.018	0.022	0.032	0.051	0.089	0.164	0.315
0.0375	0.032	0.038	0.048	0.066	0.104	0.179	0.330
0.1500	0.059	0.069	0.087	0.117	0.164	0.239	0.388
0.6000	0.129	0.145	0.174	0.224	0.305	0.433	0.628
Output rise transition							
0.0075	0.012	0.016	0.025	0.043	0.079	0.151	0.294
0.0375	0.018	0.021	0.027	0.043	0.079	0.151	0.294
0.1500	0.036	0.042	0.052	0.067	0.092	0.152	0.294
0.6000	0.095	0.100	0.114	0.139	0.183	0.252	0.353

Table 2.2. Setup times for a rising input to a nominal D-Flip Flop as a function of the input transition and the clock transition.

	Clock transition (ns)						
Input slew	0.005	0.0141	0.0281	0.056	0.113	0.225	0.450
Setup time							
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							
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

fall transitions, as a function of the input transition (slew), and the clock slew. Table 2.2 shows a small example of the setup time for a nominal sized D-flip flop.

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

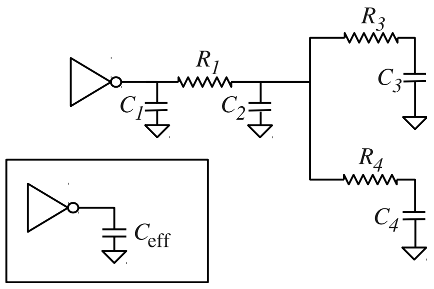


Fig. 2.4 Example of effective capacitance: an inverter driving an RC tree. The effective capacitance is less than the total capacitance of the tree: $C_{\text{eff}} < \sum_{i=1}^{4} C_i$. Adapted from [137].

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 [14, 57, 102, 128]) to predict accurate delays. Figure 2.4 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 [57]. 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, [81]). 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.

2.4.2 Current Source Models

Current source models were introduced to increase the accuracy of the gate models [57, 84, 160]. 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 [160].

The receiver models are used to capture the nonlinear characteristics of the receiver [84, 160]. 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 0.43 fF to 0.78 fF.

2.4.3 Liberty 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 2.3 shows the leakage information for the NAND2_X1 gate from the 45 nm 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.

Table 2.3. Leakage powers for the NAND2_X1 gate. A1 and A2 are the two input pins.

When	Value
!A1 & !A2	1017.82
!A1 & A2	7340.61
A1 & !A2	1204.57
A1 & A2	10286.09

2.5 Slew Propagation

An important part of static timing analysis is the propagation of the input transitions, or slews [14]. The output waveforms play a major role in determining the output delay. For example, Table 2.1 shows that the difference in delay between the minimum and maximum transition is 2–7×. 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 2.5 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. This is because the timing analysis is

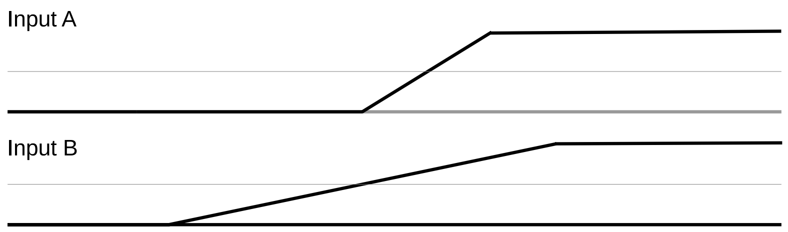


Fig. 2.5 Output transition (slew) dilemma. The input waveforms to a two-input gate are shown.

intended for design validation, and because it is important to use a worst-case approach. Thus, in Figure 2.5, 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 2.5, 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.

2.6 Power and Clock Domain Considerations

Most modern designs will also use multiple clock domains, and possibly multiple power domains [14]. Multiple clock domains arise naturally, as one clock might be used to synchronize the input of the design, another clock for the processing within the design and a third clock to synchronizes the output of the design. Multiple power domains may show up due to performance considerations. A low-voltage power domain may be used to handle data paths that are not-critical, while a higher-voltage might be used for parts of the domain that need a faster speed.

Multiple power domains require the computation of the appropriate delays for the proper supply voltage, and the delays through the voltage-level shifters. This affects the delays and slews, and the method to compute the slews. For example, the delay would need to be computed with a higher voltage, and also the arrival time calculated using the 50% of the *new* voltage.

Multiple clock domains are a trickier consideration. While timing the design is straightforward within a clock domain, complexities emerge when *crossing* clock domains. Generally, the data signals use synchronization or handshaking methods between the different clock domains. However these paths require special attention to make sure that these paths are handled correctly, or designers mark them to be ignored.

2.7 Additional Considerations

There are several additional concepts that are used in Static Timing Analysis [14] 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 [14].

2.8 Difference 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 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 5.2.1.

2.9 Incremental Timing Analysis

Gate sizing and other optimization methods rely heavily on incremental timing analysis methods [90, 137]. 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 3.9(a). 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.

2.10 Statistical Static Timing Analysis (SSTA)

Statistical Static Timing Analysis algorithms update the STA framework to compute statistical delays and arrival times. Instead of numbers for arrival times and slacks, *distributions* are used. There is an extensive literature on the subject[4] and in this section, we briefly describe the basic categories and describe the two main categories of SSTA methods: block-based methods and path-based methods.

2.10.1 Block-based Methods

Block-based methods for SSTA [25, 86, 161, 165] are a natural extension of the STA framework. In STA, there are two main mathematical operations: the summation, when gate delays are added to arrival times, and the maximum operation, when the latest arrival time is propagate down the circuit (see Section 2.2). SSTA extends these operations to work with *distributions*, thereby making it possible to propagate *statistical* arrival times.

[4] For a good overview see [15, 61, 94].

Gate delays and arrival times can be described using the canonical delay model [165]:

$$a_0 + \sum_{i=1}^{n} a_i \Delta \mathbf{X}_i + a_{n+1} \Delta \mathbf{R}_a \tag{2.3}$$

where $\Delta \mathbf{R}_a$ and the $\Delta \mathbf{X}_i$ are zero mean, unit variance random variables. Each of the $\Delta \mathbf{X}_i$ describe the n different global sources of variation, with the a_i representing the sensitivity of the delay to the variation source.[5] $\Delta \mathbf{R}_a$ describes the *independent* local variation, and the a_{n+1} describes the sensitivity to the local variation.

When the timing variations are Gaussian, the sum is straightforward to compute. As the sum of two Gaussian random variables is also Gaussian, the sum of d_a and d_b given by

$$d_a = a_0 + \sum_{i=1}^{n} a_i \Delta \mathbf{X}_i + a_{n+1} \Delta \mathbf{R}_a \tag{2.4}$$

$$d_b = b_0 + \sum_{i=1}^{n} b_i \Delta \mathbf{X}_i + b_{n+1} \Delta \mathbf{R}_b \tag{2.5}$$

can be summed directly to form the Gaussian random variable

$$d_{a+b} = (a_0 + b_0) + \sum_{i=1}^{n} (a_i + b_i) \Delta \mathbf{X}_i + \sqrt{a_{n+1}^2 + b_{n+1}^2}\, \Delta \mathbf{R}_{a+b}. \tag{2.6}$$

The maximum operation, however, is problematic, and is the main difficulty for block-based methods. The maximum of two Gaussian random variables is not, in general, Gaussian, and the resulting distribution cannot be computed in closed form. However, the mean and the variance of the resulting distribution can be computed using the formulas in [41], which [25] uses to reconstruct a Gaussian distribution that approximates the maximum. Applied to the canonical delay model, the maximum of d_a and d_b is approximated as [165] as

$$\begin{aligned} \hat{d}_{\max\{a,b\}} &= (T_A a_0 + (1 - T_A) b_0) + \sum_{i=1}^{n} (T_A a_i + (1 - T_A) b_i) \Delta \mathbf{X}_i \\ &\quad + c_{n+1} \Delta \mathbf{R}_{\max\{a,b\}} \end{aligned} \tag{2.7}$$

[5] Sensitivity can be computed as the partial derivative of the delay with respect to that variation source.

where T_A is the probability that $d_a > d_b$, and c_{n+1} is computed so that the variance of the approximation $\hat{d}_{\max\{a,b\}}$ matches the actual variance.

Extensions to this approximation to handle non-Gaussian variations and delay expressions are studied in [26, 32, 38, 85, 147, 171, 172, 173]. The non-Gaussian methods improve the accuracy of block-based SSTA by incorporating higher-order effects, or accounting for the skewed delay distributions.

2.10.2 Path-based Methods

Path-based methods are studied in [3, 5, 19, 35, 62, 97, 99, 103, 120, 129]. These methods select the most critical paths in the design, and compute the statistical delay pdf using these critical paths, either by Monte-Carlo, or by using analytical method. They have the advantage of being more accurate because they do not need to approximate the statistical maximum of two random variables [129]. On the other hand, these methods are much more computationally intensive than the block-based methods, as the number of paths is generally believed to be exponential in the number of cells. However, [129] argues that the number of paths is sub-quadratic for a commonly used set of benchmarks, and that path-based SSTA can be used in conjunction with block based methods to improve accuracy.

3

Gate Sizing and V_t Assignment Fundamentals

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. Larger gates are used to drive 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 trade-off is the central mechanism behind gate sizing and threshold voltage assignment.

3.1 Delay Trade-offs

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* [136]:

$$\text{Delay} \propto \frac{1}{\frac{W}{L}(V_\mathrm{gs} - V_\mathrm{t})^{\alpha}}. \tag{3.1}$$

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.

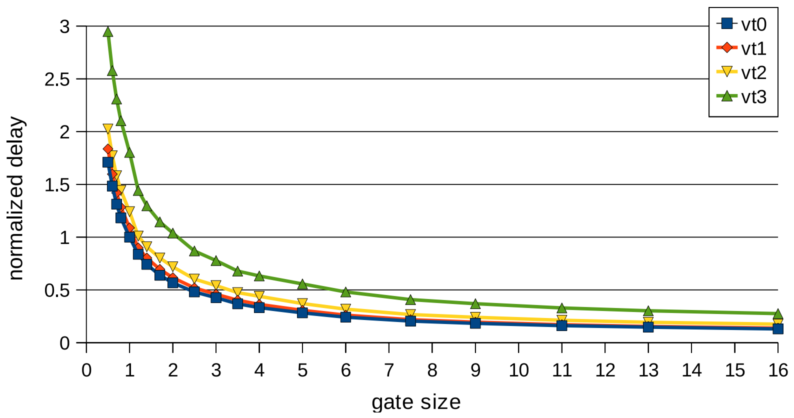


Fig. 3.1 Normalized inverter gate delays for a commercial 45 nm library.

For a 45 nm commercial library, a sample of the effects of the sizes and threshold voltages on the delays is given in Figure 3.1.[1] α for this library is approximately 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 10–15% reduction in the output delay, and the difference between the delay of the minimum size and maximum size is 12×. In the case of threshold voltages, changing from v_{t0} to v_{t1} or v_{t1} to v_{t2} increases the delay by about 10%. However, increasing the threshold voltage from v_{t2} to v_{t3} 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

[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%.

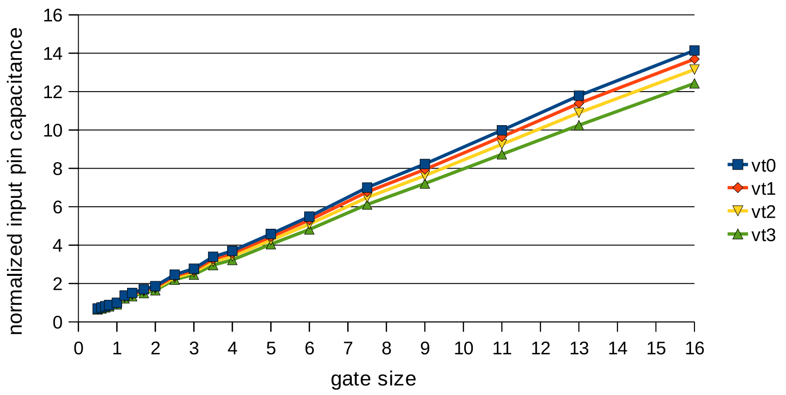


Fig. 3.2 Normalized inverter *input* pin capacitances in a commercial 45 nm library.

size will therefore cause the delay of the fanin gates to increase, and may have a cumulative negative effect on the delay. This is why gate sizing requires a *balancing* of the loads in the design (see Section 3.1.1), rather than increasing the sizes to meet the delay constraints.

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 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.

3.1.1 Delay Optimization and Balancing Loads

From a delay standpoint, and ignoring constraints on power and area, the goal of gate sizing is to balance the capacitive loads with higher gate sizes. To illustrate this concept, consider an arbitrary gate that is driven by a resistor with resistance R_{in} and that charges a capacitance C_L as shown in Figure 3.3. Using a linear delay model, the resistance of the gate R_{eff} is R_{eff0}/w and the input capacitance C_{in} is $C_{\mathrm{in0}} \cdot w$. The time needed for the signal to reach the output is the sum of the times needed to charge C_{in} and C_L. Using the relations $V = IR$ and

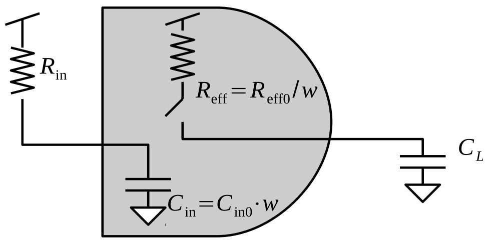

R_{in}
$R_{\text{eff}} = R_{\text{eff0}}/w$
C_L
$C_{\text{in}} = C_{\text{in0}} \cdot w$

Fig. 3.3 Example of a gate driven by a resistance R_{in} that drives a capacitance C_L.

$I = C \cdot \frac{\mathrm{d}V}{\mathrm{d}t}$ gives:

$$\ln(2) \cdot R_{\text{in}} \cdot (C_{\text{in0}} \cdot w) + \ln(2) \cdot (R_{\text{eff0}}/w) \cdot C_L, \tag{3.2}$$

where the ln(2) is used to account for the 50% delay threshold. Note that this time is not necessarily a decreasing function of the gate size w. While the time needed to charge C_L decreases as w increases, the time needed to charge C_{in} has the opposite effect. This implies that the gate size needs to be balanced between the input and the output. The minimum can be found by differentiating with respect to w:

$$\frac{\partial}{\partial w} \ln(2) \cdot (R_{\text{in}} \cdot C_{\text{in0}} \cdot w + C_L \cdot R_{\text{eff0}}/w) \tag{3.3}$$

$$= \ln(2) \cdot (R_{\text{in}} \cdot C_{\text{in0}} - C_L \cdot R_{\text{eff0}}/w^2). \tag{3.4}$$

Setting this equal to zero gives:

$$w = \frac{C_L \cdot R_{\text{eff0}}}{R_{\text{in}} \cdot C_{\text{in0}}}. \tag{3.5}$$

This gives an optimal size for minimum delay that *balances* the load between the input of the gate, and the output load.

These relations can be used for multi-gate and general circuits as well. The idea is that to achieve minimum delay, the gate sizes should balance the drive strength of the fanin gate, R_{in}, and the output load C_L to minimize the delay. Given the input and output loading information, the expression in (3.5) can be used to determine the best size.

Logical effort [154, 155] provides a back-of-the envelope way to perform this optimization using the linear delay model. This method is a rule-based and practical way to apply the relations in (3.5) for arbitrary circuits. The methods in logical effort are often used to perform quick and approximate computations to size circuits.

3.2 Power Trade-offs

Power is an important concern in gate sizing and threshold voltage assignment [42, 79]. Power dissipation has increased super-linearly with the 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 45 nm 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 [115].

3.2.1 Dynamic Power

Dynamic power is related to the energy used to switch logic states. This includes the power used for charging and discharging the interconnect and also includes the internal energy that is lost in the process of switching.

The dynamic power due to charging and discharging gates ($p_{\text{dynamic(cap)}}$) is directly related to the capacitance that is charged. Thus, if the total capacitance of a gate is C_g, and the gate switches at a frequency of f_{switch}, then the power dissipation is approximately:

$$p_{\text{dynamic(cap)}} = f_{\text{switch}} \cdot \left(\frac{1}{2} C_g V_{\text{dd}}^2\right). \tag{3.6}$$

This expression is just the energy stored on the gate, $\frac{1}{2} C_g V_{\text{dd}}^2$, times the number of times the energy will be dissipated per second (f_{switch}).

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

$$p_{\text{dynamic(cap)}} \propto W \cdot L. \tag{3.7}$$

Thus, 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 p-MOS and n-MOS are temporarily on 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 5–10× increase in the short-circuit power, and using minimum sized devices may hurt overall power [138]. From [136], a short-circuit current expression based on the α-power law is:

$$p_{\text{dynamic(sc)}} \propto \tau \cdot V_{\text{DD}} \frac{W}{L} \cdot \frac{(1 - 2(\frac{V_{\text{t}}}{V_{\text{DD}}}))^{\alpha+1}}{(1 - (\frac{V_{\text{t}}}{V_{\text{DD}}})^{\alpha}}, \tag{3.8}$$

where τ is the input transition time, and α is a fitting coefficient for the technology library.[2] 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.[3]

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

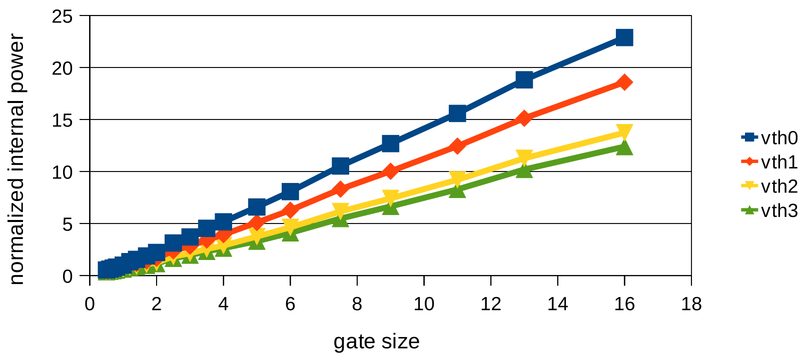


Fig. 3.4 Normalized inverter gate *internal* powers a commercial 45 nm library.

