An Open Architecture for Improving VLSI Circuit Performance

By

Fred W. Obermeier

B.S. (University of Cincinnati) 1983
M.S. (University of California) 1985

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved:

Randy H Katz	Chair	4/19/89 Date
Richard Newton		4/20/89
Andrew W. Shogan		4/25/89

****

An Open Architecture for Improving VLSI Circuit Performance

by

Fred W. Obermeier

(Reduced version of the Ph.D. thesis.)

An Open Architecture for Improving VLSI Circuit Performance

Fred W. Obermeier

ABSTRACT

Electrical performance and area improvement are important parts of the overall integrated circuit design task. However, few design tools allow easy exploration of the design space (area, delay, and power) or offer designers different performance alternatives. Given designer specified constraints on area, delay, and power, EPOXY will size a circuit's transistors and will attempt small circuit changes to help meet the constraints. The system provides an open flexible framework for developing and evaluating the effects of different area and electrical models, optimization algorithms, and circuit modifications.

EPOXY takes a physical and electrical description of the circuit and produces a series of symbolic equations that model its performance. This results in circuit performance evaluation 5 times faster than Crystal and 56 times faster when these equations are subsequently compiled. EPOXY employs a virtual-grid area model since the sum of transistor area is a better measure of dynamic power than cell area. Optimization of a CMOS eight-stage inverter chain illustrates this difference; a typical minimum power implementation is 32% larger than the one for minimum area.

Next EPOXY attempts to find a parameter assignment for the input variables of these equations, transistor widths, to meet the constraints while minimizing the user defined objective function. Previous transistor sizing systems are limited to fixed electrical models and only consider time and power tradeoffs. After evaluating two non-linear optimization techniques, the TILOS-style heuristic and augmented Lagrangian algorithm, a combination of the two was found to produce quality results rapidly.

If the performance constraints cannot be met by transistor sizing, EPOXY considers inserting buffers stages, rearranging transistors within a pull-down or pull-up tree, and splitting large transistors so that cell height and width can be traded off. This level handles the discrete decisions of proposing circuit alternatives while the two lower levels determine the best possible implementation for this alternative. From an implementation viewpoint, EPOXY's underlying equation abstraction of circuit performance automatically provides critical path information and allows rapid modification of the circuit structure. A typical speed improvement of 23% for several CMOS circuits was achieved over transistor sizing alone while satisfying difficult height (pitch) constraints.

Acknowledgements

First, I like to express my deep appreciation for Randy Katz for his patience, support, and encouragement. He often provided need inspiration during those difficult times. I'd also like to thank my research committee; Randy Katz, Richard Newton, and Andrew Shogan, for suggesting improvements to this dissertation.

My early association with the members of the Magic team; John Ousterhout, Michael Arnold, Gordon Hamachi, Robert Mayo, Walter Scott, and George Taylor, provided valuable exposure to research in computer aided design (CAD) for integrated circuits. Their work showed that a powerful and useful design system could be constructed cleanly from good basic ideas.

I particularly value the stimulating conversations with my fellow officemates; Gaetano Borriello, David Wood, David Gedye, and Ellis Chang. Despite our cramped quarters, the close interaction has enriched my graduate experience by exposure to a broader scope of research. I like to thank Bill Lin and Gino Cheng for their comments and suggestions. In addition, I would also like to express my deep appreciation to Thuan Nguyen for his personal support and encouragement.

David Marple provided a copy of his transistor sizing program, COP. His work served as a model for implementing the augmented Lagrangian optimization algorithm.

Throughout my college experience I have had the pleasure of working with many talented people. My assignment at the IBM T.J. Watson Research Center introduced me to a wonderful research setting. In particular, Ed Adams and Rolf-Dieter Fiebrich encouraged my work towards a Ph.D. degree. As an undergraduate, Thomas Ridgway provided an enjoyable and rewarding setting to develop hardware and software PC systems. His projects have stimulated my creative efforts and have provided invaluable knowledge of microprocessor-based systems and assembly language development.

This research was supported in part by an American Electronics Association Faculty Development Fellowship and a NSF grant MIP-83-52227. My first year at Berkeley was funded by a California, Microelectronics Fellowship (MICRO).

Table of Contents

List of Figures

List of Tables

1. Introduction

The computer industry is undergoing a rapid evolution. Each successive generation of computers is faster and more powerful than the previous one. Many of these improvements can be directly traced to developments in the underlying integrated circuit (IC) technology. Therefore, a company's success directly depends on how effectively these innovations can be rapidly incorporated into new products.

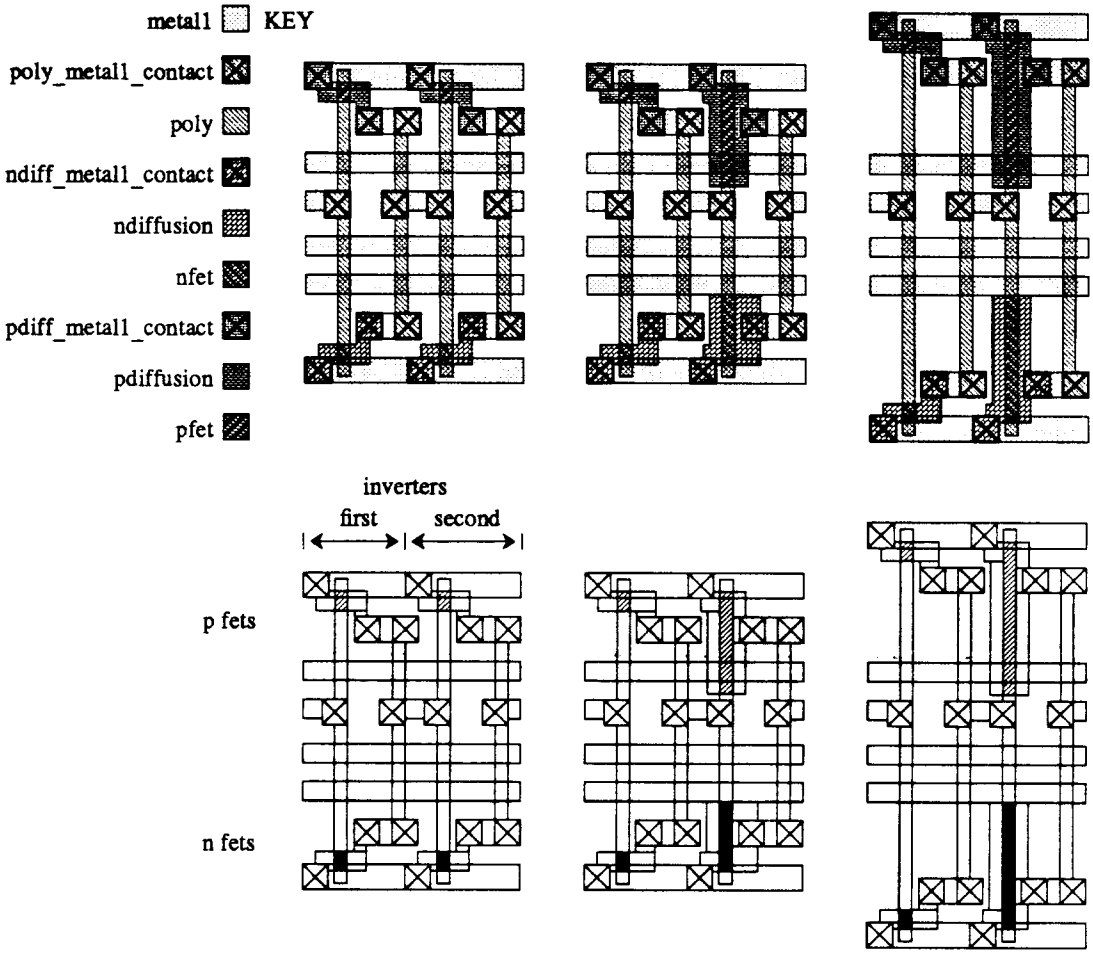
VLSI designers play a crucial role in the development of new products. These designers are faced with the difficult task of producing layout that meets aggressive electrical performance and area requirements. The current trend in the design of VLSI integrated circuits is to start with a functional specification using a high-level description. Next, a designer selects an architecture of major functional blocks that satisfies the overall functional requirements and general design strategy. The implementation of each block is constrained by the performance requirements dictated by the functional partitioning. Once these requirements are refined to a sufficiently detailed level, the designer can select a suitable layout generator or library element.

Typical layout generators and standard-cell library elements contain fixed over-sized or minimum-sized transistors. Often these systems produce only a single design. This gives a designer little control over the electrical and area performance of the resulting implementation. Even when transistor sizes can be specified, it is difficult for a designer, without some assistance, to determine the appropriate sizes for the large number of transistors to achieve the best implementation. Therefore automated transistor sizing plays an important role in the overall design process.

Figure 1.1 illustrates how the performance of a small example changes as a function of transistor widths. The layout implements two static CMOS inverters in a standard-cell organization. The upper layouts show all interconnect layers, while only the transistors are shaded in the lower layouts. Layouts on the left represent the design with all minimum sized transistors. The second inverter for the middle and right layouts employs wider transistors. As the transistors in the second inverter are widened, the delay is reduced at the expense of additional power. The trend continues until the fastest design is achieve. Thereafter, increases in the transistor widths will cause an undersirable increase in power and delay. The height and area changes only if the larger transistors cannot fit into the layout. This example does not adequately illustrate the increased complexity involved in determining the sizes for a large number of transistors when subject to constraints on performance.

Transistor sizing alone may not be sufficient for meeting the required performance goals for a defined functional block. Even though circuit changes at the logic level can have a significant impact on the quality of the circuit, most logic reorganization programs cannot consider the important detailed parasitics which can have a dramatic effect on the overall performance.

EPOXY, the design tool described in this dissertation, not only sizes transistors but also considers detailed circuit modifications for meeting the performance goals and size limitations. The system improves VLSI circuit performance by generating the equations that model circuit performance, formulating the transistor sizing



Delay	(ns)	45.0	11.5	7.4
Power	(μW)	26.4	27.4	28.4
Height	(λ)	50	50	67
Width	(λ)	34	34	34
Area	(λ^2)	1,700	1,700	2,278

Figure 1.1. **Layouts and performance of two inverters.** Two static CMOS inverters are implemented by layout in a standard-cell organization. In standard cells, the p- and n-type transistors (fets.) are grouped into rows. The complete layouts are shown in the top portion of this figure. Each transistor type is shaded in the lower layouts. The layouts on the left employ transistors of minimum size. Increasing the transistor widths of the second inverter decreases the delay (increases the speed) at the expense of additional power and height.

problem as a non-linear program, and then modifying the circuit to improve the solution further. Therefore, EPOXY can complete the design process by modifying the layout resulting from logical reorganization.

VLSI Performance Improvement Systems

System	Author	Obj.	Constr.	Method	Model	Tech.	Mode	Comments
–	[Glas84]	min T min AA min P	m≤AA T≤k & m≤AA T≤k & m≤AA	opt.	LS RC	nMOS	batch	Single path.
–	[Lee84]	min T min AA	 T≤k	opt.	L RC	CMOS	batch	(step 1) k=minT+ΔT
–	[Mats86]	min P	T≤k m≤AA≤k	opt.	Macro	nMOS	batch	
–	[Shyu88a]	min AA	T≤k	opt.	D RC	CMOS	b/i	
AESOP	[Hedl87]	min T min P	m≤AA≤k m≤AA≤k	opt.	L RC	n/CMOS	inter.	Fixed paths (gate model)
ANDY	[Trim83]	min T min P	m≤AA≤k T = k	heur.	L RC	nMOS	b/i	
APLSTAP	[Bray81]	min f(n)	f(n)≤k	opt.	ASTAP		inter.	Very general
DELIGHT	[Nye88]	min f(n)	f(n)≤k	opt.	SPICE		inter.	Very general
EO	[Hedl84]							1 path (PLAOPT)
EPOXY	[Ober88b]	min f(n)	f(n)≤k	all†	all†	all†	b/i	Also [Ober88a]
LTIME	[Ciri87]	min P	T≤k	opt.	D RC	MOS	inter.	Single path
MOST	[Pinc86]	min AA	T≤k	heur.	L RC	CMOS	b/i	A framework.
OPT	[Wolf78]	min P	T≤k	opt.	L RC	nMOS	batch	Place using T & P
PLAOPT	[Hedl85]	min T min P	m≤AA≤k T≤k	opt.	L RC	nMOS	inter.	PLA critical path oriented
PLATO	[Marp86]	min A	T≤k	opt.	LS RC	CMOS	batch	Dynamic PLA
COP	[Marp87]	min P min T min A	T≤k P≤k T≤k	opt.	LS RC	CMOS	batch	Static comb.
TILOS	[Fish85]	min T min AA min AA×T^c	m≤AA≤k T≤k & m≤AA≤k m≤AA≤k	heur.	D RC	CMOS	batch	Extracts sequential and combinational transistor nets.
TV/IA	[Joup83]	min AA	T≤k	heur.	D RC	nMOS	inter.	
XTRAS	[Kao85]	min AA	T≤k	heur.	LS RC	n/CMOS	inter.	T = max. all paths.

Key						
	A	total design area	T	delay	L RC	Lumped RC model
	AA	total transistor area	f(n)	any function	LS RC	Lumped Slope RC model
	H	total design height	c, k, m	constants	D RC	Distributed RC model
	W	total design width	opt.	optimization based	batch	batch mode opt.
	P	power	heur.	heuristic	inter.	interactive opt.

Table 1.1. **A comparison of VLSI performance improvement systems.** The objective function and constraints for each program are given under (Obj.) and (Constr.) respectively. The type of technique used for design improvement is given under (Method). The model which determines response times (Model) and the technology supported (Tech.) are then listed. The last two columns give the supported operating modes and addition comments. († EPOXY supports a variety of methods, models and technologies.)

EPOXY's major innovation is its efficient flexible open system architecture for developing and evaluating performance models, non-linear optimization techniques and circuit changes. The system delivers these advantages by representing circuit performance by a set equations. Each of these issues (performance modeling, optimization and circuit changes) corresponds to a component in EPOXY.

1.1. Comparison of Performance Improvement Systems

Since the relationship between transistor size (and layout dimensions) and delay is non-linear even for simple models, the transistor sizing problem can be formulated as a non-linear program (NLP). Even though a transistor sizing tool many not employ non-linear programming techniques, a transistor sizing program can be succinctly described by the non-linear program (an objective function and constraints) it solves.

The value of any performance improvement tool depends on how well the system helps a VLSI IC designer achieve the desired performance goals. However, these requirements vary radically depending on the intended application. Designers of high volume parts place chip area at a premium while requiring a sufficient speed and maximum power dissipation necessary for an application. Alternatively, advanced specialized parts often require high speed at the expense of additional area and operating power. Therefore, one metric of comparison of performance improvement systems is how flexible they are in specifying user constraints and an objective function, and assisting a user to attain these design requirements.

Table 1.1 summarizes several VLSI MOS performance improvement systems. The program name and reference for each system is given under the first two columns. The next two columns define the objective function and corresponding design constraints each system considers; a separate line is given for each type of optimization a system performs. The standard constraint, that transistor widths must be larger than some minimum width as defined by the technology, has been omitted from this table. The abbreviations for each of the design metrics is given in the lower portion of the table. The technique that a system uses to improve a circuit's performance can be classified as either optimization-based or heuristic. Optimization techniques rely on formal mathematical methods to assure converge to a minima while heuristic techniques encode designer experience in improving designs. Although heuristic techniques usually do not yield optimal results, they do improve designs rapidly. The electrical model for circuit response time and the technology supported by each system is listed in the two subsequent columns. The next column describes a user's interaction with each system as either batch or interactive. The final column provides terse comments which detail some unique aspects or limitations of each system.

Performance improvement systems can also be classified by the level of detail considered in determining the transistor sizes. The electrical model (Model in *Table 1.1*) can be applied at the transistor (e.g.: TILOS) or gate level (e.g.: AESOP). Gate level models typically reduce the input design space by associating a single factor with a logic gate. The transistor sizes are assigned based on the resultant logic gate factor. While gate level models reduce the input design space and the amount of computation, they often yield inferior results. The single gate factor cannot encompass significant differences in input arrival times and intermediate parasitic capacitance that dramatically effect the overall circuit performance. EPOXY is flexible enough to implement both transistor and gate level models.

1.2. Unique Aspects of EPOXY

EPOXY offers a unique approach to performance optimization. Most transistor sizing programs integrate fixed electrical models with a fixed optimization technique (e.g.: ANDY, COP, OPT, PLATO, TILOS, TV/AI). Performance improvement systems (e.g.: APLSTAP, DELIGHT) that permit a selection of models and optimization algorithms are computationally expensive because these two tasks are handled by separate programs. Separation of modeling and optimization routines results in extra storage and communication expense to maintain essentially equivalent information. However, EPOXY provides a unique flexibility in selecting, substituting, and evaluating performance models and optimization algorithms while retaining the computational efficiency of integrated systems.

The flexibility of performance improvement systems can be classified by the design metrics they consider. Typical design metrics include worst-case response times (circuit delay), maximum power consumption, total cell height, width, and area. However, most systems are restricted to minimizing a single predefined design metric and fixed constraints. TV/IA can provide a user with sensitivity information so that other design metrics can be taken into account manually. However, these systems inhibit a user from automatically achieving the desired tradeoff among other design metrics since they incorporate fixed objective functions and constraints. EPOXY allows a user to specify the objective function and constraints.

Even at the modeling level, many of the transistor sizing programs quote total transistor (active) area as an area metric. However, total transistor area does not accurately represent the overall area required for the design since it does not consider transistor placement and routing effects imposed by larger transistors. Total transistor area is actually a better measure of worst-case maximum dynamic power rather than total cell area. Therefore *Table 1.1* differentiates between these two metrics. The only two systems that model the area of the cell are COP and PLATO. Although COP's area model considers the relative placement of transistors, it does not take advantage of the significant area underneath the routing channel. However, PLATO does model details such as snaking transistors for PLA's. EPOXY's virtual grid area model accurately relates changes in transistor sizes to changes in the overall cell area for all layout styles.

Matson and Glasser [Mats86] discuss how the parameters for their nMOS gate model are automatically derived. EPOXY extends this concept by deriving the performance equations symbolically for the entire circuit of interest. Then the partial differential equations required by most non-linear optimization methods are derived. Therefore, the sensitivity information can be quickly and accurately provided by evaluating these differential equations without having to resort to approximations such as finite differences. In addition, EPOXY can easily incorporate new models since the underlying derivation and manipulation routines for the equations need not change. A new performance model need only generate the appropriate equations. The remaining routines can create the partial derivative equations and evaluate all of the generated equations.

Traditional circuit optimization programs have concentrated on transistor sizing to improve the speed of a circuit at the expense of increased power and area. However, if the design goals cannot be met by transistor sizing alone, structural techniques can be applied. EPOXY considers several circuit changes such as splitting large transistors into several smaller ones, inserting buffers, and rearranging transistors in pull-down or pull-up trees. While circuit changes can be considered during synthesis as in MIS [Bray86], MTA [Hofm87], and [Mich87], these tools consider the whole circuit and thus are forced to employ simplified models for performance and area estimation. EPOXY, on the other hand, makes local modifications at the layout level by examining the circuit in greater detail with more accurate models.

Global changes in circuit structure such as logic resynthesis are best made earlier in the design process. EPOXY deals with the remaining complexity of detailed circuit performance improvement given design constraints and performance goals. Therefore, the limited circuit structural changes that may be necessary can be made at a level where the performance impact of these changes can be accurately assessed.

1.3. Goals of this Research

Many of the previous transistor sizing tools tightly couple the electrical models with an optimization routine that solves a fixed predefined optimization problem. However, it is difficult for users to determine how much of the performance improvements come from using less accurate electrical models or from a particular optimization routine. EPOXY can answer these questions by providing a framework in which the user specifies an objective function and a set of constraints, and which can substitute the performance models and optimization routines. EPOXY's use of performance equations combines the execution speed comparable to integrated systems with the flexibility of applying different performance models, optimization routines and circuit modification techniques.

EPOXY is a system for sizing transistors and modifying circuits to help meet performance constraints. EPOXY's unique features addresses the concerns of VLSI circuit designers and design tool developers:

flexible A VLSI designer can specify (1) an objective function to minimize, and (2) additional performance constraints. Therefore, a designer can more readily determine the tradeoffs among several of the design metrics of interest such as area, time, power, etc.

A user can select the appropriate performance models, optimization routine and circuit modification techniques.

open The effects of model accuracy and speed of evaluation in producing an optimized design are easily determined. For example, a fair comparison of several electrical models can be made using the same optimization algorithm. The results of these tests can also suggest which models and optimization routines are appropriate for various technologies and design styles.

architecture Representing the performance by deriving a set of equations is the key to providing the **flexible** and **open** features. This strategy leads to equations that can be evaluated rapidly, but at the expense of additional memory. EPOXY provides several basic operations that support the equation representation of circuit performance:

1. Macros and function calls construct the equations directly in memory.
2. Circuits with feedback lead to equations that reference each other in a cycle. These cycles are removed by expanding each unique path through the cycle.
3. Equations are then ordered so they can be evaluated rapidly. Some of the possible evaluation modes are:
   a. Evaluate all equations.
   b. Evaluate only those equations affected by a change in one variable (incremental evaluation).
   c. Evaluate those equations affected by a change in a small number of variables (a larger-grain incremental evaluation). A variable whose value has changed is marked so that the corresponding equations can be re-evaluated.
   d. Output the equations symbolically so they can be compiled for additional speed and storage advantages.
2. New equations can be incrementally inserted in the linear ordering.
3. Partial derivative equations are generated from the performance equations automatically. These equations form the sparse Jacobian matrix required by several non-linear optimization-based routines.
4. Macros and function calls can alter the equations and corresponding node and transistor data structures. These routines directly manipulate the underlying circuit structure without re-extracting and rebuilding the entire circuit.

1.4. Organization of this Dissertation

In this dissertation, transistor sizing and local circuit modifications are investigated as a means of improving VLSI circuit performance. Each of these issues corresponds to a component of EPOXY. EPOXY's approach to improving VLSI circuit performance is to derive the equations from the layout which model the circuit performance, formulate the transistor sizing problem as a non-linear program (NLP), and then modify the circuit to improve the solution further. A key benefit to this representation is that clear comparisons can be made between different performance models, optimization routines, and circuit modification techniques.

The first two chapters will examine performance models and optimization algorithms necessary for sizing transistors to meet performance goals. *Chapter 2* details

the performance models currently available in EPOXY and describes how these models affect the final optimized results. The choice of models requires a careful balance between simulation accuracy and evaluation speed. The following chapter (*Chapter 3*) describes the optimization algorithms implemented within the tool. Each optimization algorithm is evaluated by the quality of solutions obtained for some representative circuits and by the running times to find these solutions. Solution quality is evaluated by the user specified performance goals.

Chapter 4 describes how several techniques are used to improve the solutions that fail to meet the design constraints after transistor sizing. Although specific circuit modifications are evaluated, the technique of how these modifications are implemented is described so that they may serve as a template for implementing other kinds of beneficial circuit modifications.

The conflicting requirements of rapid circuit modification and rapid evaluation speed were satisfied by generating equations that determine the circuit performance. The equations provide increased flexibility and improve the overall execution speed at the expense of additional memory. The larger memory requirement, which limits the maximum circuit size, is evaluated for several representative circuits. A discussion of the implementation strategy for the equations and evaluation of their effectiveness within EPOXY follows in *Chapter 5*. An example circuit illustrates the information that a user specifies and the equations that EPOXY generates.

The final chapter (*Chapter 6*) summarizes the major contributions of this thesis and outlines several research topics for future investigation.

2. Modeling Circuit Performance

The task of any performance improvement system is to determine an assignment of the input parameters that best satisfies the performance constraints. This problem has two main components: modeling the circuit performance and selecting the best implementation (optimization). This chapter will concentrate on the issues associated with circuit performance modeling.

A designer of integrated circuits is interested in several metrics that characterize the quality of layout. For MOS VLSI circuits, these include the response time (circuit delay), maximum power dissipation, total layout area, cell height and width. The designer has control over many input parameters such as the placement of transistors, their width and lengths, and parasitic capacitance. Electrical models relate these physical parameters to the electrical performance metrics, i.e., delay (time) and power. The area model provides total height and width and an estimation of layout parasitics. This chapter presents a virtual grid area model that considers detailed layout effects caused by changes in transistor sizes. As illustrated later in this chapter, this area model is well suited for many layout styles such as PLA's and standard cells.

The choice of the appropriate model for each design metric must carefully balance the conflicting requirements of accuracy in estimating the actual circuit performance and computational expense. Since the goal is to produce a quality layout that meets the desired performance constraints, the crucial issue is how these models affect the optimization results. The next subsections present models for each of the typical design metrics of interest to VLSI designers.

2.1. Time

A VLSI designer is faced with many node delays of potential interest. In his work on ATV [Wall88], an abstract timing verifier, Wallace outlines several timing models: single numbers, ranges (min-max), and statistical descriptions (mean and standard deviation), and asymmetric rise/fall versions of these. MOS designers are usually concerned with the worst-case response times (a single number model) for their layout as dictated by worst-case input arrival times and required output response times.

The input arrival times and required output response times are quantities given to a designer. Therefore, the input node times will be represented as constants in the problem formulation of the non-linear optimization, while the output node times will be specified as constraints.

Worst-case response times can be determined using gate-level or transistor-level models. Gate level models identify logic gates by grouping the transistors that drive an output node. All transistor sizes that compose the logic gate are then related by a single factor. For example, AESOP [Hedl87] scales all of the transistors in a gate using the same factor. The gate factors also determine the response times. Alternatively, transistor-level models determine node response times by calculating the effects of each transistor on the attached nodes. Optimization programs based on gate-level models such as AESOP are usually faster since the ensuing

optimization routine considers a much smaller input vector space (the gate factors). Because several transistors are used to create a single gate, there are many more transistor sizes than gate factors. Unfortunately, transistor sizing systems based on gate models produce many over-sized transistors since the single gate factor will unnecessarily increase the size of transistors that do not participate in any critical path. These larger transistors will consume more power by loading the output of the previous logic gate. Therefore, transistor-level models are better suited for producing quality optimization results.

RC Model	Transistor Model	Evaluation Time fets/sec	Average Error to SPICE†
Lumped	Linear	82.3	24%
Lumped	Slope	66.0	8%
Distributed	Slope	55.8	6%
$t_{out} = f(t_{in}, R, C)$	R = f(w,l,type) C = f(w,l,type)		

Table 2.1. **Summary of Crystal time models.** The evaluation time and accuracy for Crystal's time models in the first two columns are given by the third and fourth columns. The evaluation time is the average over 1000 timing analyses for a full 32 bit adder on a standard Sun-3/140‡. († Data in this column from [Oust85].) The transistor model determines the transistor's equivalent on-resistance and capacitance. The RC model combines these resistances and capacitances to determine the worst-case response times.

Ousterhout has outlined several value-independent (transistor-based) switch-level models for calculating worst-case delays given layout parasitics and transistor widths and lengths [Oust85] as described in *Table 2.1.* These include the lumped RC, lumped slope RC, and the distributed slope RC models. The lumped and distributed models differ in how the transistor resistance and capacitance values are combined to produce a worst-case node time. These models determine the delay effect of each transistor switching independently from all others. The linear transistor model assigns a single resistance depending on the transistor width, length, and type (p- or n-type). For most circuits, the linear transistor model is sufficient. However, slow rising and falling edges cause transistors to switch more slowly than fast rising edges. The slope transistor model also considers the signal rising and falling times in determining its resistance. For most logic gates, only one transistor switches at a time. Therefore the more accurate slope transistor model is used for the single switching transistor while all others use the simple linearly-interpolated default transistor model.

Equations for the response time of a node are derived using the template: $t_{output\ node} = \max(t_{input\ node} + delay())$ where $t_{input\ node}$ is the time the input changes. EPOXY currently provides a worst-case distributed linear RC electrical

‡ Sun-3 is a trademark of Sun Microsystems, Inc.

model for *delay*(), although other models could be substituted. EPOXY is not restricted to RC trees, as *Figure 2.1* demonstrates.

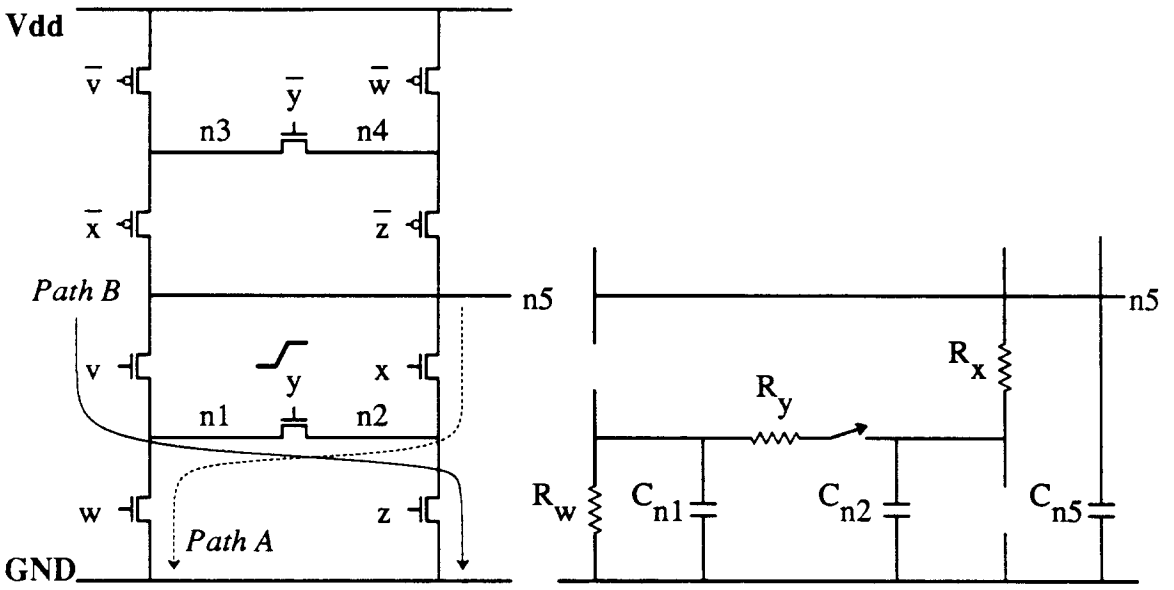


Figure 2.1. **Critical path derivation of a CMOS static gate.** A static CMOS combinational logic gate and the corresponding RC equivalent circuit for deriving one possible discharge path (*path A*) for node **n5** caused by a rising signal on node **y**, the gate of the transistor (**y**). R_{fet} is the on-resistance of the transistor (*fet*) and C_{node} is the total capacitance on the *node*.

