PERFORMANCE OPTIMIZATION OF INTEGRATED CIRCUITS

by

Jyuo-Min Shyu

Memorandum No. UCB/ERL M88/74

22 November 1988

PERFORMANCE OPTIMIZATION OF INTEGRATED CIRCUITS

by

Jyuo-Min Shyu

Memorandum No. UCB/ERL M88/74

22 November 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

PERFORMANCE OPTIMIZATION OF INTEGRATED CIRCUITS

by

Jyuo-Min Shyu

Memorandum No. UCB/ERL M88/74

22 November 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Performance Optimization of Integrated Circuits

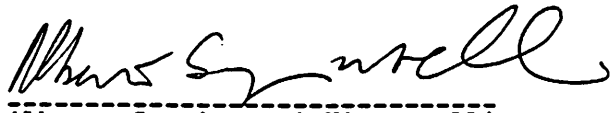
Jyuo-Min Shyu

Ph.D.

Department of Electrical Engineering
and Computer Science

Abstract

Optimization tools are valuable in designing high-performance integrated circuits (ICs). Depending on the design environment, these tools perform optimization in two ways – interactive and automatic. Interactive optimization is performed in an interactive design environment where it is desirable to guide tradeoffs among competing performances of the circuit. Automatic optimization is embedded in an automated design environment where design specifications are easily obtainable from higher-level design tools. This thesis addresses techniques for both interactive and automatic optimization of ICs at the transistor level – an area in which early efforts were not very successful. Two tools have been developed – one primarily for analog design in the interactive case, and the other solely for digital CMOS design in the automatic case. The interactive system developed for analog design uses a forms-based user interface to allow the designer to formulate design problems easily. In this system, both gradient-based and random search algorithms are employed in the interest of avoiding numerical difficulties in the optimization. The automatic system developed for digital CMOS design is a transistor sizer which incorporates desirable features of both heuristic and nonlinear programming techniques. Experimental results are presented to demonstrate the effectiveness of the proposed approaches.



Alberto Sangiovanni-Vincentelli
Thesis Committee Chairman

Acknowledgments

I wish to express my sincere gratitude first and foremost to my research advisor, Professor Alberto Sangiovanni-Vincentelli for his guidance and support throughout my study at Berkeley. His comprehensive and deep knowledge of computer-aided design has served as an example, motivating me to spend the many many hours invested in this research work. I would also like to thank him along with Professors Robert Brayton and Henry Helson for serving on my thesis committee. Professor Elijah Polak also took interest in my research. I have benefited from the many discussions I had with him.

I wish also to gratefully acknowledge the financial support received from DARPA under Grant N00039-87-C-0182.

I am very grateful to those at AT&T Bell Laboratories who made my appointment there in the summer of 1986 so valuable and unforgettable. Specifically, Al Dunlop and Jack Fishburn provided me with an excellent research environment. On my first day there, Jack patiently explained his highly regarded transistor sizing algorithm to me, and valuable discussions about techniques for CMOS performance optimization became a daily occurrence.

I would like to thank my office partners, Dave Riley, Nick Weiner, and Hormoz Yaghutiel, for their advice and help on a wide variety of problems both technical and administrative. Dave's effort in reviewing and making suggestions for oral and written presentations of my research has been especially significant. In addition, Dave has allowed me to be more comfortable in my working and living environments by clearly answering many questions about English language usage and about American culture.

The implementation of ECSTASY would not have been possible without the work of Tom Quarles and Wayne Christopher, on SPICE3, and Dave Riley and Umakanta Choudhury, on techniques for the computation of sensitivity. Many of the ideas Bill Nye has incorporated in the DELIGHT

optimization system have had a major influence on ECSTASY. Bill also impacted my results positively by bringing my attention to a wider range of relevant published research than I might otherwise have seen.

I am very grateful to Nick Weiner for spending one of his busiest nights proofreading a preliminary draft of the first three chapters of this thesis. I would also like to thank Joe Higgins and Joe Wiest, both very knowledgeable about optimization theory, for proofreading Chapter 3.

The friendship of many Chinese students, especially Wen-Bin Hsu, has added greatly to the enjoyment of my years in Berkeley. I am also grateful to many of the fellow students in CAD Group for their help and friendship. Among them are Andrea Casotto, Kwang-Ting Cheng, Srinivas Devadas, Tammy Huang, Theologos Kelessoglou, Young Kim, Ken Kundert, Bill Lin, Tony Ma, Abdul Malik, Kartikeya Mayaram, Linda Milor, Fabio Romeo, Kanwar Singh, Rick Spickelmier, Ren-Song Tsay, and Don Webber.

Finally, I would like to express my most sincere gratitude to members of my family. My grandmother (ARMAH), mother, and brothers have provided support and encouragement throughout my Berkeley years. It is difficult to express in words my appreciation for the role that my wife Yih-Ting and my daughters, Chih-Min (Cory) and Chia-Chi (Terry) played in my Berkeley experience. I will simply say that they have endured many of my absences, and many other hardships, so that I might reach my goal.

Contents

Table of Contents iii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Optimization Environment . . . 2
  - 1.2.1 Interactive Optimization . . . 3
  - 1.2.2 Automatic Optimization . . . 4
- 1.3 Difficulties and Strategies . . . 5
- 1.4 Outline of the Thesis . . . 6

2 Survey of Existing Tools 8
- 2.1 Introduction . . . 8
- 2.2 General-Purpose Interactive Systems . . . 8
  - 2.2.1 Early Efforts . . . 8
  - 2.2.2 Shortcomings of Existing Approaches . . . 10
- 2.3 Transistor Sizers . . . 12
  - 2.3.1 Early Efforts . . . 12
  - 2.3.2 Shortcomings of Existing Approaches . . . 15

3 Algorithmic Considerations in IC Optimization 17
- 3.1 Introduction . . . 17
- 3.2 Problem Formulation . . . 18
- 3.3 Semi-infinite Programming . . . 19
- 3.4 Finding a Feasible Design . . . 20
- 3.5 Heuristic Methods . . . 22
- 3.6 Multiobjective Optimization . . . 25
- 3.7 Feasible Directions Algorithms . . . 28
- 3.8 Numerical Difficulties Arising in Circuit Optimization . . . 35
  - 3.8.1 Scaling . . . 37
  - 3.8.2 Finite Difference Approximation . . . 41
  - 3.8.3 A Random Search Scheme to Remedy the Problem . . . 42

4 A General-Purpose Interactive Optimization System **46**
4.1 Introduction 46
4.2 Optimization Algorithms 47
4.2.1 Sensitivity Computation 49
4.2.2 Controlled Random Search 51
4.3 User Interface Design 52
4.3.1 Design Criteria 52
4.3.2 Forms-Based Interface 56
4.3.3 Simulation Control 60
4.3.4 User Interaction 60
4.4 Circuit Simulation Interface 61
4.5 Design Examples 65
4.5.1 CMOS I/O Driver Circuit 66
4.5.2 Switched-Capacitor Filter 70
4.5.3 Bipolar Operational Amplifier 78

5 A Transistor Sizer for Digital CMOS Circuits **81**
5.1 Introduction 81
5.2 A Two-Stage Combined Approach 82
5.2.1 The Approach 82
5.2.2 The Algorithm 84
5.2.3 A Variable Bump-Size Scheme 86
5.3 Modeling Circuit Delays 86
5.3.1 MOSFET Model 88
5.3.2 Distributed RC Delay Model 88
5.4 Feasible Directions Algorithm and Generalized Gradient . . . 95
5.4.1 Feasible Directions Algorithm 95
5.4.2 Generalized Gradient 96
5.4.3 Problem Scaling 104
5.5 Delay Sensitivity Computation 104
5.6 Experimental Results 109

6 Conclusions and Future Research **112**
6.1 Conclusions 112
6.2 Future Research 113

Bibliography **115**

Chapter 1

Introduction

1.1 Motivation

Hierarchical design of integrated circuits (ICs) is an iterative procedure consisting of several levels of design tasks. Given a design specification, the iterative procedure is to design the architecture, modularize each functional block of the architecture, realize the circuit to perform each module function, and layout the circuit. The process is repeated until the final circuit satisfies the specification requirements.

Usually, the design task at each level involves a number of design options, each affecting the circuit performance. Optimization, i.e., the procedure to modify the design to minimize the deviation of actual performance from the desired, can be applied at all levels of design. Since the process to find manually the best possible choice for meeting the performance requirements is very time-consuming, and seldom produces a circuit with optimum performance, optimization tools become important in achieving high-performance design. Unfortunately the complexity and sophistication of modern ICs hinder the development of fully automatic design and optimization systems.

This thesis focuses on the problem of performance optimization at the *transistor level* - an area in which early efforts were not very successful.

It is assumed that the circuit topology and component types are given and fixed throughout the optimization. Furthermore, parasitic effects, which can only be modeled after physical design is complete, and statistical effects on parameters such as transistor sizes and resistor values, are ignored. The design goals in the work presented here are described in terms of the performance of individual circuits which may, as in the case of totally analog ICs, comprise the entire IC, but, more generally, comprise a subcircuit of the entire IC.

The problem is simplified further by another approximation. Due to the limitations in IC process technology, the design parameters are subject to finite process resolution constraints. As a consequence, their values are restricted to a discrete set of values. Direct solution of optimization problems with design parameters restricted to discrete set is in general very difficult to solve. For this reason, it is initially assumed that the design parameters may take on real values in an interval. Once optimal real-valued results are obtained, these are rounded to the nearest values belonging to the required discrete set.

1.2 Optimization Environment

At the transistor level, performance optimization can be viewed as an integral part of module generation which iterates the procedure of synthesis, optimization, and layout to realize a given design specification. There are two ways of performing this optimization task – *interactive*, and *automatic*. Both require an integration of sophisticated optimization algorithms and circuit analysis tools. Interactive optimization is performed in an interactive design environment where it is desirable to trade off the various aspects of the circuit behavior; automatic optimization is embedded in an automated design environment where design specifications are easily obtainable from higher-level design tools.

1.2.1 Interactive Optimization

Consider the design of analog circuits (which, for the purpose of this research, encompasses aspects of the design of digital systems in which detailed circuit behavior, such as switching waveforms and output driving capabilities, is of interest). Two approaches can be used. One of them is to use synthesis techniques to produce circuits directly from design specifications. This approach has been used successfully in the design of some limited classes of circuits, such as filters [1]. Recently, researchers [2] [3] [4] have made considerable progress in the synthesis of certain classes of analog building blocks such as operational amplifiers and voltage references. However, these synthesis tools have limited optimization capabilities. This may lead to their failure to produce any solution satisfying a given set of design specifications, even though one may exist.

Much more common than the analog synthesis approach is one in which design is viewed as an iterative procedure consisting of heuristic synthesis, circuit analysis, and optimization. This approach depends heavily on the designer's past experience, knowledge, and intuition. The designer starts with an initial circuit realization which is to satisfy the design specifications. This initial design is then analyzed to determine the discrepancy between the actual and desired performance. Optimization is then used to reduce the deviation to within given tolerances. If, after the optimization step, the circuit performance still cannot meet the required specification, the circuit topology is modified and the iterative cycle is repeated. Clearly in this design approach, the designer plays an active role throughout.

The design of analog (in the sense defined above) circuits generally entails consideration of numerous performance criteria (such as input offsets, noise, gain, and slew rate for a strictly analog circuit and rise time and quiescent power consumption for a digital system subcircuit). These performance criteria depend in a complex way on the design parameters of interest, and may present conflicting considerations in setting design parameters, leading

to a need for the designer to trade off the criteria, one against the other, until a compromise is reached. Consequently, it is desirable that the optimization be based on circuit simulation, and be performed in an interactive way. Tools for this purpose are therefore *interactive systems* for *general-purpose* use.

Note that, in this environment, circuit element values that can be treated as design parameters may include resistance, capacitance, and transistor size.

1.2.2 Automatic Optimization

In automated digital design systems, by contrast, it is generally desirable that optimization be carried out fully automatically. In the case of digital systems, design often proceeds in a top-down manner, partitioning the initial problem into smaller subproblems until their complexity becomes manageable. The design abstractions in this hierarchy are rather well-developed and allow the systematic use of synthesis tools. In fact, recent advances in computer-aided design (CAD) tools have made possible fully automatic compilation of digital systems from high-level descriptions [5]. Tools with this capability, however, generally do not contain mature solutions to the problem of automatic optimization of performance objectives such as power, speed, and area.

Note that, in this hierarchical design environment, while each design abstraction may have a number of design options, the corresponding performance requirement is rather well-defined. Typically, at the subcircuit level, only a few aspects of circuit performance, such as delay or power, are of interest. Also, since digital design has to deal with very large circuit size, circuit behavior is often characterized based on macromodelling or timing analysis using simplified device models.

For example, in the Berkeley synthesis environment [6], the objective function is total active area, and the constraints are delay constraints. To achieve this goal, the approach is first to minimize the area without consid-

eration of delay, then to meet the system timing requirements obtained from higher-level synthesis tools. Optimization proceeds in two stages. The first is logic restructuring, where alternatives of the structure of logic equations are searched for better timing characteristics. The second stage involves iterations between transistor sizing and timing-driven placement and routing techniques. Similar approaches have been reported in other performance-oriented synthesis systems [7] [8], although the objective and constraint functions are somewhat different from those described above.

Therefore, in digital design, it is desirable to perform automatic optimization, based on macromodelling or timing analysis, in an environment where design specifications are available from other tools. In this environment, since the only designable parameters are the transistor sizes, tools, assuming fixed circuit topology, for this purpose are often called *transistor sizers*.

1.3 Difficulties and Strategies

Optimization theory has provided many numerical techniques applicable to circuit optimization problems formulated as standard optimization programs.

The goal of these techniques is to find a stationary point, i.e., a point that satisfies a set of necessary optimality conditions. In the application of robust algorithms such as the feasible directions algorithms [9] to circuit problems, convergence to a stationary point may be slow. Furthermore, convergence may not even be achieved due to the finite precision of the computations performed to evaluate the objective functions, the constraint functions and their derivatives. Nevertheless, at least theoretically, convergence is guaranteed under rather mild conditions.

Despite the fact that optimization has been applied successfully in many applications, the acceptance of either automatic or interactive optimization by circuit designers is unfortunately still limited, due to difficulties

they have faced in applying these techniques. Most of these difficulties arise either from poor problem formulation, or from difficulties in the implementation of the optimization algorithms. While optimization experts working in an interactive environment can generally overcome some of these difficulties, most IC designers without a background in optimization, even with the aid of a powerful computer and a library of algorithms, cannot.

This thesis attempts to alleviate both the problem formulation and the algorithmic difficulties. For problem formulation, when interaction is desirable, user interface is designed so that the designer can formulate the problem easily without understanding optimization theory. To aid in avoiding algorithmic problems, special methods based on considerations particular to IC design are provided, and optimization algorithms are designed so that they can proceed without human intervention.

1.4 Outline of the Thesis

The organization of the thesis is as follows. Chapter 2 reviews the previous efforts in general-purpose optimization systems for the interactive optimization of analog circuits, and in transistor sizers for the automatic optimization of digital circuits. Both contributions and limitations of the existing tools are discussed. Chapter 3 details the algorithmic considerations in IC optimization. Beginning with a standard formulation for IC optimization problems, the chapter surveys solution methods suitable for circuit optimization. In particular, heuristic, nonlinear programming, and random search algorithms are discussed, in the context of IC performance optimization. Chapter 4 describes a general-purpose interactive system, ECSTASY, for optimization of analog circuits. A forms-based, menu-driven user interface is proposed for easy problem formulation and user interaction. The implementation of optimization algorithms based on sensitivity computation and random search is discussed, and several design examples are provided. Chapter 5 presents an automatic transistor sizer for combinational

static CMOS circuits. The program uses a combined heuristic and nonlinear programming approach to size transistors. Results from the optimization of some benchmark circuits are shown, to justify the proposed approach. Chapter 6 summarizes the main contributions of this thesis and suggests some directions for future research.

Chapter 2

Survey of Existing Tools

2.1 Introduction

The use of optimization techniques in IC design has been a goal of the computer-aided circuit design community for a considerably long time. Many software tools for performance optimization have been designed, targeting at different circuit applications. This thesis focuses on interactive optimization systems that interface with general-purpose circuit simulators to optimize circuits whose detailed circuit behavior is of importance, and on transistor sizers that use macromodels, or interface with timing analysis tools, to optimize digital CMOS circuits.

Section 2.2 reviews three important interactive optimization systems. Section 2.3 surveys existing tools and techniques for transistor sizing. Both contributions and limitations of these tools are discussed.

2.2 General-Purpose Interactive Systems

2.2.1 Early Efforts

In the past years, advances in optimization theory and design methodologies based on interactive computing have led to the appearance of power-

ful general-purpose interactive optimization systems. These systems provide environments for designers to design and optimize electronic circuits in an interactive way. Specifically, the systems report undesirable results, if any, and allow designers to modify the circuit, relax design constraints, and change problem formulations.

Chronologically, there are three important contributions in interactive optimization: A0P [10], APLSTAP [11], and DELIGHT.SPICE [12]. These systems, although not widely used in industry, have proved the feasibility and usefulness of interactive optimization techniques in the design of complex ICs.

A0P is based on IBM's ASTAP [13] circuit simulator with adjoint sensitivity computation capability. The system allows the designer to specify, in ASTAP input language, a wide range of objective and constraint functions, and to request analysis, sensitivity computation, and optimization. To deal with multiple objectives, A0P employs a weighted-sum approach to combine objective and constraint functions into a single cost function, with constraints as penalty terms. The resulting function is then minimized using a variable metric rank-one method. The system has been used to minimize the memory cycle of a read-only memory cell.

APLSTAP is also based on ASTAP. It provides an APL environment with good interactive capabilities and ease of experimentation with various optimization algorithms. APLSTAP uses an interactive linear programming (LP) approach to capture the customary optimization practice of trading off multiple objectives and constraints. The system presents linear prediction of circuit performance, at the solution of LP step, to the designer, who then selects a maximally effective LP step to update the design. It places emphasis on the first few optimization steps, emphasizing design *improvement* rather than optimization. APLSTAP has been used to solve many general circuit optimization problems such as nominal design, modeling, and worst-case design [14].

DELIGHT.SPICE is based on the widely distributed circuit simula-

tor SPICE2 [15] with sensitivity computation capability. The environment is based on a structured programming language RATTLE (evolved from rational FORTRAN, RATFOR [16]) which allows the designer to describe design problems in a structured way. Optimization is placed in a framework of circuit design oriented problem formulation and places heavy emphasis on designer interaction. The multiobjective problem formulation employed by the system provides a means of effectively conveying the relative importance of the design specifications. A methodology for performing design tradeoffs using a graphical display is also provided.

As an environment for sophisticated designers with optimization background, DELIGHT.SPICE provides a a large library of optimization algorithms. By default, the system uses an algorithm which is an extension of Phase I/Phase II Method of Feasible Directions [17] [18]. The algorithm can handle functional constraints (i.e., constraints which depend, not only on design parameters, but also on an independent parameter such as temperature, time, or frequency) which are very important in IC design. The system has been used to improve the circuit performances of many industrial analog circuits and digital cells.

2.2.2 Shortcomings of Existing Approaches

An interactive circuit optimization system is composed of a user interface, optimization algorithms, and circuit simulator. While the system cannot work without any of the constituent parts, its success depends heavily on the user interface. In other words, the effectiveness of an interactive system is judged from the point of view of how much effort a designer spends in guiding the system to achieve a satisfactory design.

To use A0P, a designer has to specify the weights required in the problem formulation. Since design specifications may involve arbitrary functions with units ranging from nano-second to mega-herz, choosing a set of weights with such differing units is cumbersome. Unfortunately, these

weights affect the results of multiobjective optimization, as a result, the designer may have to spend a long time adjusting these weights before any noticeable improvement is achieved.

APLSTAP assumes that the designers have little optimization background but still places the responsibility for algorithmic control in their hands. Problems therefore arise when the selected LP step does not improve the design significantly or when the algorithm jams. Furthermore, the APL language suffers from being difficult to read and, as a result, it may be difficult for the designer to use it to formulate a design problem as a well-posed optimization problem.

DELIGHT.SPICE offers more flexibility, more transportability, and more mathematical sophistication than A0P or APLSTAP. However, from the user interface point of view, the system is not friendly. Specifically, the designer has to use the RATTLE language to prepare various problem description files. Although, as a programming language, RATTLE has many good features, it was originally designed for general purpose engineering optimization use [19], the designer is thus forced to specify rather tedious low-level constructs of the IC optimization problem formulation.

A common problem with these systems lies in the user model. The optimization algorithms in these systems involve numerous problem-dependent decisions which are reflected in certain parameters associated with the implementation of the algorithms (e.g., those for use in the computation of step size). Since, to use the algorithms effectively, the designer has to know optimization theory and be able to adjust these algorithm-related parameters, it is not uncommon for ordinary designers to face convergence problems even for simple circuits.

To summarize, existing interactive systems rely too heavily on the designer to pose a design problem as a standard nonlinear program and to manipulate the optimization algorithms. Recognition of these shortcomings motivated the idea of employing a new user interface and built-in optimization algorithms in ECSTASY, to be described in Chapter 4.

2.3 Transistor Sizers

2.3.1 Early Efforts

The speed of a digital IC is limited by its critical paths, i.e., paths with worst-case delay. Most approaches to transistor sizing are therefore based on optimizing critical paths. Since speeding up one path may slow down another due to the capacitive loading effects, achieving the optimum design requires the optimization of multiple paths simultaneously.

Analytic approaches to the optimization of delay or power consumption along a delay path have been addressed in many papers. Algorithms using different delay models for optimizing the propagation delay of an inverter chain driving a capacitive load are proposed in [20] and [21]. Hedenstierna and Jeppson [22] consider the slope of input waveforms, and proposed an analytic timing model for CMOS inverter chains. The model is used in the design of CMOS output buffers. Minimization of delays associated with driving and sensing signals from large capacitance paths for NMOS circuits is discussed by Mohsen and Mead [23]. The idea is to optimize the fanout factor of the driver stages, the gain of the input sensing stages, and the path voltage swing. Lewis [24] extends the idea to deal with general I/O interfacing problems in gate-array, semi-custom, and full-custom CMOS designs. Kang [25] describes an algorithm for the minimization of delay-area product under noise margin conditions for CMOS polycells. Glasser and Hoyte [26] use a simple macromodel for gate delays to minimize NMOS power consumption under delay constraints. Lee and Soukup [27] propose a method to minimize delay and area along a path of simple CMOS logic gates. The algorithm associates a scale factor with each gate for transistor sizing; all the transistors within a gate are therefore of the same size. Sizing is hence performed *per gate*. A closed form solution to size simple CMOS (library) cells is proposed in [28].

A heuristic approach to optimizing delay and power for NMOS

circuits is proposed by Trimberger [29]. Starting from the output node of a critical path, the algorithm computes the load of the output gate, then sizes the gate according to an optimum fanout factor [30]. The algorithm then goes backward to size the preceding stage with the output gate as the load. The procedure is repeated until the input of the path is reached. Power optimization is performed on paths of gates off the critical paths which can be slowed down without degrading the speed of the circuit as a whole. The sizes of the gates on these paths are decreased until the path delays are slowed down to that of the critical paths.

The hueristic algorithm of [31] sizes transistors per gate. Given a critical path violating the speed requirement, it picks a gate with the largest delay-per-area contribution. The widths of all the transistors within the gates are increased by one micron. This process is repeated until the worst case delay of the entire circuit satisfies the specification.

The TILOS heuristic by Fishburn and Dunlop [32] minimizes total active area subject to delay and minimum device size constraints. The algorithm selects iteratively the output that is failing to meet its user-specified delay by the greatest amount, and examines transistors along the worst-delay path leading to this output. For each transistor that could affect the delay of this path, the sensitivity of the delay with respect to transistor size is calculated. The transistor with greatest sensitivity is increased by multiplying it by a user-settable constant (greater than one), after which the static timing analysis is updated. This procedure is repeated until all outputs meet their specifications. The paper also points out that, under a simple delay model, the transistor sizing problem for static CMOS circuits is convex.

Both Hofmann [8] and De Micheli [7] employ TILOS-like heuristic sizing techniques, using table-driven approaches to characterize gate delays. Hofmann searches from outputs backward to inputs for a gate with the largest "sensitivity". The critical input devices of the gate are grown or shrunk by one unit at each sizing, according to the sign of the sensitivity. De Micheli searches for the driver gates with greatest "sensitivity" and increments the

sizes of the gates by one unit.

A more global approach that considers all design constraints simultaneously is used by many authors. In this approach, the transistor sizing problem is formulated as a constrained, nonlinear mathematical program. The delay constraints are evaluated either through repeated calls to timing analysis tools during optimization or through macromodeling performed prior to optimization. (DELIGHT.SPICE and APLSTAP can also be used to size transistors accurately, but the computation time limits their use to fairly small circuits only.)

Ruehli, et al [33] [34] [35] use a simple analytic macromodel for the delay and power of general MOS circuits. The design parameter in a gate is the width of the active device. A smoothing function is defined to deal with the nondifferentiability of circuit delays which are maximum functions of path delays. (The effect of the parameter controlling the sharpness of the smoothing function on the convergence is not investigated.) A Quasi-Newton method is then employed in the optimization using sparse matrix updating. The same optimization technique is used by Hedlund [36] [37] [38] to optimize delay and area for both NMOS and CMOS circuits. The gate delay is estimated by a very simple linear RC single time constant approximation without considering the capacitance contribution of the internal nodes. Transistor sizing is thus performed per gate by Hedlund.

An improved macromodel is used by Matson [39] to optimize power consumption for NMOS circuits. The behavior of each logic gate is modeled in a set of simple yet accurate formulas. Most of the modeling work is performed prior to optimization. The separability of power and delay functions of MOS circuits is exploited and a dual approach utilizing the Lagrangian is proposed to simplify the optimization problem.

Marple [40] optimizes layout area for CMOS circuits. The delay model is based on RC trees [41] and optimum circuits are defined in terms of graphical constructs. The delay-area optimization problem with physical layout constraints is then formulated as a nonlinear constrained optimiza-

tion problem which is solved using the augmented Lagrangian and projected Lagrangian algorithms. Obermeier and Katz [42] implement augmented Lagrangian algorithm and the TILOS heuristic in their program to size transistors with circuit performance evaluated by symbolic polynomials.

2.3.2 Shortcomings of Existing Approaches

Tools which associate with each gate a scale factor, and size transistors per gate generally work faster than those treating each transistor size as a separate design parameter. However, since it is necessary for optimum design to size each transistor according to its position within a gate [43], a transistor sizer of the latter type is often needed for high performance chip design.