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

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

the switching power is indeed a linear function of the size. When comparing threshold voltages, the $v_{\rm t0}$ gate has the highest internal power dissipation. Switching from $v_{\rm t0}$ to $v_{\rm t1}$ or from $v_{\rm t1}$ to $v_{\rm t2}$ gives an internal power savings of about 20%. The reduction from $v_{\rm t2}$ to $v_{\rm t3}$ is smaller, at 10%.

3.2.2 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 dynamic power. Leakage became a significant source of power dissipation from the 130 nm technology node and at 45 nm 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 has a current proportional to [142]:

$$\frac{W}{L} \cdot \mathrm{e}^{\frac{V_{\rm gs}-V_{\rm t}}{v_T}} \left(1 - \mathrm{e}^{\frac{-V_{\rm ds}}{v_T}}\right), \tag{3.9}$$

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

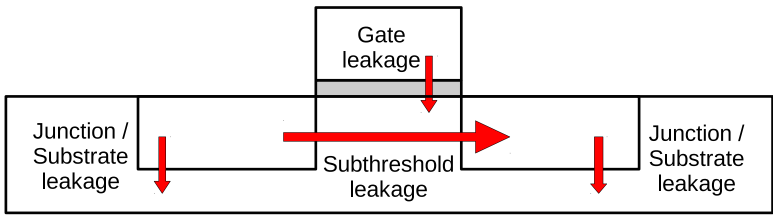


Fig. 3.5 Sources of leakage power.

Table 3.1. Leakage power for a NAND3 cell as a function of its inputs.

State	Leakage (nW)
A1 & A2 & A3	19.78
A1 & A2 & !A3	14.72
A1 & !A2 & A3	7.41
A1 & !A2 & !A3	0.48
!A1 & A2 & A3	15.59
!A1 & A2 & !A3	1.63
!A1 & !A2 & A3	7.15
!A1 & !A2 & !A3	1.27

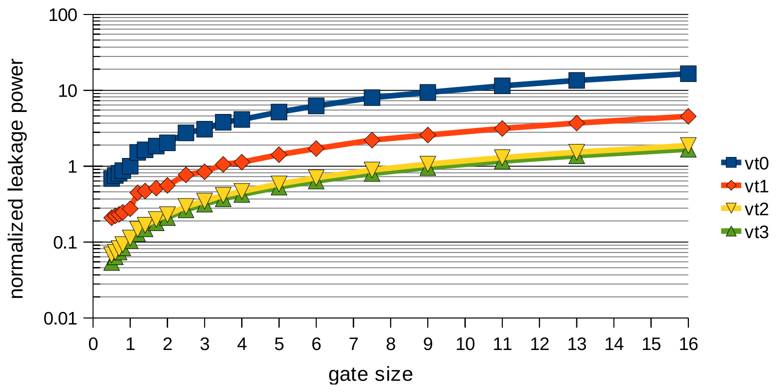


Fig. 3.6 Normalized inverter gate *leakage* powers in a commercial 45 nm library.

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, Table 3.1 shows the leakages for the nominal sized NAND3 cell as a function of the input. The leakage values range from 0.48 nW to 19.78 nW — a range of over 40×. 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

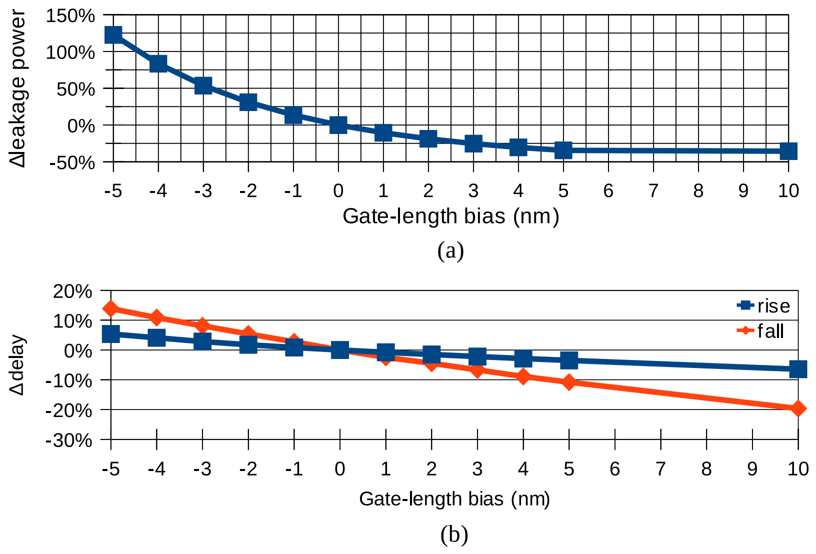


Fig. 3.7 Plot of (a) the gate-length bias vs. power and (b) gate-length bias vs. delay. The data is for a nominal sized inverter in a commercial 45 nm process, with a nominal gate length of 40 nm.

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 [67] 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 leakage power while creating a 3–11% increase in the delay. Adjusting the gate-lengths also provides a finer resolution than the V_t-assignment.

3.3 Gate Sizing Examples

In this section, we provide examples that illustrate the trade-offs involved in gate sizing. The first and second examples show how the delay target affects the power, while the third example shows the sizes needed for minimum delay. These examples are adapted from the sizing benchmarks in [65].

3.3.1 Inverter Chain Example

The first example is an inverter chain driving a fanout load equivalent to 32 minimum-sized inverters. Figure 3.8 shows the gate sizes, dynamic power, and leakage power as a function of the circuit delay. The power vs. size shows an intuitive trend — the gate sizes increase as the power constraint decreases. This is because there is a large load at the end of the inverter chain, and larger gates are useful for driving this load.

The gate sizes increase beginning with the gates closest to the load at the end. This points out an important fact about gate sizing — larger gates do not always help. For example, with all other gates at minimum size, increasing the size of gate g2 to 32× will *increase* the delay by 50%! It is important to remember that the goal in minimizing the delay is to balance the capacitive loads in the design (see Section 3.1.1). Thus, the size of gate g4 is increased first. Next, g3 is increased to account for the increased size of g4, and thereafter, g2 and g1 are increased.

The switching power (Figure 3.8(c)) and leakage power (Figure 3.8(d)) also follow this trend. The switching power increases because the cumulative sizes of the gates, and hence, the total capacitance increases. The trend for the switching power follows (3.6), and the leakage power trend follows (3.9). Thus the trend in these plots is very similar to the trend in Figure 3.8(b).

The internal power, however, is not a monotonic function of the cumulative gate sizes. As the internal power is a function of the internal node capacitances *and* the short-circuit current, it is affected by the input transition times. Initially, as the gate sizes increase, the input transition times decrease, causing the total internal power to increase. However, as the gate sizes continue to increase, the internal capacitances dominate the internal power, and the power follows a similar trend as the switching power.

In these figures, the scales on the dynamic power (Figure 3.8(c)), and the scale on the leakage power (Figure 3.8(d)) are different. This might suggest that the dynamic power consumption will be 1000×

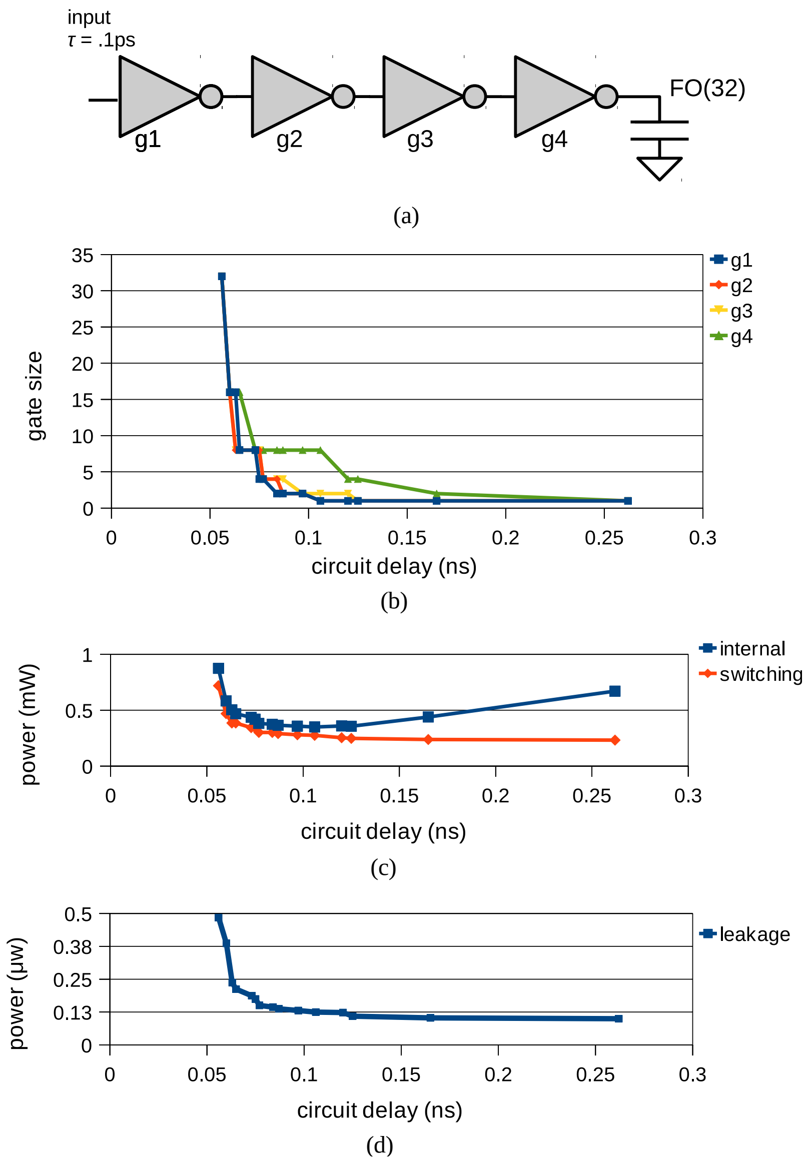


Fig. 3.8 Inverter chain driving a fanout of 32 (a capacitance equal to 32 minimum sized inverters). The schematic is shown in (a). Plots (b)–(d) show the gate sizes, dynamic power, and leakage power as a function of the circuit delay.

larger than the leakage power. However, this is misleading in this example. The switching power is a function of the switching frequency (see (3.6)), and in large designs, the likelihood that a node will switch is much less than 1.

3.3.2 Mesh Circuit Example

The mesh circuit (see [65]) is an example where there are a few gates with large loads (g2, g3 and g5), and the rest have a fanout of just one gate. To create a minimum delay configuration, the gate with the most fanout load, g5, reaches its maximum size (X4), and the remainder of the gates are sized to balance the loads (see Section 3.1.1). For example, increasing the sizes of g2 and g3 will increase the load on g1, prompting g1 to be 32×!

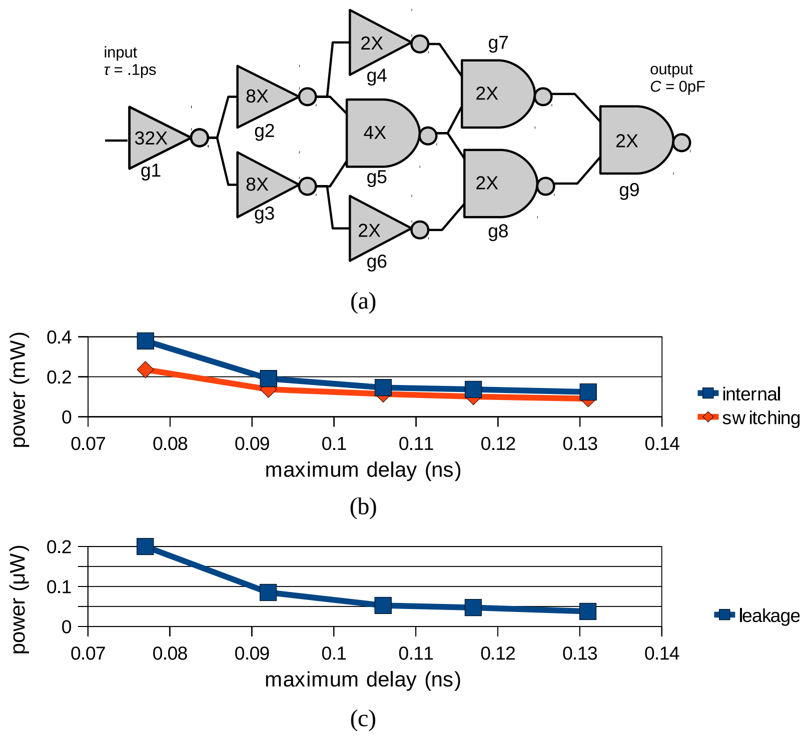


Fig. 3.9 Mesh example from [65]. The optimum sizes for minimum delay are shown.

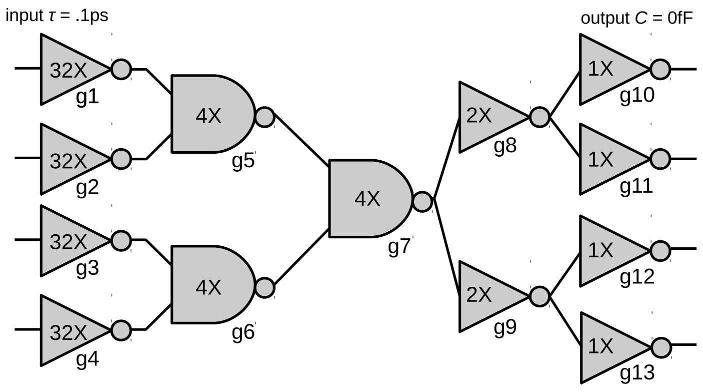


Fig. 3.10 Star example from [65]. The optimum sizes for minimum delay are shown.

The delay in this example ranges from 0.077 ns for the fastest configuration, and 0.131 ns for the configuration with all gates at minimum size. The spread in the total powers is larger, ranging from 0.214 mW at the minimum, and 0.614 mW at the maximum. Note that there is no dip in the internal power (compared to the inverter chain example in Section 3.3.1), as the input transitions do not dominate the internal power.

3.3.3 Star Circuit Example

The star circuit in Figure 3.10 is an example where there is one critical node at the center. In this design, the g7, g8 and g9 gates have an increased load — each has two fanouts. However, the size of g7 is limited to its maximum size of 4×, which limits the sizes of the other gates.

The minimum delay to maximum delay range is 0.112–0.073 ns, with a corresponding total power range of 0.247–1.025 mW. For the minimum delay sizing, gate g7 is set to its maximum and the remainder of the gates are sized to balance the loads (see Section 3.1.1). Notice that the input gates are at maximum size — this is because the delay needed to drive the *primary inputs* is not part of the total delay, it is considered to be driven externally. In the design model the arrival time is the same, irrespective of the size of the primary inputs. Thus, they can be maximized to improve the transition times in the circuit.

4

Methods for Discrete Gate Sizing and V_t Assignment

Discrete gate sizing and threshold voltage assignment methods choose a cell from standard cell libraries for each gate in the design. As the range of gate options is discrete, the problem is inherently combinatorial, and has been shown to be NP-hard [92].

There are many methods that approach the problem of discrete sizing and threshold voltage assignment using continuous sizing.[1] The methods that use rounding generally have at least 8 gate sizes per cell, as in [134]. Other methods use the continuous solution as a starting point, and then employ heuristics to map the gate to discrete sizes [40, 76, 141].

Other methods attack the discrete problem directly. For example, [46] uses an optimization-derived approach that uses gradient-like functions to minimize the delay or power. Liu and Hu [100] use a dynamic programming and Lagrangian relaxation based heuristics, and [117] uses slack allocation via linear programming to improve power. This section will survey the field of discrete gate sizing and threshold voltage assignment, however, the coverage is not exhaustive. Methods that

[1] See, for example [40, 76, 134, 141].

are not covered here include the randomized exhaustive search based methods [169], and graph based methods [28, 29, 30, 167].

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

4.1 Preliminaries

4.1.1 Finding Candidate Moves and Cell Options

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 4.2 and the slack budgeting methods in Section 4.3 use the idea of a sensitivity between the power and the delay:

$$\rho_d = \Delta t_a(g,\omega;\omega_0)/\Delta\mathrm{Power}(g,\omega;\omega_0), \tag{4.1}$$

as in [60], where

$$\Delta\mathrm{Power}(g,\omega;\omega_0) = \mathrm{Power}(g,\omega) - \mathrm{Power}(g,\omega_0) \tag{4.2}$$

$$\Delta t_a(g,\omega;\omega_0) = t_a(g,\omega) - t_a(g,\omega_0) \tag{4.3}$$

and $\mathrm{Power}(g,\omega)$ 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 trade-offs to meet a timing constraint with a lower power design.

A variant of this method is to use a power vs. slack sensitivity [66]:

$$\rho_s = \Delta\mathrm{Power}(g,\omega;\omega_0)/\Delta\mathrm{Slack}(g,\omega;\omega_0). \tag{4.4}$$

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.

