Macromodeling and Optimization of Digital MOS VLSI Circuits

by

Mark Douglas Matson

S.B., Massachusetts Institute of Technology (1979)
S.M., Massachusetts Institute of Technology (1981)
E.E., Massachusetts Institute of Technology (1981)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

at the

Massachusetts Institute of Technology
January, 1985

Signature of Author _____
Department of Electrical Engineering and Computer Science
January 24, 1985

Certified by _____
Lance A. Glasser
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Macromodeling and Optimization of Digital MOS VLSI Circuits

by

Mark Douglas Matson

Submitted to the
Department of Electrical Engineering and Computer Science
on January 24, 1985 in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy.

Abstract

Power consumption and signal delay are crucial to the design of high-performance VLSI circuits. This thesis presents CAD tools for modeling and optimizing digital MOS designs. The tools determine the transistor sizes that minimize circuit power consumption subject to constraints on signal path delays. Computational efficiency is obtained through macromodeling techniques and a specialized optimization algorithm. The macromodels are based on device equations, and encapsulate logic gate behavior in a set of simple yet accurate formulas. The optimization algorithm exploits properties of the digital MOS domain to convert the primal optimization problem into a dual form which is much easier to solve. The result is a pair of CAD tools that can optimize a circuit in roughly the amount of time needed to perform a transistor level simulation of the circuit.

Thesis Supervisor: Lance A. Glasser

Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

I am very grateful to my thesis advisor, Lance Glasser, for his patient support and encouragement throughout the course of this research. I am also pleased to acknowledge the assistance of my readers John Wyatt, Paul Penfield, Jr., and Ron Rivest. Chris Terman, Dimitri Bertsekas, and Rich Zippel read drafts of this thesis and gave valuable comments. Barbara Lory instructed me in the nuances of English grammar as I prepared this document. I am indebted to James Roberge and Rick Carley for teaching me circuit theory, and to Dimitri Bertsekas and Kevin Tsai for introducing me to the art and science of nonlinear optimization. Jack Hilibrand and Larry Rosenberg provided much technical advice and guidance. Jeff Fox, Rich Olsen, and Gerry Sussman were helpful in defining the thesis topic, while Dan Dobberpuhl and Mark Horowitz kindly shared their knowledge of MOS circuit design and CAD tools.

Thanks are also due to my fellow graduate students in this research group. I thank John Wroclawski, whose initial predictions concerning this thesis ("What we have here is the original Pandora's box.") proved to be excruciatingly accurate, for many enlightening discussions on what it means to transfer an algorithm from the realm of theory into a computer, as well as for nurturing our sometimes cooperative, sometimes capricious computer system. John Hoyte's single path optimizer was the precursor to this thesis; he gladly related his experiences to me. Steve McCormick supplied a good deal of help with the text formatter used to generate this document, and Bob Armstrong wrote the graphics editor that created the figures. I am appreciative of my office mates, Charles Zukowski and Anne Park, for their insight and companionship.

I thank my parents and sisters, and my friends in the church, for their prayers and support.

This work was supported in part by an RCA fellowship, in part by the Defense Advanced Research Projects Agency of the Department of Defense under Contract No. N00014-80-C-0622, and in part by the Air Force Office of Sponsored Research under Contract No. F49620-84-C-0004.

To my Parents

To Him Who loves us and has loosed us from our sins by His blood, And made us a kingdom, priests to His God and Father, to Him be the glory and the might forever and ever. Amen.

Revelation 1:5-6

Table of Contents

Table of Figures

Table of Tables

CHAPTER ONE

Introduction

The design of a VLSI circuit is an enormous task. Sophisticated CAD tools are essential if designers are to take full advantage of the power offered by fabrication technology. This thesis describes tools for modeling digital MOS circuits and optimizing their performance. These tools find the transistor sizes that minimize power consumption subject to constraints on signal path delays. The principal advantage is an increase in designer productivity. At present, designers size transistors based on intuition and numerous SPICE simulations. This process is so time consuming—for both man and machine—that designers are hard-pressed to arrive at any circuit that meets delay specifications and can rarely afford the extra effort needed to minimize power consumption as well. This hinders not only the design of the circuit at hand, but also the comparison of alternate topologies for implementing functional blocks, as the performance benefits offered by different topologies cannot be truly ascertained unless the corresponding circuits have been optimized.

Another application is automatic module generation for silicon compilers. The module's transistors must be properly sized in order to meet system performance specifications, but it would be unthinkable to have a human perform the sizing. The task could involve thousands of transistors, making it too mundane and complicated. A special purpose optimizer can accomplish the chore far more efficiently.

1.1 Previous Work

Several authors have studied optimization work of this nature. General purpose optimization packages such as DELIGHT [1] and APLSTAP [2] perform much of the work in the optimization process. They iteratively improve the design solution as a designer would, but by employing nonlinear optimization algorithms, choose the next solution point more accurately and efficiently than a human could. The key advantage is that an optimal solution is reached. However the

optimization process tends to be computationally expensive for a number of reasons. First, since the optimization package is general purpose in nature, it cannot exploit properties of digital MOS logic and use algorithms which would be more problem specific and hence potentially faster. Second, because the optimization package is isolated from the circuit's data base, communicating solely via the simulator, there is no mechanism to access the circuit's structural description or to embed additional information in the data base which could assist the optimization. This hampers the application of more efficient algorithms that would require such provisions. Third, the circuit's signal path delays must be determined fairly accurately; this generally entails the use of a device level simulator such as SPICE, which is rather expensive computationally. The consequence of these three factors is that general purpose optimizers are typically restricted to circuits with at most about thirty design parameters.

In an effort to address larger designs, some researchers have investigated more specialized techniques [3]. By using a resistive model for transistors and neglecting the changes in a logic gate's input capacitance induced by sizing its transistors, these workers were able to simplify the optimization problem greatly. They reformulated the original problem, a minimization subject to nonlinear constraints, as an unconstrained minimization. This allows for much simpler optimization algorithms, leading to fast convergence times. Nonetheless, the simplifications needed to reformulate the problem seriously reduce the accuracy of both the power minimization and the satisfaction of the delay constraints, making the approach inappropriate for high-performance circuit design.

Other authors have aimed for fast computation times by simplifying the logic gate models and the optimization techniques. Examples are TV [4] and Andy [5]. Both tools use resistor models for transistors instead of the computationally expensive device level models. Heuristics, rather than nonlinear optimization algorithms, guide the sizing of transistors in critical paths. In particular, TV speeds up paths by widening the transistors of slow logic gates, while Andy uses a fixed sizing ratio from gate to gate when a chain drives a large capacitive load and then reduces the speed and power consumption of gates that are not on the critical paths. Although these approaches are computationally fast enough to be applied to large circuits, our problem domain requires more accuracy and efficiency. The resistor model is not accurate enough for high-performance design, and the heuristic sizing rules are inexact and somewhat inefficient. The heuristics are an attempt to decouple the sizing problem to the point where individual logic cells can be sized independently of the bulk of the circuit, and consequently these rules take limited

account of interactions among cells and signal paths. For instance, the heuristics ignore common subpaths. If two paths sharing a common portion need to be sped up, often it is best to speed up the common portion, since then both paths benefit. TV and Andy do not account for this. Moreover, both tools ignore the sensitivity of delay to power. When speeding up a chain of logic gates, additional power should be given to the gate which will provide the greatest increase in speed. This results in a chain that consumes the minimum power for the speed it offers. While the heuristics will mimic this rule under certain conditions, they will not do so in general. Furthermore, schemes that simultaneously consider all signal paths are potentially more efficient than those that apply heuristics to each path in sequence and then iterate. Owing to these deficiencies in the heuristic sizing rules, their final solution will be neither optimal, nor will it have necessarily been arrived at in a computationally efficient fashion.

1.2 Overview of the Thesis

This thesis presents a novel approach to the transistor sizing problem. We attack the competing needs for accuracy, computational speed, and a nearly optimal solution by combining the benefits of the previous approaches we examined. Like TV and Andy, we work at a higher level of abstraction than SPICE, transcending the details of actual transistor operation. However we acquire additional computational speed by modeling entire logic cells rather than just individual transistors. Like the general purpose optimizers, we employ nonlinear optimization techniques. This helps to assure that we reach an optimal solution in an efficient manner. However we exploit properties of digital MOS circuits and apply a specialized algorithm to the problem, yielding striking improvements in computational speed.

Chapter 2 discusses the theory and implementation of the logic cell macromodeler. We begin with a resistor-capacitor model and examine its limitations. We then develop a more elaborate model, one accounting for waveform shape effects. The theory gives us the form of the macromodel equations. In the implementation section we describe how the equations' parameters are determined with a sophisticated macromodeling support package. The resulting macromodels are accurate and computationally fast, and are pertinent to timing simulation and verification as well as optimization.

Chapter 3 presents the theory and implementation of the optimization algorithms. We explore several possible optimization methods. We choose a method particularly suited to our

problem, taking advantage of the properties of the digital MOS domain, and of our ability to create a circuit data base customized for the transistor sizing problem. Our approach, called duality, allows us to partition the problem into many simpler, smaller subproblems, and to transform the nonlinear delay constraints into a much simpler form. These techniques lead to very fast computation times. The chapter closes with a description of the software and several examples of the optimizer's performance.

Chapter 4 concludes the thesis with a summary of the contributions of this work and the perspectives gained from it. We also discuss several areas for future investigation, especially as related to automatic circuit design.

CHAPTER TWO

Macromodeling

2.1 Overview

This chapter discusses accurate, computationally efficient models for MOS logic gates. The models are well suited for simulation and optimization of high-performance VLSI circuits. The models are based on device equations, and acquire much of their accuracy through careful consideration of waveshape effects.

The significance of waveshape effects has been investigated by other workers. Crystal [6], a timing simulator, models transistors as resistors, but uses different values for transistor resistances depending on input waveform. While this leads to good accuracies (typically within 10% of SPICE predictions), the approach does have some limitations. For example, the tables of effective transistor resistances depend on a uniform trigger point voltage (the point on a logic gate's transfer curve where $v_{OUT} = v_{IN}$) and can produce substantial errors if this restriction is removed, for instance by varying beta ratios. Moreover the table interpolations can generate jagged delay functions; this can make the optimization task more difficult.

For these reasons we chose to base our models entirely on device equations. Horowitz [7] pursued a similar strategy in modeling the delay of a MOS inverter. He derived equations for the gate's response and then obtained estimates of parameters from the gate's drive curves (curves of v_{OUT} versus v_{IN} for different values of load current).

In this chapter we describe a more general and sophisticated model. We develop equations for power consumption, output waveform, and input capacitance of a general MOS logic gate. These equations are the culmination of several passes at modeling MOS logic gates; they are computationally fast because they incorporate the minimum level of detail needed to acquire the desired precision. To obtain high accuracy in the model, we wrote a macromodeling support

package to determine the equations' parameters. The package curve fits the model equations to SPICE simulation results and finds the parameter set which provides the greatest accuracy.

Section 2 describes the basic principles of the macromodeling approach. Section 3 presents models for MOS inverters. Our analysis opens with a treatment of the resistor-capacitor model. After studying its range of validity, we construct an improved model that accounts for waveform shape effects. The analysis is extended to more general logic gates in section 4. The theory gives us the form of the macromodel equations. In section 5 we discuss a macromodeling support package that solves for the equations' parameters.

2.2 Motivation and Intent

Circuit optimization is a computationally expensive process. It is an iterative procedure, requiring multiple simulations at each step to evaluate delays and their gradients. Moreover, high-performance circuit design requires fairly precise delay estimates, but using a device level simulator would be out of the question for all but small circuits.

Since it is too time consuming to compute circuit responses during the optimization, we instead pursue an approach where much of the work is performed in advance, prior to the optimization. We divide a large circuit into many small pieces. This partitioning is done such that the pieces have limited, well-understood interactions, while the elements inside the pieces have strong, complex interactions. Thus computing the interactions among elements within a piece would be very expensive, and it behooves us to characterize the behavior of the pieces beforehand to avoid having to compute the behavior during the optimization.

This approach is called macromodeling. In the digital MOS domain, candidates for pieces would be cells such as logic gates and storage elements. We model the attributes of the cells as functions of the cell's internal description and boundary conditions. In particular, we are concerned with a cell's power, input load, and output waveform attributes. The cell's internal description consists of its transistor sizes, layout parasitics, and process parameters. Boundary conditions are imposed on the cell by external agents. These include input waveforms from drivers and output loading from receivers and wiring capacitances. We characterize waveforms as time-shifted ramps with exponential tails. This waveshape is representative of those found in

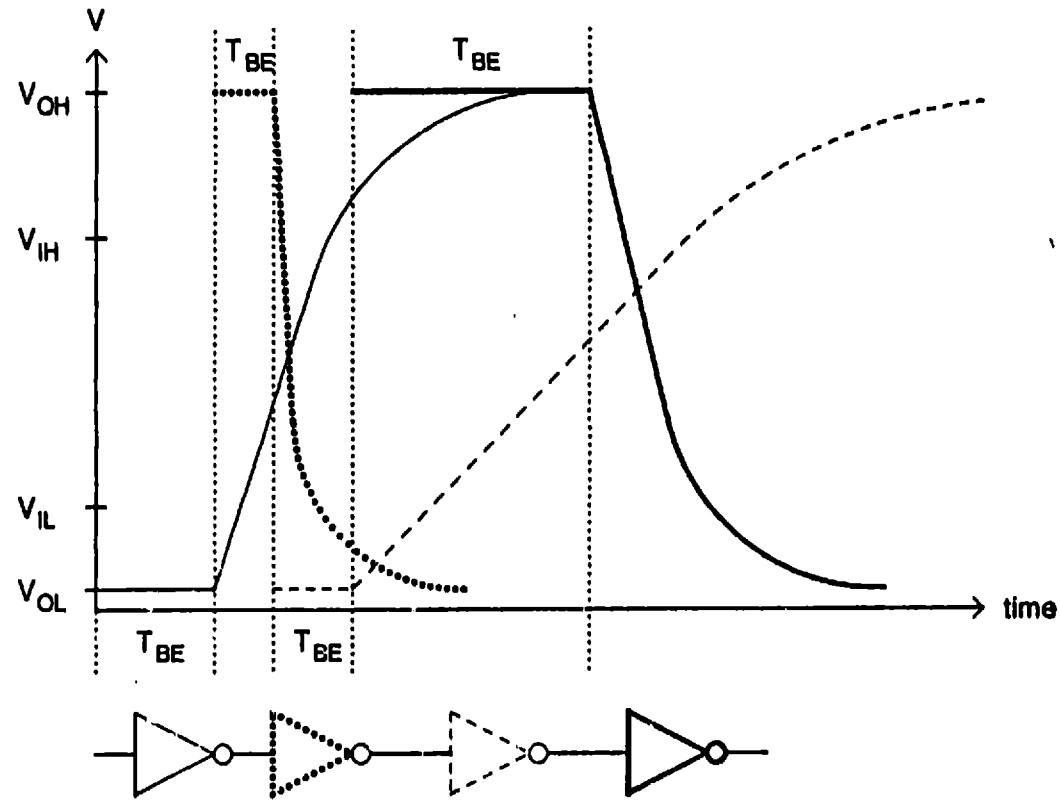


Figure 2-1: Waveform Characterization

digital MOS circuits.[1] Figure 2-1 displays an example. The chain of inverters is driven by a falling input waveform; the figure shows the output waveform of each gate. Here T_{BE} denotes the time shift and T_{SW} the time constant of the exponential portion.[2] Conceptually T_{BE} is the time until the output begins to move in response to an input transition and T_{SW} is a measure of how quickly the output switches once it does begin to change. We curve fit actual circuit waveforms to the time-shifted ramps with exponential tails. From the figure we see that the output waveform of the chain of inverters is described by

---

[1] Actual circuit waveforms begin more smoothly than our approximation. However the error is negligible because the logic gate driven by the waveform does not really begin to switch until the waveform approaches the trigger point voltage (the point on the dc transfer curve where $v_{IN} = v_{OUT}$) and is therefore insensitive to the shape of the first part of the waveform.

[2] For each waveform, the ramp ends and the exponential begins after T_{SWin} (e - 1) seconds of the transition have elapsed. This time was chosen based on typical circuit waveforms. The ramp's slope matches that at the beginning of the exponential, giving a smooth waveform.

$$chain\,T_{BEout} = \sum_{i=1}^{n} T_{BEout\,i}$$

$$chain\,T_{SWout} = T_{SWout\,n}$$

We characterize output loads in terms of an effective capacitance, dividing charge transferred by change in voltage. This is illustrated in Figure 2-2. Note that this allows us to model RC interconnection networks since the effective capacitance can be a function of waveform slope.

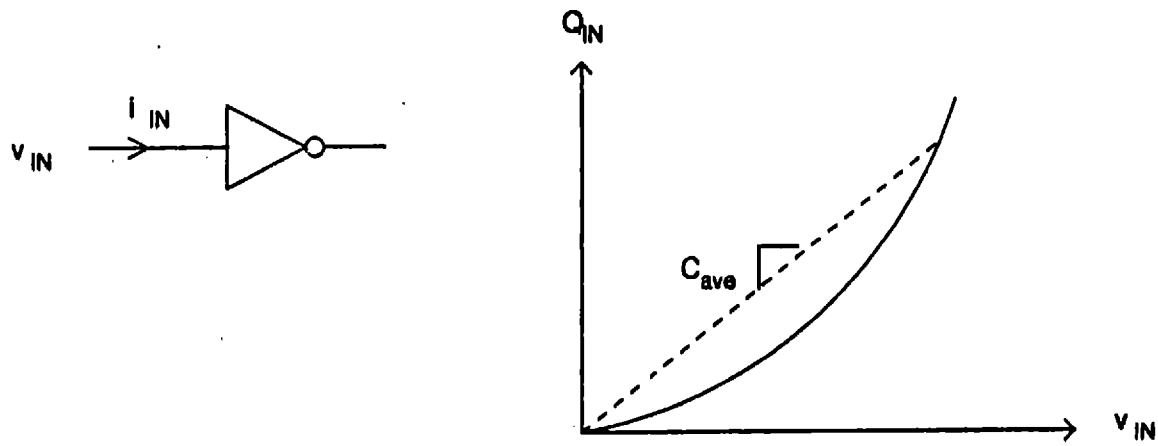


Figure 2-2: Load Characterization

We have effectively "black-boxed" the cell as shown in Figure 2-3. The cell is affected by its environment via the boundary conditions T_{SWin} and C_L. It interacts with its neighbors via its interface attributes C_{in} and T_{SWout}. The internal attributes power and T_{BEout} are isolated from the environment and have no influence on the attributes of the cell's neighbors.

2.3 Inverters

We begin our macromodeling analysis with the ubiquitous inverter, illustrated in Figure 2-4. The results will be extended to more general gates in a subsequent section. For the sake of conciseness, our analysis is only shown for rising input, falling output nMOS gates, although we will present macromodel equations for both transitions. The macromodel equations for CMOS are similar.

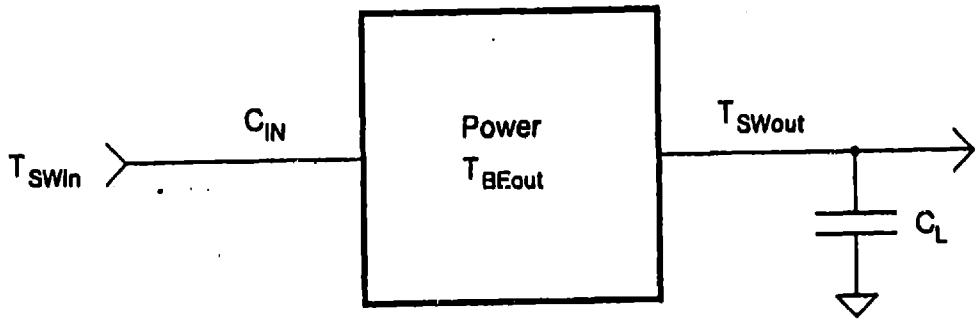


Figure 2-3: Macromodel Representation

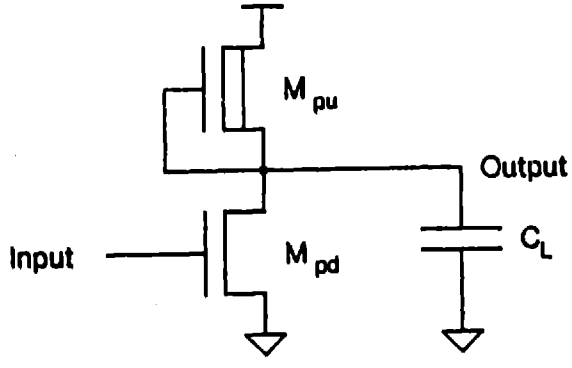


Figure 2-4: A Depletion Load nMOS Inverter

2.3.1 Objective Function

The practicing engineer typically must design circuits such that they satisfy delay specifications. The engineer also desires to minimize some objective function subject to those delay constraints. Power dissipation is a major concern in nMOS technology, since generating an output low requires that both the pullup and pulldown networks be on simultaneously. We accordingly choose to minimize power dissipation, which for nMOS is dominated by static power consumption. The static power consumed by an nMOS inverter is roughly proportional to the shape factor (ratio of channel width to length) of the pullup; that is,

$$Power = a_1 + a_2 S_{pu}$$

where a_1 and a_2 are constants that depend on the fabrication process and power supply voltage.

The choice of an objective function for CMOS circuits is not as clear. Usually a designer wishes to minimize area, power dissipation, or some combination of the two. Characterizing the area consumption is difficult because it is highly dependent on layout techniques. However we can easily describe the contribution of the transistors. This is simply

$$Area = Poly\ Pitch \times \sum_{i=1}^{n} stack\ width_i$$

where we have omitted the transistor lengths because for CMOS they are set to the minimum channel length.

As density increases, power dissipation is becoming a vital issue in CMOS technologies. Unlike the nMOS case, ac power dissipation is dominant. This ac term accounts for the charging and discharging of wiring, parasitic, and transistor gate capacitances. If we desire to have minimum impact on the layout, we might choose to vary only the transistor sizes. A gate's transistors contribute to the ac power dissipation through their effect on the total load capacitance that the gate's driver sees. Their effect on the capacitance, and hence on the ac power since it is proportional to capacitance, is proportional to the total transistor area. Another power term comes from the switching behavior of the logic gate. As the input switches there is a brief interval where both pullup and pulldown networks are on. The magnitude of the ensuing short-circuit current depends on the shape factors of the transistors, while the duration of the interval depends on the slope of the input waveform.

2.3.2 Output Waveform

2.3.2.1 Resistive Model

As we have seen, computational limitations mandate the use of a simple delay model. The simplest transistor representation that provides tolerable accuracy is a switched resistor. The MOS transistor is modeled by a capacitor from the gate to ground and a switched linear resistor from drain to source. The gate to source voltage controls whether the resistor is switched on or off. Figure 2-5 shows an example. The delay characteristics of the model, along with their implications for circuit optimization, have been analyzed by John Hoyte and Lance Glasser in

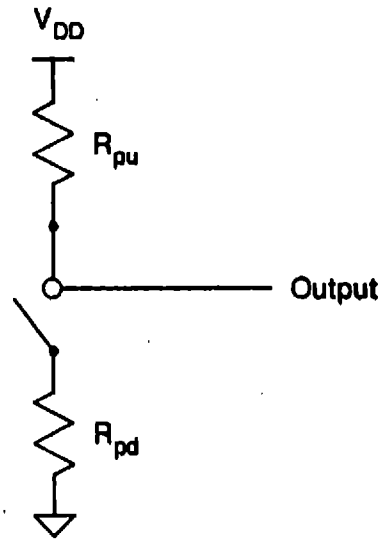


Figure 2-5: Switched Resistor Model

[8] and [9]. The principal advantage of the model is its simplicity, which allows one to derive closed form expressions for the optimal transistor sizes, leading to fast run times. Unfortunately the model can be alarmingly inaccurate. Moreover the errors can be exacerbated by the optimization. These workers found that for a chain of similar gates where the capacitive loading on each stage is dominated by the input capacitance of the next stage (rather than the wiring capacitance), pushing the chain for speed results in equal stage delays. That is, if for a given input transition we desire that the chain respond as quickly as possible, the model predicts that the total delay be uniformly apportioned among the gates. A gate with a rising output switches just as quickly as one whose output is falling. For nMOS this virtually guarantees that while stages with rising outputs are insensitive to input waveshape, those with falling outputs are highly sensitive to input slope. (We will cover this in more detail in the next section.) This sensitivity means that the pulldown transistor cannot be accurately modeled as a resistor, and the effect on total chain delay is significant because the stage delays are equal. Rising output stage delays, for which the resistive model tends to be valid, do not dominate the total delay. The model exhibits errors of up to 70%, clearly unacceptable for serious circuit design.

2.3.2.2 Extended Model

Faced with the inability of the resistive model to account for waveshape effects, we are compelled to derive a more elaborate model. Ever mindful of computation time limitations, we pursue the simplest possible extensions that will provide the needed accuracy. We begin by studying the inverter's response to different input waveform slopes, paying particular attention to the different regions of transistor operation. The equations describing these regions are

$$i_{DS} = 0 \qquad v_{GS} - v_{TH} < 0 \qquad \textit{off}$$

$$i_{DS} = K(v_{GS} - v_{TH})^2 \qquad 0 < v_{GS} - v_{TH} < v_{DS} \qquad \textit{saturated}$$

$$i_{DS} = 2K(v_{GS} - v_{TH} - \tfrac{1}{2} v_{DS}) v_{DS} \qquad v_{GS} - v_{TH} > v_{DS} \qquad \textit{linear}$$

where v_{TH} = threshold voltage

$K = \tfrac{1}{2}(w/l)\mu C_0$

The corresponding i-v characteristic curves are shown in Figure 2-6.

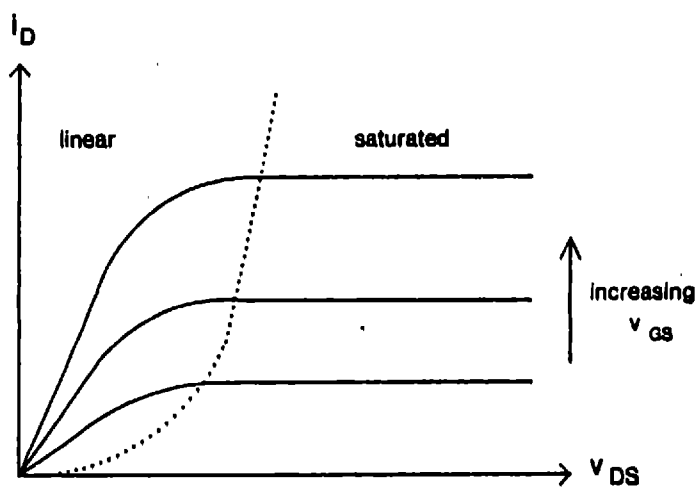


Figure 2-6: MOSFET I-V Characteristics

As the inverter's input rises and its output falls, the pullup and pulldown transistors sequence through different regions of operation. These regions are summarized in Table 2-1. For the fast input response[3] the bulk of the delay accrues from the last states where the pulldown

[3] "Fast" means that the input transition time is fast relative to the output transition time.

is in its linear region. From the transistor's i-v characteristics we see that a line passing through the origin is a reasonable approximation of device behavior in this region. Hence the pulldown can be approximated by a resistor, and the resistive model works well here. However for slow inputs the pulldown is saturated for a significant portion of the transition, causing the inverter to behave like an amplifier. In this mode the inverter is highly sensitive to the input waveform and consequently the resistive model breaks down.