The approach of using symbolic expressions or closed form solution to evaluate circuit performance can avoid repeated calls to timing analysis tools. However, existing algorithms make use of very complicated expressions and require large memory to store the expressions and the associated Jacobian matrix. Therefore, unless better analytic models can be devised, the approach can only be used in applications where the circuit size is small.

Both analytic and heuristic approaches to sizing transistors along a critical path offer the designer a quick way of improving circuit performance. These approaches are useful in applications where limited objective functions, such as delay or total active area, are desired. However, when the designer wants to minimize arbitrary objective functions such as layout area or delay-area product, it is difficult to generalize these algorithms to satisfy the wide range of design requirements. Furthermore, it is difficult to claim the optimality of the entire circuit design.

The nonlinear programming approach can optimize arbitrary objective functions subject to arbitrary design constraints. By considering all of the constraints simultaneously, this approach guarantees that, if the algorithm converges, then the solution is at least locally optimum. (As a conse-

quence, if the objective and constraints are convex functions, the solution is a global optimum.)

However, from a computational point of view, this approach has several drawbacks. Firstly, it lacks an efficient method of obtaining a good initial guess for the transistor sizes. Secondly, it solves the problem in an unnecessarily large dimensional space by treating all of the transistors as design parameters. Thirdly, if numerical convergence is a problem (which is not uncommon in nonlinear optimization), the result obtained may not yield a satisfactory design. Furthermore, the use of optimization algorithms in their standard form requires the objective function and the constraints to be continuously differentiable functions of the design parameters. If the constraint or the objective functions are expressed as circuit delays, then the mathematical expression of these delays is the maximum among the path delays, i.e., of the delays along all the paths from the input ports to the output ports. Since the *max* function is not continuously differentiable, versions of the optimization algorithms especially devised for nondifferentiable functions have to be used.

Recognition of the shortcomings of both heuristic and nonlinear programming approaches leads to the combined approach in Chapter 5.

Chapter 3

Algorithmic Considerations in IC Optimization

3.1 Introduction

In order to apply the numerical techniques of optimization theory to solve circuit optimization problems, it is necessary to bring together circuit design knowledge and optimization theory. In fact, blindly applying a general purpose optimization algorithm often results in premature termination of the algorithm.

As with all engineering analysis techniques, posing a circuit optimization problem as a nonlinear mathematical program is only an approximation to the original design problem. Many factors such as the selection of circuit performance criteria or the choice of design parameters may influence the quality of the results. Furthermore, although optimization theory has suggested many methods to solve a nonlinear program, in practice, the choice and implementation of a particular solution method, as well as the computational accuracy of the circuit analysis tools, may also affect the quality of the solution.

This chapter begins with the formulation of a general circuit optimization problem. By considering the techniques of semi-infinite program-

ming, the problem is simplified to a multiobjective constrained optimization program. Section 3.4 describes a finite termination Newton-type algorithm to find a feasible solution. Section 3.5 discusses the possibility of employing heuristic methods to solve for a feasible design. After a feasible solution is found, the multiobjective optimization problem is solved, in Section 3.6, by classical optimization algorithms. The final section addresses the numerical difficulties arising in circuit optimization and proposes a random search scheme to remedy the problem.

3.2 Problem Formulation

IC design is characterized by complex tradeoffs of multiple objectives and constraints. In general, the objectives and constraints are nonlinear functions of the design parameters, and, in some applications, these functions may involve an independent variable such as time, frequency, temperature, or power supply, ranging over a given interval. In contrast to the *ordinary* specifications which depend only on design parameters, these functions are commonly referred to as *functional* specifications. Mathematically, the design problem can be formulated as determining a design parameter vector $x \in R^n$ such that a *multiple*-objective optimization problem $\mathcal{P}$:

minimize:

$$f^j(x),\ j = 1, 2, ..., r$$

$$\phi^j(x, \omega),\ j = 1, 2, ..., s,\ \omega \in [\omega_{1_j}, \omega_{2_j}]$$

such that:

$$g^j(x) \leq 0,\ j = 1, 2, ...p$$

$$\varphi^j(x, \omega) \leq 0,\ j = 1, 2, ..., q,\ \omega \in [\omega_{3_j}, \omega_{4_j}]$$

is solved. In its general form, $\mathcal{P}$ is known as a *semi-infinite programming problem*, since a functional specification can be viewed as an infinite number of functions.

3.3 Semi-infinite Programming

Several authors have addressed techniques for solving a semi-infinite programming problem [17] [44] [45] [46] [47] [48] [49] [50]. Algorithms with guaranteed convergence properties require the computation of better and better approximations to local maxima of $\phi^j(x,\omega)$ and $\varphi^j(x,\omega)$ with respect to ω. Under mild assumptions on the continuity and differentiability of $\phi^j(x,\omega)$ and $\varphi^j(x,\omega)$ with respect to ω, these algorithms make use of the fact which is usually true in engineering design, that for each x, both $\phi^j(x,\cdot)$ and $\varphi^j(x,\cdot)$ have only a finite number of local maxima with respect to ω. Computing accurately such maxima is obviously an expensive proposition. Adaptive schemes thus have been devised to provide better and better approximations while the optimization algorithms progress towards a solution. These schemes discretize the ω axis with finer and finer meshes. In practice, however, convergent algorithms such as [45], while working well with a small discretization mesh, may jam in the early stages when the discretization is still coarse (although a globally convergent algorithm can be used to remedy the problem [50]). In addition, the overall speed of convergence may be affected by the quality of the approximations computed even in the early stage of the optimization algorithm. Indeed, semi-infinite programming techniques are still relatively immature compared to the standard finite dimensional nonlinear programming techniques.

Most of the design problems encountered in this research involve specification of the frequency and/or time domain behavior of the circuit. Often these specifications are formulated as infinite dimensional functions of the form introduced above. To perform simulation in the frequency domain with a circuit simulator, an IC designer usually has to provide a set of frequencies where the circuit response is to be computed. These frequencies are selected so that an accurate representation of the frequency domain behavior of the circuit is computed. This *manual* discretization of frequency can be considered to be the finest approximation the designer is interested in to

characterize the circuit. It is thus reasonable to use the mesh provided by the designer. In this thesis, it is assumed that, when ω represents variables other than time (e.g., temperature, frequency, or voltage), the discretization scheme provided by the designer can accurately represent the circuit behavior.

When a circuit specification is given in terms of its time domain response, a circuit simulator has to perform a transient simulation to evaluate the corresponding function. Any efficient circuit simulator automatically selects a time-step that yields a suitably accurate representation of the time-domain behavior of the circuit. In this case, it is natural to use the discretization of time selected by the circuit simulator as the mesh.

When the ω interval over which the functions are given is discretized, the optimization can be recast into a standard nonlinear mathematical programming problem of the form $\mathcal{Q}$:

minimize $f^j(x)$, $j = 1, 2, ..., t$

such that $g^j(x) \leq 0$, $j = 1, 2, ..., m$

Note that in digital applications where the specification functions, such as area, power consumption (which depends only on a *single* clocking frequency), or delay, are typically of non-functional type, the problem can often be cast in this form directly without discretization schemes.

3.4 Finding a Feasible Design

The first step in solving $\mathcal{Q}$ is to find a *feasible* design in the feasible region $\mathcal{F} = \{x | \psi(x) \leq 0\}$, where

$$\psi(x) \triangleq max\{g^j(x) | j \in \underline{m}\},\ \underline{m} \triangleq \{1, 2, ..., m\}$$

is the maximum function of all the constraints. Finding a feasible design for $\mathcal{Q}$ is very important in the context of circuit optimization, since in many situations, the design specifications only involve a set of inequality constraints.

Several algorithms have been proposed to solve the problem. One of the approaches is to use standard feasible directions algorithms to decrease progressively $\psi(x)$ until it becomes non-positive [9] [18]. While it can be shown that the method can find a feasible design in a finite number of iterations, it is rather slow. Another approach is to use Newton's method to compute a search direction which points toward the interior of the linearized feasible region. The method can generate a sequence of points which converge superlinearly to a point in $\mathcal{F}$. While some of the proposed algorithms, such as [51], do not converge in a finite number of iterations, some do [52] [53] [54]. Note that, although the rate of convergence is not relevant to an algorithm stopping in a finite number of iterations, several examples in [53] [54] show that it does contribute to the efficiency of the algorithm. In this thesis, the algorithm in [54] is used since it behaves well experimentally and is rather insensitive to the choice of algorithm-related parameters, in contrast with most feasible directions algorithms where the choice of these values affects convergence rather significantly.

The algorithm is based on the idea of finding a search direction $h \in R^n$, of minimum norm, which solves:

$$g^j(x) + < \nabla g^j(x), h > \leq -\epsilon, \; j \in \underline{m}$$

where $< \cdot, \cdot >$ represents inner product and $\epsilon \geq 0$ is chosen to be as large as possible. Since ϵ cannot be chosen *a priori*, and such an h may not exist, a modification of the basic idea is necessary. Consider

$$\hat{\psi}(x, h) \triangleq max\{g^j(x) + < \nabla g^j(x), h > | j \in \underline{m}\}$$

which is the first-order estimate of $\psi(x + h)$. Then the Newton step at x, if exists, is that h which solves:

$$min\{\|h\| | \hat{\psi}(x, h) \leq 0\}$$

However, this Newton step may not exist, i.e., the set $\{h | \hat{\psi}(x, h) \leq 0\}$ may be empty. The modified search direction h_ϵ which solves the quadratic program:

$$min\{\|h_\epsilon\| | \hat{\psi}(x, h_\epsilon) \leq \hat{\psi}_\epsilon^0\}$$

where:

$$\hat{\psi}_\epsilon^0 \triangleq max\{\hat{\psi}^0(x), -\epsilon\}$$

$$\hat{\psi}^0(x) \triangleq min\{\hat{\psi}(x, h) | h \in R^n, \|h\|_\infty \leq L\}$$

for $\epsilon \geq 0$ and L some suitably chosen large number (to ensure that the solution is bounded), is used instead. Then even if the Newton step does not exist, the step still is a descent direction for $\psi(x)$. These considerations have motivated Algorithm 3.1.

To ensure that the algorithm terminates in a finite number of iterations, [54] assumes that, for all x in the set $\{x|\psi(x) \geq 0\}$, $\hat{\psi}^0(x) - \psi(x)$, which is a first-order estimate of $\psi(x+h) - \psi(x)$, is negative (this ensures that $\psi(x)$ can be decreased at all points where it is non-negative). It then shows that, if $\{x_i\}$ and $\{\epsilon_i\}$ are infinite sequences generated by the algorithm, then there exists $\epsilon^* > 0$ such that $\epsilon_i = \epsilon^*$ for all i sufficiently large, and x_i converges superlinearly to an $\hat{x}$ satisfying $\psi(\hat{x}) \leq -\epsilon^*$. Since $\psi(\hat{x}) < 0$, it follows that there exists a finite i such that $\psi(x_i) < 0$, i.e., the inequalities are solved in a finite number of iterations. Note that, in [54], the condition checked to perform ϵ adjustment in Step 5 is: "If $\hat{\psi}^0(x_i) \geq -\epsilon_i$, then ..." However, since $\hat{\psi}^0(x_i)$ may be positive (when the Newton step does not exist) and, consequently, no matter how ϵ_i is adjusted, the condition: "$\hat{\psi}^0(x_i) \leq -\epsilon_i$" can never be satisfied, resulting in looping forever in the adjustment step for ϵ. The condition is thus modified here to avoid this problem.

3.5 Heuristic Methods

At this point, it is appropriate to discuss the approach of using heuristic methods to solve optimization problems. While optimization theory has provided numerical methods for solving general nonlinear programs,

Algorithm 3.1 (Finite Termination Newton Algorithm)

Data: $x_0 \in R^n, \epsilon' \in (0,1), L >> 1, \beta \in (0,1)$.

Step 0:

Set $i = 0$, $\epsilon_0 = \epsilon' \psi(x_0)$.

Step 1:

If $\psi(x_i) \leq 0$, *stop.*

Step 2:

Compute $\hat{\psi}^0(x_i), \hat{\psi}^0_{\epsilon_i}(x_i)$.

Step 3:

Compute search direction h_{ϵ_i} *using:*

$$h_{\epsilon_i} = argmin \{\|h\|^2 | \hat{\psi}(x_i, h) \leq \hat{\psi}^0_{\epsilon_i}(x_i)\}.$$

Step 4:

Set step size λ_i *to the largest value in the set* $\{1, \beta, \beta^2, ...\}$ *such that:*

$$\psi(x_i + \lambda_i h_{\epsilon_i}) - \psi(x_i) \leq \lambda_i [\hat{\psi}^0_{\epsilon_i}(x_i) - \psi(x_i)]/2.$$

Step 5:

Update $x_{i+1} = x_i + \lambda_i h_{\epsilon_i}$.

If $\hat{\psi}^0(x_i) < 0$, *then*

set ϵ_{i+1} *equal to the largest in the set* $\{\epsilon_i, \epsilon_i/2, \epsilon_i/4, ...\}$

such that $\hat{\psi}^0(x_i) \leq -\epsilon_i$.

else

set $\epsilon_{i+1} = \epsilon_i$.

Set $i = i + 1$.

Go to Step 1.

in practice, if specific knowledge can be utilized to exploit general circuit properties, then heuristic methods [1] may be effective in finding a feasible design.

A heuristic method is an exploratory problem-solving technique that utilizes circuit properties to improve the progress towards an acceptable solution. Oftentimes, based on intuition and experience, practicing designers use heuristics to analyze and improve the circuit performance.

In digital applications when delay constraints are involved, a common technique to speed up a circuit is to use large devices on critical paths and on output stages to drive large capacitive loads. The reason is that larger transistors have larger current driving capabilities and hence can charge and discharge node capacitance in shorter time, even though larger parasitic capacitance may be produced. Interestingly, in practical applications, it often suffices to widen a subset of transistors on critical paths to speed up the circuit. This is more efficient than solving a nonlinear program directly in the entire parameter space. From this observation and the fact that most high performance designs impose constraints on the circuit delay, the TILOS heuristic [32] exploits a general circuit property which does not depend on the particular circuit topology. Therefore, it is a good heuristic algorithm; it is effective in selecting and adjusting a set of design parameters to satisfy the delay constraints of a digital circuit, even though theoretical convergence is not guaranteed. (In Chapter 5, a scheme to speed up the TILOS algorithm is described.)

In the most general case, when additional constraints are to be satisfied, the result obtained from applying such a heuristic can be used as a good starting point which already satisfies the delay constraints. The problem of finding a feasible design for all of the constraints then becomes a constrained optimization problem which is to be iterated to find a point

[1] Although the *down-hill* process of most optimization methods are heuristic in nature, algorithms which are not based on rigorous mathematical treatment are called heuristics in this thesis.

where all of the constraints are non-positive.

By contrast, due to the complex nature of analog circuits, it is difficult to develop a heuristic that works for a large class of circuits. Recently, advances in analog synthesis techniques have made it possible to capture and store an experienced designer's knowledge as *rules* for estimating the performance of some class of circuits [2] [3] [4]. These rules are mostly based on first-order analytic models and are stored in the database of the synthesis systems. In theory, if the models are accurate enough, they can be used directly by optimization without performing circuit analysis. In practice, however, especially in complex analog designs, accurate analytic models may be difficult to obtain or too complex to handle. Therefore, good synthesis results are still not available for practical use. Good heuristics for analog design are yet to be developed.

From the CAD point of view, an algorithmic approach, if realizable, is often desirable due to its deterministic nature and predictable results. A good heuristic for solving a general problem thus should not simply contain a set of rules each corresponding to a certain type of circuit. Rather, it should be an algorithm that works for at least a large class of circuits, irrespective of the particular circuit topology. Therefore, instead of employing a set of rules as heuristic optimization algorithms, circuit optimization tools for analog design should be integrated with synthesis tools in a module generator through proper user interface so that a synthesis result can be used as an initial design.

3.6 Multiobjective Optimization

Once a feasible design has been obtained, it is desirable to optimize the multiple objective functions within the feasible region. However, unlike the classical nonlinear programming with a single objective function, the solution of a multiobjective optimization problem is difficult since, in problem $\mathcal{Q}$, it is not clear how to compare solutions x_1 and x_2 where it cannot be

claimed that $f^j(x_1) \leq f^j(x_2)$, for all j. Since only the designer, i.e., the decision maker, can judge whether or not a solution point is preferred to another, the concept of *optimal* solutions to multiobjective optimization problems is therefore closely related to the preference attitudes of the designer.

In multiobjective optimization, a commonly adopted concept of optimality criterion is *Pareto* optimality [14] [55]. By definition, a *Pareto point* is a point where no other point can cause one objective function to decrease without causing another to increase. More precisely, a point $x \in \mathcal{F}$ is called a Pareto point for the problem $\mathcal{Q}$ if there exists no other point $x' \in \mathcal{F}$ such that:

$$f^j(x') \leq f^j(x), j = 1, ..., t$$

where not all of the above inequalities are equalities. The same definition can be used to define a *Pareto critical point* if x' is restricted to a small neighborhood of x. Thus, by analogy with the case of dealing with a single objective function, Pareto points can be considered as global optimum points and Pareto critical points as local optimum points. It is easy to show that the best circuit design occurs at a Pareto point. Multiobjective optimization is therefore linked to finding Pareto points in $\mathcal{F}$ (although optimization algorithms can only hope to find Pareto critical points). Note that, since there is not enough information in the problem formulation to allow for the automatic selection of the "best" point, the choice among the Pareto points has to be left to the designer.

Many techniques have been proposed to convert a multiobjective optimization problem into a standard optimization problem with a single objective function [14] [56] [55]. One of the techniques is to combine the multiple objectives into a single weighted sum,

$$\Phi(x) = \sum_j w_j f^j(x)$$

where the weight w_j represents the designer's preference of minimizing $f^j(x)$. One of the problems with this approach is that adjusting the weights is very

cumbersome for the designer. Another problem is that, in general, not all Pareto points are realizable through the adjustment of these weights, thus excluding *a priori* some interesting solutions. Another approach is to convert the problem into a weighted *minimax* optimization problem. It can be shown that, by adjusting the weights interactively, this approach can obtain all of the Pareto points. However, to adjust a set of weights effectively, the designer has to wait until the completion of an optimization corresponding to these weights. Since in circuit optimization, it is very expensive to wait for even a complete optimization run, this approach is still cumbersome. In fact, a key point in multiobjective optimization is to extract additional information from the designer in the easiest possible way without forcing him/her to exclude points that may be interesting.

One of the contributions of DELIGHT.SPICE is to encourage the designer to specify a *good* value and a *bad* value for each objective (and constraint) function. The good and bad values of a specification function represent the level of the designer's satisfaction with respect to that function. This methodology provides a simple way for the designer to perform tradeoffs. The idea is adopted in this research. However, the good and bad values are used not only as a means of trading off and scale the objectives and constraints, but also as the goals to achieve at different phases of optimization.

The solution method for $\mathcal{Q}$ consists of five phases. Phase I searches for a solution point such that the bad values of all of the constraints are satisfied. Phase II attempts to satisfy the good values of the constraints. Phase III improves the design until the bad values of all of the objective functions are satisfied, within the feasible region corresponding to the good constraint values. Phase IV pushes the design to a point within good objective values. At this point, since all the good values, which quantify the full degree of the designer's satisfaction, have been met, it is reasonable to assume that further improvements in all the objective functions are equally valued. Phase V thus combines all the objectives, scaled according to the good and bad

values, into a single equally-weighted sum. The weighted sum is minimized within the feasible region formed by the good values of all the objective and constraint functions. Note that, in the first four phases, if the assumption made in Algorithm 3.1 holds, then the desired solutions are achieved in a finite number of iterations.

Since the same technique to obtain a feasible point can be used in all first four phases, the remaining problem is a standard constrained optimization problem $\mathcal{R}$:

minimize $f(x)$

such that $g^j(x) \leq 0,\ j = 1, 2, ..., m$

where only one objective function is to be optimized. Hence classical algorithms for solving nonlinear constrained optimization problems can be applied.

3.7 Feasible Directions Algorithms

Before discussing the algorithms to solve $\mathcal{R}$, it should be stressed that, since it is generally impossible to find a global optimum to $\mathcal{R}$, one must usually be satisfied with computing a stationary point which satisfies a first order necessary optimality condition. However, since even this computation may require infinite time, practical algorithms often settle for an approximation to a solution and are stopped when no more noticeable improvements in the objective function are possible.

There has been a great diversity in methods for solving a nonlinear constrained optimization problem. Each method has advantages and disadvantages and is best applied where its advantages can be fully exploited. While evaluations of nonlinear optimization algorithms are abundant [57] [58] [59] [60], and many good methods, such as *sequential quadratic programming* (SQP) methods, have been used in many applications, the *methods of feasible directions* (MFD) are used in this research due to the following considerations [61]:

- These methods generate feasible points in the search procedure. Since in many practical design situations, especially in interactive circuit optimization, convergence is not affordable and the process often has to be terminated before reaching a stationary point, it is desirable that the termination point is feasible. This final point, being feasible, may thus represent an acceptable solution to the practical problem that motivated the nonlinear program.

- These methods are globally convergent, i.e., under rather mild conditions, convergence to a point satisfying some necessary conditions for optimality is guaranteed, independent of the initial value of the design parameters. Thus, if they generate a convergent sequence, the accumulation points of that sequence are often at least local constrained minima. In contrast, several optimization algorithms have only local convergence properties and must start sufficiently close to a point that satisfies optimality conditions in order to converge to it.

- These methods do not rely on special problem structure, such as convexity, and hence are applicable to general nonlinear programming problems.

In fact, MFD has been used in DELIGHT.SPICE to solve many circuit optimization problems successfully. The algorithm used is based on [18] and has been enhanced by Tits, Nye, and Sangiovanni-Vincentelli. They improved the computational efficiency by special schemes to handle many practical difficulties, such as box constraints, arising in circuit optimization environment [62].

The algorithm has a nice geometrical interpretation, and is based on the observation that, if $\epsilon \geq 0$, x is a feasible point, and $J_\epsilon(x)$ is the index set of the ϵ-*active* constraints:

$$J_\epsilon(x) \triangleq \{j \in \underline{m} | g^j(x) \geq -\epsilon\},$$

then the vector,

$$h(x) = -Nr\ co\{\{\nabla f(x)\} \cup \{\nabla g^j(x), j \in J_\epsilon(x)\}\},$$

where $Nr\ co$ denotes the nearest point to the origin in the convex hull of a set, is a feasible descent direction for $f(x)$. In fact, the search direction $h_\epsilon(x)$ used by the algorithm is computed by:

$$h_\epsilon(x) = \frac{h(x)}{\|h(x)\|}$$

By defining an optimality function $\theta_\epsilon(x) \triangleq -\|h(x)\|$, it is easy to show that, at a stationary point $\hat{x}$, $\theta_0(\hat{x}) = 0$. The algorithm is formalized in Algorithm 3.2. It can be shown that, if the initial point is feasible, then the algorithm generates a sequence of feasible points. Furthermore, any accumulation point (or the last point, if the sequence is finite) of that sequence is a stationary point of $\mathcal{R}$.

Recently, MFD algorithms with a superlinear rate of convergence have been developed [63] [64]. The algorithm in [64] is particularly attractive since, at each iteration, it only needs to solve three sets of linear systems of equations, as compared to that of [63] where two quadratic programs and a linear least squares problem must be solved at each iteration. Since the algorithm is very complicated and is presented in detail in [64], only the essence is described here.

The idea is to use a quasi-Newton method to solve the system of equations in the Kuhn-Tucker first order necessary optimality conditions:

$$\nabla_x L(x, \lambda) = 0$$

$$\lambda_j g^j(x) = 0, j \in \underline{m}$$

where $L(x, \lambda) \triangleq f(x) + \sum \lambda_j g^j(x)$ is the Lagrangian and λ the Lagrange multiplier vector associated with $\mathcal{R}$. Let:

H = estimate of the Hessian of L

x = current estimate of the solution to $\mathcal{R}$

h^0 = search direction

Algorithm 3.2 (Basic Feasible Directions Algorithm)

Data: $x_0 \in \mathcal{F}$, $\epsilon_0 > 0$, $\alpha, \beta \in (0,1)$, $\gamma > 0$.

Step 0:

Set $i = 0, x_i = x_0$.

Step 1:

Set $\epsilon = \epsilon_0$.

Step 2:

If $\theta_0(x_i) = 0$, *stop.*

Step 3:

Compute $h_\epsilon(x_i), \theta_\epsilon(x_i)$.

Step 4:

Set ϵ *to the largest value in* $\{\epsilon, \epsilon/2, \epsilon/4, ...\}$ *such that:*

$$\theta_\epsilon(x_i) \geq -\gamma\epsilon.$$

Step 5:

Set step size λ_i *to the largest value in the set* $\{1, \beta, \beta^2, ...\}$ *such that:*

$$\psi(x_i + \lambda_i h_\epsilon(x_i)) \leq 0$$

$$f(x_i + \lambda_i h_\epsilon(x_i)) - f(x_i) \leq \alpha\lambda_i\theta_\epsilon(x_i)$$

Step 6:

Update $x_{i+1} = x_i + \lambda_i h_\epsilon(x_i)$.

Set $i = i + 1$.

Go to Step 1.

μ = current estimate of λ

λ^0 = next estimate of λ

then each quasi-Newton iteration yields (h^0, λ^0) which solves the linear system,

$$Hh^0 + \nabla_x L(x, \lambda^0) = 0$$

$$\mu_j < \nabla g^j(x), h^0 > + \lambda_j^0 g^j(x) = 0, j \in \underline{m}$$

which is a first-order Taylor series approximation to the previous system of equations about (x, μ). The solution h^0 can be shown to be a descent direction for f at x if $\mu_j > 0$, $g^j(x) < 0$, for all j, and H positive definite.

A closer look at the system of equations reveals that, as any $g^j(x)$ approaches zero, h^0 tends to a direction tangent to the feasible set. Therefore, to ensure feasibility at each iteration, after h^0 is obtained, a small negative term $-||h^0||^\nu, \nu > 1$, which tends to zero faster than h^0, is added to the right-hand side of the last m equations of the previous linear system. The result is a new system of equations in (h^1, λ^1):

$$Hh^1 + \nabla_x L(x, \lambda^1) = 0$$

$$\mu_j < \nabla g^j(x), h^1 > + \lambda_j^1 g^j(x) = -\mu_j ||h^0||^\nu, j \in \underline{m}$$

where h^1 is solved for as a correction direction. Note that the introduction of μ_j in the right-hand side of the equations reflects the fact that the correction is performed only close to the constraint boundaries. However, since h^1 may not be a descent direction for f, a search direction of the form:

$$h = (1 - \rho)h^0 + \rho h^1$$

is computed, with $\rho \in [0, 1]$ as large as possible, such that

$$< \nabla f(x), h > \leq \theta < \nabla f(x), h^0 >$$

is satisfied for some $\theta \in (0,1)$. It can be shown that, with such a search direction, the convergence properties of the quasi-Newton iteration are preserved.