4.1.2 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 Section 4.1.1). Candidate gates and cell options are evaluated by how they individually affect local delay, slacks, or power. However, the effects of 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 [47] in Section 4.2 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 a 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 4.3 and 4.7 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 4.6 also use circuit-wide metrics to guide local optimization. The Lagrange multipliers are updated according to 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 4.5 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 [60], and they incorporate specific methods to avoid the pitfalls of greedy optimization.

4.1.3 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. Using the path-based method example in Figure 2.2, the path from G1→G2→`OUT[0]` would be computed separately from the path G1→G2→G3, even though they overlap significantly. This method is more accurate, as less pessimism is needed when the actual path is known, but has the drawback of being computationally expensive; 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 methods. 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:

$$t_{a(g')} + d_{g'} \leq t_{a(g)}, \quad \forall g \in \text{fi}(g'), \tag{4.5}$$

or equivalently:

$$t_{a(g)} + d_g \leq t_{a(g')}, \quad \forall g' \in \text{fo}(g). \tag{4.6}$$

For primary input gates, we have:

$$d_g \leq t_{a(g)}, \quad \forall g \in \text{PI}, \tag{4.7}$$

and for primary outputs, we can write

$$t_{a(g)} \leq T_{\max}, \quad \forall g \in \text{PO}, \tag{4.8}$$

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

$$t_{a(g)} \leq t_{a(\max)}, \quad \forall g \in \text{PO}, \tag{4.9}$$

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.,

$$t_{a(g)} = \max_{\forall g \in \text{fi}(g')} \{t_{a(g')} + d_{g'}\}, \tag{4.10}$$

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 (4.5) can be improved to

$$t_{a(g'(\text{rise}))} + d_{g'(\text{rise}) \to g(\text{fall})} \leq t_{a(g(\text{fall}))}, \quad \forall g \in \text{fi}(g'). \tag{4.11}$$

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

$$t_{a(g'(\text{rise}))} + d_{g'(\text{rise}) \to g(\text{fall})}(\omega, \tau_{g'(\text{rise})}) \leq t_{a(g(\text{fall}))}, \quad \forall g \in \text{fi}(g'). \tag{4.12}$$

The differences in the input pins can also be accommodated by adding new variables, such as $t_{a(g'(\text{rise}),\text{in}1)}$, $t_{a(g'(\text{rise}),\text{in}2)}$. 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:

$$t_{a(g')} + d_{g'} \geq t_{a(g)}, \quad \forall g \in \text{fi}(g'). \tag{4.13}$$

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

$$t_{a(g)} \geq t_{a(\text{min})}, \quad \forall g \in \text{PO}. \tag{4.14}$$

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

4.2 Score and Rank Algorithms

Score and Rank algorithms (SR)[2] for gate sizing and threshold voltage assignment were developed as fast heuristics for gate sizing. These methods include TILOS [60], other greedy heuristics [67, 73, 96, 123, 168], and its variants [46, 53, 148, 149, 150]. The defining characteristic of these methods is a scoring method for each gate that measures its ability to provide a power versus delay 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. 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.

[2] 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.

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

```
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 2: The general Infeasible-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
```

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

- TILOS [60][3] and [168] use the power vs. delay metric

$$\rho_d(g,\omega;\omega_0) = \Delta t_a(g,\omega;\omega_0)/\Delta \mathrm{p}(g,\omega;\omega_0), \tag{4.15}$$

  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

[3] 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.

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, [66] uses the score:

$$\rho_s(g,\omega;\omega_0) = \Delta\mathrm{p}(g,\omega;\omega_0)/\Delta s(g,\omega;\omega_0), \tag{4.16}$$

  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 (4.15). Pant et al. [123] approximates slew effects using sensitivities instead of using the timer.

- The Duet method [148, 149], along with [151], adapts the scoring function from [53]:

$$\rho_{\mathrm{duet}}(g,\omega;\omega_0) \tag{4.17}$$
$$= -\frac{1}{\Delta\mathrm{p}(g,\omega;\omega_0)} \sum_{\forall\alpha\in\mathcal{G}} \Delta d_\alpha(g,\omega;\omega_0) \cdot \frac{1}{s_\alpha - s_{\min} + \epsilon} \tag{4.18}$$

  where $\alpha \in \mathcal{G}$ are the timing arcs in the design (see Section 2.1), $\Delta d_\alpha(g,\omega;\omega_0)$ 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 [47] uses a relax function to score the gates:

$$\rho_{\mathrm{Relax}}(g,\omega;\omega_0) = (\alpha(\Delta\mathrm{p}(g,\omega;\omega_0)) - \epsilon) \cdot \phi\left(\frac{\Delta s(g,\omega;\omega_0)}{|s(g,\omega_0)| + \epsilon}\right) \tag{4.19}$$

  where

$$\phi(x) = \begin{cases} 1 + x & \text{if } x \geq 0 \\ \dfrac{1}{1-x} & \text{otherwise.} \end{cases} \tag{4.20}$$

Table 4.1. Summary of Score and Rank methods for gate sizing and threshold voltage assignment.

Method	Input design	Score	Application
[60]	Min area	(4.15)	Transistor sizing
[47]	Min delay	(4.19)	Gate sizing
[168]	Min area	(4.15)	V_{t}, gate sizing
[67]	Min delay	(4.16)	Gate length biasing
[148, 149]	All gates at high-V_{t}	(4.17)	V_{t}

These methods are summarized in Table 4.1. 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 [60, 67, 148, 149, 168] 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, [47, 67] 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.

4.2.1 Greedy Methods

Although Greedy methods were first applied to gate sizing over two decades [60] ago, they are still widely in use [33]. These methods have the benefit of being scalable, easy to implement, and they perform reasonably well (see Section 5). They use a *locally optimal* choice at each step in the algorithm, which gives these methods the title *greedy*.

The well known TILOS method [60] 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 (4.15). 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. Wu et al. [168] 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.

Gupta et al. [67] 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 an 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.

References [148, 149, 150] 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 (4.17)). Each timing arc and its delay vs. power sensitivity is weighted as:

$$\frac{1}{s_\alpha - \min(\text{slacks}) + \epsilon}. \tag{4.21}$$

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.

4.2.2 Global Sizing

Coudert [47] attempts to improve upon the greedy method. A score function "Relax" (4.19) is proposed to gradually trade-off the delay for the power. The rationale for this approach is given in [48]:

> "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."

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.

Algorithm 3: The Global Sizing method [47].

```
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 4.19);
    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
```

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.

4.3 Slack and Delay Budgeting Methods

Delay budgeting methods [31, 117, 122] were developed as fast, scalable methods for gate sizing and threshold voltage assignment [31].[4] These methods are an improvement over the greedy methods in Section 4.2 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 can account for the topology of the design by prioritizing low sensitivity gates that are bottle-necks for many critical paths.

In this paper, we will focus on the recent approach by [117]. 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.

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 (4.1). These sensitivities are then used in the objective of a linear programming method (Algorithm 4). These sensitivities *weight* the slacks allocated to the

[4] Dai et al. [49] also provides a method for budgeting-based sizing, applied to a single logic gate.

Algorithm 4: Slack budgeting method in [117].

```
Input: Design with gates optimized for minimum delay
$P_{\text{best}} \leftarrow \infty$;
while Power($\mathcal{G}$) < $P_{\text{best}}$ do
    $P_{\text{best}} \leftarrow$ Power($\mathcal{G}$);
    Setup: Compute Power vs. Delay sensitivities;
    foreach $g \in \mathcal{G}$ do
        Compute $\rho_g =$
            $\max_{\omega \in \{\text{CellOptions}(g)\}} \{\rho_d(g,\omega;\omega_0) = \Delta t_a(g,\omega;\omega_0)/\Delta \text{p}(g,\omega;\omega_0)\}$    (4.22)
    end
    Phase I: Allocate slacks to gates using LP;
    Solve:
        $\begin{array}{ll} \text{minimize} & \sum_{g \in \mathcal{G}} \rho_g s_g \\ \text{subject to} & t_{a(g)} + s_g \leq t_{a(g')} \quad \forall g' \in \text{fanout}(g) \\ & 0 \leq t_{a(g)} \leq T_{\max} \\ & 0 \leq s_g \leq D_{g,\max} \end{array}$    (4.23)
     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;
        $\text{argmin}_{\omega \in \{\text{CellOptions}(g) \mid s(g,\omega) - s(g,\omega_0) \leq s_g\}} \{\text{Power}(\omega)\}$
    end
end
```

gates, and the solution to the linear program "allocates the optimal delays to gates depending on their ability to convert slack to power" [117].

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 slack 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 have 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.

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 all of the budget is used.

4.4 Continuous Sizing Based Methods

Continuous sizing based methods are well studied, 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,[5] which returns a set of continuous sizes. Next, these continuous sizes are "snapped" to discrete sizes. This section will cover algorithms for discrete sizing and threshold voltage assignment, snapping methods, and a short discussion on the limitations of continuous sizing based methods.

[5] For further reference on modeling, please see [17, 83, 132].

4.4.1 Linear Programming Methods

The simplest models, which are also the fastest to optimize, are the linear models [11, 158]. Power models are roughly linear (see Section 3.2), and can be approximated as:

$$\text{Power} = \sum_{g \in \mathcal{G}} p_{g,\text{nom}} \cdot w_g, \tag{4.24}$$

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.

The gate delay is not linear in the gate size (see Section 3.1), thus linear models for the gate delay require fitting and approximations. In [11], the gate delay is linearized by:

$$d_g = c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'}, \tag{4.25}$$

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:

$$\begin{aligned} d_g &\geq c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ d_g &\geq c_4 + c_5 w_g - c_6 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ d_g &\geq c_7 + c_8 w_g - c_9 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \\ &\cdots \end{aligned} \tag{4.26}$$

The resulting linear program, using the block-based delay formulation in Section 4.1.3, is:

$$\begin{array}{ll} \text{minimize} & \sum_{g \in \mathcal{G}} p_g w_g \\ \text{subject to} & t_{a(g)} + d_g \leq t_{a(g')}, \quad \forall g' \in \text{fanout}(g) \\ & c_1 + c_2 w_g - c_3 \sum_{g' \in \text{fo}(g)} C_{g'} w_{g'} \leq d_g \\ & 0 \leq t_{a(g)} \leq T_{\max}, \quad \forall g \in \mathcal{G} \\ & w_{\min} \leq w_g \leq w_{\max}, \end{array} \tag{4.27}$$

whose optimum, $w^{\star}$, are the optimal continuous sizes, that need to be "snapped" (see Section 4.4.3).

4.4.2 Geometric Programming Methods

An improvement on the linear model is to use posynomial models [60]. Posynomials are equations of the form:

$$\sum_k \prod_j x_j^{\alpha_{jk}}, \tag{4.28}$$

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[6] for delay:

$$\begin{aligned} \mathrm{Delay}_g &= R_g \frac{\sum_{g' \in \mathrm{fo}(g)} C_{g'} w_{g'}}{w_g} && (4.29) \\ &= R_g w_g^{-1} \sum_{g' \in \mathrm{fo}(g)} C_{g'} w_{g'}. && (4.30) \end{aligned}$$

This results in the geometric programming problem:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{g \in \mathcal{G}} p_g w_g \\ \text{subject to} & t_{a(g)} + R_g w_g^{-1} \displaystyle\sum_{g' \in \mathrm{fo}(g)} C_{g'} w_{g'} \le t_{a(g')}, \quad \forall g' \in \mathrm{fanout}(g) \\ & 0 \le t_{a(g)} \le T_{\max}, \quad \forall g \in \mathcal{G} \\ & w_{\min} \le w_g \le w_{\max}. \end{array} \tag{4.31}$$

While this is not convex as written, it is convex when the change of variables $w_g = e^{y_g}$ is made.[7] Extensions to these models can be made for V_{t} 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.[8] However, convex models are preferred, as they have the property that there exists a unique optimum that can be found in polynomial time [18, 131].

[6] For example, the alpha power law model in Equation (3.1).
[7] Note that the objective is no longer linear under the substitution.
[8] See [17, 83] for a discussion on posynomial models for gate sizing.

4.4.3 "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, [77] 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. Wu et al. [169] 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 not be one of these choices* [40]; however results in [169] show it is within 1% of optimal for a set of 20 commonly used benchmarks.

Heuristics for snapping continuous sizes are studied in [40, 104, 134, 141, 175]. Simple snapping to the closest library cell is used in [104, 134]; in [141], the gate sizing is re-performed after the V_{t} is snapped, followed by the snapping of the gate sizes. In [40], 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 the 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".

Hu et al. [77] 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.

[9] See [9, 70] for a discussion on selecting library cells.

4.4.4 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 [83] for a 0.25 μm technology and furthermore, these errors were reported for custom transistors. Standard cells are more difficult model because 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 a transistor in custom layout. In [133] errors of 5–23% were reported for a 0.13 μm 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% may introduce a significant suboptimality in the design (see, for example, Figure 5.8). Furthermore, modeling errors associated with ignoring slew effect affect gate sizing methods heavily. For example, Figure 5.4 shows the performance of sizing methods with and without slews. The performance is very different between the two slew options.

4.4.5 Limitations of Continuous Sizing Methods

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 [47, 77], but they are severely limited in post-layout contexts.

4.5 Dynamic Programming Based Algorithms

Dynamic programming (DP) [10] 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

[10] See [13] for an introduction to the subject.

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 [24, 77, 95],[11] and reverse topological order [100]. In the forward topological order version, the power and the arrival times are propagated in the form of *cost-to-go functions*:

$$J_g(t_a) = \min_{\omega \in \text{CellOptions}(g)} \left\{ \text{Power}(g,\omega) + \sum_{g' \in \text{fi}(g_k)} J_{g'}(t_a - d(g,\omega)) \right\}. \tag{4.32}$$

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, 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 4.1. 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

[11] Note that [24] does not explicitly refer to itself as Dynamic Programming, but has the characteristics of DP.

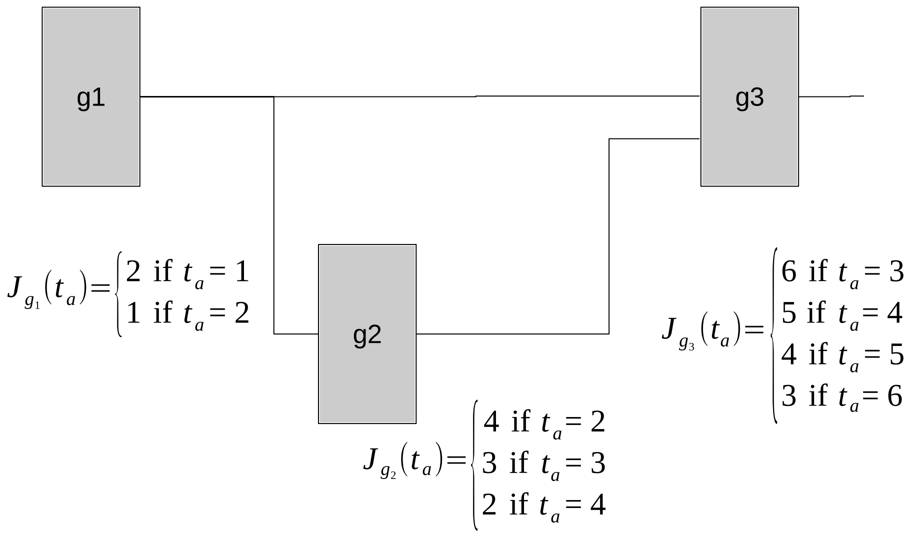

g1
g2
g3
$J_{g_1}(t_a) = \begin{cases} 2 \text{ if } t_a = 1 \\ 1 \text{ if } t_a = 2 \end{cases}$
$J_{g_2}(t_a) = \begin{cases} 4 \text{ if } t_a = 2 \\ 3 \text{ if } t_a = 3 \\ 2 \text{ if } t_a = 4 \end{cases}$
$J_{g_3}(t_a) = \begin{cases} 6 \text{ if } t_a = 3 \\ 5 \text{ if } t_a = 4 \\ 4 \text{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

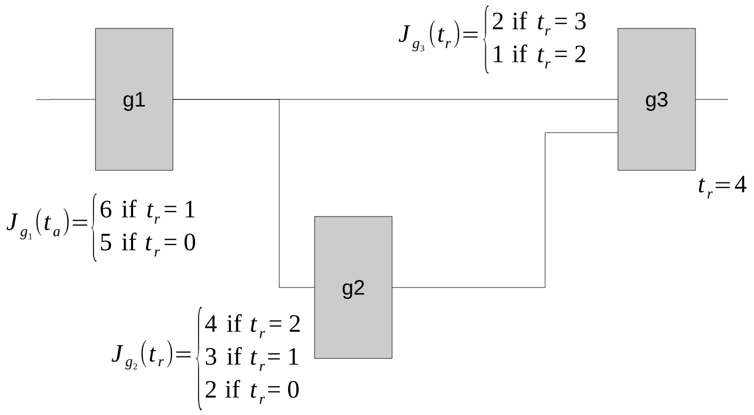
Fig. 4.1 Forward topological order dynamic programming example. A simple delay vs power model is used for the gates, where either Delay = 1 and Power = 2, or Delay = 2 and Power = 1.