The equations that determine the worst-case response time of node **n5** in *Figure 2.1* due to a rising transition on node **y** are listed below.

$$\textit{(Path A)} \qquad t_{fall,n5} \geq t_{rise,y} + (R_y + R_w)C_{n2} + (R_x + R_y + R_w)C_{n5};$$

$$\textit{(Path B)} \qquad t_{fall,n5} \geq t_{rise,y} + (R_y + R_z)C_{n1} + (R_v + R_y + R_z)C_{n5};$$

Since the derivation of these equations involves finding all paths from each output node through attached transistors to the power supply rails, EPOXY actually represents these equations using intermediate variables and equations. The equations that EPOXY generates to determine the response time for node **n5** are given below. Note that even a simple circuit fragment can generate a large number of equations. Several intermediate variables are used in the following equations to represent the resistance through a path of transistors from a particular node to the power supply in response to an input transition (rising or falling) on a transistor ($R_{transition,node,transistor,path}$) and the inherent delay of the switching transistor and all transistors between it to the power supply subject to an input transition ($t_{transition,transistor}$). This example illustrates that a simple circuit can generate several equations. These equations were generated automatically by EPOXY.

$$R_{fall,n1,fet\ w,path0} = R_{fet\ w};\quad R_{fall,n1,fet\ y,path0} = R_{fet\ y} + R_{fall,n2,fet\ z,path0};$$
$$R_{fall,n2,fet\ z,path0} = R_{fet\ z};\quad R_{fall,n2,fet\ y,path0} = R_{fet\ y} + R_{fall,n1,fet\ w,path0};$$

$$R_{fall,n3,fet\ \bar{v},path0} = R_{fet\ \bar{v}};\quad R_{fall,n3,fet\ \bar{y},path0} = R_{fet\ \bar{y}} + R_{fall,n4,fet\ \bar{w},path0};$$
$$R_{fall,n4,fet\ \bar{w},path0} = R_{fet\ \bar{w}};\quad R_{fall,n4,fet\ \bar{y},path0} = R_{fet\ \bar{y}} + R_{fall,n3,fet\ \bar{v},path0};$$
$$R_{fall,n5,fet\ v,path0} = R_{fet\ v} + R_{fall,n1,fet\ y,path0};$$
$$R_{fall,n5,fet\ v,path1} = R_{fet\ v} + R_{fall,n1,fet\ w,path0};$$
$$R_{fall,n5,fet\ x,path0} = R_{fet\ x} + R_{fall,n2,fet\ z,path0};$$
$$R_{fall,n5,fet\ x,path1} = R_{fet\ x} + R_{fall,n2,fet\ y,path0};$$
$$R_{fall,n5,fet\ \bar{x},path0} = R_{fet\ \bar{x}} + R_{fall,n3,fet\ \bar{y},path0};$$
$$R_{fall,n5,fet\ \bar{x},path1} = R_{fet\ \bar{x}} + R_{fall,n3,fet\ \bar{v},path0};$$
$$R_{fall,n5,fet\ \bar{z},path0} = R_{fet\ \bar{z}} + R_{fall,n4,fet\ \bar{y},path0};$$
$$R_{fall,n5,fet\ \bar{z},path1} = R_{fet\ \bar{z}} + R_{fall,n4,fet\ \bar{w},path0};$$
$$t_{fall,fet\ \bar{v}} \geq C_{n5}R_{fall,n5,fet\ \bar{x},path1} + C_{n3}R_{fall,n3,fet\ \bar{v},path0};$$
$$t_{fall,fet\ \bar{v}} \geq C_{n5}R_{fall,n5,fet\ \bar{z},path0} + C_{n4}R_{fall,n4,fet\ \bar{y},path0} + C_{n3}R_{fall,n3,fet\ \bar{v},path0};$$
$$t_{fall,fet\ \bar{w}} \geq C_{n5}R_{fall,n5,fet\ \bar{x},path0} + C_{n3}R_{fall,n3,fet\ \bar{y},path0} + C_{n4}R_{fall,n4,fet\ \bar{w},path0};$$
$$t_{fall,fet\ \bar{w}} \geq C_{n5}R_{fall,n5,fet\ \bar{z},path1} + C_{n4}R_{fall,n4,fet\ \bar{w},path0};$$
$$t_{fall,fet\ \bar{x}} \geq C_{n5}R_{fall,n5,fet\ \bar{x},path0};\qquad t_{fall,fet\ \bar{x}} \geq C_{n5}R_{fall,n5,fet\ \bar{x},path1};$$
$$t_{fall,fet\ \bar{y}} \geq C_{n5}R_{fall,n5,fet\ \bar{x},path0} + C_{n3}R_{fall,n3,fet\ \bar{y},path0};$$
$$t_{fall,fet\ \bar{y}} \geq C_{n5}R_{fall,n5,fet\ \bar{z},path0} + C_{n4}R_{fall,n4,fet\ \bar{y},path0};$$
$$t_{fall,fet\ \bar{z}} \geq C_{n5}R_{fall,n5,fet\ \bar{z},path0};\qquad t_{fall,fet\ \bar{z}} \geq C_{n5}R_{fall,n5,fet\ \bar{z},path1};$$
$$t_{fall,n5} \geq t_{rise,v} + t_{rise,fet\ v};\qquad t_{fall,n5} \geq t_{rise,w} + t_{rise,fet\ w};$$
$$t_{fall,n5} \geq t_{rise,x} + t_{rise,fet\ x};\qquad t_{fall,n5} \geq t_{rise,y} + t_{rise,fet\ y};$$
$$t_{fall,n5} \geq t_{rise,z} + t_{rise,fet\ z};$$
$$t_{rise,fet\ v} \geq C_{n5}R_{fall,n5,fet\ v,path0};\qquad t_{rise,fet\ v} \geq C_{n5}R_{fall,n5,fet\ v,path1};$$
$$t_{rise,fet\ w} \geq C_{n5}R_{fall,n5,fet\ v,path1} + C_{n1}R_{fall,n1,fet\ w,path0};$$
$$t_{rise,fet\ w} \geq C_{n5}R_{fall,n5,fet\ x,path1} + C_{n2}R_{fall,n2,fet\ y,path0} + C_{n1}R_{fall,n1,fet\ w,path0};$$
$$t_{rise,fet\ x} \geq C_{n5}R_{fall,n5,fet\ x,path0};\qquad t_{rise,fet\ x} \geq C_{n5}R_{fall,n5,fet\ x,path1};$$
$$t_{rise,fet\ y} \geq C_{n5}R_{fall,n5,fet\ v,path0} + C_{n1}R_{fall,n1,fet\ y,path0};$$
$$t_{rise,fet\ y} \geq C_{n5}R_{fall,n5,fet\ x,path1} + C_{n2}R_{fall,n2,fet\ y,path0};$$
$$t_{rise,fet\ z} \geq C_{n5}R_{fall,n5,fet\ v,path0} + C_{n1}R_{fall,n1,fet\ y,path0} + C_{n2}R_{fall,n2,fet\ z,path0};$$
$$t_{rise,fet\ z} \geq C_{n5}R_{fall,n5,fet\ x,path0} + C_{n2}R_{fall,n2,fet\ z,path0};$$
$$t_{rise,n5} \geq t_{fall,\bar{v}} + t_{fall,fet\ \bar{v}};\qquad t_{rise,n5} \geq t_{fall,\bar{w}} + t_{fall,fet\ \bar{w}};$$
$$t_{rise,n5} \geq t_{fall,\bar{x}} + t_{fall,fet\ \bar{x}};\qquad t_{rise,n5} \geq t_{fall,\bar{y}} + t_{fall,fet\ \bar{y}};$$
$$t_{rise,n5} \geq t_{fall,\bar{z}} + t_{fall,fet\ \bar{z}};$$

The transistor model provides the on-resistances and parasitic capacitances for a given width, length and type (n- or p-channel). EPOXY currently supports a

first-order approximation of a transistor's resistance and capacitance, the linear model. The constants, Kr_{type}, Kg_{type}, and Ksd_{type} are automatically determined by a least squares fit of these modeling equations to SPICE simulations of small representative circuits. Kr_{type} relates the transistor strength (length over width) to its average resistance subject to a fast switching input signal. The gate oxide capacitance per unit area, Kg_{type}, and the diffusion capacitance per transistor width, Ksd_{type}, can be determined instead from the process parameters for transistors. Since the source and drains of MOS devices are symmetric, equal values for the source and drain capacitances ($C_{sd,fet}$) are used.

$$R_{type,fet} = Kr_{type}\left[\frac{l_{fet}}{w_{fet}}\right]; \qquad \begin{aligned} C_{gate,fet} &= Kg_{type}\, w_{fet}\, l_{fet}; \\ C_{sd,fet} &= Ksd_{type}\, w_{fet}; \end{aligned}$$

Finally, node capacitance is determined by adding the parasitic capacitance and the capacitance of attached transistor gates, sources, and drains. The variables appearing in the node capacitance equation are determined by the interconnection of transistors within the circuit. For the example CMOS circuit, the capacitance of node n1 is given below.

$$C_{n1} = C_{parasitic\ n1} + C_{sd,v} + C_{sd,w} + C_{sd,y};$$

Although simple models may be sufficient for ordering critical paths, they are not likely to be accurate enough for optimization that requires absolute comparisons to delay constraints (e.g., "critical paths must be less than 20ns"). Aggressive designs are typically forced into the region where a designer must balance minimum delay (maximum speed) with an overall area limitation. A small error in estimating delay to meet a fixed time constraint can have a large impact on overall area. That is, if the timing model overestimates the delay, a faster than necessary design will be produced. The design is larger because it employs larger transistors. Similarly, a timing model that underestimates the delay will produce designs that do not meet the timing constraints. To correct the design, substantial additional area may be required. *Figure 2.2* clearly shows that overestimating the delay will produce larger designs for the minimization of area and power for an eight-stage CMOS inverter chain (using the MOSIS SCMOS λ=0.7μm design process). A 1ns time error results in a 3,956 μm^2 (8,074 λ^2) error for area minimization and 2,636 μm^2 (5,380 λ^2) error for power minimization. Therefore, if the model underestimates the delay by 16% (1ns), a circuit 100% larger than necessary will be reported for a minimum area implementation and 50% for a minimum power implementation. Since the lumped RC model has an average error of 25%, transistor sizing tools based on this model would exhibit even larger area and power errors.

Some designers are interested producing circuitry that is hazard and race free. Kohavi [Koha78] describes several conditions that can cause hazards or races. A hazard is a situation where a change in a single input may cause a momentary incorrect output. Whether or not an incorrect signal is actually generated depends on the exact delay associated with the various circuit elements. A race condition arises when two or more state variables are required to change their value simultaneously in a finite state machine. If the final state of the circuit does not depend on the order in which the input variables change, then the race is noncritical.

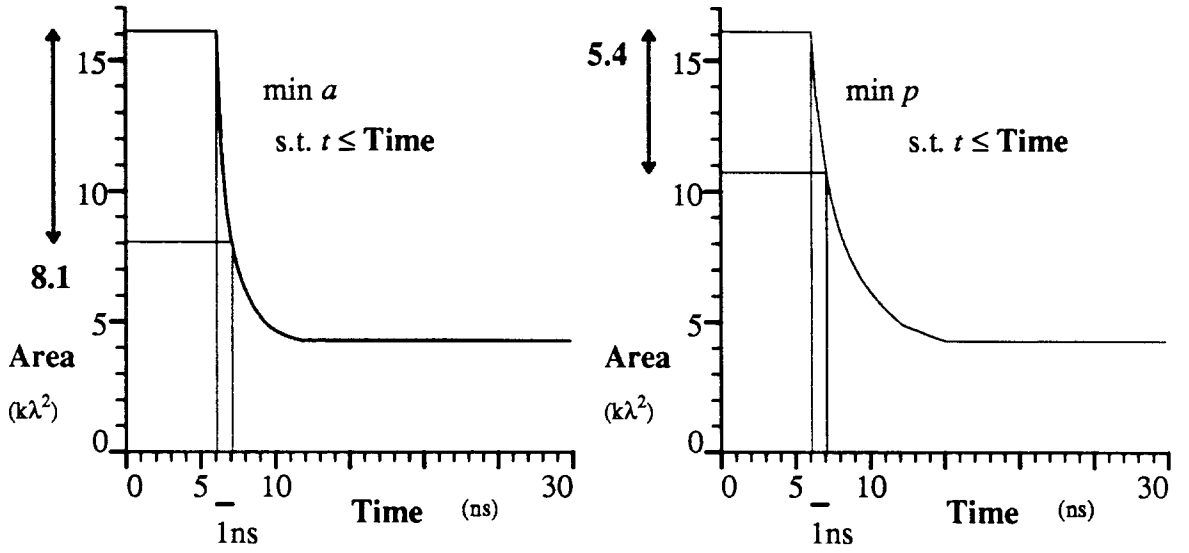


Figure 2.2. **Total cell area ≠ total transistor area.** The graphs indicate the total cell area (**Area**) required by an eight-stage inverter chain (MOSIS SCMOS λ=0.7μm) to achieve a specific delay (**Time**). The graph on the left shows the minimum area implementation while the one on the right gives the minimum power configuration. The smallest circuit that meets a 7.1ns time constraint requires an area of 8,044λ^2 (3,942 μm^2). The smallest circuit that meets a 6.1ns time constraint requires an area of 16,118λ^2 (7,898 μm^2). Therefore, a 1ns variation translates into an 8,074λ^2 (3,956 μm^2) area difference. Similarly, the lowest power circuit that meets the 7.1ns time constraint requires an area of 10,738λ^2 (5,262 μm^2). This translates into a 5,380λ^2 (2,636 μm^2) difference for a 1ns variation. (The output has an additional 1pf capacitance load.)

Designers of asynchronous circuits should avoid both of these difficulties.

Designers can eliminate some of the hazard or race conditions by constraining the performance problem. Lower bound constraints on some of the delay paths assure that a signal is stable long enough to avoid generating an incorrect signal. For asynchronous designs that employ level-sensitive latches, the fastest path though the combinational logic should not be less than half of the slowest path through the logic; otherwise, an input signal may feed through the circuit more than once per cycle. Unfortunately, single number timing models are incapable of generating the required information. However, a min-max (worst-case/best-case) timing model can generate the required fastest and slowest response times. A min-max timing model is easily implemented within EPOXY since the best-case equations are similar to the worst-case equations already described. The following equations implement a min-max timing model.

$$R_{min,type,fet} = Kr_{min,type}\left[\frac{l_{fet}}{w_{fet}}\right]; \qquad R_{max,type,fet} = Kr_{max,type}\left[\frac{l_{fet}}{w_{fet}}\right];$$

$$C_{gate,min,fet} = Kg_{min,type}\, w_{fet}\, l_{fet}; \qquad C_{gate,max,fet} = Kg_{max,type}\, w_{fet}\, l_{fet};$$

$$C_{sd,min,fet} = Ksd_{min,type}\, w_{fet}; \qquad C_{sd,max,fet} = Ksd_{max,type}\, w_{fet};$$

$$C_{min,node} = C_{parasitic} + \sum_{attachedfets} C_{gate,min,fet} + \sum_{attachedfets} C_{sd,min,fet};$$

$$C_{max,node} = C_{parasitic} + \sum_{attachedfets} C_{gate,max,fet} + \sum_{attachedfets} C_{sd,max,fet};$$

$$t_{min,output\ node} = \min(t_{min,input\ node} + \sum_{paths} R_{min,fet} C_{min,node});$$

$$t_{max,output\ node} = \max(t_{max,input\ node} + \sum_{paths} R_{max,fet} C_{max,node});$$

2.2. Power

The IC package imposes an maximum power dissipation limit (constraint) on the entire design. Total power has dynamic and static components. Static power usually dominates for nMOS and pseudo-nMOS design styles, while dynamic power dominates for most other CMOS design styles [West85]. Since dynamic power is a function of the operating frequency, a maximum operating frequency must be defined along with the maximum allowable power.

The static component is determined by the minimum resistance between the power supply signal lines. For nMOS, the smallest resistance occurs when all the transistors are on. Since the static current is mainly determined by the depletion-mode pull-up of each gate, the static maximum power can be approximated by:

$$power_{static,\ nMOS} = \sum_{gates} \left[\frac{V_{dd}^2}{R_{depletion,\ gate}} \right];$$

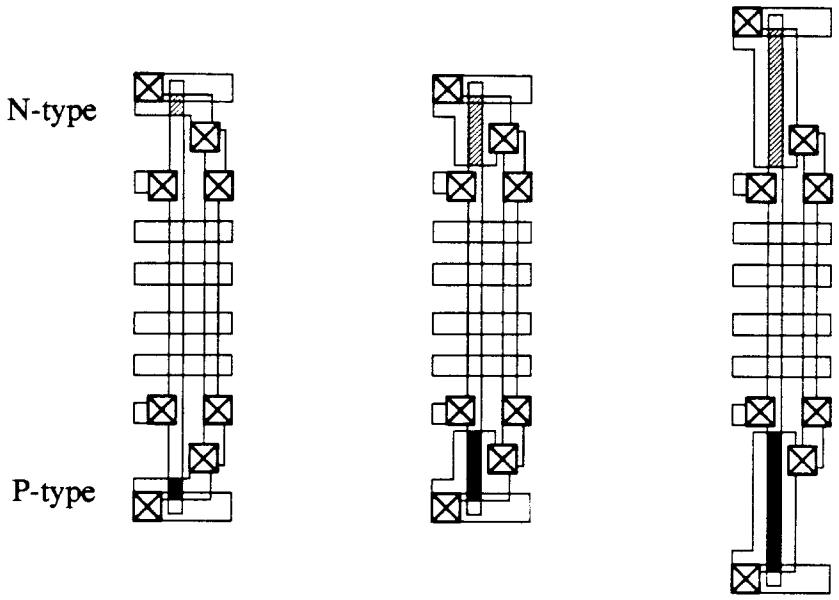
For CMOS, the small static power consumption is due to reverse bias leakage between the diffusion regions and substrate. The power due to leakage current can be described by the diode equation. The maximum dynamic power required by a circuit is directly related to the capacitance it drives. Since the power to drive the input nodes comes from external circuitry, the maximum dynamic power a circuit consumes is related to the internal and output load capacitance (excluding the supply nodes), whose total will be represented the variable, C_{driven}. For the CMOS example, the total power will be approximated by the maximum dynamic power component. The following equations result:

$$power_{CMOS} \approx power_{dynamic} = C_{driven}\ V_{dd}^2\ frequency_{\max};$$

$$C_{driven} = C_{n1} + C_{n2} + C_{n3} + C_{n4} + C_{n5};$$

2.3. Area

There are many possible choices for representing the area design metric. A VLSI designer usually needs to generate a layout to fit within a given space in an existing design. Alternatively the aspect ratio (ratio of height to width) of a circuit must be maintained so that it can be placed in a realistic package. Therefore, an appropriate area metric for an optimization program should include cell height and width.



Transistor width	3 (min)	10 ($w=w_{free}$)	20 λ
Total Transistor Area	12	40	80 λ^2
Total Cell Area	896	896	1176 λ^2

Figure 2.3. **Layout effects of free transistor width.** The effect of transistor widths on total cell height and width are illustrated by layout fragments of a static CMOS standard-cell inverter. The layout on the left contains transistors of minimum width (3 λ, 2.1μm). As long as the transistor width falls below the *free* limit, the layout area will not change (center layout fragment). The layout on the right illustrates the additional cell area required if the transistors increase well above their *free* limit.

Most transistor sizing tools (e.g.: [Shyu88b], MOST, TILOS, TV/AI, XTRAS) do not model the total cell height and width. Instead, they use total active transistor area, which is actually a component of dynamic power rather than a measure of total cell area. The chapter on optimization techniques will show the difference in area, delay, and power resulting from a change in the objective function. Typically this can be as much as 32% for even the simple eight inverter chain.

Another consequence of choosing a realistic area metric is that there is no longer any need to artificially constrain the maximum size on every transistor as in AESOP. Instead, transistors take on the appropriate size necessary to meet the design constraints.

Cell height and width are not determined simply by transistor and routing dimensions alone. The overlap between routing area and total transistor area, significantly impacts the overall area and performance. Therefore, an important element in characterizing layout is how well the area under the routing channel is utilized before the overall dimensions must change. This factor is identified by the *free* transistor width, the width that a transistor can grow without affecting the area of the cell. If a transistor width falls within the minimum and *free* width, then the

overall cell width will not change. The *free* transistor width is determined by local routing and connections.

In a structured design environment such as standard cells, if one transistor has exceeded its *free* width, then all transistors in the same row (a track) also benefit from the additional width. This simple factor can rapidly relate transistor size changes to cell height and width changes. EPOXY considers splitting of large transistors into smaller ones [Hill87] in the restructuring section, since width/height tradeoffs are best made once these large transistors are identified by transistor sizing. *Figure 2.3* illustrates the effects of *free* transistor width on overall cell area.

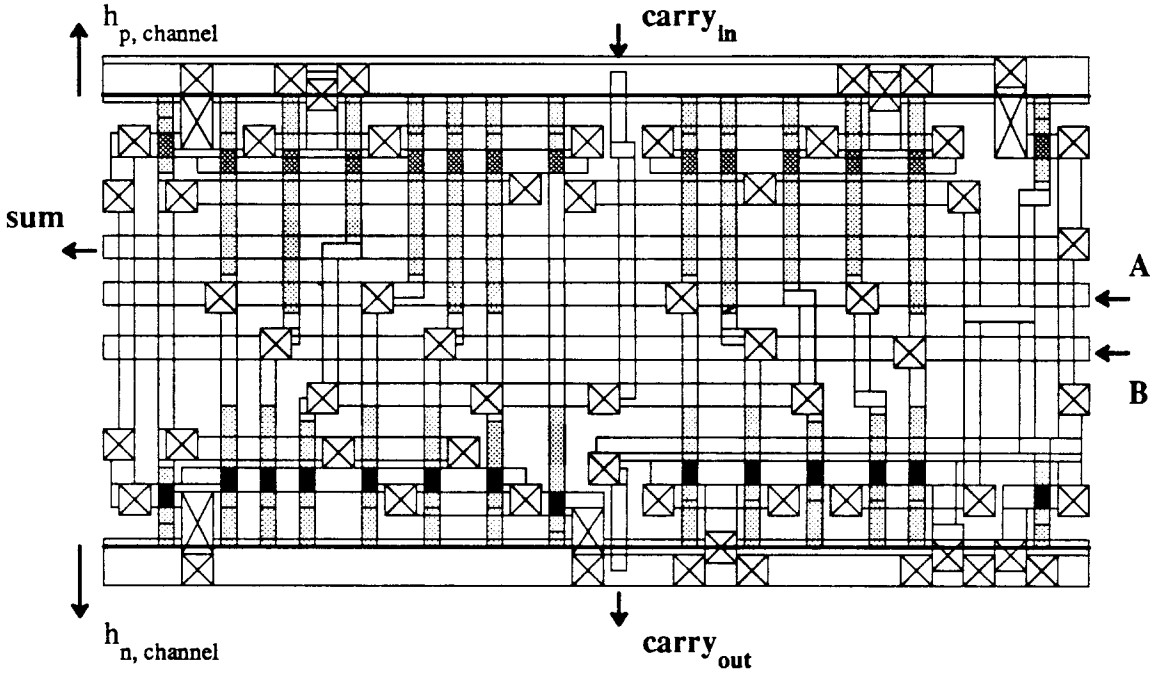


Figure 2.4. **Free transistor widths for a CMOS adder.** The rows of p- and n- type transistors are highlighted in the standard-cell CMOS layout for a static full adder. The inputs to this full adder are **A**, **B**, and **carry**$_{in}$; the outputs are **sum** and **carry**$_{out}$. The free width for each transistor is described by the lighter shading around the transistor. If any transistor exceeds its free width, the track must be enlarged to accommodate it.

Given a circuit architectural style, total height and width can be used as the area metric rather than the simple sum of transistor area. The virtual grid area model takes into account routing effects on each transistor by determining its *free* transistor width. Since typical standard-cell layout styles employ columns of n- and p-channel transistors, the formulation of the area equations is straightforward. *Figure 2.4* illustrates a standard-cell implementation of a static CMOS full adder. The format of the area equations for a standard-cell layout are listed below. The last constraint shows that the sizing of power busses to provide sufficient current can be incorporated into this model.

$$area = height \; width;$$

$$width = \max w_{channels}; \qquad w_{channel} = w_{interconnect} + \sum l_{fet};$$

$$height = \sum h_{channels}; \qquad h_{channel} = h_{routing,channel} + h_{n,channel} + h_{p,channel};$$

$$h_{type,channel} \geq w_{fet} - w_{free,fet}; \qquad h_{type,channel} \geq 0;$$

$$h_{type,channel} \geq h_{power\ buss,channel} - h_{power\ buss,\min};$$

COP [Marp87] estimates total area for standard cells by adding the widths of the maximum p- and n- channel transistors to the height of the routing channel. While this is a measure of total cell area, it fails to consider the space available to larger transistors under the particular routing channel.

The area model based on free transistor width is general enough to accurately model other layout styles. In essence, this approach results in a virtual grid expansion technique for detailed layout. First, free widths are identified for each transistor. Next, transistors are grouped into tracks (virtual grid lines). When a transistor exceeds its free width, the layout is expanded along the track line (the virtual grid line is moved). Transistors that share a track must have the same orientation and contain layout (routing) which can be expanded perpendicular to the track.

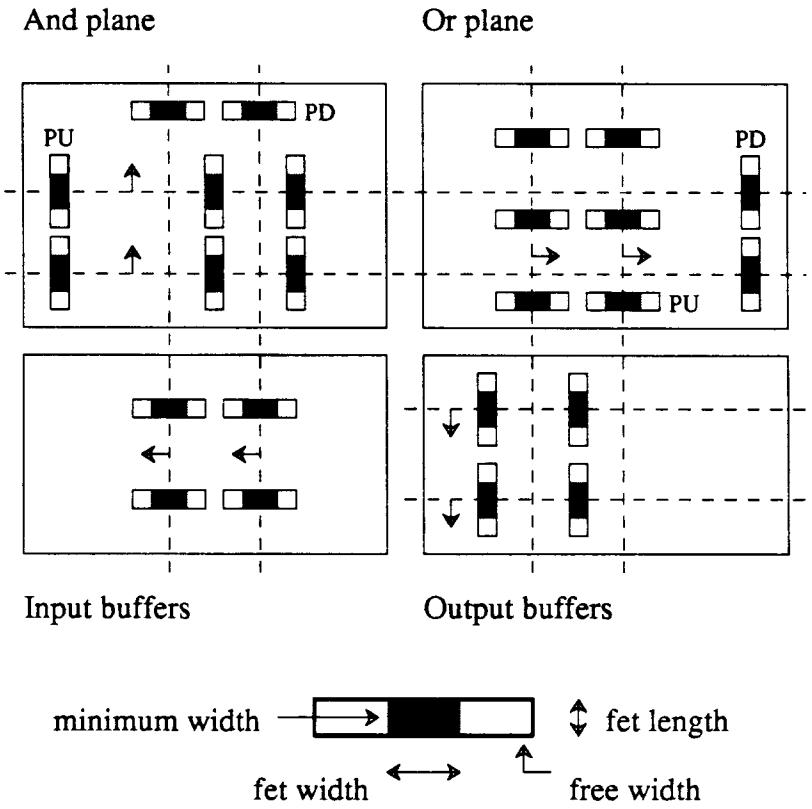


Figure 2.5. **Virtual grid model for a generic PLA.** Transistors and their free widths are identified in a PLA. If a transistor exceeds its free width, the design is widened along the corresponding dashed track line in the direction of the arrow. All transistors that share the track (virtual grid line) have additional space in which to expand. The pull-up (PU) transistors and the pull-down (PD) transistors of dynamic designs may be omitted in static PLA's.

PLATO [Marp86] accurately models the height and width of PLAs. Similarly, EPOXY's general virtual grid based area model also accurately models the height and width of PLAs, as *Figure 2.5* illustrates. Changes in width of a transistor are reflected in the overall PLA area if the transistor dominates its virtual grid track. Unlike standard cells, the virtual grid lines for PLA's will intersect, since transistors assume different orientations. Once the virtual grid lines are place, the computation of overall PLA area is similar to that for standard cells. The only difference is that transistor widths may effect the PLA cell height and width. PLATO also considers snaking of transistors in the AND and OR planes. The free widths for transistors and track information can be provided by the PLA template as in MPLA [Scot85].

2.4. Noise Margins

Digital circuits require immunity from noise in order to produce logic results. Noise margins specify the allowable noise voltage on the input so that acceptable logic voltage values appear on all outputs. The noise margin for a gate is directly related the ratio of the pull up resistance to the pull down resistance [West85]. Since changes in transistor size effect this ratio, a designer may wish to restrict the ratio to assure an acceptable noise margin. CMOS circuits are less sensitive to variations in this ratio than are nMOS circuits.

$$NM_{min} \leq \frac{R_{pull\ up}}{R_{pull\ down}} \leq NM_{max};$$

2.5. Conclusions

This chapter outlines models for determining total cell area, height, width, maximum power dissipation and worst-case delay. These models are represented by a set of equations so that they can be easily substituted in order to evaluate their effect on the optimized results. EPOXY currently represents these models by creating equations using the following templates:

Distributed linear RC delay model:

$$R_{type,fet} = Kr_{type}\left[\frac{l_{fet}}{w_{fet}}\right]; \qquad C_{gate,fet} = Kg_{type}\, w_{fet}\, l_{fet};$$

$$C_{sd,fet} = Ksd_{type}\, w_{fet};$$

$$C_{node} = C_{parasitic} + \sum_{attachedfets} C_{gate,fet} + \sum_{attachedfets} C_{sd,fet};$$

$$delay() = \sum_{paths} R_{fet} C_{node};$$

$$t_{output\ node} = \max(t_{input\ node} + delay())$$

Maximum dynamic power:

$$C_{driven} = \sum_{internalnodes} C_{node};$$

$$power_{dynamic} = C_{driven}\, V_{dd}^2\, frequency_{\max};$$

Total area, cell height and width depend on the transistor orientations:

$$area = height\ width;$$

$$width = \max w_{channels}; \qquad w_{channel} = w_{interconnect} + \sum l_{fet};$$

$$height = \sum h_{channels}; \qquad h_{channel} = h_{routing,channel} + h_{n,channel} + h_{p,channel};$$

$$h_{type,channel} \geq w_{fet} - w_{free,fet}; \qquad h_{type,channel} \geq 0;$$

This chapter considers and analyzes several alternative models for representing the design metrics. The key concepts from this chapter are:

1. Accurate timing models are crucial in performance optimization, since a small error in meeting a time constraint (16%) can typically result in much larger area (100% more area) and power dissipation (50%) than necessary.
2. The sum of transistor areas is not an appropriate measure of total cell area. Rather it is a component of dynamic power. The optimization of an eight-stage inverter chain shows that the minimum power implementation is 32% larger than the one for minimum area for a 10ns time constraint.
3. A virtual grid area model accurately represents the cell height, width and area by considering the detailed routing using free transistor width. Two examples demonstrate how the equations are easily derived from standard cell and PLA layouts.
4. The optimization problem is uniformly formulated using equations. The flexibility of EPOXY is demonstrated by allowing the user to specify an objective function and performance constraints so that the user can pursue a layout with the desired performance tradeoffs. From this information, a performance envelope that describes the full range of performance implementations of interest can be derived.

3. Optimization and Heuristic Techniques

Systems that improve a design's performance primarily rely on either optimization-based or heuristic techniques. Optimization-based algorithms make use of gradient information (partial derivatives) to indicate specific changes in the input parameters that lead to better values for the objective function. However, the constraint equations limit the changes in the input parameters so that the constraints are satisfied. On the other hand, heuristic techniques are based on designer experience that certain changes usually improve the design.

Ideally, a user could select the most appropriate heuristic or optimization technique for a design problem based on extensive analysis. However, most transistor sizing programs tightly couple an electrical model with a fixed optimization algorithm to improve the overall execution speed. By representing the performance by a set of equations, EPOXY allows electrical models and optimization techniques to be substituted, yet retains the rapid execution comparable to integrated systems. *Chapter 5* explains the advantages of the equation representation within EPOXY. This chapter describes and compares the results of a few optimization and heuristic techniques.

The performance design problem can be formulated as a classical optimization program:

$$\textbf{min } f(x) \quad \textbf{subject to } g(x) \leq 0$$

The objective function, $f(x)$, defines the quality of a configuration represented by the independent vector of variables, x. For transistor sizing problems, a design is specified by the assigned transistors widths, x. The VLSI design quality is usually described by the area, delay, or power consumed by the circuit. The constraint functions, $g(x)$, define requirements on the input variables such as minimum transistor width, overall maximum cell height and width, maximum dynamic power and delay constraints.

When the design problem is cast as a non-linear program, the performance models provide the equations and constraints that determine the overall design metrics. EPOXY automatically generates these core set of equations from the circuit layout. The non-linear program is complete when the user defines an objective function and performance constraints.

The task of an optimization routine is to determine the best feasible implementation, as described by an objective function, that meets all of the performance constraints. The performance optimization problem involves non-linear equations, since delay is a non-linear function of transistor width (a total cell area metric) even for a simple lumped RC electrical model. This non-linear relationship will appear in at least one of the constraint equations. In general, non-linear programs (NLP) are difficult to solve because of many potential local minima.

To simplify the design rules, lambda-based designs [Mead80] restrict transistor sizes to integer multiples of lambda. However, this arbitrary limitation results in an integer non-linear program that is much more difficult to solve than the original NLP. Therefore, as in most transistor sizing programs, the transistors widths are not restricted to integer multiples of lambda. A user can appropriately round the

transistor widths to achieve the desired integer values. Although an integer-valued optimization cannot be solved by rounding the corresponding real-valued solution, a design with performance similar to the real-valued solution can be produced by rounding the transistor sizes.

The following subsection will describe the design space for a CMOS example to illustrate the performance tradeoffs. Next, the augmented Lagrangian optimization technique and several heuristics will be discussed. The augmented Lagrangian algorithm has been implemented within EPOXY since it represents one of the classic optimization techniques whose convergence rate to a local optimum can be derived. The TILOS-style heuristic will be examined in greater detail since it rapidly sizes transistors subject to timing constraints. Then the results of several optimization problems are compared using the augmented Lagrangian optimization technique and TILOS-style heuristic.

3.1. Performance Envelope

Since a VLSI designer is interested in balancing several performance metrics, the non-linear program results in a multi-dimensional problem. Unfortunately, it is difficult to visualize multi-dimensional graphs or to extract numerical information from them. Therefore, projections of the multi-dimensional boundary onto area-delay and power-delay graphs bound the region within which lay all designs of interest. This region, which we call the *performance envelope*, is illustrated in *Figure 3.1* for a standard-cell layout of an eight-stage CMOS inverter chain. These curves were generated by substituting a variety of objective functions and constraints within EPOXY.

The graph on the left of *Figure 3.1* shows the area required to implement a standard-cell static CMOS eight inverter chain to meet a given time constraint. The **X** on the upper left part of the graph marks the performance of the fastest (minimum time) implementation regardless of power consumption and area. The lower right **X** indicates the performance of the implementation with minimum device sizes. This configuration needs the smallest area and power. Since the transistor widths can assume any non-integer value, these two points are connected by continuous curves that give the minimum area and minimum dynamic power implementations for a given time constraint. For the area versus time graph, there exists no circuit smaller than the minimum area implementation that meets the time constraint (the region *below* the minimum *area* curve). Circuits that draw more than the minimum power implementation would not be of interest (the region *above* the minimum *power* curve). Therefore, the *only* circuits of interest lie between the two curves as illustrated by the shaded region. The maximum dynamic power versus time graph describes a similar situation. Circuits between the two curves represent a tradeoff between the requirements of area, power, and time.