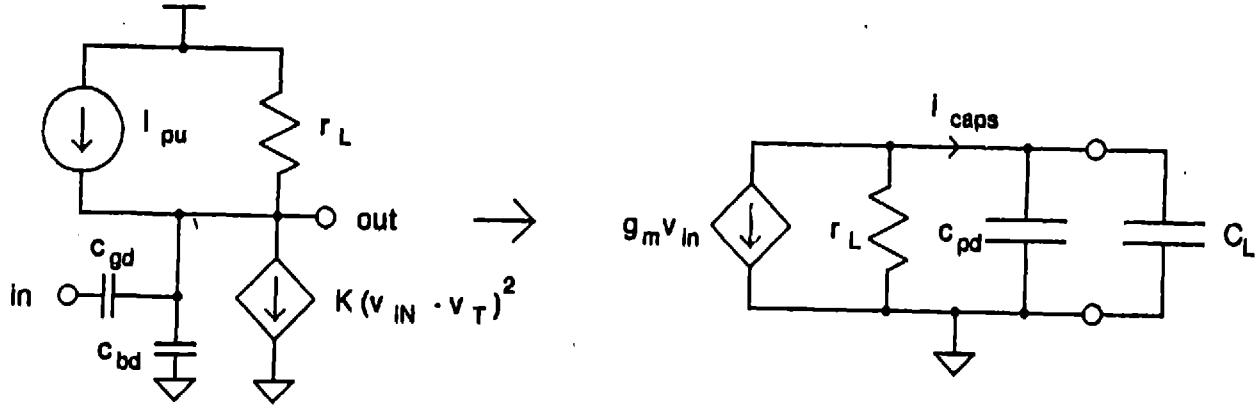
Fast Input Response		***Slow Input Response***	
pullup	*pulldown*	*pullup*	*pulldown*
linear	off	linear	off
linear	sat	linear	sat
linear	linear	sat	sat
sat	linear	sat	linear

Table 2-1: Pullup/Pulldown Regions of Operation

We seek a simple model that includes both amplifier and resistor behavior. We are especially concerned with the middle and latter parts of the input transition, for it is here that the inverter's output is sinking the most current. The beginning of the transition, where the noise margin requirement limits the output current, is not as crucial. For slow inputs the inverter can be modeled as an amplifier when the pulldown is saturated, and as a resistor tied to V_{OL} when the pulldown is in its linear region. As the input transition becomes faster, the inverter spends proportionately less time as an amplifier and more as a resistor. The mapping of the inverter to an amplifier and resistor is shown in Figure 2-7. We begin with the amplifier model when the input voltage reaches V_{IL}. This is an ac model; it measures perturbations from $(v_{IN}, v_{OUT}) = (V_{IL}, V_{OH})$. We change to the resistor model as the pulldown transistor firmly enters its linear region. For continuity of v_{OUT} and i_{OUT} when the change occurs, we use the resistor model when $V_{OL} \geq r_{pd} i_{caps} + v_{OUT}$.

We can acquire much insight into the behavior of the model by studying its drive curves. These curves show the model's v_{OUT} versus v_{IN} relationship for different values of output load current. Figure 2-8 presents the circuit that measures the relationships; Figure 2-9 compares the model's drive curves with those of an actual nMOS inverter. In both of the drive curve figures, the inverter's DC transfer curve is indicated by the solid curve, and corresponds to $I_L = 0$. The transfer curve shifts as I_L becomes nonzero.

Amplifier



Resistor

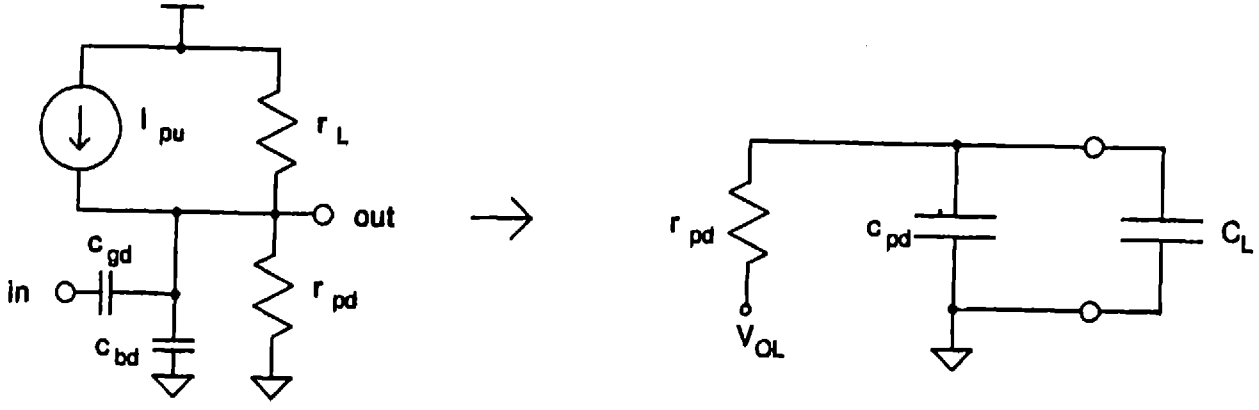


Figure 2-7: Delay Mapping

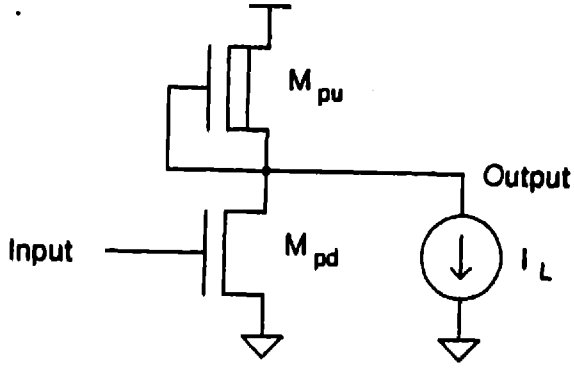
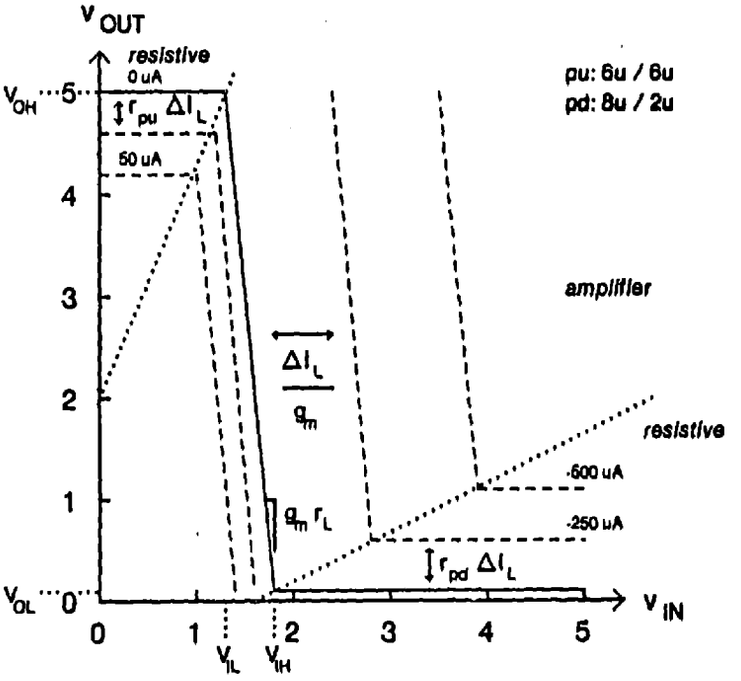
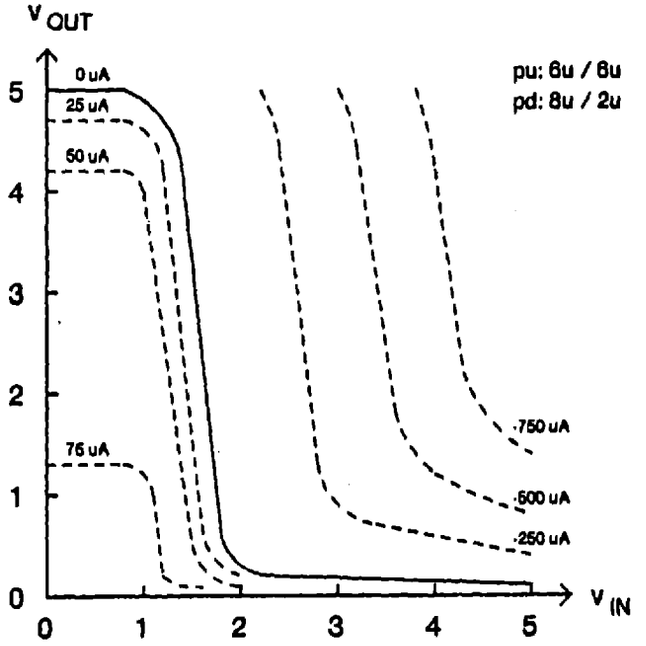


Figure 2-8: Circuit for Determining Drive Curves



Drive Curves for the Inverter Model



Drive Curves for an Actual Inverter

Figure 2-9: Comparison of Drive Curves

The model's drive curves possess several features that have important implications. First, the DC transfer curve is flat outside of the range $v_{IN} \in [V_{IL}, V_{IH}]$. This means that the gate will not respond until a rising input has reached V_{IL}, or until a falling input has dropped to V_{IH}. These "dead zones" provide immunity to noise in the input signal. Second, the two modes of model operation are clearly visible. The horizontal lines on the left and right sides of the figure display the resistive behavior. Here changes in output current produce a proportionate shift in output voltage,[4] and the output voltage is independent of input voltage. The amplifier mode is represented by the slanted lines in the center of the drive curves. Here the output waveform is entirely determined by the input waveform, the output slope being proportional to the input slope. For sufficiently slow inputs the inverter's output will closely track the DC transfer characteristic, exhibiting a strong dependence on the input waveform's shape. For faster inputs the inverter will not be able to track the input waveform, and will be forced out of the amplifier regime into the resistive regime. Therefore the inverter becomes less sensitive to input waveform shape as the input transition speeds up.

The ac model used for the amplifier behavior clashes with traditional engineering philosophy. Normally one creates an ac model by linearizing a circuit about a quiescent operating point. Here however we are interested in large signal behavior. Consequently, while we can perturb the system from an initial point (in this case $(v_{IN}, v_{OUT}) = (V_{IL}, V_{OH})$), we have no easy method to calculate the model's parameters such as g_m and r_L. We cannot simply evaluate the parameters at a quiescent operating point because we have none. We instead view the problem at a more objective-oriented level, and seek to determine which values of g_m, r_L, etc. will provide the closest approximation to observed response times. Moreover, rather than using the same set of parameter values for the rising input and falling input responses (which would correspond to using a single drive curve to characterize the inverter), we procure additional accuracy by using distinct sets for each transition's begin and switch responses. This leads us to the following strategy: analytically derive expressions for the form of the macromodel equations, then curve fit to observed data to solve for the parameters in the equations.

---

[4]The reader may have noticed that this is not true for the low v_{IN}, high v_{OUT}, portion of the actual inverter's drive curves for larger I_L. This is because the pullup has moved from its linear region into saturation, and is behaving like a current source rather than a resistor. However the macromodel equations that we will develop depend only on the pullup's current sourcing ability being proportional to its shape factor, which will be true whether the pullup is best modeled as a resistor or as a current source. This is a consequence of our delay parameterization; T_{BE} and T_{SW} measure the duration of the inverter's response, which is inversely proportional to shape factor for either a resistor or current source model.

Output Waveforms for Different Input Slopes

We can derive closed form expressions for the model's response to different input waveforms. We begin by studying the response to fast input transitions and work toward very slow transitions. The inverter is driven by a ramp

$$v_{IN} = \left(\frac{\Delta V}{T_{SWRIn}}\right) t + V_{OL}$$

where ΔV is an arbitrary positive voltage difference and we vary T_{SWRIn} to select the time for the ramp to transit this difference. The ramp is held at V_{OH} once it reaches that voltage. The implementation of the modeler curve fits actual responses to time-shifted ramps with exponential tails. We approximate this by taking T_{BEF}, the delay before the output begins to fall, as the time required for the output to drop to some V_{BEF}. We take T_{SWF}, the measure of output switching time, to be proportional to the reciprocal of the slope at some V_{SWF}.

Very Fast Inputs

For sufficiently fast inputs the inverter model changes from an amplifier to a resistive form immediately. In other words, the first order resistive model is valid. Figure 2-10 shows the model. Lumping the self-capacitances c_{gd} and c_{bd} from the pulldown (the self-capacitance from the pullup is negligible, or so SPICE claims) into one effective capacitance[5] c_{pd}, we find

$$v_{OUT} = (V_{OH} - V_{OL}) \exp\left(-\frac{t - t_{IL}}{T_{De}}\right) + V_{OL}$$

$$T_{BEF} = T_{De} \ln\left(\frac{V_{OH} - V_{OL}}{V_{BEF} - V_{OL}}\right) + t_{IL}$$

$$T_{SWF} \propto \frac{T_{De}}{V_{SWF} - V_{OL}}$$

[5]The effective capacitance c_{pd} is not simply the sum of c_{gd} and c_{bd}. The contribution from the gate to drain coupling capacitance is larger than c_{gd} because the gate voltage is not fixed, but rising. Our macromodel equations will account for this.

where $T_{De} = r_{pd} C_{total}$, $C_{total} = c_{pd} + C_L$

t_{IL} = time for input to reach V_{IL}

$$= T_{SWRin} \frac{V_{IL} - V_{OL}}{\Delta V}$$

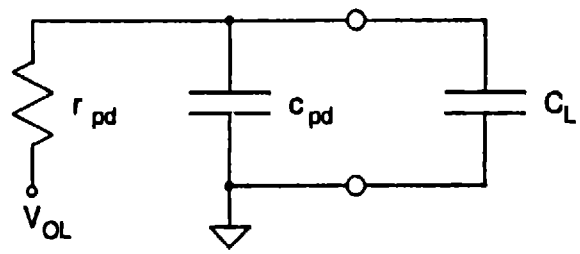


Figure 2-10: Resistor Model for a Very Fast Input

Fast Inputs

In this regime the inverter model does not change from an amplifier to a resistive form immediately, but does change before the output drops to V_{SWF}. Figure 2-11 shows the amplifier model. The output switches quickly enough that the current in the total load capacitance C_{total} dominates that of r_L, allowing us to neglect the resistor. Horowitz [7], who derived delay estimates and bounds for inverters using a simpler model, has analyzed the behavior of this regime. The response prior to the model change is

$$v_{OUT} = V_{OH} - \frac{g_m \Delta V}{2 C_{total} T_{SWRin}} (t - t_{IL})^2$$

$$T_{BEF} = \left[(V_{OH} - V_{BEF}) \frac{2 C_{total} T_{SWRin}}{g_m \Delta V} \right]^{1/2} + t_{IL}$$

The model changes to a resistor once the condition $V_{OL} \geq r_{pd}\, i_{caps} + v_{OUT}$ is satisfied. Denoting the time at which this occurs as t_{change}, we have

$$t_{change} = r_{pd} C_{total} \left\langle -1 + \left[1 + \frac{2}{g_m r_{pd} \cdot r_{pd} C_{total}} \frac{T_{SWRin}}{\Delta V} (V_{OH} - V_{OL}) \right]^{1/2} \right\rangle + t_{IL}$$

$$V_{change} = g_m r_{pd} \cdot r_{pd} C_{total} \frac{\Delta V}{T_{SWRin}} \left[1 + \frac{2}{g_m r_{pd} \cdot r_{pd} C_{total}} \frac{T_{SWRin}}{\Delta V} (V_{OH} - V_{OL}) \right]^{1/2} + V_{OL}$$

$$v_{OUT} = (V_{change} - V_{OL}) \exp\left(- \frac{t - t_{change}}{T_{De}} \right) + V_{OL}$$

$$T_{SWF} \propto \frac{T_{De}}{V_{SWF} - V_{OL}}$$

where $T_{De} = r_{pd} C_{total}$

$$t_{IL} = T_{SWRin} \frac{V_{IL} - V_{OL}}{\Delta V}$$

Moderate Inputs

Here the input waveform is slow enough that the model changes after the output has fallen to V_{SWF}. The current in C_{total} still dominates that of r_L. The simplified circuit is shown in Figure 2-11. The equations for v_{OUT} and T_{BEF} are identical to those of the previous case, but the expression for T_{SWF} becomes

$$T_{SWF} \propto \left[\frac{C_{total} T_{SWRin}}{(V_{OH} - V_{SWF}) 2 g_m \Delta V} \right]^{1/2}$$

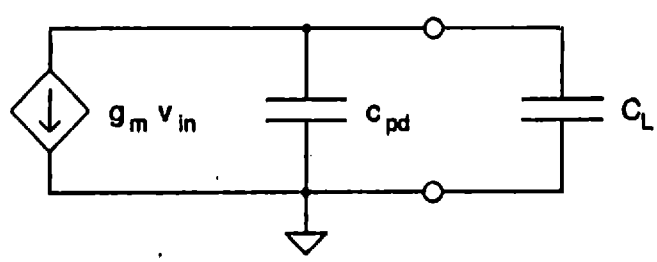


Figure 2-11: Amplifier Model for Fast and Moderate Inputs

Slow Inputs

As we continue to slow down the input waveform, we eventually reach a point where we can no longer neglect the current through r_L. The resulting circuit is shown in Figure 2-12. This yields the following differential equation in the ac component of v_{OUT}:

$$g_m \frac{(t - t_{IL})\Delta V}{T_{SWRin}} + \frac{v_{out}}{r_L} + C_{total}\frac{dv_{out}}{dt} = 0$$

Unfortunately the solution for v_{OUT} contains both exponential and linear terms, and we cannot obtain closed form expressions for T_{BE} and T_{SW}.

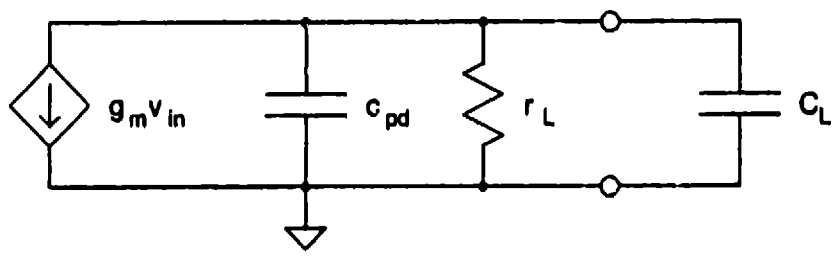


Figure 2-12: Amplifier Model for Slow Inputs

Very Slow Inputs

Here the input and output waveforms have slowed to the point where the current through C_{total} is almost negligible compared to that through r_L. The amplifier system reaches steady state, exhibiting a constant tracking error to the ramp input, being entirely limited by the speed of the input. A Laplace transform analysis of the system reveals that the output crosses the inverter's trigger point[6] at a time $r_L C_{total}$ after the input does,[7] with a slope equal to the input's divided by the DC gain [10]. Accordingly we have

$$t_{TPRin} = \frac{V_{TP} - V_{OL}}{\Delta V} T_{SWRin}$$

---

[6] The point on the inverter's DC transfer curve where the input and output voltages are equal.

[7] The reason for the delay can be clarified by considering a simpler circuit, where a very slow voltage ramp drives a series RC connection. The capacitor voltage will lag the ramp by the RC delay. Our amplifier model can be transformed into this representation by taking its Thevenin equivalent, indicating an output delay of $r_L C_{total}$.

$$T_{BEF} = t_{TPRin} + r_L C_{total}$$

$$T_{SWF} \propto \frac{T_{SWRin}}{g_m r_L \Delta V}$$

where V_{TP} = the inverter's trigger point voltage

Modeling the Model

Combining the results from the previous section gives us the response for the entire range of input waveforms. This is illustrated in Figure 2-13. Figure 2-14 shows actual data for the rising and falling output transitions. The figures also include curves for T_{TPout}, the delay from when the input signal crosses the inverter's trigger point voltage to when the output does. Note the differences in the actual data for the two transitions. For nMOS technologies, static gates turn on both pullup and pulldown networks in order to generate a low output. Guaranteeing a valid output low requires that the resistance of the pulldown network be much lower than that of the pullup network. For very fast inputs we expect T_{SWout} to be roughly constant. This is apparent for the rising output transition, but for the falling transition, due to the low r_{pd}, this region is scarcely noticeable. The effect can also be seen in the T_{BEout} and T_{TPout} curves. The input to output coupling capacitance and the pullup/pulldown resistance form a high pass filter. Since r_{pu} is much larger than r_{pd}, the filter's cutoff point for rising outputs occurs at slower inputs than it does for falling outputs. Furthermore, because the pulldown transistor is in its linear region during fast falling input transitions, c_{gd} is larger, and more of the input transition couples to the output. Consequently the T_{BERout} and T_{TPRout} curves become convex as we move toward very fast inputs. The capacitive coupling is forestalling the rise of the output. The dissimilarity in resistances also implies a low trigger point voltage. For the same input slope, it takes longer for a falling input to transit the dead zone and approach the trigger point voltage than it does for a rising input. Correspondingly the T_{BERout} curve is steeper than the T_{BEFout} curve. For CMOS technologies the resistances are nearly symmetric, and both output transitions exhibit curves resembling those shown for the nMOS falling output transition.

Having developed equations describing the inverter's response to various input slopes, we now seek a means of coalescing the results into one conglomerate expression. It is common to use smoothing functions to effect this combination. However many workers fail to consider the

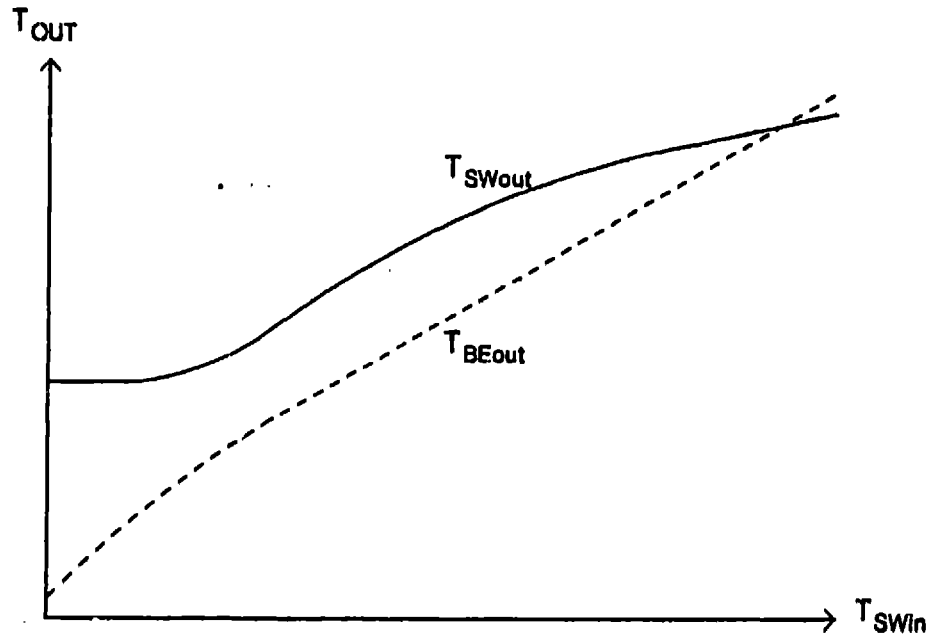


Figure 2-13: Inverter's Predicted Response

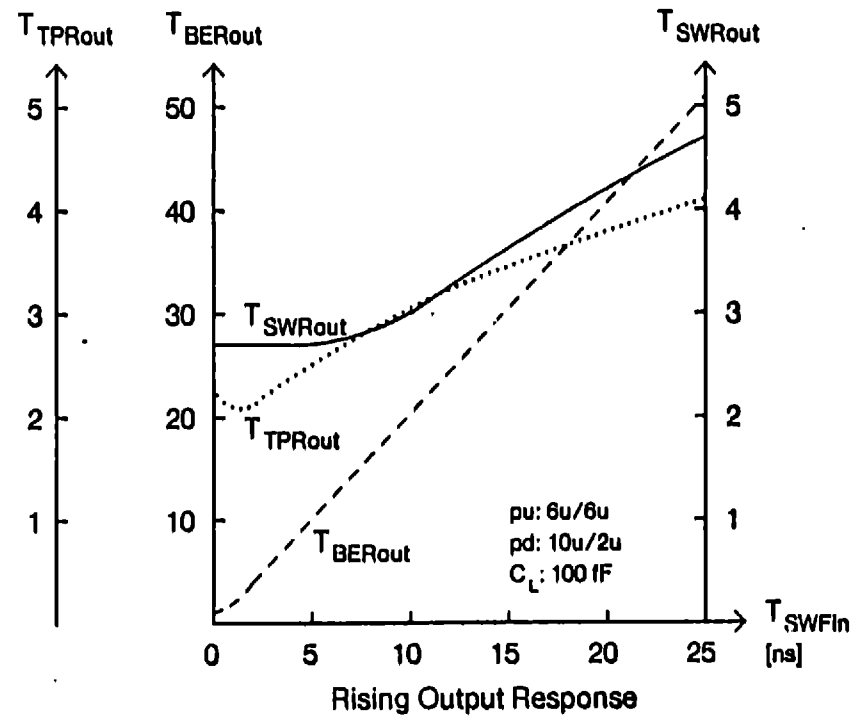
computational overhead incurred with these functions. We instead create simple functions that exhibit the desired behavior in each of the input slope regimes. To avoid placing any unnecessary burdens on the optimization algorithms, we choose functions that are twice continuously differentiable.[8] Moreover we prefer functions that do not contain multiple maxima or minima; i.e., that are unimodal. This helps eliminate cusps that could trap the optimizer's iterative solution technique.

To describe the time until the inverter's output begins to fall in response to a rising input we use

$$T_{BEFout} = T_{BEFO} + m T_{SWRin}$$

Here the intercept T_{BEFO} is obtained from our analysis of the inverter's response to a moderate input; the slope m includes both the time to reach the inverter's V_{IL} and the moderate input behavior. Fixing the length of the pulldown transistor's channel, we have

[8] Optimization algorithms exist for solving problems with ill-behaved (e.g., discontinuous) functions, but because of their added generality these algorithms tend to be slower.



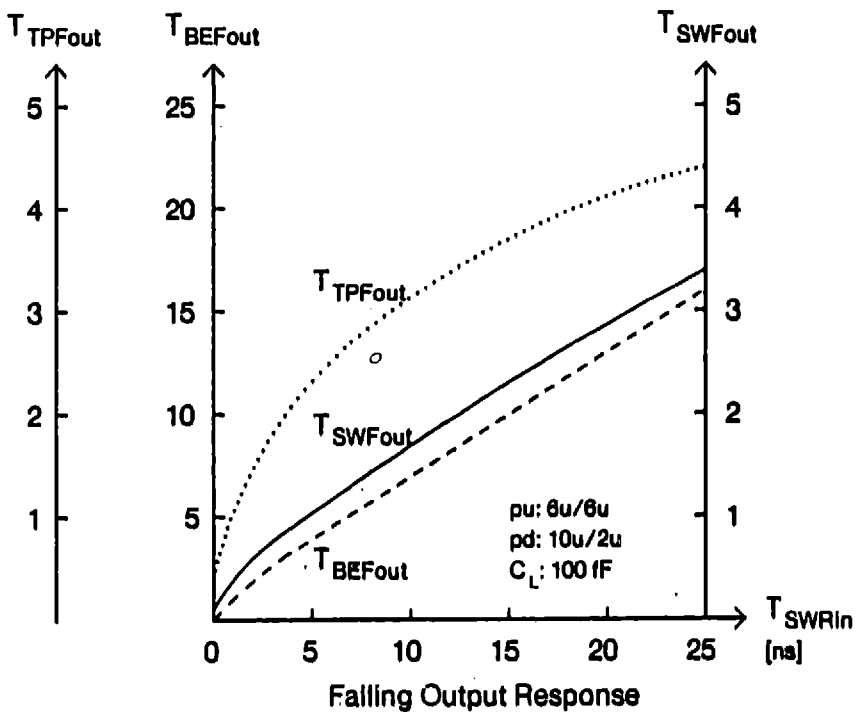


Figure 2-14: Inverter's Actual Output Responses

$$T_{BEFO} = a_1 + b_1 \frac{1}{g_m} C_{total}$$
$$\rightarrow a_1 + b_1 \frac{1}{w_{pd}} C_{total}$$

$$m = d_1 + d_2 \frac{g_{pu}}{g_{pd}} + d_3 \frac{C_{total}}{g_m}$$
$$\rightarrow d_1 + d_2 \frac{S_{pu}}{w_{pd}} + d_3 \frac{C_{total}}{w_{pd}}$$

The switching time of the output's fall is described in a similar fashion:

$$T_{SWFout} = T_{SWFO} + m T_{SWRin}$$

T_{SWFO} is the inverter's response to a very fast input; it is proportional to the product of the pulldown's resistance and the total capacitance. The slope m chiefly depends on the inverter's very slow and moderate input behaviors.

$$T_{SWFO} = a_1 + r_{pd} C_{total}$$
$$\rightarrow a_1 + \frac{b_1}{w_{pd}} C_{total}$$

$$m = d_1 + d_2 \frac{1}{g_m r_L} + d_3 \frac{C_{total}}{g_m}$$
$$\rightarrow d_1 + d_2 \frac{S_{pu}}{w_{pd}} + d_3 \frac{C_{total}}{w_{pd}}$$

We can characterize the delay until the output begins to rise in response to a falling input as

$$T_{BERout} = T_{BERO} + m T_{SWFin}$$

Due to the high resistance of the pullup, T_{BERO} depends primarily on the inverter's very fast input behavior. The slope m depends on the delay in reaching V_{IH} as well as the moderate input behavior.

$$T_{BERO} = a_1 + b_1 r_{pu} C_{total}$$
$$\rightarrow a_1 + b_1 \frac{1}{S_{pu}} C_{total}$$

$$m = d_1 + d_2\frac{g_{pu}}{g_{pd}} + d_3\frac{C_{total}}{g_m}$$
$$\rightarrow d_1 + d_2\frac{S_{pu}}{w_{pd}} + d_3\frac{C_{total}}{w_{pd}}$$

The equation for the switching time of the output's rise is more complicated than that for the output's fall, since both the very fast input and moderate input behaviors are significant, whereas for T_{SWFout} the RC regime can be ignored because the curve's shape is dominated by the gain limited regime. We write

$$T_{SWRout} = \frac{(T_{SWRO} - T_{SWRO}^{asym})^2}{(T_{SWRO} - T_{SWRO}^{asym}) + mT_{SWFin}} + (T_{SWRO}^{asym} + mT_{SWFin})$$

T_{SWRO} is determined by the very fast input behavior, and reflects the product of the pullup resistance and the total capacitance. The slope m is again related to the very slow and moderate input behaviors. T_{SWRO}^{asym} is the intercept of the curve's slow input asymptote.

$$T_{SWRO} = a_1 + r_{pu}C_{total}$$
$$\rightarrow a_1 + \frac{b_1}{w_{pd}}$$

$$m = d_1 + d_2\frac{1}{g_m r_L} + d_3\frac{C_{total}}{g_m}$$
$$\rightarrow d_1 + d_2\frac{S_{pu}}{w_{pd}} + d_3\frac{C_{total}}{w_{pd}}$$

$$T_{SWRO}^{asym} = e_1 + e_2 T_{SWRO}$$

The preceding equations all depend on the total capacitance C_{total} driven by the inverter. The pulldown's gate to drain and body to drain capacitances contribute to this; both are proportional to the pulldown's width. In our analysis of the inverter's response we lumped these capacitances into one effective capacitance, c_{pd}. We have

$$c_{pd} = c_1 w_{pd}$$

2.3.3 Input Capacitance

Calculating a gate's delay requires knowledge of the input capacitances of the gates that it drives. In this section we derive expressions for an inverter's input capacitance. Our results will be extended to more general logic gates in a later section.