To see the step size rule, it is necessary to understand a problem, called *Maratos effect* [65], arising in some SQP-type methods for solving $\mathcal{R}$. Consider the method of Han-Powell to combine the quasi-Newton approach with an absolute-value penalty function and sequential quadratic programming. The algorithm proceeds as follows [61]:

Step 0. Start with an initial point x_0 and an initial positive definite matrix H_0. Set $k = 0$.

Step 1. Compute the search direction h_k by minimizing the quadratic function: $\frac{1}{2}h^T H_k h + \nabla f(x_k)h$, subject to the constraints: $\nabla g(x_k)h + g(x_k) \leq 0$, where h^T represents the transpose of h.

Step 2. Stop if $h_k = 0$; the current point satisfies the first-order necessary condition for a solution to $\mathcal{R}$.

Step 3. Compute a step size $\alpha_k \in [0,1]$ by performing a line search along the direction h_k, using the absolute-value penalty function: $P(x_k) = f(x_k) + \sum \mu_j |g^j(x_k)|$ as a merit function, where the parameters $\mu_j, j \in \underline{m}$ are set to some sufficiently large positive constants.

Step 4. Set $x_{k+1} = x_k + \alpha_k h_k$, and update H_k so that it is positive definite (typically using a modification of the BFGS formula [61]).

This algorithm tries to generate the matrices $\{H_k\}$ so that the search directions $\{h_k\}$ satisfy the superlinear convergence condition:

$$\lim_{k \to \infty} \frac{\|x_k + h_k - x^*\|}{\|x_k - x^*\|} = 0$$

where x^* is a solution to $\mathcal{R}$. It can be shown that under some standard assumptions, if the initial point is sufficiently close to a solution and the step sizes taken equal to unity, then the algorithm converges superlinearly [66].

Unfortunately, the assumption of unit step size may not hold in some cases. In fact, examples have been found that, even when x^* is close, i.e., when $\|x_k + h_k - x^*\| \ll \|x_k - x^*\|$ and $\|x_k - x^*\|$ is small, the step size $\alpha_k = 1$ may not be acceptable (because it causes the merit function to increase, rather than decrease) [65]. This destroys the superlinear convergence.

One of the methods to remedy the problem, proposed by Polak and Mayne [67], is to modify the search direction so that searching along a suitably defined "arc" (tangent to the search direction computed in the previous SQP algorithm) preserves the superlinear convergence. This idea is also adopted by Panier, Tits, and Herskovits [63] [64]. They compute a correction direction $\bar{h}$ to "bend" the search direction towards the feasible region, and the search is performed along the arc $x + \alpha h + \alpha^2 \bar{h}$, for α the largest value in the set $\{1, \beta, \beta^2, ...\}$, where $\beta \in (0, 1)$. It is shown in [64] that the algorithm converges superlinearly to a stationary point with all the iterates in the feasible region.

However, since this algorithm requires a full $n \times n$ matrix to approximate the Hessian for use in quasi-Newton iteration, for large problems in digital applications, it may need large memory, and sparse updating techniques. Therefore, in this research, the algorithm is only used in analog design where the number of design parameters is fairly small; the enhanced version of MFD [62] is employed in transistor sizing for digital circuits. (In fact, the SQP method of Han-Powell has once been studied and implemented in this research. It is observed that, the iterates generated by the algorithm vary rather "wildly" in the parameter space, which is often not acceptable in practice. In circuit optimization, most of the function values are obtained through performing circuit simulation. If the values of the design parameters are too far away from that of a good "nominal design", e.g., a very small resistor value, numerical difficulties may arise in simulation. As a result, even though the simulation of the initial design converges easily, intermediate iterates may be too close to or even in the infeasible region, and are difficult to simulate. For this reason, the method is not used in the final

implementation.)

3.8 Numerical Difficulties Arising in Circuit Optimization

In theory, algorithms based on gradient information can compute descent directions of the objective functions and are effective in solving optimization problems. In practice, however, numerical difficulties often arise and cause premature termination of these algorithms. The problem can be traced to poor problem formulation and implementation difficulties, assuming that no programming errors exist.

The formulation of a circuit optimization problem consists of three parts: the selection of performance criteria, the choice of design parameters, and the definition of how the circuit performance is related to the design parameters. Each part influences the quality of the solution. As an example, it is important to include in the problem formulation all the design parameters that have significant effects on circuit performance, since the exclusion of some of them may lead to suboptimal solutions. In fact, problem formulation plays the key role in the success of circuit optimization. However, since some of the factors influencing problem formulation are related to user interface design and the designer's knowledge, and will not be considered here, it is assumed that the formulated problem is a proper representation of the design problem.

Unfortunately, such problems may be ill-conditioned in the numerical sense such that, when standard numerical methods are applied, satisfactory solutions are difficult to obtain. For instance, in Algorithm 3.2, a feasible directions algorithm, the search direction is determined by the gradient vectors of the objective and ϵ-active constraint functions. If the magnitudes of these gradient vectors are not balanced, then the search direction will be dominated by the vector with the smallest magnitude. Furthermore, a mere

scaling of a constraint by a scalar can move it in or out of the ϵ-active set. Both affect the search direction computation and hence the convergence. However, practical circuit optimization problems may involve specifications, such as amplifier gain and settling time, possibly with different orders of magnitude in both functions and their gradient vectors. Such problems arise in all but the most trivial problems formulated by even the most experienced designers and cannot be considered pathological. Schemes to overcome the problem thus should be standard components of circuit optimization algorithms. In fact, this problem has motivated the development of many methods to enhance the efficiency of optimization. Unfortunately, most existing schemes are empirical and may fail in some cases. User interaction is thus desirable to guide the optimization program towards a successful design.

Implementation difficulties have come from several sources. Firstly, most algorithms require the tuning of some algorithm-related parameters. When a problem is formulated as a nonlinear program, it is often possible to tune these parameters experimentally to give the best performance. Unfortunately, when the algorithms used are to optimize a wide variety of circuits, it is difficult to find the best values for these parameters to "hard-wire" into the implementation. Secondly, circuit analysis tools have inherent error tolerances. The finite precision of function and sensitivity evaluations may generate "noise" which misleads the algorithm, especially when a finite difference approximation is employed to compute the gradient vectors.

The numerical difficulties mentioned above may cause optimization algorithms to jam at some point. For example, it may not be possible to compute a descent direction from the gradient information. As a result, in many cases, the solution of the optimization problem cannot be carried out by a single procedure. Since, for the purpose of this research, it is desirable that the algorithms be transparent to the designers, it is proposed that these difficulties be remedied by reverting to methods, such as random search, which are less sensitive to these numerical problems.

In this section, two main causes of numerical difficulties, scaling and

finite difference approximation, are analyzed, and a random search scheme is proposed to continue the optimization run automatically when numerical difficulties arise.

3.8.1 Scaling

Scaling refers to the relative magnitude of the quantities that appear in the optimization problem to be solved. Its importance and difficulties have been emphasized by Gill, Murray, and Wright [68]:

> The term "scaling" is invariably used in a vague sense to discuss numerical difficulties whose existence is universally acknowledged, but cannot be described precisely in general terms. Therefore, it is not surprising that much confusion exists about scaling, and that authors tend to avoid all but its most elementary aspects.

Luenberger [61] shows that the convergence rate of many nonlinear programming algorithms depends on the condition number of the Hessian $\nabla_x^2 L(x^*, \lambda^*)$ where x^* is the optimum solution, λ^* an optimum multiplier, and L the Lagrangian function of the problem. To perceive this, consider the simplest case of applying the method of *steepest descent* to solve a quadratic program: $q(x) = \frac{1}{2}x^T Q x - b^T x$, where Q is an $n \times n$ symmetric positive definite matrix with eigenvalues $0 < \lambda_{min} = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n = \lambda_{max}$. The method of steepest descent is defined by the iterative process:

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

where α_k is a nonnegative scalar minimizing $q(x_k - \alpha \nabla q(x_k)^T)$. Obviously, q is strictly convex and has a unique minimum at $x^* = Q^{-1}b$. It is easy to show that, the method of steepest descent has linear rate of convergence and that the *asymptotic error constant* is:

$$\beta \triangleq \lim_{k \to \infty} \frac{\|q(x_{k+1}) - q(x^*)\|}{\|q(x_k) - q(x^*)\|} \leq (\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}})^2 = (\frac{r-1}{r+1})^2$$

where $r = \lambda_{max}/\lambda_{min}$ is the condition number of Q. Note that the asymptotic error constant β gives the factor of reduction in the error at each iteration. The smaller β is, the more rapid the convergence is. It is clear that, when the condition number r becomes large, β can be close to unity, resulting in slow convergence.

Since second derivatives are costly to compute in circuit optimization and good approximations to x^* and λ^* are generally unknown, choosing scale factors to reduce the condition number is difficult. It is for this reason that most scaling schemes are heuristic in nature [68]. Unfortunately, these schemes may work in some cases but fail in other cases [69]. Reliable automatic scaling methods thus remain a research topic. Nevertheless, in practice, scaling should be a regular component of the problem formulation under all circumstances.

The first step in scaling is to scale the objective and constraint functions. Consider a nonlinear program where the constraint functions $g^j(x), j = 1, 2, ..., m$ have different orders of magnitude, e.g., in an operational amplifier, the settling time is of the order of 10^{-6} (micro-second) and the unity-gain bandwidth is of the order of 10^6 (mega-herz). Since most algorithms test for a constraint to be active using absolute tolerances of the form: $g^j(x) \leq \epsilon$ where ϵ is a small positive number, if $g^j(x)$ is very large in magnitude, then a small perturbation in x may have large effect on $g^j(x)$ and hence affects the selection of $g^j(x)$ as an active constraint. Therefore, it is desirable to scale all of the constraint functions so that their magnitudes are near unity. On the other hand, if two objective functions have different orders of magnitude, then the optimization algorithm will tend to minimize the objective with larger magnitude (depending, of course, on the relative sizes of the gradients of the functions).

DELIGHT.SPICE suggests a scheme for scaling the problem specification functions, i.e., the objectives and constraints. For each specification function $f(x)$ which is associated with a good value and a bad value, the value of $f(x)$ is scaled to:

$$\frac{f(x) - GoodValue}{BadValue - GoodValue}$$

This scheme not only normalizes all the specification functions, but also correlates 0 and 1, respectively, to the designer's good and bad values so that it is always desirable to achieve a decrease of the normalized functions. However, in some cases, the designer may only need to achieve a goal which is specified by a set of desirable specification values. In this case, the good and bad values may be viewed as equal, and therefore, it does not make sense to correlate them to 0 and 1, respectively. Thus, to avoid computational problems and to normalize each specification function relative to its desired goal, i.e., the good (or bad) value, the scaling scheme in this thesis is modified to:

$$\frac{f(x) - GoodValue}{(GoodValue + BadValue)/2}$$

Of course, since this function scaling may affect the magnitude of the corresponding gradient vector and this, in turn, may affect search direction computation, convergence will be influenced. Hence, it is important for the designer to make tradeoffs between the specifications during optimization runs, if some specification functions cannot be satisfied.

The second issue in scaling is parameter scaling, which is even more crucial to obtaining reasonable convergence in real situations. Since the design parameters in a circuit optimization problem may have widely different magnitudes, it is desirable to scale the parameters, as is often recommended in practical optimization. The purpose is to improve the convergence. To be more specific, consider the problem of minimizing $f(x)$. The effect of changing the scales of the parameters is equivalent to performing a linear transformation: $x = Ay$, where y is the new parameter vector. The problem is thus equivalent to finding y to minimize $f(Ay)$. But then the Hessian of the new problem becomes $A^T H(x)A$, where H is the Hessian of $f(x)$. It is hoped that the condition number of the new Hessian is changed in such a way as to improve the convergence.

To this end and to exploit the designer's experience and intuitive knowledge of the problem, DELIGHT.SPICE employs a *uniform parameter influence rule* to let the designer manually adjust parameters such that a change in each parameter by its nominal variation influences the most binding objectives and constraints to roughly the same degree. Although this scheme can be effective for competent designers, it is difficult to use if they do not have a clear idea of what the nominal variations are. This is especially true in complex designs. Automatic scaling is hence preferred in practical optimization.

Many researchers have recommended scaling parameters to a value near unity [68] [70] [71], according to the upper and lower bounds of the design parameters estimated by the designer. The scheme is adopted in this thesis to scale each parameter to between 0 and 1 by using its *nominal value* computed as the mean value of its upper and lower bounds (Note that these parameter bounds are usually needed if the search direction calculations require the use of linear or quadratic programming subroutine. For example, in Algorithm 3.1, the computation of $\hat{\psi}^0(x)$ requires solving a linear programming problem, and the computation of h_ϵ requires solving a quadratic programming problem. To use standard subroutines such as from those Harwell library, it is necessary to specify the parameter bounds). In practice, while the initial bounds of the parameter values may not be good enough to provide good nominal values, the designer still can adjust these values through interaction. For example, if the optimization algorithm pushes against a lower or upper bound in an attempt to achieve a better result, and the designer is willing to compromise the original limits, then these values may be changed. It should be stressed that, numerical difficulties still may arise if information provided by the designer, such as good and bad values, is improper.

3.8.2 Finite Difference Approximation

Accurate sensitivity computation is essential to the success of using a gradient-based optimization algorithm. However, in applications where sensitivity is difficult or expensive to compute, or when high precision is not important, the finite difference scheme is often employed to approximate the sensitivity information. This method is easily implemented, as it requires only repeated calls to the circuit analysis tools, with design parameter values perturbed from the nominals.

One of the problems associated with this method is the high computational cost. As an example, for a circuit with n design parameters, it is necessary to call circuit analysis tools $n+1$ times to get a sensitivity vector. In analog designs when time-consuming transient analysis is involved, the computational cost may be too high, even if there are only a few design parameters. In transistor sizing for a digital design, this approach is obviously unacceptable since the number of design parameters is often very large (over 100).

Another problem is that, due to the finite tolerance control in most circuit simulators such as SPICE, the inaccuracies incurred in gradient computation may invalidate the search direction computation and hence make the nonlinear optimization technique useless. This problem is most serious in applications involving transient analyses for analog design. To be precise, suppose the *forward difference* scheme (with a finite difference interval Δx) is used to estimate the first derivative:

$$g'(x) \approx \frac{g(x+\Delta x) - g(x)}{\Delta x}$$

where $g : R \rightarrow R$. However, since each circuit simulator has an internal error tolerance control, the values of $g(x)$ and $g(x+\Delta x)$ in the expression for finite difference approximation are actually the *computed* values:

$$\bar{g}(x+\Delta x) = g(x+\Delta x) \pm \epsilon |g(x+\Delta x)|$$

$$\bar{g}(x) = g(x) \pm \epsilon |g(x)|$$

where ϵ is the *relative error tolerance* in circuit simulation (default value is 0.1 % in SPICE). Since by Taylor expansion,

$$g(x+\Delta x) = g(x) + \Delta x g'(x) + \frac{\Delta x^2}{2} g''(\xi)$$

where $\xi \in [x, x+\Delta x]$, the computed finite difference, when substituted into these expressions, differs from the exact first derivative by an amount:

$$\frac{\Delta x}{2}|g''(\xi)| + \frac{1}{\Delta x}\epsilon\{|g(x)| + |g(x+\Delta x)|\}$$

As a result, choosing a Δx too large or too small can result in large errors. While automatic algorithms exist for determining a good finite difference interval Δx throughout the course of optimization [68], the adjustment of Δx requires additional simulation and does not actually eliminate the inherent finite errors.

It must be stressed here that, in cases where $|g(x)|$ is large, the error may not be acceptable even for a 1 % finite difference interval $\Delta x/x$. Furthermore, although it is possible to control this term by controlling the relative tolerance ϵ, the convergence of simulation may be slowed down, especially for transient analysis. Experience with SPICE shows that Δx should not be below a few percent for transient analysis, although it may be as low as 0.1 % for AC analysis, when function values are small.

3.8.3 A Random Search Scheme to Remedy the Problem

As is clear from the previous sections, gradient-based algorithms may not work well in practice when numerical difficulties arise. More robust methods, such as random search, should thus be provided in a system in which optimization algorithms are hidden from the designers. While many random sampling schemes have been proposed in the literature [72] [73] [74] [56], in this thesis, a *controlled random search* is proposed.

Consider using the pure random search to find a feasible design. Let $\mathcal{A}$ be the entire parameter space and $\mathcal{F} = \{x|\psi(x) \leq 0\}$ be the feasible

region. To find a feasible point, a pseudo-random number generator is used to generate random trial points which are uniformly distributed in the parameter space. The trial points are used to evaluate the value of $\psi(x)$ until it becomes non-positive. Clearly, if $m(\mathcal{F}) < \infty$, where $m(\cdot)$ represents the *Lebesgue measure*, then the probability of obtaining a feasible design is equal to $\alpha = m(\mathcal{F}) \ / \ m(A)$.

In general, since random search requires only function evaluations and can be applied to any functions, smooth or not, the method is very reliable in yielding improved designs and is very easy to code (It seems this technique will be valuable in the future when computing power becomes cheaper). However, the method may not be as efficient as gradient-based deterministic algorithms, since the probability of obtaining a feasible design can be low, especially in applications such as transistor sizing where the number of design parameters is large. Experience shows that, waiting for a feasible design through pure random sampling is often not acceptable when time-consuming circuit simulations are required at each trial point.

However, the idea can be modified and used as a robust algorithm in analog design. In fact, it can be used to continue the optimization runs when gradient-based algorithms encounter numerical difficulties. To see this, consider the efficiency of the algorithm. Suppose N trial points are sampled uniformly in the parameter space. Then the probability that at least one point lies in $\mathcal{F}$ is:

$$1 - (1 - \alpha)^N$$

Unfortunately, the feasible region is in general unknown. Thus the probability α is not predictable.

In analog design, the initial design may be quite close to a feasible region, and it is desirable to improve the design from that point. A good approach to obtaining a feasible design thus seems to be to avoid sampling in the "bad regions" experienced from the past trials. This consideration leads to a *controlled random search* algorithm.

Algorithm 3.3 (Controlled Random Search Algorithm)

Notation: $x^j = j$*-th component of* x. $x_{min} \leq x \leq x_{max}$.

Data: $x_0 \in \mathcal{A}$, $\epsilon_0, \epsilon_{min} \in (0,1), \epsilon_{min} < \epsilon_0$, $N >> 1$.

Step 0:

Set $i = 0, \epsilon = \epsilon_0, x_i = x_0$, *failure* $= 0$.

Step 1:

If termination criterion is satisfied, stop.

Step 3:

Generate a random trial point y_i *in the neighborhood of* x_i *such that:*

$$y_i^j = x_i^j + \epsilon\xi(x_{max}^j - x_{min}^j), j = 1, 2, ..., n$$

where ξ *is a random number in* $[-0.5, +0.5]$.

Step 5:

If acceptance criterion is satisfied, then

set $x_i = y_i$, *failure* $= 0$.

else

set failure $=$ *failure* $+ 1$.

Step 6:

If failure $> N$, *then*

set $\epsilon = \epsilon^2$, *failure* $= 0$.

Step 7:

If $\epsilon < \epsilon_{min}$, *then*

generate a random point $x_0 \in \mathcal{A}$, *and go to Step 0.*

else

set $i = i + 1$, *and go to Step 1.*

One way of improving the probability of being in $\mathcal{F}$ is to limit the sampling of random points to a region $\mathcal{A}' \subset \mathcal{A}$ so that $\mathcal{F} \subset \mathcal{A}'$ and hence $m(\mathcal{A}') \leq m(\mathcal{A})$. In fact, it is possible to limit the search space by "windowing" dynamically the parameter space. Suppose x_0 is the initial value for the design parameters. If x_0 is not a local minimum, there must be a region $\tilde{\mathcal{A}}(x_0)$ where $\forall x \in \tilde{\mathcal{A}}(x_0)$, $\psi(x) < \psi(x_0)$. Now pick a δ-neighborhood of x_0: $N_\delta(x_0) \triangleq \{x | \|x - x_0\| \leq \delta\}$. Under the previous assumption, the set $N_\delta(x_0) \cap \tilde{\mathcal{A}}(x_0)$ is nonempty. Therefore, if $\psi(x)$ is continuous, then $m(N_\delta(x_0) \cap \tilde{\mathcal{A}}(x_0)) > 0$. The idea is to limit the random search to $N_\delta(x_0)$ and to stop after a point x_1 is produced so that $\psi(x_1) < \psi(x_0)$; x_1 is then chosen as the next base point for random search. If the random search fails to produce such x_1 after a certain number of trials, then δ is reduced and the procedure is repeated. However, if the random search still fails when δ is reduced to a very small value, then it is assumed that the current design is a local minimum of $\psi(x)$. A point in $\mathcal{A}'$ is then chosen randomly and the process restarted. Algorithm 3.3 formalizes this procedure.

The algorithm can be applied both to find and to optimize a feasible design. Referring to Algorithm 3.3, to find a feasible design, the *termination criterion* at i-th iteration is: $\psi(x_i) \leq 0$, and the *acceptance criterion* for the trial point y_i is: $\psi(y_i) < \psi(x_i)$. Note that, in case the feasible region is empty, the design can still be improved iteratively. To optimize a feasible design, the termination criterion is when a stationary point is "detected", i.e., when $\epsilon \leq \epsilon_{min}$, and the acceptance criterion is: $y_i \in \mathcal{F}$ and $f(y_i) < f(x_i)$.

Experience shows that, it is effective to set N to fifty times the number of design parameters and to limit the initial search space to within 10% of each parameter dimension, i.e., $\epsilon = 0.1$.

Chapter 4

A General-Purpose Interactive Optimization System

4.1 Introduction

The value of interactive nonlinear programming as a general optimization framework for complex IC design has been demonstrated in earlier efforts using this framework. But while powerful, the approach is limited by difficulties designers have to face in posing design problems of interest as nonlinear programs, and in controlling convergence of the optimization algorithms implemented.

ECSTASY is an experimental system to alleviate these difficulties. The system provides a forms-based, menu-driven user interface for problem formulation and user interaction, as well as built-in optimization algorithms, and a simulation interface to SPICE3 [75]. Fig. 4.1 shows the system configuration. On start-up, the system reads in a SPICE3 input deck provided by the designer to evaluate the initial circuit performance. Through the user interface, the designer can easily describe a design problem. The system then transforms the design problem into a standard formulation and solves it using built-in optimization algorithms. The algorithms interface with the circuit simulator to improve the circuit performance iteratively. During the

optimization runs, the designer is informed of the status of the circuit performance. They are able to interact with the system to trade off problem specifications until satisfactory results are obtained.

Gradient-based algorithms are used to begin the problem solution in Section 4.2. If numerical difficulties arise, a random search algorithm is automatically invoked, to enable the solution process to continue. The gradient-based algorithms implemented are provably convergent, but require the computation of circuit sensitivities. Techniques for this computation, for arbitrary specification functions, are detailed in Section 4.2. Section 4.3 describes the user interface design. In particular, the design issues of forms-based interface, simulation control, and user interaction are detailed. Section 4.4 describes the interfacing of the SPICE3 circuit simulator to the optimization algorithms. Section 4.5 presents four problem examples, which demonstrate the performance of the proposed algorithms.

4.2 Optimization Algorithms

As is clear from Chapter 3, the general formulation of an IC optimization problem is a semi-infinite multiobjective program $\mathcal{P}$. Under the assumption that the designer can approximate the problem with accurate discretization schemes, the problem can be transformed into a finite-dimensional multiobjective optimization program $\mathcal{Q}$. In ECSTASY, the problem is solved in five phases, with the first four solving a set of inequality constraints, and the last a classical constrained optimization problem.

Based upon the considerations discussed in Chapter 3, the algorithms of Mayne and Sahba [54], for solving a set of inequalities, and of Panier and Tits [64], for solving a constrained optimization problem, are employed as built-in optimization algorithms.

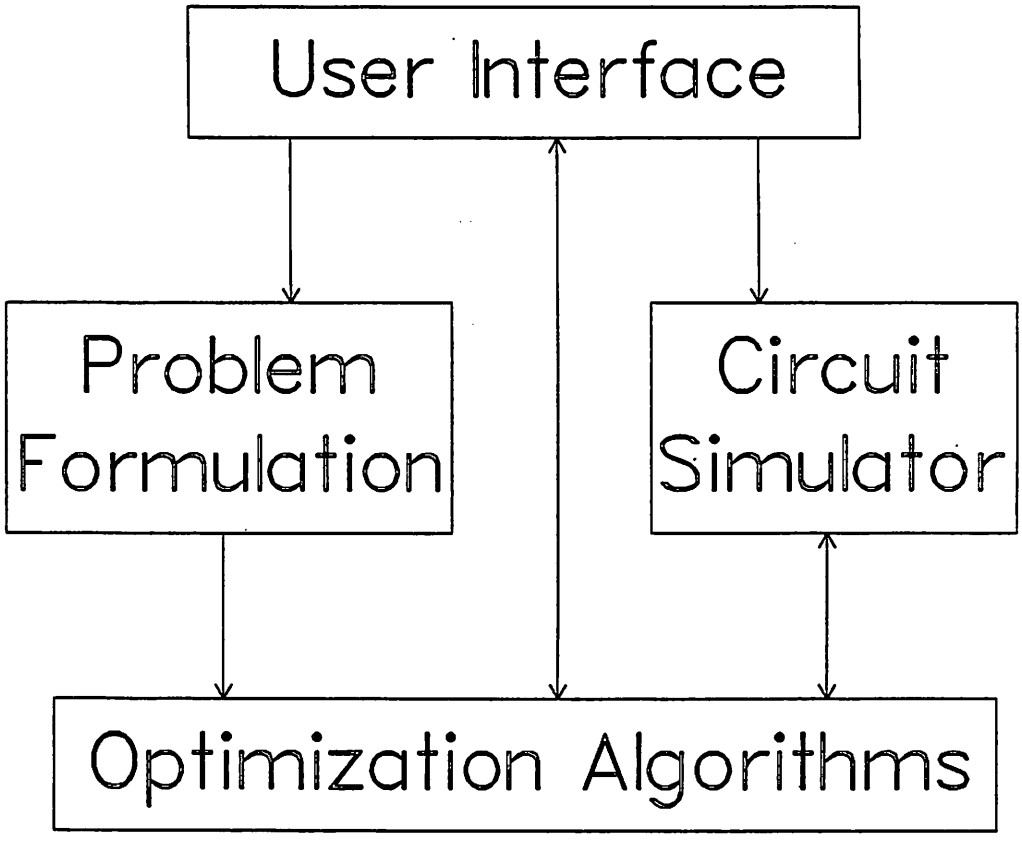


Figure 4.1: ECSTASY System Configuration

4.2.1 Sensitivity Computation

These algorithms require the computation of the gradient vectors of the objective and constraint functions with respect to all the design parameters. In general, problem specifications are arbitrary functions of the circuit variables and design parameters. However, since the circuit sensitivities which are available form circuit simulators are the derivatives of the circuit variables with respect to the design parameters, special techniques are required to compute the desired gradient vectors.