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 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 toward 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:

$$J_g(t_r) = \min_{\omega \in \text{CellOptions}(g)} \left\{ \text{p}(g,\omega) + \sum_{g' \in \text{fo}(g)} J_{g'}(t_r - d(g,\omega)) \right\}. \quad (4.33)$$

An example of the reverse topological version is shown in Figure 4.2. 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



$J_{g_3}(t_r) = \begin{cases} 2 \text{ if } t_r = 3 \\ 1 \text{ if } t_r = 2 \end{cases}$
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
$J_{g_2}(t_r) = \begin{cases} 4 \text{ if } t_r = 2 \\ 3 \text{ if } t_r = 1 \\ 2 \text{ if } t_r = 0 \end{cases}$

Fig. 4.2 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.

be eliminated.[12] Once the cost-to-go functions are 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, (4.32) and (4.33), 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, [76] uses locality-sensitive hash functions to reduce the number of entries to the table. In [24, 100], the slews are ignored, reducing

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

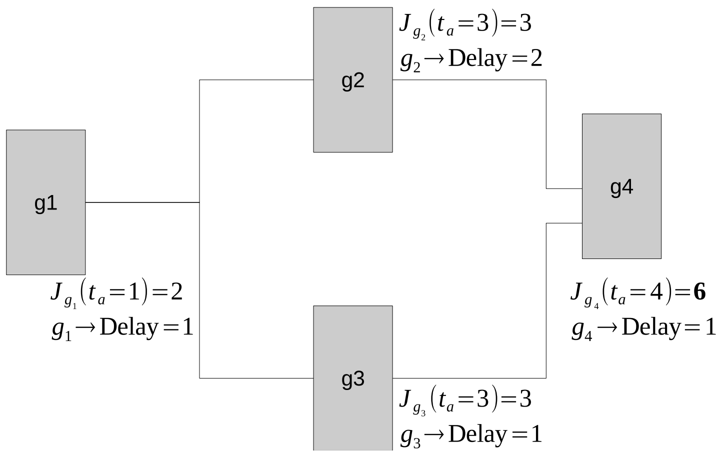

g1
g2
g3
g4
$J_{g_2}(t_a=3)=3$
$g_2 \rightarrow \text{Delay}=2$
$J_{g_1}(t_a=1)=2$
$g_1 \rightarrow \text{Delay}=1$
$J_{g_4}(t_a=4)=\mathbf{6}$
$g_4 \rightarrow \text{Delay}=1$
$J_{g_3}(t_a=3)=3$
$g_3 \rightarrow \text{Delay}=1$

Fig. 4.3 Design with reconvergent fanout. This structure is problematic in Dynamic Programming formulations.

the number of entries to a manageable size. Furthermore, while [100] uses the dynamic programming methods for slack minimization, they rely on Lagrangian methods to perform the timing constrained power optimization (see Section 4.6).

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 4.3; to account for this, the cost-to-go function must be adjusted. As written in (4.32), $J_{g4}(t_a = 4) = 8$ because the cost of g_1 is double counted — it is present in J_{g2} and J_{g3}. However, the actual minimum cost is $J_{g4}(t_a = 4) = 6$.

There is also an inconsistency in the cell options that are assigned by the DP. In the example in Figure 4.3, 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 $J_{g4}(t_a = 4)$, which is Delay = 1/Power = 2 option. Next, suppose that g_2 is also set to Delay = 2/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.

Liu and Hu [100] use 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 (4.32) 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.

4.6 Lagrangian Relaxation

Lagrangian relaxation is widely used in gate sizing and threshold voltage assignment to perform continuous sizing with posynomial models (see, for example, [27, 39, 140, 166]), and it has also been applied directly to discrete gate sizing [78, 100, 121]. 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:

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \ldots, m \\ & g_i(x) = 0, & i = 1, \ldots, p \\ & x_{\min} \leq x \leq x_{\max}, & \end{array} \tag{4.34}$$

the associated *Lagrangian* is:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i g_i(x), \tag{4.35}$$

where λ and ν are the Lagrangian multipliers. Note that the constraints $x_{\min} \leq x \leq x_{\max}$ do not appear in the Lagrangian (4.35). In

[13] See [18] for a good reference on the subject.

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 (4.35) is restricted to $x_{\min} \leq x \leq x_{\max}$.

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

$$\max_{\lambda \geq 0, \nu \in \Re} \left\{ \min_{x_{\min} \leq x \leq x_{\max}} \{L(x, \lambda, \nu)\} \right\}. \tag{4.36}$$

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 $\min_{x_{\min} \leq x \leq x_{\max}} \{L(x, \lambda, \nu)\}$ can be performed faster than solving (4.34) directly.

The main motivation for solving the Lagrange dual problem (4.36) comes from two theoretical results.

- The solution to the dual problem (4.36) is always *less than* the primal (4.34).
- The solution to the dual problem (4.36) is equal to the solution of the primal (4.34) 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

[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 [91] for a good reference on Lagrangian relaxation.

for (4.31) is:

$$L(w,t_a,\lambda) = \sum_{g\in\mathcal{G}} p_g w_g \tag{4.37}$$

$$+\sum_{g}\sum_{g'\in \mathrm{fo}(g)} \lambda_{g,g'}(t_{a(g)} + d(g,w) - t_{a(g')}) \tag{4.38}$$

$$+\sum_{g\in \mathrm{PO}} \lambda_g (t_{a(g)} - T_{\max}). \tag{4.39}$$

The associated Lagrangian dual problem is:

$$\max_{\lambda\geq 0}\left\{\min_{t_a, w_{\min}\leq w\leq w_{\max}} \{L(w,t_a,\lambda)\}\right\}. \tag{4.40}$$

In this expression, the minimization over t_a is equal to $-\infty$ for certain values of λ. This can be seen by reorganizing the terms in (4.43) as:

$$\begin{aligned} L(w,t_a,\lambda) = &\sum_{g\in\mathcal{G}}\left(\sum_{g''\in \mathrm{fo}(g)} \lambda_{g,g''} - \sum_{g'\in \mathrm{fi}(g)} \lambda_{g',g}\right)\cdot t_{a(g)} \\ &+\sum_{g\in\mathcal{G}} p_g w_g + \left(\sum_{g\in \mathrm{PO}} \lambda_g\right)\cdot T_{\max} \\ &+\sum_{g}\left(\sum_{g'\in \mathrm{fo}(g)} \lambda_{g,g'}\right) d(g,w) \end{aligned} \tag{4.41}$$

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 $t\rightarrow -\infty$ and negative coefficients would have $t\rightarrow\infty$. Thus, to avoid these cases, the λ is restricted to the *dual feasible* set that satisfies:

$$\sum_{g''\in \mathrm{fo}(g)} \lambda_{g,g''} - \sum_{g'\in \mathrm{fi}(g)} \lambda_{g',g} = 0, \tag{4.42}$$

thereby making all the coefficients of t_a in (4.41) equal to zero. For these dual feasible λ, the Lagrangian reduces into a weighted delay-power minimization:

$$\begin{aligned} L(w, t_a, \lambda) = &\sum_{g \in \mathcal{G}} p_g w_g \\ &+ \sum_g \left(\sum_{g' \in \mathrm{fo}(g)} \lambda_{g,g'} \right) d(g, w) \\ &+ \left(\sum_{g \in \mathrm{PO}} \lambda_g \right) \cdot T_{\max}. \end{aligned} \tag{4.43}$$

For continuous gate sizes and V_{t}, there are fast methods to solve this problem (see [131]) 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 [78, 100, 121]. In this case, idea is to solve the discrete version of the Lagrangian dual problem:

$$\max_{\lambda \geq 0} \left\{ \min_{t_a, \omega \in \mathrm{CellOptions}} \{ L(\omega, t_a, \lambda) \} \right\}, \tag{4.44}$$

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 unconstrained, the minimization is still combinatorial. Liu and Hu [100] use 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 4.5) limit the ability to apply dynamic programming. Ozdal et al. [121] 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 the updating of the multipliers λ. Traditionally, the multipliers are updated according to the sub-gradient (a generalized gradient).

$$\Delta\lambda = \frac{\partial}{\partial\lambda}(L(\omega, t_a, \lambda)). \tag{4.45}$$

λ next updated using two steps as

(1) $\lambda = \lambda + \alpha\Delta\lambda$
(2) λ is projected to satisfy (4.42) 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. Huang et al. [78] 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.

4.7 Slew Targeting Methods

Slew targets [74] 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 as the slew targeting is used 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 [74] 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). However, as slew has a large effect on the delays, ignoring slews may introduce sub-optimality into the design.

Held's algorithm [74] 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:

$$\mathrm{lc}(g) = \max_{g' \in \mathrm{fi}(g)} \{s(g) - s(g'), 0\}. \tag{4.46}$$

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, $\Theta_k \to 0$, prompting the algorithm to use the true slew values in place of the estimates for increased accuracy.

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" [74]. 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:

$$\tau_g^{\mathrm{target}} = \tau_g^{\mathrm{target}} + (\lambda) \cdot (s^p - \tau_g^{\mathrm{target}}). \tag{4.47}$$

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

Algorithm 5: Slew targeting method in [74].

```
$\Theta_k$ = 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
            $\Delta\tau_{\mathrm{target}} = -\min\{(\Theta_k \cdot \gamma \cdot |s_g|), \Delta\tau^{\mathrm{target}}_{\max}\}$;
        else
            $\mathrm{sm} = \max\{s(g), \mathrm{lc}(g)\}$;
            $\Delta\tau^{\mathrm{target}} = \min\{(\Theta_k \cdot \gamma \cdot |\mathrm{sm}|), \Delta\tau^{\mathrm{target}}_{\max}\}$;
        end
        $\tau^{\mathrm{target}}_g = \tau^{\mathrm{target}}_g + \Delta\tau^{\mathrm{target}}_g$;
        Project the slew target to feasible range
        $\tau^{\mathrm{target}}_g = \max\{\tau^{\mathrm{target}}_g, \Delta\tau^{\min}_g\}$;
        $\tau^{\mathrm{target}}_g = \min\{\tau^{\mathrm{target}}_g, \Delta\tau^{\max}_g\}$;
    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
            $\hat{\tau}(g') = \Theta_k \cdot \tau^{\mathrm{target}}_{g'} + (1 - \Theta_k) \cdot \tau_{g'}$
        end
        Using the estimated input slews $\hat{\tau}(g')$ set $\omega_g$ as:
        the minimum power cell option $\omega$ that satisfies
        $\tau(g, \omega) > \tau^{\mathrm{target}}_g$
    end
    Step 3: Refine slew targets;
    foreach $g \in \mathcal{G}$ in topological order do
        $s^p = \max_{\{g' |\ s_{g'} = s^{\mathrm{fi}}_g\}} \{\hat{\tau}(g')\}$
        $\tau^{\mathrm{target}}_g = \tau^{\mathrm{target}}_g + (\lambda) \cdot (s^p - \tau^{\mathrm{target}}_g)$
    end
end
```

report good results for timing closure with an average worst negative slack of 6% reduced to 2% after applying their algorithm.

4.8 Linear Programming Based Assignment Methods

Linear programming can also be used to assign gates to specific cell options [2, 34, 89]. This is in contrast to the continuous sizing methods in Section 4.4.1, where the variables represent the continuous gate sizes. In linear programming based assignment, each variable in this context is a binary variable, $x_{g\leftarrow\omega}$, where:

$$x_{g\leftarrow\omega} = \begin{cases} 1 & \text{if gate } g \text{ is assigned to cell } \omega \\ 0 & \text{otherwise.} \end{cases} \tag{4.48}$$

A value of $x_{g\leftarrow\omega} = 1$ means that gate g is implemented by the library cell ω. These options ω can vary in sizes, threshold voltages, gate lengths, etc. The variables $x_{g\leftarrow\omega}$ 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:

$$\sum_{\forall g\in\mathcal{G}} \sum_{\omega\in\Omega_g} \Delta p(g,\omega;\omega_0) \cdot x_{g\leftarrow\omega} \tag{4.49}$$

where

$$\Delta p(g,\omega;\omega_0) = p(g,\omega) - p(g,\omega_0). \tag{4.50}$$

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]:

$$
\begin{aligned}
t_{a(g)} + \sum_{\omega \in \Omega_g} \Delta d(g,\omega;\omega_0) \cdot x_{g \leftarrow \omega} \le t_{a(g')}, \quad \forall g' \in \text{fanout}(g) \\
0 \le t_{a(g)}, \quad \forall g \qquad (4.51) \\
t_{a(g)} \le T_{\max}, \quad \forall g \in \text{PO}. \qquad (4.52)
\end{aligned}
$$

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 [89].

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

$$
\begin{aligned}
\text{minimize} \quad & \sum_{\forall g \in \mathcal{G}} \sum_{\omega \in \text{CellOptions}(g)} \Delta p(g,\omega;\omega_0) \cdot x_{g \leftarrow \omega} \\
\text{subject to} \quad & t_{a(g)} + \sum_{\omega \in \text{CellOptions}(g)} \Delta d(g,\omega;\omega_0) \\
& \quad \cdot x_{g \leftarrow \omega} \le t_{a(g')}, \quad \forall g' \in \text{fanout}(g) \\
& 0 \le t_{a(g)} \le T_{\max}, \quad \forall g \in \mathcal{G} \\
& 0 \le x_{g \leftarrow \omega} \le 1, \quad \forall g, \ \omega \in \text{CellOptions}(g) \\
& \sum_{\omega \in \text{CellOptions}(g)} x_{g \leftarrow \omega} \le 1, \quad \forall g.
\end{aligned}
\qquad (4.53)
$$

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.

Once (4.53) is solved, then the next step is to map the $x_{g \leftarrow \omega}$ into gate cell changes. In [34], the authors apply any $x_{g \leftarrow \omega} > 0.99$. However, due

[16] See Section 4.1.3 for more information on the block-based delay formulation. For simplicity, the difference between the rise and fall times are omitted, as well as the difference in delay for different input–output paths in a gate. However, they are commonly incorporated by adding separate rise and fall arrival-time variables and additional delay models for different input–output paths in a gate.

to the delay interactions, this may cause timing violations. They correct these timing violations by running a linear programming assignment for minimizing timing:

$$
\begin{aligned}
\text{minimize} \quad & \max\{t_{a(\text{output})}, T_{\max}\} \\
& \quad + \cdots \epsilon \sum_{\forall g \in \mathcal{G}} \sum_{\omega \in \Omega_g} \Delta p(g, \omega; \omega_0) \cdot x_{g \leftarrow \omega} \\
\text{subject to} \quad & t_{a(g)} + \sum_{\omega \in \Omega_g} \Delta d(g, \omega; \omega_0) \\
& \quad \cdot x_{g \leftarrow \omega} \cdots \leq t_{a(g')}, \quad \forall g' \in \text{fanout}(g), \\
& 0 \leq t_{a(g)} \leq t_{a(\text{output})}, \quad \forall g \in \mathcal{G} \\
& 0 \leq x_{g \leftarrow \omega} \leq 1, \quad \forall g, \omega \in \Omega_g \\
& \sum_{\omega \in \Omega_g} x_{g \leftarrow \omega} \leq 1, \quad \forall g
\end{aligned}
\tag{4.54}
$$

where ϵ is a small constant used to give more weight to the timing objective. In the minimizing timing case, values of $x_{g \leftarrow \omega} > 0.01$ 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 a increase in the gate size of g. Due to this limitation, [89] adds a further constraint on neighboring gates. That is, out of every pair of gates, only one gate should change at a time. This reduces the problems due to the changing capacitances of neighboring gates.

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, [89] 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).

4.9 Summary

The discrete gate sizing and threshold voltage assignment methods presented in this section are summarized in Table 4.2. This chart indicates whether the method is well-suited (✓), applicable (⋆), 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 the 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 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 an minimum delay design, or to meet a timing constraint.

Table 4.2. Summary of discrete gate sizing and threshold voltage assignment methods.

	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	⋆	✓	✓	✓

5

Comparing Sizing and Assignment Methods

This section considers the different factors involved in comparing gate sizing and threshold voltage assignment methods, and provides some comparative results. There are many contexts in which these methods are used (see Section 1.2 for more information):

- **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 (e.g., have set 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.

Along with these different contexts is the experimental setup. The library, buffering, timing model, benchmark type, timing constraints, and power measurements all affect the resulting optimizations. 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.

5.1 Setting Up Experiments

In gate sizing and threshold voltage assignment, there are a significant number of variables that affect the success of a method. In this section we will discuss:

(1) Standard Cell Library: what cell options are available for each gate?
(2) Benchmarks and synthesis options: what are the characteristics of the test circuit?

5.1.1 Standard Cell Library

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 1–2× of the minimum gate width, can provide only small impacts, as the sizing method can increase gate sizes to just 2× 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 5.2× to 5× is much better than rounding down to 4×.

In contrast, other methods may fair 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 also fair poorer than their continuous sizing based counterparts. The granularity question — is the library sparse or dense — may cause one method to perform better than another.

However, there is no *typical* library density or range. One 65 nm commercial library provides 11 versions of the buffer and inverter cells, and 4 versions of other cells. The Nangate Library [114], 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 45 nm library provides 20 versions of the buffer and inverter cells (ranging from 0.5× to 16×), and between 5 and 11 versions of other cells (ranging from 0.5× to 6×). In addition, there are three 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.

5.1.2 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 contain combinatorial designs representing controllers, arithmetic logic units, and error correcting circuits [71]. The ISCAS ‘89 benchmarks contain sequential designs[1] and are sections of real designs, traffic light controllers, and digital fractional multipliers [20].

[1] For example, these designs include flip-flops.

Table 5.1. Effect of synthesis on the ISCAS '85 benchmarks [71] (adapted from [89]). All times are in ns.

	$T_{\max}$	$\|\mathcal{G}\|$	$T_{\max}$	$\|\mathcal{G}\|$	$T_{\max}$	$\|\mathcal{G}\|$	$\%_{\Delta T_{\max}}$	$\%_{\Delta\|\mathcal{G}\|}$
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%

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.