These graphs clearly show the effects of the more accurate area model. As the circuit delay decreases, the required area does not change above the minimum size implementation (the flat portion of the area versus time graph for the minimum *area* configurations) until at least one transistor exceeds its *free width*. At this

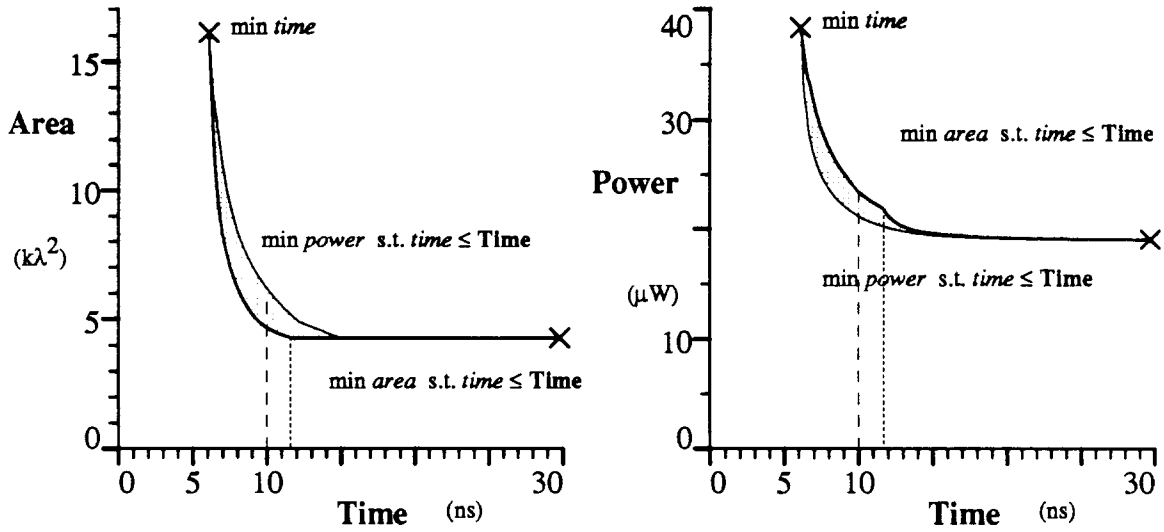


Figure 3.1. **Typical performance envelope.** These graphs illustrate the area required (left graph) and dynamic power required (right graph) to implement the minimum dynamic power (*min power*) and minimum area (*min area*) configurations of an eight-stage static CMOS inverter chain to achieve a specific delay (**Time**). Only the region between these curves represents the designs of interest to a VLSI designer. The dashed line highlights the range of performance for designs that operate at 10ns. The dotted line indicates the fastest design that still requires the smallest cell area. The output has an additional 1 pf capacitance load.

point (as highlighted by the dotted line at 11.8ns), the other transistors in the circuit can assume sizes that still meet the required response time, but which draw less power. Therefore, the curve for the power required by the minimum area implementations changes slope on the power versus time graph at 11.8ns. Thereafter, additional area is required for circuits with less delay.

Minimum	Constraint	Area λ^2	(μm^2)	Power μW
power	time ≤ 10ns	6188.	3032.	21.22
area	time ≤ 10ns	4677.	2292.	23.48

Table 3.1. **Comparison of an area and power optimization.** This table gives the area and maximum dynamic power required (column headers) to implement the minimum area and minimum power configurations (row headers) of an eight inverter chain. Both optimizations are subject to a 10ns response time constraint. *Figure 3.2* illustrates the layout for these two designs.

Table 3.1 contains data extracted from the graphs of *Figure 3.1.* For a 10ns implementation (as marked by the dashed line in *Figure 3.1*), the minimum power implementation is 32% larger than the minimum area implementation. Total transistor area is closely related to the maximum dynamic power. Therefore if one uses total transistor area (a power metric) to *approximate* total cell area, the resulting solution may be as much as 32% larger than necessary since the minimum

power implementation is 32.3% larger than the minimum area design.

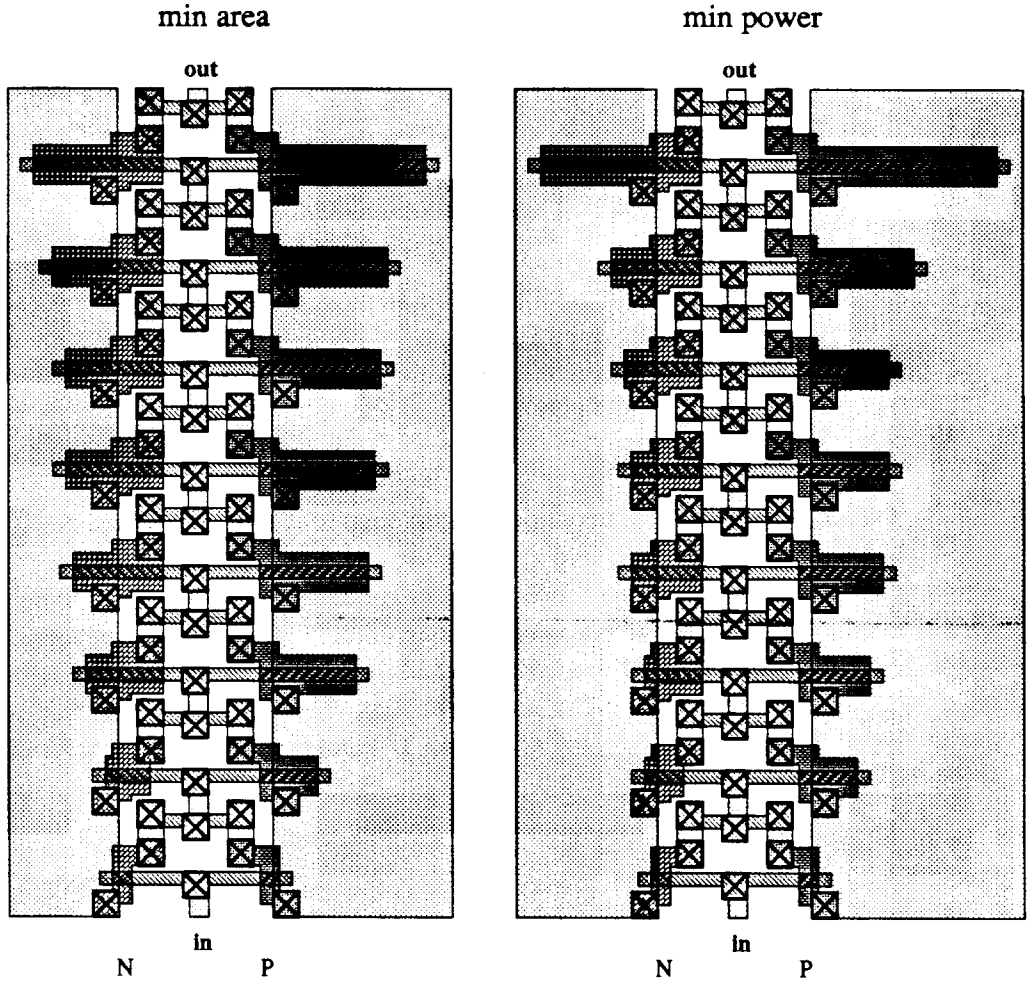


Figure 3.2. **Layouts for an eight stage inverter chain.** This figure illustrates the standard-cell layouts for an eight-stage static CMOS inverter chain. The left design requires the minimum cell area to meet a 10ns delay constraint while the right design requires the least dynamic power to meet this delay constraint. The input is applied at the bottom of each design at the node marked **in**. A 10pf load is added to the output node at the top of each design marked **out**.

Figure 3.2 contains the layouts for the minimum area and minimum power implementations subject to the 10ns delay constraint. The eight stage inverter takes its input at the bottom at the node marked **in** and produces the output at the top as marked by the node **out**. The first stage is fixed to minimum sized devices so that the loading on the previous stage is constant. The remaining transistors are sized appropriately. Note that the overall cell width for the minimum power implementation is larger than the minimum area design since the minimum power configuration uses larger transistors in the last stages located at the top of the layout. However, the minimum area design uses larger transistors in the middle of the design than the minimum power implementation.

Graphs of the performance envelope were generated by varying the constraint values and then solving the optimization problem. However, only a small portion of the graph is of interest to VLSI designers. Typically, designers must balance the area and power requirements for a given operating speed. This involves solving a minimum power and a minimum area constraint problem to determine the maximum performance limits. Then a designer can formulate an appropriate objective function that encodes the tradeoff among the performance metrics. EPOXY facilitates this process of exploration that is crucial to a designer in understanding the incremental tradeoffs on each design constraint so that an appropriate objective function can be developed and, in turn, a good implementation realized.

EPOXY's flexible architecture allows a comparison of different optimization techniques, independent of performance models. An augmented Lagrangian algorithm similar the one used in COP [Marp87] and a TILOS-style heuristic technique were implemented within EPOXY. These represent different approaches to solving the NLP. The augmented Lagrangian algorithm is based on classic optimization techniques with known convergence properties. However, the TILOS-style heuristic converges rapidly for many typical VLSI design problems. These two techniques will be described and compared in the following subsections. The next subsection examines two step size techniques that have a significant influence on the overall execution speed of the augmented Lagrangian algorithm.

3.2. Step Size Algorithms

The basic format of many optimization-based algorithms involves determining a suitable search direction vector and then computing an appropriate step length. The partial derivative information is used to calculate the search direction. Then the input vector is updated by traversing this search direction vector by an amount as indicated by the step length. Therefore, the step length calculation plays a significant role in the running time of most optimization-based techniques.

For a minimization problem, the step length minimizes the function value along the search direction. COP [Marp87] uses the Golden section method to compute the step length for the augmented Lagrangian method. The Golden section method derives its name from the Golden ratio known to the early Greeks for solving a classical problem of dividing line segments or rectangles in a particular way:

$$\tau = \frac{\sqrt{5}-1}{2} \approx 0.6180339887$$

The Golden section method can be used to find the minimum of a unimodal function as shown in *Figure 3.3*. A unimodal function has only one local minimum over the interval of interest [L, R]. The method begins by determining two points, l and h, within the interval. The search interval is then restricted to the point whose functional value is largest. Only one functional evaluation is required at each iteration. In the example of *Figure 3.3*, the point r' is set to the value of the previous l. The next iteration must only evaluate the function at the new l' point. The following pseudo code in *Figure 3.4* describes the basic steps in the Golden section method.

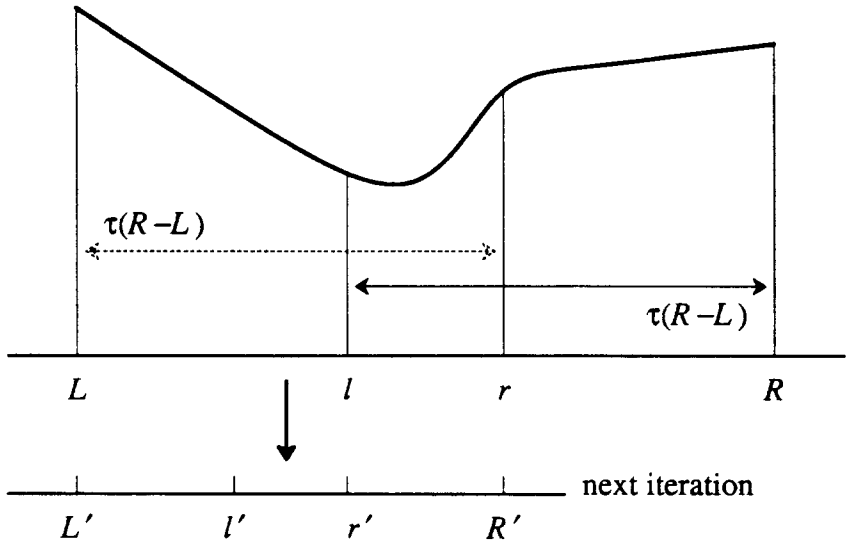


Figure 3.3. **Example of the Golden section method.** This figure illustrates how the Golden section method subdivides an interval, [*L*, *R*], to located the minimum of a function. Since the functional value at *l* is lower than at *r*, the search interval is restricted to [*L*, *r*]. This process continues until the interval is less than the prescribed tolerance.

```
golden_section (L, R) {
    l = 0.381966*L + 0.618034*R;     fl = f(l);
    r = 0.618034*L + 0.381966*R;     fr = f(r);
    while (R–L >= error) {
        if (fl <= fr) {
            R=r; r=l; rf=rl;
            l = 0.381966*L + 0.618034*R;     fl = f(l);
            step = l;
        } else {
            L=l; l=r; rl=rf;
            r = 0.618034*L + 0.381966*R;     fr = f(r);
            step = r;
        }
    }
    return (step);
}
```

Figure 3.4. **Pseudo code for Golden section method.** This figure shows the pseudo code for implementing a Golden section minimization of a function, f(x). The Golden ratio is represented by 0.618034 in this code. The 0.381966 figure is the square of the Golden ratio. Note that the Golden ratio and its square total to 1.

Since the Golden section method subdivides the interval by a fixed factor, this method converges linearly to the overall minimum of a unimodal function with a convergence ratio equal to the Golden ratio [Luen84]. This techniques makes constant but slow progress toward a minimum value because of the large number of functional evaluations. The Golden section method is also quite robust. However, it does not take advantage of any smoothness that a function may possess to

improve the rate of convergence.

Another step size technique based on the Armijo rule only requires one gradient evaluation. The following equation describes the computation to determine the step size, σ.

$$\sigma = \beta^k = \underset{k \in N}{\operatorname{argmax}} \left\{ \beta^k \mid f(x) - f(x - \beta^k \nabla f(x)) \geq -\beta^k \alpha \| \nabla f(x) \|^2 \right\}$$

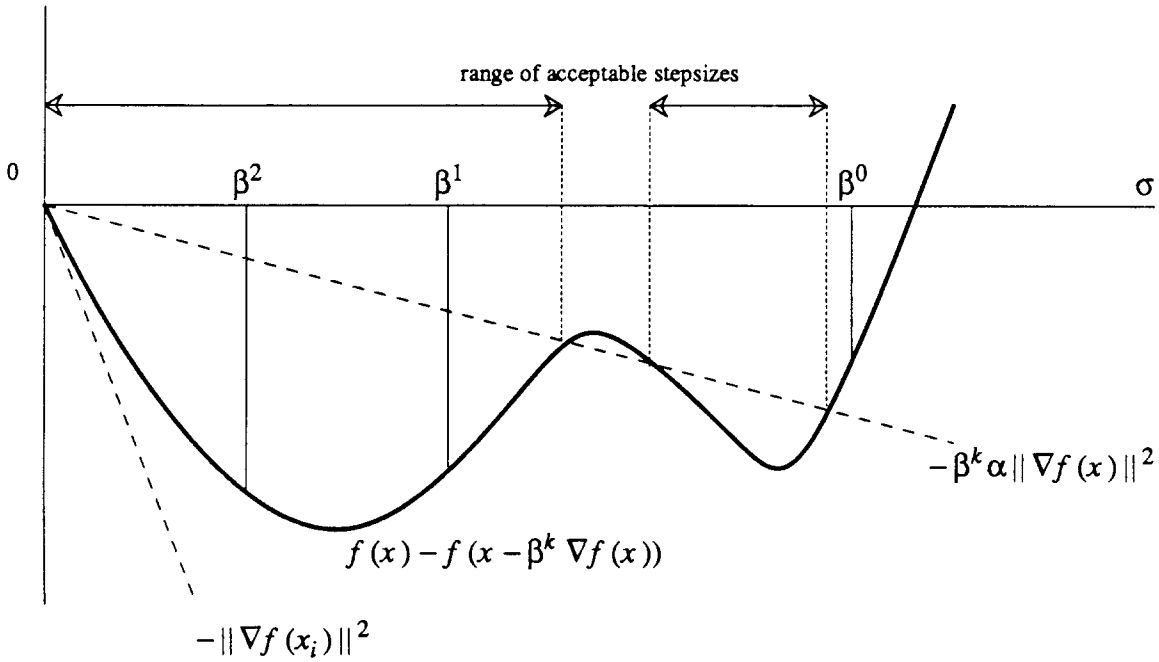


Figure 3.5. **Example of the Armijo method.** This figure illustrates how the Armijo method selects the appropriate step size. The function describing the left hand side (*lhs*) of the Armijo inequality is graphed against various step sizes $\sigma = \beta^k$. Starting with k=0 (β^0), successively larger integer values of k (β^k) are evaluated until the function value falls below $-\beta^k \alpha \| \nabla f(x) \|^2$ as the dashed line illustrates. The derivative of the *lhs* function is evaluated at a step size of 0 as indicated by the lower dashed line. Since all of the *lhs* values lay above this line, a value of k that satisfies the Armijo inequality must exist.

The process by which the step size, σ, is determined is shown in *Figure 3.5*. The method initially starts with a step size of 1 (k=0) and continues with successively larger integer k values until the inequality is satisfied. For subsequent evaluations, the previous value of k is retained as a starting point. If the this value does not satisfy the constraints, k is updated as usual. However, if k all ready satisfys the constraints, k is decremented. Usually α is chosen close to zero, for example, 0.3. The scalar β is usually chosen from .5 to 0.1 depending on the quality of the initial step size. For example, *Figure 3.5* illustrates the acceptable range of step sizes. The Armijo method will select β^1 as the first value that satisfies the inequality. An implementation of the Armijo method is described in *Figure 3.6*.

```
int    k = 0; /* Initial starting point, first time through.*/
double     b = 1.0, beta = 0.5;     /* b = beta^k */
double     alpha = 0.3;

armijo (x) {
      df = derivative(x);
      obj_base = try_step (0.0);   /* f(x) */

      rhs = -1.0 * alpha * b * norm(df);
      lhs = try_step (b) - obj_base;      /* f(x - b*df) - f(x)*/

      if (lhs > rhs) {
            while (lhs > rhs) {       /* While not satisfied, increase k */
                  k++; b *= beta; rsh *= beta;
                  lhs = try_step (b) - obj_base;
            }
      } else {
            while ((lhs <= rhs) && (k >= 0)) {/* While sat., decrease k */
                  k—; b /= beta; rhs /= beta;
                  lhs = try_step (b) - obj_base;
            }
            k++; /* Keep the last valid stepsize.*/
            b *= beta;
      }
      return (b);
}
```

Figure 3.6. **Pseudo code for the Armijo method.** This figure describes one possible implementation of the Armijo step size algorithm. Several of the variables in this example are vectors: e.g., x, df. Therefore, the try_step() subroutine performs the vector calculation necessary to evaluate the objective function at the new x value. Since only successive powers of beta are required, the intermediate variable b is updated. The actual code restores the updated b value when k=0 or k=1 to avoid excessive round-off errors.

The Armijo method performs an inexact line search since it restricts its step size to discrete values. However, when combined with a minimization routine, the Armijo method typically minimizes a function faster than the Golden section method. The Armijo method converges faster than linear (superlinearly) with a convergence ratio of β:

$$\left[\frac{M-m}{M+m}\right]^2 \leq \beta$$

where M and m are the largest and smallest eigenvalues of the Hessian, respectively [Bert82]. The following example illustrates the advantages of the Armijo method.

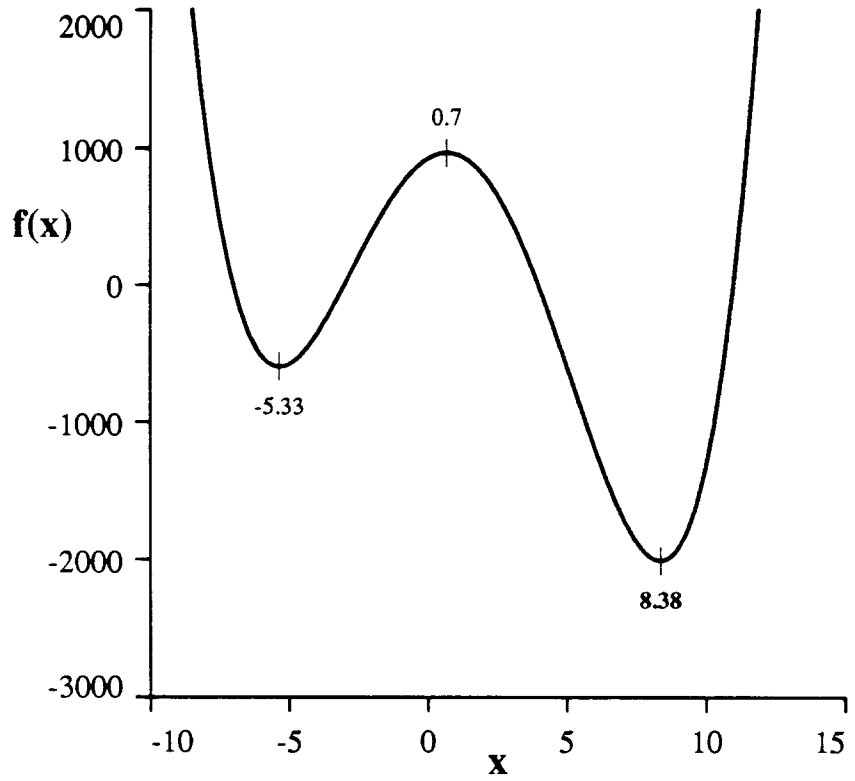


Figure 3.7. **Example function for testing step size techniques.** The example function, $f(x) = (x+7)(x+3)(x-4)(x-11)$ contains a global minimum at $x = 8.38$ and a local minima at $x = -5.33$. The extreme points are labeled on the graph.

Figure 3.7 describes a function with two minima, a global minimum at $x = 8.38$ and a local minima at $x = -5.33$. The following paragraphs will compare the performance of each step size algorithm for the steepest descent algorithm. The method of steepest descent is one of the oldest and most widely known techniques for minimizing a function of several variables. The following equations describe the method of steepest descent in mathematical terms [Luen84].

$$\min_x f(x): \qquad d(x_k) = \nabla f(x_k)^T \qquad x_{k+1} = x_k - \sigma_k d(x_k)$$

The minimization of a function, $f(x)$, with continuous first partial derivatives is performed by iteratively updating x until the magnitude of search direction, $d(x_k)$, is nearly zero. The step length, σ_k, is the nonnegative scalar minimizing $f(x_k - \sigma_k d(x_k))$. In other words, from the point x_k, the step size algorithm searches along the direction of the negative gradient $-d(x_k)$ to a minimum point on this line; the minimum point is taken to be x_{k+1}. Since the example function has only one input variable, the search direction is simply the negative of the derivative of the function. *Figure 3.8* describes the pseudo code for the steepest descent algorithm for minimizing the example function.

When each of the step size techniques were substituted, this program produced the information in *Table 3.2*. Since the function contains two minima, two final x

```
steepest_descent (x) {
      max_direction_error = 0.025;

      direction = – derivative();
      /* While we're not close to a local minima */
      while (fabs(d) > max_direction_error) {
            stepsize = linesearch(x, direction);
            x = x + stepsize * d;
            direction = – derivative();
      }
      return (x);
}

function () {
      function_evaluations ++;
      return ((x+7.0)*(x+3.0)*(x–4.0)*(x–11.0));
}

derivative () {
      derivative_evaluations ++;
      return (4.0*x*x*x – 15.0*x*x – 170.0*x + 125.0);
}
```

Figure 3.8. **Pseudo code for a steepest descent algorithm.** This pseudo code executes a steepest descent minimization of the function $f(x) = (x+7)(x+3)(x-4)(x-11)$. The number of function and derivative evaluation are recorded to compare various step size techniques as implemented by the linesearch function.

values may be produced depending on the starting point and the accuracy of the intermediate step sizes. When the Golden section and Armijo routines produce similar final x values, the number of function and derivative evaluations are averaged as listed in *Table 3.2*. The Armijo method requires less than half the number of function calls than the Golden section technique. Even though the Armijo routine requires a slightly large number of derivative evaluations, the large savings in function evaluations far outweighs these additional derivative calculations.

	Golden section			Armijo		
starting x	final x	func. calls	deriv. eval.	final x	func. calls	deriv. eval.
−10.00	−5.32784	52	5	−5.32781	38	10
−9.00	**8.37772**	91	8	−5.32781	38	10
−8.00	−5.32787	39	4	−5.32789	33	9
−7.00	**8.37772**	78	7	−5.32782	33	9
−6.00	−5.32784	52	5	−5.32782	33	9
−5.00	−5.32784	39	4	−5.32789	28	8
−4.00	**8.37769**	52	5	−5.32778	28	8
−3.00	**8.37768**	78	7	−5.32781	35	10
−2.00	−5.32784	52	5	−5.32789	30	9
−1.00	**8.37774**	78	7	−5.32788	24	7
0.00	−5.32784	39	4	−5.32787	33	10
1.00	**8.37771**	65	6	**8.37771**	28	8
2.00	**8.37775**	65	6	**8.37768**	25	7
3.00	−5.32785	52	5	**8.37768**	22	6
4.00	**8.37775**	52	5	**8.37769**	25	7
5.00	−5.32786	65	6	**8.37772**	28	8
6.00	**8.37770**	78	7	**8.37770**	25	7
7.00	**8.37772**	78	7	**8.37777**	25	7
8.00	−5.32784	52	5	**8.37771**	26	7
9.00	**8.37772**	78	7	**8.37777**	23	6
10.00	−5.32784	52	5	**8.37769**	26	7
11.00	−5.32783	65	6	**8.37777**	30	8
12.00	**8.37772**	65	6	**8.37775**	28	7
13.00	−5.32784	78	7	**8.37777**	28	7
14.00	**8.37776**	91	8	**8.37771**	25	6
Average		60.4	5.6		28.5	7.9

Table 3.2. **Golden section and Armijo step size routines.** The performance of the Golden section and Armijo step size routines are compared for the steepest descent minimization of $f(x) = (x+7)(x+3)(x-4)(x-11)$. Since this function has two minima, two possible final x values are produced when starting from any x value. The number of function and derivative evaluations are given when each of the step size routines were substituted. The last entry in the table provides the average number of function and derivative evaluations when both step size routines produce the same approximate final value for x.

3.3. Augmented Lagrangian Method

Augmented Lagrangian methods convert a constrained minimization problem into an unconstrained sub-problem [Gill81]. The augmented Lagrangian algorithm handles non-linear inequality constraints by smoothing the discontinuous derivatives through the function, $L_A(x,\lambda,\rho)$. The following equations describe the augmented Lagrangian algorithm as implemented within EPOXY:

$$\min f(x) \;\; subject\ to\ g_i(x) \le 0 \quad \equiv \quad \max_{\lambda} \min_{x} L_A(x,\lambda,\rho)$$

$$L_A(x,\lambda,\rho) = f(x) + \sum_i \begin{cases} \lambda_i\, g_i(x) + \dfrac{\rho\, g_i(x)^2}{2}, & \text{if } \lambda_i + \rho\, g_i(x) > 0; \\ -\dfrac{\lambda_i^2}{2\rho}, & \text{if } \lambda_i + \rho\, g_i(x) \le 0; \end{cases}$$

$$d_j = \frac{\partial f(x)}{\partial x_j} + \sum_j \begin{cases} \dfrac{\partial g_j(x)}{\partial x_j}\left[\lambda_i + \rho g_i(x)\right], & \text{if } \lambda_i + \rho g_i(x) > 0; \\ 0, & \text{if } \lambda_i + \rho g_i(x) \le 0; \end{cases}$$

$$x_j^{k+1} = x_j^k - \sigma\, d_j\ ;$$

$$\lambda_i^{v+1} = \begin{cases} \lambda_i^v + \rho\, g_i(x), & \text{if } \lambda_i^v + \rho\, g_i(x) > 0; \\ 0, & \text{if } \lambda_i^v + \rho\, g_i(x) \le 0; \end{cases}$$

Note that the function, $L_A(x,\lambda,\rho)$, and its gradient (as represented by the search direction, d_j), are continuous at any point where a constraint changes from *active* ($\lambda_i + \rho\, g_i(x) > 0$) to *inactive* ($\lambda_i + \rho\, g_i(x) \le 0$). The computation of the search direction involves evaluating the partial derivative equations. A matrix of these partial derivatives is called a Jacobian matrix. Associated with each constraint equation, $g_i(x)$, is an extended Lagrangian multiplier, λ_i. The parameter ρ is a penalty factor.

The inner loop, the minimization of $L_A(x,\lambda,\rho)$ with respect to x, is performed by an iterative line search that follows the steepest descent direction, d_j. PLATO [Marp86b], a PLA transistor sizing program, and COP [Marp86a][Marp87], a CMOS standard-cell transistor sizing tool, employ an Golden section line search. However, the Armijo-based line search [Luen84] provides a better convergence rate than the Golden section method as described in the previous subsection. EPOXY provides both of these methods. The input vector, x is updated after each minimization iteration, k.

As in PLATO and COP, if the search direction, d_j, is too small (nearly a zero vector) or the traversal in the search direction, σ, is too small, an outer iteration, the maximization of $L_A(x, \lambda, \rho)$, is performed by updating λ (iteration count v). If the rate at which the constraints become satisfied is too slow, the penalty factor, ρ, is slightly increased. Similarly, if the average value of λ is growing too quickly, ρ, is slightly decreased.

The augmented Lagrangian method converges reasonably fast (linearly) to an optimum solution (if one exists) [Marp86a]. As with all optimization-based algorithms, its primary advantage is that progress toward an optimum solution can be guaranteed regardless of the initial solution. Although there are newer optimization-based algorithms that may converge faster such as recursive quadratic programming and method of feasible directions [Shyu88a] [Shyu88b], these techniques should be examined in context of VLSI design problems to evaluate their effectiveness.

3.4. Heuristic Improvement Techniques

The minimization problem can be solved by another class of techniques called heuristics. Heuristics encode design expertise for specific optimization problems. For example, "if a transistor exceeds its free width, then split the largest transistor into two smaller ones to reduce the overall cell area." Convergence to an optimal solution cannot be assured; however, they provide rapid improvement toward a potentially better implementation. Therefore, heuristics can be used to rapidly locate a feasible solution. Since heuristics are generally problem specific, the types of objective functions permitted would also be restricted.

Many of the heuristics are based on critical delay path information. A critical delay path is the series of nodes and transistors along the slowest path or paths through the circuit. These are the only transistors that may improve the slowest path. Therefore, many of the heuristics that improve the delay of the circuit concentrate on the transistors and nodes along critical delay path.

A description of several performance improvement heuristics used in other systems are listed in *Table 3.3*. These techniques usually help meet difficult timing constraints by reducing the delay through a circuit.

ANDY improves the delay of a circuit in two passes. The first pass assigns transistor sizes using a ramped driver heuristic. This heuristic is based on the results of designing drivers for large loads. There is an optimal scale by which each stage is larger than the previous one to produce the fastest driver. A gate can be sized only when its load, including the inputs of other gates, is known. Therefore, the output gate is sized first since its output load is fixed. The remaining gates are sized from the outputs to the inputs. ANDY's second pass reduces the power of the circuit while maintain its speed by resizing the gates that do not lie on the worst critical delay paths. The gates are slowed by reducing the transistor widths which also reduces the power consumption for the gate. ANDY runs quite fast since it uses a simple gate level model; however, it produces sub-optimal results since the derivation of the optimal scale factor cannot take into account

System	Heuristic
ANDY [Trim83]	Logic stages are resized using the optimal fanout factor for the given output capacitance (ramped driver heuristic). After resizing logic stages, power is minimized by slowing (resizing gates) on non-critical paths.
[Glas84] (proposed)	Buffers can be inserted to make the fanout factor as close to optimal as possible. Duplicate buffer stages could be introduced to reduce fanout.
MOST [Pinc86]	Increase the transistor width by one which contributes most to the delay. Identify the transistor width that is most beneficial to the critical delay path. Increase this transistor width by a varying amount. Resize one entire critical path using partial derivative information.
TILOS [Fish85]	Increase the transistor width that causes the greatest timing improvement (along one of the critical delay paths) with the least impact on the total active area.
XTRAS [Kao85]	Increase the logic gate size (a gate-level delay model) in one micron increments that causes the greatest timing improvement along a critical path with the least impact on the total active area.

Table 3.3. **Summary of heuristics used by several systems.** This table describes the heuristics used by several optimization systems to improve the performance of MOS circuits.

significant fanout capacitances that are highly problem dependent. Nemes [Neme84] describes two additional effects that influence the optimal scale factor: nonzero delay of an unloaded inverter and the difference between propagation delay time and rise time.

Glasser and Hoyte [Glas84] propose two techniques for reducing the critical path delay. The first inserts one or more buffer stages between large capacitance nodes and its driver. This approach is based on the observation that there is an optimal number of buffer stages to drive a given load capacitance. The other technique is to decouple the loads on a node by replicating the driver. Since both of these techniques involve a modification of the circuit structure, their usefulness will be evaluated in the next chapter.

Pincus [Pinc86] considers and evaluates several transistor sizing strategies for minimizing the circuit delay. One technique is to increase the transistor width by one that contributes most the critical path delay. This approach is similar to TILOS [Fish85]. Another variation of this technique is to change the transistor by a varying amount. However, this technique usually produces larger transistors than necessary since the proper size for a transistor cannot be determined before the surrounding transistors reach their final size. MOST also considers sizing the transistors on an entire critical path using partial derivative information. Again, the additional computation expense for setting the transistors on a particular critical path is only effective if there are few competing critical paths. A simulated annealing approach required a much larger execution time than the other heuristics to achieve

a similar circuit delay reduction.

XTRAS [Kao85] uses a gate-level delay model to determine the circuit delays. A logic gate that contributes the most to the total circuit delay/area will be increased by one micron. This approach is essentially a gate-level version of the TILOS heuristic. The next section will describe and evaluate the TILOS heuristic for sizing transistors to reduce the overall circuit delay.

3.5. TILOS Heuristic

TILOS [Fish85] implements one of the more popular heuristics for improving the delay of a circuit. This transistor sizing program iteratively increases a transistor width along the worst critical delay path. TILOS selects a transistor that can most significantly reduce the delay while incurring the least total transistor area increase. The process terminates when all delay constraints are satisfied or when no transistor exists that will decrease the delay. Since the sum of transistor areas is a component of total power, TILOS essentially minimizes power subject to delay constraints.

