DESIGN-ORIENTED MIXED-LEVEL CIRCUIT AND DEVICE SIMULATION

by

David Alan Gates

Memorandum No. UCB/ERL M93/51

23 June 1993

DESIGN-ORIENTED MIXED-LEVEL CIRCUIT AND DEVICE SIMULATION

by

David Alan Gates

Memorandum No. UCB/ERL M93/51

23 June 1993

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Design-Oriented Mixed-Level Circuit and Device Simulation

by

David Alan Gates

Doctor of Philosophy in Electrical Engineering

University of California at Berkeley

Professor Ping K. Ko, Chair

Integrated circuits (ICs) are the building blocks of modern computing and communication systems. The design of high complexity ICs has been enabled by the development of a large number of computer-aided design (CAD) tools for IC design (ICCAD). Mixed-level circuit and device simulation has begun to find its place as a CAD tool for the development of new IC technologies. In this dissertation, problems associated with providing support for mixed-level circuit and device simulation in an expanded role as an ICCAD tool are investigated.

Four issues of concern for a mixed-level simulator have focused this research: reliability, utility, portability, and performance. A new mixed-level circuit and device simulator called CIDER has been developed to address these concerns. The first three concerns are addressed in a serially executing version of CIDER. To obtain reliable simulation results, new models for physical effects that are important in present-day IC technologies are included in CIDER. An enhanced user-interface has been developed to increase the utility of CIDER. Finally, CIDER has been ported to a variety of engineering workstations.

The final concern, performance, is addressed in a version of CIDER that runs on distributed-memory multicomputers. The need for parallel computing is established by measuring the serial performance of CIDER. Single workstations are roughly 10 to 100 times too slow to support design of reasonably sized circuits.

Algorithms for exploiting parallelism in mixed-level simulation are reviewed, and an architecture is proposed for a parallel circuit and device simulator. A limited form of the proposed approach has been implemented on two multicomputers: a hypercube supercomputer and a cluster of engineering workstations. On a set of benchmark

circuits, a best speedup of 12 on 16 processors of the hypercube is achieved. Unfortunately, the implemented approach has a number of limitations that are identified here for the first time.

Several applications of CIDER are presented that demonstrate the new parallel capability. In each application, the circuits contain multiple numerically modeled devices. The hypercube version of CIDER is used to simulate these circuits in a reasonable amount of time. New insight into these circuits is obtained by examining simulation results.

Ping K. Ko
Thesis Committee Chairman

Contents

List of Figures

List of Tables

Acknowledgments

Four and a half years have passed since I first came to Berkeley in 1988. In that time, I've crossed paths with a large number of people here in the Bay Area. I owe a debt of gratitude to each and every one of them for making my time here enjoyable, interesting, and fruitful. My thanks to you all.

I've had the opportunity to work with not one but two advisors while here at Berkeley. Prof. D. O. Pederson guided me through my first 4 years, and Prof. Ping Ko has capably taken over these last 6 months. From DOP I've learned to focus on the D in C.A.D.: design. I've also learned valuable writing and presentation skills that will serve me well in the rest of my career. Through numerous (sometimes lengthy) discussions with Ping, I've come to value his ability to step back from the small problem at hand and relate it the big picture and to other possibilities. I've also found our discussions about the future of the computing industry very enjoyable. I would also like to thank Prof. Phil Colella from the ME department, the third member of my thesis committee, for reading my thesis and approving it at a time I know was very hectic for him. I appreciate the patience all three have displayed in the face of the tortuously slow pace I've taken while writing this dissertation and the unreasonable time schedules that have come about as a result.

In addition to my advisors, I've also had the good fortune to take courses from some very talented teachers while in graduate school: Professors Brayton, Gray, Hodges, Meyer, and Sangiovanni-Vincentelli all come immediately to mind. A special thanks to Prof. Howe for putting up with what must have been one of his more difficult EE105 teaching assistants.

Many students and others have come and gone from Cory Hall while I've been here. I'll begin with my cubiclemates in 550A. I thank Beorn Johnson for all of his

help with SPICE3, for listening to my gripes about same, and for generally being an interesting guy to talk to when I felt like taking a break. I thank Karti Mayaram and Theo Kellesoglou for helping me get started and keeping in touch after having gone on to other places[1]. I enjoyed collaborating with Jean Hsu, Hoa Luong, Emy Tan, and Morteza Zarrabian during those first two years of classes. Darrin Young has supplied irrepressible enthusiasm for all things EE, and Mark Vitunic provided his wry cynicism and wicked little circuit-analysis problems. Sherman Chen, Paolo Giusto, and our honorary late-night cubiclemate, Harry Hsieh, have provided company during the long nights I've spent writing my thesis.