A complicating factor is the effect of the synthesis tool on the design. Table 5.1 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 0.188 ns (c432) to 6.847 ns (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.

Table 5.2. Effect of different timing constraints used during synthesis on the final layout (adapted from [80]) for an AES encryption circuit from [6]. After synthesis, a timing constraint of 2.0 ns is used to perform placement, routing and incremental timing optimizations.

Timing Constraint used @ synthesis	With a 2.0 ns 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

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 also 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 5.2 shows the effect of a variety timing constraints used during synthesis given in [80] on the final design. A tighter timing constraint at synthesis does not always lead to a faster design, and the relationship is not well-behaved.

5.2 Post-layout Considerations

Post-layout discrete gate sizing and threshold voltage assignment are done after cell placement, interconnect routing, and clock tree synthesis (see [89]). They are performed 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 [73], as well as to recover power with accurate timing models.

5.2.1 Post-layout Timing Considerations

As placement and routing have been performed, post-layout timing estimates may utilize (see Section 2):

(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 [80]. 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 5.1. Though the timing estimates correlate well, the values differ by a substantial margin. The error is as large as 0.2 ns, 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 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

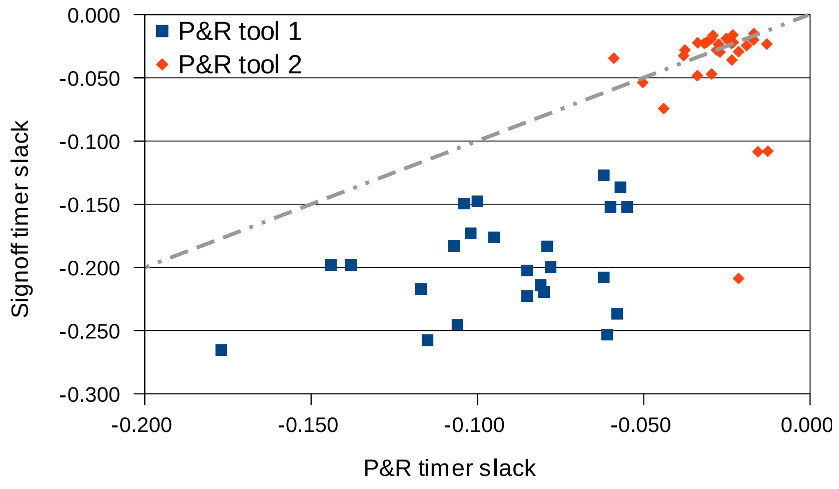


Fig. 5.1 Mismatch between the timing estimates at place and route and at sign-off. The gray line indicates when the two estimates match. Data adapted from, and courtesy of [80].

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 5.4 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.

5.2.2 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,[2] and may affect other aspects of the design.

[2] 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.

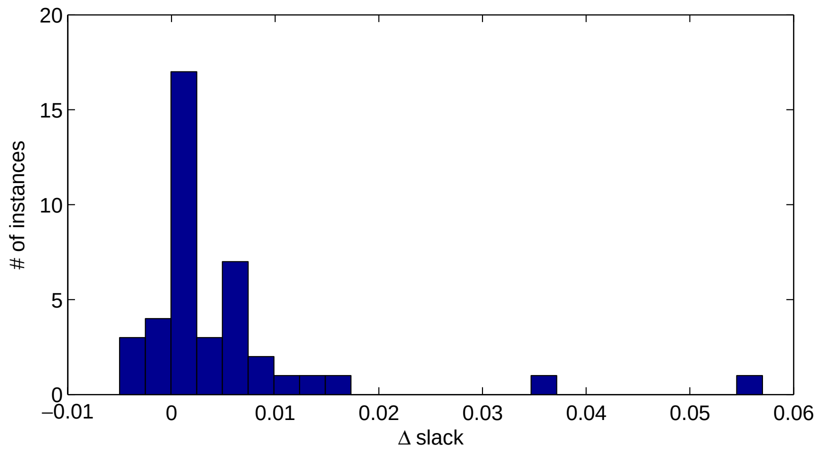


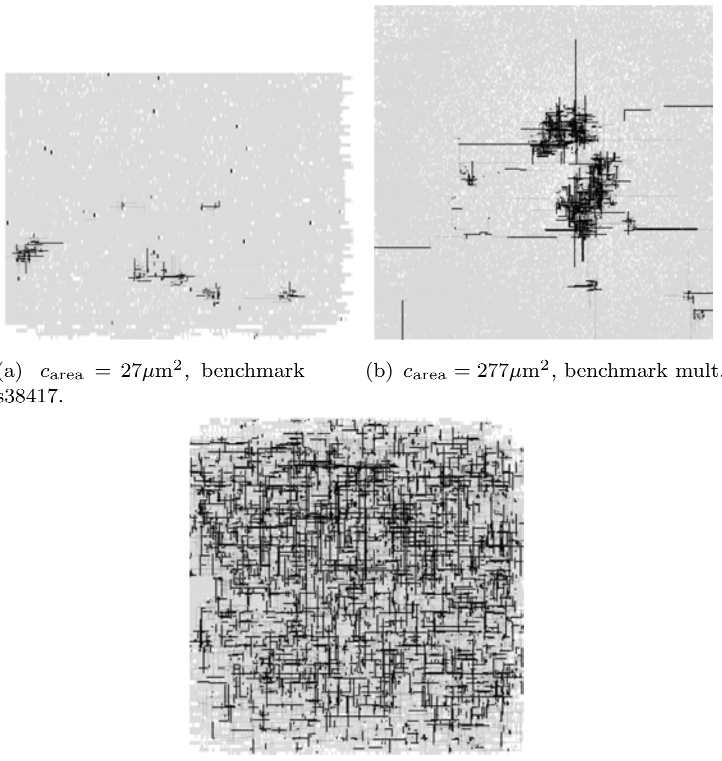
Fig. 5.2 Histogram of the change in the timing after incremental placement and routing. There are 64 different examples. The majority of the cases have a zero change in timing, and most of the changes increase the timing (the change is positive). The maximum decrease and increase are -0.007 ns and 0.057 ns, respectively.

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 5.2.

5.2.3 Measuring an Engineering Change Order (ECO)

In the post-layout context, there is an additional consideration of having a minimal design disturbance as a means to minimize the implementation cost. This is especially important when the verification process has already begun. Larger ECOs may require more time to implement and verify, and may case unwanted changes that will lead to additional ECOs.

Examples of ECOs are shown in Figure 5.3. The changes are marked in black. The changes in Figure 5.3(a) are small compared to the changes in Figure 5.3(c). Lee et al. [89] proposes that this changed area be measured as an ECO area cost.



(a) $c_{\text{area}} = 27\mu\text{m}^2$, benchmark s38417.

(b) $c_{\text{area}} = 277\mu\text{m}^2$, benchmark mult.

(c) $c_{\text{area}} = 1159\mu\text{m}^2$, benchmark s35932.

Fig. 5.3 Visual example of ECOs. The changed cells, wires and vias are marked in black.

Another consideration proposed in [89] is the use of an ECO timing cost. This is the number of positive-slack output pins in a design that had their timing signal changed. These pins did not need any changes in their signal as they were positive-slack; thus, changes in their signal are unnecessary and can be considered a cost. These measures were used in [89] to perform incremental, low ECO cost gate sizing.

5.3 Comparisons

5.3.1 Comparisons Reported in the Literature

The comparisons reported in the gate sizing and threshold voltage assignment literature are difficult to navigate. Most V_t assignment

Table 5.3. Improvements reported in literature.

		Improvement			
Method	Comparison	Min (%)	Max (%)	Avg (%)	
DP+LP [101]	Budgeting [117]	1	31	21	w, V_t
LSH [77]	Greedy [47]	9	31	18	w
Continuous based [141]	Greedy [148]	6	62	31	w, V_t
Global sizing [47]	Greedy [143]	5.3	57.3	11	w
Continuous based [40]	Greedy [98]	0	12	5	w
Continuous based [134]	Com.*	1	39	11	w
LP assignment [34]	Com.*	−3.6	44.9	15.3	w

*These methods were compared against the results of a commercial synthesis tool.

papers ([117, 148, 149, 150, 168]) report improvements over no V_t assignment. Similarly, [67] 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 results from methods in Section 4 are reported in Table 5.3. However, these methods are difficult to compare, as the experimental setup differs. The standard cell library, wire models, synthesis options and timing constraints may be too different to compare the results between papers.

An interesting set of comparisons is found in [169]. A branch-and-bound solver is implemented, which provides the optimal sizing solutions for ISCAS '85 [71] benchmarks and ISCAS '89 [20] benchmarks. For these benchmarks, the greedy method achieves between 2% and 35% suboptimality from the optimum, with an average suboptimality of 11%. A simulated-annealing method is also compared, which achieves between 0.5% and 28% suboptimality from the optimum, with an average suboptimality of 7%.

A common site for benchmarking gate sizing and threshold voltage assignment algorithms is available at http://vlsicad.ucsd.edu/SIZING.[3] This website provides methods implemented in Open Access, PrimeTime and Encounter, with the source code freely available. The goal of the website is to benchmark the current state of gate sizing and threshold voltage assignment, and provide an infrastructure for future experiments.

[3] This website is developed by researchers at UCSD and has some contribution from the authors of this paper.

5.3.2 Comparisons using UCLA Timer

Comparisons of gate sizing and threshold voltage assignment methods can be found in [109]. In this comparison, an LP assignment [117] and the LR+DP method [100] 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 [114] and a commercial 65 nm library. The static timing analysis tool used in this work is the UCLA Timer [110],[4] which is an open source project based on Open Access[5] and the defunct OA Gear.[6] The open-source nature of this timer provides flexibility in implementing the sizing methods and also increased access to timing information.

The results are shown in Table 5.4. 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

Table 5.4. Comparisons using UCLA timer.

	Nangate 45 nm		Commercial 65 nm			
	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

Percentages denote improvements over a feasible-start greedy algorithm (see Section 4.2.1).

[4] Available at http://www.nanocad.ee.ucla.edu/Main/DownloadForm.
[5] See http://www.si2.org.
[6] http://www.si2.org/openeda.si2.org/projects/oagear.

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 5.4 shows that disabling slew increases the difference between the sizing methods.

5.3.3 Comparisons Using Eyecharts

Eyecharts[7] [65] 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 5.5, which have the property that the optimal solution can be computed using a dynamic-programming-like optimization.

The appeal of Eyecharts is that the optimal solutions are known. This makes it useful as a benchmarking tool for sizing methods.

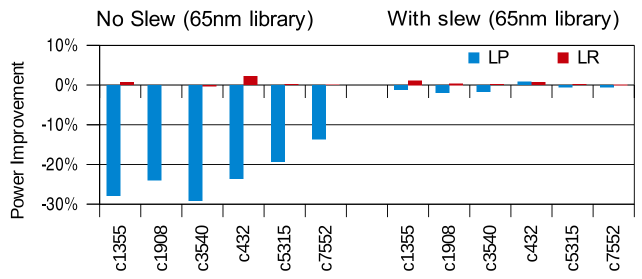


Fig. 5.4 The effects of disabling slew in sizing experiments; the performance improvement is relative to the timing-feasible greedy method. Disabling slew increases the difference between sizing methods.

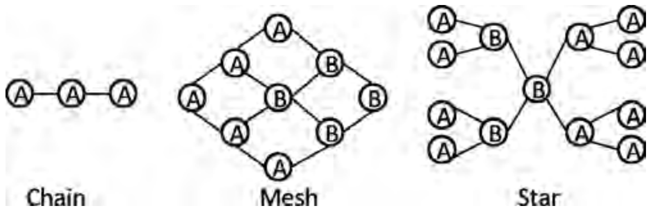


Fig. 5.5 The basic structures used in the eyechart benchmarks.

[7] Eyecharts are available for download at http://www.nanocad.ee.ucla.edu/Main/Download Form.

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 [65], two commercial tools, (Comm1) and (Comm2), the LP slack allocation in [117] (LP), a greedy sizing method using ρ_d from Equation (4.15) (GS), and a greedy sizing using ρ_s from Equation (4.16) (SBS) are compared.

Figure 5.6 show the results of the comparisons. The LP method performs the best in the gate sizing context (Figure 5.6(a)), while the commercial tool (Comm2) performs the best in the V_{t} (Figure 5.6(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 5–15% from optimal.

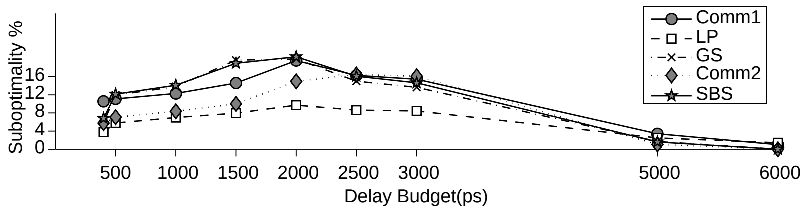
Figure 5.7 shows the impact of the number of sizes on the resulting suboptimality. The suboptimality decreases overall as the number of sizes for each gate increase. However, this is not true for the commercial tools (Comm1) and (Comm2) — they perform as well on libraries with two cells as they do with libraries with 10 cells.

5.3.4 Comparisons Using Commercial Tools

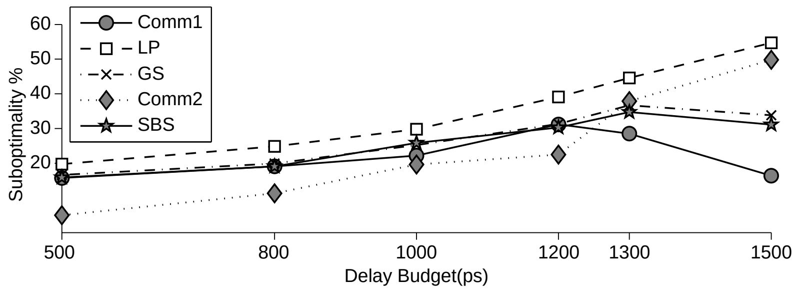
Sizing and threshold voltage assignment methods were also compared using TCL scripts implemented in Cadence Encounter [22].[8] Implementations of four gate sizing and threshold voltage assignment methods were compared:

(1) TILOS [60].
(2) Feasible-start greedy method (Algorithm 1).
(3) LP based slack assignment (Section 4.3).
(4) LP assignment method (Section 4.8).

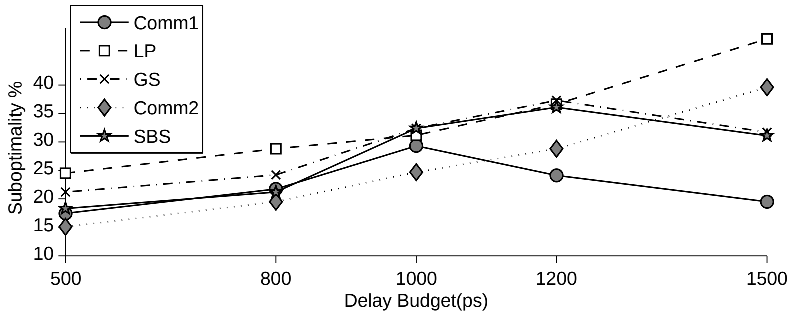
[8] These scripts are available at http://www.nanocad.ee.ucla.edu/Main/DownloadForm.



(a)



(b)



(c)

Fig. 5.6 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 three threshold voltages, (c) represents a gate length assignment with three gate lengths.

Experiments were run on the Nangate Open Cell Library [114] and a commercial 65 nm 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 [117], 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.

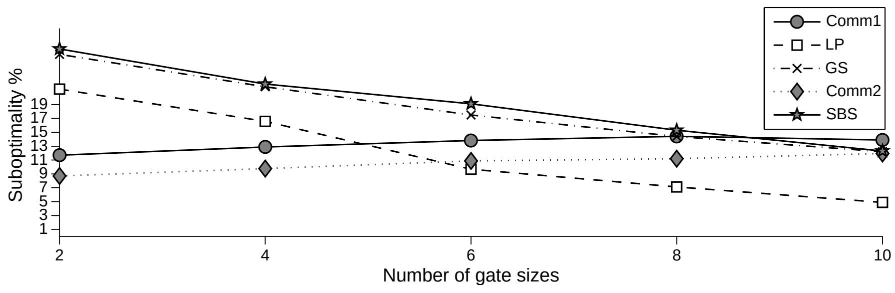


Fig. 5.7 The effect of the number of gate sizes available in the library on the suboptimality of the gate sizing method.