The advantage of the TILOS heuristic over the others is that it rapidly improves the circuit response time by increasing transistor widths by a multiplicative factor (TILOS's bumpsize). A small fixed change in a transistor width has a large impact on response time when the transistor size is small. This is because the change is a larger proportion of the small transistor width. TILOS avoids making small incremental changes to large transistors since a small change in a large transistor has little effect on the overall response time. Therefore, TILOS improves the overall delay faster by only making changes that have a large impact on the total cell delay.

EPOXY extends the TILOS heuristic for use in other optimization problems by generalizing the notion of critical paths in terms of the underlying performance equations. A set of dependent limiting or failing constraints ($g_i(x) \geq 0.0$) and the equations that interrelate them, will be considered a critical path within the generalized TILOS-style heuristic. Only the input variables to these critical equations can improve the failing constraints. Therefore, while there are failing constraints, the worst failing constraint is identified. The input variable that improves this worst failing constraint the most will be increased. After all constraints are satisfied, the input variable that best improves the objective function while maintaining feasibility (satisfied constraints), will then be increased.

TILOS relies on the general convex nature of the power/delay relationship. However, the point at which other paths become critical causes discontinuities that degrade the results TILOS can produce. One way to overcome this problem is to factor near-critical paths into the sensitivity calculations. For the equation abstraction, this simply means combining the partial derivatives for the output variable that are within some ε (e.g. $g_i(x) + \varepsilon \geq 0$).

Since TILOS only increases transistor sizes by a fixed multiple, it usually overshoots the minimum possible implementation. If the algorithm were generalized so that decreasing transistors sizes were considered to improve the overall

power/delay savings, better solutions should result. However, endless cycling between increasing and decreasing transistors may result. Unlike optimization-based algorithms which separately determine the appropriate step size, TILOS uses a fixed small factor with usually requires more iterations to complete. Even though TILOS requires more iterations than optimization-based algorithms, the step size computation for optimization-based techniques is significant. The overall running time for TILOS is usually less than those for optimization-based algorithms as demonstrated in the next subsection.

3.6. Comparison of Solution Quality and Running Times

Since VLSI designers are interested in achieving quality results rapidly, the augmented Lagrangian algorithm and TILOS-style heuristic are best compared by the solutions produced and the running times required to optimize a few representative CMOS examples. EPOXY facilities a uniform comparison between optimization techniques, since all optimization routines rely on the same evaluation subroutines for the electrical and area models.

Circuit	Optimization	Performance										
	Metric	Area		Height		Width		Power		Delay		A.L. CPU
	Units / Improve	λ^2	%	λ	%	λ	%	μW	%	ns	%	min
inv.8	$w=w_{min}$	4,288		134		32.0		19.0		29.82		0.00073
C = 1pf	min t	11,709	173	134		87.4	173	32.4	70	**6.46**	**-78**	3.3
	min a t≤7ns	**8,331**	**94**	134		62.2	66	31.6	-77	7.00	-77	3.6
	min p t≤7ns	11,296	163	134		84.3	163	**26.9**	**41**	7.00	-77	4.3
rand.20	$w=w_{min}$	7,140		140		51		31.8		26.6		0.00081
C = 1pf	min t	10,276	44	140		73.7	44	41.3	30	**6.42**	**-76**	6.9
	min a t≤7.11ns	**9,180**	**29**	140		65.6	29	40.7	28	7.11	-73	5.51
	min p t≤7.11ns	9,709	36	140		69.4	36	**40.2**	**26**	7.11	-73	4.3
adder.inv.4	$w=w_{min}$	35,378		266		133		108.3		63.6		0.0058
C = 1pf	min t	40,290	13.9	302.9	13.9	133		134.5	24.2	**17.87**	**-72**	31.3
	min a t≤30ns	**35,684**	**0.9**	268.3	0.9	133		114.2	8	29.79	-53	20.5
	min p t≤30ns	35,982	1.7	270.6	1.7	133		**113.3**	**4.6**	29.84	-53	18.6

Table 3.4. **Results of several performance optimization problems.** Results of optimizing three static CMOS circuits using the augmented Lagrangian technique are given: an unconstrained minimization of delay (min t), and time constrained min. of area (min a s.t. t≤ ...) and power (min p s.t. t≤ ...). The first row, ($w = w_{min}$), gives the performance of the circuit with all devices of minimum size. Performance is defined as the total cell area (A), height (H), width (W), power (P), maximum delay (T) and the CPU time (CPU) that the augmented Lagrangian algorithm took to produce the result. The percentage increase or decrease over the minimum sized implementation for each of the design parameters are also listed. Values in bold are minimum. An extra load capacitance is added to outputs (C = 1pf). A maximum frequency of 500 kHz was used to calculate the maximum dynamic power (P) for each of these examples.

Table 3.4 shows the significant performance improvement transistor sizing offers. It also illustrates the minimum required sacrifice in the other design metrics to achieve the desired performance. For example, $7{,}421\lambda^2$ additional area is

required to achieve the fastest implementation, 6.46ns. This translates into a 173 percent increase in area and a 70 percent increase in power for a 78 percent decrease in delay. Aggressive constraint times were chosen to force each design into the region where the tradeoff between the design metrics is difficult. The running time increases with problem size, as the four stage adder illustrates. Although the CPU running times for the adder appear to be quite large, they are comparable with other transistor sizing programs [Mats86].

Next, a graphical comparison of results for the augmented Lagrangian algorithm and the TILOS-style heuristic is given for minimizing the cell area subject to timing constraints in *Figure 3.9*. Since TILOS strictly greedily increases transistor sizes by a fixed amount, the implementations produced by a TILOS-style heuristic require more power than those for the augmented Lagrangian algorithm. TILOS produced larger transistors widths than the augmented Lagrangian method to achieve some of the same delay times (designs running slower than 30ns). In addition, the TILOS-style heuristic cannot reduce the delay less than 30ns.

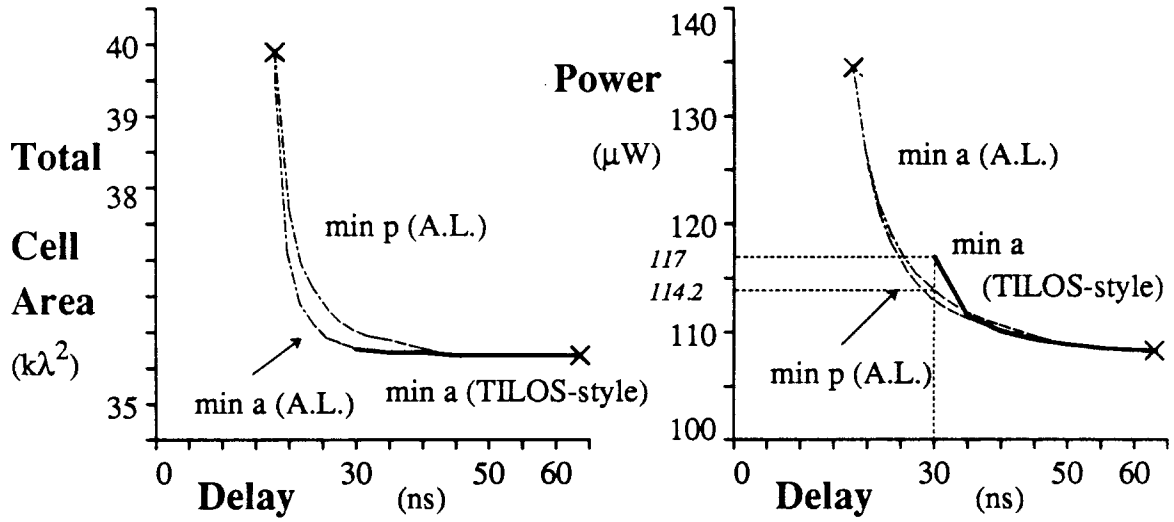


Figure 3.9. **Comparison of TILOS and augmented Lagrangian methods.** These graphs illustrate the area and power required by the minimum area and power configurations for a CMOS four-bit adder as produced by the augmented Lagrangian algorithm versus a minimum area TILOS-style heuristic. The area requirements for both minimum area algorithms are similar as shown by the superimposed lines on the left graph. However, the implementations produced by the TILOS-style heuristic require more power (117 μW for 30ns) than produced by the augmented Lagrangian algorithm (114.2 μW for 30ns). Since the TILOS-style heuristic can only increase transistor sizes, it cannot reduce the delay to less than 30ns. (A fixed-sized inverter has been added to each input and a 1pf load has been added to all outputs.)

A comparison of the convergence rates of these optimization techniques for a particular delay constraint (30ns) for the CMOS four-bit full adder (adder.inv.4) shows that the TILOS-style algorithm (for min area) met the constraint rapidly (8.66 CPU min.), requires similar area (35,473 λ^2), but more power (117 μW). When the results of the TILOS-style heuristic are used as a starting point for the

augmented Lagrangian method, results comparable to the original augmented Lagrangian method are eventually obtained (114.2 μW). For this example, the combination of both algorithms achieved results of similar quality using fewer iterations (total of 16.8 CPU min) than the augmented Lagrangian alone (20.5 CPU min. from *Table 3.4*). However, using the generalized TILOS heuristic for establishing a initial solution rapidly may not be a good strategy for problems that are very aggressive since the heuristic tends to over-shoot the optimum solution.

This demonstration illustrates that computation to achieve quality results can be reduced further by using simple models and heuristics to locate the probable region of the global minimum. Then accurate models further refine the search. Simple models are rapidly evaluated and lead to a fast solution since these simple models can eliminate many of the local minima. Accurate models take longer to evaluate and represent a more complex design space that may exhibit many local minima. This approach also avoids getting trapped in local minima early in the optimization.

3.7. Conclusions

A designer must understand the potential performance tradeoffs before an objective function and constraints are formulated to describe the desired implementations. The *performance envelope* illustrates all the designs of interest. This envelope defines the region of circuit performance available through transistor sizing. Typically, a designer concentrates on the narrow portion between the minimum area and minimum power configurations that satisfy the timing constraints.

VLSI designers are interested in producing layouts that meet the performance goals rapidly. This chapter has compared the running times and solution quality for the augmented Lagrangian algorithm and the TILOS-style heuristic on several CMOS examples. The augmented Lagrangian algorithm was chosen as an example optimization-based method, since it has been used in another transistor sizing program, COP. The TILOS heuristic is one of the more popular transistor sizing techniques despite its non-optimal results. The major conclusions of this comparison are:

1. The augmented Lagrangian method is slow but produces designs with better performance characteristics.
2. The TILOS-style heuristic is fast but misses better solutions.
3. When the TILOS-style heuristic provides the initial values to the augmented Lagrangian method, the combined algorithm produces designs with good performance faster.

The key to this uniform comparison was formulating the design problem as a standard non-linear optimization problem. Optimization routines are easily substituted since they all use the same performance evaluation routines. The flexibility of the approach is demonstrated by deriving performance envelopes which required several different constraint values.

4. Performance-Based Circuit Modifications

Although transistor sizing can dramatically improve the overall circuit performance, failing performance constraints may require modifications to the circuit structure. EPOXY considers several local circuit modifications, such as splitting large transistors, inserting buffers, and reordering transistors within pull-down or pull-up transistor networks.

There are many possible places to apply these techniques within the circuit, some better than others. However, since these circuit modifications have a significant impact on the signal routing, the original circuit is altered only when absolutely necessary to meet the required performance goals. This chapter examines the effects of several circuit modification techniques on several design problems.

Heuristics provide a useful mechanism for encoding valuable designer expertise. Circuit improvement heuristics determine where the circuit structure should be altered to help meet some failing design constraint. Many of these heuristics are motivated by a critical path analysis. Although a critical path usually refers to a limiting delay path from input to output nodes through transistors, the concept can be generalized to all design metrics. For example, a transistor width is on a cell width critical path if an increase in this transistor width changes the overall cell width. The critical path for power includes all transistor widths and lengths. Since EPOXY derives the equations that model the electrical performance and cell dimensions, a critical path is easily identified as a list of limiting (or unsatisfied) constraints and their variables.

Technique	When to apply	Possible effects
split large transistors	reduce height reduce width reduce area	increase width, area change increase height, area change height and width changes
insert a buffer: increase output drive reduce critical path load	 reduce delay reduce delay	 increase height, width, area increase height, width, area
reorder transistors	reduce delay	height, width, area changes

Table 4.1. **Summary of circuit modification techniques.** The potential benefits (when to apply) and costs (possible effects) are summarized for each technique considered within this chapter.

EPOXY separates the application of circuit restructuring heuristics from the transistor sizing aspects (optimization) to reduce the problem complexity. While the transistor widths can assume non-integer values, application of circuit modifications involves discrete design decisions. If these integer decisions were combined with the non-linear transistor sizing problem, a separate integer variable would be needed to represent each possible circuit modification. This would result in an *integer* NLP with a very large input vector space. Unfortunately a larger input vector size

will dramatically increase the running time of any solution technique. Since typical design problems only require a few circuit modifications to improve the design performance, many of these integer variables will be zero. Instead, EPOXY postpones the decision to modify the circuit until it is clear that the design constraints cannot be met by transistor sizing only. While this approach may not yield the globally optimal result, most of the performance improvements are achieved rapidly.

Table 4.1 summarizes when to apply these techniques and describes which design constraints may in turn be affected. In the next subsections, each circuit modification technique is presented in greater detail.

4.1. Split Large Transistors

Transistor sizing usually produces a few large transistors which dominate the design and cause an increase in the overall cell dimensions. Depending on how these large transistors fit into the layout, cell height and width may need to be adjusted. Large transistors can be split into a few smaller ones to decrease the overall cell height while potentially increasing the cell width (and vice-versa) [Hill87]. *Figure 4.1* shows the benefits of splitting larger transistors in a standard-cell buffer.

An area model relates changes in transistor size to changes in the overall cell height, width, and area. EPOXY employs a virtual grid area model that determines stretch lines through the layout to accommodate larger transistors. Since these lines can be stretched, the area model easily relates a change in transistor size to a potential change in the overall cell.

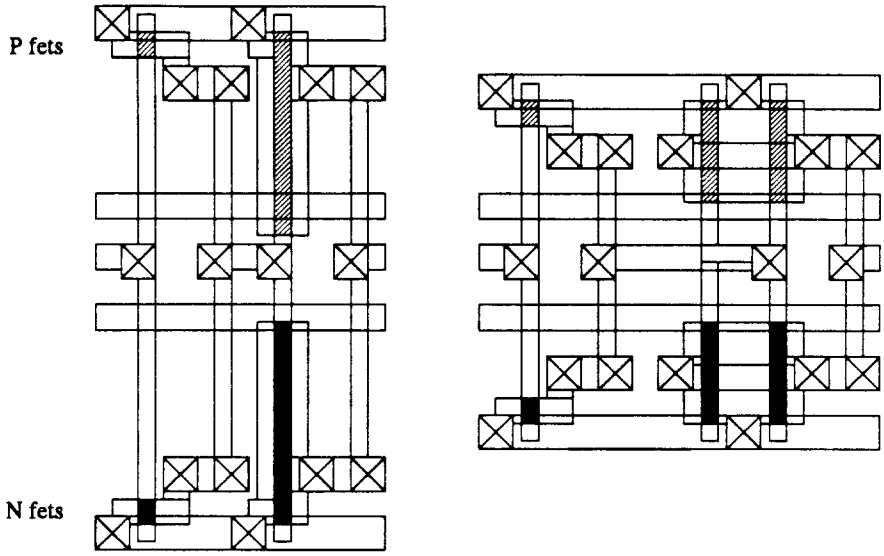
In EPOXY, larger transistors are split rapidly by directly modifying the underlying equations. For example, if the width constraints are satisfied, but the height constraints are not, the non-zero track heights are examined. The layout for the standard-cell adder was presented in *Chapter 2*. Only the non-zero track heights may affect the cell height. The transistors that determine this track height are easily identified, since their corresponding track height constraint is limiting. A constraint is limiting if the equality portion of the constraint is active. If any of these transistors do not participate in any of the limiting width constraints, splitting any one of these transistors will reduce the overall cell height without incurring any extra cell width. Therefore, the transistors that limit the cell height, but produce the least change in cell width, are split. This process continues until the height constraints are satisfied or the width constraints are no longer satisfied. For the CMOS standard-cell structure of the full adder circuit, the equations which determine the overall cell dimensions are:

$$h_{cell} \le height_{desired} \qquad h_{cell} = h_{n\ adder.1} + h_{p\ adder.1} + \cdots + K_h$$

$$h_{n\ adder.1} \ge w_{fet\ in\ n\ adder.1} - w_{free\ for\ fet} \qquad h_{n\ adder.1} \ge 0.0$$

$$w_{cell} \le width_{desired} \qquad w_{cell} \ge w_{n\ adder.1} \qquad w_{cell} \ge w_{p\ adder.1} \qquad \cdots$$

$$w_{n\ adder.1} = l_{fets\ in\ n\ adder.1} + \cdots + K_w$$



Height	(λ)	64	44
Width	(λ)	34	47
Area	(λ^2)	2,176	2,068

Figure 4.1. **Example of splitting larger transistors.** The transistors in the two instances of a buffer cell are highlighted. The overall cell height is reduced at the expense of additional cell width when each of the larger transistors are replace by two smaller transistors in parallel. For this example, the cell area is reduced by splitting the larger transistors.

If the desired cell height is not met, transistors are identified, such as those whose width (e.g., $w_{fet\ X}$) limits $h_{n\ adder.1}$. The best transistors among these are those whose length variable, $l_{fet\ X}$, minimally affects the limiting constraints for w_{cell}. Splitting a transistor requires duplicating the corresponding $h_{n\ adder.1}$ constraint equation for the new transistor width. The variable for the new transistor length is also added to the corresponding $w_{n\ adder.1}$ constraint. The constant for this width constraint, K_w, is increased by a worst-case value to account for the extra routing to connect the new transistor. The value of the variables for the old and new transistor widths is half that of the original transistor width.

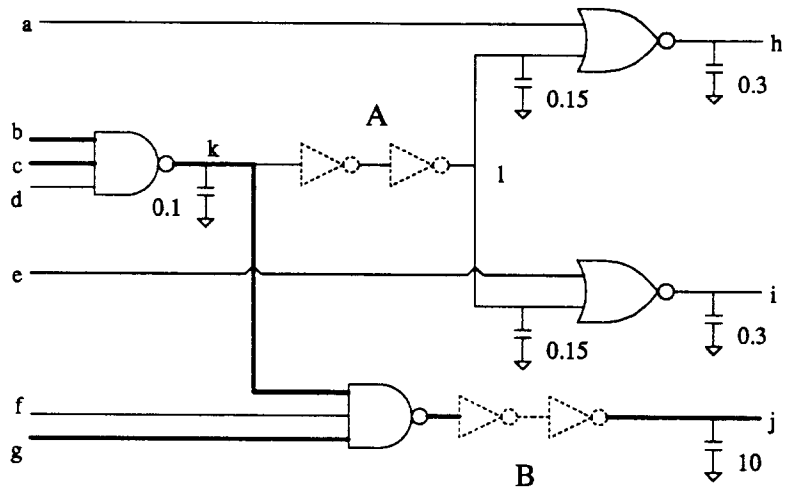
Since the original transistor is larger than a diffusion contact, the additional parasitic capacitance from the new diffusion contact is small when compared to the diffusion capacitance of the transistor. Therefore, the overall timing for the circuit does not substantially change when large transistors are split into smaller ones. However, incremental evaluation of the circuit timing due to the split transistor does not entail much computation. EPOXY incrementally updates all the equations after the large transistors are split.

Ideally, snaking of transistors, local replacement, and re-routing could be considered. However, these techniques are quite difficult to implement and

computationally expensive to evaluate in all but highly constrained design styles, such as PLA's as produced by PLATO [Marp86].

4.2. Insert Buffers

Overall circuit delay may be reduced by inserting buffers (inverter pairs). However, the added circuitry usually results in an increase in cell area and power consumption. The only buffer insertions EPOXY considers are those that do not violate cell size constraints (maximum height, width or area). Glasser and Hoyte [Glas84] outline two techniques for inserting buffers to reduce critical path delay: reduce critical path load and increase drive to large loads. These two strategies for inserting buffers are outlined in the following paragraphs and illustrated in *Figure 4.2*.



Circuit	Insert buffer	Area $k\lambda^2$	Height λ	Width λ	Time ns	Power μW
sized	none	**16.8**	**99**	169	21.5	169
reduce loading	A	21.9	130	169	20.9	171
increase drive	B	18.2	135	**135**	15.4	**166**
both buffers	A,B	22.1	162	136	**14.5**	167

Figure 4.2. **Example of buffer insertion strategies.** For this fragment of CMOS random logic, there are two possible locations for inserting buffers to improve the overall response time (Time). The area, height, width and power requirements are given for the fastest implementation of each circuit *after* transistor sizing. The critical timing path through this logic is highlighted in bold. When a buffer (A) removes the non-critical path load caused by the two NOR gates, the circuit delay is reduced to 20.9ns. Alternatively, when a buffer (B) isolates the large load on node *j*, the delay is reduce even further. When both inverters are present, the fastest design results.

EPOXY starts with the unsized layout as shown in *Figure 4.3* for the standard-cell CMOS random logic example as described *Figure 4.2*. The overall

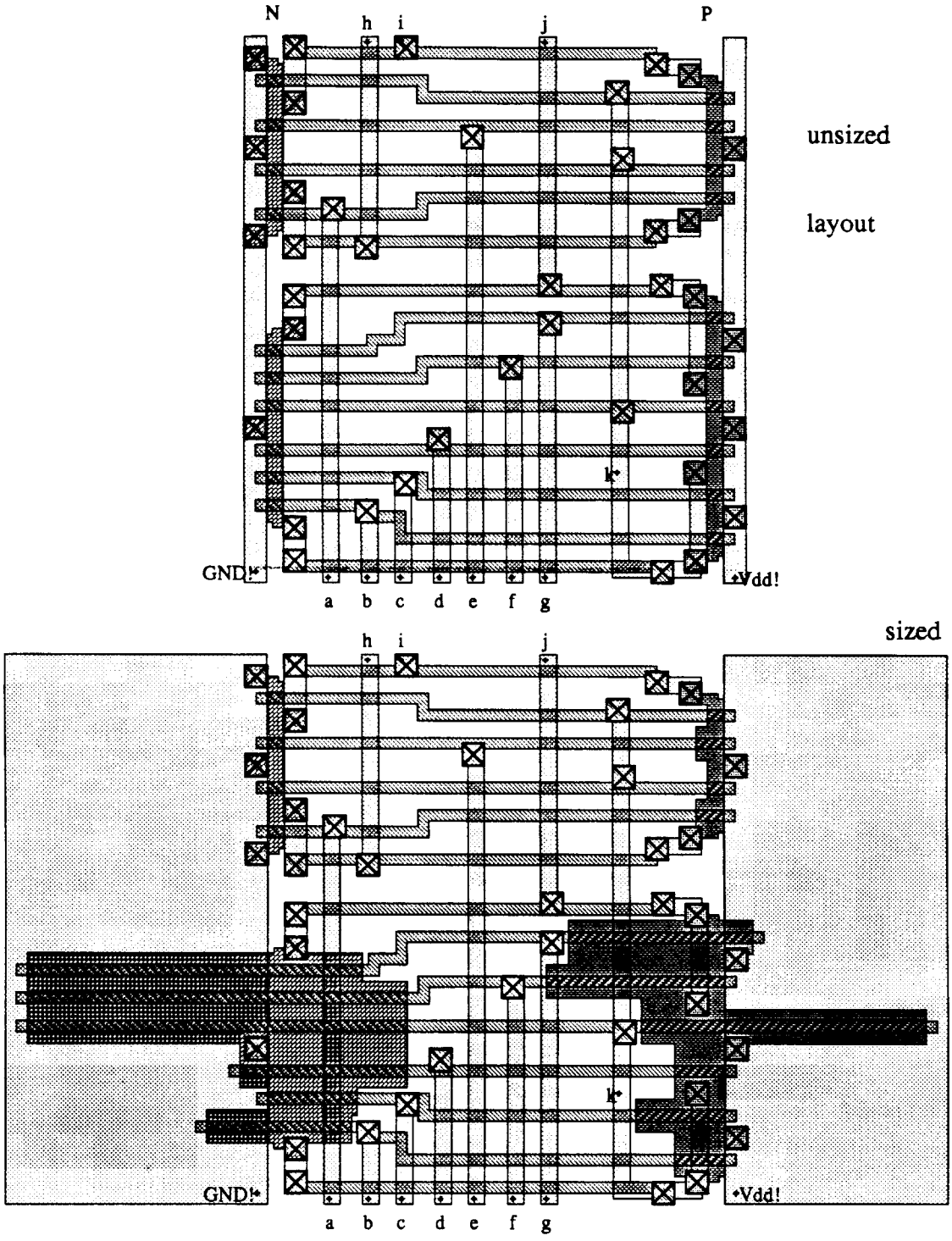


Figure 4.3. **Layouts before and after transistor sizing.** This figure illustrates the standard-cell CMOS layout for the random logic example. Starting with the *unsized* circuit, EPOXY sizes the transistors for minimum delay. The layout for the *sized* circuit results.

circuit delay is reduced by sizing the transistors for minimum delay. The performance for the sized layout fragment corresponds to the data given in *Figure 4.2*. The resulting layout is larger, since the design contains several larger transistors.

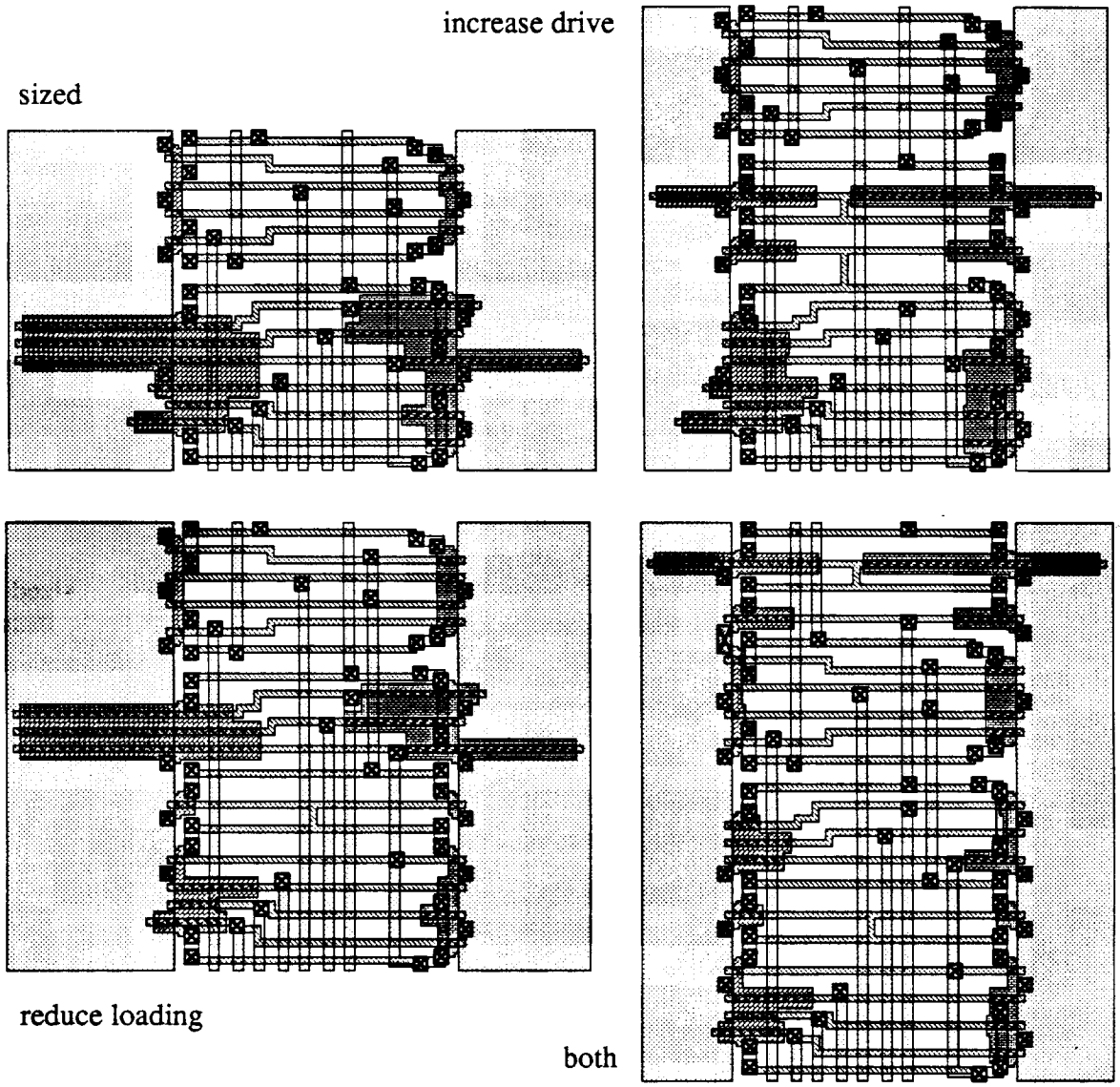


Figure 4.4. **Layouts for buffer insertion strategies.** These standard-cell CMOS layouts illustrate the results of applying each buffer insertion strategy and then resizing each design as outlined in *Figure 4.2*. Note that the *sized* design requires the least cell height. However, the design with both buffers requires the least cell width.

Circuit speed can be increased by reducing the capacitance of nodes along critical delay paths. Nodes along a critical path with a fanout greater than two can be split such that the inserted buffer drives the non-critical inputs separately. In *Figure 4.2*, the non-critical path load of the two NOR gates is removed from the critical path node *k* by inserting buffer A. However, if the buffer introduces sufficient delay to create a new limiting critical path, the buffer is removed. Then the circuit

is restored to its original state, and the node is marked to indicate that no beneficial buffer insertion was possible.

Another buffer insertion strategy is to increase the current drive capability (sourcing and sinking) of gates with large capacitive loads. In *Figure 4.2*, the large load on node *j* is isolated by inserting buffer B. Several authors, e.g. [Mead80] [Neme84] [Hede87], have shown that an optimal number of inverter stages and inverter size ratios are required to drive a large capacitance with minimum delay. By running EPOXY on several CMOS inverter chains, *Figure 4.5* shows that the load capacitance must be fairly large for the insertion of buffers to reduce the overall delay. Therefore, only large capacitance loads (> 0.78pf) on a critical paths are considered for buffer insertion. The inverters are inserted if space permits (height, width or area restrictions are not exceeded). These results are similar to those for nMOS [Ober85].

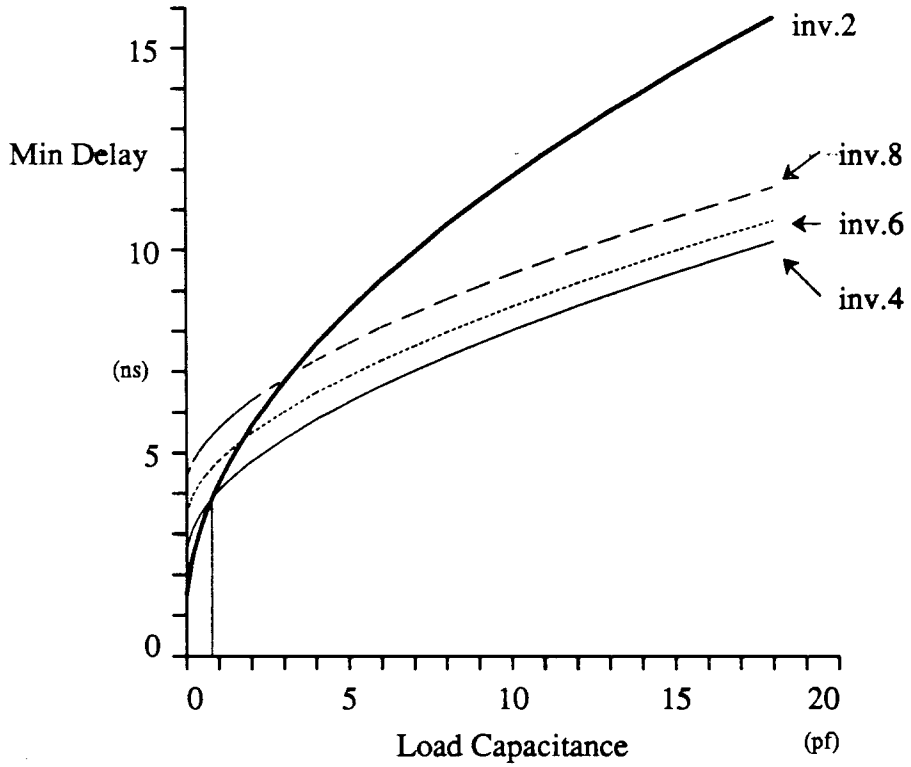


Figure 4.5. **Graphs of the fastest inverter chains.** The graph provides the response time for the fastest implementations of two, four, six, and eight CMOS inverters to drive a range of load capacitances. The first inverter in each chain is fixed to minimum sizes. Other stages are sized for minimum delay for each load capacitance. For small loads, the two inverter configuration is the fastest. However, above 0.78pf (see vertical line), the larger chain of four inverters drives the load capacitance fastest. At no point do additional stages help reduce the overall circuit delay.