Moving out from 550A into 550, I thank all the members of the CADgroup collectively for putting up with my foray into distributed computing these last few months. If unknowingly I've greatly inconvenienced any of you, I'm truly sorry. Thanks to Edoardo Charbon, Eric Felt, Enrico Malavasi, and Ed Liu for helping me through the process of my first conference presentation. Thanks also to Ed for his help while TAing EE105 and for our late-night discussions about thesis writing and our futures. I'd like to thank Ken Nishimura and Cormac Conroy for letting me try to explain why my simulator really was useful to them, and for letting me know what analog IC design is all about. To my knowledge, Clement Szeto is the only person around here who has been brave enough to work with CODECS while I've been here, and for that I am grateful. Moving out into the rest of Cory, Jian Hui Huang, another of Ping's students, is always ready with a friendly smile whenever we pass in the hallways. Thanks to Brad Krebs and Mike Kiernan for their excellent support of the CADgroup computing environment, to Flora Oviedo, Irena Stanczyk-Ng, Elise Mills and Gwyn Horn for their friendly administrative support, and to Genevieve Thiebaut and Heather Brown for their help dealing with all that departmental paperwork I hate so much.

Moving out of Cory Hall entirely, I go across the bay to Stanford. Prof. Bob Dutton has generously supported my work through access to Stanford's hypercube, and has introduced me to several of his students as well. My interactions with Greg Anderson, Goodwin Chin, and Bruce Herndon have expanded my view of the world and stimulated my own research. Thanks to Zhiping Yu for discussing mixed-level circuit and device simulation with me on several occasions. Going on to Intel, I thank

[1]Special thanks to Karti for not giving me the hook during my CICC talk.

Don Scharfetter for setting up and mentoring a very interesting summer internship in 1991, and Tim Thurgate for many challenging discussions while I was there.

In addition to donated hypercube time from Stanford and Intel, this research has been funded by a grant from the Semiconductor Research Corporation and their support is greatly appreciated. The material in this dissertation is also based on work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations in this dissertation are those of the author and do not necessarily reflect the views of the National Science Foundation.

Next, I return to the East Bay and then move on to points far beyond, to my friends and family, to the people that have made life enjoyable when I can get away from school. Thanks to Ann Blake and Greg Bobrowicz for fun afternoons and evenings talking and playing board games. Thanks to Pauline Bennett for discussing EE and grad school with me by long-distance on occasion. Thanks to Diane Foray, Jeff Vollin and Steve Bloor for helping me get out and see the wilderness once in awhile[2].

To my parents, Jacque and Bob, and my siblings, Ken, Laurie and Sheryl, I give thanks for the refuge and respite from grad-school pressure whenever I find time to come home for a visit, even if it's only by phone. I thank my father-in-law, Norm Shapiro, for his continued interest in my work and in my well-being.

Finally, I would like to thank my wife and partner, Cathy, for putting up with all the ups and downs, the late and lonely nights, the anger and frustration. Without her tremendous support during the last weeks and months, I would probably be writing this six months from now instead of today. Her ability to help me 'get it done' means a great deal to me. I am lucky to have such a talented woman at my side.

[2]By current definition that's just about anything outside the front door to Cory Hall.

Chapter 1

Introduction

1.1 Motivation

Integrated circuits (ICs) are the building blocks of modern computing and communication systems. The design of high complexity ICs has been enabled by the development of a large number of computer-aided design (CAD) tools for IC design. Among these tools are programs used for *verification* of IC designs. The most important verification tools are simulators that predict an IC's performance before it is actually fabricated. Because simulation is much cheaper and faster than fabrication, IC simulators reduce the costs associated with developing new designs.

For many years, electrical circuit simulators have served as the main tool for verifying the individual circuits that make up an IC design. However, continuing advances in IC technology require the models embedded in circuit simulators to be updated on a regular basis. As the component dimensions in ICs continue to shrink with each new technology generation, it has become increasingly difficult to develop accurate models of the IC components' behavior. For IC devices that are used infrequently, the cost of model development may be prohibitive. This discourages innovation in design by preventing an IC designer from using the full range of devices available in an IC process.