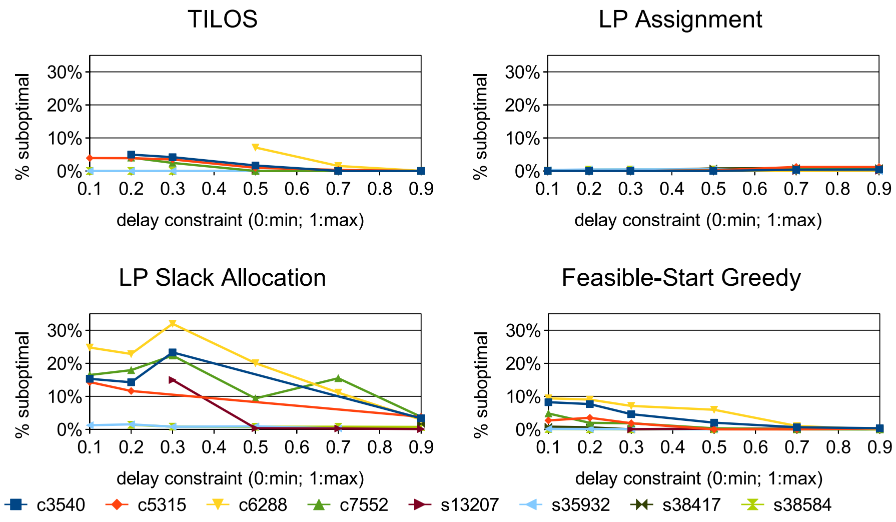


Fig. 5.8 Comparison between sizing methods using the Nangate 45 nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint — 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

Results are shown in Figures 5.8, 5.9, and 5.10 for the Nangate Library, the commercial 65 nm library with gate sizing only, and the commercial 65 nm with V_t assignment, respectively. The plots show the relative leakage power suboptimality of each of the method with the 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

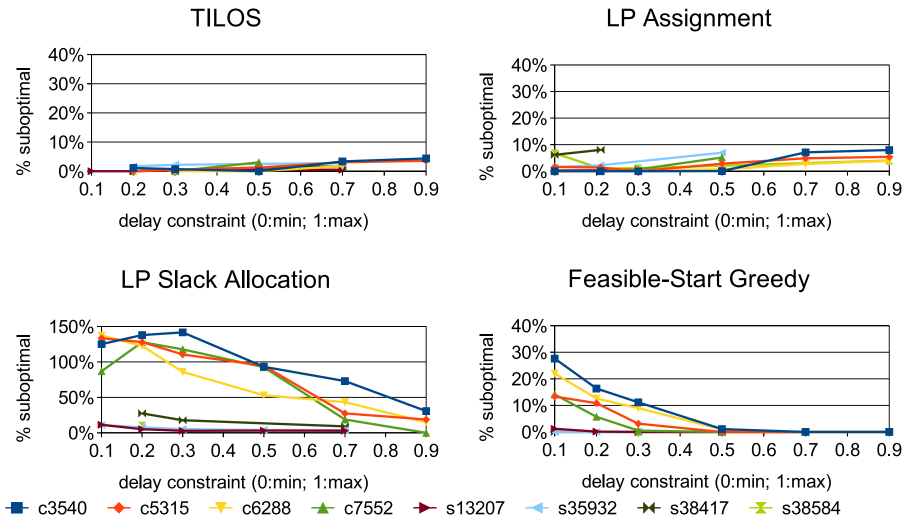


Fig. 5.9 Comparison between sizing methods using a commercial 65 nm Library. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint — 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

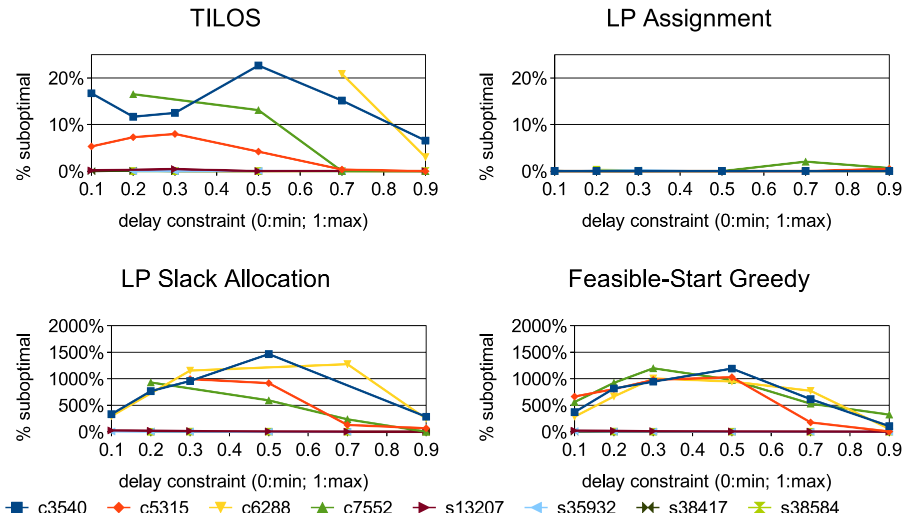


Fig. 5.10 Comparison between V_t assignment methods using a Commercial 65 nm Library=. The y-axis gives the % suboptimality relative to the best solution found. The x-axis is the delay constraint — 0.0 is the minimum delay and 1.0 is the maximum delay. When the data point is omitted, it indicated that algorithm did not return a timing-feasible solution.

of the results vary by library; in the Nangate Library results in Figure 5.8, 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 0.25%, 1.1%, 2%, and 11%, respectively.

The results for the commercial 65 nm with gate sizing (Figure 5.9) 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 0.2). 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 65 nm with V_t assignment has the most dramatic results (see Figure 5.10). 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%.

6
Statistical Gate Sizing

In nanometer designs, the effects of variability are too large to ignore [115]. This is best seen by examining the ITRS Roadmap 2009 [79], 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 6.1 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" [79]. These methods use statistical models for delay and power to create designs that are robust against variation.[1]

[1] For an in-depth discussion of statistical variations and modeling, see [94].

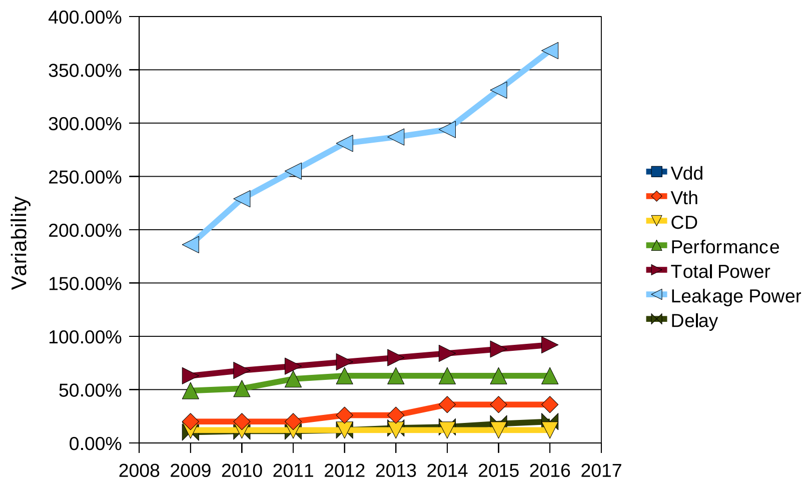


Fig. 6.1 Projected variability in key design parameters (adapted from [79]).

Parametric variations are generally divided into two types [152]: (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 [93], such as dishing, diffraction effects, line-edge roughness, lens aberrations, and dopant fluctuations. Methods to reduce the impact of these variations will be outlined in this section.

6.1 Motivating Examples

6.1.1 The Slack Wall

An excellent motivational example for statistical design is the "slack wall" [7]. 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 allowed.
(2) Paths faster than the maximum delay are not optimal — they can be slowed down to save power.

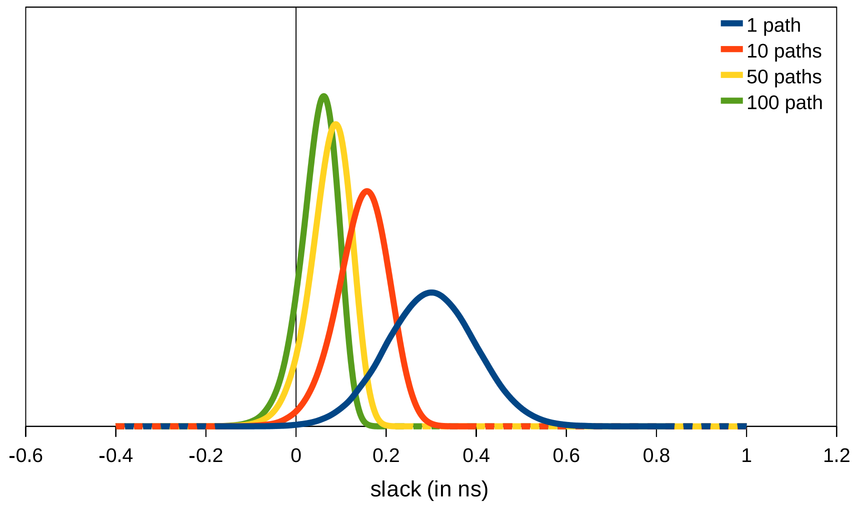


Fig. 6.2 Effects of a "slack wall" on statistical slack. The PDF for the worst-case slack is shown for 1, 10, 50 and 100 paths. The delay of each path is Gaussian with variance 100 ps.

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. This effect is illustrated in Example 6.1.

Example 6.1. Suppose that a 300 ps guard-band is used to ensure a high yield. Also suppose that the delay variation of each path has a standard deviation of 100 ps. Thus, the yield would be expected to be the $3 - \sigma$ 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 6.2 plots the statistical slack probability density distribution (PDF) as a function of the number of independent paths. As the number of independent paths increases, the slack distribution shifts toward 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 [164].

6.1.2 Worst-case Corners and Over-design

Most earlier design methods relied on corner based methods for design [111, 116, 162]. 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 45 nm Library are shown in Table 6.1.

When the variations become large, as in Figure 6.1, 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 $3-\sigma$ value of the variations. However, when there are n independent sources of variation, each with standard variations $\{\sigma_1,\ldots,\sigma_n\}$, the corner must be fitted to provide an overall $3-\sigma$ value. Using the $3-\sigma$ value for each of these parameters will result in a $\sqrt{n}\cdot 3\sigma$ value [69], 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, $\mathbf{D}_1,\ldots,\mathbf{D}_N$ that can be generated from

Table 6.1. Corners in the Nangate 45 nm library [114].

Corner	Temperature	Supply voltage	Process
Fast	0°C	1.25 V	FastFast
Typical	25°C	1.1 V	TypicalTypical
Slow	125°C	0.95 V	SlowSlow

$N+1$ IID $\mathcal{N}(0,1)$ random variables $\mathbf{X}_1, \ldots, \mathbf{X}_N$ as:

$$\begin{bmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_N \end{bmatrix} = 1 + \begin{bmatrix} \alpha & 0 & \ldots & 0 & (1-\alpha) \\ 0 & \alpha & \ldots & 0 & (1-\alpha) \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \ldots & \alpha & (1-\alpha) \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{N+1} \end{bmatrix}. \qquad (6.1)$$

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 and, moreover, 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 $3-\sigma$ points may be overly pessimistic.

These criticisms are not uncontested. In [111] 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 6.2. A larger number of critical paths will cause the yield to decrease, and thus a conservative approach may be needed.

6.2 Slack-wall Methods

In [7], the authors present a method to avoid the effect of the "slack wall" in Section 6.1.1. This slack wall is the result of a deterministic timing constraint where each of the paths is required to be less than or equal to the clock period, but there is no incentive to be any faster. Thus, many of the paths have delays that are near the clock period, and when statistical delays are added, many of the paths violate timing.

To counter this, the deterministic timing constraint is replaced with a modified timing constraint that gives paths an incentive to be faster than needed. This is done using a *penalty*; at each primary output i, the penalty is:

$$\sum_{i \in \mathrm{PO}} \mathrm{e}^{-t_{a\,i}/\theta}, \tag{6.2}$$

where θ is a constant that is used to tailor the profile of the slack wall. This is then added to the power objective and the resulting problem is solved as a multi-objective problem.

6.3 SSTA Based Methods

One branch of statistical gate sizing methods utilizes the developments in Statistical Static Timing Analysis, which adapts the traditional Static Timing Analysis methodology, to account for statistical variations (see Section 2.10). Agarwal et al. [4] implements the greedy sizing heuristic from Section 4.2.1 using the modified score function:

$$\rho_{t_{a\beta}} = (t_{a\beta}(g,\omega) - t_{a\beta}(g,\omega_0))/\Delta p(g,\omega;\omega_0), \tag{6.3}$$

where $t_{a\beta}$ is the β-percentile of the arrival time. Guthaus et al. [69] uses the statistical slack version:

$$\rho_{s_\beta} = (s_\beta(g,\omega) - s_\beta(g,\omega_0))/\Delta p(g,\omega;\omega_0), \tag{6.4}$$

where s_β is the β-percentile of the slack. Srivastava et al. [151] utilizes this score as well, but uses the statistical slack version of (4.17) from Section 4.2 for timing closure. This is an adaptation of prior SSTA work; by changing the timing to statistical timing, methods become *statistical* sizing methods.

6.4 Gate Delay Approximation Heuristics

Several works use a continuous sizing framework that uses approximations for the statistical gate delay [36, 104, 106, 124, 146]. The idea here is to *approximate* the statistical delay of the design by using the statistical library cell delays in place of the deterministic gate delay. Specifically, a percentile of the cell delay over the variation sources, ξ, is used:

$$d_\beta(g) = [\beta\text{−percentile}](d(g,\xi)). \tag{6.5}$$

When the underlying delays are Gaussian, this is equivalent to:

$$d_\beta(g) = d(g,\xi) + \kappa\sigma_d(g,\xi), \tag{6.6}$$

where σ_d is the standard deviation of the delay. This is then incorporated a block-based delay formulation in Section 4.4.2.

At first glance, this looks equivalent to the corner based formulation — each gate is set to be at a "worst-case" type delay corner. However, there are two differences. The first difference is that the standard deviation of the delay is gate dependent. According to Pelgrom's model [125], the variation in $V_{\rm t}$ is $\propto \frac{1}{\sqrt{WL}}$ and larger gates will have a smaller σ_d. Thus, these heuristics have the advantage of using larger gates to reduce the variation in $V_{\rm t}$, and consequently, in the delay.

The next difference is in the choice in the κ. Worst-case corner methods apply the same value of κ for all gates in the design (see [139]). However, this precludes the ability to adjust the κ values to match the actual criticalities. Some gates may heavily influence the statistical delay if it is the bottleneck of many critical paths, while other gates may effect the statistical delay less if they contribute to only a few paths. In [105, 145], heuristic methods to tune the κ are presented to improve the accuracy, and to reduce the pessimism described in Section 6.1.2.

6.5 Convex Functions of Statistical Delay

Another approach to sizing with a yield constraint is discussed in [51] and [43]. The idea here is to use convex functions of statistical delay, as they have the attractive property that there are no local minima that are not global minima.

Davoodi et al. [51] minimizes the Bin–Yield Loss function:

$$\mathrm{BYL} = -\int_{-\infty}^{0} s \cdot \mathbf{Prob}(\mathrm{slack} = s) \cdot \mathrm{d}s, \tag{6.7}$$

where $\mathbf{Prob}(\mathrm{slack} = s)$ is the probability density function of the slack. This expression is the negative of the expected value of the negative slack. It is zero when there is no delay violations, in which case the probability of a negative slack is *zero*. This function is convex when the delay of the circuit is convex, and is used as a convex proxy to maximize the yield.

Cong et al. [43] presents a convex approximation to the yield function. Their mean-excess delay function is given as:

$$\mathrm{MED}_\beta = \min_z \left\{ z + \frac{1}{1-\beta} \int_z^{\infty} (t - z) \cdot \mathbf{Prob}(\mathrm{delay} = t) \cdot \mathrm{d}t \right\}, \tag{6.8}$$

where $\mathbf{Prob}(\mathrm{delay} = z)$ is the probability density function for the delay. The utility of this function is in approximating percentiles, using the following property of the MED_β:

$$\mathrm{MED}_\beta \geq \mathrm{percentile}_\beta. \tag{6.9}$$

Thus, a constraint that the β percentile delay is less than $T_{\max}$ can be approximated as:

$$\mathrm{MED}_\beta \leq T_{\max}. \tag{6.10}$$

This is preferred over the percentile constraint, as the percentile is generally not a convex function of the delay.

The MED is a general version of the BYL; when t is equal to the clock period, the MED is identical to the Bin–Yield Loss function.

6.6 Statistical Power Considerations

All of the prior discussion in statistical gate sizing has dealt with *statistical delay*. However, a natural question is whether the statistical power should also be considered — is it beneficial to perform gate sizing with a statistical power objective?

The statistical power can be represented as the sum of the statistical leakage power and the statistical dynamic power random variables:

$$\mathbf{P} = \mathbf{P}_l + \mathbf{P}_d. \tag{6.11}$$

As Figure 6.1 indicates, the variations in the leakage power dominates the total variation in power, and this will be the focus of this section. The leakage power random variable is generally modeled as log-normal, as in [130], and can be expressed as a function of the gate level leakage as:

$$\mathbf{P}_l = \sum_{g \in \mathcal{G}} \mathrm{p}(g,\omega) \mathrm{e}^{-\gamma_g \Delta \mathbf{V}_{\mathrm{t}g}} \mathrm{e}^{-\eta_g \Delta \mathbf{L}_g} \tag{6.12}$$

where

- $\mathrm{p}(g,\omega)$ is the nominal power.
- $\Delta \mathbf{L}_g$ is a zero-mean random variable that describes the gate length variations for gate g.
- $\Delta \mathbf{V}_{\mathrm{t}g}$ is a zero-mean random variable that describes the threshold voltage variations for gate g.
- γ_g and η_g are fitting coefficients.