The layouts for the standard-cell CMOS fragment, as given in *Figure 4.4*, shows that the sized design requires the least cell height. When a buffer is added to reduce the non-critical path load (*Figure 4.2*, buffer A) and the circuit is then

resized, a faster design results. However, when a buffer is inserted to increase the output drive (*Figure 4.2*, buffer B) and this circuit is resized, EPOXY produces an even faster design with the least cell width. This is design is also the most power efficient. When both inverters are used, the fastest design results. Which of these implementations is best depends on the user supplied performance constraints. Therefore, the decision to apply these buffer insertion techniques must consider the resulting effects on the performance constraints.

Lee and Soukup [Lee84] consider two optimization problems: minimum delay and minimum relative area for a single critical delay path. They describe an algorithm for determining the number of logic stages and the size of each logic gate. EPOXY solves these problems at a more detailed level; individual transistor sizes and actual cell dimensions are determined and the effects of multiple critical paths are considered.

4.3. Reorder Transistors

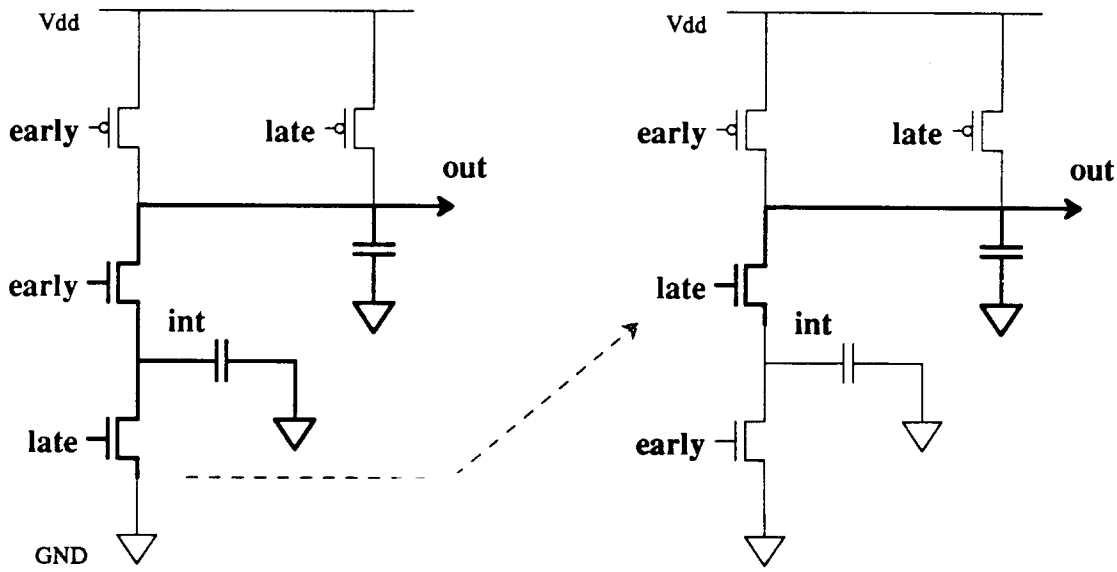


Figure 4.6. **Reordering transistors for a NOR logic gate.** This static CMOS implementation of NOR logic gate demonstrates the advantage of placing late arriving signals close to the output node. The delay for the left circuit diagram is determined by the late arriving signal and the discharge of the parasitic capacitance on nodes **int** and **out**. The output node is discharged a slower rate due to the additional resistance of the **early** transistor and the extra capacitance on node **int**. However, if the **late** signal is attached to the transistor closer to the output node as the dashed arrow indicates, the circuit delay is reduced. For the circuit on the right, the **late** transistor directly discharges the capacitance on the output node **out**. The critical delay path through each circuit is highlighted.

Reordering transistors along a pull-down or pull-up path can dramatically affect the delay of the output node as shown by MTA [Hofm87]. Transistor inputs (gate nodes) that switch relatively early should be placed near the supply rails so that the sources and drains of the remaining transistors can be properly set. Alternatively,

transistor inputs that switch later should be placed near the output nodes, since they can limit the overall delay of the logic gate. *Figure 4.6* illustrates the advantage of placing late arriving signals near the output node.

A more complex example in *Figure 4.7* illustrates that only the latest signal should be moved. The input signal names and latest response time (in nano-seconds) are given for each transistor in a static CMOS gate. The transistors and nodes that lie on the critical delay path are highlighted. Since signal **D** is the latest arriving signal to this gate, the transistor it drives is placed closest to the output node, *out*. The p-channel transistor driven by signal **A** was not moved because the gate response time is dominated by the slower signal **D**.

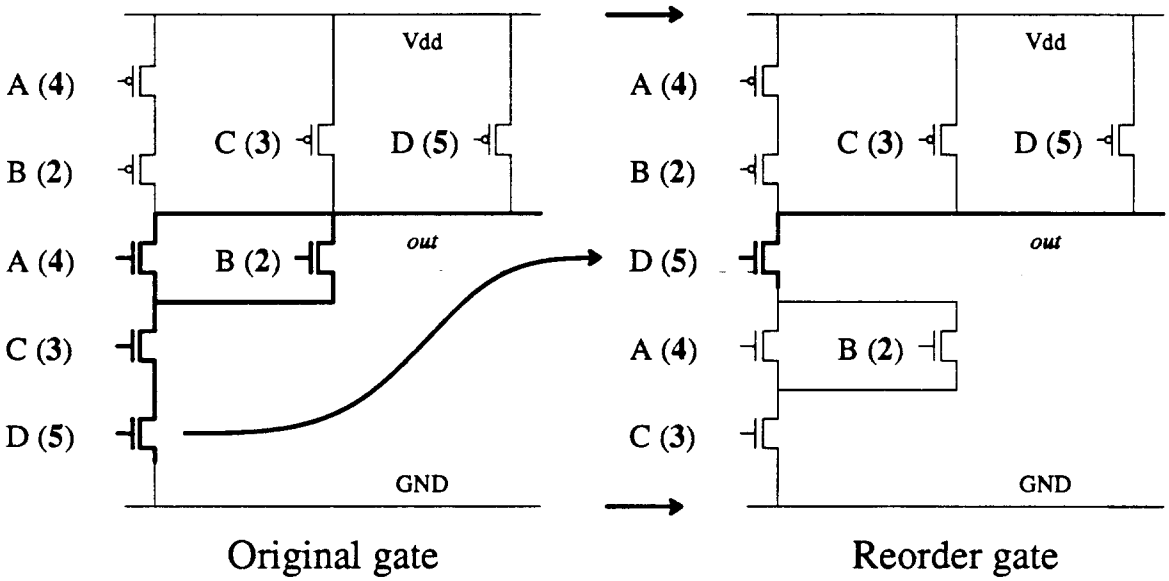


Figure 4.7. **Example of reordering transistors.** The input signal names and latest response time (in nano-seconds) are given for each transistor in a static CMOS gate. The transistors and nodes that lie on the critical delay path are highlighted. Since signal **D** is the latest arriving signal to this gate, the transistor it drives is placed closest to the output node, *out*. The p-channel transistor driven by signal **A** was not moved because the gate response time is dominated by the slower signal **D**.

Rearrangement of transistors must preserve the logical function of the gate (i.e., series and parallel groups). However, the re-routing of signals to these transistors may require additional height or width. Therefore, EPOXY only reorders those transistors that effect the overall response time. Only the latest arriving inputs to the logic gate are moved closer to the output nodes since they determine the overall gate delay. *Figure 4.7* illustrates which transistors should be reorder to improve the response time of the example logic gate.

Dynamic gates impose a restriction on the maximum parasitic capacitance on the non-output nodes to avoid potential charge sharing problems. EPOXY currently does not impose restrictions on the maximum parasitic capacitance; however, this effect is minimal when considering static logic based standard-cell layout. One way to support dynamic gates within EPOXY is to simply increase the capacitance

on the output node by enlarging the transistors that this node drives. Therefore, the effects of charge sharing with the internal nodes to the logic gate will be reduced.

4.4. Strategy for Applying Several Circuit Modification Techniques

There are many possible ways to apply circuit modification techniques to improve overall performance. The decision to apply a modification can be encoded as an integer selection problem. Unfortunately, the underlying transistor-sizing sub-problem is non-linear and computationally intensive. Therefore, any strategy for selecting a good set of modifications should reduce the number of circuit implementations for transistor sizing.

One way to reduce the number of modifications is first to apply these techniques to parts of the design where they provide the most benefit. Heuristics provide a convenient mechanism for encoding this designer knowledge. Since these heuristics many not accurately predict a sufficient improvement, EPOXY applies the performance improvement heuristics in a greedy manner with a single backtracking level if the design is not improved. Therefore, if a circuit cannot meet some design constraint by sizing the transistors, then the circuit structure will be altered. The pseudo code in *Figure 4.8* describes the current implementation of circuit restructing within EPOXY.

```
improve_design ()
{     size transistors;
      while (failing or limiting constraints && heuristics to apply) {
            if (failing or limiting height, width, area) {
                  consider (splitting large transistors);
            }
            if (failing or limiting delay) {
                  consider (increasing drive on large loads);
                  consider (reducing critical path load by inserting buffers);
                  consider (reordering transistor within logic gates);
            }
      }
}

consider (heuristic)
{     try heuristic;
      resize transistors;
      if (failing or limiting constraints reduced)      {accept modification;}
      else                              {undo heuristic;
                                         mark heuristic failure;
                                         resize transistors;}
}
```

Figure 4.8. **Pseudo code to apply several heuristics.** This pseudo code describes how each of these circuit modification techniques can be combined to improve the performance of a design. Each of the techniques is considered in a greedy manner with a single level of backtracking if the resized circuit was not improved. The appropriate element is marked to prevent reapplication of the technique.

EPOXY first tries to meet the design goals by transistor sizing alone since transistor sizing has the least impact on the layout. While there are any failing performance constraints and there are places in the design where these techniques can applied, each heuristic is considered. Splitting of large transistors is considered only if there are failing height, width or area constraints. This technique does little in the way of altering the overall circuit timing.

While there are failing delay constraints, buffers may be inserted to increase a gate's output drive for large loads or to reduce the critical path load capacitance by isolating non-critical path loads. The increased drive technique is considered before reducing critical path load capacitance since the increased drive usually improves the overall timing the most. Transistors within logic gates are reordered if they improve the overall timing as well.

When a heuristic is considered, the best candidate area of the design is selected. Then the change is made and the transistors are resized. If a better solution results, this new configuration is accepted. Otherwise, the component of the circuit is marked so that this change will only be considered once. These marks prevent reapplication of circuit changes that do not improve the design.

The section that considers circuit changes does not keep track of the entire previous circuit configuration and simulation values. Rather, only the components that are affected by the modification are marked. These aid in restoring the circuit to its previous configuration, if its performance has not been improved. Also, the marks require much less storage than a copy of the entire design, and the restored circuit can be quickly resimulated using the incremental circuit simulation mode.

4.5. Performance Improvement for a Few Examples

The value of these circuit modification techniques is best demonstrated by a few VLSI design examples. A CMOS 16-bit ripple adder shows that splitting large transistors can reduce cell height at the expense of cell width. Inserting buffers and reordering transistors further reduces operating delay. As described in *Chapter 2*, the CMOS dynamic PLA shows that the area model is not restricted to standard-cell style layout. The only inherent difference in the area model between these layout styles is that the stretch lines for the tracks intersect for the PLA's while these lines run strictly parallel for standard-cell designs. Due to the PLA's structured design, all of the beneficial design changes for the PLA were made to the input and output buffers. An array of JK flip-flops illustrates the timing improvements that result from application of all of the the circuit modifications.

Table 4.2 summarizes the performance improvements by applying EPOXY to a static CMOS 16-bit ripple adder. Data path designs typically require a strict cell pitch so that other predefined cells directly abut. A common example is in a processor design where the registers are predefined by a layout generator. For this example, the starting cell size will be used as the height constraint (cell Height $\leq$ 1064). The goal is to produce the fastest implementation that drives a 16 bit bus with 5pf loads. The 16-bit ripple adder contains 514 transistors and 292 nodes. EPOXY required 4.9 Mbytes of SUN-3/140 storage to complete this example.

Circuit: adder.16 Optimization: **min** Time **subject to** Height ≤ 1064λ						
	Times applied	Area $k\lambda^2$	Height λ	Width λ	Time ns	Power μW
unsized circuit	–	134	1064	126	116.7	1190
transistor sizing	1	136.7	1085	126	57.4	1482
split large trans.	5	153.2	1064	144	57.4	1482
insert buffer/load	0	–	–	–	–	–
insert buffer/drive	2	202.2	1105	183	49.7	1543
reorder transistors	2	136.7	1084	126	51.1	1447
optimize ()	1	213.9	1064	201	44.1	1524

Table 4.2. **Performance improvement for 16 stage ripple adder.** The fastest implementation of a 16 stage ripple adder subject to a predefined height constraint (1064 λ) is desired. The first two rows give the performance improvement resulting from transistor sizing over the original unsized circuit. Since the resized circuit does not meet the height restriction, each of the heuristics are considered. The next few lines list the performance resulting from applying each heuristic repeatedly. As expected, splitting large transistors reduces the cell height at the expense of the cell width. For this example, it was not beneficial to insert buffers to reduce the critical path load; however, two buffers were used to isolate the 5pf loads on the last adder stage. The final entry gives the fastest implementation when all heuristics are considered as outlined by the pseudo-code. This implementation satisfies the difficult height constraint while improving the worst-case response time by 23% over the version produced by transistor sizing alone. The maximum dynamic power at 1MHz is given in the last column.

Rather than replicating 16 instances of the CMOS standard-cell adder as given in *Chapter 2*, the outline of each cell is shown in *Figure 4.9*. All the transistors in the leftmost design are minimum size (unsized). The design (sized) that results from transistor sizing for minimum delay alone is illustrated using a dotted outline. Note that most of the individual cells are still minimum size since most transistors fit within their *free* width. The results of applying each circuit modification individually are illustrated in the next three cell outlines: splitting large transistors, inserting buffers to increase the output drive, and reordering transistors. When all of these techniques are employed, the rightmost cell outline results in *Figure 4.9*.

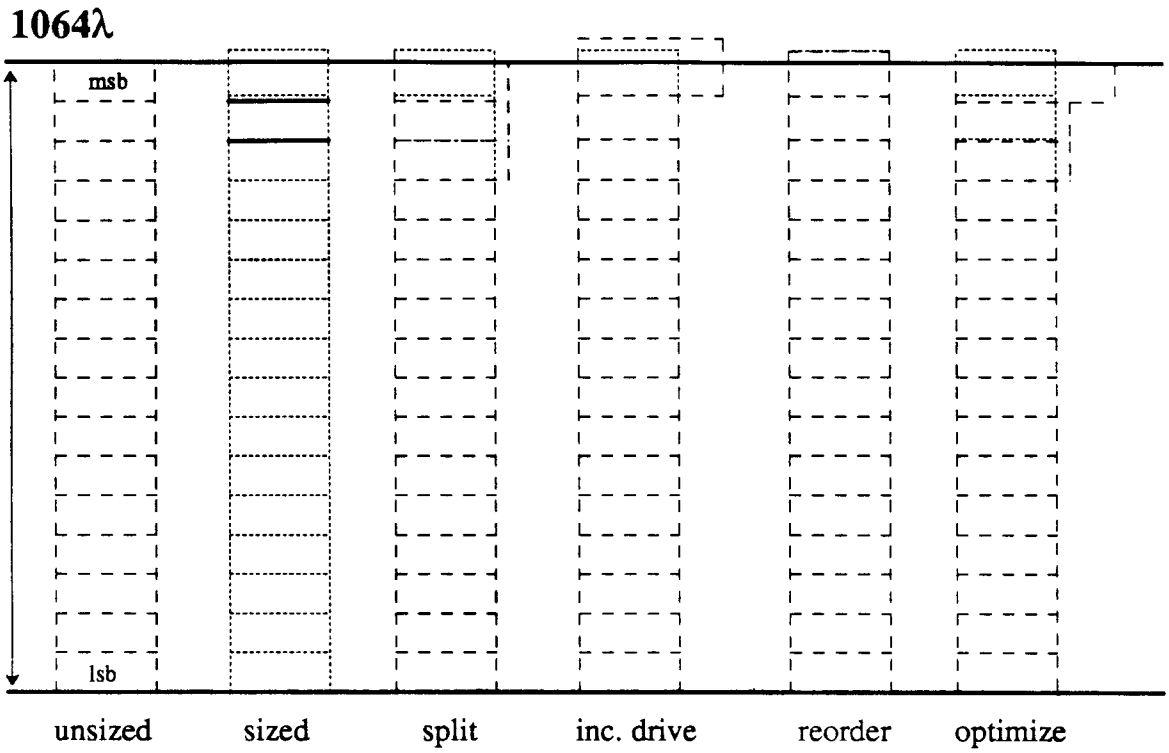


Figure 4.9. **Outlines of the 16 stage adder cells.** This figure illustrates the outlines for the CMOS standard-cell layouts of the 16 stage adder cell as given in *Figure 4.8.* The bold lines indicate the maximum height pitch. The least significant bit is at the bottom of the array and the most significant bit is produced at the top. The result of transistor sizing alone (sized) is shown using dotted lines in the outlines to the right. The results of apply each technique are shown using the dashed cell outlines: splitting large transistors, inserting buffers to increase the output drive, and reordering transistors. The rightmost design shows the result of applying all the techniques as outline in the pseudo code for optimize().

Table 4.3 gives the performance improvements for a PLA with 12 inputs, 10 outputs and 17 product terms. The dynamic CMOS implementation required 213 transistors and 80 nodes. A 5pf load was added to each output. Due to the restricted circuit structure of PLAs, two of the heuristics were not applied: inserting a buffer to reduce critical path load and reordering transistors within logic gates.

Due to the highly regular structure of the PLA, all improvements were limited to the output buffers. Essentially, all delay paths through the PLA are limited to either four or five gates levels (1 or 2 for the input buffer, 1 for the and-plane, 1 for the or-plane, and 1 for the output drivers). The ramped driver heuristic indicates that the last stage that drives a large load capacitance should be enlarged before any of the others. For a PLA driving a large load, this means that all changes are essentially limited to the output drives. The situation changes when the delay due to the capacitive load of the and-plane transistors dominates the overall delay through the PLA. Then the resulting problem mainly involves resizing the input drivers.

Circuit: pla.cpu1 Optimization: **min** Area **subject to** Time ≤ 24ns						
	Times applied	Area $k\lambda^2$	Height λ	Width λ	Time ns	Power μW
unsized circuit	–	94.7	266	356	105.6	752.9
transistor sizing	1	142.3	266	535	26.6	774.5
split large trans.	20	126.6	298	425	26.6	774.5
insert buffer/load	0	–	–	–	–	–
insert buffer/drive	10	218.7	428	511	24.0	812.1
reorder transistors	0	–	–	–	–	–
optimize ()	1	184.9	460	402	24.0	812.1

Table 4.3. **Performance improvement for a CMOS PLA.** The organization of the performance data is identical to the previous table. For this circuit a 5pf load was added to each output.

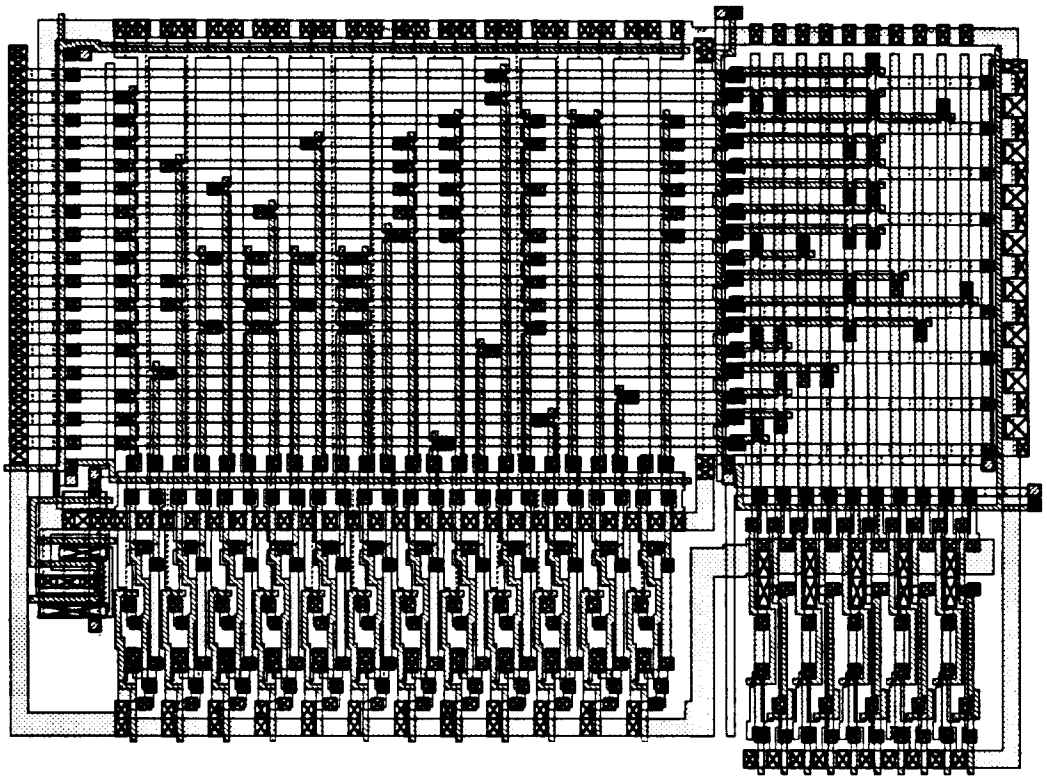


Figure 4.10. **Layout for the CMOS PLA: pla.cpu1.** This figure shows the CMOS layout for a dynamic PLA with minimum sized transistors except for the clock signals (**clk1** and **clk2**). The 12 inputs are applied along the lower left side and the 10 outputs are produce on the lower right side. The center PLA core contains 17 product terms as represented by the horizontal metal lines.

Circuit: jk.4 Optimization: **min** Time **subject to** Height ≤ 385λ and Time ≤ 16.0ns						
	Times Applied	Area $k\lambda^2$	Height λ	Width λ	Time ns	Power μW
unsized circuit	–	42.1	324	130	35.2	368.8
transistor sizing	1	53.0	408	130	19.7	492.0
split large trans.	8	58.5	380	154	19.7	492.0
insert buffer/load	4	61.4	391	157	19.1	591.4
insert buffer/drive	4	60.8	387	157	16.8	566.4
reorder transistors	8	51.4	395	130	19.4	492.0
optimize ()	1	68.8	380	181	15.2	577.4

Table 4.4. **Performance improvement for a JK array.** The organization of the performance data is also identical to the previous table. Each output has an additional 2pf load.

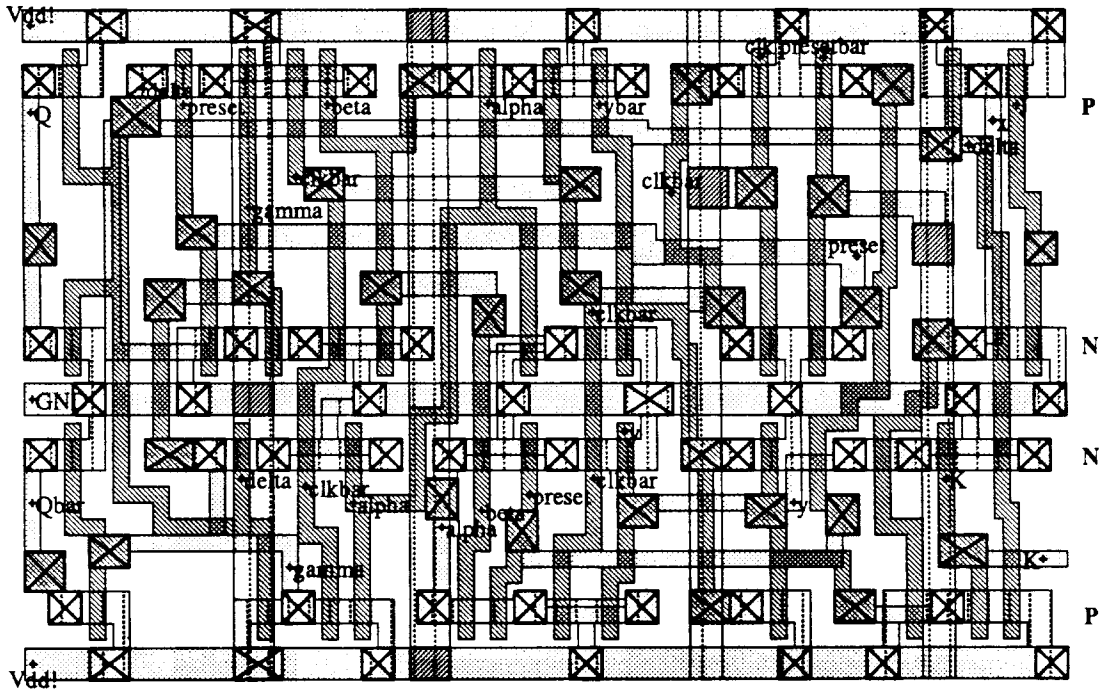


Figure 4.11. **Layout for one JK flip-flop.** This figure shows the CMOS layout for one element of the static JK flip-flop array. The jk.4 array is composed of four of these cells stacked vertically.

The performance for an array of CMOS JK flip-flops driving 2pf loads is given in the following table, *Table 4.4.* The layout for one element of the vertical array is shown in *Figure 4.11.* The response time is the maximum delay through each flip-flop. The height constraint was satisfied by splitting large transistors. Inserting buffers to reduce the loading on non-critical nodes improves the circuit

preformance slightly. However, inserting buffers to increase the drive on the output nodes improves the delay response further. Reordering some of the transistors marginally improves the circuit performance. The circuit that results from considering all these techniques satisfies the height and timing constraints, at the expense of additional cell width and area.

4.6. Conclusions

EPOXY provides an effective environment for developing and applying circuit modification heuristics in conjunction with transistor sizing. Transistor sizing can decrease circuit delay; however, the circuit may still not meet the design constraints. EPOXY applies circuit modification heuristics to alter the layout dimensions and to further improve the electrical performance. Since EPOXY takes advantage of previous sizing information after applying a circuit modification heuristic, the transistors are rapidly resized.

An advantage of representing design performance by equations is that limited circuit modifications result in limited changes in the equations. Therefore, circuit modifications are quick, and costly layout resynthesis and performance extraction are avoided. The next chapter will describe and evaluate the advantages of representing the performance by equations.

Global structural decisions should be made at the design synthesis level. Since EPOXY deals with the remaining optimization at the detailed layout level, this system only considers local circuit modifications. These heuristics have improved the 16-bit adder delay by a factor of 2.6 over the initial design and 23% over transistor sizing alone. Similarly, the speed of a dynamic CMOS PLA was improved by 10% and an array of CMOS JK flip-flops by 23%. In addition, the height constraints for these circuits could only be satisfied by applying the circuit modification techniques.

5. System Architecture and Implementation

EPOXY provides a unique flexible environment for evaluating the effects of circuit modification heuristics, optimization algorithms, and different electrical and area models. Modeling the circuit performance as a set of symbolic equations is the key to delivering the advantages of this flexible open environment for rapid performance improvement. This chapter describes how these equations are generated, evaluated, and modified. First, the advantages of representing circuit performance by static symbolic equations are presented. These equations tie together the major aspects of EPOXY: performance modeling, optimization and circuit modifications.

The goal of transistor sizing programs is to produce the best performance as defined by an objective function, subject to constraints on some of the design metrics. Many of these programs require sensitivity information to determine which transistor sizes may improve the overall circuit performance. Once these transistors sizes are changed, the circuit must be reevaluated. Therefore, solving the non-linear performance optimization problem involves repeated evaluation of the circuit performance and sensitivity information.

Circuit optimization programs spend most of their time in the analysis portion, since many devices must be reevaluated for each major circuit change. Crystal [Oust85] determines worst-case node delays by constructing RC paths through the circuit dynamically. After a node delay is determined, the data structures corresponding to the RC path are released. Therefore, performance optimization tools built directly into Crystal (such as AESOP [Hedl87]) would constantly recreate and destroy the same data structures.

EPOXY takes a different approach. It *statically* builds the symbolic equations that model the circuit performance and sensitivity information. Although this approach requires more memory, the computation involved in evaluating the circuit performance is significantly reduced. The tradeoff between evaluation speed and storage requirements will be discussed in greated detail later in this chapter.

Most transistor sizing programs integrate *fixed* electrical models with a *fixed* optimization algorithm to improve overall running time. Typically, these systems improve only a single design metric while considering only a few predefined constraints. Unfortunately, this approach restricts the types of improvements a user can automatically pursue. Therefore, another advantage of representing the circuit performance as a set of static symbolic equations is that EPOXY can combine a *user* specified objective function and constraints at run-time.

Since all of the non-linear optimization techniques implemented within EPOXY handle the same representation of symbolic equations, optimization algorithms are substituted easily. Therefore, alternative optimization techniques can be compared and evaluated on typical design problems. A user then can select or combine the most appropriate techniques to solve the performance optimization problem of interest.

EPOXY is composed of three major parts: performance modeling, optimization, and circuit restructuring, as *Figure 5.1* illustrates. A design's performance is based on electrical information derived from a net-list and layout. From this information,

EPOXY constructs symbolic equations that model the circuit's performance. The optimization techniques use these equations to resize the transistors. If the overall design constraints are still not satisfied, structural changes to the circuit are considered (e.g., inserting buffer stages). The information describing the improved circuit is output as a new net-list and modified layout.

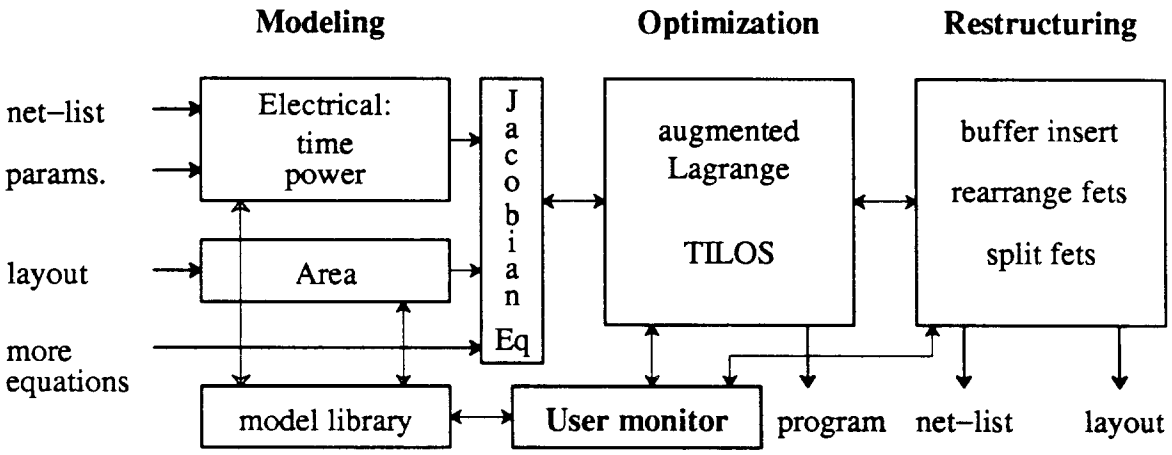


Figure 5.1. **EPOXY system architecture.** The net-list, layout, and circuit parameters are converted into symbolic equations. Also, a user can supply additional equations. Next, equations that describe the sparse Jacobian matrix, as described later in this chapter, are derived. An optimization technique is applied to these equations to produce feasible numerical values while minimizing the user specified objective function. Circuit modifications are made by altering the symbolic equations and internal net-list. The optimal numeric values (transistor sizes) and modified circuit layout are returned. Alternatively, the performance equations can be compiled (as a separate object module) for faster evaluation.

Modeling, the first section, derives the symbolic equations that compute a circuit's electrical performance and area. EPOXY currently applies an accurate area model for standard-cell layout and the distributed RC worst-case electrical model. This area model is based on a virtual grid strategy for minimally expanding compacted layout. A user can supply additional performance equations and constraints.

The next stage, *optimization*, attempts to find an assignment for the input variables of the equations to meet the constraints while minimizing the user defined objective function. The optimization problem is formulated so as to find an assignment for the transistor sizes that satisfies the specified performance requirements. However, the relationship between size and delay is non-linear even for the simple lumped RC model. Since the underlying relationship between delay and area is non-linear, the optimization problem is formulated as a non-linear program (NLP). The system can make use of several different non-linear optimization algorithms with varying convergence rates and accuracy.

The final stage, *restructuring*, alters the optimization problem by making limited circuit changes. The circuit changes are made by directly modifying the symbolic equations and transistor net-list. These circuit modifications include inserting or removing buffer stages, rearranging transistors within a pull-down or pull-up tree,

and splitting large transistors so that cell height and width can be traded off. This level handles the discrete decisions of proposing circuit alternatives while the two other levels determine the best possible implementation for this alternative.

5.1. Implementation Strategy

The chapter on modeling has presented typical equations that determine the performance of the design. To obtain quality results quickly, all optimization algorithms require rapid performance evaluation. In addition, other optimization techniques require partial derivative information. Since EPOXY determines the circuit performance and partial derivatives from equations, the primary concern in the design of the underlying data structures is to provide fast equation evaluation.

Another goal of EPOXY is to determine the effects of circuit modifications on performance and to select the best set of circuit changes. Therefore, the data structures must also support rapid modifications to reflect changes in the circuit structure. Later sub-sections will evaluate the memory requirements to achieve rapid circuit evaluation and modification.