In parallel with the development of CAD tools for IC design (ICCAD), separate CAD tools have evolved for creating new IC technologies (TCAD). Figure 1.1 shows the flow of simulation tools in a typical TCAD environment. ICCAD and TCAD overlap at the circuit simulation step. Process simulators mimic the behavior of the various

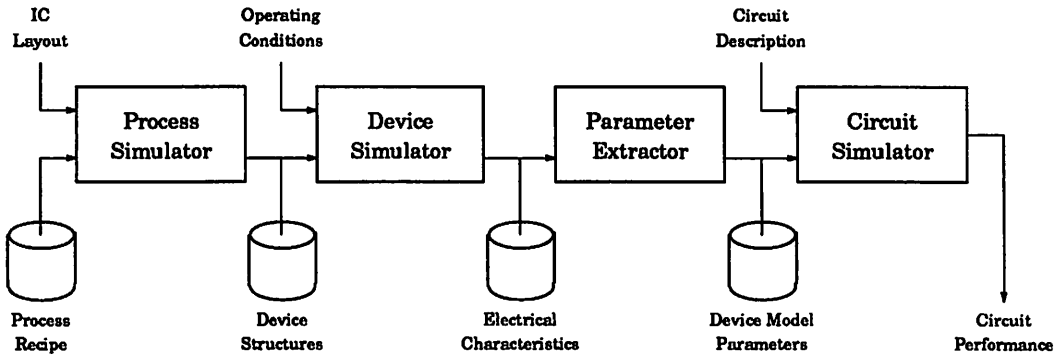


Figure 1.1: Standard TCAD Simulation Flow

chemical, thermal, and mechanical processes that are used to actually fabricate an IC. They provide information about the structures of the various devices available. The most important structures are those corresponding to transistors, such as metal-oxide-semiconductor field-effect transistors (MOSFETs) or bipolar junction transistors (BJTs). Device simulators take input in the form of device structures and predict the electrical behavior of those structures. The electrical characteristics are then passed to a parameter extractor that yields an appropriate set of parameters for the device models installed in the circuit simulator. When combined with a description of a particular circuit, the device models can then be used by the circuit simulator to predict the target circuit's performance.

The accuracy of the overall TCAD simulation system is affected by each of the component steps. As noted already, developing good compact analytical device models[1] to install in a circuit simulator can be difficult and expensive. In addition, the parameter extraction step can introduce artificial effects that obscure the true physical behavior of a circuit. This makes it difficult to modify the IC process to optimize circuit performance. As an alternative to the traditional TCAD simulation system, a mixed-level circuit and device simulator can be used to bypass parameter extraction and compact-model development, as shown in Figure 1.2. The mixed-level simulator provides a direct link between the underlying IC technology and the circuit performance. In addition, mixed-level simulation can provide device models for unusual devices and for device behaviors that are difficult to include in compact

[1]Compactness is a reference to the fact that typically a small set of analytical expressions is used to predict device operation over a wide range of bias conditions.

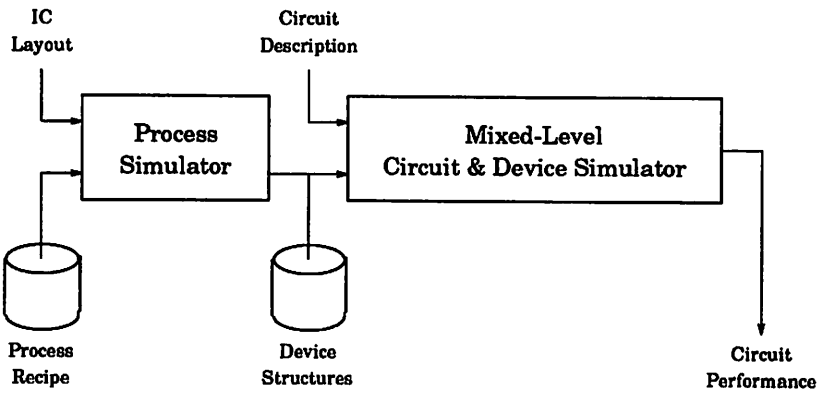


Figure 1.2: Alternate TCAD Simulation Flow

models.

Unfortunately, there is a price to be paid for mixed-level simulation. Typically, device models based on numerical device simulation (numerical models) are two to four orders of magnitude more time consuming to evaluate that those using compact sets of analytical expressions (compact models). Therefore, mixed-level simulations are restricted to applications where the extra time can be tolerated, or where accuracy is of paramount concern. Typical applications are the evaluation of key indicator circuits during technology development, investigation of the effects of difficult-to-model behaviors on circuit performance, and in the design of small reusable subcircuits that are used to build up very-large-scale-integrated (VLSI) circuits.

In the last two applications, mixed-level circuit and device simulation passes out of the realm of TCAD and crosses over into the ICCAD domain. The focus of this research has been to develop mixed-level simulation to further support its role as an ICCAD tool. By investigating and solving the problems associated with this design-oriented approach to mixed-level circuit and device simulation, it is hoped that IC designers will be able to use such a tool to develop innovative, better performing, and more cost-effective circuit designs.