When the $\Delta \mathbf{L}_g$ and $\Delta \mathbf{V}_{\mathrm{t}g}$ are Gaussian, e.g., normal, the expression (6.12) is log-normal. Generally, $\Delta \mathbf{V}_{\mathrm{t}g}$ and $\Delta \mathbf{L}_g$ contain parts related to the intra-die variations ($\Delta \mathbf{V}_{\mathrm{twid},g}$ and $\Delta \mathbf{L}_{\mathrm{wid},g}$) and inter-die variations ($\Delta \mathbf{V}_{\mathrm{tdtd}}$ and $\Delta \mathbf{L}_{\mathrm{dtd}}$). Note that there is no g subscript for the inter-die variations — all gates on the same die see the same variation. The random variable (6.12) is difficult to work with directly. Generally, the random variable must be interpreted; for example, the *mean* leakage power of (6.12) can be expressed as:

$$\mathbf{E}[\mathbf{P}_l] = \sum_{g \in \mathcal{G}} \mathrm{p}(g,\omega) \mathrm{e}^{\gamma_g^2 \sigma^2_{\Delta \mathbf{V}_{\mathrm{t}g}}/2} \mathrm{e}^{\eta_g^2 \sigma^2_{\Delta \mathbf{L}_g}/2} \tag{6.13}$$

The percentile, or quantile, of (6.12) is generally difficult to express in closed form. As an approximation, the mean + 3σ measure is often used:

$$\mathbf{E}[\mathbf{P}_l] + 3(\mathbf{E}[\mathbf{P}_l^2] - \mathbf{E}[\mathbf{P}_l]^2) \tag{6.14}$$

with

$$\mathbf{E}[\mathbf{P}_l^2] = \sum_{g\in\mathcal{G}}\sum_{g'\in\mathcal{G}} \mathrm{p}(g,\omega)\mathrm{p}(g',\omega)\mathbf{E}[\mathrm{e}^{-(\gamma_g \Delta \mathbf{V}_{\mathrm{t}g} + \gamma_{g'}\Delta \mathbf{V}_{\mathrm{t}g'})}\mathrm{e}^{-(\eta_g \Delta \mathbf{L}_g + \eta_{g'}\Delta \mathbf{L}_{g'})}]. \tag{6.15}$$

This function is non-linear, and requires the computation of the covariance terms.

In [42], the benefits of using a statistical power objective are evaluated. When the mean power is used to measure statistical power, the improvements were small, on the order of 1%. For the mean $+ 3\sigma$ measure, the improvements can be noticeable (>5%). However, the following linear proxy measure can be used in place of the more complex non-linear mean $+ 3\sigma$ measure:

$$[\text{proxy}] = \sum_{g\in\mathcal{G}} \mathrm{p}(g,\omega)\mathrm{e}^{\gamma_g \sigma_{\Delta \mathbf{V}_{\mathrm{tdtd}}} + \eta_g \sigma_{\Delta \mathbf{L}_{\mathrm{dtd}}}}\mathrm{e}^{\sqrt{2}(\gamma_g \sigma_{\Delta \mathbf{V}_{\mathrm{t}g,\mathrm{wid}}} + \eta_g \sigma_{\Delta \mathbf{L}_{g,\mathrm{wid}}})/2}. \tag{6.16}$$

Using this measure as an objective can be used to optimize to within 5% of the mean $+3\sigma$ optimum.

6.7 Statistical Delay Considerations

There are many papers that show the benefits of using statistical delay [36, 104, 151]. For example, [21] cites a 20–30% power improvement from using statistical delay. Unlike the case of statistical power objectives, a statistical delay objective or constraint is widely understood to provide a improvement.

It is important to recognize the comparison point for these improvements. In [36] a 19% improvement is found over worst-case design with a $6 - \sigma$ guardband; [104] provides a 31% static power improvement over a 100%-yield worst-case scenario. These comparisons with overly conservative worst-case scenarios may exaggerate the benefits of statistical optimization, but they show the benefits of preventing over-design.

References [21, 151, 163] provide comparisons between full statistical optimization, and a design method called global guardbanding (see [21]). In global-guardbanding, the optimization is performed using nominal design methods, but the timing constraint is checked using an SSTA. The SSTA is used to adjust the timing constraint and to create a design that meets the statistical delay constraint exactly. This reduces the additional costs associated with over-designing, and the infeasibility associated with under-designing.

Comparisons between full statistical optimization and global guard-banding are mixed. In [151], an improvement between 15% and 35% is shown over a sensitivity based method that employs an SSTA to verify the timing constraint. However, in similar experiments in [21, 163], a 6% improvement is shown. This indicates that a majority of the improvement comes from reducing the pessimism; however, there is still a small but significant improvement that can be gained from a full statistical optimization.

7 Conclusion

Gate sizing and threshold voltage assignment are versatile methods used to optimize the power and timing of a design. The research over the past three decades has produced methods such as the greedy method, slack and delay budgeting, continuous sizing based, dynamic programming, Lagrangian relaxation, and linear programming based assignment methods. This survey has also covered the background needed to understand gate sizing and threshold voltage assignment via sections on static timing analysis, and on the mechanisms behind gate sizing and threshold voltage assignment.

The comparisons given in Section 5 point to varying conclusions. The need and potential benefits for new gate sizing and threshold voltage assignment methods is not clear. To advance gate sizing and threshold voltage assignment, it is important to quantify the current best practices for gate sizing, and its potential benefits. The most important step in proceeding is to understand the current state-of-the-art.

The work on Eyecharts in [65] are a good first step toward this end, as they make comparisons in several different contexts. The optima for these benchmarks are known and they give a broad overview of how far the methods are from optimal. However these benchmarks are artificial, and their correlation to real designs must be justified.

The largest difficulty in comparing existing methods is a lack of a common framework for the algorithms. The results provided in literature are difficult to compare, as the experimental setups are rarely similar. However, an avenue for improvement might be found by examining the areas of placement and routing. The ISPD contests in placement [112] and routing [113] provided benchmarks and a common framework to compare different methods. The results from the contests provided insight into the state of placement and routing, along with its best practices. It also motivated new research on that combined the best practices found in the contest. toward this end, the authors, along with researchers at UCSD, have released a benchmarking website at http://vlsicad.ucsd.edu/SIZING, that can be used to develop new algorithms, and test existing ones.

Another area for research is in metrics to measure how difficult a benchmark is, or what kind of method would perform well. As gate sizing and threshold voltage assignment is an NP-hard combinatorial problem, it may be difficult for one method to perform well across all benchmarks. Thus, it may be worthwhile to consider metrics that can determine whether one method would work better than another, and even give an estimate on the amount of improvement that can be gained by optimization.

There are also new challenges due to the increased variability. Statistical gate sizing methods have been studied for over decade, and it is time to quantify the benefits and drawbacks of statistical gate sizing, and find scalable methods for large scale design. The increased variability has also motivated questions into incremental design. How can designs be optimized incrementally to account for changes in the manufacturing process? These are areas to consider for the coming decade.

Acknowledgments

We would like to acknowledge Prof. Andrew Kahng and Dr. Kwangok Jeong for their help with the post-layout timing, Amarnath Kasibhatla for his support with the eyecharts, and Santiago Mok for his support with the UCLA Timer. We are also grateful for support from the National Science Foundation Award 0811832.

References

[1] A. Abou-Seido, B. Nowak, and C. Chu, "Fitted Elmore delay: a simple and accurate interconnect delay model," *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 12, no. 7, pp. 691–696, 2004.

[2] H. Abrishami, J. Lou, J. Q. J. Froessl, and M. Pedram, "Post sign-off leakage power optimization," in *Proceedings of Design Automation Conference*, 2011.

[3] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda, "Statistical delay computation considering spatial correlations," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 271–276, 2003.

[4] A. Agarwal, K. Chopra, and D. Blaauw, "Statistical timing based optimization using gate sizing," in *Proceedings of Design, Automation and Test in Europe*, pp. 400–405, 2005.

[5] C. Amin, N. Menezes, K. Killpack, F. Dartu, U. Choudhury, N. Hakim, and Y. Ismail, "Statistical static timing analysis: How simple can we get?," in *Proceedings of Design Automation Conference*, pp. 652–657, 2005.

[6] Available on http://www.opencores.org.

[7] X. Bai, C. Visweswariah, P. Strenski, and D. Hathaway, "Uncertainty-aware circuit optimization," in *Proceedings of Design Automation Conference*, pp. 58–63, 2002.

[8] D. K. Beece, J. Xiong, C. Visweswariah, V. Zolotov, and Y. Liu, "Transistor sizing of custom high-performance digital circuits with parametric yield considerations," in *Proceedings of Design Automation Conference*, pp. 781–786, 2010.

[9] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 545–550, November 1998.
[10] R. Bellman, *Dynamic Programming.* 1957.
[11] M. R. C. M. Berkelaar and J. 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.
[12] R. Berridge, R. Averill, A. Barish, M. Bowen, P. Camporese, J. DiLullo, P. Dudley, J. Keinert, D. Lewis, and R. Morel et al., "IBM POWER6 microprocessor physical design and design methodology," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 685–714, 2007.
[13] D. Bertsekas, *Dynamic Programming and Optimal Control*, vol. I. Athena Scientific, 2005.
[14] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer Verlag, 2009.
[15] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: From basic principles to state of the art," *IEEE Transactions on Computer-Aided Design*, vol. 27, no. 4, pp. 589–607, 2008.
[16] K. Boese, A. Kahng, B. Mccoy, and G. Robins, "Fidelity and near-optimality of Elmore-based routing constructions," in *Proceedings of International Conference on Computer Design*, pp. 81–84, 1993.
[17] S. Boyd, S. Kim, D. Patil, and M. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, no. 6, p. 899, 2005.
[18] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.
[19] R. Brawhear, N. Menezes, C. Oh, L. Pillage, and M. Mercer, "Predicting circuit performance using circuit-level statistical timing analysis," in *Proceedings of the European Conference on Design Automation (EDAC). European Test Conference (ETC). The European Event in ASIC Design (EUROASIC)*, pp. 332–337, 1994.
[20] F. Brglez, D. Bryan, and K. Kozminski, "Combinatorial profiles of sequential benchmark circuits," in *Proceedings of International Symposium on Circuits and Systems*, pp. 1929–1934, May 1989.
[21] 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 *Proceedings of Design Automation Conference*, pp. 238–243, 2007.
[22] Cadence, "Encounter v. 10.11," Avalilable on http://www.cadence.com.
[23] M. Celik, L. Pileggi, and A. Odabasioglu, *IC Interconnect Analysis.* Springer Netherlands, 2002.
[24] P. Chan, "Algorithms for library-specific sizing of combinational logic," in *Proceedings of Design Automation Conference*, pp. 353–356, 1990.
[25] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *Proceedings of International Conference on Computer-Aided Design*, p. 621, 2003.
[26] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah, "Parameterized block-based statistical timing analysis with non-Gaussian parameters,

nonlinear delay functions," in *Proceedings of Design Automation Conference*, pp. 71–76, 2005.

[27] C. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 7, pp. 1014–1025, 1999.

[28] C. Chen and M. Sarrafzadeh, "Power reduction by simultaneous voltage scaling and gate sizing," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 333–338, 2000.

[29] C. Chen and M. Sarrafzadeh, "Simultaneous voltage scaling and gate sizing for low-power design," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 6, pp. 400–408, 2002.

[30] D.-S. Chen and M. Sarrafzadeh, "An exact algorithm for low power library-specific gate re-sizing," in *Proceedings of Design Automation Conference*, pp. 783–788, 1996.

[31] H. Chen and S. Kang, "iCOACH: A circuit optimization aid for CMOS high-performance circuits," *Integration, the VLSI journal*, vol. 10, no. 2, pp. 185–212, 1991.

[32] L. Cheng, J. Xiong, and L. He, "Non-linear statistical static timing analysis for non-Gaussian variation sources," in *Proceedings of Design Automation Conference*, pp. 250–255, 2007.

[33] D. Chinnery and K. Keutzer, *Closing the Power Gap between ASIC & Custom: Tools and Techniques for Low Power Design.* New York Inc.: Springer-Verlag, 2007.

[34] D. G. Chinnery and K. Keutzer, "Linear programming for sizing, vth and vdd assignment," in *Proceedings of International Conference on Low Power Electronics and Design*, pp. 149–154, 2005.

[35] B. Choi and D. Walker, "Timing analysis of combinational circuits including capacitive coupling and statistical process variation," in *Proceedings of VLSI Test Symposium*, pp. 49–54, 2000.

[36] S. H. Choi, B. C. Paul, and K. Roy, "Novel sizing algorithm for yield improvement under process variation in nanometer technology," pp. 454–459, 2004.

[37] 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 *Proceedings of International Conference on Computer-Aided Design*, pp. 1023–1028, 2005.

[38] K. Chopra, B. Zhai, D. Blaauw, and D. Sylvester, "A new statistical max operation for propagating skewness in statistical timing analysis," in *Proceedings of International Conference on Computer-Aided Design*, pp. 237–243, 2006.

[39] H. Chou, Y. Wang, and C. Chen, "Fast and effective gate-sizing with multiple-vt assignment using generalized lagrangian relaxation," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 381–386, 2005.

[40] W. Chuang, S. Sapatnekar, and I. Hajj, "Timing and area optimization for standard-cell VLSI circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, p. 308, 1995.

[41] C. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, no. 2, pp. 145–162, 1961.

[42] J. Cong, P. Gupta, and J. Lee, “Evaluating statistical power optimization,” *IEEE Transactions on Computer-Aided Design*, vol. 29, no. 11, pp. 1750–1762, 2010.

[43] J. Cong, J. Lee, and L. Vandenberghe, “Robust gate sizing via mean excess delay minimization,” in *Proceedings of International Conference on Physical Design*, pp. 10–14, 2008.

[44] A. Conn, P. Coulman, R. Haring, G. M. C. Visweswariah, and C. Wu, “JiffyTune: Circuit optimization using time-domain sensitivities,” *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 12, pp. 1292–1309, 1998.

[45] A. Conn, R. Haring, C. Visweswariah, P. Coulman, and G. Morrill, “Optimization of custom MOS circuits by transistor sizing,” in *International Conference on Computer-Aided Design*, p. 174, 1996.

[46] O. Coudert, “Gate sizing: A general purpose optimization approach,” in *Proceedings of Design, Automation and Test in Europe*, p. 214, 1996.

[47] O. Coudert, “Gate sizing for constrained delay/power/area optimization,” *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 5, pp. 465–472, December 1997.

[48] O. Coudert, R. Haddad, and S. Manne, “New algorithms for gate sizing: A comparative study,” in *Proceedings of Design Automation Conference*, pp. 734–739, 1996.

[49] Z. Dai and K. Asada, “MOSIZ: A two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates,” in *Proceedings of Custom Integrated Circuits Conference*, pp. 17–3, 1989.

[50] A. Davoodi and A. Srivastava, “Variability driven gate sizing for binning yield optimization,” in *Proceedings of Design Automation Conference*, pp. 959–964, 2006.

[51] A. Davoodi and A. Srivastava, “Variability driven gate sizing for binning yield optimization,” *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 16, no. 6, pp. 683–692, 2008.

[52] M. Degrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. Goffart, E. Vittoz, S. Cserveny, C. Meixenberger, and G. Van der Stappen et al., “IDAC: An interactive design tool for analog CMOS circuits,” *IEEE Journal of Solid-State Circuits*, vol. 22, no. 6, pp. 1106–1116, 1987.

[53] 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 *Proceedings of International Conference on Computer Design*, pp. 143–148, October 1997.

[54] A. Dharchoudhury and S. Kang, “Worst-case analysis and optimization of VLSI circuit performances,” *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 4, pp. 481–492, 1995.

[55] F. El-Turky and E. Perry, “Blades: An artificial intelligence approach to analog circuit design,” *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 680–692, June 1989.

[56] W. Elmore, “The transient response of damped linear networks with particular regard to wideband amplifiers,” *Journal of Applied Physics*, vol. 19, pp. 55–63, 1948.

[57] P. Feldmann and S. Abbaspour, "Towards a more physical approach to gate modeling for timing, noise, and power," in *Proceedings of Design Automation Conference*, pp. 453–455, 2008.

[58] P. Feldmann and S. Director, "Integrated circuit quality optimization using surface integrals," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1868–1879, December 1993.

[59] P. Feldmann and R. Freund, "Efficient linear circuit analysis by pade approximation via the lanczos process," *IEEE Transactions on Computer-Aided Design*, vol. 14, pp. 639–649, May 1995.

[60] J. Fishburn and A. Dunlop, "TILOS: A posynomial approach to transistor sizing," in *Proceedings of International Conference on Computer-Aided Design*, 1985.

[61] C. Forzan and D. Pandini, "Statistical static timing analysis: A survey," *Integration, the VLSI Journal*, vol. 42, no. 3, pp. 409–435, 2009.

[62] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, "Timing yield estimation from static timing analysis," in *International Symposium on Quality Electronic Design*, pp. 437–442, 2001.

[63] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.

[64] G. Gielen, H. Walscharts, and W. Sansen, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 707–713, 1990.

[65] P. Gupta, A. Kahng, A. Kasibhatla, and P. Sharma, "Eyecharts: Constructive benchmarking of gate sizing heuristics," in *Proceedings of Design Automation Conference*, 2010.

[66] P. Gupta, A. Kahng, P. Sharma, and D. Sylvester, "Gate-length biasing for runtime-leakage control," *IEEE Transactions on Computer-Aided Design*, vol. 25, no. 8, pp. 1475–1485, 2006.

[67] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester, "Selective gate-length biasing for cost-effective runtime leakage control," in *Proceedings of Design Automation Conference*, pp. 327–330, 2004.

[68] R. Gupta, B. Tutuianu, and L. Pileggi, "The Elmore delay as a bound for RC trees with generalized input signals," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 1, pp. 95–104, 1997.

[69] M. Guthaus, N. Venkateswaran, C. Visweswariah, and V. Zolotov, "Gate sizing using incremental parameterized statistical timing analysis," in *Proceedings of International Conference on Computer-Aided Design*, pp. 1029–1036, 2005.

[70] R. Haddad, L. van Ginneken, and N. Shenoy, "Discrete drive selection for continuous sizing," in *Proceedings of International Conference on Computer Design*, pp. 110–115, October 1997.