All of the performance equations and constraints described in the modeling chapter, *Chapter 2*, are represented by signomials. Signomials are simply the sum of several terms that are composed of the product of a constant coefficient and variables raised to some integer powers [Ecke80]. Therefore, the data structures in EPOXY support this basic equation form. Also, primitives for function calls can be employed so that more accurate and complex models can be incorporated, such as the slope transistor model.

Many optimization techniques rely on information derived from a Jacobian matrix. An entry in the Jacobian matrix is defined by a partial derivative equation for a constraint equation with respect to an input variable. Another advantage of representing circuit performance by a set of symbolic signomial equations is that the equations that determine the Jacobian matrix can be *symbolically* derived in *closed form* and evaluated along with the performance equations. The expense of computing finite difference approximations of the Jacobian matrix is avoided since the partial derivative equations are *statically* derived.

The Jacobian matrix for typical design problems has many zero entries. For each performance equation, the partial derivative equations are created. Therefore the Jacobian matrix is never explicitly stored; rather it is represented by these derived partial derivative equations. This approach results in a *sparse* representation of the Jacobian matrix.

As illustrated in *Figure 5.1*, the parameter file specifies information used in the derivation of the Jacobian matrix such as declared input and output nodes, and transistors of fixed size.

The next subsection will describe how the implemented data structures help attain these system goals by considering a small example.

5.2. Example Formulation

Many of the design decisions for the internal data structures in EPOXY are best explained in context of a small example. This section will examine a static NAND gate whose inputs are buffered by single inverters. These inverters will be fixed in size to assure that the loading on the input nodes, **ba** and **bb**, is fixed. Without this realistic restriction, the resulting design problem may not have a solution, since every increase in the input gate would always improve the output response time. The task is to determine the size of the transistors for the NAND gate that offers the best improvement in the objective function while satisfying the performance constraints.

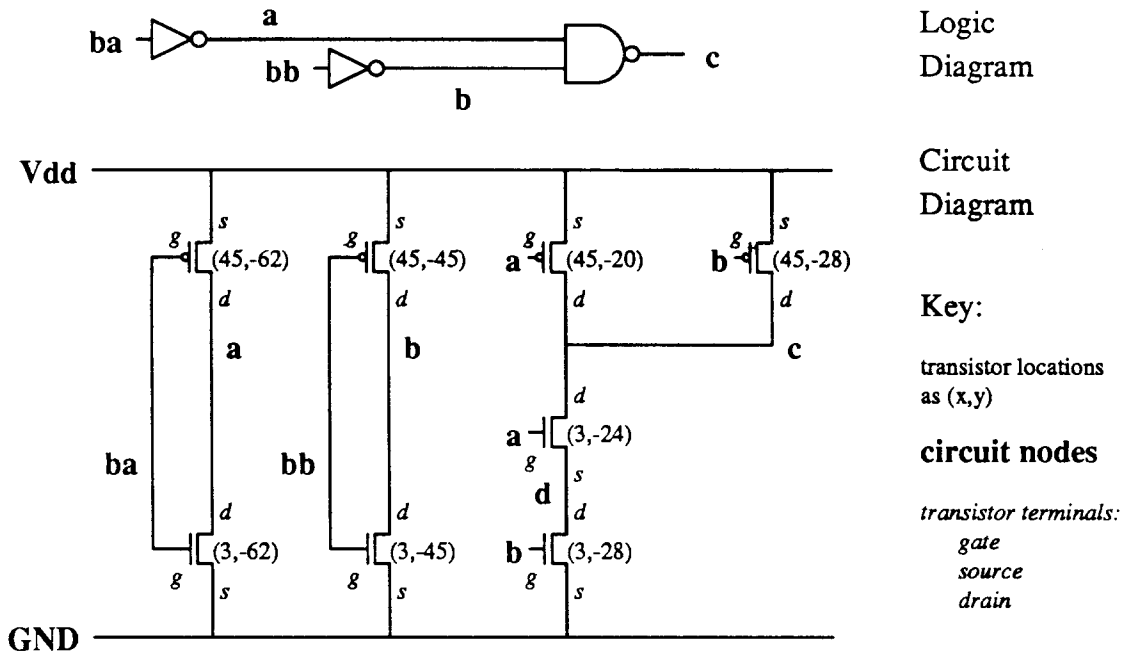


Figure 5.2. **Logic and circuit diagrams for the buffered NAND gate.** The top portion shows a logic diagram for a two-input NAND gate with inverters placed at each of its inputs. The lower circuit diagram describes a static CMOS implementation of this logic diagram. The node names are given in bold and the MOS transistor terminals are labeled in script (*gate*, *source* and *drain*). The location of the lower left hand corner for each transistor in the layout is given as (x, y) pairs.

The system diagram for EPOXY shows that the equations for the response times and dynamic power are derived from a net-list file and an input parameter file. The area equations are extracted from layout. A user can specify additional equations for EPOXY's consideration. These input files to EPOXY (the layout, parameter file, and additional equations) were created to correspond with the logic and circuit diagrams in *Figure 5.2*. However, the transistor locations (as specified by the lower left hand corner of the active area) were added to the circuit diagram after the layout was completed.

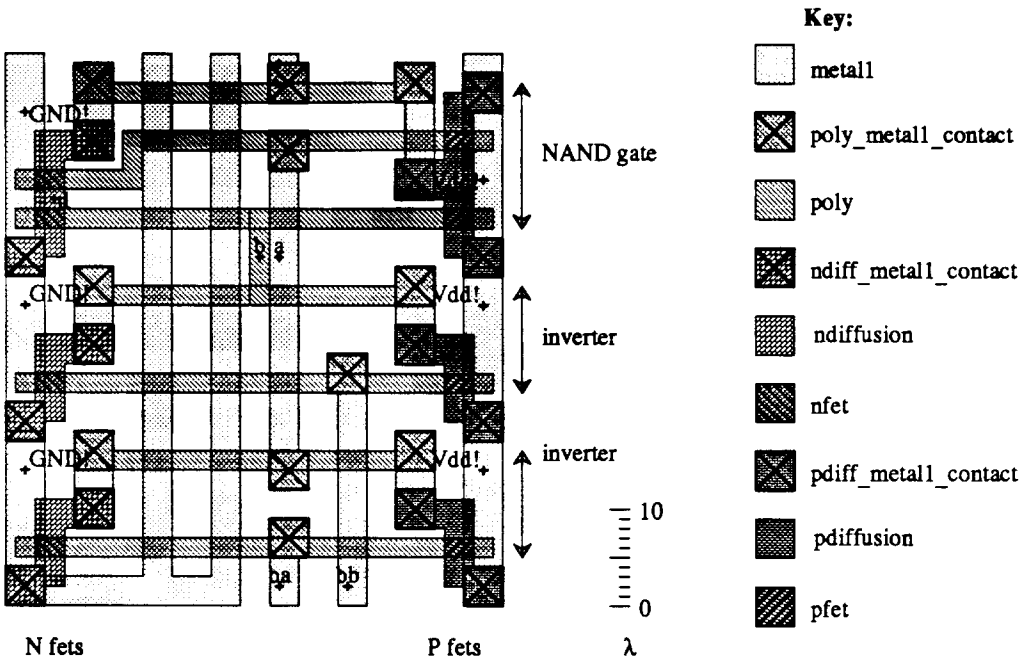


Figure 5.3. **Layout for a standard-cell CMOS buffered NAND gate.** This standard-cell style layout implements a NAND gate whose inputs are buffered by single inverters. The transistors that form each of the basic static CMOS gates are indicated by the region (on the right of the layout) bracketed by the arrows. Each of the lower two inverters drives the NAND gate above. As in a standard-cell implementation, the n-type and p-type transistors are arranged in columns as indicated. Signal names are listed near each of the corresponding small crosses. Lengths and widths for this layout are measured in λ units.

A static CMOS implementation of the buffered NAND gate is given in *Figure 5.3*. For this rotated standard-cell style layout, the n-type and p-type transistors are placed in parallel columns. The ground line runs vertically on the left side of the cell and Vdd lies on the right side. Various signals are also routed vertically using *metal1*. For this example, two of the lines are tied to ground to simulate an increased parasitic capacitance between the vertical signal lines and horizontal *polysilicon* lines that run to each of the transistors. Connections between the horizontal *polysilicon* lines and the vertical *metal1* signal lines are made using *polysilicon / metal1* contacts. Several signals of interest are labeled in the layout.

From this layout, a net-list description is extracted by using MAGIC and ext2sim [Scot85]. The following file in *Figure 5.4* results. This file also gives location of each transistor as well as its width (2 λ) and length (3 λ).

The input parameter file, as shown in *Figure 5.5*, contains information for specifying the performance optimization problem as a non-linear program. Output electrical nodes of interest (such as node c) should be declared. This requirement arises from the technique used to derive the delay times for each node. As in the timing analyzer Crystal, delay times for all transistor gates are determined by considering all conduction paths to Vdd or Gnd. Delay times for nodes whose only

File: bnand.1.sim	(continued)
| units: 70 tech: scmos	C a b 1
n ba a GND 2 3 3 –62	C Vdd c 1
p ba a Vdd 2 3 45 –62	C d GND 3
n bb b GND 2 3 3 –45	C c GND 40
p bb b Vdd 2 3 45 –45	C bb GND 9
p a Vdd c 2 3 45 –20	C b GND 40
n a c d 2 3 3 –24	C Vdd GND 45
n b d GND 2 3 3 –28	C ba GND 6
p b c Vdd 2 3 45 –28	C a GND 48

Figure 5.4. **Net-list file for the buffered CMOS NAND gate.** The net-list file was extracted from the corresponding layout for the static CMOS buffered NAND gate. The transistor lines list the transistor type, gate, source and drain nodes, transistor length and width and absolute position (lower left corner) of the transistor as x and y. Significant parasitic resistances and capacitances are also provided. The capacitance value is given by the lumped parasitic capacitance (in fempto-farads) between two nodes.

connections are transistor sources or drains are not derived since these node times are not needed in computing the node delay times of any transistor gate. For the buffer NAND gate example, the delay to node **d** in the circuit diagram is not derived, since it was not requested (by the parameter file) and this node delay is not required in the derivation of any other node delay.

File: bnand.1.prm

```
# Parameter file for bnand.1
output nodes          c

# Declare input and constants name templates
variable_style        C

constants             ^l_f ^K ^Cp_n ^F
constants             ^t[rf]_n[b[a-b]]$
constants             ^w_f[3_n62]$ ^w_f[3_n45]$
constants             ^w_f[45_n62]$ ^w_f[45_n45]$
# Const declarations override input declarations.
input variables       ^w_f

# output variables    ^t[rf]_n[c]$

# For general formulation:
output variables      ^obj$
```

Figure 5.5. **Parameter file for the buffered CMOS NAND gate.** The parameter file contains information that specifies how to set up the performance optimization as a non-linear program. Lines beginning with "#" are comments. Note that variables are described using regular expression syntax. Several variables can be declared on each line as this example illustrates.

Variables referenced by any symbolic equation that are not set by an equation (via =) must be declared as a constant or are assumed to be inputs to the Jacobian matrix. Variable names are declared using regular expression syntax as described in the UNIX‡ manual pages on ed (see also regex() [Kare84]). The advantage of this kind of description is that a large number of string matches can be made using a small descriptive string. The '^' specifies the start and '$' indicates the end of a character string. Therefore, '^l_f' matches all variables that start with 'l_f' (all transistor lengths). Characters in square brackets match the characters specified (e.g. '[rf]' matches only 'r' or 'f') or a character range when separated by a dash (e.g. '[a-b]' matches any letter, a through b inclusive). Since the square brackets have special meaning, a backslash '\' must be used to suppress the regular expression expansion (e.g.: \[matches the '[' character only).

The parameter file specifies certain variables as constant or inputs to the circuit simulator (evaluation routines) and the Jacobian matrix. A variable is a Jacobian input if it is declared as an input in the parameter file, or if the variable is constrained by an equation ($\leq$ or $\geq$). Variables assigned to a fixed value by "=" cannot be Jacobian inputs for the Jacobian matrix. Variables that are not defined as constants are assumed to be Jacobian inputs for the Jacobian matrix (default action). The circuit evaluation routines will not change the values for input or constant variables.

The objective function for the non-linear program is specified by a single variable in the parameter file. For the buffered NAND example, the variable *obj* will serve as the objective function.

The last input file contains equations that a user wishes to add explicitly. Typically, the *user* declares the objective function and overall constraints in this file. For the buffered NAND gate example, a general objective function is defined as shown in *Figure 5.6*. Note that the values of the various constants that begin with *F* dictate which variables affect the objective function. Therefore, setting *Fa* to 1 and the remaining constants to 0 will result in a strict area minimization. In addition, upper-bound constraints are declared for each of the major design metrics (area, output delay, maximum dynamic power, overall height and width). The overall cell area is the scaled (by 0.001) product of the cell width and height. Finally, the overall circuit response time is the larger of the rising and falling worst-case output times on node **c**.

Currently the area equations are generated from layout by a separated external program. EPOXY is actually a composite of several programs and program modules. The output of this external program is a set of equations as shown in the latter half of *Figure 5.6*.

‡ UNIX is a trademark of Bell Laboratories.

File: bnand.1.more.eq

```
/* Additional equations for bnand.1 */
obj >=   Fa * a + Fp * p + Ft * t + Fw * w + Fh * h;

a <=     Ka;
t <=     Kt;
p <=     Kp;
w <=     Kw;
h <=     Kh;

a >=     0.001 * w * h;
t >=     tr_n[c];
t >=     tf_n[c];

/* Area equations follow */
w_n >= 0;
```

(continued)

```
w_n >= w_f[3_n62] + -12;
w_n >= w_f[3_n45] + -12;
w_n >= w_f[3_n24] + -12;
w_n >= w_f[3_n28] + -12;
w_p >= 0;
w_p >= w_f[45_n62] + -16;
w_p >= w_f[45_n45] + -10;
w_p >= w_f[45_n20] + -16;
w_p >= w_f[45_n28] + -20;
w >=   w_n + w_p + 51;

h >=   49 + l_f[3_n62] + l_f[3_n45]
       + l_f[3_n24] + l_f[3_n28];
h >=   49 + l_f[45_n62] + l_f[45_n45]
       + l_f[45_n20] + l_f[45_n28];
```

Figure 5.6. **Additional equations for the buffered CMOS NAND gate.** A user may specify additional equations for the simulator and optimization programs. Typically, the objective function and additional performance constraints are defined. Note that C-style comments are permitted.

5.3. Deriving the Performance Equations

The equations that model the circuit performance are either derived from or provided by the input files, as described in the previous section. An advantage of this approach is that a user can estimate the circuit performance without having to actually implement it. That is, a set of equations can be created to estimate some circuit aspect. However, by default, EPOXY will automatically provide equations that model the circuit performance metrics for area, height, width, worst-case delay and maximum dynamic power as described in the chapter on modeling. The following paragraphs in this subsection will illustrate how each of these models are implemented.

Throughout this dissertation, the equations that model circuit performance can be described using several representations. In the modeling section, subscripted variables are used to describe the general format of the equations. Another variable format similar to arrays within the C language more accurately represent how these variables are actually implemented. Instead of numeric array values, descriptive symbolic names for the nodes and transistors will be employed. The following equations demonstrate the various representations of an equation that specifies the rise time of node **c** in response to a falling signal on the gate of the transistor at location (3, –28).

Equation template:

$$t_{rise,fet,node} = C_{node} * R_{fet,node,path} + C_{node} * R_{fet,node,path}$$

Example equation:

$$t_{rise,fet\ at\ [3,-28],node\ c} = C_{node\ c} * R_{fet\ at\ [3,-28],node\ c,path\ 0} + C_{node\ d} * R_{fet\ at\ [3,-28],node\ d,path\ 0};$$

C-style symbolic representation:

```
tr_f_n[3_n28][c] = C_n[c] * R_f_n_i[3_n24][c][0]
                         + C_n[d] * R_f_n_i[3_n28][d][0];
```

Each array entry for the C-style representation corresponds to a subscript in the original equation. The performance models in EPOXY use a naming convention that helps describe the type of each subscript. The base name for the variable contain letters that are separated by an underscore '_'. These letters correspond to the type of subscript (*f*et, *n*ode or *i*nteger). The position of the transistor within the layout (and its name) are described by concatenating the x and y values. Negative signs are converted to the character 'n' since C does not allow negative signs in specifying variable names. The advantage of this C-style description is that these equations can actually be compiled provided that the symbolic names for each of the subscripts are assigned unique integers. The technique of creating a C program to rapidly evaluate the performance equations will be discussed later in this chapter.

Area information is either derived directly from layout or provided as additional input equations. Area equations relate a change in transistor size to a potential change in the overall size of the layout. The first step in deriving these equations is to determine the free transistor width. Next, the transistors that can share the additional width are grouped. These transistor widths determine how much the design should be locally expanded to accommodate all the transistors within the group. This expansion factor is represented by a track width variable. For the buffered NAND gate, two such variables were generated: w_n and w_p. The effects of a change in transistor length is described by the height constraints. These equations are included in the latter half of the additional equations file, bnand.1.more.eq, presented in the previous section.

The worst-case timing model is composed of two parts: the transistor model and the model that combines the equivalent resistances and capacitances. EPOXY currently supports a linear transistor model (RC_model) and the distributed RC model (RP_model) similar to the distributed model within Crystal (a timing analyzer). The distributed RC model is based on the equations developed by J. Rubinstein and P. Penfield [Rubi83]. *Table 5.1* describes the templates used to generate the worst-case response times.

As in Crystal, EPOXY also supports flow statements given in the net-list file. These flow labels at the source or drain of a transistor restrict the signal flow direction considered in delay analysis. For example, *s=Cr:In*, specifies that the only signal path considered through the transistor is into the *s*ource as an *input*. These

RC_model():	Description
For each fet:	
w_f[f] >= Kwmin{fet-type}	Minimum size constraint
R_f[f] = Kr{type} * l_f[f] * w_f[f]^-1	Linear transistor resistance model
Cg_f[f] = Kg{type} * l_f[f] * w_f[f]	Transistor gate capacitance
Csd_f[f] = Ksd{type} * w_f[f]	Source and drain transistor capacitance
For each node:	
Cg_n[n] = Cg_f[f] + Cg_f[f] + ...	Total fet gate capacitance for the node
Csd_n[n] = Csd_f[f] + Csd_f[f] + ...	Total fet source and drain cap. for node
C_n[n] = Cp_n[n] + Cg_n[n] + Csd_n[n]	Total capacitance for the node
RP_model ():	**Description**
For each path (numbered i) to GND or Vdd:	
Ri_f_n_i[f][n][i] = R_f[f]	Resistance through fet from a node via path i
Ri_f_n_i[f][n][i] = R_f[f] + Ri_f_n_i[f1][n][i1]	Paths can references other paths
tr_f_n[f][n] >= C_n[n] * Ri_f_n_i[f][n][1]	Rise time on node due to resistance
+ C_n[n1] * Ri_f_n_i[f][n][2] + ...	and capacitance above fet
tf_f_n[f][n] >= C_n[n] * Ri_r_n_i[f][n][1]	Fall time on node due to resistance
+ C_n[n1] * Ri_r_n_i[f][n][2] + ...	and capacitance below fet
tf_n[n] >= tr_n[n1] + tr_f_n[f][n]	Fall time on node due to rise on n1 (gate of f)
tr_n[n] >= tf_n[n1] + tf_f_n[f][n]	Rise time on node due to fall on n1 (gate of f)

Table 5.1. **Equation templates for the electrical models.** This table describes the templates for creating the equations that determine the worst-case response times for the nodes of interest. The RC model generates the equations for the transistor resistances and node capacitances from the net-list (sim file). The RP model combines these resistances and capacitances to determine the worst-case node times.

labels effectively remove many pull-down or pull-up paths that do not make physical sense. In addition to classic example of networks of pass gates, there are also many such illegal pull-up paths in the *and* and *or* planes of dynamic PLA's. For example, *Figure 5.7* illustrates one pull-up path that does not occur in practice. This problem arises because of the value-independent nature of critical path analysis. For this small example, there are 9 separate pull-up paths for node **and1** by traversing every combination of transistor pairs from **and2** to **and1**. Therefore, there are a total of 18 illegal pull-up paths (9 for each of the *and* nodes) in the *and* plane alone for this small example.

Equations for maximum dynamic power are derived by creating an equation that totals the capacitance of all internal and output nodes (C_driven). The driven nodes are those that are not declared as inputs. Presumably the power to charge (or discharge) the input nodes comes from external circuitry.

The units for each of the various input metrics are determined by the selected electrical model and input values. Given an assignment of units for the input parameters, the electrical models must adhere to a consistent set of units. That is, if the resistance values are given in ohms and the capacitance values are supplied in pico-farads, then the product (all circuit node times) must be in nano-seconds. However, the electrical model provides several constants that convert between unit

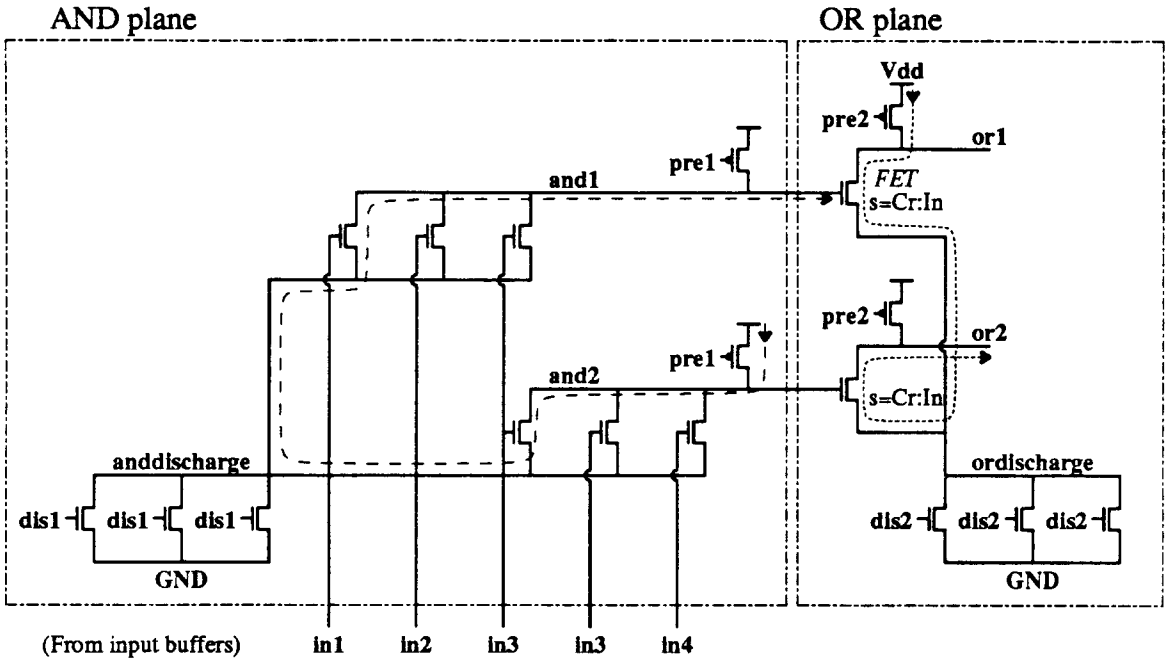


Figure 5.7. **False precharge paths in a dynamic CMOS PLA.** Due to the value-independent nature of critical path analysis, certain precharge (and discharge) paths may be considered that do not make electrical sense. For a fragment of a PLA, one such path is highlighted by the dashed line. During normal operation, node **and1** *cannot* be pulled high via node **and2** since **pre1** will be off and **anddischarge** will not allow **and2** to pull **and1** high. There are 9 such unique similar paths from **and2** to **and1** alone. A similar situation (dotted line) in the *or* plane is **prevented** by declaring the proper signal flow direction (*s=Cr:In*) on transistor *FET*.

values. For example, the constant *Krn* converts between the n-type transistor area and its resistance. The value and units for *Krn* are determined such that the appropriate unit conversion takes place. Since the transistor width and length are given in lambda units and the transistor resistance is quoted in ohms, *Krn*'s units correspond to ohms per square lambda.

5.4. Non-Linear Problem Formulation and the Jacobian Matrix

EPOXY solves the performance problem of interested by constructing and then solving a non-linear optimization program (NLP). Besides the performance equations, the partial derivatives are created since many of the optimization techniques rely on this information.

The augmented Lagrangian algorithm requires data from a Jacobian matrix to numerically solves the NLP: $\min f$ *subject to* $g(x) \leq 0$. An entry in the Jacobian matrix is specified by the partial derivative of each constraint with respect to each Jacobian input variable: $\left[\frac{\partial g_i}{\partial x_j}\right]$. For a NLP with n constraints and m Jacobian input variables, this results in a $n \times m$ Jacobian matrix.

Since the objective function and constraints are represented by symbolic signomial equations, the derivation of the signomial partial derivative equations is straightforward. However, since many of these constraints reference only a few of the Jacobian input variables, the Jacobian matrix has many zero entries. Instead of creating an explicit matrix, the Jacobian matrix will be represented by the attached partial derivatives to each of the original equations.

EPOXY's formulation of the NLP also contains several equations involving equal signs. Each of these equations could be converted into two constraint equations (one for an upper-bound and one of a corresponding lower-bound constraint). However, this approach would unnecessarily increase the size of the Jacobian matrix and the computation to evaluate it. Instead, the partial derivatives for this equation are created similar to those for constraint equations. That is, the partial derivatives for the constraint equations reference the partial derivative results of the equations with equal signs.

To simplify the code to generate the partial derivative information, EPOXY restricts the formulation of the objective function as an upper-bound constraint. In this way, the partial derivative equations for the objective function can be generated along with the other constraints, without the need for additional code to handle the objective function separately.

Rather than store the constraint equations in NLP standard form ($g(x) \leq 0$), EPOXY employs a formulation that is better suited for evaluating the performance of the circuit. Each set of constraints determines the value of a single variable. For example, these two constraint equations determine the worst-case falling time on node **c** are:

```
tf_n[c] >= tr_n[a] + tr_f_n[3_n24][c];
tf_n[c] >= tr_n[b] + tr_f_n[3_n28][c];
```

In effect, the constraints are viewed as *min* and *max* operations:

```
tf_n[c] = max ((tr_n[a] + tr_f_n[3_n24][c]), (tr_n[b] + tr_f_n[3_n28][c]));
```

Therefore, this formulation determines performance of the circuit given a particular assignment of the input variables (transistor sizes and input node times) by simply evaluating each of the constraint equations.

The conversion of the constraints between a performance evaluation formulation and the NLP standard form is quite easy. For example, the first constraint is converted to standard form:

```
tf_n[c] >= tr_n[a] + tr_f_n[3_n24][c];        (performance constraint)
tr_n[a] + tr_f_n[3_n24][c] - tf_n[c] <= 0;    (standard form)
```

The single variable on the left of the constraint is moved to the right hand side (by subtraction). This leaves the necessary 0 on one size of the equation. If necessary, the equation is converted from a upper-bound to a lower-bound constraint by simply multiplying each side by −1.