We begin by considering the components of the input capacitance. Figure 2-15 shows our model. The input capacitance has two constituents: the gate to drain and gate to source transistor capacitances.

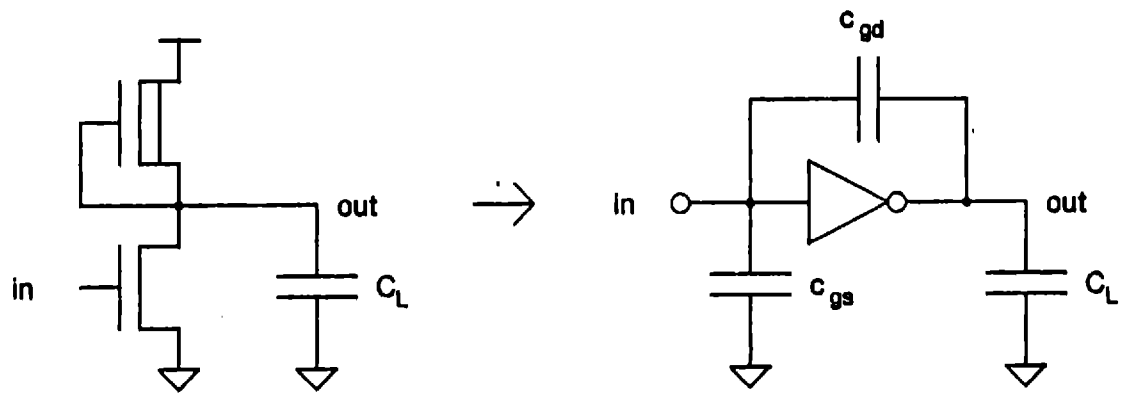


Figure 2-15: Input Capacitance Model

The input capacitance presented to the driver can change during the course of the input transition. This effect is largely due to the input to output coupling capacitance c_{gd}. Consider a rising input transition of moderate speed. During the beginning portion of the input waveform—that is, before the input voltage has reached V_{IL}—the inverter's output has not yet moved significantly. The input capacitance is therefore simply $c_{gs} + c_{gd}$. Since both terms are proportional to the pulldown transistor's width, we have

$$C_{BEin} = c_{gs} + c_{gd} \rightarrow a_1 + a_2 W_{pd}$$

However as the input voltage passes V_{IL} the inverter begins to pull its output low. Consequently the driver must supply more current to charge c_{gd} than it would have had the output voltage remained fixed. This is called the Miller effect [11]. The effective input capacitance has increased. Note that the total voltage change across c_{gs} is always $V_{OH} - V_{OL}$, while that across c_{gd} is 2 $(V_{OH} - V_{OL})$, but we are only interested in the capacitance seen during the beginning and

switching portions of the input waveform. For very fast input transitions the output will not have moved until after the input has fully switched; hence the driver will not have seen any Miller capacitance during the actual transition. As we slow the speed of the input transitions, more of the output's switching time overlaps with the input's and we see more Miller capacitance. Eventually all of the output's switching time overlaps with the input's and C_{SWIn} reaches a plateau. The expected behaviors of the effective input capacitances C_{BEin} and C_{SWIn} appear in Figure 2-16.

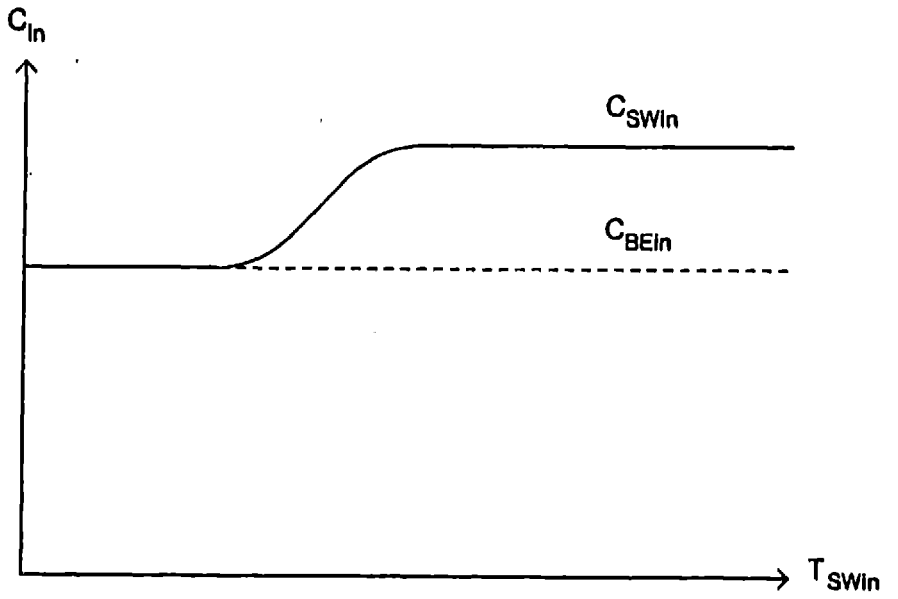


Figure 2-16: Expected Input Capacitance

The analysis is complicated slightly by the fact that since c_{gd} and c_{gs} are functions of v_{GS} and v_{GD}, they not only vary as the gate switches, but their average value during the input transition changes as the input transition slows down. Figure 2-17 displays their variation as a function of the drain to source voltage. Detailed discussions of the capacitances' origin and behavior may be found in [12, 13].

The outcome is that we see two effects as the input transition slows down: more of the output switches during the input transition, and the average c_{gs} and c_{gd} seen during the input transition changes. For the falling input, rising output transition the pulldown begins in its linear region and terminates in saturation. As the switching time of the input increases, the pulldown

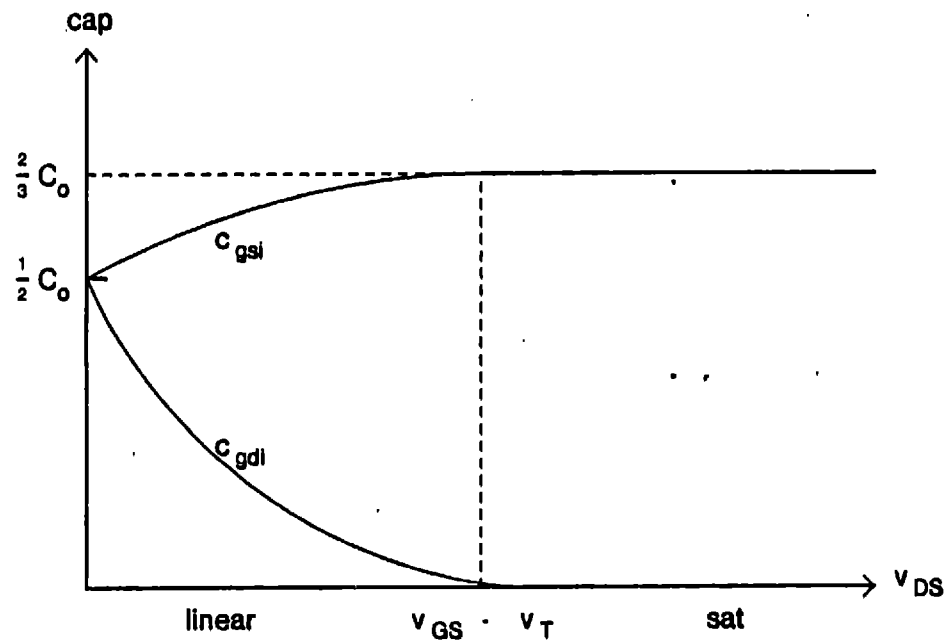


Figure 2-17: Gate Capacitances

spends proportionately less time in its linear region and more in the saturation region.[9] From the c_{gs} and c_{gd} curves in Figure 2-17 we see that c_{gs} increases slightly while c_{gd} drops. Also, more of the output transition overlaps with the input transition. Hence, although the coupling capacitance c_{gd} has dropped, this is offset by the increased overlap between transitions, and the Miller capacitance is virtually unchanged. Figure 2-18 provides an illustration from SPICE simulations of an inverter. C_{SWFin} is roughly independent of T_{SWFin}, and only depends on the size of the pulldown. For a fixed pulldown width we have

$$C_{SWFin} = a_1 + a_2 w_{pd}$$

For the rising input, falling output transition the pulldown begins saturated and ends in its linear region. For fast inputs the pulldown will remain saturated for most of the input transition. As we slow down the input transition, the inverter is more able to track the change at its input, and there is less delay between the rise of the input and fall of the output. Consequently the pulldown spends proportionately more time in the linear region and less in the saturated region. The gate

[9] We saw this same phenomenon in the section on delay: as the input transition slows, the pulldown spends more time in saturation and the inverter behaves more like an amplifier.

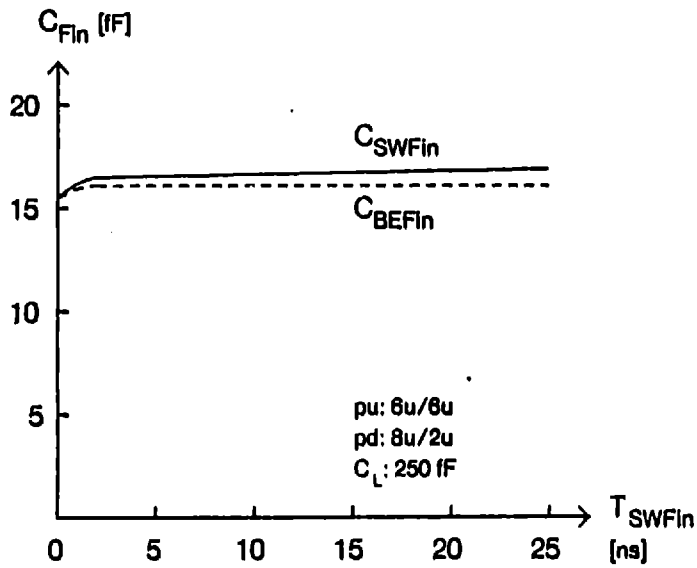


Figure 2-18: Capacitance for a Falling Input

capacitance curves in Figure 2-17 show us that c_{gs} diminishes slightly while c_{gd} increases. As we have seen, slower input transitions imply that more of the output transition overlaps with the input transition. Unlike the case for C_{SWFin}, here the two Miller capacitance effects have complemented rather than offset one another. The coupling capacitance and the input and output transition overlap both increase, giving rise to a significant Miller effect.

Figure 2-19 displays data from SPICE simulations; the data agrees fairly closely with our analysis. The input capacitance begins at a minimum, and then increases as the input transition slows, increasing the input to output transition overlap and the coupling capacitance. Once the input transition extends beyond the entire output transition, the overlap cannot be further increased and the curve reaches a plateau. For very fast inputs the input transition completes before the output has a chance to drop. We have

$$C_{SWROIn} = c_{gs} + c_{gd}$$

where c_{gs} and c_{gd} are the average gate capacitances during the input transition. Since both are proportional to w_{pd}, we may write

$$C_{SWROIn} = a_1 w_{pd}$$

For slow inputs we see the full effect of the Miller capacitance. We have

$$C_{SWRoo\,in} = c_{gs} + 2c_{gd}$$

$$\rightarrow b_1 w_{pd}$$

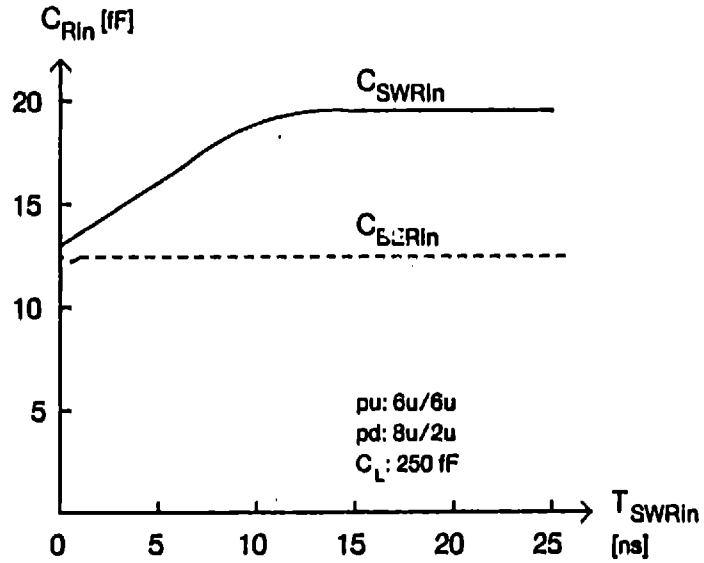


Figure 2-19: Capacitance for a Rising Input

We have seen that the input capacitance C_{SWRIn} begins at a minimum. Owing to the Miller effect, it rises to a plateau as T_{SWRIn} increases. We will now characterize the transition region between the fast and slow T_{SWRIn} regimes. As always, we desire an accurate yet computationally simple characterization, preferably twice continuously differentiable. We choose to parameterize the transition region in terms of an initial slope m, as shown in Figure 2-20. Here we have taken advantage of the fact that the interval of constant C_{SWRIn} prior to the transition region is insignificant relative to the duration of the region. This arises because for very fast inputs the output begins to fall almost immediately. The drop in the pulldown's v_{DS} brings the pulldown out of saturation, causing it to exhibit significant c_{gd} during even fairly fast input transitions. Consequently C_{SWRIn} begins to grow virtually as soon as the input transition slows and is no longer a step.

The slope of the transition region is governed by the load capacitance, transconductance, and c_{gd} of the pulldown. As the input transition slows the Miller effect increases. Since the coupling capacitance c_{gd} is proportional to the width of the pulldown, this increase in Miller

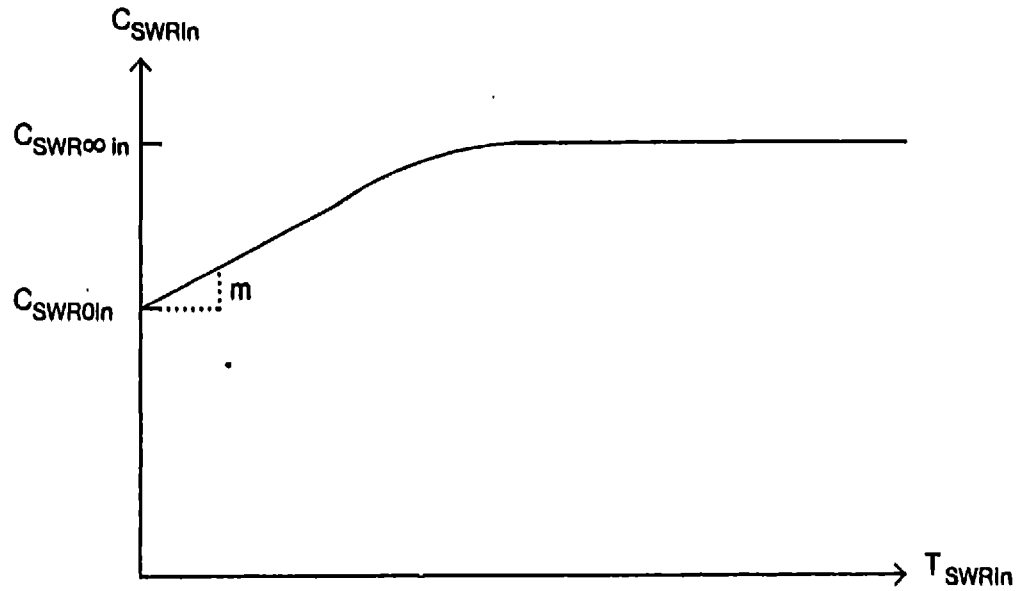


Figure 2-20: Approximate Capacitance for a Rising Input

effect, and hence in m as well, must also be proportional to the width. The load capacitance and transconductance are pertinent because of their influence on the output waveform. For transistor sizes and loads that provide fast output waveforms, the transition region will be narrow because the input transition need only slow down slightly in order to completely overlap the output transition. This implies a steep slope m. In contrast, large C_L and small g_m will lead to relatively shallow slopes. From our analysis of delay we saw that the ratio of C_L to g_m determines the speed of the output transition, and so we may write

$$m = w_{pd}\left(c_1 + c_2\frac{g_m}{C_L}\right)$$

$$\rightarrow w_{pd}\left(c_1 + c_2\frac{w_{pd}}{C_L}\right)$$

Summarizing, our final equation for C_{SWRIn} is

$$C_{SWRIn} = C_{SWRO} + \Delta C_{SWRIn}\left(\frac{m T_{SWIn}}{\Delta C_{SWRIn} + m T_{SWIn}}\right)$$

$$\Delta C_{SWRIn} = C_{SWR\infty in} - C_{SWR0In}$$

$$C_{SWROin} = c_{gs} + c_{gd}$$
$$\rightarrow a_1 w_{pd}$$

$$C_{SWRooin} = c_{gs} + 2c_{gd}$$
$$\rightarrow b_1 w_{pd}$$

$$m = w_{pd}\left(c_1 + c_2 \frac{g_m}{C_L}\right)$$
$$\rightarrow w_{pd}\left(c_1 + c_2 \frac{w_{pd}}{C_L}\right)$$

For a CMOS inverter both the rising and falling input switching capacitance exhibit significant Miller capacitance. Recall that the magnitude of the Miller capacitance is affected by the input to output transition overlap and the change in average c_{gd} and c_{gs} as the input waveform slows down. As we saw in the nMOS case, for a transistor that is being turned off the two effects roughly cancel, while for a transistor being turned on they complement one another, producing a substantial Miller effect. Since for a CMOS inverter an input transition always turns on one device and turns off another, we always see a Miller effect capacitance, and model both C_{SWFin} and C_{SWRin} as above.

2.4 General Logic Gates

Inverters are but one of a myriad of logic gates found in circuit designs. In this section we will extend our results to cover a more general class of logic gates. We limit our analysis to logic gates with a single active input, as shown in Figure 2-21. Transitions at multiple inputs are not supported by our abstract model; accurate evaluation of their effects requires a low-level simulator that computes node voltages and branch currents. We feel that this represents an excessive computation cost and therefore choose a worst case gate state with a single active input to model multiple input transitions.

We will derive macromodel equations for the general logic gate by extending our inverter equations. As regards the objective function—be it power or area—the equations are basically unchanged. The power consumption of an nMOS gate is still proportional to the shape factor of the pullup, and the power or area consumption of a CMOS gate is still dependent on the stack

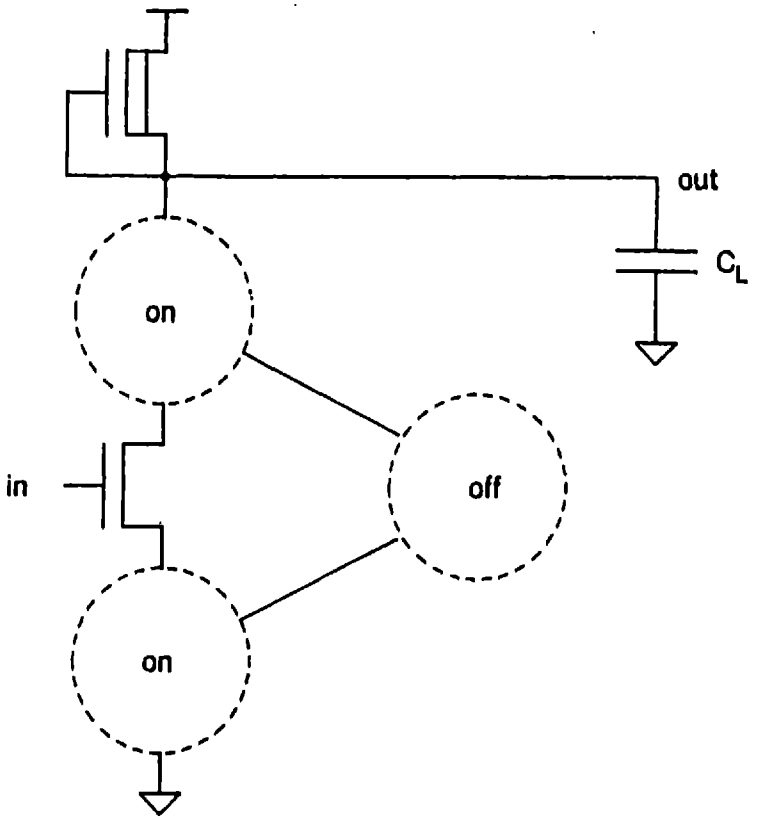


Figure 2-21: General Logic Gate

widths. However the equations for the output waveform and input capacitance require moderate extensions.

2.4.1 Output Waveform

Additional transistors in a logic gate introduce two complications. If they are part of the path that switches the output by forming a path to V_{DD} or ground, their resistance and capacitance impede the output transition. If they are included in a side path that does not connect the output to a supply rail, their channel capacitance could add to the load capacitance and hinder the output transition. During the output switching transient, transistors with high inputs are predominantly in their linear region. Hence we model them as RC lines formed of their drain to source resistance and channel to gate and substrate capacitance.

Figure 2-22 contains an example. Note that although the top transistor in the right pulldown

branch is not in the conducting path, its capacitance adds to the total load. The general situation is depicted in Figure 2-23.

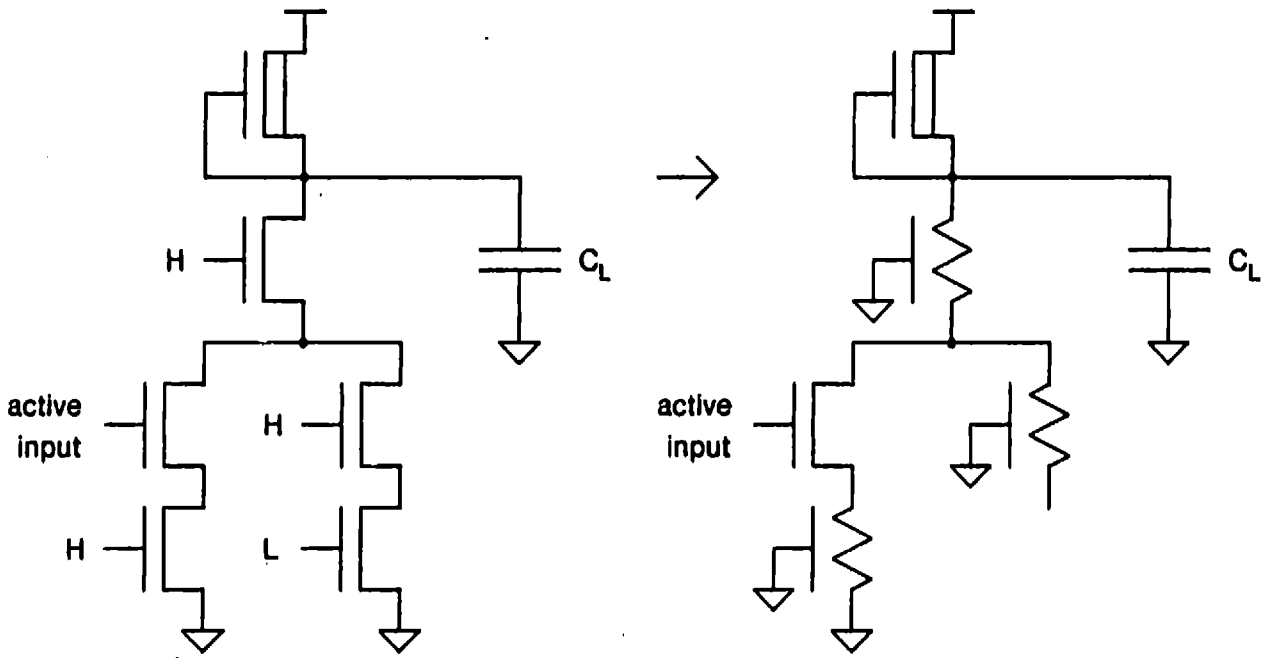


Figure 2-22: Example of a General Logic Gate

Output Waveforms for Different Input Slopes

In this section we will indicate the basic modifications needed to the inverter output waveform equations.

Very Fast Inputs

For very fast inputs transitions we again take advantage of the fact that the switching transistor is primarily in its linear region, and model it as an RC line. Although we cannot derive a closed form solution for the response since the circuit contains a distributed RC line,[10] we can

[10] We could approximate the RC lines as lumped elements, but this would lead to very complicated and computationally expensive equations.

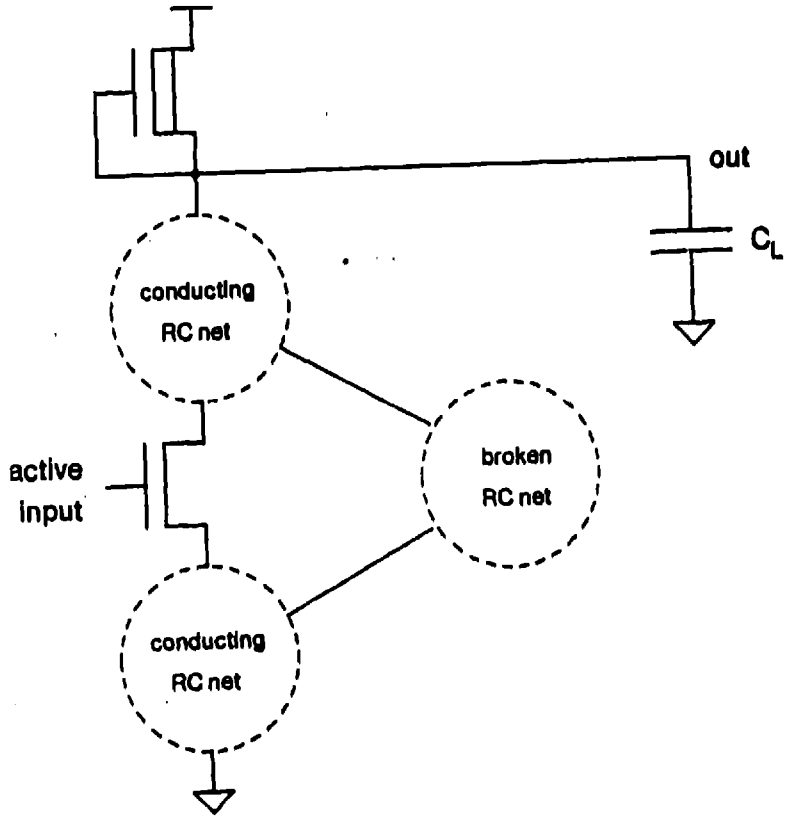


Figure 2-23: Circuit Model for a General Logic Gate

find the approximate response by using an approach first proposed by Elmore [14]. The true response is approximated as a single time constant exponential of the form

$$v_{OUT} = (V_{OH} - V_{OL})\, e^{-(t - t_{IL})/T_{De}} + V_{OL}$$

where t_{IL} = time for input to reach V_{IL}
T_{De} = Elmore delay

Although Elmore originally developed the approximation for analyzing cascaded linear amplifiers, his work has recently resurfaced in waveform estimation and bounding work in MOS circuits [15, 16]. The Elmore delay is the time constant that gives the minimum error between the approximate and true responses. Intuitively, the Elmore approximation simply superposes the effect of each capacitor's current on the output node. For an RC tree, the Elmore estimate for a falling voltage at a particular node *i* is

$$T_{De} = \sum_k R_{kl} C_k$$

Here R_{kl} is the resistance in common between the path from the input to node k and the path from the input to the output of interest. For example, in Figure 2-24 the Elmore delay for the top output is

$$T_{De} = R_1 C_1 + (R_1 + \tfrac{1}{2} R_2) C_2 + (R_1 + R_2 + R_3) C_3 + (R_1 + R_2) C_4$$

We apply the Elmore delay formula to compute T_{BEFout} and T_{SWFout} for our general logic gate.