The technique used in ECSTASY is based on [76]. Using chain-rules, a *perturbation scheme* is employed to compute all the partials except the circuit sensitivities. These are computed efficiently by the so-called *direct method*, which exploits the internal Modified Nodal Analysis formulation of SPICE3 and camputes the transient sensitivities forward in time [12] [77].

At this point, it is necessary to discuss a special type of design parameters: *track parameters*. In IC design, the values of some circuit parameters may need to match (e.g., to be equal or to be in proportional to each other) to a fairly high degree. The accuracy with which these parameters can match has a substantial effect on the circuit performance. Therefore, the designer only need to declare one of the parameters in the matched set as a design parameter; the others in the set will track, in some manner, the value of the declared design parameter. For example, in designing a differential pair, it is necessary for the two transistor areas to be equal. Thus, one is declared as a design parameter and the other as a track parameter.

In IC design, the specification functions to be treated are of the form: $g(x_d, x_t, v(x_d, x_t, \omega), \omega)$, where the arguments of g are:

x_d = vector of design parameters

x_t = vector of track parameters

v = vector of circuit variables

ω = independent variable such as frequency or time

For the purposes of this research, two cases are considered. In the first case,

ω does not depend on any conditions; the desired gradient with respect to x_d is therefore given by:

$$\nabla_{x_d} g = \frac{\partial g}{\partial x_d} + \frac{\partial g}{\partial x_t}\frac{\partial x_t}{\partial x_d} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial x_d} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial x_t}\frac{\partial x_t}{\partial x_d}$$

In the second case, ω varies with the values of the design parameters and track parameters, i.e., ω is actually $\omega(x_d, x_t)$, and is forced to satisfy an equation:

$$h(x_d, x_t, v(x_d, x_t, \omega), \omega(x_d, x_t)) = 0$$

An example is the phase margin computation, in which ω is computed by solving the equation when the magnitude of the output node voltage is *0 db*. By differentiating g and the equation $h = 0$, and canceling the common terms, the following expression can be obtained:

$$\begin{aligned}\nabla_{x_d} g \;=\; & \frac{\partial g}{\partial x_d} + \frac{\partial g}{\partial x_t}\frac{\partial x_t}{\partial x_d} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial x_d} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial x_t}\frac{\partial x_t}{\partial x_d} \\ & - ratio\left\{\frac{\partial h}{\partial x_d} + \frac{\partial h}{\partial x_t}\frac{\partial x_t}{\partial x_d} + \frac{\partial h}{\partial v}\frac{\partial v}{\partial x_d} + \frac{\partial h}{\partial v}\frac{\partial v}{\partial x_t}\frac{\partial x_t}{\partial x_d}\right\}\end{aligned}$$

where *ratio* [1] is equal to:

$$\left(\frac{\partial g}{\partial \omega} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial \omega}\right) / \left(\frac{\partial h}{\partial \omega} + \frac{\partial h}{\partial v}\frac{\partial v}{\partial \omega}\right)$$

As proposed, the perturbation method is used to compute all the partials except $\partial v/\partial x_d$ and $\partial v/\partial x_t$ which are available from the circuit simulator SPICE3.

Note that when transient sensitivity is computed by the direct method, the result is obtained immediately after a simulation run. For the AC analysis case, however, this method may not be efficient. Since AC analysis is performed at each frequency independently, and only the sensitivity information at the local maxima of $\phi^j(x, \omega)$ and $\varphi^j(x, \omega)$ with respect to ω are needed in the optimization algorithms, it is recommended to perform

[1] In [76], the *ratio* is computed as $\frac{\partial g/\partial \omega}{\partial h/\partial \omega}$ which is not correct.

normal simulation runs without sensitivity computation during the problem evaluation and step size computation phases of the optimization. Sensitivity is computed only after these local maxima are known. In other words, for AC analysis, after evaluating all the functional specifications, the sensitivity computation is performed at each local maximum point – a single frequency. This technique saves computation time for the AC case.

4.2.2 Controlled Random Search

Gradient-based algorithms, although effective in optimizing circuits when sensitivity information is either available, or can be approximated by finite differences, may encounter numerical difficulties. Since the built-in algorithms are hidden from the designer, even if this problem is encountered by the designer, the situation may not be easily improved. To resolve this problem, ECSTASY automatically switches to the *controlled random search algorithm* to enable solution of the problem to continue. In the meantime, in order to allow the designer to interact with the system to rescale the problem, or to perform simulation tolerance control, informative messages are displayed on the screen for designer's reference.

From circuit design point of view, the algorithm has the following desirable features:

- It effectively avoids the past "bad regions" and moves along a good *descent* direction toward the feasible region.

- It is relatively insensitive to the parameter dimensionality, compared to pure random search, since it searches in a small region and the probability of obtaining an improved design is high.

- It gives the designer "sensitivity" information, since it improves the design *locally*. In fact, since the parameter changes are shown on the screen in ECSTASY immediately after the circuit performance is improved, the user can trade off the design specifications, as well as adjust

design parameter values from this information.

- It is easy to maintain feasibility while improving the objective functions after a feasible point is obtained.

4.3 User Interface Design

An ideal interactive circuit optimization system is one which not only solves a given optimization problem within shortest possible time, but also provides a friendly environment for the circuit designer to enjoy the task of interactive optimization.

Solving an optimization problem effectively can be achieved through the use of efficient optimization algorithms, as has been described in the previous section. Providing a friendly environment requires an analysis of human factors in circuit optimization systems, a choice of design criteria, and the exploitation of modern design principles for friendly user interfaces.

4.3.1 Design Criteria

There have been quite a few quality measures for man-machine interface designs [78] [79] [80]. Most of them are more or less linked to, or a combination of the three *primary criteria* [78]:

- the *time* the user spends accomplishing a task which the system is intended to support,
- the *accuracy* with which the user can accomplish the task, and
- the *pleasure* the user derives from using the system.

These criteria are in turn influenced by a number of *secondary criteria*, such as learning time, error and fatigue susceptibility, naturalness, and so on. Although these quality measures are obtained from experiments with general

interactive systems, they can also be applied to the user interface design of interactive optimization systems.

The design of complex ICs generally involves complicated specification functions. In order to formulate an optimization problem, the designer has to describe these specification functions precisely so that the system knows how to evaluate the functions needed in the optimization. Since most designers do not have a background in optimization theory, and no system can provide a good solution if the problem is not well-posed, the first design criterion of ECSTASY pertains to its problem formulation capability. The user interface must allow the designer to describe a problem easily and precisely, without the use of technical terminology relating to optimization.

After formulating the problem, the designer begins the optimization. A salient feature of circuit optimization is that it can be a very time-consuming process. Achieving a noticeable performance improvement for a complex design often requires a considerable length of time. Since user interface mediates between the designer and the optimization functions performed by the system, a poorly designed user interface will result in user frustration and degraded productivity. Minimizing the designer's boredom, frustration, and discomfort in using the system before a performance improvement is observed is therefore an important goal of user interface. Since the designer may want to interact with the system to trade off the design specifications, they must be informed of the current optimization status, and be able to control the optimization runs at will.

Note that the capabilities and performance of the circuit simulator used in a circuit optimization system are in general a very important determinant of the overall effectiveness of the system. However, in designing an interactive optimization system, a circuit simulator is treated as a "black box". Therefore, it is assumed that the circuit simulator has reasonably good performance.

To summarize, the three design criteria for ECSTASY are:

- Design problem must be specified easily and precisely.
- Optimization status must be perceived by the designer immediately.
- The system must respond to the designer's request promptly.

Many general design guidelines have been proposed for designing a friendly user interface [81] [82]. To achieve the three design criteria, the following design principles are employed to design ECSTASY user interface:

- Know the user.
- Communicate visually.
- Speak the user's language.
- Give the user control.
- Avoid frustrating the user.
- Respond to the user's actions.

For example, to meet the first criterion and avoid the programming language problem with existing systems, ECSTASY employs a forms-based interface for problem description, in the interest of eliminating the need for a programming language. To achieve the second and third criteria, informative visual feedback is provided through the use of tiled windows, and two concurrent processes are created for efficient user interaction.

The resulting user interface provides five *static* menus for user's selection: *objectives, constraints, parameters, simulation control*, and *optimization control.* The first three are for problem descriptions; the last two are for controlling the circuit simulator and monitoring optimization runs. Each menu pane contains several *selections*, including on-line *helps* and printer commands for producing design reports. Fig. 4.2 shows the screen format design. Note that, although on-line help and the documentation capability are not directly related to the functionalities of interactive optimization, they are important to the designer's productivity.

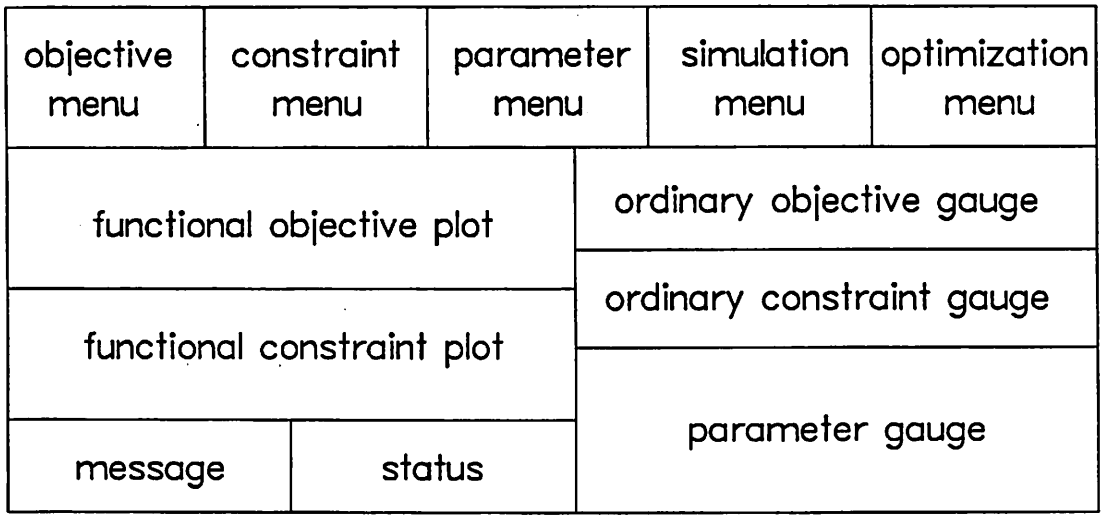


Figure 4.2: Screen Format Design

4.3.2 Forms-Based Interface

In IC design, a specification is characterized by a value (for ordinary specifications) or a curve (for functional specifications) and a sequence of measurement steps to compute the corresponding function value. ECSTASY provides a forms-based interface for the detailing of problem specifications. The various specially-formated forms offer uniform interfaces which standardize the way data is entered into the system, and allow the user to specify *computing rules* for problem specifications.

The advantages of forms-based user interface approach to developing *interactive information systems*, over approaches based on extending conventional programming languages, have been discussed in [83]. The approach has the desirable feature of uniform interfaces which standardize how data is displayed and entered into the system. While this observation may not be related directly to interactive optimization systems, the idea can be exploited.

Unlike ordinary forms in which only relevant entries are entered as input data, ECSTASY forms are designed to allow the user to set up (i.e., to *program*) test configurations for optimization purposes. According to user's mouse selection, the system displays various forms for the user to input. Each form consists of two parts: the *data* part, and the *rule* part, each consisting of two columns of *fields*, separated into a left-hand side (LHS) and a right-hand side (RHS). In the data part, *keywords* are displayed by the system in the LHS; the user only needs to fill in data in the RHS. The specification title, the command to perform simulation runs, the specification values, etc, are entered in this part. In the rule part, the user has to fill in both the LHS and the RHS to set up the measurement procedure. In this part, the LHS is a variable or a parameter name; the RHS is an expression to compute the LHS value. Since each computing rule in the form represents a procedure to compute a specification function using the interfaced simulator, the expression is of the same syntax as the simulator front-end language -

the designer's language (In user interface design, it is important to speak the same language as the users [81]). This effectively eliminates the need for a new language.

Two examples of problem formulation using the forms are illustrated. It is assumed that all the simulation input decks (prepared before optimization) contain appropriate elements for test configurations. Fig. 4.3 describes a constraint where the delay of a buffer circuit is to be less than 5 *ns*. Suppose the circuit is driven by a pulse with a 0.5 *ns* delay and a 20 *ns* duration. Then the delay requirement can be satisfied if the output waveform v(5) from 5.5 *ns* to 20 *ns* stays above 2.5 V. This requires transient analysis, and is a functional constraint.

Fig. 4.4 shows a phase margin constraint. It is an ordinary constraint, requiring AC analysis to be performed. To ensure that the circuit works properly, initial values for RI and RF have to be provided. The 0-db frequency of the output waveform v(3) is computed by passing the *magnitude* waveform, vm(3), to a built-in routine *zdbfreq* (ECSTASY provides some built-in routines that can be invoked by the user). The computed frequency is used to find the corresponding phase from the *phase* waveform vp(3). Since the phase is in *radians*, some computation is necessary to obtain the desired value. After the measurement, RI and RF are reset to their default values. Note that all the computing rules are in SPICE3 front-end language.

ECSTASY provides a *what-you-see-is-what-you-get* type editor for the user to fill in or to modify a form. When a form is displayed, the system directs the cursor to the beginning of the first field. Then whatever the user types is displayed in the form; the content displayed in the form is exactly what is stored internally in the system buffer. The user can use CTRL-H, CTRL-L, CTRL-K, CTRL-J, and carriage return to move the cursor without changing the contents. In case the problem specification is so long that the current form cannot accommodate it, the system will *scroll-up* the form (when the cursor reaches the bottom of the form) and display more blank

Functional Constraint		1
label		Delay Constraint
analysis		tran 0.4ns 20ns 0 0.4ns
>= or <=		>=
good value		2.5 V
bad value		2.5 V
sweep	variable	TIME
	from	5.5ns
	to	20ns
	scale	lin
	increment	0.4ns
computing rules		delay
delay		v(5)

Figure 4.3: Delay Constraint

Ordinary Constraint	2
label	Phase Margin
analysis	ac dec 10 1MEG 20MEG
>= or <=	>=
good value	−135
bad value	−160
computing rules	phase
RI	10K
RF	2MEG
FREQ	zdbfreq (vm(3), 1MEG, 20MEG)
phase	57.29578 * vp(3) − 180
RI	10MEG
RF	1

Figure 4.4: Phase Margin Constraint

fields to be filled in by the user.

4.3.3 Simulation Control

In circuit optimization, there are special needs, such as for track parameters, which need special treatment. From the algorithmic point of view, a design parameter is simply an independent variable in the parameter space. However, to a designer, some of the circuit parameters are closely related. While track parameters need to track the changes of design parameters, the device model parameters AD, AS, PD, and PS in MOS models also play similar roles. These model parameters are often set in proportional to the values of the size of a MOS transistor to account for second-order parasitic effects. Furthermore, some circuit elements, called *settable* parameters, are used only for setting up test configurations for optimization purposes; these parameter values are not tunable by optimization algorithms.

Since these requirements are transparent to optimization algorithms but play a very important role in circuit design, a special simulation control interface must be provided to perform this special task. In ECSTASY, the designer can specify three types of parameters: design parameters, track parameters, and settable parameters. Simulator option control such as temperature and tolerance controls are also provided through this simulation control interface. Again, this design is forms-based.

4.3.4 User Interaction

After the specifications are described, the user may request optimization runs and perceive the performance improvement through visual feedback displayed on the screen. ECSTASY converts the problem into a standard formulation, carries out the optimization, and displays circuit performances automatically during optimization runs.

To allow efficient user interaction, ECSTASY creates two concurrent processes: one to talk to the user in the foreground, and the other to

run simulations in the background. Since the foreground process does not consume CPU power while waiting for the user's mouse event, the system can respond to user's requests immediately without degrading the system's performance. Thus, any time during optimization runs, the user can interact with the system by clicking the mouse to halt the optimization session, trade off problem specifications, and resume the job.

To provide visual feedback in both analog and digital ways, EC-STASY displays *gauges* and functional plots in tiled windows. Each gauge represents an ordinary specification, and is divided into a*bad* region (in red), an *ok* region (in yellow), and a *good* region (in green); each functional plot represents a functional specification, with two lines to indicate if the performance is good (green line) or bad (red line). The ultimate goal is to push every *dial* in the gauge into the green region, and every functional plot under the green line. To compare two consecutive optimization runs, the current performance is shown in *black*, while the previous one is in *plum*. Fig. 4.5 shows an example of a performance gauge in which the offset voltage is improved from 10 mV to 8.06 mV.

4.4 Circuit Simulation Interface

Circuit simulation interface bridges the optimization algorithms and the circuit simulator. Since each *computing rule* in the form represents a step to measure a specification function using the interfaced simulator, the expression of that rule is assumed to be of similar syntax to the simulator front-end language (it is desirable to speak the same *language* as the simulator user). Before optimization, a built-in parser converts each expression (in *infix* notation) into a stack of identifiers, operators, function calls, and parentheses (in postfix notation) [84] [85]; the piles of stacks are then maintained by each form. Thus, during optimization, whenever the optimizer needs to evaluate a function value, the stacks are popped out one by one to be evaluated or to request simulations.

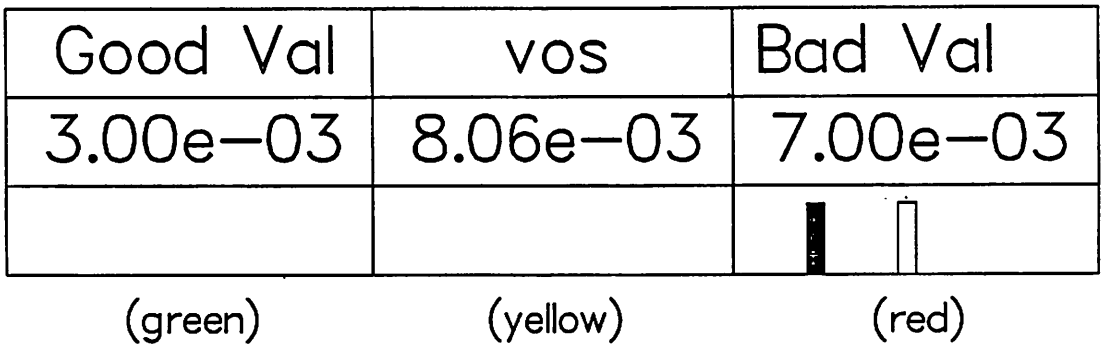


Figure 4.5: Performance Gauge

To parse an expression, the parser operates on two stacks S1 and S2. S1 is a push-down stack which stores the output of the parser and maintains the final postfix notation; S2 is a temporary scratchpad stack. Currently, the allowed syntactic units are: identifier, operators (+, -, *, /, %, and exponent ^), keyword functions (db, zdbfreq, findzero, etc), separator (,), parentheses, and end of expression "END". Table 4.1 shows the *parsing action table* where the top row represents the input symbol, and the leftmost column represents the top of S2.

As an example, if the input expression is db(vm(3)-2), then *db* is a function, *vm(3)* and 2 are identifiers, and "-" is an operator. According to the parsing action table, S1 will contain "db", "-", "2", "vm(3)", with "db" at the top, and "vm(3)" at the bottom. When the optimizer needs to evaluate the expression whose postfix notation is in S1, the interpreter treats S1 as a queue by reading the syntactic units from bottom to top, one by one, using S2 for temporary storage. At first, when "vm(3)" is read, the interpreter calls for the simulator to run an AC analysis, accesses the *waveform* of the magnitude of node voltage at node 3, and pushes it onto S2. Then "2" is read and is pushed onto S2 since no simulation call is needed. Next, when "-" is read, the interpreter pops out the top two elements on S2, applies the operator to get the waveform "vm(3)-2", and pushes this result onto S2. Finally, when "db" is read, the interpreter pops out the top atom of S2 as the argument to the function call "db". The returned waveform thus represents "db(vm(3)-2)". Note that, in general, the operands are waveforms, so interpolation techniques must be used to extract the desired responses in case the sampling points are different on two operands, especially for transient analyses.

Since each rule has a left-hand side (a variable) and a right-hand side (an expression), after the expression is evaluated, the resulting waveform is assigned to the left-hand side variable, which can then be used as a waveform in the succeeding rules. Of course, the parser knows that a design parameter cannot be assigned a waveform; type checking is performed

	id	+ -	* / %	^	(	)	func	,	END
NULL	s1	s2	s2	s2	s2	s2	s2	ERR	ERR
+ -	s1	u1	s2	s2	s2	uc	s2	d	u2
* / %	s1	u1	u1	s2	s2	uc	s2	d	u2
^	s1	u1	u1	u1	s2	uc	s2	d	u2
(	s1	s2	s2	s2	s2	uc	s2	d	u2
func	ERR	ERR	ERR	ERR	s2	ERR	ERR	ERR	ERR

s1: stack input onto S1

s2: stack input onto S2

u1: unstack S2 $\Rightarrow$ S1;
unstack all the top of S2 which has higher priority than input;
stack input onto S2

u2: unstack S2 $\Rightarrow$ S1 until S2 empty

uc: unstack S2 $\Rightarrow$ S1 until "(" is encountered; discard "(";
if top of S2 is a function, then unstack S2 $\Rightarrow$ S1

d: discard input

ERR: invalid input, error occurred

Table 4.1: Parsing Action Table

during parsing. (It should be noted that changing a design parameter in the computing rule is different from doing so in the optimization routine. In the computing rules, when a design parameter is changed, it is reflected directly into the simulator, no other side effect will happen, e.g., need to change a corresponding track parameter; while in the optimization routine, whenever a design parameter is changed, the corresponding track parameter must be changed. This is very important for applications in which offset effects such as voltage offset are to be measured.)

One of the original goals in forms-based user interface is to keep the syntax of the "measurement" language as simple as possible, since it is not desirable for the user to have to learn a complex new language. Hence only SPICE3-like simulation language (particularized to the specific simulator) is allowed. While such syntax is rather limited and may not be powerful for general programming use (for sophisticated users), it is considered best that the library of keyword functions be augmented as needs arise. For example, to cope with the measurement of quantities which satisfy an equation, the *findzero* function call is used to find the zero of a waveform. As an example, to measure the switching time of node 3 at a constant level K, *findzero(v(3)-K)* can be called.

4.5 Design Examples

Interfacing with SPICE3, ECSTASY has been run on a DEC VAXstation II/GPX under Ultrix V2.0. A C language interface to the X window system [86] is used as the graphics interface. The use of the network-transparent window system X allows ECSTASY to utilize displays on other machines. Currently, for efficiency purposes, ECSTASY interfaces to the circuit simulator directly. A future direction is to use *remote procedure call* (RPC) protocols, so that circuit simulations can be performed in a more powerful distributed computing environment.

To justify the proposed optimization algorithms, ECSTASY runs

remotely on a DEC VAX 8650, with the outputs displayed on a VAXstation II/GPX. Three design problems are presented: a CMOS driver circuit, a switched-capacitor (SC) filter, and a bipolar operational amplifier (Op-Amp). All the examples have as their goal the achieving of a set of specifications, i.e., inequality constraints. To compare the efficiencies of the algorithms, the (internal) scaled violations, i.e., the values of the maximum of all of the constraint functions, are shown. These become non-positive when all the specifications are satisfied. Note that, in all cases, the controlled random search algorithm uses an initial search space generated by allowing each parameter dimension to change 10%.

4.5.1 CMOS I/O Driver Circuit

Fig. 4.6 shows the schematic of the CMOS I/O driver circuit. The circuit is from a real chip design where a high-speed driver is required at the I/O bonding pads. Due to the inherent inductance associated with each power and ground lines, the overshoot/undershoot at the output may invalidate the logic when the capacitive load is large. The objective is to adjust the transistor sizes such that the delay is within 5 *ns* when driving a 140 pF load. (The original design was 11.2 *ns* delay.) The circuit has four design parameters and four track parameters. The results for the CMOS driver circuit are shown in Fig. 4.7. Fig. 4.8 compares the circuit performance before and after the optimization.

It is clear that the sensitivity computation gives the best result. Finite difference approximation, due to the inherent inaccuracies with the scheme, dies at iteration no. 9, marked by x (the algorithm complains that a descent direction cannot be computed from the gradient information) and random search is used to continue the optimization run. Note that, from Fig. 4.7, the overhead in sensitivity computation makes the average time for a function evaluation approximately 15 seconds, compared to the 4 seconds when normal simulations are performed without computing sensitivities.