Figure 5.8 shows the gradient equations below each of the performance equations for the buffered CMOS NAND gate example. The results of the partial derivative equations are stored in a special variable *D*. The first subscript for this derivative variable is the variable name whose value is determined by the corresponding constraint. The second subscript is an input variable (Jacobian input) to the constraint equations. Therefore, the results of the partial derivative constraint equations are stored in a two-dimensional sparse array, whose indexes are the constraint equation and Jacobian input variable. Constraints without comments are constant with respect to the Jacobian matrix.

```
min obj

subject to:

w_f[45_n28] >= Kwminp;          (Jacobian constraint 0)      Width of p-type fet in nand gate
w_f[3_n28] >= Kwminn;           (Jacobian constraint 1)      Width of n-type fet in nand gate
w_f[3_n24] >= Kwminn;           (Jacobian constraint 2)      Width of n-type fet in nand gate
w_f[45_n20] >= Kwminp;          (Jacobian constraint 3)      Width of p-type fet in nand gate
w_f[45_n45] >= Kwminp;                                       Fixed size fets for input inverters
w_f[3_n45] >= Kwminn;                                        Fixed size fets for input inverters
w_f[45_n62] >= Kwminp;                                       Fixed size fets for input inverters
w_f[3_n62] >= Kwminn;                                        Fixed size fets for input inverters
h <= Kh;                        (Jacobian constraint 4)      cell height
h >= l_f[3_n28] + l_f[3_n24] + l_f[3_n45] + l_f[3_n62] + 49;  (Jacobian constraint 5)
h >= l_f[45_n28] + l_f[45_n20] + l_f[45_n45] + l_f[45_n62] + 49;  (Jacobian constraint 6)
w_n >= 0.0;                     (Jacobian constraint 7)      N-type fet track width
w_n >= -12 + w_f[3_n62];        (Jacobian constraint 8)
w_n >= -12 + w_f[3_n45];        (Jacobian constraint 9)
w_n >= -12 + w_f[3_n24];        (Jacobian constraint 10)
    D[w_n][w_f[3_n24]] = 1;                                  Partial derivative of w_n to fet (3,-24) width
w_n >= -12 + w_f[3_n28];        (Jacobian constraint 11)
    D[w_n][w_f[3_n28]] = 1;
Csd_f[3_n62] = Ksdn * w_f[3_n62];                            Source or drain cap. for fet (3,-62)
Cg_f[3_n62] = Kgn * l_f[3_n62] * w_f[3_n62];                 Gate cap. for fet (3,-62)
R_f[3_n62] = Krn * l_f[3_n62] * (1/w_f[3_n62]);              Resistance of fet (3,-62)
w_p >= 0.0;                     (Jacobian constraint 12)     P-type fet track width
w_p >= -16 + w_f[45_n62];       (Jacobian constraint 13)
w_p >= -10 + w_f[45_n45];       (Jacobian constraint 14)
w_p >= -16 + w_f[45_n20];       (Jacobian constraint 15)
    D[w_p][w_f[45_n20]] = 1;
w_p >= -20 + w_f[45_n28];       (Jacobian constraint 16)
    D[w_p][w_f[45_n28]] = 1;
Csd_f[45_n62] = Ksdp * w_f[45_n62];
Cg_f[45_n62] = Kgp * l_f[45_n62] * w_f[45_n62];
R_f[45_n62] = Krp * l_f[45_n62] * (1/w_f[45_n62]);
Csd_f[3_n45] = Ksdn * w_f[3_n45];
Cg_f[3_n45] = Kgn * l_f[3_n45] * w_f[3_n45];
R_f[3_n45] = Krn * l_f[3_n45] * (1/w_f[3_n45]);
Csd_f[45_n45] = Ksdp * w_f[45_n45];
```

```
Cg_f[45_n45]  = Kgp * I_f[45_n45] * w_f[45_n45];

R_f[45_n45]  = Krp * I_f[45_n45] * (1/w_f[45_n45]);

Csd_f[45_n20]  = Ksdp * w_f[45_n20];                          Width of fet at (45,-20) is an input
     D[Csd_f[45_n20]][w_f[45_n20]]  = Ksdp;                   So, there's a partial derivative

Cg_f[45_n20]  = Kgp * I_f[45_n20] * w_f[45_n20];
     D[Cg_f[45_n20]][w_f[45_n20]]  = I_f[45_n20] * Kgp;

R_f[45_n20]  = Krp * I_f[45_n20] * (1/w_f[45_n20]);
     D[R_f[45_n20]][w_f[45_n20]]  = -1 * (1/(w_f[45_n20]*w_f[45_n20])) * I_f[45_n20] * Krp;

Csd_f[3_n24]  = Ksdn * w_f[3_n24];
     D[Csd_f[3_n24]][w_f[3_n24]]  = Ksdn;

Cg_f[3_n24]  = Kgn * I_f[3_n24] * w_f[3_n24];
     D[Cg_f[3_n24]][w_f[3_n24]] = I_f[3_n24] * Kgn;

R_f[3_n24]  = Km * I_f[3_n24] * (1/w_f[3_n24]);
     D[R_f[3_n24]][w_f[3_n24]]  = -1 * (1/(w_f[3_n24]*w_f[3_n24])) * I_f[3_n24] * Km;

Csd_f[3_n28]  = Ksdn * w_f[3_n28];
     D[Csd_f[3_n28]][w_f[3_n28]]  = Ksdn;

Cg_f[3_n28]  = Kgn * I_f[3_n28] * w_f[3_n28];
     D[Cg_f[3_n28]][w_f[3_n28]] = I_f[3_n28] * Kgn;

R_f[3_n28]  = Km * I_f[3_n28] * (1/w_f[3_n28]);
     D[R_f[3_n28]][w_f[3_n28]]  = -1 * (1/(w_f[3_n28]*w_f[3_n28])) * I_f[3_n28] * Km;

Csd_f[45_n28]  = Ksdp * w_f[45_n28];
     D[Csd_f[45_n28]][w_f[45_n28]]  = Ksdp;

Cg_f[45_n28]  = Kgp * I_f[45_n28] * w_f[45_n28];
     D[Cg_f[45_n28]][w_f[45_n28]]  = I_f[45_n28] * Kgp;

R_f[45_n28]  = Krp * I_f[45_n28] * (1/w_f[45_n28]);
     D[R_f[45_n28]][w_f[45_n28]]  = -1 * (1/(w_f[45_n28]*w_f[45_n28])) * I_f[45_n28] * Krp;

R_f_n_i[45_n28][c][0]  = R_f[45_n28];                         Res. of node c to Vdd through fet (45,-28)
     D[R_f_n_i[45_n28][c][0]][w_f[45_n28]] = D[R_f[45_n28]][w_f[45_n28]];

Cg_n[b]  = Cg_f[3_n28] + Cg_f[45_n28];                        Total gate cap. on node b
     D[Cg_n[b]][w_f[3_n28]]  = D[Cg_f[3_n28]][w_f[3_n28]];
     D[Cg_n[b]][w_f[45_n28]]  = D[Cg_f[45_n28]][w_f[45_n28]];

Csd_n[c]  = Csd_f[45_n20] + Csd_f[3_n24] + Csd_f[45_n28];     Total source/drain cap. on node b
     D[Csd_n[c]][w_f[45_n20]]  = D[Csd_f[45_n20]][w_f[45_n20]];
     D[Csd_n[c]][w_f[3_n24]]  = D[Csd_f[3_n24]][w_f[3_n24]];
     D[Csd_n[c]][w_f[45_n28]]  = D[Csd_f[45_n28]][w_f[45_n28]];

R_f_n_i[3_n28][d][0]  = R_f[3_n28];
     D[R_f_n_i[3_n28][d][0]][w_f[3_n28]]  = D[R_f[3_n28]][w_f[3_n28]];

Csd_n[d]  = Csd_f[3_n24] + Csd_f[3_n28];
     D[Csd_n[d]][w_f[3_n24]]  = D[Csd_f[3_n24]][w_f[3_n24]];
     D[Csd_n[d]][w_f[3_n28]]  = D[Csd_f[3_n28]][w_f[3_n28]];

Cg_n[a]  = Cg_f[45_n20] + Cg_f[3_n24];
     D[Cg_n[a]][w_f[45_n20]]  = D[Cg_f[45_n20]][w_f[45_n20]];
     D[Cg_n[a]][w_f[3_n24]]  = D[Cg_f[3_n24]][w_f[3_n24]];

R_f_n_i[45_n20][c][0]  = R_f[45_n20];
     D[R_f_n_i[45_n20][c][0]][w_f[45_n20]] = D[R_f[45_n20]][w_f[45_n20]];

R_f_n_i[45_n45][b][0]  = R_f[45_n45];

Cg_n[bb]  = Cg_f[3_n45] + Cg_f[45_n45];

Csd_n[b]  = Csd_f[3_n45] + Csd_f[45_n45];

R_f_n_i[3_n45][b][0]  = R_f[3_n45];

R_f_n_i[45_n62][a][0]  = R_f[45_n62];
```

```
Cg_n[ba]  = Cg_f[3_n62] + Cg_f[45_n62];

Csd_n[a]  = Csd_f[3_n62] + Csd_f[45_n62];

w <= Kw;                          (Jacobian constraint 17)      Cell width
w >=  51  + w_p + w_n;            (Jacobian constraint 18)
     D[w][w_p] =  1;
     D[w][w_n] =  1;

R_f_n_i[3_n62][a][0]  = R_f[3_n62];

a <= Ka;                          (Jacobian constraint 19)      Cell area
a >=  0.001 * h * w;              (Jacobian constraint 20)
     D[a][h]    =  0.001 * w;
     D[a][w]    =  0.001 * h;

C_n[a]  = Csd_n[a] + Cg_n[a] + Cp_n[a];                          Total cap. on node a
     D[C_n[a]][w_f[45_n20]]    = D[Cg_n[a]][w_f[45_n20]];
     D[C_n[a]][w_f[3_n24]] = D[Cg_n[a]][w_f[3_n24]];

C_n[ba]  = Cg_n[ba] + Cp_n[ba];

C_n[b]  = Csd_n[b] + Cg_n[b] + Cp_n[b];
     D[C_n[b]][w_f[3_n28]] = D[Cg_n[b]][w_f[3_n28]];
     D[C_n[b]][w_f[45_n28]]    = D[Cg_n[b]][w_f[45_n28]];

C_n[bb]  = Cg_n[bb] + Cp_n[bb];

C_n[d]  = Csd_n[d] + Cp_n[d];
     D[C_n[d]][w_f[3_n24]] = D[Csd_n[d]][w_f[3_n24]];
     D[C_n[d]][w_f[3_n28]] = D[Csd_n[d]][w_f[3_n28]];

R_f_n_i[3_n24][c][0]  = R_f[3_n24] + R_f_n_i[3_n28][d][0];
     D[R_f_n_i[3_n24][c][0]][w_f[3_n24]]    = D[R_f[3_n24]][w_f[3_n24]];
     D[R_f_n_i[3_n24][c][0]][w_f[3_n28]]    = D[R_f_n_i[3_n28][d][0]][w_f[3_n28]];

C_n[c]  = Csd_n[c] + Cp_n[c];
     D[C_n[c]][w_f[45_n20]]    = D[Csd_n[c]][w_f[45_n20]];
     D[C_n[c]][w_f[3_n24]] = D[Csd_n[c]][w_f[3_n24]];
     D[C_n[c]][w_f[45_n28]]    = D[Csd_n[c]][w_f[45_n28]];

C_driven  = C_n[d] + C_n[c] + C_n[b] + C_n[a];                   Total cap. this circuit drives
     D[C_driven][w_f[45_n20]]  = D[C_n[a]][w_f[45_n20]] + D[C_n[c]][w_f[45_n20]];
     D[C_driven][w_f[3_n24]]   = D[C_n[a]][w_f[3_n24]] + D[C_n[c]][w_f[3_n24]] + D[C_n[d]][w_f[3_n24]];
     D[C_driven][w_f[3_n28]]   = D[C_n[b]][w_f[3_n28]] + D[C_n[d]][w_f[3_n28]];
     D[C_driven][w_f[45_n28]]  = D[C_n[b]][w_f[45_n28]] + D[C_n[c]][w_f[45_n28]];

tf_f_n[45_n20][c]  = C_n[c] * R_f_n_i[45_n20][c][0];             Rising delay on node c due to falling input
     D[tf_f_n[45_n20][c]][w_f[45_n20]]= D[R_f_n_i[45_n20][c][0]][w_f[45_n20]] * C_n[c];to fet (45,-20)
                              + D[C_n[c]][w_f[45_n20]] * R_f_n_i[45_n20][c][0];
     D[tf_f_n[45_n20][c]][w_f[3_n24]]  = D[C_n[c]][w_f[3_n24]] * R_f_n_i[45_n20][c][0];
     D[tf_f_n[45_n20][c]][w_f[45_n28]]= D[C_n[c]][w_f[45_n28]] * R_f_n_i[45_n20][c][0];

tf_f_n[45_n28][c]  = C_n[c] * R_f_n_i[45_n28][c][0];
     D[tf_f_n[45_n28][c]][w_f[45_n20]]= D[C_n[c]][w_f[45_n20]] * R_f_n_i[45_n28][c][0];
     D[tf_f_n[45_n28][c]][w_f[3_n24]]  = D[C_n[c]][w_f[3_n24]] * R_f_n_i[45_n28][c][0];
     D[tf_f_n[45_n28][c]][w_f[45_n28]]= D[R_f_n_i[45_n28][c][0]][w_f[45_n28]] * C_n[c]
                              + D[C_n[c]][w_f[45_n28]] * R_f_n_i[45_n28][c][0];

tr_f_n[3_n24][c]  = C_n[c] * R_f_n_i[3_n24][c][0];               Falling delay on node c due to rising input
     D[tr_f_n[3_n24][c]][w_f[45_n20]]  = D[C_n[c]][w_f[45_n20]] * R_f_n_i[3_n24][c][0];to fet (3,-24)
     D[tr_f_n[3_n24][c]][w_f[3_n24]]   = D[R_f_n_i[3_n24][c][0]][w_f[3_n24]] * C_n[c]
                              + D[C_n[c]][w_f[3_n24]] * R_f_n_i[3_n24][c][0];
     D[tr_f_n[3_n24][c]][w_f[3_n28]]   = D[R_f_n_i[3_n24][c][0]][w_f[3_n28]] * C_n[c];
     D[tr_f_n[3_n24][c]][w_f[45_n28]]  = D[C_n[c]][w_f[45_n28]] * R_f_n_i[3_n24][c][0];
```

```
tr_f_n[3_n28][c]  = C_n[c] * R_f_n_i[3_n24][c][0] + C_n[d] * R_f_n_i[3_n28][d][0];
     D[tr_f_n[3_n28][c]][w_f[45_n20]] = D[C_n[c]][w_f[45_n20]] * R_f_n_i[3_n24][c][0];
     D[tr_f_n[3_n28][c]][w_f[3_n24]]  = D[C_n[d]][w_f[3_n24]] * R_f_n_i[3_n28][d][0]
                                      + D[R_f_n_i[3_n24][c][0]][w_f[3_n24]] * C_n[c]
                                      + D[C_n[c]][w_f[3_n24]] * R_f_n_i[3_n24][c][0];
     D[tr_f_n[3_n28][c]][w_f[3_n28]]  = D[R_f_n_i[3_n28][d][0]][w_f[3_n28]] * C_n[d]
                                      + D[C_n[d]][w_f[3_n28]] * R_f_n_i[3_n28][d][0]
                                      + D[R_f_n_i[3_n24][c][0]][w_f[3_n28]] * C_n[c];
     D[tr_f_n[3_n28][c]][w_f[45_n28]] = D[C_n[c]][w_f[45_n28]] * R_f_n_i[3_n24][c][0];

tf_f_n[45_n45][b]  = C_n[b] * R_f_n_i[45_n45][b][0];
     D[tf_f_n[45_n45][b]][w_f[3_n28]] = D[C_n[b]][w_f[3_n28]] * R_f_n_i[45_n45][b][0];
     D[tf_f_n[45_n45][b]][w_f[45_n28]]= D[C_n[b]][w_f[45_n28]] * R_f_n_i[45_n45][b][0];

tr_f_n[3_n45][b]  = C_n[b] * R_f_n_i[3_n45][b][0];
     D[tr_f_n[3_n45][b]][w_f[3_n28]]  = D[C_n[b]][w_f[3_n28]] * R_f_n_i[3_n45][b][0];
     D[tr_f_n[3_n45][b]][w_f[45_n28]] = D[C_n[b]][w_f[45_n28]] * R_f_n_i[3_n45][b][0];

tf_f_n[45_n62][a]  = C_n[a] * R_f_n_i[45_n62][a][0];
     D[tf_f_n[45_n62][a]][w_f[45_n20]]= D[C_n[a]][w_f[45_n20]] * R_f_n_i[45_n62][a][0];
     D[tf_f_n[45_n62][a]][w_f[3_n24]] = D[C_n[a]][w_f[3_n24]] * R_f_n_i[45_n62][a][0];

tr_f_n[3_n62][a]  = C_n[a] * R_f_n_i[3_n62][a][0];
     D[tr_f_n[3_n62][a]][w_f[45_n20]] = D[C_n[a]][w_f[45_n20]] * R_f_n_i[3_n62][a][0];
     D[tr_f_n[3_n62][a]][w_f[3_n24]]  = D[C_n[a]][w_f[3_n24]] * R_f_n_i[3_n62][a][0];

tf_n[a]  = tr_n[ba] + tr_f_n[3_n62][a];                    Falling time for node a due to rise
     D[tf_n[a]][w_f[45_n20]]  = D[tr_f_n[3_n62][a]][w_f[45_n20]];   on node ba
     D[tf_n[a]][w_f[3_n24]] = D[tr_f_n[3_n62][a]][w_f[3_n24]];

tr_n[a]  = tf_n[ba] + tf_f_n[45_n62][a];                   Rising time for node a due to fall
     D[tr_n[a]][w_f[45_n20]]  = D[tf_f_n[45_n62][a]][w_f[45_n20]];   on node ba
     D[tr_n[a]][w_f[3_n24]] = D[tf_f_n[45_n62][a]][w_f[3_n24]];

tf_n[b]  = tr_n[bb] + tr_f_n[3_n45][b];
     D[tf_n[b]][w_f[3_n28]] = D[tr_f_n[3_n45][b]][w_f[3_n28]];
     D[tf_n[b]][w_f[45_n28]]  = D[tr_f_n[3_n45][b]][w_f[45_n28]];

tr_n[b]  = tf_n[bb] + tf_f_n[45_n45][b];
     D[tr_n[b]][w_f[3_n28]] = D[tf_f_n[45_n45][b]][w_f[3_n28]];
     D[tr_n[b]][w_f[45_n28]]  = D[tf_f_n[45_n45][b]][w_f[45_n28]];

p <= Kp;                          (Jacobian constraint 21)     Total dynamic power
p >=  12.5 * C_driven;            (Jacobian constraint 22)
     D[p][w_f[45_n20]]  =  12.5 * D[C_driven][w_f[45_n20]];
     D[p][w_f[3_n24]]   =  12.5 * D[C_driven][w_f[3_n24]];
     D[p][w_f[3_n28]]   =  12.5 * D[C_driven][w_f[3_n28]];
     D[p][w_f[45_n28]]  =  12.5 * D[C_driven][w_f[45_n28]];

tf_n[c] >= tr_n[a] + tr_f_n[3_n24][c];  (Jacobian constraint 23)
     D[tf_n[c]][w_f[45_n20]]  = D[tr_f_n[3_n24][c]][w_f[45_n20]] + D[tr_n[a]][w_f[45_n20]];
     D[tf_n[c]][w_f[3_n24]] = D[tr_f_n[3_n24][c]][w_f[3_n24]] + D[tr_n[a]][w_f[3_n24]];
     D[tf_n[c]][w_f[3_n28]] = D[tr_f_n[3_n24][c]][w_f[3_n28]];
     D[tf_n[c]][w_f[45_n28]]  = D[tr_f_n[3_n24][c]][w_f[45_n28]];

tf_n[c] >= tr_n[b] + tr_f_n[3_n28][c];  (Jacobian constraint 24)
     D[tf_n[c]][w_f[45_n20]]  = D[tr_f_n[3_n28][c]][w_f[45_n20]];
     D[tf_n[c]][w_f[3_n24]] = D[tr_f_n[3_n28][c]][w_f[3_n24]];
     D[tf_n[c]][w_f[3_n28]] = D[tr_f_n[3_n28][c]][w_f[3_n28]] + D[tr_n[b]][w_f[3_n28]];
     D[tf_n[c]][w_f[45_n28]]  = D[tr_f_n[3_n28][c]][w_f[45_n28]] + D[tr_n[b]][w_f[45_n28]];
```

```
tr_n[c] >= tf_n[a] + tf_f_n[45_n20][c]; (Jacobian constraint 25)
     D[tr_n[c]][w_f[45_n20]]     = D[tf_f_n[45_n20][c]][w_f[45_n20]] + D[tf_n[a]][w_f[45_n20]];
     D[tr_n[c]][w_f[3_n24]] = D[tf_f_n[45_n20][c]][w_f[3_n24]] + D[tf_n[a]][w_f[3_n24]];
     D[tr_n[c]][w_f[45_n28]]     = D[tf_f_n[45_n20][c]][w_f[45_n28]];

tr_n[c] >= tf_n[b] + tf_f_n[45_n28][c]; (Jacobian constraint 26)
     D[tr_n[c]][w_f[45_n20]]     = D[tf_f_n[45_n28][c]][w_f[45_n20]];
     D[tr_n[c]][w_f[3_n24]] = D[tf_f_n[45_n28][c]][w_f[3_n24]];
     D[tr_n[c]][w_f[3_n28]] = D[tf_n[b]][w_f[3_n28]];
     D[tr_n[c]][w_f[45_n28]]     = D[tf_f_n[45_n28][c]][w_f[45_n28]] + D[tf_n[b]][w_f[45_n28]];

t <= Kt;                              (Jacobian constraint 27)      Delay of the entire circuit
t >= tr_n[c];                         (Jacobian constraint 28)
     D[t][tr_n[c]] =  1;
t >= tf_n[c];                         (Jacobian constraint 29)
     D[t][tf_n[c]] =  1;

obj >= h * Fh + w * Fw + t * Ft + p * Fp + a * Fa;     (constraint for objective function)
     D[obj][h]   = Fh;
     D[obj][w]   = Fw;
     D[obj][t]   = Ft;
     D[obj][p]   = Fp;
     D[obj][a]   = Fa;
```

Figure 5.8. **NLP formulation for the buffered CMOS NAND gate.** EPOXY generates these performance equations and partial derivative equations from the input files to formulate the performance design problem as a non-linear program. If an equation or constraint references an Jacobian input variable, the corresponding partial derivative equations follow (D[...][...] = ...). These equations (and partial derivative equations) are ordered such that they may be consecutively (linearly) evaluated. That is, a variable's value is defined before it is needed. Each constraint equation is assigned a unique number corresponding to the first entry in the sparse Jacobian array: J[constraints][inputs].

The partial derivative variables, D[...][...], do *not exactly* represent the sparse representation of the Jacobian matrix because the original equations were not in standard NLP form. However, the creation of a Jacobian matrix from these variables is also straightforward. The technique is similar to the conversion of the original performance equations to NLP standard form. For each of the performance equations, the partial derivative for the left-hand side variable should be added (the standard D[x][x] = 1; equation). The partial derivative results of the corresponding upper-bound constraint are negated since the original equation would have been multiplied by -1. The code that determines the search direction modifies the partial derivative data to conform with the standard NLP formulation.

The size of the Jacobian matrix is the number of constraints times the number of Jacobian inputs. *Figure 5.8* provides the unique numbering for the constraint equations. *Table 5.2* describes the Jacobian input vector for the Jacobian matrix.

EPOXY also provides several statistics when circuits are evaluated. *Table 5.3* describes several circuits and the number of constraint equations and variables created to evaluate their performance and Jacobian matrix. The "eqs" entry in the table counts *each* constraint equation separately. For example, the *w_n* variable requires 5 performance equations to determine its value for the buffered NAND example (bnand.1). The Jacobian matrix is represented by deriving the partial

The sparse Jacobian matrix: J [constraint][input]		
Jacobian input vector:		
Input	Variable	Description
0	w_p	extra width needed by the channel of p-type transistors
1	w_n	extra width needed by the channel of n-type transistors
2	h	overall cell height
3	w	overall cell width
4	t	overall response time for the circuit
5	p	maximum dynamic power
6	a	overall cell area
7	tr_n[c]	worst-case rising time for node c
8	tf_n[c]	worst-case falling time for node c
9	w_f[45_n20]	width of p-type transistor at (45, −20)
10	w_f[3_n24]	width of n-type fet at (3, −24)
11	w_f[3_n28]	width of n-type fet at (3, −28)
12	w_f[45_n28]	width of p-type fet at (45, −28)

Table 5.2. **Input vector for the Jacobian matrix.** The entries for the input vector of the Jacobian matrix are described in this table. Each Jacobian input is assigned a unique number by EPOXY. Note that several intermediate variables (in the performance analysis) are inputs to the Jacobian matrix (e.g.: w_p). Therefore, the Jacobian input vector can include many more variables than the performance analysis input vector. The previous figure provides the numbering for the constraints.

Circuit Statistics									
Circuit	Nodes	Fets	Performance		Jacobian matrix				
			vars	eqs	vars	eqs	Jin	cnstr.	%nzJe
bnand.1	8	8	113	94	96	103	13	30	26.4
inv.8	11	16	191	169	373	387	21	44	41.9
inv.10	13	20	231	207	513	531	25	52	40.8
rand20	16	20	236	211	280	309	25	60	20.6
adder.1	22	34	359	360	898	1242	59	129	16.3
adder.8	131	224	2249	2366	5678	7953	437	947	0.2

Table 5.3. **EPOXY Circuit Statistics.** This table gives the number of nodes and transistors (fets) for several example circuits. EPOXY constructs a number of equations (eqs) and variables (vars) to evaluate circuit performance using the worst-case distributed RC electrical model and the standard-cell area model. Partial derivatives (variables and equations) for these performance equations are derived to represent the Jacobian matrix. The table describes the size of the Jacobian matrix by the number of Jacobian input variables (Jin) and the number of constraints (cnstr.). The last column is a measure of the sparsity of the Jacobian matrix as the maximum percentage of entries (%nzJe) that can be non-zero (i.e., have defining equations).

derivative equations from the performance equations. The number of variables and equations to represent the Jacobian matrix are also listed. Next, the number of Jacobian inputs and constraints describe the size of the Jacobian matrix. The last column illustrates that only a small number of entries in the Jacobian matrix can assume a value other than 0 (i.e., the Jacobian matrix is sparse). Circuits that contain inverters require fewer constraint equations than circuits with multi-input gates. Therefore, circuits with more inverters require more equations to represent a smaller sparse Jacobian matrix. The larger the circuit, the sparser the matrix will become.

The TILOS-style algorithm requires different partial derivative information. As in circuit analysis, the effects on upper-bound constraints by a change on the input variables must be determined. The lower-bound constraints are eliminated by the *max* operation. Therefore, the information for the TILOS-style algorithm is derived from the partial derivative information in the Jacobian matrix by using the chain rule. For example, the partial derivative of the cell width with respect to one of the input variables, the width of the transistor at (3, −24), is:

$$\frac{\partial w}{\partial w_{fet\ at\ \left[3,-24\right]}} = \frac{\partial w}{\partial w_{n\ channel}} \frac{\partial w_{n\ channel}}{\partial w_{fet\ at\ \left[3,-24\right]}}$$

In C-style notation:

```
D[w][w_f[3_n24]] = D[w][w_n] * D[w_n][w_f[3_n24]]
```

5.5. Handling Sequential Circuits

The previous section presented equations that determine the performance of a buffered CMOS NAND gate. These equations were sequentially ordered so that they can be quickly re-evaluated by a single pass. However, if the circuit contains feedback, these equations cannot be linearly ordered. In addition, the partial derivative equations cannot be determined by a single evaluation pass either.

Timing analyzers solve this problem by traversing the feedback circuitry several times, until the response times on all the nodes can be fixed. In particular, Crystal propagates critical delay paths from the input nodes in a depth first manner. Delay paths are restricted to traversing circuitry only once. A critical path is terminated if a previous critical path can set the node response times to a larger value (a later response time). In this way, signals traverse the feedback circuitry at most once.

Crystal's restriction that critical paths can traverse the same circuitry at most once makes physical sense as well. Multiple cycles through the same circuit indicate the presence of an oscillator.

Rarely do performance optimization tools consider circuitry with feedback because of the implementation complexity in resolving feedback and the additional computation involved in evaluating the circuit performance when compared to combinational logic. While not much can be done to reduce the inherent computation, EPOXY's use of equations helps resolve the implementation complexity. Circuits with feedback will produce equations that reference each other cyclically. The

algorithm within EPOXY removes cycles of equations by replacing them with another set of equations that involve traversing all of these equations at most once. Since EPOXY deals with feedback at a high level of abstraction (as equations), EPOXY analyzes circuits with feedback *independent* of the electrical models chosen.

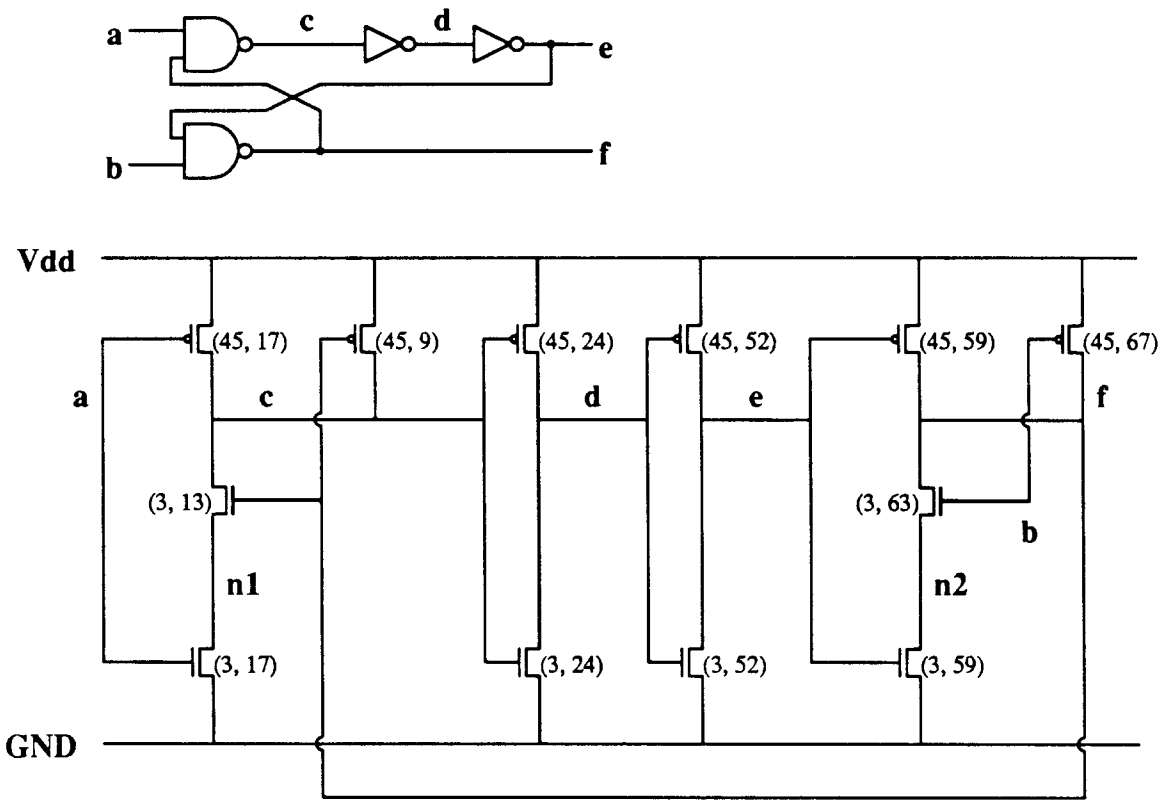


Figure 5.9. **Logic and circuit diagrams for an RS-latch and a buffer.** The upper logic diagram shows an NAND gate implementation of a RS-latch with two extra inverters (a buffer) inserted before one of its outputs. The lower circuit diagram describes a CMOS implementation of this logic diagram using static logic gates. The node names are given in bold and the location of the lower left hand corner for each transistor in the layout is given by (x, y) pairs.

The details of how the performance equations for circuits with feedback are modified is best illustrated by another example. *Figure 5.9* shows an RS-latch with a buffer inserted before one of its outputs. The inverter pair was inserted to demonstrate the implications (additional equations) of simple circuit modifications on a design with feedback. Constraints for the overall response time t are added. The response time is the maximum of the rise and fall times on nodes **e** and **f**.

While examining all the delay paths through the portion of the circuit that involves feedback, the following potential critical paths result as illustrated in *Figure 5.10*. All critical paths through the original circuit are represented on this critical path diagram. Equations that define the values for variables in the graph are

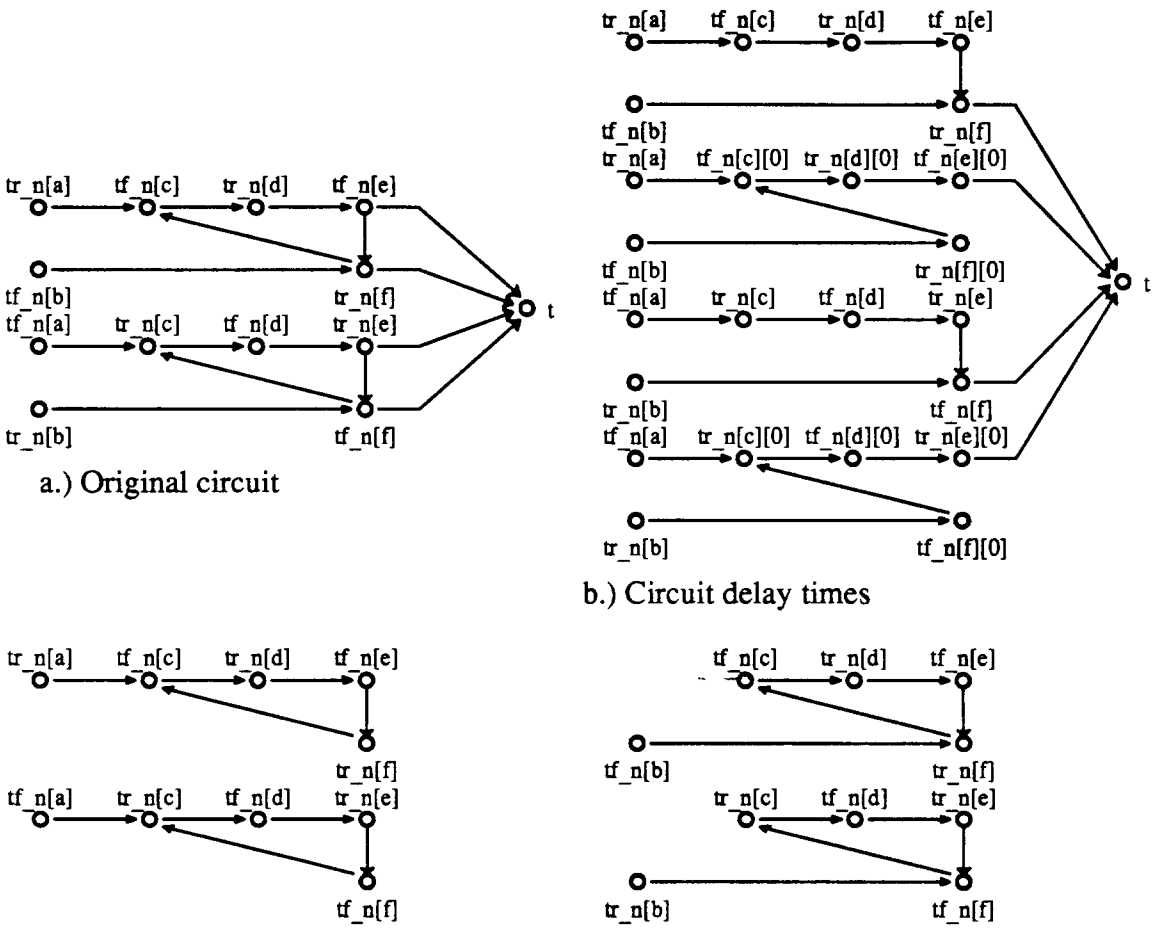


Figure 5.10. **Critical delay paths through the RS-latch and buffer.** These graphs illustrate *all* of the critical paths through the RS-latch and buffer. A node in the original circuit produces two nodes in these graphs corresponding to the worst-case rising and falling times. The arcs show the signal flow within the circuit. They also indicate that the equation for the variable at the head of the arc references the variable at the tail of the arc. (E.g.: the equation for tf_n[c] references the variable tr_n[a].) The graph for the original circuit (part a) contains two cycles. These cycles are removed by traversing each node only once for each critical path (part b). The 4 cycles that are removed represent the hold-time requirements on the RS-latch and buffer circuit (part c).

replaced by another set of equations that represent all the critical paths as illustrated in part b. The variable t is assigned the worst-case response time for all paths through the circuit with feedback by traversing each circuit component no more than once. Part c contains the critical path cycles that were removed from the graph for part b. These cycles represent the hold times necessary to latch a value into the RS-latch and buffer. Currently, EPOXY does not generate the equations for the hold times, since the overall delay through the additional circuit is much larger than the hold time for the latch.

Equations for the critical delay paths in the original circuit.	**Equations for the expanded critical delay paths.**
tf_n[c] ≥ tr_n[a] + tr_f_n[3_17][c];	tr_n[c] = tf_n[a] + tf_f_n[45_17][c];
tf_n[c] ≥ tr_n[f] + tr_f_n[3_13][c];	tf_n[f][0] = tr_n[b] + tr_f_n[3_63][f];
tr_n[d] = tf_n[c] + tf_f_n[45_24][d];	tr_n[c][0] ≥ tf_n[f][0] + tf_f_n[45_9][c];
tf_n[e] = tr_n[d] + tr_f_n[3_52][e];	tr_n[c][0] ≥ tf_n[a] + tf_f_n[45_17][c];
tr_n[f] ≥ tf_n[e] + tf_f_n[45_59][f];	tf_n[d] = tr_n[c] + tr_f_n[3_24][d];
tr_n[f] ≥ tf_n[b] + tf_f_n[45_67][f];	tr_n[e] = tf_n[d] + tf_f_n[45_52][e];
	tf_n[d][0] = tr_n[c][0] + tr_f_n[3_24][d];
tr_n[c] ≥ tf_n[a] + tf_f_n[45_17][c];	tr_n[e][0] = tf_n[d][0] + tf_f_n[45_52][e];
tr_n[c] ≥ tf_n[f] + tf_f_n[45_9][c];	tf_n[f] ≥ tr_n[b] + tr_f_n[3_63][f];
tf_n[d] = tr_n[c] + tr_f_n[3_24][d];	tf_n[f] ≥ tr_n[e] + tr_f_n[3_59][f];
tr_n[e] = tf_n[d] + tf_f_n[45_52][e];	tf_n[c] = tr_n[a] + tr_f_n[3_17][c];
tf_n[f] ≥ tr_n[e] + tr_f_n[3_59][f];	tr_n[f][0] = tf_n[b] + tf_f_n[45_67][f];
tf_n[f] ≥ tr_n[b] + tr_f_n[3_63][f];	tf_n[c][0] ≥ tr_n[f][0] + tr_f_n[3_13][c];
	tf_n[c][0] ≥ tr_n[a] + tr_f_n[3_17][c];
	tr_n[d] = tf_n[c] + tf_f_n[45_24][d];
	tf_n[e] = tr_n[d] + tr_f_n[3_52][e];
	tr_n[d][0] = tf_n[c][0] + tf_f_n[45_24][d];
	tf_n[e][0] = tr_n[d][0] + tr_f_n[3_52][e];
	tr_n[f] ≥ tf_n[b] + tf_f_n[45_67][f];
	tr_n[f] ≥ tf_n[e] + tf_f_n[45_59][f];
t ≤ Kt;	t ≤ Kt;
t ≥ tr_n[e];	t ≥ tr_n[e];
t ≥ tf_n[e];	t ≥ tr_n[e][0];
t ≥ tr_n[f];	t ≥ tf_n[e];
t ≥ tf_n[f];	t ≥ tf_n[e][0];
	t ≥ tr_n[f];
	t ≥ tf_n[f];

Figure 5.11. **Equations for the critical path diagrams.** The delay equations are listed for the critical delay paths in the original RS-latch and buffer circuit and the equivalent non-cyclic replacement. An additional numerical subscript is added to the new variables to differentiate between each of the unique critical paths. Note that the resultant expanded equations are ordered for sequential evaluation.