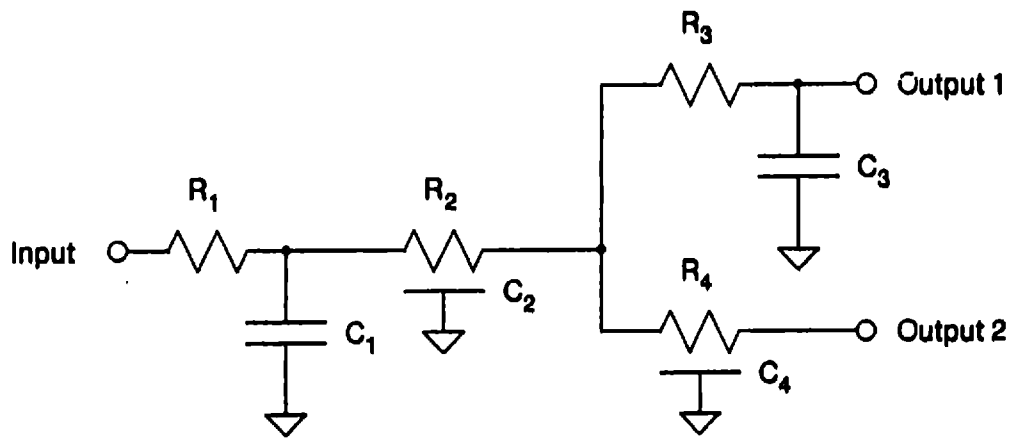


Figure 2-24: An RC Tree Network

Moderate Inputs

In this regime the speed of the output transition is limited by the slope of the input and transconductance of the switching transistor. Consequently, transistors in the conducting path which are electrically after the switching transistor have small v_{DS} and we can neglect their voltage drops.[11] However we must add their capacitances (along with those of any transistors connected to them) to the total load capacitance. Transistors which are before the switching transistor do not impose any additional load since their capacitances are already discharged; nonetheless their resistance will decrease the switching transistor's effective g_m if they are in the conducting path, impeding the output transition. This effect is illustrated in Figure 2-25. The effective g_m has been reduced to $g_m / (1 + g_m r_b)$.

[11] This is not a cascode connection because these transistors are in their linear regions and behave as resistors, not current sources. Hence they are detrimental rather than beneficial to performance.

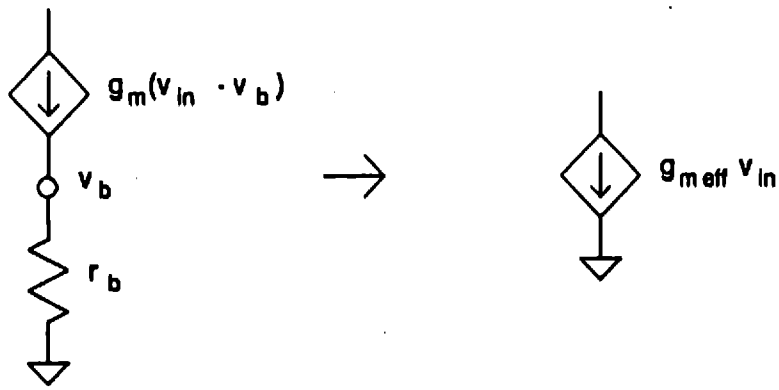


Figure 2-25: Reduction of Effective Transconductance

Very Slow Inputs

The effects are identical to those of the previous case: transistors which are electrically after the switching transistor increase the total load capacitance, while those before it decrease the effective transconductance.

Curve Fitting the Response

Combining our results, we extend the inverter's macromodel to encompass more general gates as follows:

Let w_{pd} = width of the switching device

$w_{pdtotal}$ = effective total w_{pd} of devices in the conducting path (treat as if w_{pd}'s were conductances)

$w_{pdbefore}$ = effective total w_{pd} of devices before the switching transistor in the signal path

$$C_{total} = c_{pddriver} + c_{pdafter} + C_L$$

$$c_{pddriver} = c_1 w_{pd}$$

$$c_{pdafter} = \sum_{after} c_2 w_{pdi}$$

Delay until output begins to fall

$$T_{BEFout} = T_{BEFO} + mT_{SWRin}$$

T_{BEFO}: use moderate input slope approximation

$$= a_1 + b_1 \frac{1}{g_m} C_{total}$$

$$\rightarrow a_1 + b_1 \frac{1}{w_{pd}} C_{total}$$

m: t_{IL} effect plus moderate input slope behavior

$$= d_1 + d_2 \frac{g_{pu}}{g_{pd\,total}} + d_3 \frac{C_{total}}{g_{meff}}$$

$$\rightarrow d_1 + d_2 \frac{S_{pu}}{w_{pd\,total}} + d_3 \frac{C_{total}}{w_{pd}}$$

Switching time of falling output transition

$$T_{SWFout} = T_{SWFO} + mT_{SWRin}$$

T_{SWFO}: a_1 + Elmore delay approximation with $r_{pd} = \frac{b_1}{w_{pd}}$ for each conducting pulldown

m: very slow input behavior plus moderate input slope behavior

$$\rightarrow d_1 + d_2 \frac{1}{g_{meff} r_L} + d_3 \frac{C_{total}}{g_{meff}}$$

$$\rightarrow d_1 + d_2 \frac{S_{pu}}{w_{pd}} + C_{total}\left(\frac{d_3}{w_{pd}} + \frac{d_4}{w_{pd\,before}}\right)$$

Delay until output begins to rise

$$T_{BERout} = T_{BERO} + mT_{SWFin}$$

T_{BERO}: use very fast input slope approximation

$$= a_1 + b_1 r_{pu} C_{total}$$
$$\rightarrow a_1 + b_1 \frac{1}{S_{pu}} C_{total}$$

m: t_{IH} effect plus moderate input slope behavior

$$= d_1 + d_2 \frac{g_{pu}}{g_{pd\,total}} + d_3 \frac{C_{total}}{g_{m\,eff}}$$
$$\rightarrow d_1 + d_2 \frac{S_{pu}}{w_{pd\,total}} + d_3 \frac{C_{total}}{w_{pd}}$$

Switching time of rising output transition

$$T_{SWRout} = \frac{(T_{SWRO} - T_{SWRO}^{asym})^2}{(T_{SWRO} - T_{SWRO}^{asym}) + mT_{SWFin}} + (T_{SWRO}^{asym} + mT_{SWFin})$$

$T_{SWRO} = a_1 +$ Elmore delay approximation with $r_{pu} = \frac{b_1}{S_{pu}}$

$m =$ same as T_{SWFout}

$$T_{SWRO}^{asym} = e_1 + e_2 T_{SWRO}$$

2.4.2 Input Capacitance

As we have seen, the addition of extra transistors to form more complicated logic functions has an effect on a gate's output response. We have examined the effective capacitance of an nMOS inverter during the beginning and switching phases of the rising and falling input. Of these four modes, only one depended on the output waveform. The input capacitances during the beginning portion of the input transition had no dependence because the output was still stationary. The capacitance during the switching portion of a falling input had none because the average input to output coupling capacitance dropped as the input waveform slowed down,

leading to no net Miller effect. Hence the input capacitance for these three modes depends only on the pulldown transistor's size, being proportional to the transistor's width (assuming a fixed channel length). Only the switching portion of the rising input possesses a significant output waveform dependence. To account for this dependence we must analyze the conducting path containing the switching transistor. Figure 2-26 shows an abstract gate model along with its circuit level representation. We model 'on' transistors as resistors (linear region approximation) and have added the appropriate capacitances from nonconducting paths to the total load capacitance.

We find that r_b causes a significant drop in the input capacitance. This fact has been exploited for many years by amplifier designers to raise input impedance and thereby improve the transfer characteristic. For very fast inputs, the output remains stationary throughout the input transition. We have

$$C_{SWROin} \approx \frac{c_{gs}}{1 + g_m r_b} + c_{gd}(1 + g_m r_a)$$

Since the switching transistor is saturated during the input transition, we know $c_{gs} \gg c_{gd}$ and can simplify the expression to

$$C_{SWROin} \approx \frac{c_{gs}}{1 + g_m r_b} + c_{gd}$$

For very slow inputs node a will have dropped to V_{OL} by the completion of the input transition and we may write

$$C_{SWR\infty in} = c_{gs} + 2c_{gd}$$

The slope m is affected in a variety of ways. The effective g_m modifies the C_L / g_m term. The gate's trigger point voltage is higher due to the resistance r_b, and the resistances also cause the active transistor's v_{DS} to be lower during the input transition. These two effects imply that the transistor is further in its linear region during the input transition and hence c_{gd} is larger and the Miller effect increases. Also, the resistance r_b lowers C_{SWR0in}, while $C_{SWR\infty in}$ remains unchanged. Hence the total capacitance increase during the transition region from fast to slow inputs grows, leading to a larger slope.

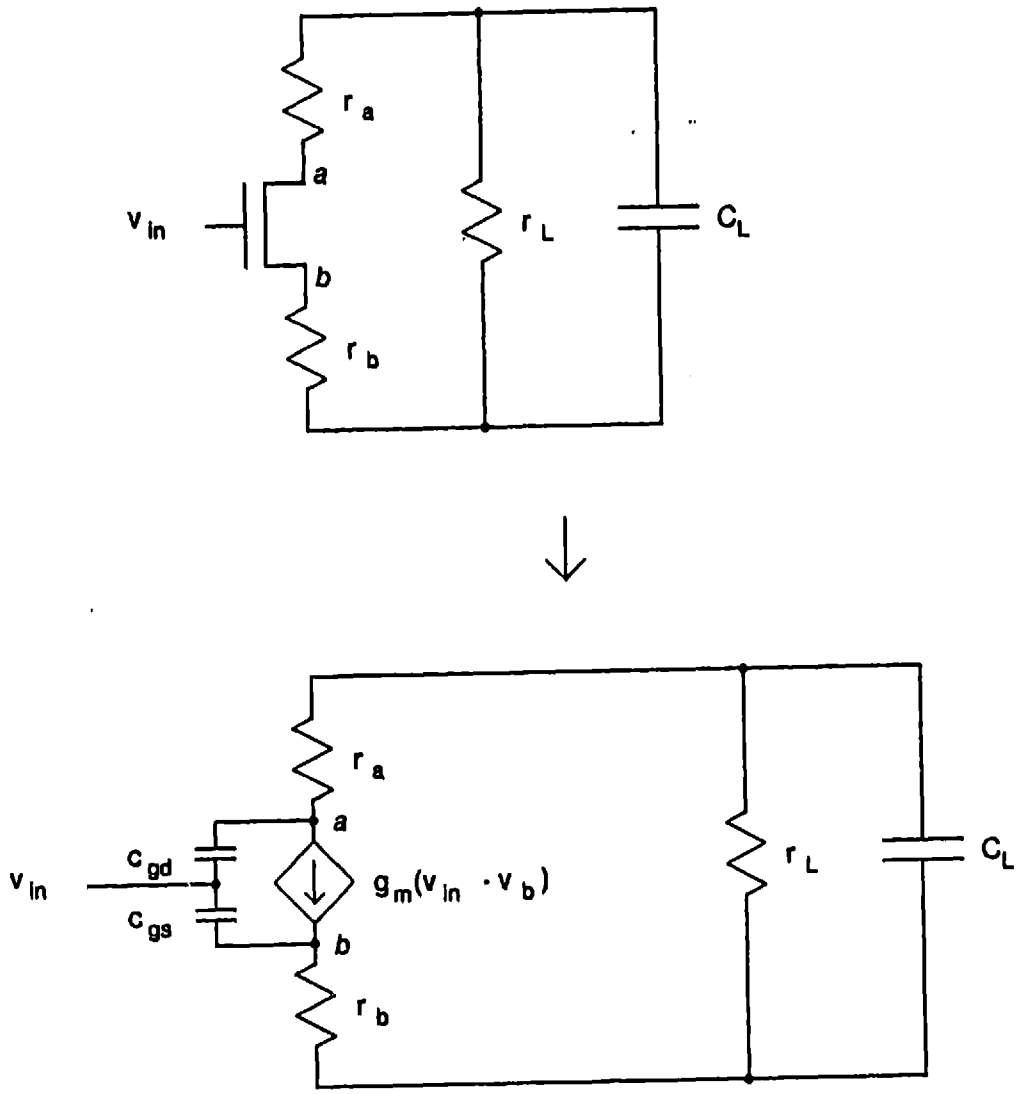


Figure 2-26: Circuit Model for Input Capacitance

We could add terms to the macromodel equation to account for each of these influences, but the additional accuracy scarcely justifies this in light of the computational overhead that would be incurred. Moreover complicated equations are often notoriously good at confusing curve fitters by enduing the fitter's error function with many local minima, rather than a single, global minima. Consequently the fitter may converge to a local minimum which could be far from the global minimum. We instead exercise prudence and focus on the dominant effect, the total

change in input capacitance during the transition region, and add an r_b term to our equation for the slope.

Our analysis yields the following equation for CSWRIn:

$$C_{SWRIn} = C_{SWROIn} + \Delta C_{SWRIn}\left(\frac{m T_{SWRIn}}{\Delta C_{SWRIn} + m T_{SWRIn}}\right)$$

$$\Delta C_{SWRIn} = C_{SWR\infty In} - C_{SWROIn}$$

$$C_{SWROIn} = \frac{c_{gs}}{1 + g_m r_b} + c_{gd} - \frac{a_1 w_{pd}}{1 + a_2 w_{pd} / w_{pdbefore}}$$

$$C_{SWR\infty In} = c_{gs} + 2c_{gd} \rightarrow b_1 w_{pd}$$

$$m = w_{pd}\left(c_1 + \frac{g_m}{C_{total}}\left[c_2 + c_3 \frac{r_b}{r_{pd}}\right]\right)$$

$$\rightarrow w_{pd}\left(c_1 + \frac{w_{pd}}{C_{total}}\left[c_2 + c_3 \frac{w_{pd}}{w_{pdbefore}}\right]\right)$$

Sample curves for a two-input NAND gate are shown in Figures 2-27 and 2-28. These curves illustrate the influence of the bottom pulldown transistor's resistance. In comparison with the capacitance curves for the bottom input, those for the top have a lower C_{SWROIn} and a steeper transition slope.

2.5 Implementation

We have developed a general purpose macromodeling software package. The modeler is called *Miranda*, which stands for *Modeler for Improved Range and Accuracy*. It processes cell template files, a macromodel control file, and macromodel equations. Each cell template file contains a logic cell's general topology. The macromodeler inserts values for device sizes, capacitive loads, and input waveforms into the template, and then runs SPICE on the resulting

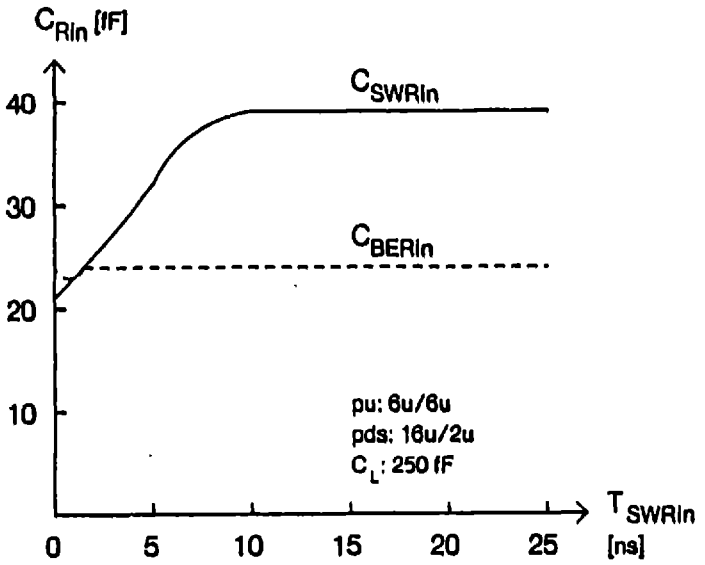


Figure 2-27: Input Capacitance for the Top Input of a NAND Gate

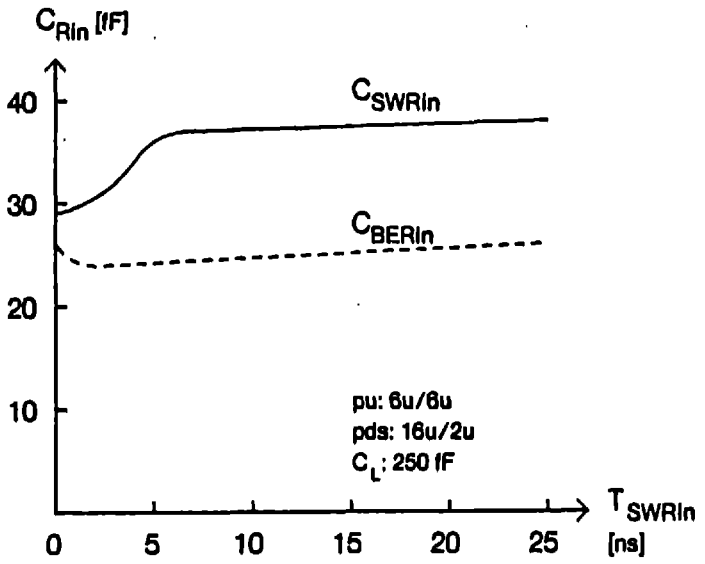


Figure 2-28: Input Capacitance for the Bottom Input of a NAND Gate

circuit. The values of the input capacitance and output waveform are extracted from the SPICE output and stored. This procedure repeats for every combination of device sizes, loads, and input waveforms specified in the control file. At present 216 SPICE runs are performed for the general logic gate analyzed in this chapter. The particular logic cells used are inverters and NAND gates. Owing to the simplicity of the cells, the SPICE simulations are quite fast, each requiring about ten cpu seconds on a DEC 20/60.

Once the data points have been obtained, the macromodeler solves for the parameters in the macromodel equations by using nonlinear curve fitting algorithms. We minimize the sum of squared error; minimizing the maximum error might also be acceptable but it is too sensitive to noise in the data. The curve fitter uses a Davidon-Fletcher-Powell algorithm [17] with modifications to accept upper and lower bounds on the parameters [18]. This is essential for ensuring that the final equations make physical sense. Otherwise local minima in the error function could draw the curve fitter toward nonphysical values for the parameters. Local minima in the error function also mandate that higher order effects be successively included in the model equations. That is, we solve for the first order terms in the equations first and then progressively solve for higher order terms. For example, when curve fitting the macromodel equation for output switching times, we first start with the simple RC model. We select a subset of data points with fast inputs and large capacitive loads—those points for which the model is most accurate—and solve for the RC terms in the equation. We lock these parameters and then solve for the waveshape terms. Next we solve for self-capacitance terms. Finally we unlock all parameters and curve fit again. This technique helps to guarantee that we reach the global minimum of the error function, and adds very little to the total computation time because the time is dominated by the SPICE runs.

The modeler is written in a computer language called CLU [19]. It consists of SPICE interface, minimization, and matrix manipulation program modules. These modules contain 3200, 1800, and 1000 lines of code, respectively. All told, the modeling support routines comprise about 6000 lines of CLU code; the modeling programs specific to the general logic gate discussed in this chapter represent an additional 1700 code lines.

Pertinent curve fit statistics are shown in Table 2-2. The macromodel equations are typically within several percent of SPICE predictions, a major improvement over RC models. These benefits come at a small price in computational overhead because we have modeled the response

of the entire cell, rather than using a more sophisticated transistor-model and then having to compute the transistors' interactions to obtain the cell's response. The accuracy and computational speed of the macromodels make them well suited for both simulation and optimization applications.

Rising Input, Falling Output			*Falling Input, Rising Output*		
Model Eqn	% Error		Model Eqn	% Error	
	ave	max		ave	max
C_{BERIn}	1.5	5.6	C_{BEFIn}	1.5	6.9
C_{SWRIn}	3.7	12.3	C_{SWFIn}	1.3	9.7
T_{BEFout}	5.7	18.3	T_{BERout}	4.6	13.2
T_{SWFout}	8.6	27.8	T_{SWRout}	3.0	10.6

Table 2-2: Macromodel Curve Fit Accuracies

CHAPTER THREE

Optimization

3.1 Overview

In this chapter we will present the theory and implementation of our optimization algorithms. Our emphasis is on developing basic understanding and intuition rather than a rigorous mathematical structure via formal proofs. The theory of nonlinear optimization is fairly extensive and we cannot discuss all of it here; readers interested in more detailed treatments are referred to [20], [21], [22], and [2]. We begin with a description of techniques for solving unconstrained minimization problems. After next covering the special characteristics of our optimization problem, for these always have a significant impact on the algorithm chosen, we move on to constrained problems. We examine several approaches to constrained nonlinear optimization and select one which is ideally suited to our problem. We follow the theoretical development with a description of implementation issues and close with a summary of the advantages and disadvantages of our approach along with results from several circuits.

3.2 Unconstrained Minimization

Unconstrained minimization problems are of the form

$$\min f(\mathbf{x})$$
$$\text{subject to } \mathbf{x} \in \mathfrak{S}$$

where f is a real-valued function, $\mathfrak{S}$ is (for this unconstrained case) $\mathbb{R}^n$, and n is the dimension of the parameter vector. Typical applications are curve fitting, where an error function is to be minimized, or speed critical circuit designs, where the signal propagation delay is to be minimized.

The most straightforward case occurs when the dimension of the vector x is one. We shall treat f as a function of a scalar by examining its dependence on the component of the vector x.

Consider the example shown in Figure 3-1. We lie at some point x_k and wish to generate a next point x_{k+1} that is closer to the minimum x^*. To determine the new point x_{k+1} we must select a search direction and a step size to move in that direction. It is clear that the direction should be governed by the sign of the derivative of f. This choice of direction is called a steepest descent algorithm. In our example the derivative at x_k is negative so we move in a positive direction, in this case to the right.

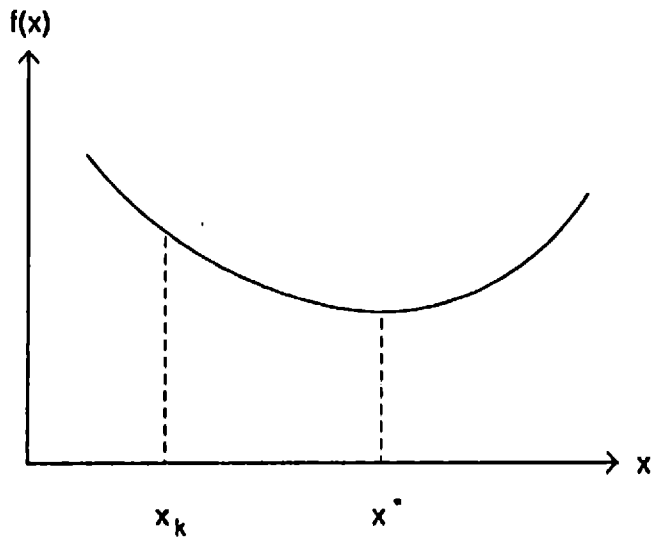


Figure 3-1: Minimizing a Function of a Scalar

Once the search direction is known, the step size is computed by probing in the search direction. A sequence of successively distant points is probed until an interval containing the minimum is found. A typical procedure is illustrated in Figure 3-2, where a pattern of three points has been identified for which

$$a < b < c$$
$$f(a) > f(b) < f(c)$$

The next task is to predict where the minimum lies within the interval. Specifically we wish to find a next point x_{k+1} which is closer to the true minimum than the current point x_k; we do this by setting

$$x_{k+1} = x_k - \alpha_k \frac{df(x_k)}{dx}$$

and we must determine the step size α_k. Often one can use a quadratic or cubic approximation to the function f(x) along the search direction and then compute where the minimum of the approximation occurs. However these approximations require that the function f(x) be continuous and smooth. Moreover these step size formulas can be sensitive to computational noise (e.g., roundoff error) and may tend to behave erratically in regions where the second derivative of f(x) approaches zero. The golden section and Armijo step size rules are more robust, both in terms of the amount of computational noise that they can tolerate and the class of functions that they can address, but the rate at which they converge to the minimum is slower.

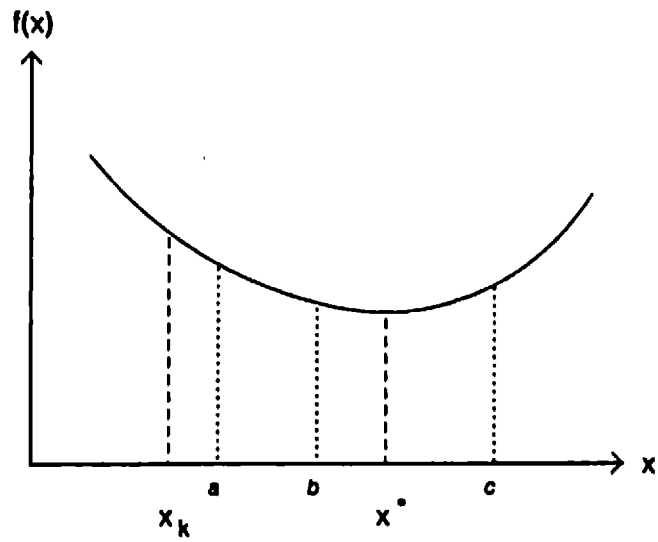


Figure 3-2: Finding an Interval Containing the Minimum

We can considerably improve the convergence rate by applying Newton's method to our problem. This method employs a quadratic approximation with second derivative information to compute x_{k+1}. We write

$$x_{k+1} = x_k - \alpha_k \left(\frac{d^2 f(x_k)}{dx^2}\right)^{-1} \left(\frac{df(x_k)}{dx}\right)$$

For a quadratic function the step size α_k is exactly one and the minimum will be reached in only one iteration. For nonquadratic but smooth f(x) a unity step size is usually a good initial guess. Note that while the method converges quickly, it finds a point where the first derivative is zero (a stationary point), which could be either a local minimum, local maximum, or saddle point.