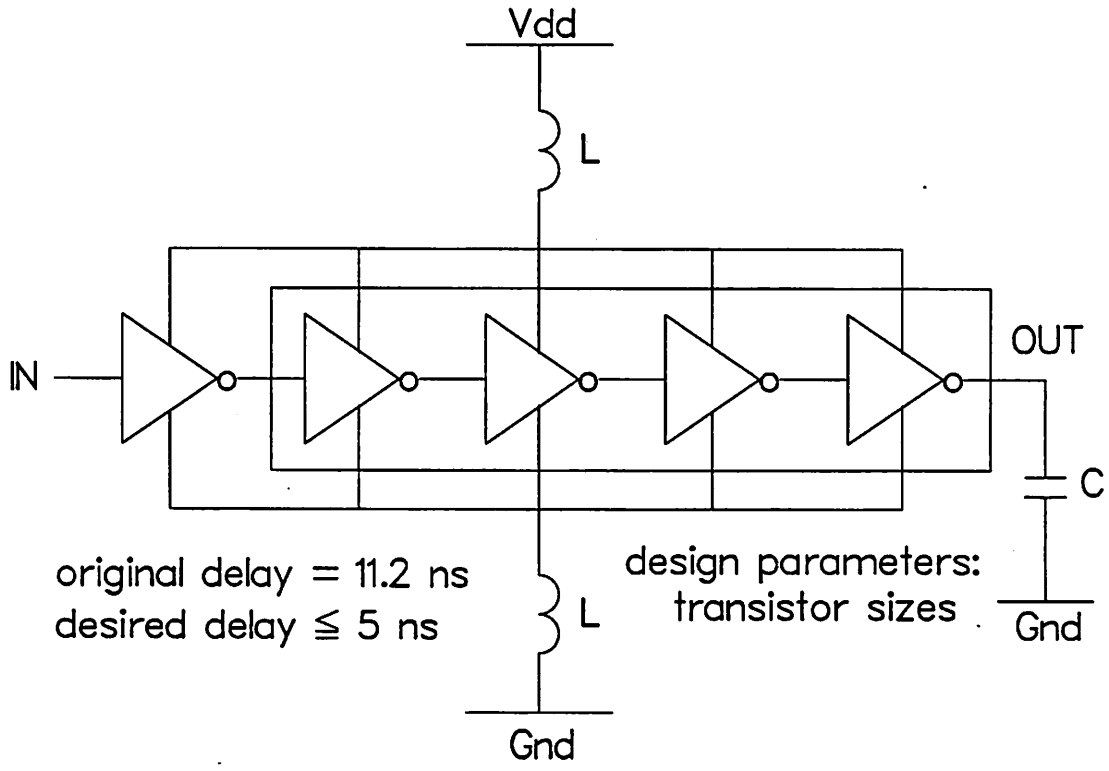


Figure 4.6: CMOS I/O Driver Circuit

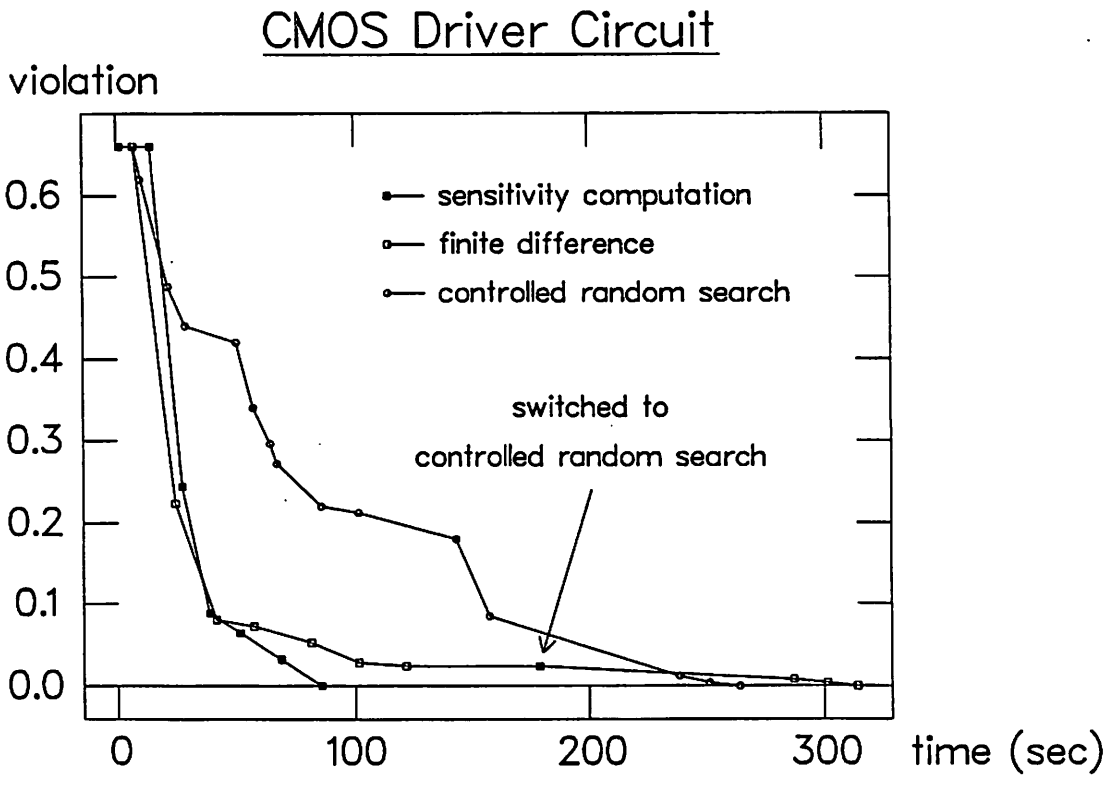


Figure 4.7: Computational Statistics for CMOS I/O Driver Example

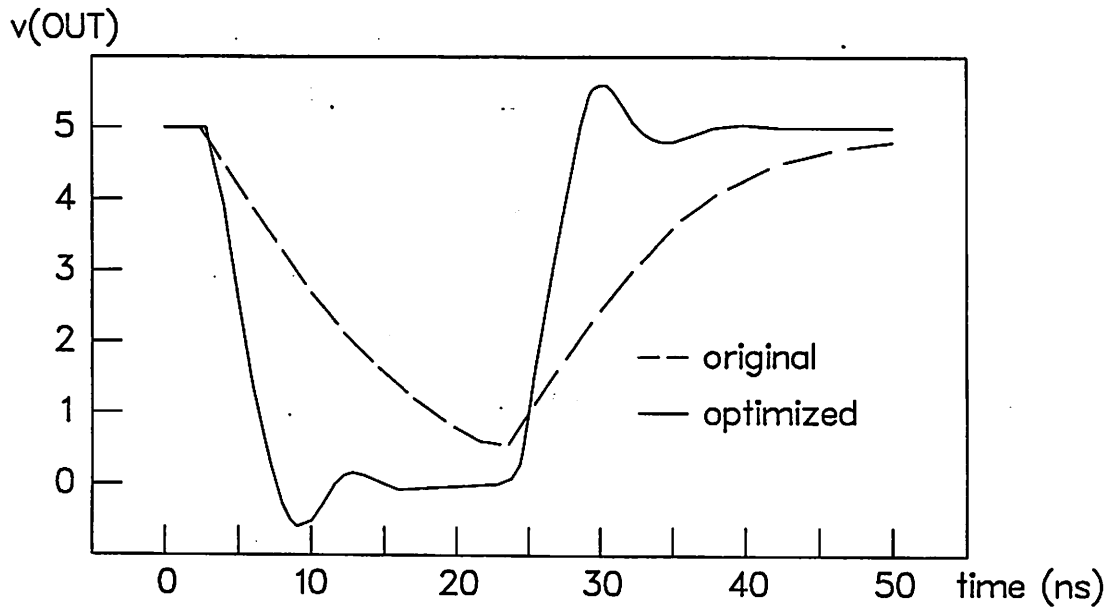


Figure 4.8: Optimization of CMOS I/O Driver Circuit

To justify the robustness of the proposed random search algorithm and the arguments on scaling, the same initial point was used to restart the optimization run. However, the upper bounds of three of the design parameters were increased significantly. Fig. 4.9 shows the execution results for sensitivity computation and the controlled random search. (Some intermediate results for random search were omitted.) Due to poor scaling, the gradient-based algorithm, even with accurate sensitivity computation, has difficulty in obtaining a feasible design after several iterations and was terminated, since it took too much time. In contrast, random search worked just as well, although spending more time due to the *augmented* search space.

4.5.2 Switched-Capacitor Filter

The second example is an SC filter design. The circuit is a ninth-order SC low-pass filter, clocked at 64 KHz, serving as an anti-aliasing filter for a digital signal processor. Fig. 4.10 shows the schematic. The specification requires a ± 0.125 dB passband ripple with cutoff frequency at 6 KHz. The attenuation requirements for the stop-bands are: -26 dB at 7.5 KHz, -45 dB at 9 KHz, and -80 dB at 18 KHz. The design was based on a common approximation method for ladder SC filters. The approximation is valid for clock frequencies many times (> 100) larger than the pass-band. For this filter, the ratio is only 10:1 (due to other considerations), and hence the deviation from the desired frequency response is large. For this reason, adjustment of the capacitor ratios is required to satisfy the specifications.

To simulate the circuit in the frequency domain (other than the z-domain) using SPICE3, an equivalent transmission-line network [87] modeling the delay caused by the switch is used as the input to SPICE3. The original ripple is 0.44 dB. Five design parameters were specified by the designer. At first, the bounds on the design parameters were arbitrarily specified. After one iteration, guided by the parameter changes displayed on the

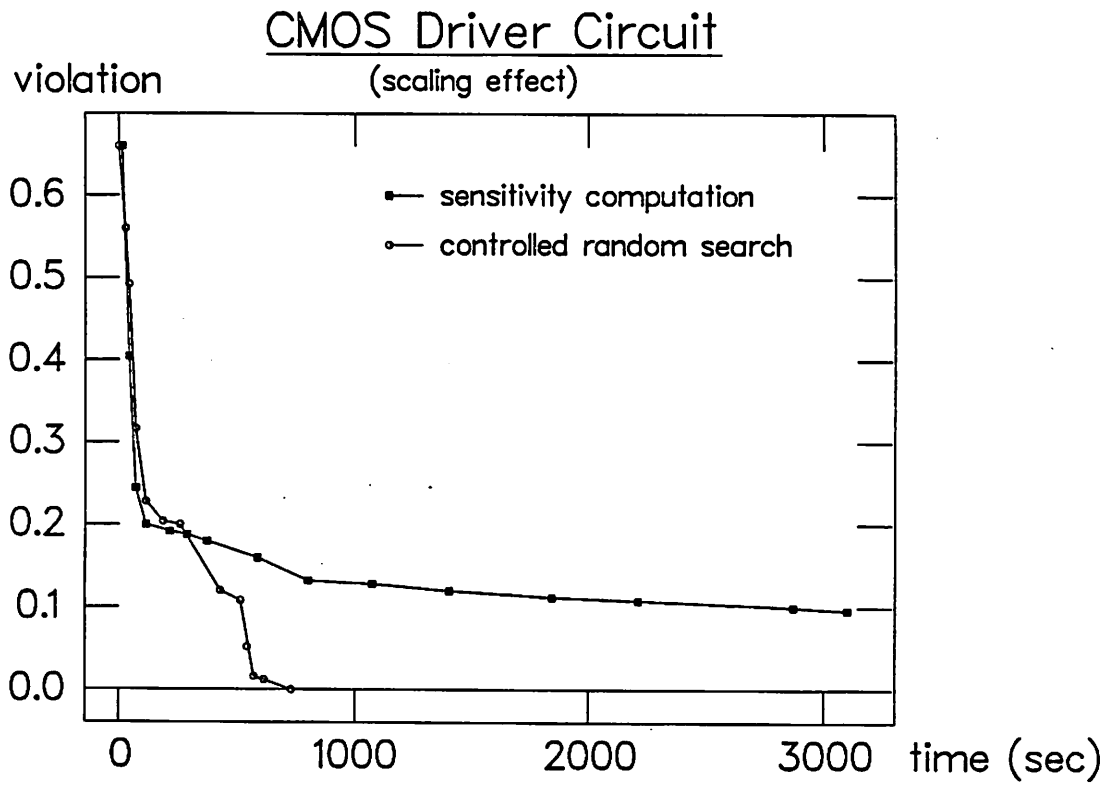


Figure 4.9: Effect of Scaling on Optimization Algorithms

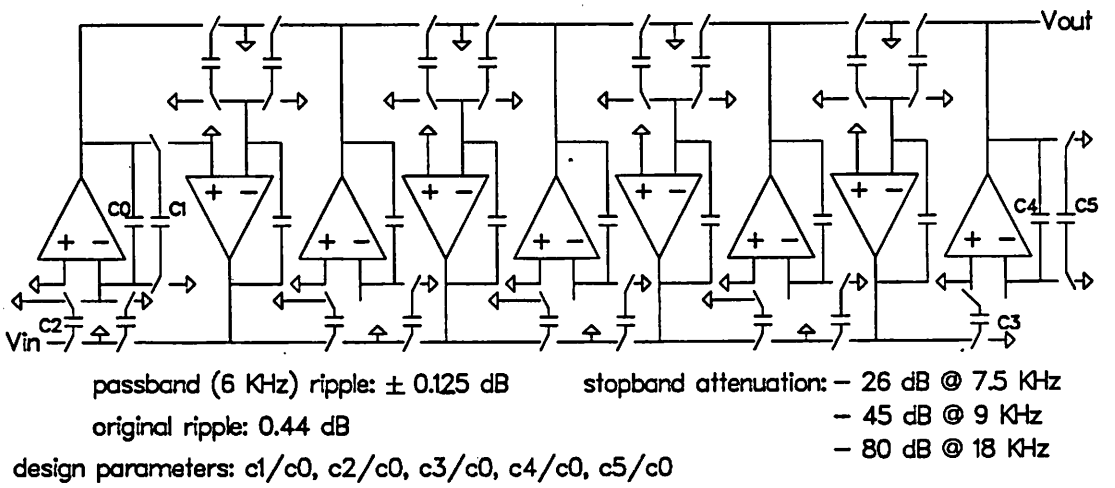


Figure 4.10: SC Filter Circuit

screen, the designer felt that all the lower bounds were too high and decreasing all the parameter values might help. All the parameter values were reset (rather arbitrarily) to a much lower value, and the optimization process was restarted. Fig. 4.11 shows the optimization results after the restart. Fig. 4.12 compares the circuit performance before and after the optimization.

Again, the gradient-based algorithm based on sensitivity computation gives the best performance. Finite difference approximation, although not as efficient as exact sensitivity (since it needs more simulation calls), performs well. This is because it is rather reliable for AC analysis in which, after the DC operating point is obtained, only backward and forward substitutions are required to compute the solution, and hence less errors are created than for transient analysis. It should be noted that, by the proposed sensitivity computation scheme, some simulations are performed at a single frequency and others at all the sampling frequencies.

From this example, it might seem that for AC and DC cases, a finite difference interval may be fixed that would work well for most circuits. This is not true, as has been explained in Chapter 3. Consider the bipolar current source in Fig. 4.13. The circuit is to provide a low temperature coefficient (TC) current source from a band-gap voltage reference. The design goals are:

$I_{o1}, I_{o2} = 400\ \mu\text{A} \pm 0.1\%$

TC of $I_{o1}, I_{o2} < 50$ ppm/°C over -55°C to $+125$°C

Maximum current drawn from Vcc is 3 mA

The same finite difference interval as in the SC filter example is used. Optimization results show that when finite difference approximation is used, the gradient-based algorithm dies after 7 iterations (marked by x). Again, controlled random search are used to take over the job. Fig. 4.14 shows the numerical results.

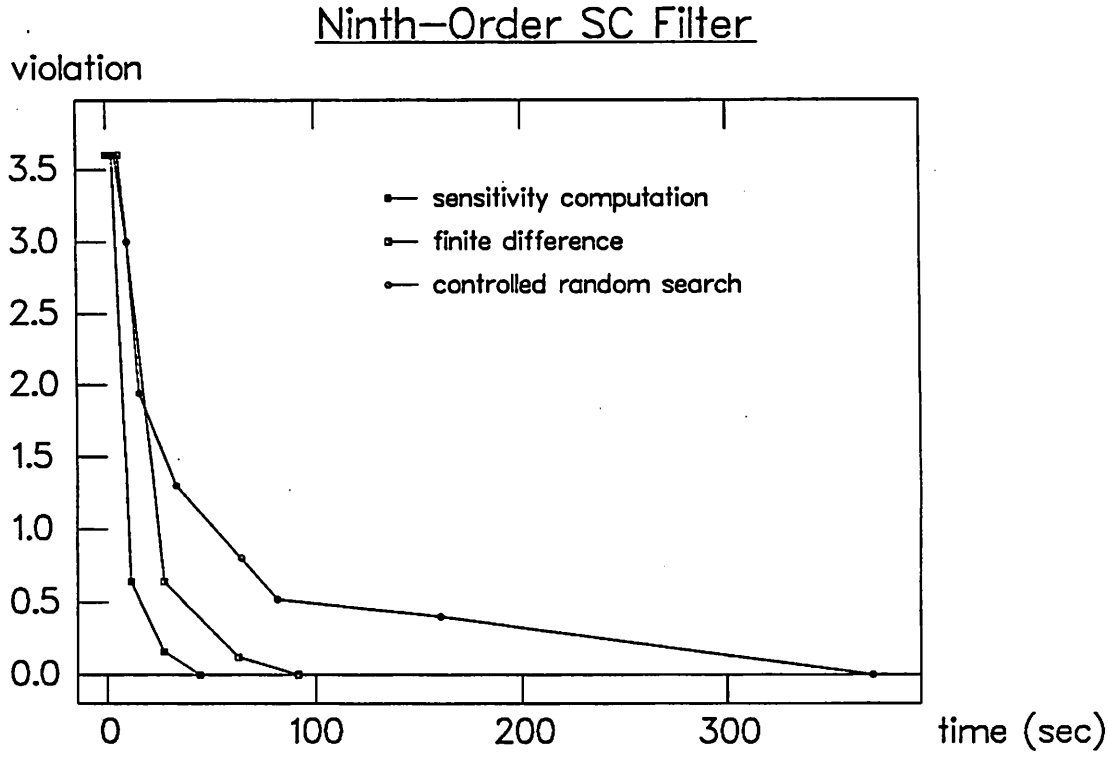


Figure 4.11: Computational Statistics for SC Filter Circuit

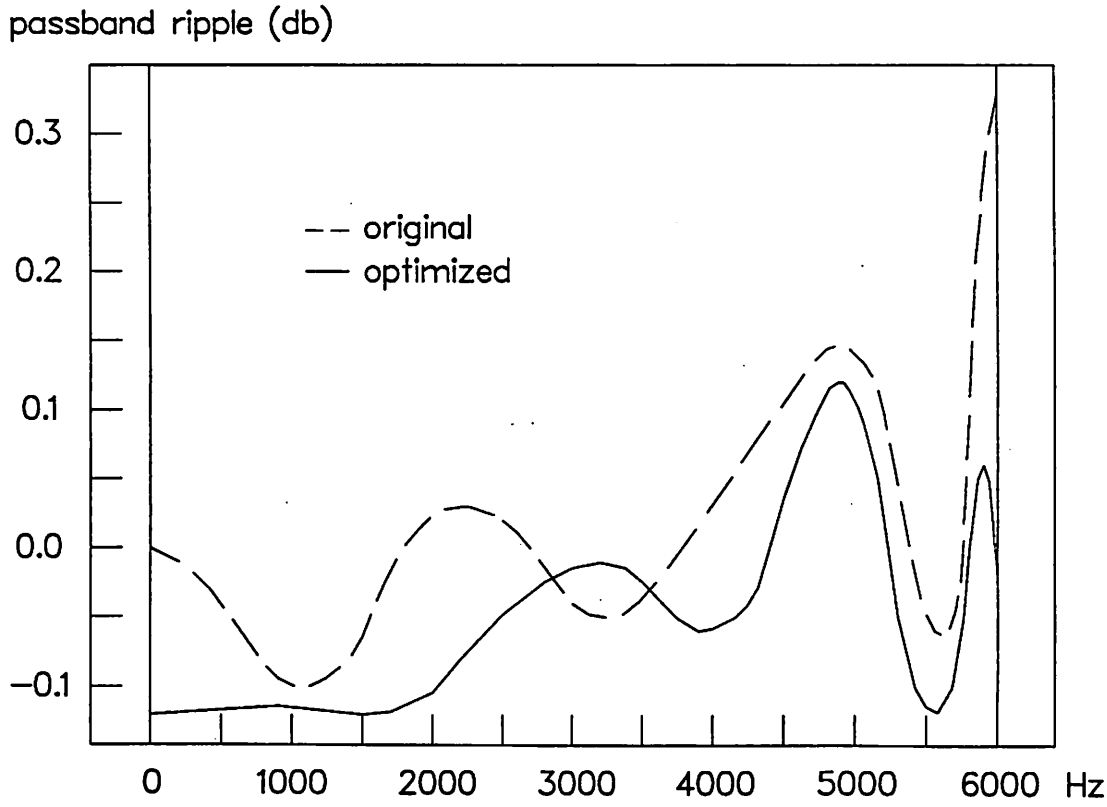


Figure 4.12: Optimization of SC Filter Circuit

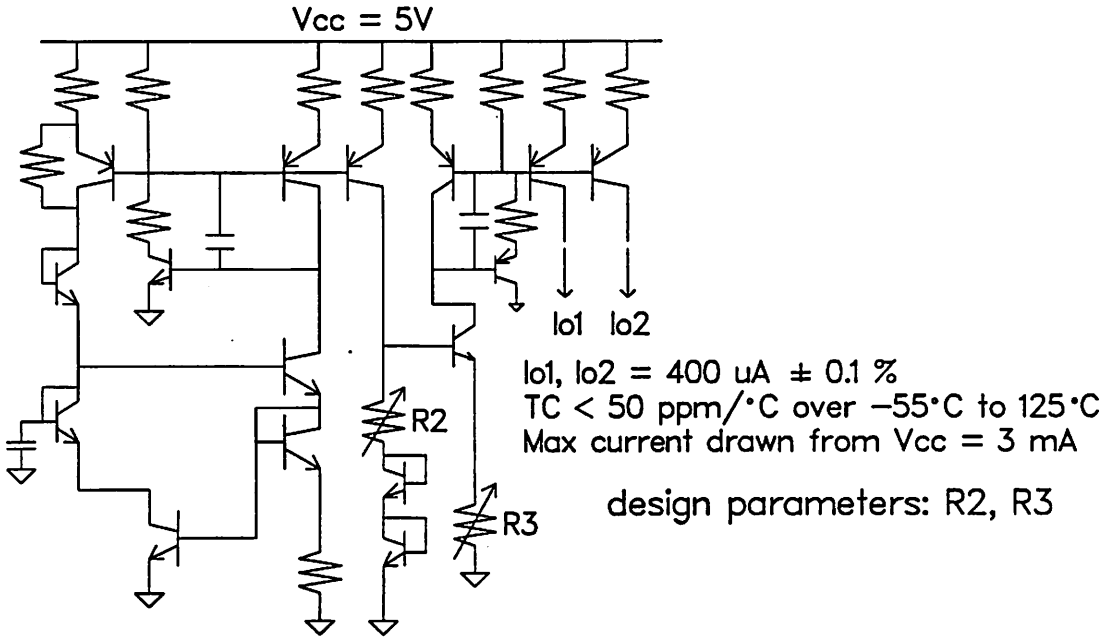


Figure 4.13: Bipolar Current Source

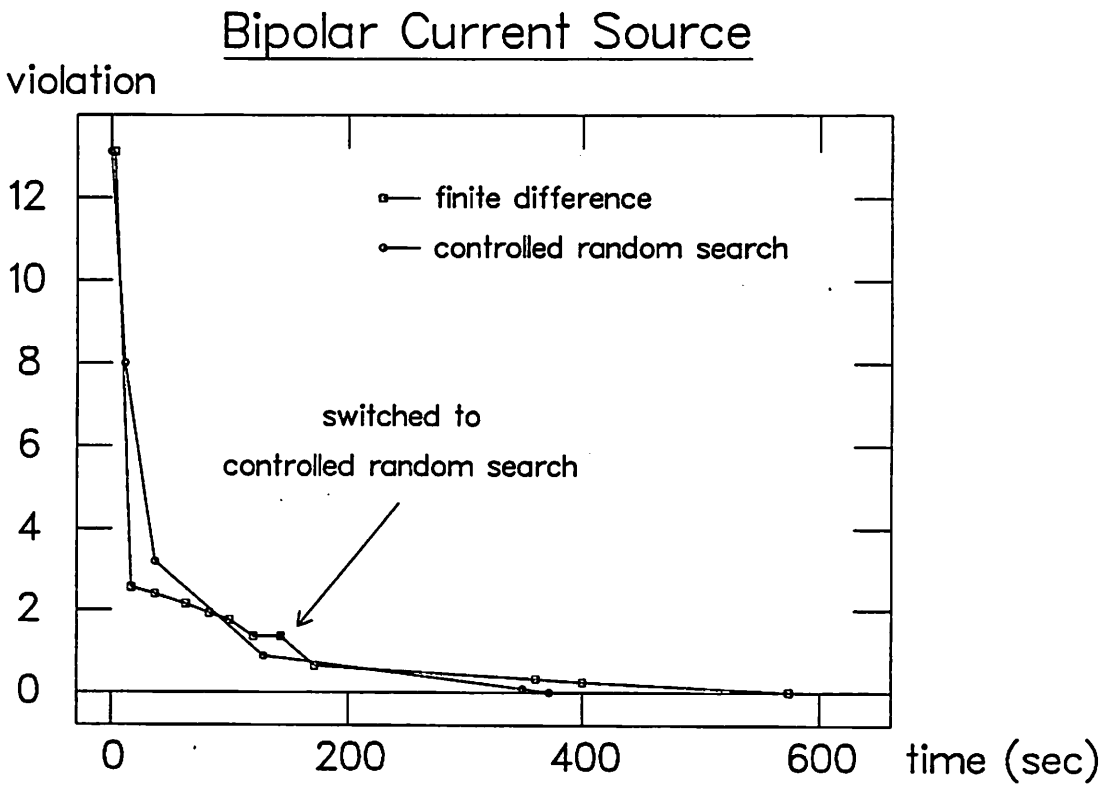


Figure 4.14: Computational Statistics for Current Source Example

4.5.3 Bipolar Operational Amplifier

Fig. 4.15 shows the bipolar Op-Amp circuit. The optimization results are shown in Fig. 4.16. The example is taken from [19], and has five design parameters and three track parameters. After trading off some constraints, the interaction focuses on improving the settling time of the circuit from $2.1\mu s$ to $1.5\mu s$, while keeping the gain-bandwidth product within the desired specification.

As in the first example, due to the errors incurred in the approximation scheme with transient analyses, convergence problem occurs in the finite difference approximation (marked by x). Controlled random search is used to continue the searching. It is interesting to note that, although sensitivity computation achieves a feasible design in 10 iterations and requires only 18 function evaluations, the computational cost is higher than that of the other two algorithms. The average time for a function evaluation with sensitivity computation is approximately 40 seconds, compared to the 10 seconds when normal simulations are performed without sensitivity computations. In this example, controlled random search is the most efficient.