[71] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Design and Test*, vol. 16, pp. 72–80, July–September 1999.

[72] R. Harjani, R. Rutenbar, and L. Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 12, pp. 1247–1266, 1989.

[73] M. Hashimoto and H. Onodera, "Post-layout transistor sizing for power reduction in cell-based design," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 359–365, 2001.

[74] S. Held, "Gate sizing for large cell-based designs," in *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium), pp. 827–832, 2009.

[75] M. Hershenson, S. Boyd, and T. Lee, "Optimal design of a CMOS op-amp via geometric programming," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 1, pp. 1–21, 2001.

[76] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," in *Proceedings of Design Automation Conference*, pp. 847–852, 2007.

[77] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell-library-based designs," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 6, pp. 818–825, 2009.

[78] Y. Huang, J. Hu, and W. Shi, "Lagrangian relaxation for gate implementation selection," in *Proceedings of International Conference on Physical Design*, pp. 167–174, 2011.

[79] International Technology Roadmap for Semiconductors — Design Available on http://www.itrs.net.

[80] K. Jeong and A. Kahng, "Methodology from chaos in ic implementation," in *Proceedings of International Conference on Quality Electronic Design*, pp. 885–892, 2010.

[81] A. Kahng, B. Liu, and X. Xu, "Constructing current-based gate models based on existing timing library," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 37–42, 2006.

[82] T. Karnik, Y. Ye, J. Tschanz, L. 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 *Proceedings of Design Automation Conference*, pp. 486–491, 2002.

[83] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, "A new class of convex functions for delay modeling and its application to the transistor sizing problem [cmos gates]," *IEEE Transactions on Computer-Aided Design*, vol. 19, no. 7, pp. 779–788, 2000.

[84] I. Keller, K. Tam, and V. Kariat, "Challenges in gate level modeling for delay and SI at 65 nm and below," in *Proceedings of Design Automation Conference*, pp. 468–473, 2008.

[85] V. Khandelwal and A. Srivastava, "A general framework for accurate statistical timing analysis considering correlations," in *Proceedings of Design Automation Conference*, pp. 89–94, 2005.

[86] T. I. Kirkpatrick and N. R. Clark, "Pert as an aid to logic design," *IBM Journal of Research and Development*, vol. 10, pp. 135–141, March 1966.

[87] H. Koh, C. Sequin, and P. Gray, "OPASYN: A compiler for CMOS operational amplifiers," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 2, pp. 113–125, 1990.

[88] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley, "Maelstrom: Efficient simulation-based synthesis for custom analog cells," in *Proceedings of Design Automation Conference*, pp. 945–950, 1999.

[89] J. Lee and P. Gupta, "Incremental gate sizing for late process changes," in *Proceedings of International Conference on Computer Design*, pp. 215–221, October 2010.

[90] J.-F. Lee and D. T. Tang, "An algorithm for incremental timing analysis," in *Proceedings of Design Automation Conference*, pp. 696–701, 1995.

[91] C. Lemaréchal, "Lagrangian relaxation," *Computational Combinatorial Optimization*, pp. 112–156, 2001.

[92] W. Li, "Strongly np-hard discrete gate sizing problems," in *Proceedings of International Conference on Computer Design*, pp. 468–471, October 1993.

[93] X. Li, J. Le, and L. Pileggi, "Statistical performance modeling and optimization," *Foundations and Trends® in Electronic Design Automation*, vol. 1, no. 4, pp. 331–480, 2006.

[94] X. Li, J. Le, and L. T. Pileggi, *Statistical Performance Modeling and Optimization*. Hanover, MA, USA: Now Publishers Inc., 2007.

[95] C. Liao and S. Hu, "Approximation scheme for restricted discrete gate sizing targeting delay minimization," *Journal of Combinatorial Optimization*, vol. 21, no. 4, pp. 497–510, 2011.

[96] H.-R. Lin and T.-T. Hwang, "Power reduction by gate sizing with path-oriented slack calculation," in *Proceedings of Asia and South Pacific Design Automation Conference*, 1995.

[97] R. Lin and M. Wu, "A new statistical approach to timing analysis of vlsi circuits," in *Proceedings of International Conference on VLSI Design*, pp. 507–513, 1998.

[98] S. Lin, M. Marek-Sadowska, and E. Kuh, "Delay and area optimization in standard-cell design," *Proceedings of Design Automation Conference*, pp. 349–352, June 1990.

[99] J.-J. Liou, A. Krstic, L.-C. Wang, and K.-T. Cheng, "False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation," in *Proceedings of Design Automation Conference*, pp. 566–569, 2002.

[100] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," in *Proceedings of International Conference on Physical Design*, pp. 27–34, 2009.

[101] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," *IEEE Transactions on Computer-Aided Design*, vol. 29, pp. 223–234, February 2010.

[102] R. Macys and S. McCormick, "A new algorithm for computing the effective capacitance in deep sub-micron circuits," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 313–316, 1998.

[103] H. Mangassarian and M. Anis, "On statistical timing analysis with inter-and intra-die variations," in *Proceedings of the Conference on Design, Automation and Test in Europe-Volume 1*, pp. 132–137, 2005.

[104] M. Mani, A. Devgan, and M. Orshansky, "An efficient algorithm for statistical minimization of total power under timing yield constraints," in *Proceedings of Design Automation Conference*, pp. 309–314, 2005.

[105] M. Mani, A. Devgan, M. Orshansky, and Y. Zhan, "A statistical algorithm for power- and timing-limited parametric yield optimization of large integrated circuits," *IEEE Transactions on Computer-Aided Design*, vol. 26, pp. 1790–1802, October 2007.

[106] M. Mani and M. Orshansky, "A new statistical optimization algorithm for gate sizing," in *Proceedings of IEEE International Conference on Computer Design*, pp. 272–277, 2004.

[107] T. Massier, H. Graeb, and U. Schlichtmann, "The sizing rules method for CMOS and bipolar analog integrated circuit synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 27, no. 12, pp. 2209–2222, 2008.

[108] T. McConaghy, P. Palmers, G. Gielen, and M. Steyaert, "Automated extraction of expert knowledge in analog topology selection and sizing," 2008.

[109] S. 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.

[110] S. Mok, "Propagation delay approximation considering effective capacitance and slew degradation," Techical Report, UCLA, Available on http://nanocad.ee.ucla.edu/pub/Main/Publications/MSTR4_paper.pdf, 2011.

[111] F. N. Najm, "On the need for statistical timing analysis," in *Proceedings of Design Automation Conference*, pp. 764–765, 2005.

[112] G. Nam, "Ispd 2006 placement contest: Benchmark suite and results," in *Proceedings of International Conference on Physical Design*, pp. 167–167, 2006.

[113] G. Nam, M. Yildiz, D. Pan, and P. Madden, "Ispd placement contest updates and ispd 2007 global routing contest," in *Proceedings of International Conference on Physical Design*, pp. 167–167, 2007.

[114] Nangate Open Cell Library v1.3 Available on http://www.si2.org/openeda.si2.org/projects/nangatelib.

[115] S. Narendra, "Challenges and design choices in nanoscale CMOS," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 1, no. 1, pp. 7–49, 2005.

[116] S. Nassif, A. Strojwas, and S. Director, "A methodology for worst-case analysis of integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 5, no. 1, pp. 104–113, January 1986.

[117] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization," in *Proceedings of International Conference on Low Power Electronics and Design*, pp. 158–163, 2003.

[118] W. Nye, D. Riley, A. Sangiovanni-Vincentelli, and A. Tits, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 4, pp. 501–519, 1988.

[119] E. Ochotta, R. Rutenbar, and L. Carley, "Synthesis of high-performance analog circuits in ASTRX/OBLX," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 3, pp. 273–294, 1996.

[120] M. Orshansky and K. Keutzer, “A general probabilistic framework for worst case timing analysis,” in *Proceedings of Design Automation Conference*, pp. 556–561, 2002.

[121] M. Ozdal, S. Burns, and J. Hu, “Gate sizing and device technology selection algorithms for high-performance industrial designs,” in *Proceedings of International Conference on Computer-Aided Design*, pp. 724–731, November 2011.

[122] P. Pant, V. De, and A. Chatterjee, “Simultaneous power supply, threshold voltage, and transistor size optimization for low-power operation of cmos circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 6, no. 4, pp. 538–545, 1998.

[123] P. Pant, R. Roy, and A. Chattejee, “Dual-threshold voltage assignment with transistor sizing for low power cmos circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 9, no. 2, pp. 390–394, 2001.

[124] D. Patil, S. Yun, S. Kim, A. Cheung, M. Horowitz, and S. Boyd, “A new method for design of robust digital circuits,” in *Proceedings of International Conference on Quality Electronic Design*, pp. 676–681, 2005.

[125] M. Pelgrom, A. Duinmaijer, and A. Welbers, “Matching properties of mos transistors,” *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, 1989.

[126] R. Phelps, M. Krasnicki, R. Rutenbar, L. Carley, and J. Hellums, “ANACONDA: Robust synthesis of analog circuits via stochastic pattern search,” in *Proceedings of Custom Integrated Circuits*, pp. 567–570, 1999.

[127] L. Pillage and R. Rohrer, “Asymptotic waveform evaluation for timing analysis,” *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 4, pp. 352–366, 1990.

[128] J. Qian, S. Pullela, and L. Pillage, “Modeling the effective capacitance for the RC interconnect of CMOS gates,” *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 12, pp. 1526–1535, 1994.

[129] A. Ramalingam, A. Singh, S. Nassif, G. Nam, M. Orshansky, and D. Pan, “An accurate sparse-matrix based framework for statistical static timing analysis,” *Integration, the VLSI Journal*, 2011.

[130] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester, “Statistical analysis of subthreshold leakage current for VLSI circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 12, no. 2, pp. 131–139, 2004.

[131] D. Rautenbach and C. Szegedy, “A class of problems for which cyclic relaxation converges linearly,” *Computational Optimization and Applications*, vol. 41, no. 1, pp. 53–60, 2008.

[132] S. Roy, W. C. aad C. Chen, and Y. Hu, “Numerically convex forms and their application in gate sizing,” *IEEE Transactions on Computer-Aided Design*, vol. 26, no. 9, pp. 1637–1647, 2007.

[133] S. Roy and W. Chen, “Convexfit: An optimal minimum-error convex fitting and smoothing algorithm with application to gate-sizing,” in *Proceedings of International Conference on Computer-Aided Design*, pp. 196–203, 2005.

[134] S. Roy, Y. H. Hu, C. C.-P. Chen, S.-P. Hung, T.-Y. Chiang, and J.-G. Tseng, “An optimal algorithm for sizing sequential circuits for industrial library based

designs," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 148–151, 2008.

[135] A. Ruehli, P. Wolff, and G. Goertzel, "Analytical power/timing optimization technique for digital system," in *Proceedings of Design Automation Conference*, pp. 142–146, 1977.

[136] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 584–594, April 1990.

[137] S. Sapatnekar, *Timing*. Springer, Netherlands, 2004.

[138] S. Sapatnekar and W. Chuang, "Power vs. delay in gate sizing: Conflicting objectives?," in *Proceedings of International Conference on Computer-Aided Design*, pp. 463–466, November 1995.

[139] N. Satish, K. Ravindran, M. Moskewicz, D. Chinnery, and K. Keutzer, "Evaluating the effectiveness of statistical gate sizing for power optimization," Technical Report, University of California at Berkeley, ERL Memorandum M05/28, August 2005.

[140] C. Sechen and H. Tennakoon, "Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step," in *Proceedings of International Conference on Computer-Aided Design*, pp. 395–402, 2002.

[141] 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 *Proceedings of International Conference on Computer-Aided Design*, pp. 705–712, 2005.

[142] B. Sheu, D. Scharfetter, P. Ko, and M. Jeng, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 4, pp. 558–566, 1987.

[143] J. Shyu, A. Sangiovanni-Vincentelli, J. Fishburn, and A. Dunlop, "Optimization-based transistor sizing," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 400–409, 1988.

[144] L. Silveira, M. Kamon, J. White, and I. Elfadel, "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits," in *Proceedings of International Conference on Computer-Aided Design*, p. 288, 1996.

[145] J. Singh, Z.-Q. Luo, and S. Sapatnekar, "A geometric programming-based worst case gate sizing method incorporating spatial correlation," *IEEE Transactions on Computer-Aided Design*, vol. 27, pp. 295–308, February 2008.

[146] J. Singh, V. Nookala, Z. Luo, and S. Sapatnekar, "Robust gate sizing by geometric programming," in *Proceedings of Design Automation Conference*, pp. 315–320, 2005.

[147] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis," in *Proceedings of Design Automation Conference*, pp. 155–160, 2006.

[148] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw, "Duet: An accurate leakage estimation and optimization tool for dual-Vt circuits," *IEEE Transactions on Very Large Scale Integration (VLSI)*, vol. 10, no. 2, pp. 79–90, 2002.

[149] 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 *Proceedings of Design Automation Conference*, pp. 436–441, 1999.

[150] A. Srivastava, D. Sylvester, and D. Blaauw, “Power minimization using simultaneous gate sizing, dual-vdd and dual-vth assignment,” in *Proceedings of Design Automation Conference*, pp. 783–787, 2004.

[151] A. Srivastava, D. Sylvester, and D. Blaauw, “Statistical optimization of leakage power considering process variations using dual-Vth and sizing,” in *Proceedings of Design Automation Conference*, pp. 773–778, 2004.

[152] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power.* Springer Verlag, 2005.

[153] V. Stojanovic, D. Markovic, B. Nikolic, M. Horowitz, and R. Brodersen, “Energy–delay tradeoffs in combinational logic using gate sizing and supply voltage optimization,” in *Proceedings of European Solid-State Circuits Conference (ESSCIRC)*, pp. 211–214, 2005.

[154] I. Sutherland and R. Sproull, “Logical effort: Designing for speed on the back of an envelope,” in *Proceedings of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI*, pp. 1–16, 1991.

[155] I. Sutherland, R. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits.* Morgan Kaufmann, 1999.

[156] Synopsys, “Liberty open source library modeling,” Web resource, http://www.opensourceliberty.org.

[157] Synopsys, “NanoTime,” http://www.synopsys.org.

[158] Y. Tamiya, Y. Matsunaga, and M. Fujita, “Lp based cell selection with constraints of timing, area, and power consumption,” in *Proceedings of International Conference on Computer-Aided Design*, pp. 378–381, 1994.

[159] T. Tokuda, J. Korematsu, O. Tomisawa, S. Asai, I. Ohkura, and T. Enomoto, “A hierarchical standard cell approach for custom VLSI design,” *IEEE Transactions on Computer-Aided Design*, vol. 3, no. 3, pp. 172–177, 1984.

[160] R. Trihy, “Addressing library creation challenges from recent liberty extensions,” in *Proceedings of Design Automation Conference*, pp. 474–479, 2008.

[161] S. Tsukiyama, M. Tanaka, and M. Fukui, “A statistical static timing analysis considering correlations between delays,” in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 353–358, 2001.

[162] P. Tuohy, A. Gribben, A. Walton, , and J. Robertson, “Realistic worst-case parameters for circuit simulation,” in *Proceedings of Communications, Speech and Vision*, pp. 137–140, 1987.

[163] V. Veetil, D. Sylvester, and D. Blaauw, “A lower bound computation method for evaluation of statistical design techniques,” in *Proceedings of International Conference on Computer-Aided Design*, pp. 562–569, 2010.

[164] C. Visweswariah, “Death, taxes and failing chips,” in *Proceedings of Design Automation Conference*, pp. 343–347, 2003.

[165] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, “First-order incremental block-based statistical timing analysis,” in *Proceedings of Design Automation Conference*, pp. 331–336, 2004.

[166] J. Wang, D. Das, and H. Zhou, "Gate sizing by lagrangian relaxation revisited," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 7, pp. 1071–1084, 2009.

[167] Q. Wang and S. Vrudhula, "Static power optimization of deep submicron cmos circuits for dual vt technology," in *Proceedings of International Conference on Computer-Aided Design*, pp. 490–496, 1998.

[168] L. Wei, K. Roy, and C. Koh, "Power minimization by simultaneous dual-Vth assignment and gate-sizing," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 413–416, 2000.

[169] T. Wu and A. Davoodi, "PaRS: Parallel and near-optimal grid-based cell sizing for library-based design," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 11, pp. 1666–1678, 2009.

[170] T. Xiao and M. Marek-Sadowska, "Gate sizing to eliminate crosstalk induced timing violation," in *Proceedings of the International Conference on Computer Design*, p. 0186, 2001.

[171] Y. Zhan, A. Strojwas, X. Li, L. Pileggi, D. Newmark, and M. Sharma, "Correlation-aware statistical timing analysis with non-Gaussian delay distributions," in *Proceedings of Design Automation Conference*, pp. 77–82, 2005.

[172] L. Zhang, W. Chen, Y. Hu, J. Gubner, and C. Chen, "Correlation-preserved non-Gaussian statistical timing analysis with quadratic timing model," in *Proceedings of Design Automation Conference*, pp. 83–88, 2005.

[173] L. Zhang, Y. Hu, and C. Chen, "Statistical timing analysis with path reconvergence and spatial correlations," in *Proceedings of Design, Automation and Test in Europe*, p. 112, 2006.

[174] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Transactions on Computer-Aided Design*, vol. 25, no. 1, pp. 155–166, 2006.

[175] C. Zhuo, D. Blaauw, and D. Sylvester, "Variation-aware gate sizing and clustering for post-silicon optimized circuits," in *Proceedings of the International Conference on Low Power Electronics and Design*, pp. 105–110, 2008.