Our discussion readily generalizes to multidimensional problems. The function f becomes a

function of a multidimensional vector x. Figure 3-3 displays an example of a two-dimensional problem. The ovals are contours of constant function values. The dot in their center is the function's minimum. The solid jagged line shows a typical sequence of points that might be generated by a steepest descent algorithm, which for multidimensional problems takes the form

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

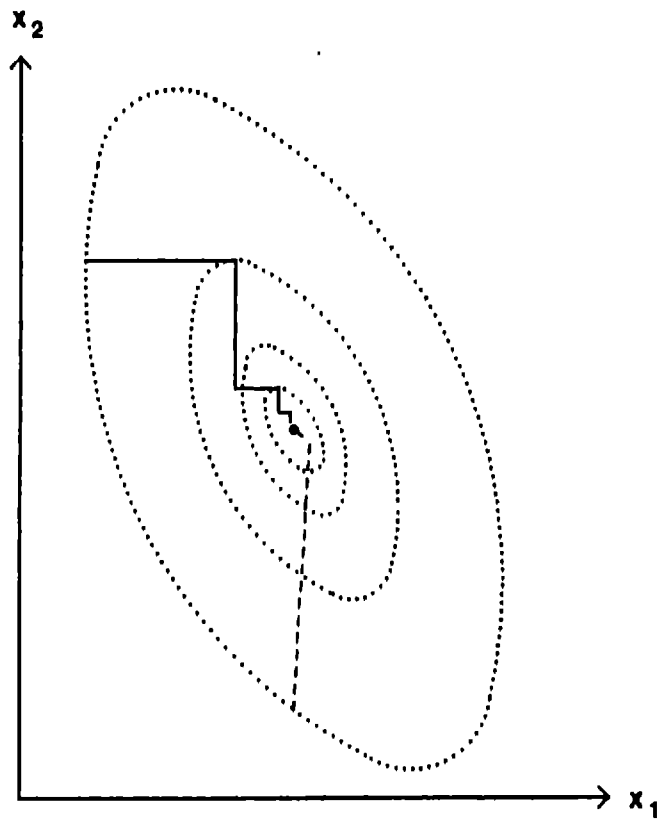


Figure 3-3: Minimizing a Function of a Vector

The zigzagging is due primarily to the simplicity of the method that generates the search directions. The steepest descent algorithm searches in a direction opposite to the gradient. These directions will point to the objective function's minimum only if its contours are circular. If the contours are elliptical, as they are in our example, the imprecision of the search directions will introduce zigzagging as shown, especially for functions whose contours are highly noncircular.

One can sometimes decrease the amount of zigzagging through careful choice of the parameters used by the step size algorithm. This routine seeks a minimum of the function in the

current search direction. Recall that the routine probes function values along the search direction and then provides an approximation of where the minimum lies. If the routine is not allowed to reach the minimum, but is instead restricted to an interval of points along the search direction prior to it, the amount of "overshoot" may be reduced. In practice one usually experiments with different step size parameters and then chooses the parameters that give the quickest convergence. While this may work reasonably well if the functions to be minimized have similar characteristics, it will not perform favorably for our circuit optimizer because the shapes of the functions are circuit dependent. There is no "typical" circuit for which one can tune the step size parameters.

Newton's method offers considerable advantages over the steepest descent algorithm. The dashed line in Figure 3-3 shows a typical trajectory. The method generalizes from the one-dimensional case to

$$x_{k+1} = x_k - \alpha_k F^{-1}(x_k) \nabla f(x_k)$$

where $F(x_k)$ is the Hessian (matrix of second derivatives) of f at x_k. By accounting for second derivatives the method provides better search directions and hence requires fewer iterations to converge than steepest descent. Note that for quadratic functions the minimum would be reached in one iteration with a step size $\alpha_k = 1$. For functions that are well approximated by a quadratic, and for regions in $\mathfrak{X}$ near a minimum, a step size of one is fairly accurate, or at least the step size routine can begin its search with a step size of about one. The ensuing reduction in the number of function evaluations has a favorable impact on computation time. One drawback is that the method requires computation of the inverse Hessian of f. However methods exist which will build an approximation to the inverse Hessian during the course of the minimization using only gradient information [17]. An additional advantage is that the algorithm can be modified to handle box and linear constraints[12] in the vector space $\mathfrak{X}$ [18].

[12] A box constraint stipulates that a component of x lie within some closed interval; i.e., *lower bound* $\leq x_i \leq$ *upper bound*. A linear constraint dictates that some linear combination of the components of x be nonpositive; i.e., $\sum_i a_i x_i \leq 0$.

3.3 Properties of Our Problem

Having acquired a rudimentary understanding of several key optimization issues, we move on to choosing an optimization technique that is appropriate for our problem. Selection of the technique is highly problem dependent, as "appropriateness" in nonlinear optimization is nearly synonymous with fast computation times, requiring that the optimization technique be closely matched with the problem's characteristics. We therefore take a short respite from our informal theoretical presentation and pause to consider the properties of our optimization problem. We shall see that the problem can be separated, allowing us to apply a divide and conquer strategy and thereby greatly increase computational speed.

We desire to minimize a circuit's power consumption subject to constraints on signal path delays and transistor sizes. The objective function, total power, is the sum of the power consumptions of each circuit cell. Moreover for nMOS the power consumption of each cell is solely determined by, and is linear in, the shape factor of its pullup transistor. For CMOS the power consumption is linear in the capacitive loads that must be driven, which are due to the area of the transistor gates and interconnect capacitance, but also depends somewhat on the input waveforms. Hence for nMOS, and nearly for CMOS, the power consumption of a circuit is a separable function. That is, it is the sum of independent terms from each cell, and can be expressed in the form

$$P_{total} = \sum_{i=1}^{n} P_i$$

where P_i is a function of cell i only and models cell behavior for the fabrication process' worst case power consumption parameters.

The problem's constraints are of two varieties: worst case delay specifications and transistor size design rules. The delay along a signal path is a nonlinear function of the circuit's transistor sizes and is nonlocal, being composed of contributions from each cell along the signal path. These contributions are not entirely independent, but fortunately transistor sizing is very nearly a separable operation, because both waveshape and capacitive loading effects diminish rapidly with electrical distance. Consider the inverter chain shown in Figure 3-4. Whether the input signal is slow or fast, by the time the waveform has propagated to the chain's output its shape will be predominantly determined by the last gate in the chain. As we saw in the macromodeling chapter, fast inputs put the gate in an RC response mode where the output

waveform's switching time is governed by the gate's effective output resistance and capacitive load. Slow inputs place the gate in a gain limited mode where the gate's gain increases the sharpness of the waveform's transition. Thus a gate behaves as a crude wave shaper.

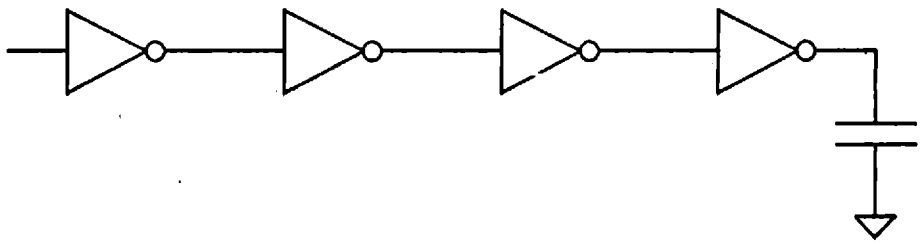


Figure 3-4: A Chain of Inverters

Capacitive loading effects also attenuate quickly with electrical distance. Suppose the chain is driving a large capacitive load. As every practicing engineer knows, the last gate will have to be fairly wide in order to drive the load. The second to last gate will in turn have to be somewhat large to drive the wide pulldown transistor of the last gate. We need progressively less widening as we work our way backwards from the load. Within a few gates we reach a point where we are fully shielded from the size of the load.

Transistor size design rules and layout considerations restrict the minimum and maximum sizes that a transistor can have, and for nMOS there is minimum beta ratio (ratio of pulldown to pullup shape factors) that a gate can have. The former is a box constraint; the latter is a linear constraint. The constraints on a circuit's transistors are entirely local to each cell and are therefore separable.

Accuracy requirements are also important and exhibit a peculiar ambivalence in our problem. The delay specifications on the signal paths must be met to the full accuracy afforded by the macromodels. However the power minimization is less critical. We can tolerate small errors in minimizing the circuit's power consumption, especially if the inaccuracies are accompanied by large savings in computation time. In fact, at present designers use only crude heuristics or almost ignore the power consumption issue entirely.

In summary the problem embraces characteristics ranging from the trivial to the extremely

difficult. The objective function is a simple summation of contributions from each logic cell, each contribution being linear in the cell's transistor sizes. On the other hand we anticipate hardship with the delay constraints since they are global and nonlinear in the circuit's transistor sizes. Fortunately there are not many of them; typically a designer will specify delays for only a few percent of the paths through a functional block. In contrast the transistor size constraints are quite simple, consisting of linear and box constraints. However there are a large number of them, at least one for every transistor in the circuit, carrying the potential for huge run times. The objective and constraint functions are essentially separable, linearly composed of nearly independent contributions from the circuit's cells. We would prefer a nonlinear optimization algorithm that can exploit this separability, pursuing a divide and conquer strategy where the problem is partitioned into many smaller subproblems. This segmentation is beneficial because with most optimization algorithms run times grow superlinearly with the number of design variables. Thus by breaking up a large problem, faster run times can be achieved. In particular, if the problem could be partitioned down to the cell level, the size of the vector space for each subproblem would be the number of transistors in each cell. Small vector spaces usually imply fast run times.

3.4 Constrained Minimization

Having examined our problem from a nonlinear optimization perspective, we now proceed to select a suitable algorithm. We commence by surveying several nonlinear optimization techniques. We describe their conceptual framework and their strengths and weaknesses in the context of our problem. Every technique has its pluses and minuses; none is a panacea. We shall choose the most appropriate technique, one which promises both computational efficiency and robustness.

3.4.1 Feasible Directions

The method of feasible directions has seen moderate use in circuit optimizers (such as DELIGHT, described in chapter one) and in nonlinear optimization as a whole. These algorithms are geared toward solving problems of the form

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{subject to } \mathbf{x} \in \mathfrak{S}, \text{ a subset of } \mathbb{R}^n \\ &\qquad g_i(\mathbf{x}) \le 0 \end{aligned}$$

Points in $\mathfrak{G}$ satisfying the constraints $g_i(x) \leq 0$ are said to be feasible. We wish to find the point that minimizes f within the feasible set. The basic idea is that given a point within the feasible set, find a search direction that will keep the next point within the feasible set (for a step size within (0, ε] for some $\varepsilon > 0$) while decreasing f. Figure 3-5 shows an example. The dotted lines are contours of f; the dashed lines are boundaries of the feasible set. We begin at a point x_0 and move toward the minimum of f in the feasible set, improving the intermediate solution at each iteration until the optimum is reached.

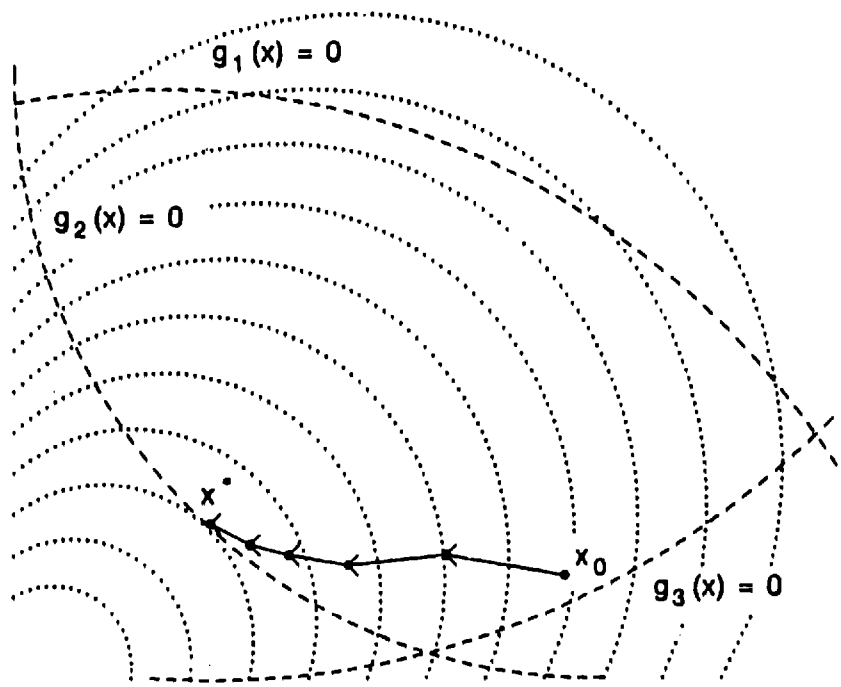


Figure 3-5: Feasible Directions Algorithm

The principal advantage of this scheme is that each design solution x_k is feasible. When satisfied with the circuit's performance, the designer is free to stop the procedure, knowing that all constraints have been met. This will be true even if the final solution has not yet been reached. This option is useful because often achieving the final solution is too expensive either computationally or in terms of the designer's time; the extra performance provided by the optimum simply does not warrant the additional expenditure of effort.

Feasible directions methods have a number of drawbacks, especially as regards our

problem. First, the algorithm does not partition the problem,[13] and so works in a very large vector space, in this case the number of transistors in the circuit. This implies long run times. Second, the control structures for these algorithms (at least for those dealing with nonlinear constraints) are rather complicated. The underlying reason for this is that, in order to guarantee convergence, the algorithm generates search directions that move away from nearly active constraints. This is necessary owing to the feasible set boundaries from nonlinear constraints. Due to their nonlinearity, these boundaries do not form straight lines and hence one cannot predict their behavior based only on their value and gradient at some intermediate solution point. There is a possibility that they may curve in and intersect the proposed search direction. This dilemma can be remedied by moving away from nearly active constraints. Of course eventually the final solution may lie on a constraint boundary; this is accomplished by reducing the threshold of what constitutes a "nearly active" constraint as the optimization progresses. The impact that this modification has on the algorithm's complexity is considerable. The control structure becomes quite complicated, and requires a fair amount of "magic numbers" (parameters whose values are crucial to determining computational speed; these numbers are usually tuned to a class of problems to provide reasonable run times, and if poorly tuned can sometimes cause the algorithm to fail entirely), and the convergence rate is only linear. Consequently the algorithm tends to be slow and may be unreliable, especially for large, complicated problems such as ours.

The method might not be able to reach the optimum if the feasible region is not a convex set. We show an example in Figure 3-6. Here the algorithm has stopped at the point $x^{\#}$ rather than reaching the true optimum x^{*}, because reaching the optimum would require a temporary drop in the objective function. This behavior is typical of numerical methods algorithms. The algorithms have only local information from the points that have been previously explored, and no global picture of the constraint and objective functions' behaviors over the entire feasible set. Consequently they have no a priori knowledge of where to look for the optimum. The alternative is to probe different points in the feasible set to find the best starting point, or to accept temporary losses in the objective function in the hope of accruing a large gain eventually [23, 24, 25]. These approaches tend to be quite expensive computationally, especially for vector spaces of many dimensions, and in practice the best technique is simply to have a designer specify a good starting point based on intuition and past experience. We also note that a sufficiently ill-behaved

[13] Of course one could always apply sparse matrix techniques in the actual implementation of the algorithm, but such an approach is not as computationally efficient as an algorithm which inherently partitions the problem.

objective or constraint function will break any optimization algorithm, causing it to reach a nonoptimal solution or fail to run at all. Different algorithms are sensitive to different aspects of ill behavior and manifest their confusion in different ways. As far as this goes, feasible direction methods tend to do rather well in design applications and have seen substantial use.

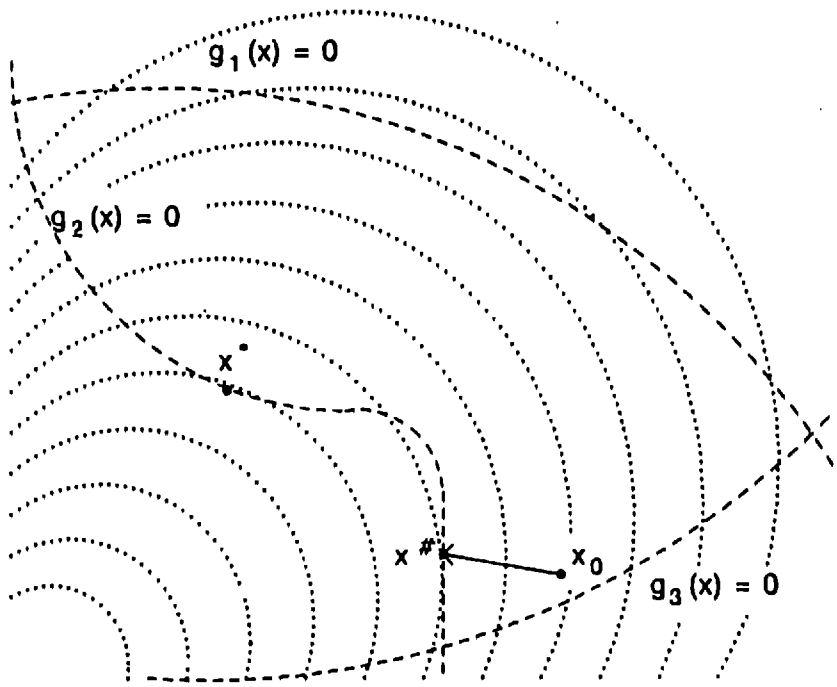


Figure 3-6: Feasible Directions Algorithm with a Nonconvex Set

Another drawback is that the method itself has no means of generating the initial feasible point. In general a separate mechanism must be provided to obtain an initial feasible point. The optimization is split into two phases. The first phase finds an initial feasible solution while the second performs the actual optimization. This dichotomy is not a hindrance if an initial solution exists. However for our problem it is likely that overly optimistic designers might specify path delays that cannot be met. In this case we would like the optimizer to do its best to meet those constraints while optimizing those paths with achievable specifications. Implementing this scheme in a feasible directions algorithm necessitates a blurring of the boundary between the two phases. This can have nightmarish consequences as far as the software implementation is concerned.

In summary the basic philosophy of the feasible directions method seems well suited to engineering design problems, and has led to its use by a number of workers [1, 2]. However its complexity and failure to partition the problem renders it somewhat inappropriate for problems involving a large number of parameters and constraints such as ours.

3.4.2 Penalty Methods

Penalty methods offer another means of solving constrained nonlinear minimization problems. The underlying concept is to express the problem as an unconstrained minimization by adding a penalty term to account for the constraints. For example, consider the problem with equality constraints

$$\min f(\mathbf{x})$$
$$\text{subject to } h_i(\mathbf{x}) = 0$$

We can convert this to an unconstrained minimization of the form

$$\min \{f(\mathbf{x}) + \tfrac{1}{2} c|h(\mathbf{x})|^2\}, \quad c>0$$

If we embed this minimization in an outer loop where we progressively increase the value of c, the constraint h(x) will be driven to zero to avoid having the penalty term $\frac{1}{2} c|h(\mathbf{x})|^2$ tend to infinity.[14] The algorithm readily generalizes to problems with inequality constraints.

These algorithms offer a number of advantages over feasible directions methods. First, the control structure for the algorithm is straightforward and uses very few magic numbers, allowing for a fairly robust implementation, capable of handling even very large problems. Penalty methods have been successfully applied to optimal control problems involving thousands of variables. Second, since the algorithms convert the problem to an unconstrained minimization, Newton and quasi-Newton methods are applicable. These second order methods expedite convergence. Third, the algorithms can be started at infeasible points; there is no need to precede the algorithm with a routine to find an initial feasible solution.

There are some disadvantages, however. In comparison with feasible directions methods, penalty functions do not necessarily generate a sequence of feasible solutions. Since the

---

[14] The penalty function method can be improved by augmenting it with Lagrange multipliers. This extended algorithm is called the method of multipliers [22]; it offers a quicker convergence rate and reduced sensitivity to roundoff error.

intermediate solutions might not meet specifications, the designer is obliged to wait until the algorithm converges to a final solution. This could entail a fair amount of computer time. Furthermore the convergence rate is sensitive to both the problem and the penalty function. In our example we used a quadratic penalty function. A different type, say the absolute value of the constraint cubed, would yield a different rate. Which function is most beneficial depends on the problem's objective and constraint functions.

In the context of our circuit optimization problem, the major defect is that the minimization is not separable. The objective function for our problem is power and the constraints include path delays. In the section covering the properties of our problem we saw that these are both separable. However, since the penalty function is nonlinear, the argument of the minimization is not separable. Consequently we are compelled to operate in the vector space corresponding to all of the circuit's transistors and cannot apply divide and conquer strategies. This not only has severe repercussions as far as computational efficiency is concerned, but also hampers the employment of quasi-Newton methods, since the inverse Hessian for the problem will be enormous, requiring large amounts of computation time for its calculation and memory for its storage.

In conclusion we see that penalty methods would be ideal were it not for the above-mentioned problem. This does not mean that these algorithms are impractical for our application; quite the contrary, they have seen much use in solving very large problems such as in optimal control. They are often the method of choice for solving large problems. However for our application there is a better method, one offering substantial improvements in computational efficiency. We describe this method in the following section.

3.4.3 Duality

Spurred on by the shortcomings of feasible directions and penalty methods, in this section we examine an alternative technique for nonlinear optimization. The technique is called duality, and is a fairly exotic approach in comparison to the other two methods. We shall see that the computational efficiency that it affords more than compensates for its conceptual complexity. Here we begin with an overview as a means of introducing the key issues. The basic idea of duality is to form a so-called dual problem which can be significantly easier to solve than the original, or primal, problem In our case the primal is difficult to solve because of the global, nonlinear delay constraints and large number of transistors to size.

Duality offers several major advantages. First, the primal problem need not be feasible. This is a strong possibility because high-performance designs often push circuit topologies to the limits of their performance. It is likely that a designer will specify delays for some signal paths which cannot be met. In this event we desire that our CAD tool do its best to meet those speed specifications while optimizing the power consumption of the other paths whose delay constraints can be satisfied. Duality achieves this goal. Second, inactive constraints pose no difficulty for duality. A designer specifies maximum delays along signal paths. Due to paths sharing common portions, it is possible that one path's delay specification will be exactly met while a companion path will be faster than required, and yet this situation minimizes power consumption. This is essentially a recasting of the critical path problem; the first path is one of the circuit's critical paths. Third, and perhaps most importantly, duality can be extremely efficient computationally. This is due to two factors. Duality converts the nonlinear delay constraints into simple box constraints, allowing us to apply fairly simple optimization algorithms (which implies robustness as well) with quasi-Newton methods. The quasi-Newton methods lead to fast convergence. Also, the dual approach permits us to exploit the separability of the power and delay functions, enabling us to use a divide and conquer strategy where each cell is optimized separately. Partitioning affords significant computation speed advantages. In subsequent sections we will examine the conceptual framework of duality and justify the preceding claims.

Like any nonlinear optimization approach, the advantages are balanced by drawbacks. Duality is not applicable to all problems; it works best for those satisfying a certain convexity requirement, a property which digital MOS circuits possess. Another drawback is due to our partitioning approach rather than duality itself. Although exploiting separability provides run time improvements, it necessitates the maintenance of additional data in the circuit's data base, along with a close interaction between the control structure and the data base. Partitioning the circuit into cells implies incremental optimization of each cell in succession. This leads to a sophisticated data base and even places profound requirements on the programming language chosen to implement the optimizer.

3.4.3.1 Lagrange Multipliers

Lagrange multipliers are the key to understanding duality. We shall explain their use and significance through a simple example. Consider the inverter chain shown in Figure 3-7. Here two inverters drive a capacitive load and we wish to place a maximum delay specification on the delay of a digital signal from the source v_{IN} to the chain's output. If we fix the width of the pullup

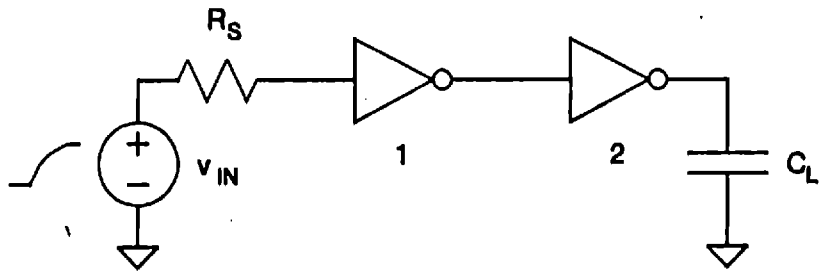


Figure 3-7: A Pair of Inverters

transistor, the length of the pulldown, and the beta ratio of the inverters, then we can treat the power consumptions of the inverters as the only free variables because specifying the power consumption of either logic gate uniquely determines the gate's transistor sizes. Let P_1 be the power consumed by the first gate and P_2 be that consumed by the second, and suppose we desire the total delay $T_{total} = T_{IN} + T_1 + T_2$ to be less than or equal to some T^*.

This maximum delay specification places restrictions on the allowable power consumptions of the gates. Certain regions in (P_1, P_2) space will not meet the speed specification. For instance if the shape factors of the transistors in the second inverter are too small, the inverter will not be able to charge the capacitor C_L quickly enough to satisfy the delay constraint. On the other hand, if the shape factors are too large, implying a wide pulldown and hence a large input capacitance, the first inverter will not be able to drive the second quickly enough. Of course the first inverter's shape factors can be made larger to drive the extra load, but after a certain point the first inverter's input capacitance becomes so large that the delay through R_S precludes meeting the delay specification. Since power consumption is linearly related to shape factor, the bounds on the shape factors imply bounds on the power consumption. Similar reasoning applies to the power consumption of the first inverter, giving us the forbidden zones shown in Figure 3-8.

We can more precisely characterize the feasible set of power consumptions. We do this by employing a simple RC model for the inverters, allowing us to derive an analytic expression for the delay through the gates as a function of their transistor sizes. The model equations are

$$R_{pu} = \frac{r_{pu}}{S_{pu}} \qquad R_{pd} = \frac{r_{pd}}{w_{pd}} \qquad C_{in} = c_{pd} w_{pd} \qquad P = p_{pu} S_{pu}$$

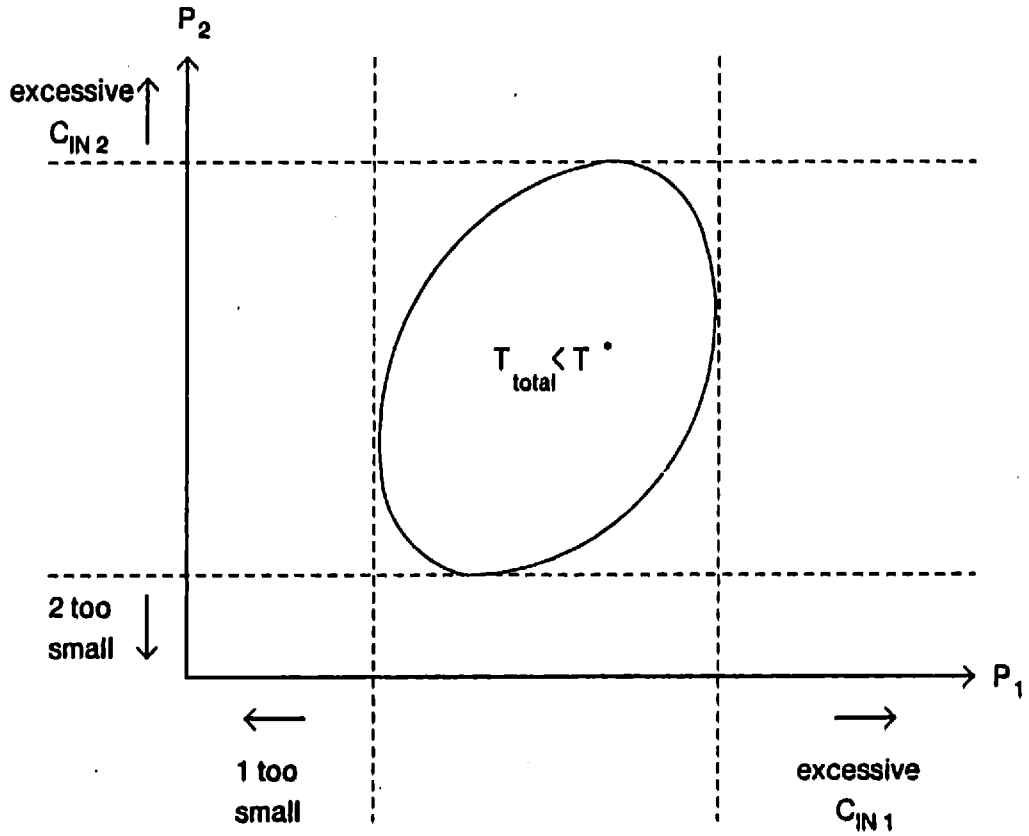


Figure 3-8: Delay Contour

Here R_{pu} and R_{pd} are the effective resistances of the pullup and pulldown transistors, C_{in} is the input capacitance, and P is the power consumption. S_{pu} and w_{pd} are free variables representing the shape factor of the pullup transistor and width of the pulldown transistor, respectively. The macromodel parameters r_{pu}, r_{pd}, c_{pd}, and p_{pu} depend on the fabrication process and power supply voltage. The total chain delay is the sum of the RC products through each stage. For a rising output we have

$$T_{total} = T_{IN} + T_1 + T_2$$

$$= R_S C_{in1} + R_{pd1} C_{in2} + R_{pu2} C_L$$

$$= R_S (c_{pd} w_{pd1}) + \left(\frac{r_{pd}}{w_{pd1}}\right)(c_{pd} w_{pd2}) + \left(\frac{r_{pu}}{S_{pu2}}\right) C_L$$

The resulting constraint surface $T_{total} = T_{IN} + T_1 + T_2 = T^*$ is elliptical as depicted in the figure.