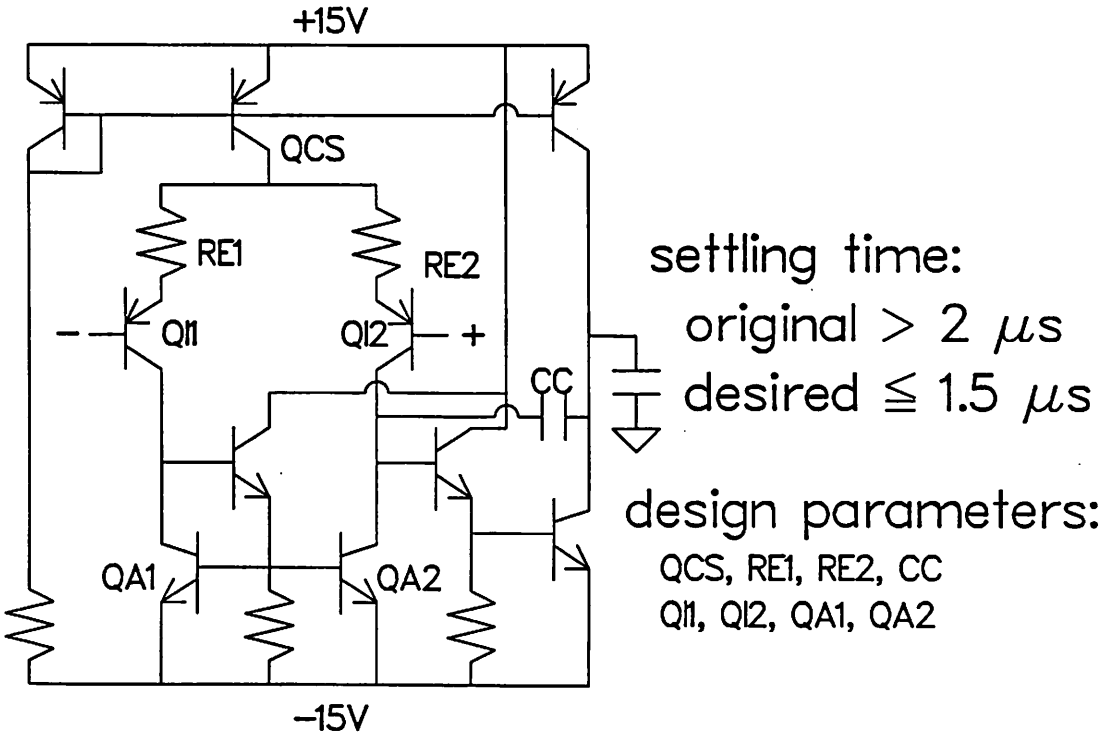


Figure 4.15: Bipolar Operational Amplifier Circuit

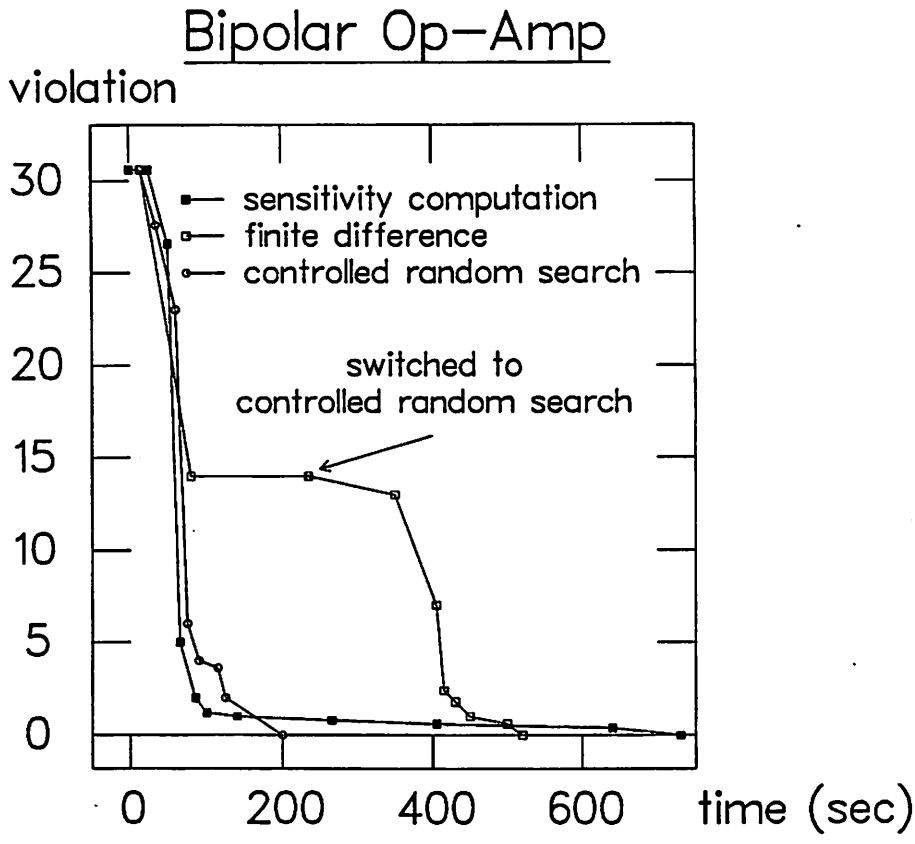


Figure 4.16: Computational Statistics for Bipolar Op-Amp Example

Chapter 5

A Transistor Sizer for Digital CMOS Circuits

5.1 Introduction

Both heuristic and nonlinear programming techniques have been shown to be effective in transistor sizing for digital CMOS circuits. Heuristic method offers a quick way of obtaining feasible design, but cannot guarantee the quality. Nonlinear programming can optimize arbitrary objective functions with guaranteed convergence properties, but lacks a systematic way of choosing relevant parameters for good inital design.

It is possible to combine the advantages of the two approaches. A heuristic algorithm is used initially to do a quick sizing of the entire circuit; then the problem is converted into a nonlinear programming problem. The problem is solved in a space of reduced dimensionality, in which, to cope with the nondifferentiability of the circuit delays, the concept of generalized gradients is used to compute the delay sensitivities.

In this chapter, a transistor sizer based on this combined approach is described. The main body of the program consists of a timing analyzer and an optimizer. It reads in a transistor netlist and the delay constraints at the I/O ports of the circuit, as well as a set of process technology data for

evaluating circuit delays. To optimize the circuit performance, the optimizer adjusts the transistor sizes through interfacing with the timing analyzer. The output of the program is an optimally-sized circuit. Fig. 5.1 shows the block diagram of the transistor sizer.

Section 5.2 describes the algorithm for the combined approach. A scheme to speed up the heuristic is proposed and justified by experiments. Sections 5.3 - 5.5 detail the ingredients of the computation. Section 5.3 describes the modeling of digital static combinational CMOS circuits. Section 5.4 discusses the optimization technique and explains the use of generalized gradients to cope with the nondifferentiability of circuit delays. Section 5.5 details the delay sensitivity computations. Experimental results are shown in Section 5.6.

To compare the results with TILOS, the transistor sizing problem is restricted to that of minimizing the total active area under delay constraints; all the test circuits are restricted to static combinational CMOS circuits.

5.2 A Two-Stage Combined Approach

5.2.1 The Approach

As has been explained in Chapter 3, the TILOS algorithm is a good heuristic for meeting the delay constraints in transistor sizing problem. In fact, the TILOS authors have used the algorithm to size many static CMOS circuits with up to 26,000 transistors. The circuits obtained are typically twice as fast as the equivalent circuits implemented with standard cells. It is observed that the algorithm works very fast and, oftentimes, only part of the transistors need to be sized. Interestingly, the algorithm is very similar to the steps a human designer takes to speed up a circuit while minimizing the power consumption.

It is shown in [32] that, under a simple distributed RC model, the transistor sizing problem for static CMOS is convex. Unfortunately, despite

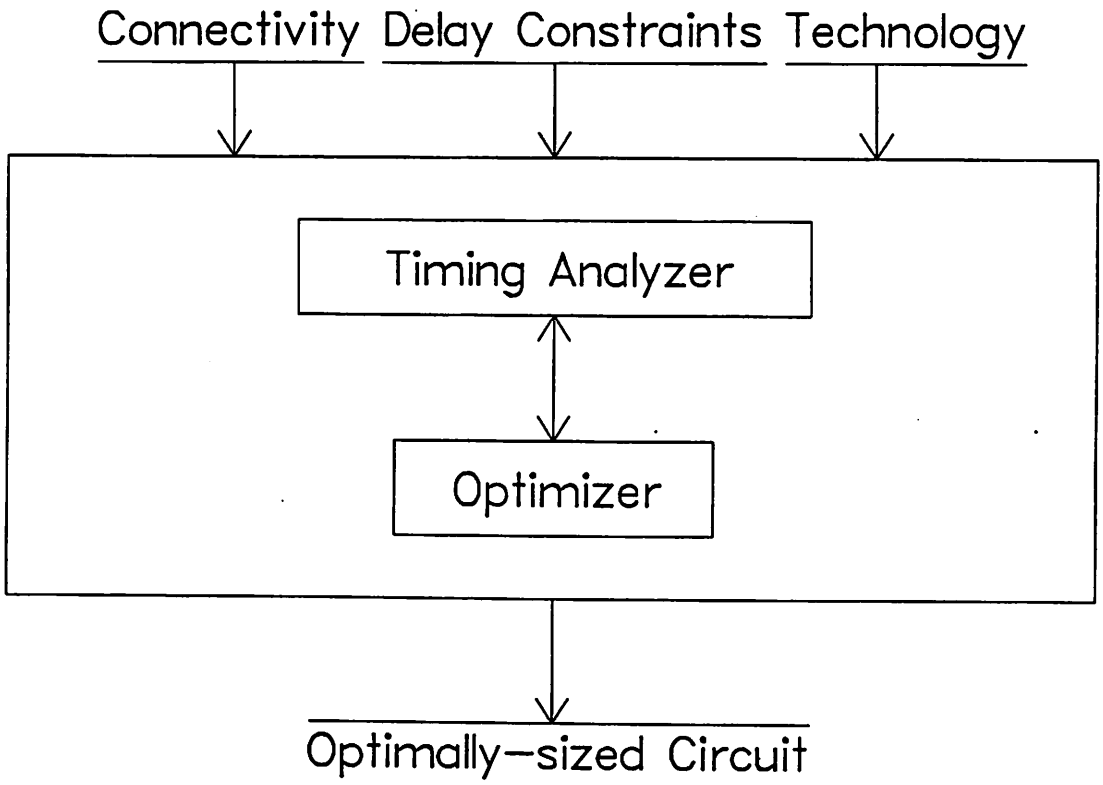


Figure 5.1: Block Diagram of the Transistor Sizer

the fact that in a convex function a local minimum is a global minimum, TILOS fails to guarantee optimality for the following reasons:

- It lacks a global view of the whole circuit – TILOS picks *a* greatest-sensitivity transistor on *a* worst-delay path to size up.
- It increases the sizes of the transistors only, while in certain cases it may be effective to decrease the size of some of the devices.
- Signal delay in a circuit is calculated as the maximum function of all possible delays within the circuit. This may cause discontinuities in sensitivity calculations, resulting in over-sized transistors.

Nonlinear programming can be employed to remedy these shortcomings since multiple constraints can be considered simultaneously and the transistor sizes can be tuned for best performance, if delay sensitivity can be computed accurately. In fact, this method can start from arbitrary transistor sizes, if robust algorithms such as the feasible directions method are used. If in addition, the objective and constraints are convex functions, these algorithms are guaranteed to converge to the global minimum.

This consideration leads to a two-stage approach to combine the speed of heuristic and the power of nonlinear programming.

5.2.2 The Algorithm

It is desired to minimize the sum of transistor sizes under delay constraints, with transistor sizes as design parameters. Since, in practice, the size of a transistor should be larger than a minimum imposed by process technology, it is necessary to satisfy the same number of minimum-sized constraints as the number of transistors in the circuit. Fortunately, although the number of minimum-sized constraints may be large for large circuits, these constraints are rarely violated once the transistors are sized, since in general transistor sizes are increased in the procedure. Often, the constraints

we strive to meet are the delay constraints. Usually, the number of delay constraints (the specifications) is not proportional to the circuit size; it is $2N$ for a circuit with N output ports (there are two delay constraints, rising and falling, at each output port). Thus, the total number of design constraints is equal to $2N$ plus the number of transistors. Note that for a delay constraint specification at an output port, there may be several delay paths that violate the constraint. In general, the number of delay constraints is much less than the minimum-size constraints.

The combined approach starts with minimum transistor sizes. At first, the TILOS heuristic is used to do an initial sizing for the entire circuit. After the heuristic finds a solution which satisfies the constraints, the sized-up transistors are used as design parameters. The problem is then converted into a standard nonlinear optimization formulation:

minimize $\sum X_i$

such that: $G(X) \leq T$ and $X \geq K$

where

X = vector of the selected transistor sizes from TILOS

$G(X)$ = the signal delays on the output ports

T = the delay constraints on output ports

K = the minimum-size constraints on X

which takes into account the global path interactions automatically. The optimizer solves this nonlinear programming problem by adjusting the design parameters such that the sum of these parameters is minimized and all the constraints are satisfied.

Note that the heuristic approach is not guaranteed to find a solution which satisfies any optimality condition. However, it gives an excellent starting point for a more rigorous approach based on nonlinear programming techniques. The nonlinear optimization algorithms that can be used to solve the sizing problem have complexity that depends superlinearly on the number of design parameters. Hence it is convenient to solve the original optimization problem by solving a sequence of problems that have a smaller

number of design parameters. Note that, by confining the problem to a subset of the design parameters, it is possible to find a solution which has a larger value for the objective function than the solution obtained using *all* the design parameters. This argument can be justified in Section 5.6, where the solutions obtained from the reduced parameter space and the entire space are compared.

5.2.3 A Variable Bump-Size Scheme

In the original TILOS algorithm, a constant bump-size factor (> 1) is used throughout the optimization to increase the transistor sizes. While the algorithm is fast enough for practical use, it can be further accelerated by using a scheme with *variable* bump-size factor (in contrast to using a *fixed* bump-size factor as in the original scheme). The idea is to adjust *adaptively* the bump-size factor in the algorithm such that when the delay violation is large, the bump-size factor is large (e.g., 1.5), while when the delay violation is small, the bump-size factor becomes small (e.g., 1.1); the value of the factor varies linearly in between.

Tables 5.1 and 5.2 show the effect of using variable bump-size factor on sizing some benchmark circuits from [88]. Throughout the experiment, all the circuits are initially of minimum transistor sizes. The fixed bump-size factor is 1.1; the variable scheme adjusts the factor from 1.5 to 1.09. It is clear that, over all the test cases with different delay constraints, the CPU time savings are more than 20% with same area quality.

5.3 Modeling Circuit Delays

The accuracy of a timing analyzer depends on the delay model, but the algorithm should be independent of the model used. In transistor sizing, it is important to compute circuit delays to within required accuracy efficiently. If the model is not accurate enough to estimate the circuit perfor-

Examples			Fixed Bump-Size		Variable Bump-Size		
Name	FETs	Delay (nsec)	Area (μm)	Time (sec)	Area (μm)	Time (sec)	Saving (%)
5xp1	318	10.0	1562	144.8	1560	90.6	37.4
9sym	180	10.0	900	37.6	892	21.1	43.9
bw	518	10.0	2319	513.6	2309	260.4	49.3
duke2	1130	19.85	4530	3539.5	4528	2763.1	21.9
rd53	124	5.0	1364	31.3	1367	22.5	28.1
rd73	278	7.0	2960	185.3	2955	127.3	31.3
sao1	350	10.0	3333	367.3	3318	249.6	32.0
sao2	416	10.0	2536	291.6	2549	202.4	30.6
vg2	266	10.0	1920	158.7	1920	103.4	34.8

Table 5.1: Fixed Bump Size Versus Variable Bump Size (I)

Examples			Fixed Bump-Size		Variable Bump-Size		
Name	FETs	Delay (nsec)	Area (μm)	Time (sec)	Area (μm)	Time (sec)	Saving (%)
5xp1	318	7.0	2367	231.2	2360	158.5	31.4
9sym	180	12.0	730	26.4	729	13.9	47.3
bw	518	7.0	3332	797.0	3339	605.7	24.0
rd53	124	3.0	4113	59.0	4102	46.4	21.4
rd73	278	5.0	5711	274.1	5662	200.8	26.7
sao1	350	7.0	6292	575.9	6292	406.5	29.4
sao2	416	7.0	4317	470.5	4301	331.0	29.6
vg2	266	7.0	3448	235.6	3445	170.7	27.5

Table 5.2: Fixed Bump Size Versus Variable Bump Size (II)

mance, then the quality of the optimization will be offset by the inaccuracies incurred. However, if the model is too complex, then delay computation will be too time-consuming to be useful in practical applications.

5.3.1 MOSFET Model

Fig. 5.2 shows the MOSFET model used in this thesis. Although the transistor size x is treated as design parameter, in CMOS design, the channel length is fixed, so x is actually the transistor width. It should be noted that the capacitances C_g, C_d, and C_s, are all proportional to x, but the channel resistance R_{ds} is inversely proportional to x.

5.3.2 Distributed RC Delay Model

Fig. 5.3 illustrates the distributed RC model [41] used in this thesis to compute time delays for logic gates. In general, a logic gate has a different delay for each input. For a rising-edge input, this delay is defined to be the time from when the input rises to the midpoint voltage to when the output drops below the midpoint voltage. In the figure, the discharge time T_{rf} for the NFET pulldown network can be expressed in terms of the channel resistances R1, R2, R3 and the node capacitances C1, C2. Note that in the expression for T_{rf}, (R1 + R2) is equal to the resistance from node *N2* to power rail and (R1 + R2 + R3) is equal to the resistance from *N3* to power rail. The model also holds for PFET pullup networks.

To compute the delays, it is necessary to know how the R's and C's in the distributed RC model are computed. First, consider the node capacitance. Let N be a node and C_N the corresponding node capacitance. Then C_N is computed as follows:

- If N sees a drain node or a source node with transistor size x (μm), then a capacitance of CJA $\times$ DIFF_HANG $\times\ x + 2 \times$ CJP $\times$ (DIFF_HANG $+\ x$) is added, where CJA = diffusion area capacitance ($pf/\ \mu m^2$),

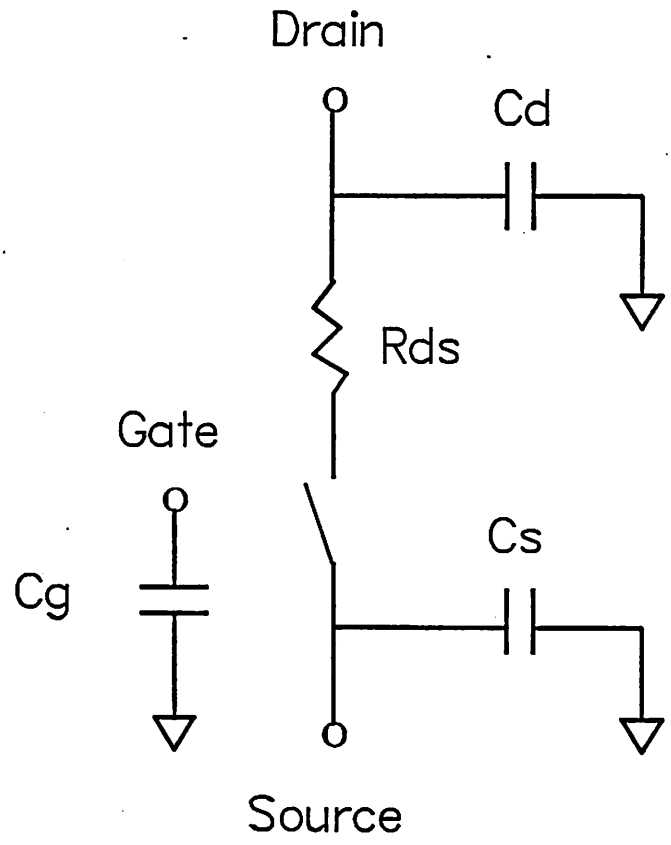


Figure 5.2: MOSFET Model

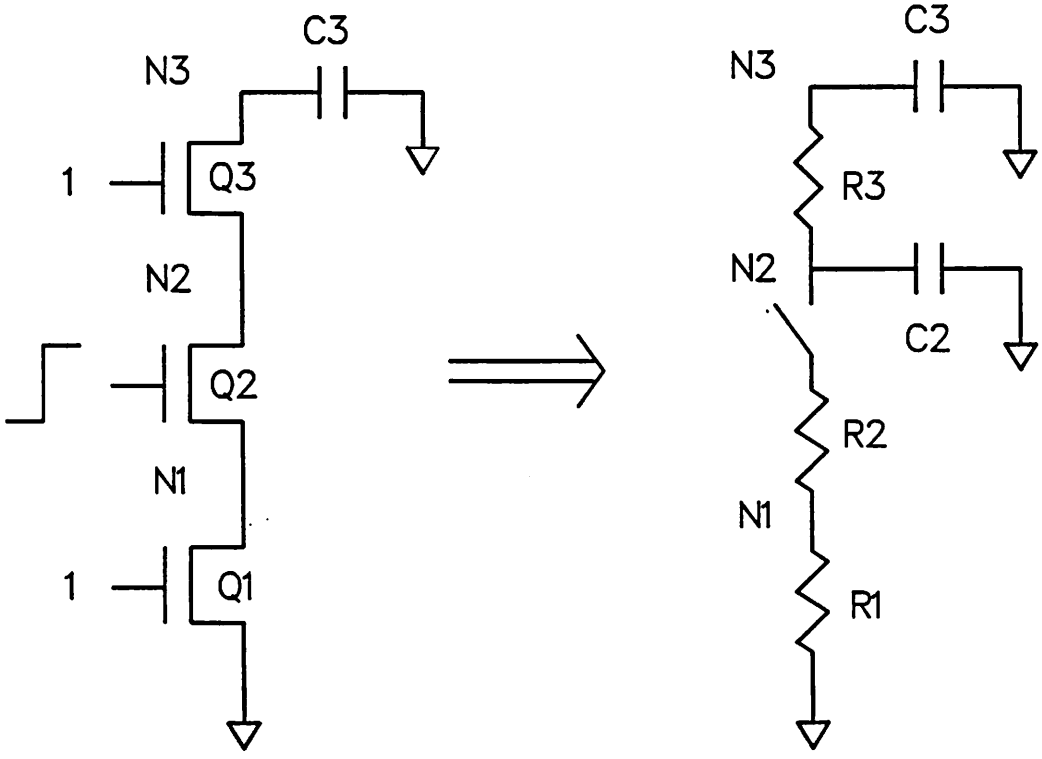


Figure 5.3: RC Delay Model

DIFF_HANG = the length diffusion extends beyond the active area (μm), CJP = diffusion perimeter capacitance ($pf/\ \mu m$).

- If N sees a gate node with transistor size x (μm), then a capacitance of CGTA $\times$ MASKCHANNEL $\times\ x$ is added, where CGTA = gate capacitance ($pf/\ \mu m^2$), MASKCHANNEL = mask channel length (μm).

- If N sees n fanouts, then a wiring capacitance is added. The wiring capacitance for N is either user specified or PARASITIC $\times\ (0.5 + n)$, where PARASITIC = default wiring capacitance per fanout (pf).

- If the user specifies that the node is connected to an external load capacitance, then that capacitance is added.

Next, consider the R's. From the delay model, only two types of R's are needed:

- R_Q - the channel resistance of a transistor Q. This is equal to the unit sheet resistance (Ω-cm) divided by the width x of Q.

- R_{N-P} - the resistance from a node N to the power rail. This is equal to the maximum value of path resistance over all possible paths to the rail from N.

Although the actual resistance may be less than this, depending on the other input values, this model gives an upper bound on the delay through the circuit. Fig. 5.4 illustrates an example.

Now the delay of a digital circuit can be computed. Since a digital circuit consists of logic gates and each logic gate consists of a pullup and a pulldown network only, the time delay of a circuit is the sum of the times to charge/discharge a chain of pullup/pulldown networks. Thus it suffices to know the delay computation for a logic gate. Suppose a logic gate has one and only one output node. (It can have more than one input node and each node can be connected to more than one input transistor.) To compute the

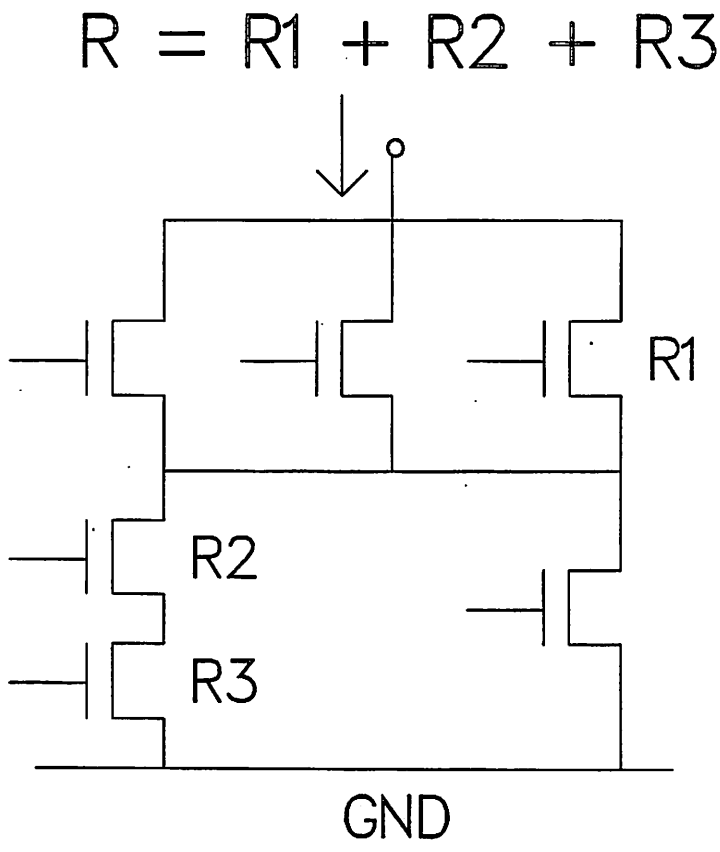


Figure 5.4: Resistance Calculation

delay through a logic gate, consider the circuit shown in Fig. 5.5. Let T_{rf}^{Q} be the discharge time for transistor Q when other transistors on the discharging path to OUT are on. Let T_h^N/T_l^N be the time reference when the signal at node N goes high/low. Then

$$T_l^{OUT} = \max_{Qi \in pulldown} \left\{ T_h^{INi} + T_{rf}^{Qi} \right\}$$

Similarly, the T_h^{OUT} can be computed by considering the pullup network. This scheme will be used to compute the delay sensitivities later.

Note that for combinational CMOS circuits which are currently supported by the program, the user only needs to specify the transistor netlist, the actual data-ready time for each input port, and the desired data-ready time for each output port. (For sequential circuits, the timing analyzer also needs to know the input clock waveforms.) Based upon this information, the transistors are grouped into logic gates. More specifically, given a node which is the output node of a gate, the transistors whose drains/sources are connected to that node are searched to form the pullup/pulldown networks; these transistors are then grouped into a gate.