Figure 5.11 describes the equations that correspond to the critical delay graph of the original circuit with feedback (*Part a* of *Figure 5.10*) and the replacement equations for the non-cyclic critical paths through the circuit (*Part b* of *Figure 5.10*). These are the actual equations produced by EPOXY. The extra numerical subscript is used to differentiate between each of the critical paths. Note that a few additional equations resulted from removing the cycles from the original circuit. However, the same delay times inherent to the gate, as represented by the tf_f_n[][] and tr_f_n[][] variables, are still referenced by all critical path equations.

5.6. Evaluating the Performance Equations

The key to delivering rapid circuit analysis is a mechanism for providing quick evaluation of the underlying symbolic performance equations. Once the performance equations are created within EPOXY, they are ordered for sequential evaluation. If the equations cannot be ordered, the cycle of referenced equations representing the feedback path is identified and removed as described in the previous section. The result is a set of symbolic equations that can be sequentially evaluated. The equations are ordered by rebuilding the linked list of all equations.

EPOXY can also evaluate the symbolic equations incrementally. If a few input variables change, EPOXY can evaluate only those equations that can be effected. Incremental evaluation entails additional computation for recursively traversing the data structure that indicates every equation that references the modified variable. Therefore, if a large number of inputs is changed, evaluation of all the equations is faster.

Even within the realm of incremental evaluation, EPOXY supports two different techniques. The first is best suited for updating the equations as a result of a change in a very few input variables. This technique recursively traverses the data structure that lists the equations that reference this variable (a reference list). The process terminates when no other equations need to be examined. If the equation's value is not modified, the search process is also terminated. Since the basic structure of node delay times involves a *max* operation, a change in an input variable does not often cause a change in the output node delay time. For rapid circuit evaluation, EPOXY provides incremental evaluation that is both structural and value dependent.

EPOXY's other incremental evaluation technique efficiently supports changes in a several variables. When variables are modified, the equations that reference this variable are marked. After all the variables values are modified, all equations are examined sequentially (as previously ordered). If the equation is marked, it is evaluated. If the result changes the variable on the left-hand side of the equation, the equations that reference this variable are also marked. Since all equations are sequentially ordered, the referenced equation must follow the recently evaluated equation. The present equation is then unmarked (since is has been updated), and the next equation is examined. If this equation is unmarked, it is ignored. Therefore, this incremental evaluation technique is also both structural and value efficient, since setting and testing the equations marks involves little overhead and only a single pass is needed to examine the effected equations.

Since EPOXY stores the symbolic equations in a format more suitable for evaluating the circuit performance, EPOXY provides another set of evaluation routines needed by algorithms that solve non-linear programs (NLP). The data values that the NLP formulation requires differ from those for performance evaluation. Therefore, EPOXY evaluates the equations differently to account for the fact that the symbolic equations and their gradients were not derived in NLP standard form. That is, evaluation of each constraint equation results in a slack value (g(x)) rather than setting the variable on the left hand side of the constraint.

Each of the equation evaluation techniques in EPOXY are listed in *Table 5.4*.

Evaluation techniques	Evaluation modes
Circuit analysis	All equations
Circuit analysis and gradient evaluation	Incremental, one input changes
NLP constraint and Jacobian matrix evaluation	Incremental, several input changes

Table 5.4. **Equation evaluation techniques and modes in EPOXY.** Three basic evaluation techniques are available for circuit analysis and NLP problems in EPOXY. For each of these techniques, three evaluation modes provide rapid evaluation.

During the course of evaluating the equations to determine circuit performance, the equation that sets the value of the constraint must be determined (which equation is the smallest upper-bound constraint or the largest lower-bound constraint). The constraint or constraints that determine the value of the variable are marked as limiting. Since the equations relating the input response times to output response times form a forest of potential delay paths, a critical delay path is simply the set of connected limiting constraints from an input to an output delay equation ($t\_...[][] \geq ...$). However, the critical paths are more easily identified by traversing the limiting equations from a limiting output to a limiting input, since reference lists are maintained for all variables.

Limiting constraint marks are generated as a simple byproduct of evaluating the circuit response time. Therefore, critical path information is easily provided. Circuit modification techniques, such as those presented in *Chapter 4*, use critical path information to determine the region of a circuit most suitable for structural changes.

One advantage of the C-style representation of the symbolic equations is that these equations can be compiled into an object module for even faster evaluation. The following section compares the evaluation of the internal data structures of EPOXY with the compiled equations.

5.7. Space/Time Tradeoff

Most of the computation in circuit optimization involves evaluating the circuit performance and gradient equations. Some programs spend unnecessary time in dynamically allocating and freeing data structures for each circuit analysis. Instead, EPOXY builds a static set of performance equations for the circuit by constructing internal data structures. The models for EPOXY can be built dynamically (once) and interpreted or compiled to form an executable binary. The decision to statically derive the equations instead of a dynamic derivation results in an increase in storage for a decrease in evaluation time.

The next table, *Table 5.5*, shows the execution time savings and extra storage requirements for determining the worst-case delay times (circuit analysis) for several circuits. The listed execution times are for simulating each static CMOS circuit 1,000 times, using Crystal and EPOXY's full equation evaluation mode.

A Comparison of Crystal and EPOXY											
Circuit	Crystal		EPOXY								
			interpret				compile				
	kby	sec	kby	build sec	run sec	total sec	kby	build sec	cc sec	run sec	total sec
inv.8	48	75.4	46	0.9	38.3	39.2	33	1.1	8.2	3.0	12.3
inv.10	56	97.5	54	1.1	47.4	48.5	33	1.4	9.8	3.7	14.9
rand.20	32	87.4	55	1.4	48.2	49.6	41	1.8	10.8	3.9	16.5
adder.1	32	360.6	88	2.7	86.9	89.6	41	3.4	21.4	7.7	32.5
adder.8	104	7328	441	7.6	508.6	516.2	115	11.6	151.7	45.9	209.2

Table 5.5. **Storage requirements and execution time savings.** All times (sec) are for a Sun-3/140 with a 68881 co-processor and 8Mbytes of memory runing Sun UNIX 4.2 Release 3.2. Crystal only lists the maximum heap storage for the circuit. Therefore, the number of kbytes reported is rounded to the nearest block size. The interpreted version gives the allocated storage (kbytes) for the circuit and the time to build (build) and run (run) the example. The compiled version produced by EPOXY gives the size of the executable binary (kbytes), time to produce the C file (build), time to compile (cc) and run (run) the program. The total running times (total) are listed for the compiled and interpreted versions.

Since *Table 5.5* shows that EPOXY *always* runs faster than Crystal (average of 56 times faster for compiled and 5 times faster for interpreted), the decision to statically derive the equations is indeed justified. It also shows that compiling the performance equations is always desirable even when the construction (build) and compile (cc) times are included. Compile time can be substantially reduced if object modules are available for frequently used circuits. However, the current version of EPOXY produces a C program that does not support changes in the circuit structure.

5.8. Implementing Circuit Modifications

EPOXY considers local modifications to the circuit structure to improve the overall performance or meet some difficult failing constraint. Modifications are made by directly manipulating the data structures associated with the circuit net-list and symbolic equations. Although these circuit changes are quick, unfortunately they are tied to the implementation technology, MOS.

Several support routines ease the difficulty in implementing the effects of circuit modifications on the underlying net-list and symbolic equation structures. For example, a buffer is inserted into a circuit by creating new nodes and transistors. Then the RC_model and RP_model routines are used to create the appropriate symbolic equations for these nodes and transistors. The new nodes are given subscripted versions of the original node name since these new nodes represent the same logical value as the original node. A similar technique is used in the creation of new transistors. Therefore, the naming convention helps locate inserted circuitry.

After the symbolic equations for the new circuitry are created, the partial derivative equations are also generated. Then these are inserted into the linked list of equations so they will be evaluated in the correct order.

Implementing the effects of splitting transistors is also straightforward. A new transistor is created with an additional subscript (unique integer). The additional subscript denotes that this transistor resulted from a circuit modification. Equations that reference the transistor's parameters (width and length) are replicated. The new variables names are then substituted for the original transistor's parameters; the corresponding references lists are also updated to reflect this substitution. Then the resulting new equations are simply placed after each of the original equations since these original equations were correctly ordered.

Reordering of transistors within logic gates requires the same types of operations as in the last two circuit modifications. Transistors are moved individually, because the changes are localized to the corresponding terms within a set of equations. In addition to the performance equations, the terms for the partial derivative equations are also moved.

5.9. Conclusions

This chapter illustrates many of the advantages of representing design performance by a set of static symbolic signomial equations. EPOXY automatically generates these equations from a net-list and layout. Then the selected optimization techniques solve the resulting non-linear optimization problem as specified by the user supplied parameter file. The design may be improved further by altering the circuit structure. The symbolic representation of circuit performance offers many advantages to each part of EPOXY. The most significant of these are:

1. The performance and derivative equations are evaluated rapidly.
2. The performance and derivative equations can be compiled for even faster evaluation.
3. Two forms of model independent incremental evaluation of the equations can be selected.
4. A user can specify the objective function, constraints and additional equations.
5. Electrical models are easily substituted.
6. Circuits with feedback are analyzed *independent* of the selected electrical model.
7. Critical paths, used by circuit modification techniques, are easily identified as a byproduct of symbolic performance evaluation.
8. The equations for the sparse Jacobian matrix are analytically derived.

Most of the computation for improving a design's performance involves evaluating the circuit performance and corresponding gradient equations. Crystal dynamically allocates the data structures for evaluating one RC path. Once this path is evaluated the data structures are again freed. Alternatively, EPOXY creates the symbolic equations statically that represent the circuit performance. When these

equations are interpreted, EPOXY is (on average) 5 times faster than Crystal. However, when these equations are compiled, EPOXY achieves an average speedup factor of 56 over Crystal. For the present implementation, a circuit with 1000 transistors adequately fits within 8 Mbytes of memory on a Sun-3/140 workstation.

Alternatively, these equations can be compiled into an optimized object module for each circuit. When common circuit blocks are used, the compiled equations can be rapidly evaluated since the object module can be executed directly.

Abstracting circuit performance by symbolic equations also reduces computation. Two incremental evaluation techniques can be selected for further speed improvements. These incremental techniques are both structural and value sensitive. Any update in the input variables triggers evaluation of only the affected equations thereby providing model independent incremental analysis. The other incremental technique is more efficient when updating equations in response to a change in several variables.

Most transistor sizing programs provide a fixed electrical model with a fixed optimization problem as defined by a predetermined objective function and constraints. This approach restricts a user from attaining the most desirable implementation. Since EPOXY formulates the non-linear program using a general signomial equation abstraction, a user can specify the objective function and additional constraints and equations.

Since all electrical models implemented within EPOXY produce symbolic equations, these electrical models are easily substituted. Effects of modeling accuracy on the resulting optimization are easily evaluated. Therefore, the tradeoff between model complexity (equation size) and numerical accuracy can be measured.

The performance of circuits with feedback are evaluated independent of the chosen electrical model. Feedback is removed by replacing the original (cyclicly referenced) equations with a larger set of equations that represent all feasible paths through the original circuit.

As a natural byproduct of evaluating the performance equations, limiting constraints are marked. A critical path is determined by traversing the marked equations from a limiting output to an input. Since reference lists are maintained for each variable, a depth first search of these equations marks is very fast.

Finally, the expense of computing finite difference approximations of the Jacobian matrix are avoided since the partial derivative equations are statically derived. Since many of the entries in the Jacobian matrix are fixed to 0 or 1, the entire Jacobian matrix is stored using a sparse representation. The optimization routines that reference the Jacobian matrix only evaluate those entries that have defining equations.

6. Conclusions and Contributions

EPOXY is a flexible open architecture for transistor sizing and circuit restructuring to meet performance constraints. This system represents a unique and powerful approach to solving the performance design problems for VLSI IC designers and evaluating the effectiveness of performance models, optimization techniques, and circuit modifications.

The design of the EPOXY system was motivated by the typical needs and concerns of VLSI IC designers in producing layout that meets electrical performance and area constraints. EPOXY integrates transistor sizing and circuit level changes in one complete system. An advantage of this approach is that circuit modifications can be made quickly without having to produce layout and re-extract the resulting design. Moreover, EPOXY can capitalize on these local changes by incrementally re-evaluating the circuit timing analysis and reducing the size of the optimization problem by selecting only the transistor widths that have changed.

EPOXY also provides a uniform platform for comparing performance models, optimization algorithms and circuit modifications. Representing the performance by polynomial equations is the key to delivering these features. Symbolic equations are an effective and powerful abstraction in providing a flexible environment for the rapid evaluation and improvement of circuit performance. Some of the key results drawn from this research are:

1. The entire desirable range of circuit performance can be represented by a performance envelope. A designer alters the constraint values and objective function parameters to balance among the design metrics to select a design within this envelope.
2. Accurate time models are crucial for designs with aggressive timing requirements since a small error in estimating circuit speed has a very large impact on the overall circuit area and power requirements.
3. While other transistor sizing programs use the total transistor area to represent the overall cell area, EPOXY represents the total circuit area more accurately by employing a virtual grid model. Total transistor area is actually a better measure of maximum dynamic power rather than total circuit area.
4. The combined strategy of applying the TILOS heuristic and then the Augmented Lagrangian optimization algorithm provides faster convergence than either of the two methods when run separately.
5. Circuit modifications, such as splitting large transistors, inserting buffers, and rearranging transistors within static logic gates, can substantially improve a circuit's performance (10 to 23% over transistor sizing alone).
6. The partial derivative equations that represent the sparse Jacobian matrix are derived in closed form. Therefore, the Jacobian matrix can be quickly evaluated without the need for finite difference approximations.

6.1. Future Research

The flexibility of implementing and combining performance models, optimization techniques and circuit modifications within one complete system opens the door to further evaluation and development. The only missing component in producing a transistor sizing program for other technologies, such as Gallium-Arsenide and bipolar, are the performance models. Once these performance models are complete, any of the optimization routines can be applied. In addition, other optimization techniques, such as recursive quadratic programming algorithms, should be evaluated to determine their effectiveness in solving the performance optimization problems. Similarly, new circuit modifications can be evaluated and applied within one complete system.

Since EPOXY represents the circuit performance by a set of equations, they can be written as a C program and compiled for a large savings in overall execution time. *Appendix 1* describes the format of the C program that EPOXY produces. The present program format should be improved so that the circuit modifications outlined in *Chapter 5* can be supported.

The virtual grid area model within EPOXY can be expanded to consider more complex transistor geometries. Snaking of large transistors can further reduce the cell area when a few large transistors are present.

A graphical user interface for EPOXY would more clearly demonstrate the performance improvement and layout impact during the optimization process. Something like a VLSI designer's cockpit would help a user visualize and control the optimization process. In addition, the overall system interaction could be simplified by using a menu driven selection process, as demonstrated by Shyu [Shyu88a].

7. References

[Bert82] D. Bertsekas, "Projected Newton Methods for Optimization Problems with Simple Constraints", *SIAM Journal on Control and Optimization,* Vol. 20 , 1982, pp. 221-246.

[Bray81] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design", *Proc. of the IEEE,* Vol. 69, No. 10, October 1981, pp. 1334-1362.

[Bray86] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, A. Yung and A. Sangiovanni-Vincentelli, "Multiple-Level Logic Optimization System", *IEEE ICCAD*, pp. 326-328, November 1986.

[Ciri87] M. A. Cirit, "Transistor Sizing in CMOS Circuits", *24th ACM/IEEE DAC Conf.*, pp. 121-124, June 1987.

[Ecke80] J. G. Ecker, "Geometric Programming: Methods, Computations and Applications", *SIAM Review,* Vol. 22, No. 3, July 1980, pp. 338-362.

[Fish85] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing", *IEEE ICCAD*, pp. 326-328, November 1985.

[Gill81] P. E. Gill, W. Murray and M. H. Wright, "Practical Optimization", *Academic Press*, London, New York, 1981.

[Glas84] L. A. Glasser and L. P. J. Hoyte, "Delay and Power Optimization in VLSI Circuits", *21st ACM/IEEE DAC Conf.*, June 1984.

[Hede87] N. Hedenstierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization", *IEEE Transactions on CAD,* Vol. CAD-6, No. 2, March 1987, pp. 270-281.

[Hedl84] K. Hedlund, "Models and Algorithms for Transistor Sizing in nMOS Circuits", *Int. Conf. on CAD*, 1984.

[Hedl85] K. S. Hedlund, "Electrical Optimization of PLAs", *22nd ACM/IEEE DAC Conf.*, pp. 681-687, June 1985.

[Hedl87] K. Hedlund, "Aesop: A Tool for Automated Transistor Sizing", *24th IEEE DAC Conf.*, pp. 114-120, June 1987.

[Hill87] D. Hill, "A Switch Level Synthesis System", *ICCD*, pp. 560-563, Oct. 1987.

[Hofm87] M. Hofmann and J. K. Kim, "Delay Optimization of Combinational Static CMOS Logic", *24th ACM/IEEE DAC Conf.*, pp. 125-132, June 1987.

[Joup83] N. Jouppi, "Timing Analysis for nMOS VLSI", *Proc. 20th IEEE DAC*, pp. 411-418, June 1983.

[Kao85] W. H. Kao, N. Fathi and C. Lee, "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits", *22nd ACM/IEEE DAC Conf.*, pp. 781-784, June 1985.

[Kare84] M. J. Karels, S. J. Leffler, W. N. Joy and M. K. McKusick, "UNIX Programmer's Manual, Reference Guide", *4.2 Berkeley Software Distribution, Virtual VAX-11 Version*, March, 1984.

[Koha78] Z. Kohavi, "Switching and Finite Automata Theory", *McGraw-Hill*, New York, NY., 2nd Edition, 1978.

[Lee84] C. M. Lee and H. Soukup, "An Algorithm for CMOS Timing and Area Optimization", *IEEE JSSC*, Vol. SC-19, No. 5, October 1984.

[Luen84] D. G. Luenberger, "Linear and Nonlinear Programming", *Addison-Wesley*, Reading, Mass., 2nd ed., 1984.

[Marp86a] D. P. Marple and A. E. Gamal, "Area-Delay Optimization of Programmable Logic Arrays", *Advanced Research in VLSI, Proc. of the 4th MIT Conference*, pp. 171-196, April 1986.

[Marp86b] D. P. Marple, *Performance Optimization of Digital VLSI Circuits*, Ph.D. Thesis, Stanford, September 1986.

[Marp87] D. P. Marple and A. E. Gamal, "Optimal Selection of Transistor Sizes in Digital VLSI Circuits", *Advanced Research in VLSI, Proc. of the 1987 Stanford Conference*, pp. 151-172, March 1987.

[Mats86] M. Matson and L. Glasser, "Macromodeling and Optimization of Digital MOS VLSI Circuits", *IEEE Trans. on CAD*, Vol. CAD-5, No. 4, October 1986, pp. 659-678.

[Mead80] C. Mead and L. Conway, "Introduction to VLSI Systems", *Addison-Wesley*, Reading, Mass., 1980.

[Mich87] G. D. Micheli, "Performance-Oriented Synthesis of Large-Scale Domino CMOS Circuits", *IEEE ICCAD*, pp. 751-765, September 1987.

[Neme84] M. Nemes, "Driving Large Capacitances in MOS LSI Systems", *IEEE Journal of Solid-State Circuits*, Vol. sc-19, No. 1, February 1984.

[Nye88] B. Nye, D. Riley, A. Sangiovanni-Vincentelli, J. Spoto and A. Tits, "DELIGHT.SPICE: An optimization-Based System for the Design of Integrated Circuits", *IEEE Trans. on CAD*, Vol. 7, No. 3, April 1988, pp. 501-519.

[Ober85] F. W. Obermeier and R. H. Katz, "PLA Driver Selection: An Analytic Approach", *Proc. 22th IEEE DAC*, pp. 798-802, June 1985.

[Ober88a] F. W. Obermeier and R. H. Katz, "An Electrical Optimizer that Consider Physical Layout", *Proc. 25th IEEE DAC*, pp. 453-459, June 1988.

[Ober88b] F. W. Obermeier and R. H. Katz, "Combining Circuit Level Changes with Electrical Optimization", *IEEE ICCAD*, pp. 218-221, November 1988.

[Oust85] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Transactions on CAD*, Vol. CAD-4, No. 3, July 1985.

[Pinc86] J. Pincus and A. M. Despain, "Transistor Sizing Using Simulated Annealing", *23nd ACM/IEEE DAC Conf.*, pp. 690-695, June 1986.

[Rubi83] J. Rubinstein, P. Penfield and M. A. Horowitz, "Signal Delay in RC Tree Networks", *IEEE Transactions on CAD*, Vol. CAD-2, No. 3, July 1983, pp. 202-211.

[Scot85] W. S. Scott, R. N. Mayo, G. Hamachi and J. K. Ousterhout, "1986 VLSI Tools: Still More Works by the Original Artists", UCB/Computer Science Dpt. 86/272, University of California at Berkeley, Computer Science Division (EECS), December 1985.

[Shyu88a] J. Shyu, "Performance Optimization of Integrated Circuits", UCB/Electronics Research Lab. M88/74, Ph.D Thesis, Unversity of California at Berkeley , November 22, 1988.

[Shyu88b] J. Shyu, A. Sangiovanni-Vincentelli, J. P. Fishburn and A. E. Dunlop, "Optimization-Based Transistor Sizing", *IEEE Journal of Solid-State Circuits*, pp. 400-409, April 1988.

[Trim83] S. Trimberger, "Automated Performance Optimization of Custom Integrated Circuits", *Proc. International Symposium on Circuits and Systems*, pp. 194-197, 1983.

[Wall88] D. E. Wallace, "Abstract Timing Verification for Synchronous Digital Systems", UCB/Computer Science Dpt. 88/425, Ph.D. Thesis, University of California at Berkeley, Computer Science Division (EECS), June 27, 1988.

[West85] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design, A Systems Perspective", *Addison-Wesley*, Reading, Mass., 1985.

[Wolf78] P. K. Wolff, A. E. Ruehli, B. J. Agule, J. D. Lesser and G. Goertzel, "Power/Timing: Optimization and Layout Techniques for LSI Chips", *Computer Science Press*, 1978.

8. Appendix: Generating C Simulation Programs

One advantage of representing circuit performance by a set of equations is that these equations can be compiled for rapid evaluation. Since compiling the symbolic equations as an object module offers a tremendous speed advantage, the structure of the C program, that EPOXY currently produces, will be examined in greater detail. EPOXY can create two basic programs: one for simulating the circuit by evaluating all the performance equations, and another for evaluating the constraints and Jacobian matrix for non-linear optimization programs. For each of the main program types, several versions can be generated, each with various features. The overall format of each program type are discussed below. Although the current program formats do not support changes in the circuit structure, they illustrate the design considerations in producing fast-running C programs.

In addition to selecting the basic program type, a user can choose the format of the intermediate variables EPOXY uses in these programs. However, the format of the input and output variables and constants is fixed so that other programs can reference the variables using a consistent set of names. *Table 8.1* lists all the various styles of intermediate variables that EPOXY can produce. The inputs, outputs and constraints are restricted to the *long readable* style except when the *associative array* style is selected. For example, an input variable, "w_f[3_n28]", is printed using the *associative array* style; while "w_f_3_n28" is the corresponding *long readable* style. Therefore, when the *associative array* style is chosen, the resulting code will not compile unless all the potential array arguments are declared as either a constant (e.g.: #define 3_n28 12) or an enumerated type (e.g.: typedef enum { ..., 3_n28, ...}). The *associative array* mode is meant as a very readable version of the original code.

Intermediate variable formats				
Flag	Style	Example	Advantages	Disadvantages
x	hex address	v531b8	terse description	overloads string table
l	long readable	R_f_n_i_3_n45_b_0	readable	worse string table
n	numbered array	v[22]	fast compile and reference	unreadable
a	associative array	R_f_n_i[3_n45][b][0]	readable	can't compile

Table 8.1. **Intermediate variable formats.** A user can select the format of the intermediate variables for the simulator or NLP program generated by EPOXY. An example of each variable format is given in the third column. The next two columns describe the advantage and disadvantage of each representation.

The first format, *hex address*, creates the intermediate variables using a terse unique name (its internal address in hexadecimal notation). The advantage of this naming strategy is that the size of the program is reduced since each variable name is fairly short (faster compile times). However, this format may cause the C compiler to run out of string table space for large input files (i.e., large circuits). This problem arises from an unfortunate limitation in the standard C compiler (cc): all variables names are stored in a *fixed* sized table. However, GNU CC † does not

suffer from this limitation.

The next format, *long readable*, produces variables names that are human readable, but require more string table storage than the hex address format. The *numbered array* format provides the fastest compilation (only a single variable array must be declared) and tersest representation. The last variable format, *associative array*, has been used throughout this dissertation since it is the most readable. The type of each array entry is contained in the base name of the array (e.g.; f for fet, n for node, i for unique integer). However, this format will not compile as valid C code unless the array arguments are specially declared (as described above).

```
/* C program format for the simulator */

/* All variables are declared globally. */
double <vars>;  /* See formats of <vars>.*/
double max;     /* Temporary variable */

initialize() {
    /* Set various constants and inputs. */
    l_f_3_28 = 2.0;
}

simulate () {
    double   _le, _ge, _temp;
    ...
    R_f_45_n20 = Krp * l_f_45_n20 * (1/w_f_45_n20);
    ...
    _temp = Kh;
    _le=_temp;
    _temp = l_f_3_n28 + l_f_3_n24 + l_f_3_n45 + l_f_3_n62 + 49 ;
    _ge=_temp;
    _temp = l_f_45_n28 + l_f_45_n20 + l_f_45_n45 + l_f_45_n62 + 49 ;
    if (_temp >= _ge) _ge=_temp;
    h = _ge;
    ...
}

output () {
    printf ("obj = %lf0, obj);
}
```

Table 8.2. **Format of the generated timing analyzer program.** EPOXY can automatically generate a C program that declares all the necessary variables and evaluates the performance equations. Note that the constraints (for h >= ..) are mapped into several statements that effectively produce the maximum of all lower-bound constraint equations. The upper-bound constraint is ignored in this calculation. These example equations were taken from the bnand.1 example.

† Copyright © 1988 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, (617) 876-3296.

EPOXY can produce C code for a circuit-specific performance analyzer by evaluating all of the performance equations. This program can be compiled and linked with a small program to call this simulation procedure. The result is a self-contained timing analyzer. The execution speed of this program format was quoted in the table on storage requirements and execution speeds. Alternatively, this program could be compiled and dynamically linked to EPOXY. Subsequent calls to the simulator within EPOXY would result in identical (rapid) execution speeds. *Table 8.2* demonstrates the overall format of the simulator program.

The timing analyzer provides three subroutines; one for initializing the constants and inputs, another for evaluating the performance equations, and one for printing the results. An external program can access the input variables by referencing the input variables that are declared globally.

EPOXY can also generate a program suitable for the needs of non-linear optimization programs. The format for this program structure is given in *Table 8.3*. Note that the user specifies the format of the variable names for both the internal variables and temporary gradient variables (D[][]).

C program format for non-linear optimization programs.

```
/* C program format for non-linear optimization programs.*/
#include <stdio.h>
#include "nlps.h"

int g_count, f_count, j_count;  /* Counts for statistics.*/

#ifdef DEFINE_NAMES
NLP_NAME  nlp_input_names[] = { "w_p", "w_n", ... }
NLP_NAME  nlp_constraint_names[] = { "g_v_i[w_f[45_n28]][0]", ... }
#endif

double  <vars>;  /* Intermediate simulation variables.  See formats of <vars>.*/
double  <gradient vars>;  /* Temporary gradient variables (see formats).*/

extern  double  Cp_n_c;  /* List of parasitic output node capacitance. */
int     number_output_nodes = 1;
double  *C_p_outputs[]  = { &Cp_n_c };

/* Sparse representation of Jacobian matrix */
/* Constraint for: w_f[45_n28] */
static  NLP_ARRAY_INPUT ji0[]   = { 12,     /* input w_f[45_n28] */
13};
/* Data vector for partial derivative of constraint */
static  NLP_DATA        jd0[]   = {-1.0};
static  NLP_SPARSE      jsparse[]       = { {ji0, jd0}, {ji1, jd1}, ... }

init_problem ()
{   nlp_inputs      = 13;
    nlp_constraints = 30;
    init_constants();
    /* allocate storage for input (x[]) and constraint (g[]) arrays.*/
}
```

C program format for non-linear optimization programs.

```
init_constants () { /* Set various constants and inputs. */ }

set_g ()
{   double _temp, _le, _ge, tempople;  /* Temporary variable.*/
    g_count++;                                  /* For statistics.*/

    g.v.a[0] = Kwminp - x.v.a[12];     /* w_f[45_n28] <= Kwminp */
    ...
}

double  f() /* The objective function */
{   f_count++;
    return (x.v.a[2] * Fh + x.v.a[3] * Fw + x.v.a[4] * Ft + x.v.a[5] * Fp + x.v.a[6] * Fa);
}

set_df ()   /* Negative of the partial derivative of the objective function */
{   d.v.a[2] = - (Fh);
    d.v.a[3] = - (Fw);
    d.v.a[4] = - (Ft);
    d.v.a[5] = - (Fp);
    d.v.a[6] = - (Fa);
}

set_j ()
{   double tempople;                /* Temporary variable.*/
    j_count++;

    ...
    /* constraint number: 23 */
    j.v.s[23].d[1] = v5a944 + v5b730;       /* J(23,9) input var: w_f[45_n20] */
    ...
}

#ifdef SIMULATOR
int     simulate_errors;

simulate()
{   double _temp, _le, _ge;         /* Temporary variable.*/
    simulate_errors=0;

    ...
    _temp = Kh;
    _le=_temp;
    _temp = l_f_3_n28 + l_f_3_n24 + l_f_3_n45 + l_f_3_n62 +  49 ;
    _ge=_temp;
    _temp = l_f_45_n28 + l_f_45_n20 + l_f_45_n45 + l_f_45_n62 +  49 ;
    if (_temp >= _ge) _ge=_temp;
    if (_le < _ge) _le_ge_err("x.v.a[2]", _le, _ge); else x.v.a[2] = _ge;
    ...
}
```

C program format for non-linear optimization programs.

```
_le_ge_err(v, _le, _ge)
char    *v;
double  _le, _ge;
{   if (not_silent) {  /* Complain if can't assign legal values */
        fprintf(stderr,"Can't assign legal value for %s:",v);
        fprintf(stderr,"between %lf and %lf.0,_le,_ge);
    }
    simulate_errors++;
}
#endif
```

Table 8.3. **Format of the C program for NLP routines.** This figure describes the general format of the program generated by EPOXY for non-linear optimization programs. The Jacobian matrix is declared using a sparse representation that is rapidly compiled and executed. The supporting routines for setting the constraint matrix, Jacobian matrix and simulator all reference this representation.

The program formulation incorporates several advantageous implementation techniques. The static storage and access of the Jacobian matrix uses an efficient sparse representation. The rows of Jacobian matrix are defined and initialized by the automatically defined arrays: *ji#* and *jd#*. The first array (*ji*{constraint#}[index] = {input number}) gives the numeric index of the Jacobian inputs defined in corresponding data array (*jd*{constraint#}[index] = data). For example, *ji0*[1] gives the numeric index (12) of the Jacobian input, *w_f[45_n28]*, for the constraint equation, *w_f[45_n28] >= Kwminp*. The data storage for this sparse matrix entry (*J*[0][1]) is provided by the second array: *jd0*[1]. The index data is only used by programs that may need to traverse the structure of this matrix. All defined equations reference the exact address of the data entry (&(*jd0*[1])) directly. Therefore, accesses to the data storage only requires one address mode calculation: one base (&*jd*) plus the offset ([1]). Most CPU's provide a single instruction for this purpose. Since there are new array entrys and they are relatively short, the storage for the string table is not easily exhausted either.

Procedures are available for evaluating the objective function (f()) and constraints (set_g()) as well as their partial derivatives (set_df() and set_j()). The constraint equations are evaluated in the appropriate NLP standard form (g(x)<= 0.0). In addition, the user can direct EPOXY to generate additional code to evaluate all the constraints as performance equations (simulate()). Also, a subroutine (_le_ge_err()) is called during performance evaluation to verify that the data does indeed fall within the upper and lower-bound constraints.

The optimization routines were written to handle equations as internal data structures within EPOXY or as a separate external program. The data structures specified by the file "nlp.h" describe the Jacobian input, constraint and objective function gradient vectors as either, a full array (external program format) or linked list (internal format). Similarly, the Jacobian matrix are represented using the sparse representation (external program format) or as a linked list (internal format).