We can also view the correlation between total power and path delay by adding contours of constant power to our figure. Figure 3-9 displays the result. To meet the delay constraint we must stay within the elliptical region, but the total power dissipation varies with position in the region. As we move toward the upper right of the feasible set, the power dissipation increases. At the point *Max* we have reached the maximum power consumption that will still allow us to satisfy the delay constraint. The delay and power contours are tangent and their gradients point in the same direction. If we instead work our way toward the lower left of the feasible set, the total power dissipation decreases. When we reach the point *Min* the dissipation will be at its lowest level that will still satisfy the delay constraint. Here the delay and power contours are again tangent, but now their gradients point in opposite directions.

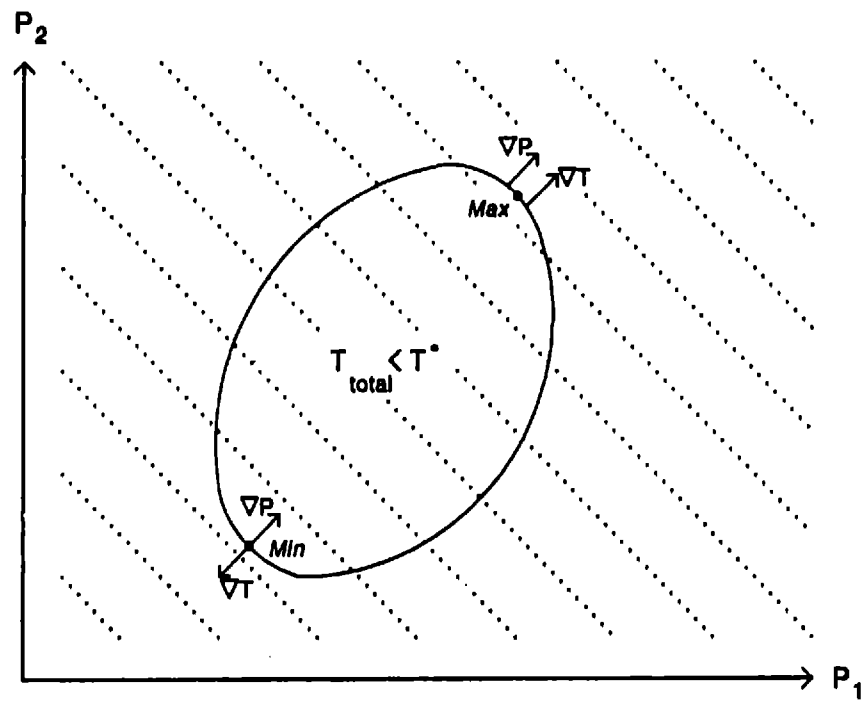


Figure 3-9: Contours of Delay and Power

We wish to find this point of minimum power consumption. The behavior of the power and delay gradients provide a means of identifying the point. We have seen that the gradients point in opposite directions; mathematically this can be expressed as

$$\nabla P = -\mu \nabla T$$
or $\nabla (P + \mu T) = 0$
where $\mu > 0$

The variable μ is called a Lagrange multiplier and offers the key to solving our nonlinear optimization problem. In the following sections we will discover how to apply this key and also see that the Lagrange multiplier has an intuitive meaning which conveys much insight regarding the optimization problem.

3.4.3.2 Finding the Optimum

We can acquire an understanding of how to find the optimum by using a graphical approach. We commence by revisiting the two-inverter chain of the previous section. Now, however, we are interested in the possible total power and total delay combinations that the circuit can exhibit. In other words, we desire the locus of points (T_{total}, P_{total}) that will be generated if we substitute all valid transistor size combinations into the circuit. This locus of points is denoted the set of all possible pairs, $\mathcal{P}$, and is displayed in Figure 3-10. The set's lower left boundary is the classic power-delay tradeoff curve (bold line); it represents designs that offer propagation delays with the lowest possible power consumption for those delays. Points toward the left of the curve are in the high-speed, high-power region. As we move down the curve to the right, we trade off speed for reduced power consumption and eventually enter the low-power, low-speed region.

Points that are not on the tradeoff curve correspond to nonoptimal circuits. These circuits either consume more power than an optimal circuit with the same delay, or are slower than an optimal circuit with the same power consumption. For example, suppose the inverter chain is driving a large capacitive load. We should make the second inverter's shape factors relatively large in order to drive the load, and then make the first inverter slightly large to drive the wide pulldown of the second inverter. If due to some confusion we reverse the ordering, making the first inverter very large rather than the second, the circuit will still consume the same amount of power as the optimal one, but will be considerably slower.

Our delay specification places a restriction on the points that we can accept. Specifically it stipulates that the total delay be less than or equal to T^*. We can focus our attention on this subset by shifting the vertical axis as shown in Figure 3-11. Points to the left of the axis have delays which are faster than T^*; hence this subset is called the feasible region. The optimum is the point in this region with the lowest power consumption, and is located at (0, P^*) in the figure.

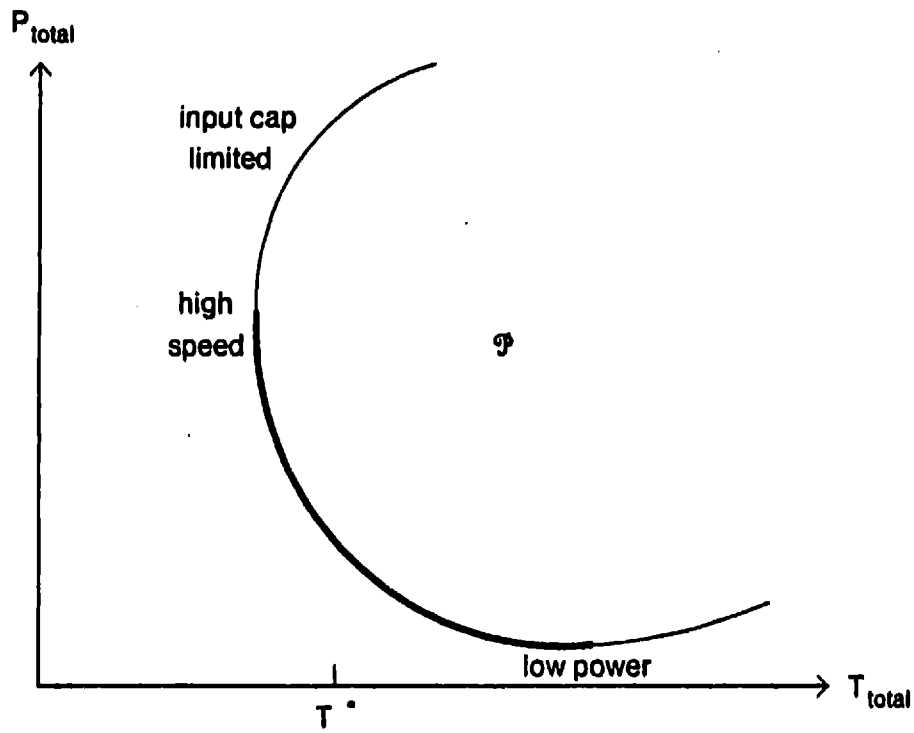


Figure 3-10: Set of Possible Pairs

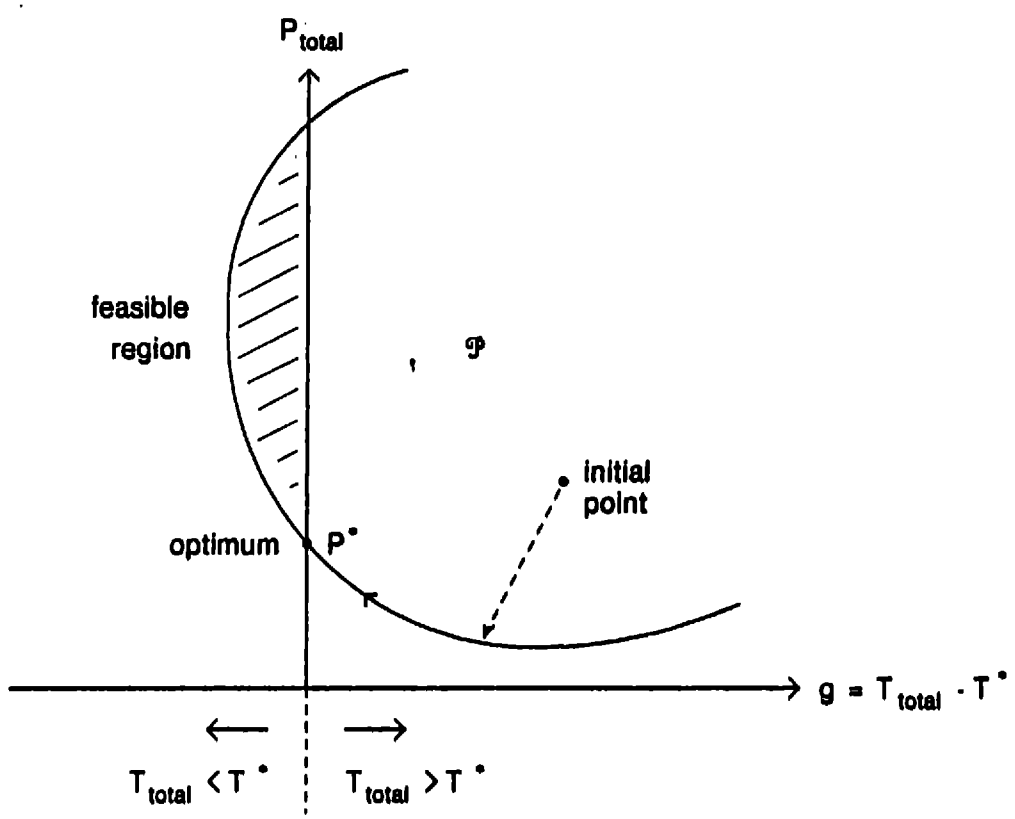


Figure 3-11: Reaching the Optimum

We must somehow reach this optimum point starting from an arbitrary point in the set $\mathcal{P}$. The approach duality takes can be thought of as a two-step process, as illustrated in the figure. The first step is to move to and remain on the lower boundary of $\mathcal{P}$. The second is to walk along this boundary to the optimum. Note that while conceptually this process may be interpreted as two steps, it must be implemented as an inner loop embedded in an outer loop. Step one corresponds to the inner loop and step two to the outer. This forces the search to follow along the lower boundary of the set.

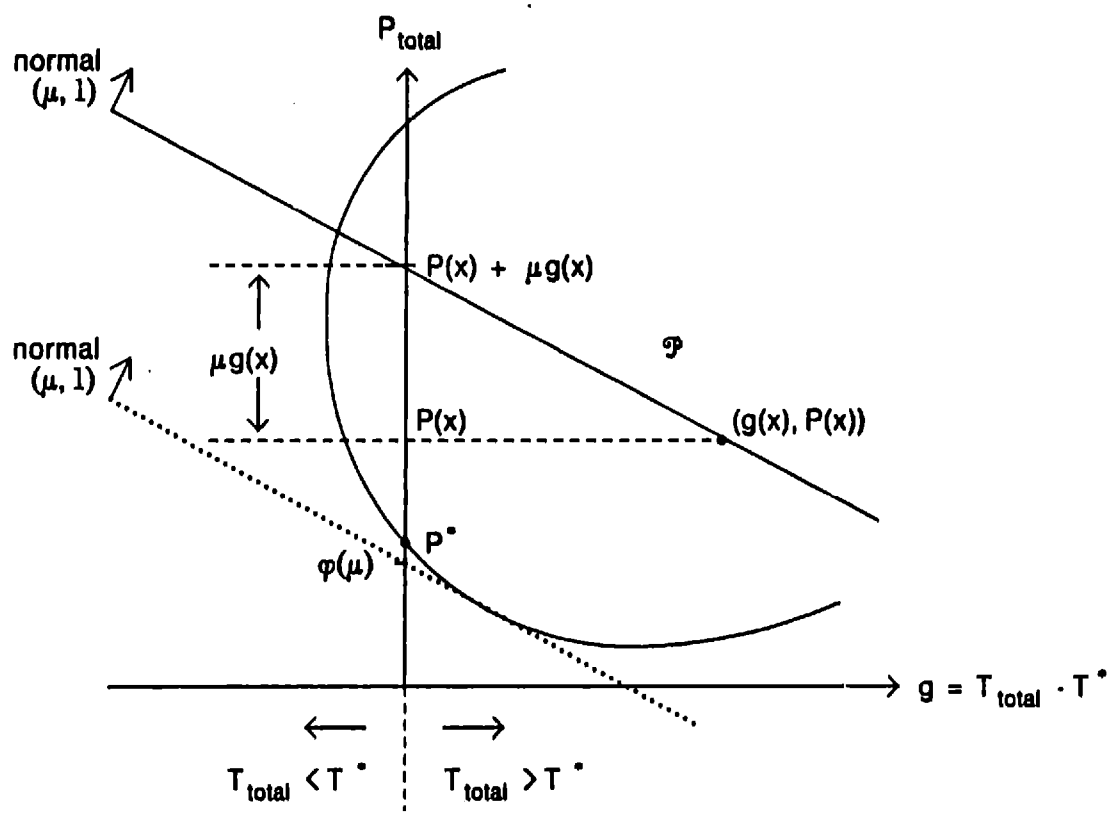


Figure 3-12: Inner Loop Minimization

We will now describe the implementation of each loop. Figure 3-12 gives a graphical representation of the inner loop. Suppose that we begin at some arbitrary assignment **x** of transistor sizes, with some arbitrary nonnegative Lagrange multiplier μ. The transistor sizes **x** map to point (g(x), P(x)) in g-P space. We can move from this point to the lower boundary of the set of possible pairs by sliding the solid line down until it is tangent to the bottom of $\mathcal{P}$, while preserving the slope of the line. By geometry we know that a line through a point (g(x), P(x)) with normal (μ, 1) intersects the vertical axis at P(x) + μg(x), and the multiplier μ fixes the slope of the

line. Hence this sliding operation is equivalent to bringing the vertical intercept down while holding μ fixed. In other words, we must perform the minimization:

$$\min \{P(\mathbf{x}) + \mu g(\mathbf{x})\}$$
subject to $\mathbf{x} \in \mathfrak{G}$, the set of valid transistor sizes,
with μ fixed

We shall denote the argument of the minimization as $L(\mathbf{x}, \mu)$, the Lagrangian, and the minimum intercept as $\varphi(\mu)$, the dual functional. Note that since the new circuit produced by this minimization maps to the lower left boundary of $\mathfrak{P}$, the circuit is well designed in the sense that it consumes the minimum power for the speed that it offers. Furthermore, since the minimization finds a point where the Lagrangian's gradient with respect to x is zero, we have $\nabla_x [P(\mathbf{x}) + \mu g(\mathbf{x})] = 0$. This is equivalent to the optimality condition we derived in the preceding section, indicating that the circuit's power and delay gradients point in opposite directions.

The outer loop walks along the lower boundary toward the optimum. We can gain insight into how this might be accomplished by contemplating the effect of different Lagrange multipliers on the inner loop's minimization. Figure 3-13 provides an illustration. We see that as we move toward the optimum point $(0, P^*)$ the intercepts $\varphi(\mu_i)$ increase in value until they reach P^*. Conversely, if we move away from the optimum in either direction, the intercepts $\varphi(\mu_i)$ decrease. We can therefore express this as a maximization:

$$P^* = \max \varphi(\mu)$$
subject to $\mu \geq 0$

The maximization gives us the Lagrange multiplier μ of the optimum, while the inner loop minimization provides the optimal transistor size assignments.

We are now in a position to grasp the intuitive significance of the Lagrange multiplier. From Figure 3-13 it is apparent that as μ increases, the line becomes more vertical, and we move up and toward the left. Power consumption increases whereas delay decreases. We are generating transistor size assignments that push the circuit topology harder for speed. The fact that the multiplier has a concrete, practical meaning is quite important, because it allows a designer to follow our CAD tool's "intent" as it optimizes a circuit, showing the signal paths that are the most troublesome in meeting the delay specifications. This knowledge is vital for directing efforts to improve the circuit, such as reduction of interconnect capacitance and modification of circuit topologies (e.g., the insertion of super buffers).

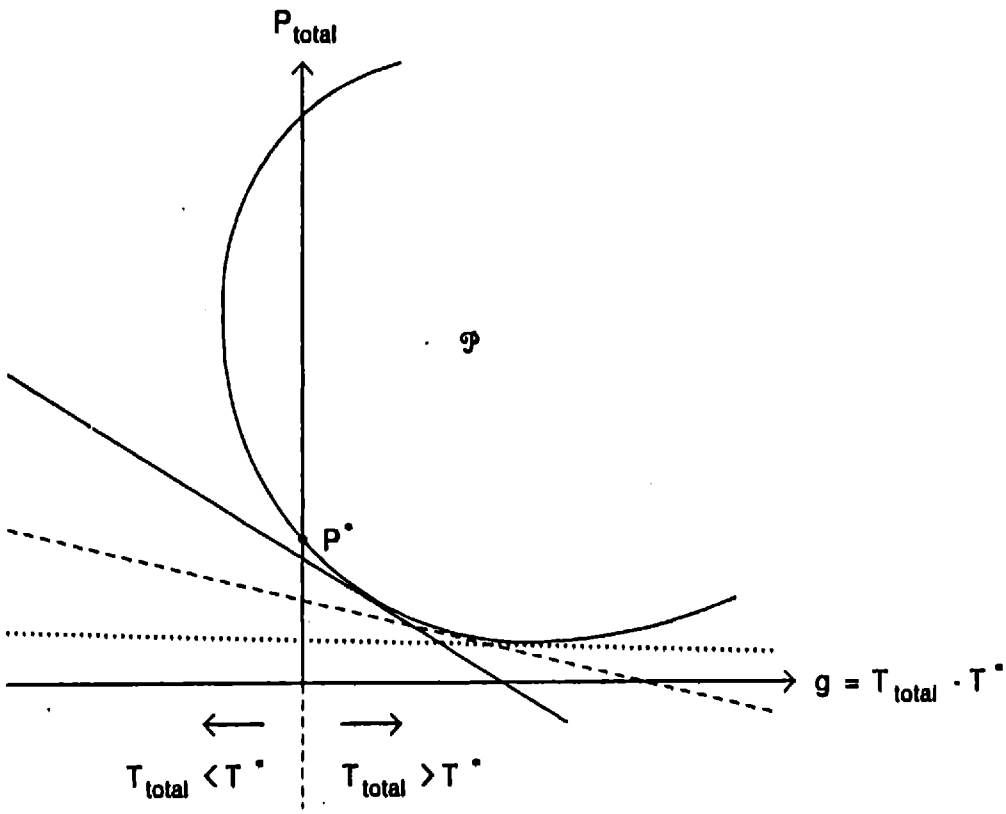


Figure 3-13: Outer Loop Maximization

3.4.3.3 Degenerate Cases

It is crucial that optimization algorithms perform properly even when faced with certain degenerate conditions in the delay constraints, such as inactive or infeasible constraints. Inactive constraints can come from one of two sources: (1) a delay specification on a signal path that is so loose that minimum size transistors along the path will satisfy it, or (2) interactions among paths give rise to a situation where meeting one path's constraint causes another's to be inactive. Of these two possibilities, the second is the most likely and occurs frequently in practice. For instance, peruse the circuit in Figure 3-14. If the delay specifications mandate that the top path be faster than the bottom path, then the top path will be the so-called critical path. This critical path will be optimized such that it precisely satisfies its delay constraint, while the bottom path will be somewhat faster than required. The bottom path could be slowed down by widening the path's pulldown transistor in the NOR gate, thereby impeding signal propagation due to the increased capacitance presented to the external driver. However, to do so may necessitate that the NOR

gate's pullup have a larger shape factor in order to satisfy a maximum beta ratio constraint.[15] Thus this would increase power consumption and hence be nonoptimal. The geometric representation in power-delay space is displayed in Figure 3-15, which shows a projection of the set of possible pairs onto the subspace of power and the second delay constraint. This constraint could be met exactly if the optimizer moves to the point $(0, P^{\#})$ on the vertical axis, but this would entail a higher power consumption than the optimum P^{*}. Regarding the implementation, the optimizer cannot reach the nonoptimal point $(0, P^{\#})$ because to do so requires a negative μ_2, and the outer loop maximization is confined to nonnegative Lagrange multipliers. Hence the optimizer could roll along the left lower boundary of $\mathcal{P}$, but would be forced to stop once it had driven μ_2 to zero.[16]

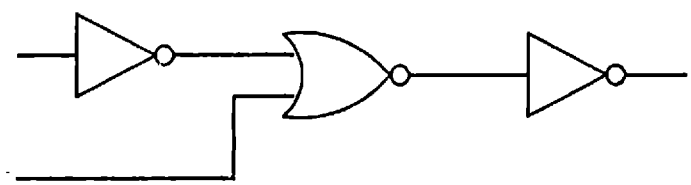


Figure 3-14: Circuit with an Inactive Constraint

Another important degenerate condition comes from infeasible delay constraints where the designer requests a maximum signal path delay that cannot possibly be met. These cases occur frequently in high-performance circuit design as the designer pushes a circuit topology and fabrication process to the limits of their performance. In these situations we desire that the optimizer do the best that it can, sizing those paths with infeasible constraints such that they switch as fast as possible, and sizing paths with feasible constraints such that their power consumption is optimized. Figure 3-16 offers an example. Here the dual algorithm drives the path's Lagrange multiplier toward infinity, sizing the transistors for maximum speed. Thus the algorithm gives useful feedback to the designer, indicating the maximum speed the circuit topology can provide.

---

[15] If a gate's beta ratio becomes too large, the trigger point voltage may be so low that the gate becomes a noise detector, responding to input perturbations that are only slightly above V_{OL}.

[16] Under certain conditions (e.g., signal paths for clocks) it might be desirable to exactly satisfy the delay constraint, even at a cost in power dissipation. In this case one could simply remove the nonnegativity constraint on the path's Lagrange multiplier.

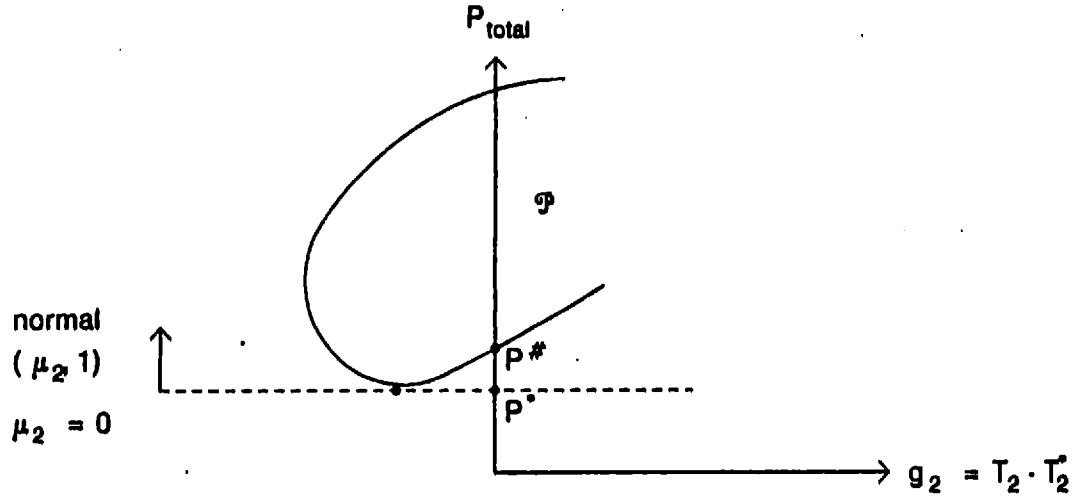


Figure 3-15: Effect of an Inactive Constraint

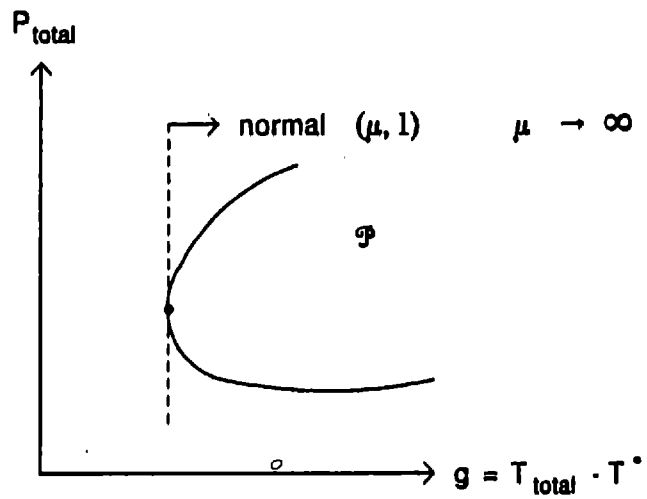


Figure 3-16: Effect of an Infeasible Constraint

3.4.3.4 Restrictions

As we mentioned at the beginning of our discussion, although duality does offer significant advantages over other optimization methods, it is limited in the scope of objective and constraint functions that it can solve. In particular, certain objective and constraint functions can produce a condition known as a duality gap, illustrated in Figure 3-17. In our previous examples we always portrayed the set of possible pairs as if it were convex. In general this need not be the case.

Certain objective and constraint functions can give rise to nonconvexities in the lower left boundary of $\mathcal{P}$. This can lead to gaps between the optimum value φ^* found by the dual algorithm and the true optimum f^*. The dual algorithm cannot reach the true optimum because the inner loop minimization brings the line down as low as possible on the boundary of the set. The minimizer cannot converge to the true optimum because, for any Lagrange multiplier, a line passing through the optimum passes through the interior of the set. The dual algorithm instead converges to either point $(g(x^+), f(x^+))$ or point $(g(x^-), f(x^-))$. In order to satisfy the $g(x) \leq 0$ constraint, one would implement the algorithm so that it converges to the former rather than the latter. The point is feasible but has a higher value for its objective function than the true optimum.

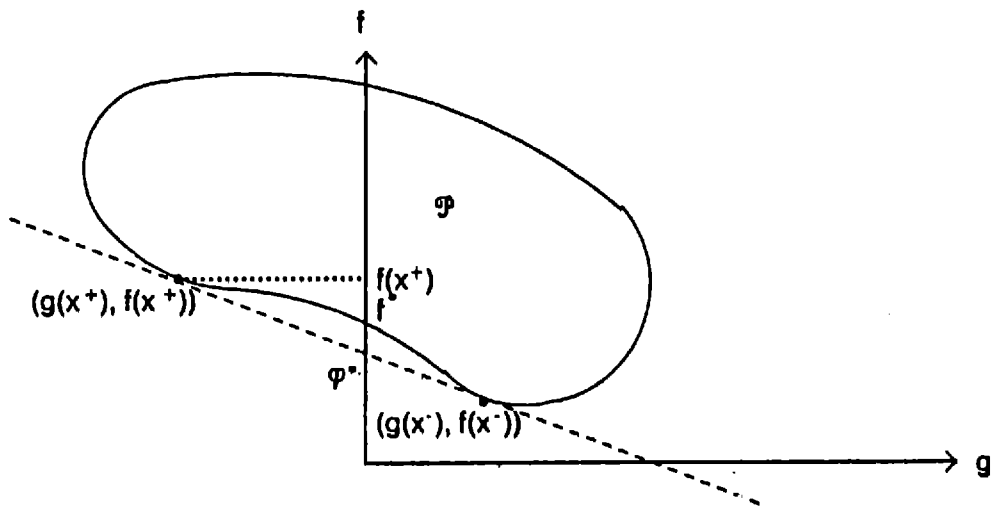


Figure 3-17: A Duality Gap

A good deal of real world problems do exhibit duality gaps. Despite this, many workers prefer to use the dual algorithm. The reasons for this are that the dual offers extremely fast computation times and in many cases the duality gap is negligibly small (if one exists at all). For instance, in one application involving optimal scheduling of electric power generation systems, the gap was only a few percent of the optimum [26]. This was a considerable improvement over the heuristic scheduling rules that had been previously employed. Other methods such as penalty functions would have been prohibitively expensive computationally and would not have significantly improved the solution owing to the smallness of the gap.