In order to validate the accuracy of the delay model with regard to complex gates, the following experiment was performed by TILOS authors using the ADVICE circuit simulator (a program based on SPICE2) which has been used extensively in AT&T Bell Laboratories: An 8-bit adder in 1.75 μm CMOS was sized five times by TILOS which uses this delay model to achieve delays of 20, 18, 15, 13.5, and 12 ns. Table 5.3 compares the delays given by ADVICE and TILOS for these circuits. Since the TILOS runs are performed prior to layout, wire capacitances are conservatively estimated (by the formula used in capacitance computation rules) based on its fanout. The circuit file input for the above ADVICE runs, by contrast, are the result of a complete mask extraction on the actual layout, in which wire capacitances happen to be so small as to be negligible. The net result is that the delay model over-estimates the wire capacitances, and hence the gate delays, when transistor sizes are small (i.e., when the delays are large). When transistor

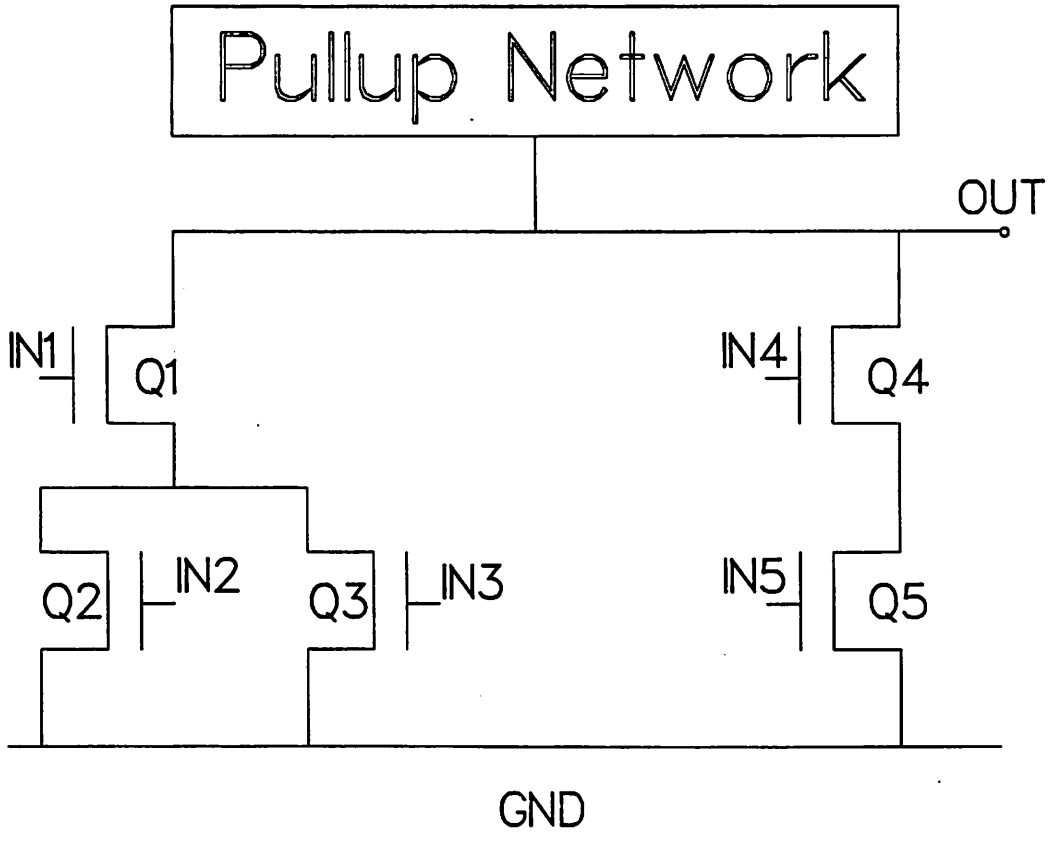


Figure 5.5: Delay Computation

TILOS delay (nsec)	ADVICE delay (nsec)	TILOS/ADVICE
20.0	17.8	1.12
18.0	16.3	1.10
15.0	14.2	1.06
13.5	13.4	1.10
12.0	12.3	0.98

Table 5.3: ADVICE Versus TILOS Delay Characterization of an 8-bit Ripple-Carry Adder

sizes are large (i.e., when the delays are small), the over-estimate is not serious, because the contribution of the wire capacitance is relatively small. This trend accounts for much of the model's over-estimate of delays for the slower versions of this circuit.

5.4 Feasible Directions Algorithm and Generalized Gradient

5.4.1 Feasible Directions Algorithm

As has been explained in Chapter 3, feasible directions methods have a desirable feature that once the feasible region (the set of transistor sizes where all the constraints are satisfied) is entered, all the subsequent improvements will remain feasible. The algorithm used here is an enhanced version of the Phase I/Phase II method of feasible directions (MFD) [18] [62]. Phase I of MFD tries to obtain a solution that satisfies all the constraints, i.e., to enter the feasible region. Once a feasible solution is obtained, the method reduces the value of the objective function without leaving the region. Phase I and Phase II have the same structure, only the objective function is different. In both phases, given a point x, a search direction, i.e., a vector in the n-dimensional space of the design parameters, is found based on the gradients of the objective and of some of the constraints. It is possible to

prove that there exists a step small enough along the search direction that decreases the objective function and some of the constraints. Once the search direction has been computed, a step along this direction is computed so that the decrease in the objective and constraint functions is large enough. In theory, the algorithm converges to a point where the length of the vector representing the search direction is zero. At this point a first order necessary optimality condition is satisfied. In practice, the computation is stopped when the length of this vector is small enough.

5.4.2 Generalized Gradient

Note that the MFD, as most of the optimization algorithms with guaranteed convergence properties, requires that the objective and constraint functions be continuously differentiable to converge to a stationary point. Unfortunately, in digital circuits, the delay between an input port and an output port is nondifferentiable since it is defined to be the maximum of all possible path delays between the two ports. As an example, consider Fig. 5.6 where $g_1(x)$ and $g_2(x)$ are two path delay functions with equal delays at some point x_0 in the design parameter space. The maximum function of the two is nondifferentiable at x_0.

At this point, the importance of optimizing all the competing delay paths at the same time should be emphasized. Consider a gate D driving several fanout gates. If all fanouts are equally critical, then obviously more resources should be devoted toward speeding up gate D than the other fanout gates, since all paths go through D. If only a longest-delay path were to be optimized, both gate D and its fanout gates would be speeded up, resulting in an over-sized circuit. In fact, an experiment has been performed optimizing along a single path on one such circuit at each iteration; the result is significantly worse than what could be obtained by optimizing all the critical paths (38% more area for a gate driving 8 fanouts).

To cope with this problem, a modification of MFD is used which

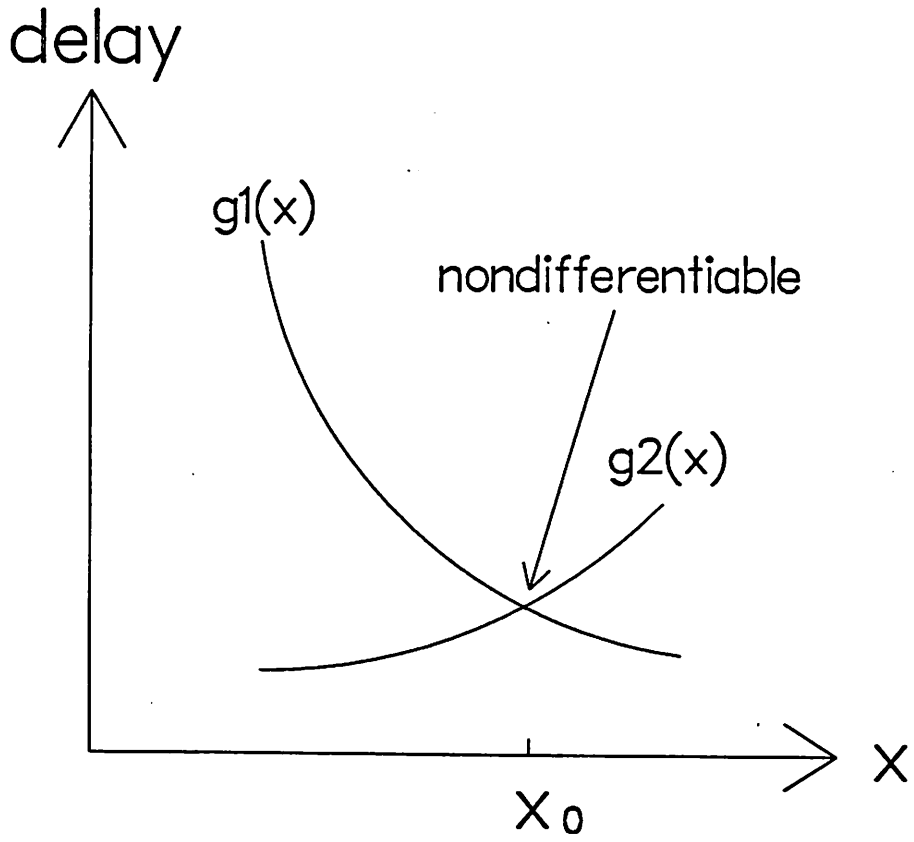


Figure 5.6: Geometry of Nondifferentiable (Max) Functions

uses the concept of *generalized gradient* [89]. This version of the algorithm can converge to a stationary point even though the objective and constraint functions are not continuously differentiable. It will be shown in the experimental result section, that algorithms with guaranteed convergence properties work faster and produce better final results than algorithms that do not possess such properties.

The nondifferentiability of the circuit delay originates from the *max* function used to define the circuit delay. Let

$$\psi(x) = \max_i g_i(x), i = 1, ..., m$$

be the circuit delay, where x is the vector of the design parameters (transistor sizes) and the g_i's are the path delays. Let:

$$I(x) = \{i \,|\, g_i(x) = \psi(x)\}$$

If the set $I(x)$ has only one element, then the circuit delay is obviously differentiable at x since all the path delay are continuously differentiable functions. However, if $I(x)$ has more than one element, then the circuit delay is nondifferentiable at x as demonstrated in Fig. 5.6. The generalized gradient is introduced to compute descent directions for $\psi(x)$ in these points. Its formal definition is as follows:

$$\partial\psi(x) = co\,\{\nabla g_i(x) \,|\, i \in I(x)\}$$

where *co* denotes the convex hull of the vectors $\nabla g_i(x)$, i.e., the set defined by

$$\sum_{i \in I(x)} \mu_i \nabla g_i(x) = 0$$

$$\sum_{i \in I(x)} \mu_i = 1, \mu_i \geq 0, i \in I(x)$$

Note that whenever $I(x)$ is a singleton, the generalized gradient is a standard gradient, while if it has more than one element, $\partial\psi(x)$ is a set as shown in Fig. 5.7.

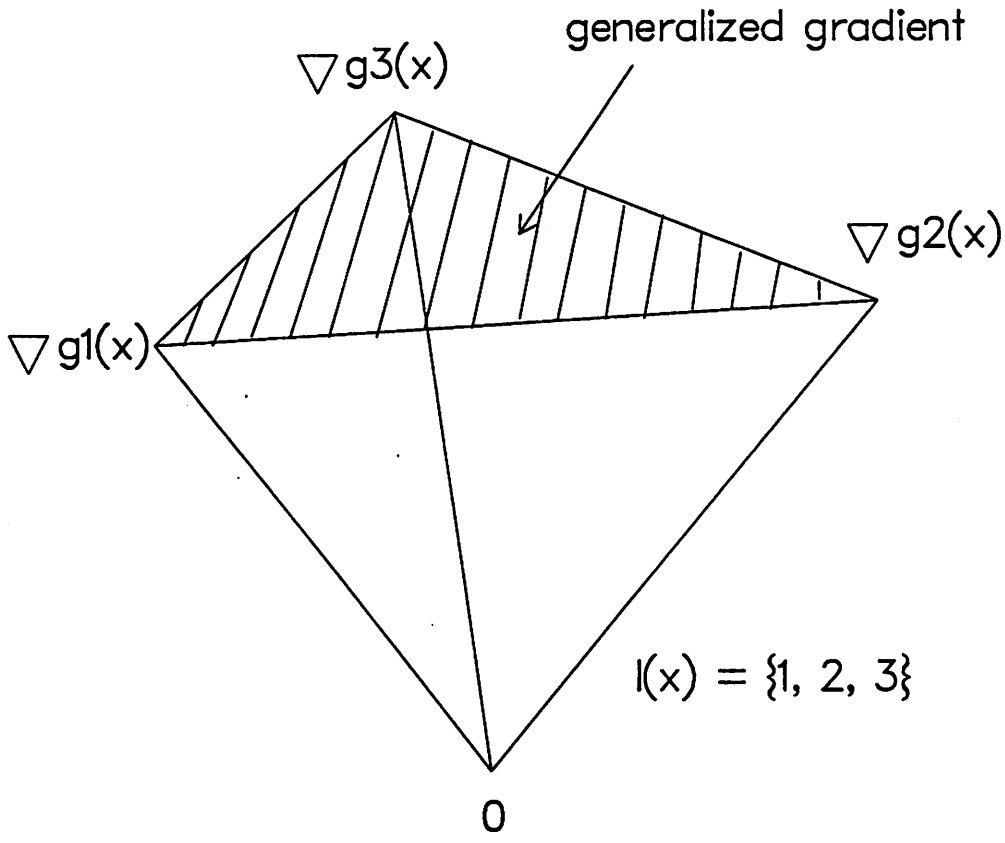


Figure 5.7: Geometrical Interpretation of Generalized Gradient

It can be shown that if $\hat{x}$ is a local minimum of $\psi(x)$, i.e., there exists $\delta > 0$ such that

$$\psi(\hat{x}) \leq \psi(x), x \in B(\hat{x}, \delta)$$

then

$$0 \in \partial\psi(\hat{x})$$

which is analogous to the standard necessary optimality condition

$$\nabla\psi(\hat{x}) = 0$$

for a continuously differentiable function.

In this version of the MFD algorithm a set which is closely related to the generalized gradient is used. This set is defined as follows: Let

$$I_\epsilon(x) = \{i \,|\psi(x) - g_i(x) \leq \epsilon\}$$

be the *ϵ-active* index set, where ϵ is a small positive number. Then the set used here is

$$\partial_\epsilon\psi(x) = co_{j\in I_\epsilon(x)} \{\nabla g_j(x)\}$$

In fact, if the generalized gradient as defined above is used to compute the search direction, it can be shown that the algorithm may jam at a non-stationary point. In this case the search direction is computed taking into account not only the "critical" path delays that define the circuit delay but also the "important" path delays, i.e., the ones that may become "critical" delays soon.

The MFD algorithm uses the shortest vector in the set $\partial_\epsilon\psi(x)$ to compute the search direction, $h_\epsilon(x)$, i.e.,

$$h_\epsilon(x) = -Nr(\partial_\epsilon\psi(x))$$

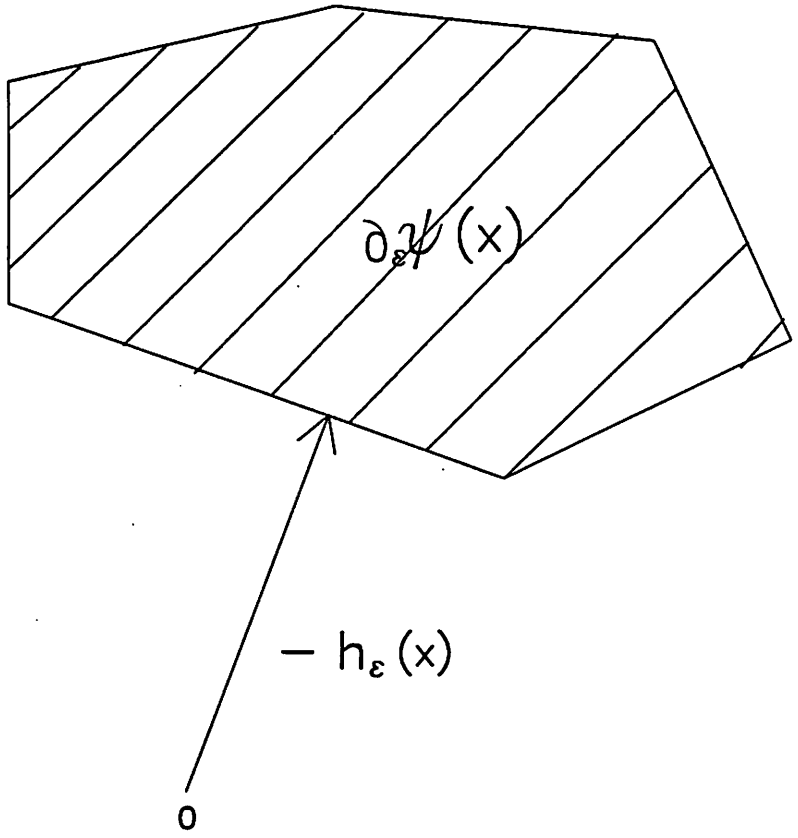


Figure 5.8: Geometrical Calculation of Generalized Gradient

where Nr denotes the nearest vector to the origin or in other words the shortest vector in the set $\partial_\epsilon\psi(x)$. In Fig. 5.8, $h_\epsilon(x)$ is computed graphically.

In summary, to apply the MFD algorithm to our problem it is necessary to find the ϵ-active delay paths and compute their derivatives with respect to the design parameters. In circuit design terminology, it is necessary to compute the sensitivities of the ϵ-active delay paths.

Note that to compute the ϵ-active delay paths, it is only necessary to consider the ϵ-active paths for the output stage. The reason is that, as the gate delays propagate to the output ports, the ϵ-active paths are reflected in the inputs of the last stage. Thus, once they are found, the algorithm can simply go backward through preceding stages to input ports for each ϵ-active path.

An efficient way of collecting the sensitivity information is depicted in Fig. 5.9 where each rectangular box represents a logic gate. Let the gradient vectors (the sensitivity information) of these ϵ-active paths be KIHFDA, KIHFDB, KIHGDA, KIHGDB, KIHGEC, KLHFDA, KLHFDB, KLHGDA, KLHGDB, and KLHGEC. These vectors can be computed by the path-delay sensitivity computation rules as the corresponding paths are traced out. To collect them efficiently, a simple recursive routine can be written to go backward from *OUT* to *IN* to trace out these paths and form the gradient vectors row by row into a matrix. Whenever more than one input in a gate (marked by *) is detected, the part of the sensitivity information which belongs to the preceding stages (looking back to *OUT*) are "duplicated". For instance, after KIHFDA is traced out, the routine goes to the * at the output of gate B; KIHFD is duplicated and B is computed taking into account the effect of the input transistors of gate D (some of those transistors are the fanouts of gate B). Thus, as the paths are traced out, all the sensitivity information can be obtained. Note that although in theory, combinatorial problem may arise, in many practical examples, only several ϵ-active delay paths need be collected.

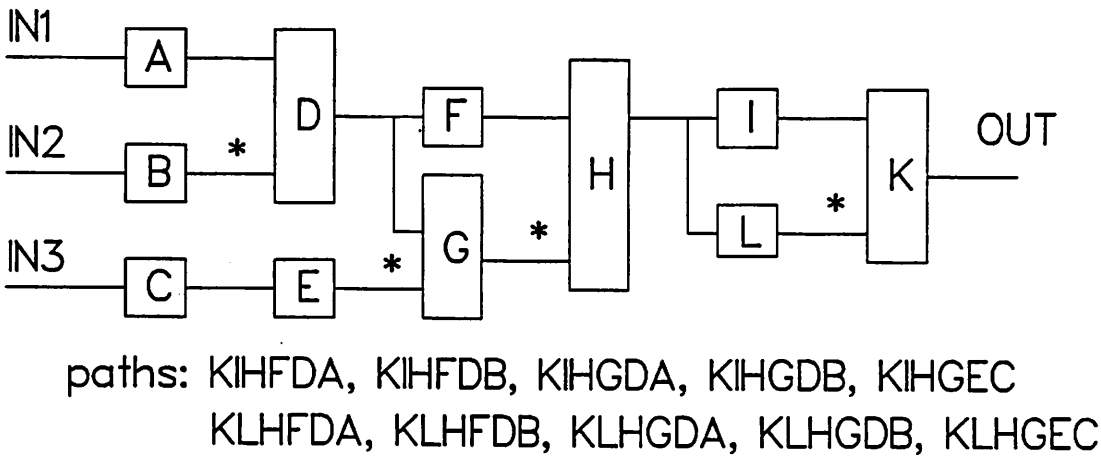


Figure 5.9: Enumerating the ϵ-active Paths

5.4.3 Problem Scaling

The use of the gradients of the objective and of the path delays with their unscaled values may produce poor descent search directions. The reason for this can be traced to the large difference in magnitude of the gradients of the objective and constraint functions. If a gradient has a magnitude that is much smaller than the other gradients, it dominates the search direction, thus producing a poor descent direction for the other constraints and/or for the objective function since the nearest vector to the origin will be almost identical to this gradient. To see that indeed the magnitude of the gradients may be quite different, consider the problem to solve:

minimize $\sum X_i$

such that: $G(X) \leq T$ and $X \geq K$

Here the objective function is the sum of the sizes of all the selected transistors, and the constraints are $G(X) - T \leq 0$ and $-X + K \leq 0$. It is obvious that the gradient of the objective function with respect to each transistor size is $[1\ 1\ ...\ 1\ 1]$ and the gradient of the k-th minimum-size constraint is $[0\ ...\ 0\ -1\ 0\ ...\ 0]$ where -1 is in the k-th position. However, the elements in the gradient of $G(X) - T$ are of the order of 10^{-7}. To solve this problem, note that (1) minimizing $\sum X_i$ is equivalent to minimizing $10^{-7} \sum X_i$ and (2) $X \geq K$ is equivalent to $10^{-7} X_i \geq 10^{-7} K$. Thus, multiplying a factor of 10^{-7} (to the objective function and the minimum-size constraints) scales down the gradients and makes the direction searching problem better conditioned. In fact, all the gradients have approximately the same magnitude.

5.5 Delay Sensitivity Computation

As was explained in the last section, the sensitivity calculation of the path delay is a crucial step in the application of the MFD to the transistor sizing problem. Although when high precision is not important, finite difference approximation can be used to compute the sensitivities, the com-

putational cost is very high in sizing even a medium size circuit. Hence sensitivity computation based on the delay model is desirable.

Before discussing the path-delay sensitivity computation, the *critical paths* and the *critical transistors* must be defined. Suppose a circuit has M input ports $IN_i, i = 1, ..., M$, and N output ports OUT_j, $j = 1, ..., N$. For any OUT_j, there is an IN_i such that the going high/low of the signal at IN_i makes OUT_j go high/low at some later time. Among all the possible paths from IN_i to OUT_j, there is at least one path with maximal delay. This path is called a critical path. For a critical path, a transistor is called a critical transistor of the path delay if it belongs to one of the following three types of transistors:

- CritTran: the transistor through whose drain and gate the critical path goes.
- BlockTran: the (blocking) transistor which is in series between the drain of CritTran and the output node (the target).
- SupTran: the (supporting) transistor in series between the source of CritTran and the power rail (SupTran is on the maximum-resistance path to power rail).

In Fig. 5.3, if the critical path goes from the gate of Q2 to N3, then Q2 is a CritTran, Q3 is a BlockTran and Q1 is a SupTran.

To see how to compute the path-delay sensitivities, consider a path between an input port and an output port. Since the path consists of a series connection of alternating pullup and pulldown networks, the total delay on the path is the sum of the gate delays on the path. If the logic gates are numbered sequentially as G_0 (the input stage), G_1, ..., G_n (the output stage), then it is clear that $T_l^i = T_h^{i-1} + T_{rf}^i$ and $T_h^i = T_l^{i-1} + T_{rf}^i$, where T_{rf}^i is the charge/discharge time depending on the pullup/pulldown transistors in G_i and the input transistors of G_{i+1}. From delay computations, it is clear that if a transistor Q is in G_i, then it is only necessary to consider its effect on T_l^i/T_h^i

and possibly T_l^{i-1}/T_h^{i-1} if Q is also an input transistor. Hence it suffices to know the sensitivity of the delay of a logic gate with respect to the size of a transistor.