It can be proven that if the objective function $f(x)$ and the constraint functions $g_i(x)$ are convex, then $\mathcal{P}$ is convex and there is no duality gap. The opposite need not be true: nonconvex

objective or constraint functions do not necessarily lead to a duality gap. From the macromodeling chapter we know that our objective function, power consumption, is convex in the shape factors of the circuit's pullup transistors. The delay of a single logic gate is convex in the shape factors of the gate's pullups and in the widths of its pulldowns. However the expressions for the delay through a group of logic cells are too complicated to analyze to ascertain convexity. The complexity of nonlinear systems taxes one's ability to analytically derive results.[17] We remind the reader that duality is not solitary in its inability to find optima in the face of ill-behaved functions. Objective functions with local minima can cause any algorithm to drop into a local, rather than global, minimum. Also, as we saw in our discussion of feasible directions algorithms, nonconvex constraints can cause those algorithms to stop before they reach the optimum. Penalty method algorithms are susceptible to the same problem, depending on the penalty function used and the degree of nonconvexity of the constraints. In summary, sufficiently ill-behaved functions will confuse any nonlinear optimization algorithm (as well as any other optimization technique, such as heuristics, for that matter).

We have chosen an algorithm that is perhaps more sensitive to ill behavior in the objective and constraint functions, but is well suited to our particular problem because the power and delay functions for individual cells are convex and the functions describing groups of cells are nearly separable. In such a situation a duality gap, if it occurs, would be due to interactions among cells, but the separability arose from the fact that these interactions were small, and hence the duality gap must also be small. Our strongest evidence is that we have applied the optimizer to many circuits and have yet to encounter a duality gap.

If however a duality gap does occur we could bound its size. In Figure 3-17 we see that the true optimum f^* lies between φ^* and $f(x^+)$, the values of the dual functional and objective function, respectively, for the dual's solution.[18] Moreover it is easy to tell when we have a duality gap. Without a gap, we will either have $\mu_j > 0$ and $g_j(x) = 0$ (active constraint), or $\mu_j = 0$ and $g_j(x) < 0$ (inactive constraint).[19] If neither of these conditions are met, we have encountered a

---

[17] John Wyatt captured this quite succinctly as follows: "With linear system theory the ratio of elegance to applicability is almost infinite; with nonlinear systems the ratio is nearly zero."

[18] These bounds also apply for duality gaps with inactive constraints, where the true optimum lies in the left half plane rather than on the vertical axis.

[19] An infeasible delay constraint will lead to $\mu_j \rightarrow \infty$ and $g_j(x) > 0$. In practice one detects this by noting when μ_j has approached a large, positive upper bound.

duality gap. If the bounds on the gap are small, indicating that we are close to the optimum, then we can safely settle for a slightly inaccurate solution. If the bounds indicate that the gap might be large, then we could initiate another optimization algorithm such as the penalty method, starting at the dual's solution. This approach would probably be much faster than using the penalty method from the problem's inception. We doubt that such a two-stage optimization will ever be necessary because digital MOS circuits do not appear to exhibit any duality gaps, let alone large ones; nor do we anticipate that circuit designers will be eager to exchange a good deal of computer time for a few percent improvement in power consumption, especially since current design practice is to use simple heuristics to minimize power consumption when sizing transistors, a technique which is far less accurate than duality.

3.4.3.5 Summary

We have seen that duality offers unique advantages over other optimization approaches. Duality converts minimizations subject to nonlinear constraints into an inner loop minimization embedded in an outer loop maximization, both subject to simple constraints. These loops produce intermediate solution points that move along the bottom of set $\mathcal{P}$ to the optimum as depicted in Figure 3-18.

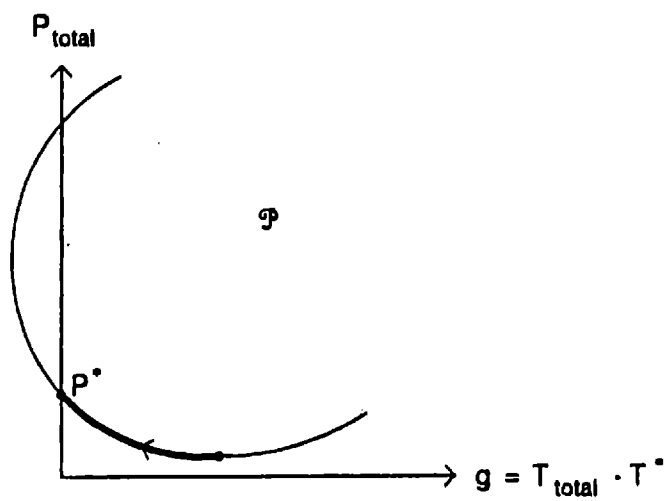


Figure 3-18: Trajectory of the Optimization

The inner loop brings the intermediate solutions down to the bottom of $\mathcal{P}$, and is expressed as

$$\varphi(\mu) = \min \{P(x) + \mu^T (T(x) - T^*)\}$$
$$\text{subject to } x \in \mathfrak{S}$$

Summing over each cell i, this can be manipulated to be

$$\min \{\sum P_i(x) + \sum \mu_i^T T_i(x)\}$$
$$\text{subject to } x \in \mathfrak{S}$$

where P_i is the power consumption of cell i, and μ_i, T_i, and T_i^* are column vectors formed of components of the corresponding circuit vectors that pertain to cell i. We have eliminated the $\mu^T T^*$ term because it is irrelevant to the minimization, contributing a constant.

Moving the summation outside of the minimization,

$$\sum \min \{P_i(x) + \mu_i^T T_i(x)\}$$
$$\text{subject to } x \in \mathfrak{S}$$

This separability allows us to size each cell in succession, operating in a small vector space whose dimension is the number of transistors in each cell. Furthermore the set $\mathfrak{S}$ is defined by fairly simple constraints. The minimum and maximum transistor size restrictions contribute box constraints, while the minimum and maximum beta ratio specifications contribute linear constraints. Hence approximations to Newton's method can be applied, giving fast convergence.

The outer loop moves the intermediate solutions along the lower boundary of $\mathcal{P}$ to the optimum. This loop is a maximization of the form

$$\varphi^* = \max \varphi(\mu)$$
$$\text{subject to } \mu \geq 0$$

where a Lagrange multiplier has been assigned to each of the delay constraints. Duality has effectively converted the nonlinear delay constraints to simple box constraints on the multipliers. Moreover we have a fairly small vector space, whose size is the number of delay constraints, likely to be about ten to twenty for typical functional block designs. Quasi-Newton methods are again applicable. An added bonus is that the gradient of the dual, which is needed to compute search directions, turns out to be the value of the constraint functions; i.e., $\nabla \varphi(\mu) = T_{total} - T^*$. Since these values have to be computed to find the value of the dual anyway, we have acquired the dual's gradient at no extra expense in computation time.

The benefit that accrues from this divide and conquer approach and the small vector spaces is computational efficiency. (We shall see specific examples in the next section.) For our transistor sizing problem, duality is much faster than standard optimization approaches. While sparse matrix techniques could enhance the performance of these other methods, they would still be at a disadvantage because (1) duality preserves the separability of the problem whereas most other algorithms do not, hindering the application of sparse matrix techniques and (2) sparse matrix techniques must first determine how to partition the problem before solving it, whereas duality can directly exploit the separability inherent in digital MOS logic by breaking up the problem along logic cell boundaries.

3.5 Implementation

> *There are two ways to write a program. One is to carefully design it. The other is to start debugging a blank piece of paper.*
>
> —Dave Gifford

Having established the theoretical framework of duality, we now proceed to its implementation. We segment our discussion into several parts, examining first the control and then the data structures. We shall see that while duality lends itself to a rather simple control structure, the data structures needed to exploit separability are rather sophisticated, and have a significant impact on the computer language chosen to implement the algorithm. We conclude the section with examples of several circuits that have been optimized by our CAD tool.

3.5.1 Control

We have seen that duality maps the nonlinear delay constraints into simple box constraints on Lagrange multipliers. This mapping leads to simple, computationally efficient control structures. The outer loop maximization uses a Davidon-Fletcher-Powell quasi-Newton method [17] with modifications for the box constraints [18]. The Davidon-Fletcher-Powell algorithm is started with a diagonal approximation to the dual's inverse Hessian. The step size algorithm is somewhat nonstandard. Although calculation of the dual functional $\varphi(\mu)$ yields both its value and its gradient, permitting the use of a cubic step size rule with a theoretically excellent convergence rate, computational noise prevented its use. Instead a step size rule that inspects only the gradient of the dual is used. We have found the rule to be quite fast. As a whole the maximizer performs very well. The simple box constraints permit a straightforward control structure which is both fast and robust.

The inner loop minimization is slightly more complicated since it must handle linear as well as box constraints.[20] The box constraints arise from minimum and maximum transistor sizes; the linear constraints are due to beta ratios. We use a minimization algorithm due to Bard [21] coupled with a diagonal approximation to the Lagrangian's Hessian $\nabla^2_{xx} L(x, \mu)$. We employ a golden section step size rule. We experimented with a quadratic step size rule, but found it to be overly sensitive to roundoff error, especially when the Lagrangian was nearly linear in the direction of the search.

The implementation of the minimization exploits the separability of digital MOS logic. Rather than attempting to minimize the Lagrangian of the entire circuit simultaneously, each cell's Lagrangian is minimized assuming that its neighbors will not change. Interactions are propagated by a relaxation technique: we iterate the minimization of the circuit three times. This manipulation carries substantial advantages. Minimizing a cell's Lagrangian has become independent of the delay specification on paths through the cell, depending only on the values of the Lagrange multipliers and the boundary conditions (such as input waveforms and output loads) applied to the cell.

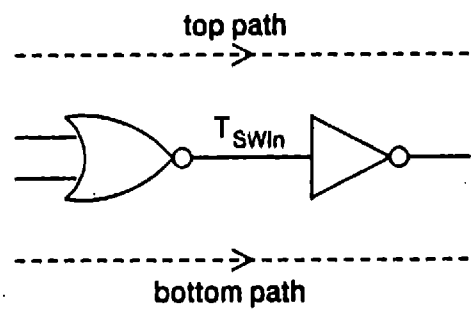


Figure 3-19: Boundary Conditions Applied to a Cell

An additional manipulation removes the dependency of the boundary conditions on the constrained paths. For instance, examine the subcircuit shown in Figure 3-19. When minimizing the Lagrangian of the inverter, we must account for both signal paths through the inverter. The

[20] Let the reader be warned that this minimization problem does not meet the so-called regularity requirement (stipulating that the gradients of the active constraints at a solution point be linearly independent) and hence most of the feasible directions algorithms for simple constraints in the literature are inapplicable here.

standard approach is to explicitly add $\mu_j (T_j - T_j^*)$ terms to the cell's Lagrangian for each constrained path j passing through the cell. As one might imagine, this approach typically leads to a rather complicated Lagrangian, especially considering that the logic cells driving the NOR gate could have many inputs, and the inverter's output may drive several paths as well. Hence the inverter could be included in many constrained signal paths; this implies a large amount of computer time. Eliminating the $\mu_j T_j^*$ term alleviates only part of the problem because the different paths present different boundary conditions; in our figure the switching time of the input transition, T_{SWin}, along the top path differs from that along the bottom path. We instead pose the following question: if we approximate and treat the inverter as if only one signal path passed through it, which boundary conditions will produce a solution that is close to the true optimum? If, immediately prior to the inverter's minimization, we linearly expand its delay T_{BEout} as a function of T_{SWin} about the current solution point (T^o_{SWin}, T^o_{BEout}), we can write

$$T^{approx}_{BEout}\{T_{SWin}\} = T^o_{BEout} + \frac{dT_{BEout}}{dT_{SWin}}(T_{SWin} - T^o_{SWin})$$

That portion of the circuit's Lagrangian which includes the inverter's interactions with the NOR gate is

$$\begin{aligned} L_{inverter}(\mathbf{x},\boldsymbol{\mu}) &= P_{inverter} + \mu_{top}(R_{BEin\,top}C_{BEin} + T^{approx}_{BEout}\{T_{SWin\,top}\}) \\ &\quad + \mu_{bottom}(R_{BEin\,bottom}C_{BEin} + T^{approx}_{BEout}\{T_{SWin\,bottom}\}) \\ &= P_{inverter} + (\mu_{top} + \mu_{bottom})(R^{ave}_{BEin}C_{BEin} + T^{approx}_{BEout}\{T^{ave}_{SWin}\}) \end{aligned}$$

Here T^{ave}_{SWin} and R^{ave}_{BEin} are weighted averages of the inverter's input boundary conditions from the two paths; i.e.,

$$T^{ave}_{SWin} = \frac{1}{\mu_{top} + \mu_{bottom}}(\mu_{top}T_{SWin\,top} + \mu_{bottom}T_{SWin\,bottom})$$

$$R^{ave}_{BEin} = \frac{1}{\mu_{top} + \mu_{bottom}}(\mu_{top}\left[\frac{\partial T_{BEin}}{\partial C_{BEin}}\right]_{top} + \mu_{bottom}\left[\frac{\partial T_{BEin}}{\partial C_{BEin}}\right]_{bottom})$$

Note that since the transistor sizes of the inverter's neighbors are fixed during its minimization, the average driver impedance R^{ave}_{BEin} applied to the inverter is nearly constant. However the input switching time T^{ave}_{SWin} varies because the inverter's input capacitance C_{SWin} changes as the inverter is sized, and the minimization must account for the resulting impact on signal delay.

Similar reasoning applies to the inverter's output boundary conditions. For general logic cells this technique is applied for every cell input to output path that is included in a constrained circuit path. This approach considerably reduces the complexity of the cell's Lagrangian. Consequently, when sizing a cell we avoid the computational cost of searching for and utilizing path data that is not local to the cell's data structure.

While this technique provides a tremendous savings in computation time, it does place rather elaborate demands on the data structures. When the minimizer sizes a cell, it is essentially performing an incremental optimization of the circuit. This is more involved than an incremental simulation. Information regarding boundary conditions imposed on a cell by its drivers and receivers must be maintained in the data base. It is this sophisticated data base with its closely coupled interaction with the control structure that makes the approach used by general purpose optimization tools inappropriate for our problem. As we saw in our review of tools such as DELIGHT and APLSTAP, these programs "black box" the circuit, interacting with it solely via a simulator. Hence the tools are in a sense isolated from the data base. They can neither access the circuit's connectivity description to guide partitioning nor embed additional information in the data base to assist the optimization. Their methodology does not support our incremental optimization approach with its accompanying savings in cpu time.

3.5.2 Data Structures

The data structures supply all the necessary circuit optimization information to the control routines. This task is more involved than it might appear at first glance due to the data structures' variety and representation. Contemplate the example in Figure 3-20. Three different structures are needed to describe the design: (1) a circuit descriptor describing the circuit's interconnection, attributes (e.g., power consumption), transistor sizes, etc. (2) a path descriptor describing the signal paths whose delays the designer has constrained, and (3) a routing capacitance descriptor describing the capacitive loading added to the various signal paths by the interconnect. All of these descriptors are hierarchical in order to communicate with the designer in a convenient fashion. Moreover the format of the circuit descriptor is not uniform, but rather depends on the type of circuit module. For example, the circuit descriptor for the ALU's adder must be different in format from that of the XOR logic gates, because the adder is composed of cells while the XOR gates are not, and the gate descriptors must contain pointers to transistor sizes, routines to compute power and delay, and the like. Furthermore the formats can even vary

according to the kind of logic cell, since a cell's topology may be encoded differently depending on the cell's type.

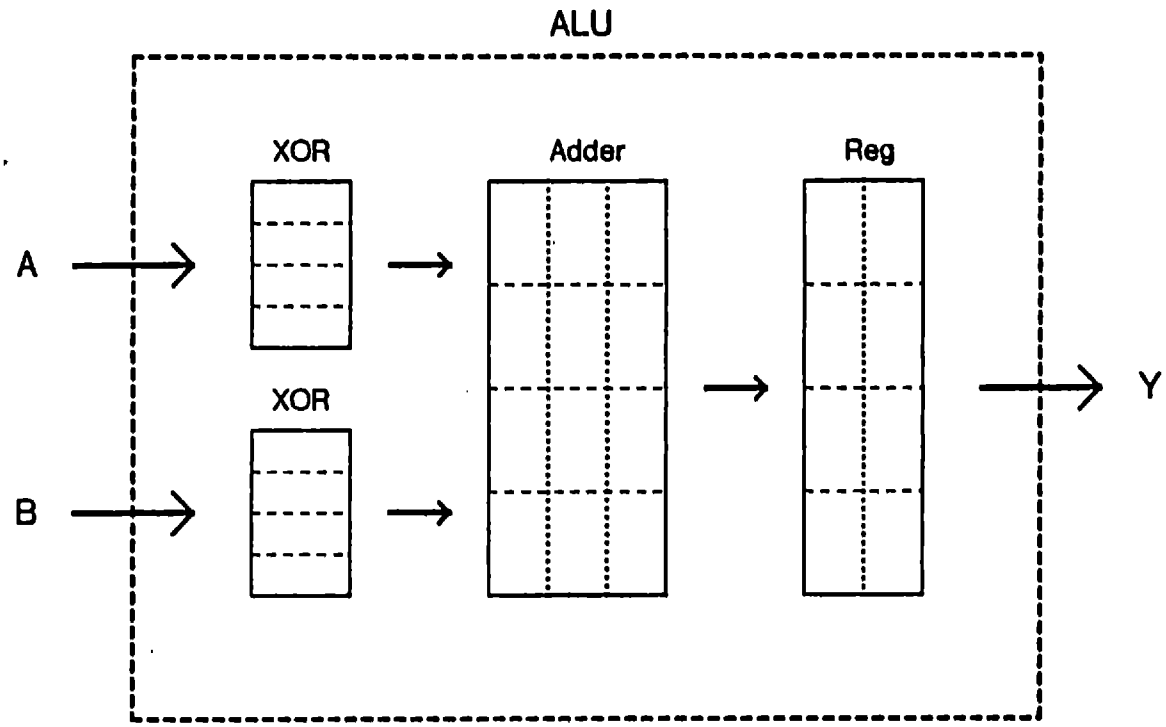


Figure 3-20: Simplified Arithmetic Logic Unit

There are several methods for handling data structure problems of this sort. We shall see that most current approaches are inefficient or inapplicable to our problem. Probably the most typical is static storage, where all of the structure's variables are explicitly declared prior to compilation and then assigned permanent storage locations upon compilation. Unfortunately the size of cell structures depends on the cell type, and this cannot be known until the circuit's design description is parsed at run time. One solution would be to assign storage for the maximum size data segments that will ever be needed. For instance, when writing matrix routines in Fortran, programmers typically allocate storage for the largest matrices that will ever be encountered. This is somewhat wasteful of storage, but often acceptable because usually matrix data structures are not excessively large. For VLSI designs the approach is totally infeasible because of the hierarchy. The levels of hierarchy and number of components at each level can potentially vary widely from circuit to circuit; allocating storage to handle the worst case at each level would consume an enormous amount of memory.

Another solution would be to parse the circuit design description and then generate computer language source code for the data structures. The code is then compiled and executed to create the data structures, tailored to the particular circuit to be optimized. This approach has been used successfully by Chris Terman in his RSIM simulator [27]. While the technique is perhaps the most efficient in its use of storage space, its implementation becomes quite unwieldly for elaborate data structures such as ours.

Some computer languages support semi-dynamic storage allocation and allow a programmer to allocate and deallocate blocks of storage at run time. The task of managing the storage falls entirely on the programmer. This has two major drawbacks. First, if the data structures are complicated the programmer is not likely to manage the storage as efficiently as could a language system that supported dynamic storage allocation and automatic reclamation (such as modern LISP). Second, depending on the language, error checking may be limited in this approach. While some languages such as Pascal and C provide a means of checking the type correctness of variables in the storage block at compile time, others do not, and run time type checking is rarely used on conventional machines because of the additional computational overhead.[21] In our experience the lack of error checking dramatically increases program development time.

The shortcomings of the previous approaches illumine the need for a computer language that offers sophisticated data structuring features. In particular, we want to be able to build data structures dynamically, hierarchically create other structures out of these structures, and easily walk the interconnection and hierarchy pointers to obtain information. For storage efficiency we divide a module's data into two categories: generic and instance information. Generic information is common to all modules of a given kind; it includes data such as input and output names and children interconnection. Instance information includes data which is unique to a particular circuit module, such as its transistor sizes. For run time efficiency we desire to avoid walking the interconnection hierarchy to access a cell's boundary conditions. We instead assign a boundary conditions descriptor to each cell instance and make it point to the attributes fields of its neighbors.

The result of these concepts is displayed in the example of Figure 3-21. Generic structures

[21] Lisp machines are an important exception. For example, the Symbolics 3600 uses special purpose hardware that performs the checks in parallel with the computations. As a result throughput is not reduced.

are shown as dashed polygons with italicized names; instance structures are depicted as solid ovals. The block object at the top represents boundary conditions imposed on the inverter chain by the outside world. The dashed lines indicate pointers to generic data structures. The solid lines are pointers for the hierarchical instance descriptors, and the dotted lines are boundary condition pointers.

3.5.3 Language Requirements

The preceding discussion conveys the necessity of a rich set of data management capabilities. The computer language chosen to implement the optimizer must support general data structures, such as records and unions (where a variable is declared as one of a set of possible types). Moreover these structures must be allocated and deallocated dynamically by the language system. For example, neither the lifetime nor the size of an array should be predetermined. The array grows and shrinks as required, and its storage space should be reclaimed when the array is no longer needed. User defined data types, along with a provision for governing their access, are also vital to our task. The ability to create and control the manipulation of abstract data types is crucial for protecting data structures and expediting program development. Through user defined types and their interface routines we can tightly control how data structures are accessed, providing both protection and flexibility in case the structure's internal representation must be changed. In addition the hierarchical nature of circuit design mandates that the language be facile with pointers. As we saw, for reasons of efficiency and error checking we desire that these be implicit pointers, handled by the language system (as in LISP), rather than explicit pointers, handled by the programmer (as in PL/I). Hence the children Huey, Dewey, and Louie of Figure 3-21 are considered to be extensions of the data structure for their parent Donald; no special mechanism is required to access them. Lastly, the language must support recursion, both in procedure calls and data structures. This is again due to the hierarchy inherent in our problem. Recursive procedure calls are the natural means for performing the optimization, and the data structures are built by combining similar structures in a tree-like fashion.

These concepts embody many of the principles of data abstraction and object oriented programming. These methodologies are geared more toward the creation and manipulation of data structures than are traditional programming languages. For each type of data object, there is a collection of programs that instantiate and modify objects of that type. For primitive types

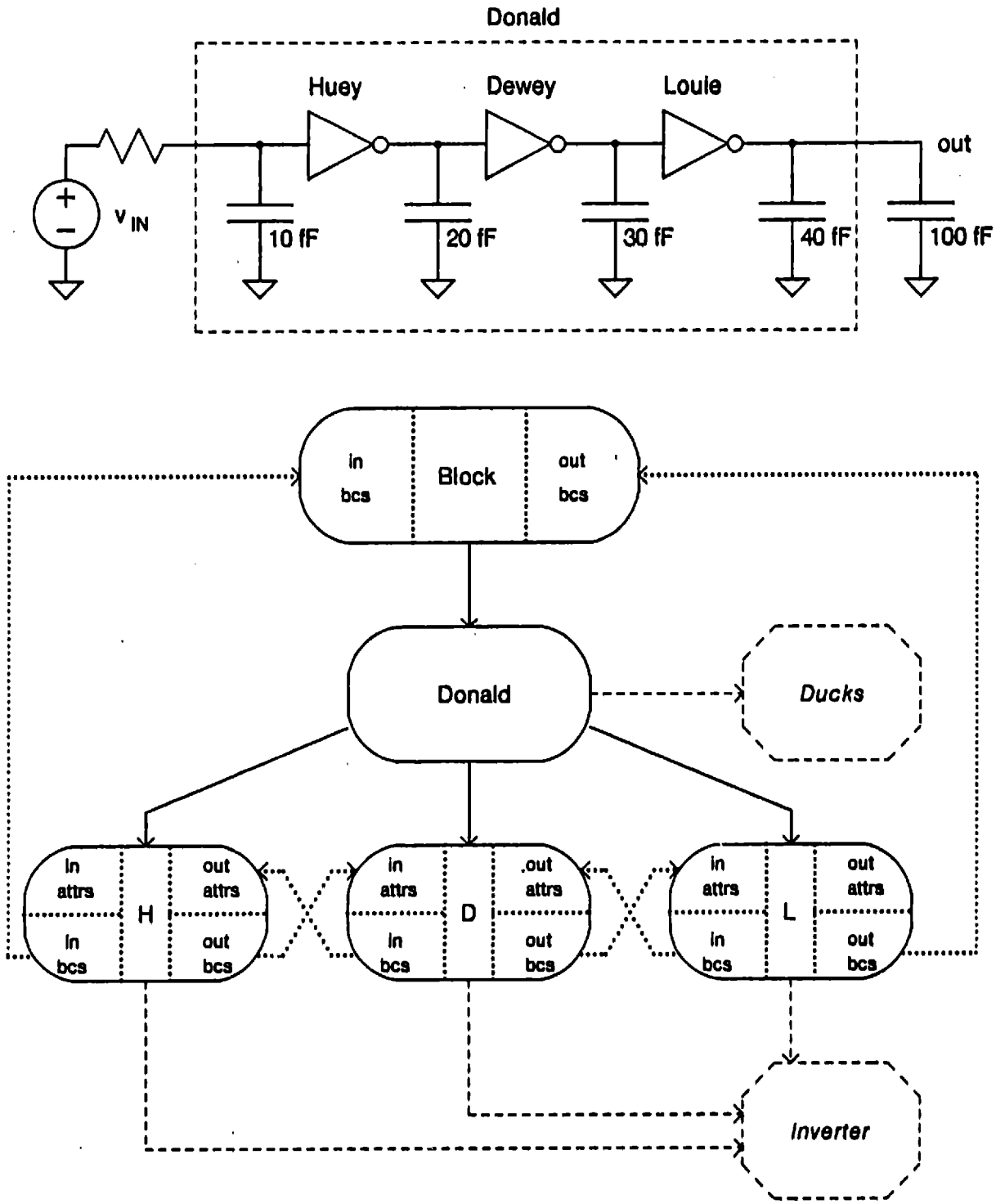


Figure 3-21: Circuit and Data Structure for a Chain of Inverters

(e.g., real, string) this collection is supplied by the language system; for user defined types the programmer writes it. Conceptually, a procedure creates and manipulates objects by sending messages to the object's collection. Objects are created and maintained in a separate portion of memory called the heap which is distinct from that portion where the programs reside. Objects are independently allocated and deallocated in the heap by the language system, not by the programmer. The system also reclaims the memory space freed by deallocated objects for future object creations.

We have chosen the CLU [19] language to implement our optimizer. This language was developed primarily to explore data abstraction issues. At present CLU compilers are available for DEC System 20's and VAX's. Other features of the language system include extensive compile time type checking, an outstanding interactive debugger, and a sophisticated exception handling mechanism for communication among program routines. All greatly facilitate program development. Another good choice would have been ZetaLISP on a Symbolics 3600.

3.5.4 Program Breakdown

Table 3-1 lists the major components of the optimizer. There are general support packages for optimization which interface to smaller packages that optimize each logic cell type. The circuit optimization package comprises the dual, module, path, Croute, opt-bcs, and parse clusters. The dual cluster formulates the dual functional, its gradient, and its second derivatives. This cluster controls the outer loop maximization. The module, path, and Croute clusters create and manipulate objects representing the circuit, path delay specification, and wiring capacitance descriptors, respectively. The opt-bcs cluster computes boundary conditions applied to cells by its electrical neighbors; this information is needed to size the cells. The parse cluster scans design specification files. The gate cluster consists of routines to compute the power, delay, and transistor sizes of the general logic gate described in the macromodeling chapter; it directs the inner loop minimization. The min, max, and matrix clusters form a support package for generic nonlinear minimization and maximization.

Program Cluster	*Lines of Code*
dual	350
module	4060
path	1010
Croute	440
opt – bcs	980
parse	550
gate	2690
max	740
min	1810
matrix	990
Total	13620

Table 3-1: Components of the Optimizer

3.5.5 Examples

We have applied our optimizer to many circuits; here we present a few representative cases. The optimizer is called *Tess*, standing for *Tool for Enhancing Silicon Systems*.[22] Our first example is the inverter chain of Figure 3-21. (The circuit is simple in order to allow a comparison with DELIGHT.) We specified the initial transistor sizes and constraints of Table 3-2 and requested maximum rise and fall delays of 8.0 ns. The optimizer stops when the delays for active constraints are within five percent of these values. Optimization statistics appear in Table 3-3. This table shows, for each maximizer iteration, the value of the dual functional $\varphi(\mu)$, the step size used to reach the next assignment of Lagrange multipliers μ, the active delay constraints, and the satisfied delay constraints. The letter 'r' designates the rising output transition while an 'f' denotes the falling output transition. The optimizer began with minimum size transistors and all Lagrange multipliers set to zero; it reached a solution in slightly over 15 cpu seconds on a DEC System 20/60. Note that the rising input, falling output delay constraint for this example remained inactive and satisfied throughout the optimization. This frequently occurs with nMOS circuits because the beta ratio requirement can introduce a large discrepancy between the speeds of rising and falling output transitions.

The accuracy of the macromodel's delay predictions is summarized in Table 3-4. In the

[22] *or Thesis to End Scholastic Servitude.*

parameter	*initial*	*minimum*	*maximum*
S_{pu}	0.5	0.5	1.0
w_{pd}	4.0 μm	4.0 μm	20.0 μm
β	4.0	4.0	8.0

Table 3-2: Initial Sizes and Constraints for the Inverter Chain

iter #	*$\varphi(\mu)$*	*step size*	*power [mW]*	*active*	*satisfied*
0	1.51	35.42	1.51	none	f
1	2.05	·	2.08	r	r, f

Optimization time (DEC 20/60 running CLU)
set up: 1.1 sec optimization: 15.2 sec

Table 3-3: Optimization Statistics for the Inverter Chain

table, T_{BEout} is the time until the output begins to move in response to an input transition and T_{SWout} is a measure of how quickly the output switches once it does begin to change. This table also gives the final values of the Lagrange multipliers. These show the speed-power tradeoff imposed by each constraint. The falling output constraint is inactive and did not affect the optimization; hence its multiplier is zero. Meeting the rising output specification required an increase in power consumption. The corresponding multiplier is positive. Furthermore its value indicates the circuit's location on the constraint's tradeoff curve. Increasing the path's speed by 1.0 ns will boost power consumption by about 0.4 mW. This information can be invaluable to a designer, for it shows the relative difficulty of meeting each path's delay specification. This can guide attempts to improve the circuit such as rerouting wires to reduce capacitance or ameliorating module topologies.

An attempt was made to run DELIGHT on the inverter chain and compare its results to those of our optimizer, but the effort met with only partial success. The complex interactions among objective and constraint functions overwhelmed DELIGHT's direction finding routine, causing the program to hang up in infinite loops in a most unpredictable and exasperating fashion.[23] This

[23]Bill Nye, the author of DELIGHT, believes that the problem lies in the direction finder's quadratic programming subroutine. He is investigating more robust routines.

Path	predicted [ns] T_{BEout}, T_{SWout}	SPICE [ns] T_{BEout}, T_{SWout}	error [%]	μ [mW/ns]
in → out, rise	4.06, 3.85	3.80, 3.76	+5	0.403
in → out, fall	5.23, 0.64	5.53, 0.77	−7	0.000

Total SPICE verification time (DEC 20/60 running FORTRAN): 16.5 cpu sec

Table 3-4: Delay Accuracies for the Inverter Chain

illustrates how general purpose optimization algorithms can fail when faced with a problem of this nature; specialized algorithms are essential. To pacify the direction finder, we eliminated the maximum beta ratio and minimum shape factor constraints, and started DELIGHT at an initial set of transistor sizes that was fairly close to the optimal solution. The problem was also simplified by not evaluating the chain's rising input, falling output response. This did not affect the final solution since this transition's delay constraint was not active, but it halved the number of SPICE runs needed and reduced the strain on DELIGHT's direction finder. DELIGHT required five iterations to converge to within five percent of the optimum, consuming 3018 cpu seconds on a VAX 750. Table 3-5 gives the statistics along with those of our optimizer for comparison.

Optimization Accuracy:

optimizer	cpu time [sec] set up	optimization	power [mW]
DELIGHT (VAX 750)	133.7	3018.0	2.02
Tess (DEC 20/60)	1.1	15.2	2.08

Inverter Transistor Sizes:

optimizer	first S_{pu}, w_{pd}	second S_{pu}, w_{pd}	third S_{pu}, w_{pd}
DELIGHT	0.72, 6.0	0.44, 7.1	0.83, 7.2
Tess	0.57, 4.6	0.50, 5.2	1.0, 8.0

Table 3-5: Comparison of Optimizers

The performances of the circuits produced by the two optimizers are quite similar. Both have falling input, rising output delays of 8.0 ns as requested, with power consumptions of 2 mW. The power consumption of DELIGHT's circuit is less than ours by about three percent, but this is mainly due to the removal of the minimum S_{pu} constraint on the second inverter. DELIGHT's

circuit allocates the chain's delay in a slightly different fashion than ours; the final stage is somewhat slower than ours while the second is a little faster, requiring a larger first stage S_{pu} in order to drive the wider second stage w_{pd}. This also affects the first stage's w_{pd} which must be wider to satisfy the minimum beta ratio constraint.

Our optimizer runs considerably faster than DELIGHT with SPICE. It is difficult to make an exact comparison of how fast DELIGHT would run on the DEC 20/60, had it been able to handle the inverter chain without simplifications, but we can make fairly accurate estimates. A DEC 20/60 will run Fortran code about three or four times faster than will a VAX 750. DELIGHT only evaluated one path transition, with fewer transistor size constraints and an initial set of sizes that was fairly close to the optimum. These simplifications halve the number of SPICE simulations per iteration and reduce the number of iterations needed to reach the optimum, leading to about a factor of five improvement in run time. Hence we believe that DELIGHT would require about 4000 cpu seconds to size the inverter chain on our DEC 20. This is about 300 times slower than our optimizer. We also feel that our run times scale better than DELIGHT's as circuit complexity increases. The partitioning scheme used by our optimizer leads to approximately linear growth while the growth rate of DELIGHT's feasible directions algorithms and SPICE's simulation algorithms are more rapid.

A more complicated example is displayed in Figure 3-22. This is a full adder module consisting of four logic cells with a total of eighteen transistors. Path specifications appear in Table 3-6. Table 3-7 gives the optimization statistics. Starting from the smallest devices allowed by the layout rules and beta ratio requirement, the optimizer required about 160 cpu seconds to size the transistors. Delay prediction accuracies are shown in Table 3-8. The accuracy is typically within several percent of SPICE.

This example illustrates several important characteristics of the optimizer. Note that the step size is large for the first iteration but near one for all others. The reason for this concerns the method used to generate the search direction. The maximizer starts with all Lagrange multipliers at zero and transistors at the minimum size permitted by design rules. In this region $\varphi(\mu)$ is linear in all components of μ and hence its second derivatives are zero.[24] Quasi-Newton methods are

[24]The transistor size restrictions place lower bounds on the power consumption and speed of the circuit. (Without these restrictions, setting the Lagrange multipliers to zero would produce a circuit with zero power dissipation and infinite delay.) Consequently, the multipliers must reach some positive threshold before the inner loop minimizer will size transistors beyond minimum size. Until some μ_j reach this threshold, $P(x)$ and $g_j(x)$ stay fixed, and hence $\varphi(\mu) = P(x) + \mu^t g(x)$ is linear in μ.

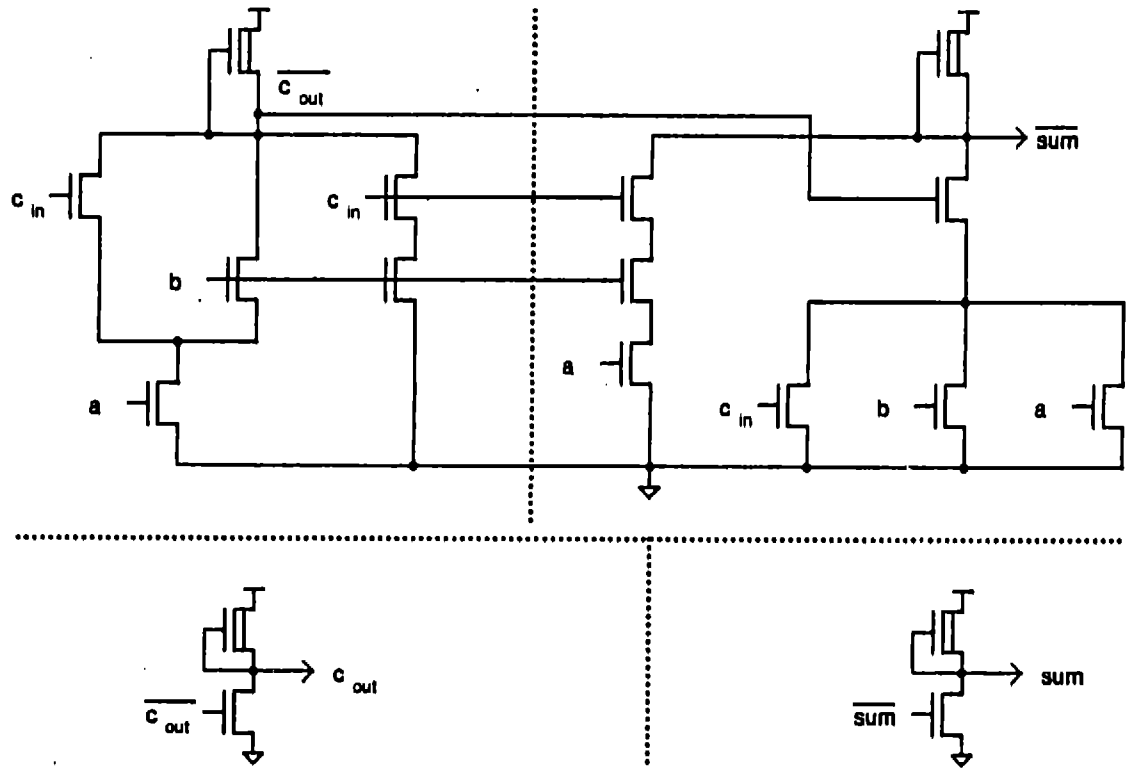


Figure 3-22: A Full Adder Module

therefore inapplicable for the first iteration, and so the optimizer begins with a steepest ascent (follow the gradient) step instead. For subsequent iterations the maximizer is no longer in the linear region and can use quasi-Newton methods. These methods have two advantages: they generate good search directions and the step size is typically about one, so the step size routine can begin its search anticipating a unity step size. The ensuing reduction in the number of function evaluations substantially improves cpu time.

Another significant point involves the active constraints. These constraints correspond to signal paths whose delay specifications are being met at the expense of increased power consumption. With nMOS circuits, and especially for fixed beta ratios, there are often large differences between the rising and falling output delays along signal paths. Experienced circuit designers occasionally increase beta ratios beyond the minimum required by design rules in order to improve circuit speed. In like fashion the optimizer uses this same technique to reduce cells' falling output delays, leading to signal paths with comparable rise and fall delays.

Path	*Number*	*Desired Delay [ns]*	*Other Inputs*
a → sum	1	8.0	b low, c_{in} high
a → c_{out}	2	8.0	b low, c_{in} high
c_{in} → c_{out}	3	6.0	a high, b low

Table 3-6: Path Delay Specifications for the Full Adder Module

iter #	*φ(μ)*	*step size*	*power [mW]*	*active*	*satisfied*
0	2.01	34.64	2.01	none	1f
1	2.20	0.78	2.26	1r, 2rf, 3rf	1rf, 2f, 3f
2	2.23	1.91	2.23	1r, 2rf, 3r	1rf, 2f, 3f
3	2.32	2.02	2.25	1r, 2rf, 3r	1rf, 2r, 3rf
4	2.33	-	2.32	1r, 2rf, 3r	all

Optimization time (DEC 20/60 running CLU)
set up: 2.2 sec optimization: 163.1 sec

Table 3-7: Optimization Statistics for the Full Adder Module

Path	*predicted [ns]* T_{BEout}, T_{SWout}	*SPICE [ns]* T_{BEout}, T_{SWout}	*error [%]*	*μ [mW/ns]*
a → sum, rise	5.96, 1.66	6.42, 1.73	−7	0.060
a → sum, fall	5.47, 0.49	6.16, 0.63	−12	0.000
a → c_{out}, rise	2.78, 3.26	2.78, 3.31	−1	0.078
a → c_{out}, fall	4.98, 0.79	5.29, 1.02	−9	0.163
c_{in} → c_{out}, rise	3.03, 3.26	3.10, 3.30	−2	0.122
c_{in} → c_{out}, fall	4.09, 0.65	4.46, 0.76	−9	0.000

Total SPICE verification time (DEC 20/60 running FORTRAN): 189.2 cpu sec

Table 3-8: Delay Accuracies for the Full Adder Module

We also note that the set of active paths changes as the optimization progresses. In the first iteration constraint 3f is active, but due to the interactions among signal paths it is inactive thereafter. It is these interactions that make the optimization task so difficult. Widening a transistor to speed up one path may slow down another because of capacitive loading effects. Some workers have advocated using heuristics to order the paths according to how "critical" they are, and then optimizing each path in succession. Unfortunately path interactions can seriously undermine the search for an optimal solution because sizing one signal path can adversely affect previously sized paths. Moreover the identity of the critical paths can change during the optimization. (In fact, one of our earlier optimization schemes was based on critical path heuristics and had to be abandoned for these reasons.) We feel that nonlinear optimization algorithms offer major advantages over heuristics in their handling of path interactions. By simultaneously considering all signal paths at each iteration, nonlinear optimization algorithms can produce solutions which are more optimal and arrive at them in a more efficient manner than heuristics.

The maximizer uses a nonstandard step size algorithm. This is necessary because of inaccuracies in the inner loop minimization. The minimization embedded in each iteration of the maximizer sizes the transistors. This sizing is a function of the value of the Lagrange multipliers, which are determined by the maximizer. If the minimizer sized transistors to extremely high accuracies then $\varphi(\mu)$ would increase at every iteration. However this would be very wasteful of computer time, especially since transistor fabrication is only accurate to a few percent at best. Consequently the minimization is done to only a few percent accuracy, and this injects a small amount of noise into the computation of $\varphi(\mu)$. The noise is especially noticeable as the optimizer closes in on the solution, for here the Lagrange multipliers begin to converge and the minimizer can exhibit a kind of hysteresis, refusing to resize devices until the Lagrange multipliers change significantly. We had originally implemented a cubic step size rule for the maximizer, but this noise caused the optimizer to behave erratically as it moved toward a solution. We replaced the cubic rule with an algorithm that only uses the gradient of the dual and not its value. This focuses the optimizer on our primary concern, the signal path delays, since the gradient of the dual is the value of the delay constraints. This rule circumvented any difficulties and is quite fast, requiring only one or two iterations to reach the vicinity of the solution. In this example the effects of the noise are not clearly visible, but in some other circuits we have optimized, the value of $\varphi(\mu)$ actually decreased during certain iterations, despite the fact that this is a maximization loop.

Four Bit Adder

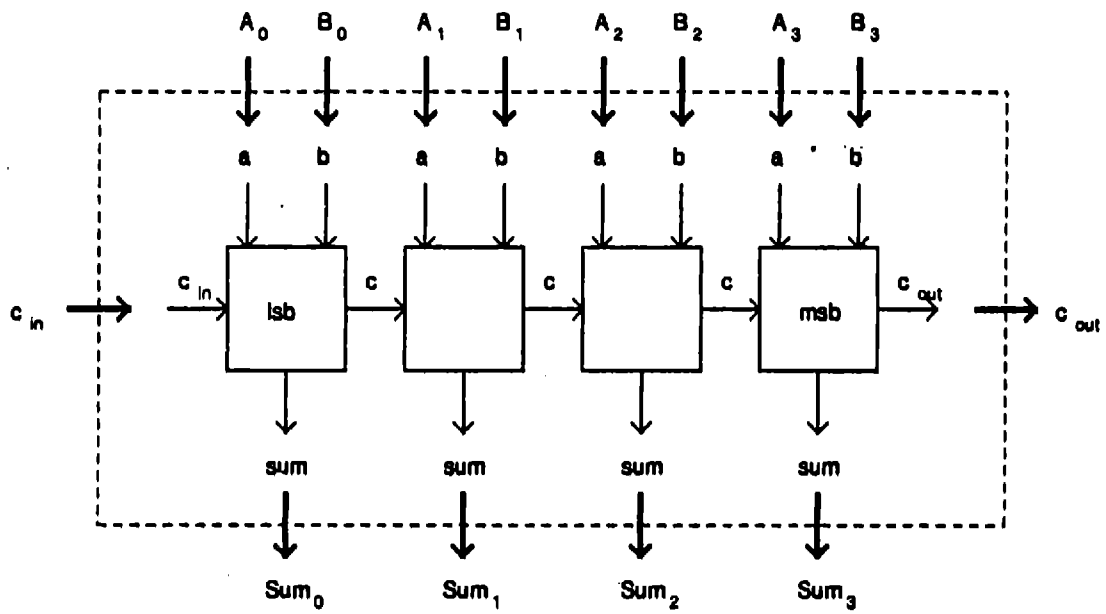


Figure 3-23: Four Bit Adder

A final example concerns the multiple bit adder shown in Figure 3-23, comprised of four of the full adder modules of Figure 3-22. The adder contains sixteen logic cells having a total of 72 transistors. The circuit was optimized subject to the delay specifications of Table 3-9. Optimization statistics appear in Table 3-10, while Table 3-11 presents a comparison of the macromodel's predicted delays versus those of SPICE. We again see a small number of outer loop iterations, with all but the first step size near one. Furthermore the computation time has grown roughly linearly with circuit complexity relative to the previous examples. Starting with minimum size transistors, the optimizer required only 520 cpu seconds to optimize the adder. In contrast, considerably more time was needed for SPICE runs to just verify the accuracy of the predicted delays.

In summary the speed and precision of the optimizer are quite impressive. These examples show run times of about one cpu second per transistor per constrained path, with accuracies typically within several percent of SPICE estimates.

Path	*Number*	*Desired Delay [ns]*	*Other Inputs*
$A_0 \rightarrow Sum_0$	1	8.0	$a_i = 1, b_i = 0, c_{in} = 1$
$A_1 \rightarrow Sum_1$	2	8.0	$a_i = 1, b_i = 0, c_{in} = 1$
$A_2 \rightarrow Sum_2$	3	8.0	$a_i = 1, b_i = 0, c_{in} = 1$
$A_3 \rightarrow Sum_3$	4	8.0	$a_i = 1, b_i = 0, c_{in} = 1$
$c_{in} \rightarrow c_{out}$	5	20.0	$a_i = 1, b_i = 0$

Table 3-9: Path Delay Specifications for the Four Bit Adder

iter #	*φ(μ)*	*step size*	*power [mW]*	*active*	*satisfied*
0	8.02	19.88	8.02	none	1f, 2f, 3f, 4f
1	9.42	1.34	9.53	1r, 2r, 3r, 4r, 5rf	1rf, 2rf, 3rf, 4rf, 5f
2	9.84	2.96	9.33	1r, 2r, 3r, 4r, 5rf	1rf, 2rf, 3f, 4f, 5f
3	10.01	-	10.25	1r, 2r, 3r, 4r, 5rf	all

Optimization time (DEC 20/60 running CLU)
set up: 2.7 sec optimization: 519.6 sec

Table 3-10: Optimization Statistics for the Four Bit Adder

Path	predicted [ns] T_{BEout}, T_{SWout}	SPICE [ns] T_{BEout}, T_{SWout}	error [%]	μ [mW/ns]
$A_0 \rightarrow Sum_0$, rise	6.20, 1.66	6.76,1.73	−7	0.049
$A_0 \rightarrow Sum_0$, fall	5.53, 0.49	6.20, 0.62	−12	0.000
$A_1 \rightarrow Sum_1$, rise	6.26, 1.66	6.67, 1.73	−6	0.101
$A_1 \rightarrow Sum_1$, fall	5.53, 0.49	5.96, 0.59	−8	0.000
$A_2 \rightarrow Sum_2$, rise	6.28, 1.66	6.72, 1.73	−6	0.098
$A_2 \rightarrow Sum_2$, fall	5.52, 0.49	5.92, 0.61	−8	0.000
$A_3 \rightarrow Sum_3$, rise	6.19, 1.66	6.78, 1.74	−8	0.114
$A_3 \rightarrow Sum_3$, fall	5.44, 0.49	5.86, 0.60	−8	0.000
$c_{in} \rightarrow c_{out}$, rise	17.43, 2.46	15.14, 2.55	+12	0.246
$c_{in} \rightarrow c_{out}$, fall	18.30, 0.55	18.99, 0.61	−4	0.242

Total SPICE verification time (DEC 20/60 running FORTRAN): 1468.6 cpu sec

Table 3-11: Delay Accuracies for the Four Bit Adder

CHAPTER FOUR

Conclusion

4.1 Summary

We have presented a novel technique for optimizing VLSI functional blocks. Our CAD tools find the transistor sizes that minimize a circuit's power consumption subject to delay constraints on signal paths. This transistor sizing problem is simply too hard for conventional methods. The difficulty is mainly due to the need for a high degree of accuracy, the potentially large number of transistors to size, and the nonlinear nature of the circuit's path delay constraints. To achieve the signal delay precision vital to high-performance circuit design, standard practice has been to have a human designer run a transistor level simulator (such as SPICE) on a circuit and manually update the transistor sizes until delay specifications are met, or to use general purpose nonlinear optimization algorithms directly interfaced to a circuit simulator. The former approach involves a good deal of designer and computer time, usually precluding reaching the optimum; the latter entails an excessive amount of computer time for all but small circuits. We also examined several methods where workers had simplified the logic gate models or applied heuristics rather than nonlinear programming algorithms in order to reduce cpu time, but these schemes were found to be too inaccurate.

Our approach combined knowledge from circuit theory, nonlinear programming, and modern computer language design. We pursued work in two areas. First, we derived accurate, computationally efficient macromodels for logic gates. The form of the macromodel equations was based on the logic gates' equivalent circuit models and reflected a careful consideration of waveshape effects; the equations' parameters were determined by curve fitting to observed data from a transistor level simulator. Second, we created a special purpose optimizer, exploiting properties of digital MOS logic gates. The optimizer used a dual algorithm which breaks the problem into an inner loop minimization embedded in an outer loop maximization. The outer loop solved for Lagrange multipliers, one for each path delay specification, subject to nonnegativity

constraints on the multipliers. The inner loop solved for the optimal transistor sizes, subject to box and linear constraints, and was partitioned down to the cell level. Thus both loops operated in small vector spaces with simple constraints, greatly expediting convergence to the solution. The optimizer was implemented in a computer language system that offered sophisticated data and control abstraction mechanisms along with extensive program development facilities. The result was an extremely effective method for solving what is surely a very difficult optimization problem.

4.2 Future Research

Like many research projects, this work has perhaps raised more issues than it has settled. The ability to optimize a functional block's power and delay carries implications regarding other areas of circuit optimization as well. Knowledge from one domain can be applied to problems in another. For example, designers frequently face situations where a speed specification on a circuit path cannot be met, no matter how the circuit's transistors are sized. In such cases designers often try to rearrange interconnect wires and module placements to reduce the capacitive loading on critical nodes, hoping to thereby meet the speed requirement. Our optimizer can be an invaluable asset in such situations. Since it maintains information on the sensitivities of cell delays to input waveforms and capacitive loading, estimates of the new circuit delays after a wiring or module placement change could be provided. Moreover the sensitivities could also be used in conjunction with the values of the Lagrange multipliers to predict the performance of the new circuit after its optimization. These estimates would be extremely useful in guiding a designer during experiments with possible floor plans for a chip.

These concepts can be further extended. Having a tool which can efficiently optimize a functional block topology with specified path delays, we can generate rough characterizations of the power-delay tradeoffs for optimized functional blocks. This gives us an aid in topology selection. For instance, the tradeoff curves indicate over which range of delay specifications an adder with a look-ahead carry circuit would be preferable to, say, an adder with a Manchester carry. By integrating these ideas into a VLSI design system, one could form a powerful designer's assistant CAD tool.

An orthogonal but equally important issue concerns the optimizer's implementation. We have gone to considerable pains to preserve the separability inherent in the problem, allowing us

to partition it and reap considerable benefits in computational speed. While we originally sought to segment the problem in order to decrease the size of the vector spaces, partitioning also allows us to run the algorithm on parallel computation machines. Recently several workers have developed special purpose parallel machines for VLSI design [28, 29]. These machines have accelerated logic level simulation tremendously. We feel similar success could be had if this optimizer were to be implemented on parallel machines.

4.3 Perspective

Through a rather intricate partnership of circuit theory, optimization algorithms, and software technology we have solved an important VLSI design problem. Many bottlenecks in the design process have yet to be overcome if designers are to keep pace with the ever-increasing power offered by fabrication technology. We believe that similar multidisciplinary strategies are essential if progress is to continue.

References

[1] W. Nye, E. Polak, A. Sangiovanni-Vincentelli, and A. Tits, "DELIGHT: An Optimization-Based Computer-Aided Design System," *Proceedings International Symposium on Circuits and Systems*, IEEE, April 1981, pp. 851-855.

[2] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design," *Proceedings of the IEEE*, Vol. 69, No. 10, October 1981, pp. 1334-1362.

[3] A. Ruehli, P. Wolff, and G. Goertzel, "Analytic Power/Timing Optimization Technique for Digital System," *Proceedings 14th Design Automation Conference*, IEEE, June 1977, pp. 142-146.

[4] N. Jouppi, "Timing Analysis for nMOS VLSI," *Proceedings 20th Design Automation Conference*, IEEE, June 1983, pp. 411-418.

[5] S. Trimberger, "Automated Performance Optimization of Custom Integrated Circuits," *Proceedings International Symposium on Circuits and Systems*, IEEE, May 1983, pp. 194-197.

[6] J. Ousterhout, "Switch-Level Delay Models for Digital MOS VLSI," *Proceedings 21st Design Automation Conference*, IEEE, June 1984, pp. 542-548.

[7] M. Horowitz, *Timing Models for MOS Circuits*, PhD dissertation, Stanford, 1983.

[8] L. P. J. Hoyte, "Automated Calculation of Device Sizes for Digital IC Designs," Master's thesis, MIT, 1982.

[9] L. Glasser and L. Hoyte, "Delay and Power Optimization in VLSI Circuits," *Proceedings 21st Design Automation Conference*, IEEE, June 1984, pp. 529-535.

[10] J. Roberge, *Operational Amplifiers: Theory and Practice*, Wiley, 1975, pp. 97-104.

[11] P. Gray and C. Searle, *Electronic Principles: Physics, Models, and Circuits*, Wiley, 1969.

[12] S. Sze, *Physics of Semiconductor Devices*, Wiley, 1969.

[13] L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, 1985.

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

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

[16] T.-M. Lin and C. Mead, "Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits," *Proceedings, Conference on Advanced Research in VLSI*, MIT, January 1984, pp. 93-99.

[17] R. Fletcher and M. Powell, "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, Vol. 6, 1963, pp. 317-322.

[18] D. Bertsekas, "Projected Newton Methods for Optimization Problems with Simple Constraints," *SIAM Journal on Control and Optimization*, Vol. 20, 1982, pp. 221-246.

[19] B. Liskov, A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," *Communications of the ACM*, Vol. 20, No. 8, August 1977, pp. 564-576.

[20] D. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1984, second edition.

[21] Yonathan Bard, *Nonlinear Parameter Estimation*, Academic Press, 1974.

[22] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, 1982.

[23] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, May 1983, pp. 671-680.

[24] S. White, "Concepts of Scale in Simulated Annealing," *Proceedings International Conference on Computer Design: VLSI in Computers*, IEEE, October 1984, pp. 646-651.

[25] J. Greene and K. Supowit, "Simulated Annealing without Rejected Moves," *Proceedings International Conference on Computer Design: VLSI in Computers*, IEEE, October 1984, pp. 658-663.

[26] D. Bertsekas, G. Lauer, N. Sandell, and T. Posbergh, "Optimal Short Term Scheduling of Large-Scale Power Systems," *IEEE Transactions on Automatic Control*, Vol. AC-28, No. 1, January 1983, pp. 1-11.

[27] C. Terman, *Simulation Tools for Digital LSI Design*, PhD dissertation, MIT, 1983.

[28] G. Pfister, "The Yorktown Simulation Engine: Introduction," *Proceedings 19th Design Automation Conference*, IEEE, June 1982, pp. 51-54.

[29] Zycad Corporation, *LE-1000 Series Logic Evaluator Intermediate Form Specification*, Roseville, Minnesota, 1983, Release 1.0.