Consider the NFET pulldown network in Fig. 5.10 where the delay path is $IN - Q_1 - Q_3 - OUT$. Note that Q_1 is a CritTran, Q_2 is a SupTran, Q_3 is a BlockTran, Q_4 is a fanout, and Q_5 is a transistor which is not on the delay path but whose drain node touches the output node. Let D be the delay from IN to OUT. To derive expressions for path-delay sensitivities, we define the following:

Q = transistor Q (Q can be Q_i, $i = 1, 2, 3, 4, 5$)

x_i = transistor size of Q_i

Q_d = the drain node of Q

Q_s = the source node of Q

Q_g = the gate node of Q

C_N = node capacitance at node N (N can be any node)

C_{N2-OUT} = capacitance looking from node N2 to output node
$= C_{N2} + C_{OUT}$

R_{N-P} = resistance looking from node N to power rail

R_Q = channel resistance of Q

By the delay model, D is given by:

$$\begin{aligned} D &= C_{N2}R_{N2-P} + C_{OUT}R_{OUT-P} \\ &= C_{N2}R_{N2-P} + C_{OUT}(R_{N2-P} + R_{Q3}) \\ &= C_{N2-OUT}R_{N2-P} + C_{OUT}R_{Q3} \end{aligned}$$

Hence, the delay sensitivities are:

$$\begin{aligned} \frac{\partial D}{\partial x_1} &= C_{N2-OUT}\frac{\partial R_{N2-P}}{\partial x_1} + \frac{\partial C_{N2-OUT}}{\partial x_1}R_{N2-P} \\ &= C_{N2-OUT}\frac{\partial R_{Q1}}{\partial x_1} + \frac{\partial C_{N2-OUT}}{\partial x_1}R_{N2-P} \\ &= C_{N2-OUT}\frac{\partial R_{Q1}}{\partial x_1} + \frac{\partial C_{Q1d}}{\partial x_1}R_{N2-P} \\ &= C_{Q1d-OUT}\frac{\partial R_{Q1}}{\partial x_1} + \frac{\partial C_{Q1d}}{\partial x_1}R_{Q1d-P} \end{aligned}$$

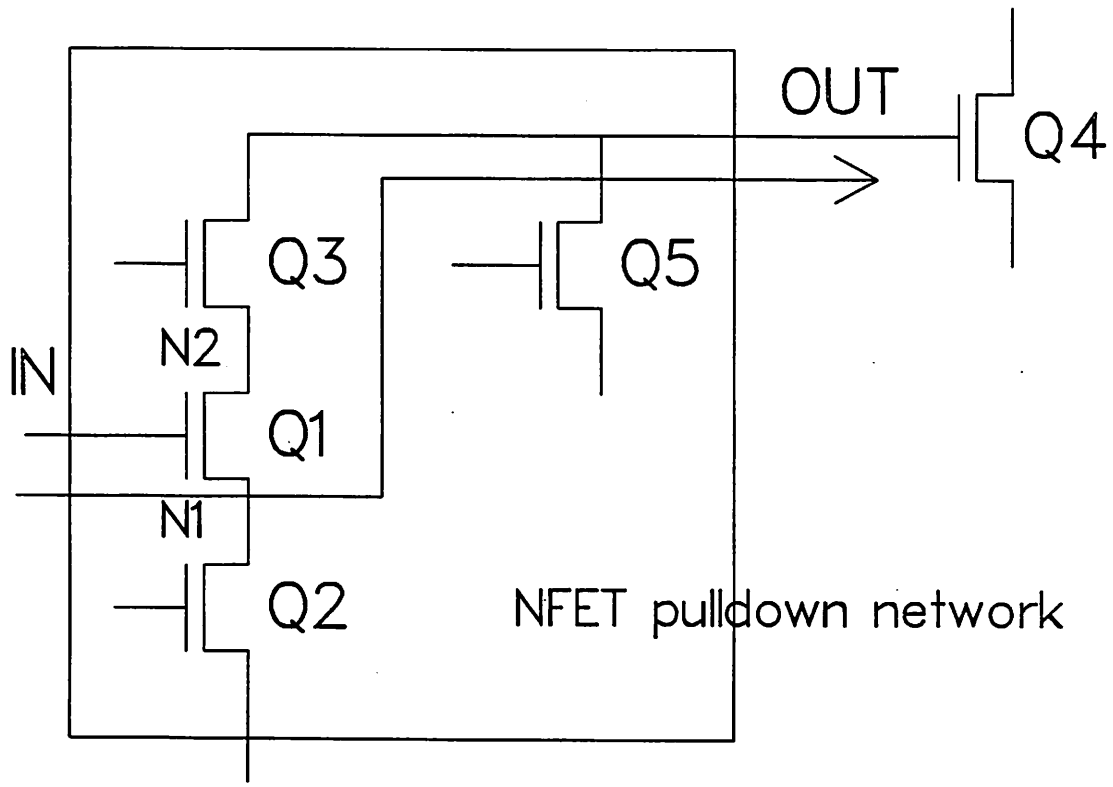


Figure 5.10: Path-Delay Sensitivity Computation

$$\begin{aligned}\frac{\partial D}{\partial x_2} &= C_{N2-OUT}\frac{\partial R_{N2-P}}{\partial x_2} \\ &= C_{N2-OUT}\frac{\partial R_{Q2}}{\partial x_2} \\ &= C_{Q1d-OUT}\frac{\partial R_{Q2}}{\partial x_2}\end{aligned}$$

$$\begin{aligned}\frac{\partial D}{\partial x_3} &= \frac{\partial C_{N2-OUT}}{\partial x_3}R_{N2-P} + \frac{\partial C_{OUT}}{\partial x_3}R_{Q3} + C_{OUT}\frac{\partial R_{Q3}}{\partial x_3} \\ &= 2 \times \frac{\partial C_{Q3d}}{\partial x_3}R_{N2-P} + \frac{\partial C_{Q3d}}{\partial x_3}R_{Q3} + C_{OUT}\frac{\partial R_{Q3}}{\partial x_3} \\ &= 2 \times \frac{\partial C_{Q3d}}{\partial x_3}R_{Q3s-P} + \frac{\partial C_{Q3d}}{\partial x_3}R_{Q3} + C_{Q3d-OUT}\frac{\partial R_{Q3}}{\partial x_3}\end{aligned}$$

$$\begin{aligned}\frac{\partial D}{\partial x_4} &= \frac{\partial C_{N2-OUT}}{\partial x_4}R_{N2-P} + \frac{\partial C_{OUT}}{\partial x_4}R_{Q3} \\ &= \frac{\partial C_{OUT}}{\partial x_4}R_{OUT-P} \\ &= \frac{\partial C_{Q4g}}{\partial x_4}R_{OUT-P}\end{aligned}$$

$$\begin{aligned}\frac{\partial D}{\partial x_5} &= \frac{\partial C_{N2-OUT}}{\partial x_5}R_{N2-P} + \frac{\partial C_{OUT}}{\partial x_5}R_{Q3} \\ &= \frac{\partial C_{OUT}}{\partial x_5}R_{OUT-P} \\ &= \frac{\partial C_{Q5d}}{\partial x_5}R_{OUT-P}\end{aligned}$$

In general, the following rules can be obtained:

- If Q is a CritTran, then

$$\frac{\partial D}{\partial x} = C_{Qd-OUT}\frac{\partial R_Q}{\partial x} + \frac{\partial C_{Qd}}{\partial x}R_{Qd-P}$$

- If Q is a SupTran, and Q_1 is the CritTran, then

$$\frac{\partial D}{\partial x} = C_{Q1d-OUT}\frac{\partial R_Q}{\partial x}$$

- If Q is a BlockTran, then

$$\frac{\partial D}{\partial x} = 2 \times \frac{\partial C_{Qd}}{\partial x} R_{Qs-P} + \frac{\partial C_{Qd}}{\partial x} R_Q + C_{Qd-OUT} \frac{\partial R_Q}{\partial x}$$

- If Q is a driven by the output node, then

$$\frac{\partial D}{\partial x} = \frac{\partial C_{Qg}}{\partial x} R_{OUT-P}$$

- If Q is not any of the previous types but touches the output node, then

$$\frac{\partial D}{\partial x} = \frac{\partial C_{Qd}}{\partial x} R_{OUT-P}$$

- If Q does not belong to any of the previous types, then

$$\frac{\partial D}{\partial x} = 0$$

Since the R_Q, C_{Qd}, and C_{Qg} are available from the MOSFET model, and C_{Qd-OUT}, R_{Qd-P}, R_{Qs-P}, and R_{OUT-P} are available from the delay model, the path-delay sensitivities can be obtained with virtually no cost.

5.6 Experimental Results

The algorithm has been implemented to size static combinational CMOS circuits. Tables 5.4 - 5.6 summarize the experimental results on several examples. The asterisk-marked examples are benchmark circuits from [88]. The CPU times are in seconds of DEC VAX 785. Table 5.4 shows some examples which are sized both in their reduced parameter space and in their complete parameter space. (It should be noted, however, that for high speed designs, a circuit may have to be sized in its complete parameter space.) It is observed that working in the complete parameter spaces does not give better results. Note that the slight differences in the areas are due to termination criterion and round-off mechanism.

Table 5.5 shows the effect of optimized transistor sizing over TILOS heuristic results. The *area* of TILOS results are obtained by using a

Examples			Complete Space		Reduced Space	
Name	FETs	Delay (nsec)	Dimension	Area (μm)	Dimension	Area (μm)
add2	54	5.0	54	604	40	602
buffer	72	5.0	72	424	56	415
alu1	86	8.0	86	969	72	952
9sym*	180	12.0	180	733	132	740
add8	222	15.0	222	2278	102	2277
5xp1*	318	11.0	318	1446	158	1451
sao1*	350	11.0	350	2862	290	2834
sao2*	416	11.0	416	2397	287	2400
bw*	518	11.0	518	2103	264	2074

Table 5.4: Complete Space Versus Reduced Space

BUMPSIZE of 1.1. Note that although for most cases, smaller BUMPSIZE (closer to 1) can produce better results (but takes longer time), for some examples such as *buffer* and *random*, the effect is not obvious. The optimized results using generalized gradients are obtained by using an initial guess with BUMPSIZE 1.5, with all the algorithm-related parameters pre-adjusted. It is interesting to note that, while in all cases, the new algorithm produces a better result, in some of the examples shown, the presented approach improves little over TILOS results.

Table 5.6 compares the sensitivity computation using generalized gradient and using finite difference. It is clear that the generalized gradient approach is much faster and also provides better solutions.

Examples			TILOS	Optimized	
Name	FETs	Delay (nsec)	Area (μm)	Area (μm)	Time (sec)
random	34	1.8	1208	900	17
add2	54	5.0	607	602	5
buffer	72	5.0	574	415	9
alu1	86	7.0	1499	1475	16
9sym*	180	10.0	900	860	73
add8	222	15.0	2286	2278	13
5xp1*	318	10.0	1563	1527	140
sao1*	350	10.0	3334	3233	100
sao2*	416	10.0	2539	2461	116
bw*	518	10.0	2317	2231	157

Table 5.5: Optimized Transistor Sizing

Examples			Generalized Gradient		Finite Difference	
Name	FETs	Delay (nsec)	Area (μm)	Time (sec)	Area (μm)	Time (sec)
random	34	1.8	900	17	1132	1406
add2	54	5.0	602	5	602	52
buffer	72	5.0	415	9	572	187
alu1	86	7.0	1475	16	1465	1845
9sym*	180	10.0	860	73	915	105
add8	222	15.0	2278	13	2278	1533
5xp1*	318	10.0	1527	140	1582	2108
sao1*	350	10.0	3233	100	3214	16967
sao2*	416	10.0	2461	116	2595	1288
bw*	518	10.0	2231	157	2305	22500

Table 5.6: Generalized Gradient Versus Finite Difference

Chapter 6

Conclusions and Future Research

6.1 Conclusions

Optimization tools are becoming important in high-performance IC designs. This thesis has addressed both interactive and automatic optimization techniques for improving the performances of ICs at the transistor level. While in theory, circuit optimization can be thought of as problem formulation plus optimization, it is observed that many factors such as poor problem formulation and implementation difficulties of algorithms, may cause numerical problems, resulting in premature termination of the algorithms.

The purpose of this research has been to provide tools to alleviate these difficulties, under the assumption that most designers do not have an extensive optimization background. For interactive optimization, the user interface has been designed so that the designer can formulate the problem easily. While efficient gradient-based algorithms are used to solve optimization problems, robust random search is invoked automatically to continue the optimization in case numerical difficulties arise in the gradient-based algorithms. For automatic optimization of digital circuits, a combination of fast heuristic and powerful nonlinear programming is used.

An interactive optimization system, ECSTASY, utilizing circuit simulation, has been proposed for optimizing general analog circuits. The system interfaces to SPICE3 with sensitivity computation capability, and optimizes circuit performance by using gradient-based algorithms. A forms-based user interface is provided to allow the designer to specify the problem easily. The system also offers informative visual feedback and menu-driven control for efficient user interaction. During optimization, when numerical difficulties occur, a controlled random search algorithm takes over the optimization process automatically. Thus, the optimization algorithms are transparent to the designer. To illustrate the performance of the proposed design system, the solutions of four circuit optimization examples have been described.

Optimization of digital circuits is performed by a proposed transistor sizer which adjusts the transistor sizes so that the total active area is optimized with all delay constraints satisfied. The program combines the TILOS heuristic and the method of feasible directions. It is observed that a variable bump-size scheme is effective in speeding up the heuristic for meeting circuit delays. After applying the TILOS heuristic to obtain a feasible design for the delay constraints, the program converts the problem into a nonlinear program, and solves it in a space of reduced dimensionality. The program also demonstrated the use of generalized gradients in computing the delay sensitivities.

6.2 Future Research

One of the assumptions in user interface design for general-purpose interactive optimization systems is "to speak the same language as the designers". This assumption is valid only if the front-end language of the interfaced simulator is powerful enough for both the simulation and optimization purposes. Since most existing simulators have their own languages, it would be beneficial to develop a standard simulation language for the circuit design

community, for this purpose.

Another possible future research direction is to perform optimization in a distributed or parallel computing environment. Although optimization is expensive on Von Neumann machines, the algorithm is inherently parallelizable. For instance, parallel versions of optimization algorithms can be devised so that function evaluations are performed concurrently. In a computer-aided engineering (CAE) environment in which optimization is an integral part of the system, it is desirable for the designer to specify the available computing facilities to perform distributed or parallel optimization from within the CAE environment. This requires the integration of parallel algorithms and simulation interfaces through remote procedure calls.

Since sensitivity computation is expensive, and may not be available in many existing circuit simulators, it is beneficial to have algorithms which can optimize circuits without using sensitivity information. Random search, although effective in improving circuit performance, may not be efficient enough in some situations. Research into nongradient-based algorithms will thus be important in the future.

Concerning transistor sizing, since the program focuses on the performance at the transistor level, many important issues such as those of placement and routing are ignored. Therefore, a worthwhile research objective would be to provide an environment to automatically integrate transistor sizing with other design tools so that the chip performance can be globally optimized.

Bibliography

[1] M. E. Van Valkenburg. *Introduction to Modern Network Synthesis.* Wiley, New York, 1960.

[2] M. G. R. Degrauwe et al. IDAC: An Interactive Design Tool for Analog CMOS Circuits. *IEEE Journal of Solid-State Circuits*, SC-22(6):1106–1116, December 1987.

[3] H. Y. Koh, C. H. Sequin, and P. R. Gray. Automatic Synthesis of Operational Amplifiers Based On Analytic Circuit Models. In *Digest of Technical Papers, 1987 IEEE International Conference on Computer-Aided Design*, pages 502–505, November 1987.

[4] R. Harjani, R. A. Rutenbar, and L. R. Carley. A Prototype Framework for Knowledge-Based Analog Circuit Synthesis. In *Proceedings of the 24th Design Automation Conference*, pages 42–49, July 1987.

[5] D. D. Gajski, editor. *Silicon Compilation.* Addison-Wesley Publishing Company, Inc., 1988.

[6] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):1062–1081, November 1987.

[7] G. De Micheli. Performance-Oriented Synthesis in the Yorktown Silicon Compiler. In *Digest of Technical Papers, 1986 IEEE International Conference on Computer-Aided Design*, pages 138–141, November 1986.

[8] M. Hofmann. Delay Optimization of Combinational Static CMOS Logic. In *Proceedings of the 24th Design Automation Conference*, pages 125–132, July 1987.

[9] E. Polak. *Computational Methods in Optimization.* Academic Press, 1971.

[10] G. D. Hachtel, M. R. Lightner, and H. J. Kelly. Application of the Optimization Program A0P to the Design of Memory Circuits. *IEEE Transactions on Circuits and Systems*, CAS-22(6):496–503, June 1975.

[11] G. D. Hachtel and P. Zug. *APLSTAP - Circuit Design and Optimization System - User's Guide.* Technical Report, IBM Yorktown Research Facility, 1981.

[12] W. T. Nye, D. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits. DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-7(4):501–519, April 1988.

[13] *ASTAP Advanced Statistical Analysis Program.* IBM Program Product Document SH20-1118-0 edition, 1973.

[14] R. K. Brayton and R. Spence. *Sensitivity and Optimization.* Elsevier Scientific, the Netherlands, 1980.

[15] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* Technical Report ERL-M520, Electronics Research Laboratories, University of California, Berkeley, 1975.

[16] B. W. Kernighan. RATFOR - A Preprocessor for a Rational Fortran. *Software - Practice & Experience*, 5(4):395–406, October-December 1975.

[17] E. Polak and D. Q. Mayne. An Algorithm for Optimization Problems with Functional Inequality Constraints. *IEEE Transactions on Automatic Control*, AC-21(2):184–193, April 1976.

[18] E. Polak, R. Trahan, and D. Q. Mayne. Combined Phase I-Phase II Methods of Feasible Directions. *Mathematical Programming*, 17(1):61–73, 1979.

[19] W. T. Nye. *DELIGHT: An Interactive System for Optimization-Based Engineering Design.* PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, May 1983.

[20] H. C. Lin and L. W. Linholm. An Optimized Output Stage for MOS Integrated Circuits. *Journal of Solid-State Circuits*, SC-10(2):106–109, April 1975.

[21] A. Kanuma. CMOS Circuit Optimization. *Solid-State Electronics*, 26(1):47–58, January 1983.

[22] N. Hedenstierna and K. O. Jeppson. CMOS Circuit Speed and Buffer Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(2):270–281, March 1987.

[23] A. Mohsen and C. Mead. Delay-Time Optimization for Driving and Sensing of Signals on High Capacitance Paths of VLSI Systems. *Journal of Solid-State Circuits*, SC-14(2):462–470, April 1979.

[24] E. T. Lewis. Optimization of Device Area and Overall Delay for CMOS VLSI Designs. *Proceedings of the IEEE*, 72(6):670–689, June 1984.

[25] S. M. Kang. A Design of CMOS Polycells for LSI Circuits. *IEEE Transactions on Circuits and Systems*, CAS-28(8):838–843, August 1981.

[26] L. A. Glasser and L. P. J. Hoyte. Delay and Power Optimization in VLSI Circuits. In *Proceedings of the 21st Design Automation Conference*, pages 529–535, June 1984.

[27] C. M. Lee and H. Soukup. An Algorithm for CMOS Timing and Area Optimization. *IEEE Journal of Solid-State Circuits*, SC-19(5):781–787, October 1984.

[28] B. A. Richman, J. E. Hansen, and K. Cameron. A Deterministic Algorithm for Automatic CMOS Transistor Sizing. In *Proceedings of the IEEE 1987 Custom Integrated Circuits Conference*, pages 421–424, May 1987.

[29] S. Trimberger. Automated Performance Optimization of Custom Integrated Circuits. In *Proceedings of the IEEE 1983 International Symposium on Circuits and Systems*, pages 194–197, May 1983.

[30] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, Reading, Massachusetts, 1980.

[31] W. H. Kao, N. Fathi, and C. H. Lee. Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits. In *Proceedings of the 22nd Design Automation Conference*, pages 781–784, July 1985.

[32] J. P. Fishburn and A. E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *Digest of Technical Papers, 1985 IEEE International Conference on Computer-Aided Design*, pages 326–328, November 1985.

[33] A. E. Ruehli, P. K. Wolff, and G. Goertzel. Power and Timing Optimization of Large Digital Systems. In *Proceedings of the IEEE 1976 International Symposium on Circuits and Systems*, pages 402–405, April 1976.

[34] A. E. Ruehli, P. K. Wolff, and G. Goertzel. Analytical Power/Timing Optimization Technique for Digital System. In *Proceedings of the 14th Design Automation Conference*, pages 142–146, June 1977.

[35] B. J. Agule, J. D. Lesser, A. E. Ruehli, and P. K. Wolff. An Experimental System for Power/Timing Optimization. In *Proceedings of the 14th Design Automation Conference*, pages 147–152, June 1977.

[36] K. S. Hedlund. Models and Algorithms for Transistor Sizing in MOS Circuits. In *Digest of Technical Papers, 1984 IEEE International Conference on Computer-Aided Design*, pages 12–14, November 1984.

[37] K. S. Hedlund. Electrical Optimization of PLAs. In *Proceedings of the 22nd Design Automation Conference*, pages 681–687, June 1985.

[38] K. S. Hedlund. Aesop: A Tool for Automated Transistor Sizing. In *Proceedings of the 24nd Design Automation Conference*, pages 114–120, June 1987.

[39] M. D. Matson. *Macromodeling and Optimization of Digital MOS VLSI Circuits.* PhD thesis, MIT, January 1985.

[40] D. P. Marple. *Performance Optimization of Digital VLSI Circuits.* PhD thesis, Stanford, September 1986.

[41] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal Delay in RC Tree Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2(3):202–211, July 1983.

[42] F. W. Obermeier and R. H. Katz. An Electrical Optimizer that Considers Physical Layout. In *Proceedings of the 25th Design Automation Conference*, pages 453–459, June 1988.

[43] M. Shoji. Electrical Design of the BELLMAC-32A Microprocessor. In *Proceedings of the IEEE 1982 International Conference on Circuits and Computers*, pages 112–115, September 1982.

[44] E. Polak and A. L. Sangiovanni-Vincentelli. Theoretical and Computational Aspects of the Optimal Design Centering, Tolerancing, and Tuning Problem. *IEEE Transactions on Circuits and Systems*, CAS-26(9):795–813, September 1979.

[45] C. Gonzaga, E. Polak, and R. Trahan. An Improved Algorithm for Optimization Problems with Functional Inequality Constraints. *IEEE Transactions on Automatic Control*, AC-25(1):49–54, February 1980.

[46] E. Polak and A. L. Tits. A Recursive Quadratic Programming Algorithm for Semi-Infinite Optimization Problems. *Applied Mathematics and Optimization*, 8(4):325–349, August 1982.

[47] D. Q. Mayne and E. Polak. A Quadratically Convergent Algorithm for Solving Infinite Dimensional Inequalities. *Applied Mathematics and Optimization*, 9(1):25–40, October 1982.

[48] R. Hettich and W. van Honstede. On quadratically convergent methods for semi-infinite programming. In Hettich, editor, *Semi-Infinite Programming*, pages 97–111, Springer-Verlag, 1979. lecture notes in control and information sciences, Vol. 15.

[49] R. Hettich and W. van Honstede. An approximation method for semi-infinite problems. In Hettich, editor, *Semi-Infinite Programming*, pages 126–136, Springer-Verlag, 1979. lecture notes in control and information sciences, Vol. 15.

[50] E. R. Panier and A. L. Tits. *Globally Convergent Algorithms for Semi-Infinite Optimization Problems Arising in Engineering Design.* Technical Report TR-87-28, System Research Center, University of Maryland, College Park, Maryland, 1987.

[51] U. M. Garcia-Palomares and A. Restuccia. A Global Quadratic Algorithm for Solving a System of Mixed Equalities and Inequalities. *Mathematical Programming*, 21(3):290–300, November 1981.

[52] E. Polak and D. Q. Mayne. On the Finite Solution of Nonlinear Inequalities. *IEEE Transactions on Automatic Control*, AC-24(3):443–444, June 1979.

[53] D. Q. Mayne, E. Polak, and A. J. Heunis. Solving Nonlinear Inequalities in a Finite Number of Iterations. *Journal of Optimization Theory and Applications*, 33(2):207–222, February 1981.

[54] D. Q. Mayne and M. Sahba. An Efficient Algorithm for Solving Inequalities. *Journal of Optimization Theory and Applications*, 45(3):407–423, March 1985.

[55] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of Multiobjective Optimization.* Academic Press, Inc., 1985.

[56] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. A Survey of Optimization Techniques for Integrated-Circuit Design. *Proceedings of the IEEE*, 69(10):1334–1362, October 1981.

[57] E. Sandgren and K. M. Ragsdell. The Utility of Nonlinear Programming Algorithms: A Comparative Study - Parts I and II. *ASME Journal of Mechanical Design*, 102(3):540–551, July 1980.

[58] F. A. Lootsma. Ranking of Nonlinear Optimization Codes According to Efficiency and Robustness. In L. Collatz, G. Meinardus, and W. Wetterling, editors, *Konstruktive Methoden der Finiten Nichtlinearen Optimierung*, pages 157–158, Birkhauser, Basel, Switzerland, 1980.

[59] K. Schittkowski. *Nonlinear Programming Codes: Information, Tests, Performance.* Volume 183 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, Berlin Heidelberg New York, 1980.

[60] J. M. Mulvey, editor. *Evaluating Mathematical Programming Techniques.* Volume 199 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, Berlin Heidelberg New York, 1982.

[61] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading, Massachusetts, 1984.

[62] A. L. Tits, W. T. Nye, and A. L. Sangiovanni-Vincentelli. Enhanced Methods of Feasible Directions for Engineering Design Problems. *Journal of Optimization Theory and Applications*, 5(3), December 1986.

[63] E. R. Panier and A. L. Tits. A Superlinearly Convergent Feasible Method for the Solution of Inequality Constrained Optimization Problems. *SIAM Journal on Control and Optimization*, 25(4):934–950, July 1987.

[64] E. R. Panier, A. L. Tits, and J. N. Herskovits. A QP-Free, Globally Convergent, Locally Superlinearly Convergent Algorithm for Inequality Constraint Optimization. *SIAM Journal on Control and Optimization*, 26(4), September 1988.

[65] M. J. D. Powell. Convergence Properties of Algorithms for Nonlinear Optimization. *SIAM Review*, 28(4):487–500, December 1986.

[66] M. J. D. Powell. A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. In G. A. Watson, editor, *Numerical Analysis Proceedings, Dundee 1977*, pages 144–157, Springer-Verlag, Berlin, 1978.

[67] D. Q. Mayne and E. Polak. A Superlinearly Convergent Algorithm for Constrained Optimization Problems. *Mathematical Programming Study*, 16:45–61, 1982.

[68] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London, 1981.

[69] L. S. Lasdon and P. O. Beck. Scaling Nonlinear Programs. *Operations Research Letters*, 1(1):6–9, October 1981.

[70] D. Agnew. Improved Minimax Optimization for Circuit Design. *IEEE Transactions on Circuits and Systems*, CAS-28(8):791–803, August 1981.

[71] D. A. Pierre. *Mathematical Programming via Augmented Lagrangians.* Addison-Wesley, Reading, Massachusetts, 1975.

[72] R. A. Jarvis. Optimization Strategy in Adaptive Control: A Selective Survey. *IEEE Transactions on System, Man, and Cybernetics*, SMC-5(1):83–94, January 1975.

[73] L. C. W. Dixon and G. D. Szego, editors. *Toward Global Optimisation.* North-Holland, Armsterdam, 1975.

[74] L. C. W. Dixon and G. D. Szego, editors. *Toward Global Optimisation 2.* North-Holland, Armsterdam, 1978.

[75] T. Quarles. *Analysis of Performance and Convergence Issues for Circuit Simulation.* PhD thesis, University of California, Berkeley, December 1988.

[76] W. Nye. Techniques for Using SPICE Sensitivity Computations in DELIGHT.SPICE Optimization. In *Digest of Technical Papers, 1986 IEEE International Conference on Computer-Aided Design*, pages 92–95, Santa Clara, CA, November 1986.

[77] U. Choudhury. *Sensitivity Computations in SPICE3.* Master Thesis, University of California, Berkeley, December 1988.

[78] J. D. Foley, V. L. Wallace, and P. Chan. The Human Factors of Computer Graphics Interaction Techniques. *IEEE Computer Graphics and Applications*, 4(11):13–48, November 1984.

[79] W. D. Penniman. A Methodology for Evaluating Interactive System Usage. *ACM-SIGCHI Bulletin*, 15(4):6–11, April 1984.

[80] K. Maeda, Y. Miyake, J. Nievergelt, and Y. Saito. A Comparative Study of Man-Machine Interfaces in Interactive Systems. *ACM-SIGCHI Bulletin*, 16(2):44–61, October 1984.

[81] P. Heckel. *The Elements of Friendly Software Design.* Warner Books, 1984.

[82] J. D. Gould and C. Lewis. Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM*, 28(3):300–311, March 1985.

[83] L. A. Rowe. Fill-in-the-Form Programming. In *Proceedings of the 1985 Very Large Database Conference*, Stockholm, Sweden, August 1985.

[84] A. V. Aho and J. D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, Massachusetts, 1979.

[85] T. G. Lewis and M. Z. Smith. *Applying Data Structures.* Houghton Mifflin Co., Boston, Massachusetts, 1976.

[86] J. Gettys, R. Newman, and T. Della Fera. *Xlib - C Language X Interface, Protocol Version 10.* MIT, Cambridge, Massachusetts, 1986.

[87] B. D. Nelin. Analysis of Switched-Capacitor Networks Using General-Purpose Circuit Simulation Programs. *IEEE Transactions on Circuits and Systems*, CAS-30(1):43–48, January 1983.

[88] A. J. de Geus. Logic Synthesis and Optimization Benchmarks for the 1986 Design Automation Conference. In *Proceedings of the 23rd Design Automation Conference*, page 78, June 1986.

[89] F. H. Clarke. *Optimization and Nonsmooth Analysis.* Wiley-Interscience, New York, 1983.