1.2 Research Goals

Development of a mixed-level simulator for circuit design is an exercise in software engineering. A mixed-level circuit and device simulator is a complex piece of

computer software, and like any other piece of software, it must address certain key issues. In this work, four major issues have influenced the choice of topics and the decisions made when resolving difficulties. Although these issues are rarely brought out explicitly in the remainder of this dissertation, their influence should be acknowledged here at the outset. The four major software issues are:

Performance A design-oriented mixed-level simulator must address the computational burden imposed when compact device models are replaced by numerical device models.

Reliability The simulator should provide answers to as many circuit/device simulation problems as is practical. In addition, good models for the physical effects that are important in present-day IC devices should be installed in the device simulation portion of the program.

Utility The various analyses typically provided by IC circuit simulators should also be supported by a mixed-level simulator. It should also be easy to describe circuits and devices to the circuit simulator, and to visualize and interpret the results of completed simulations.

Portability Today's computing environments are populated by a wide variety of computers and operating systems. Any program that wishes to be of use to a wide audience must address the issue of portability between different computing environments.

One product of this research is a new mixed-level circuit and device simulator, CIDER that has been developed with these four issues in mind. Two versions of CIDER exist. The first is a serial, uniprocessor version that addresses the issues of reliability, utility, and portability. The serial version is an enhanced version of a previous mixed-level simulator called CODECS [MAYA88]. The second version focuses on the final issue, performance. A parallel, multiprocessor version of CIDER has been developed that has been ported to two different distributed-memory multicomputing systems: a dedicated high-performance hypercube supercomputer and a network of engineering workstations operating in concert. Multiprocessor computers exploit the parallelism available in mixed-level circuit and device simulation to enhance performance beyond that achievable by conventional uniprocessor computers. Performance is the primary

factor limiting wider application of mixed-level simulation to IC design. The major portion of this dissertation is therefore devoted to characterizing performance, identifying ways to improve performance, and describing a multicomputer implementation that improves upon existing performance.

1.3 Organization of the Dissertation

The remainder of the dissertation is organized as follows. In Chapter 2, the simulation algorithms of CIDER inherited from its predecessor CODECS are described. This provides background for understanding the mixed-level circuit and device simulation problem as well as an opportunity to point out several enhancements that have been added into CIDER. In Chapter 3, the performance of CIDER is analyzed when it is run in several uniprocessor computing environments. Empirical models of CIDER's computing-resource consumption are developed and then used to predict the resource requirements of typical circuit-design applications. Chapter 4 explores the different opportunities for exploiting parallelism in mixed-level-simulation-based IC design. An architecture is proposed for a parallel mixed-level simulator that exploits multiple forms of parallelism. In Chapter 5, a limited implementation of the proposed architecture is described. This parallel version of CIDER has been ported to an Intel iPSC/860 hypercube and a network of DEC workstations. The results of a performance evaluation of these two parallel versions are presented. An analysis of these results identifies several limitations of the implemented approach and offers methods to work around these limitations. Several applications of CIDER are presented in Chapter 6. The new parallel capability allows previously unfeasible problems to be simulated in a reasonable amount of time. The applications are the characterization of a hypothetical IC process, a study of gain modeling in several analog MOS amplifiers, and an evaluation of a bipolar IC output stage. Finally, in Chapter 7, the main conclusions of the dissertation are summarized and directions for future research are suggested.

Several appendices supplement the main body of the dissertation. Appendix A is a user's manual for CIDER that describes its features and provides several examples of its use. Appendices B, C and D contain various CIDER input descriptions for some of the circuits mentioned in the body of the work. Appendix E supplies information on how to obtain the source code to CIDER.

Chapter 2

Mixed-Level Circuit and Device Simulation

2.1 Overview

In this chapter, the algorithms implemented in the mixed-level circuit and device simulator, CODECS [MAYA88], are reviewed. Throughout, additions to the original version which improve on its capabilities are identified. The upgraded version, which is used as a basis for developing a parallel implementation, is called CIDER.

In Section 2.2, the algorithms that have become standard for circuit simulation are described. Next in Section 2.3, a review of the device simulation problem is provided, Finally, in Section 2.4, methods are presented for solving the mixed-simulation problem. Throughout the chapter, differences and similarities between CODECS and CIDER are highlighted.

2.2 Circuit Simulation

As shown in Figure 2.1, circuit simulation is a process that transforms a description of a circuit and its operating environment into a summary of the circuit's behavior in that environment.

This process is supported by a general-purpose circuit simulator such as SPICE2 [NAGE75] or SPICE3 [QUAR89]. The three most common forms of analysis implemented in circuit simulators are DC analysis, transient analysis, and small-

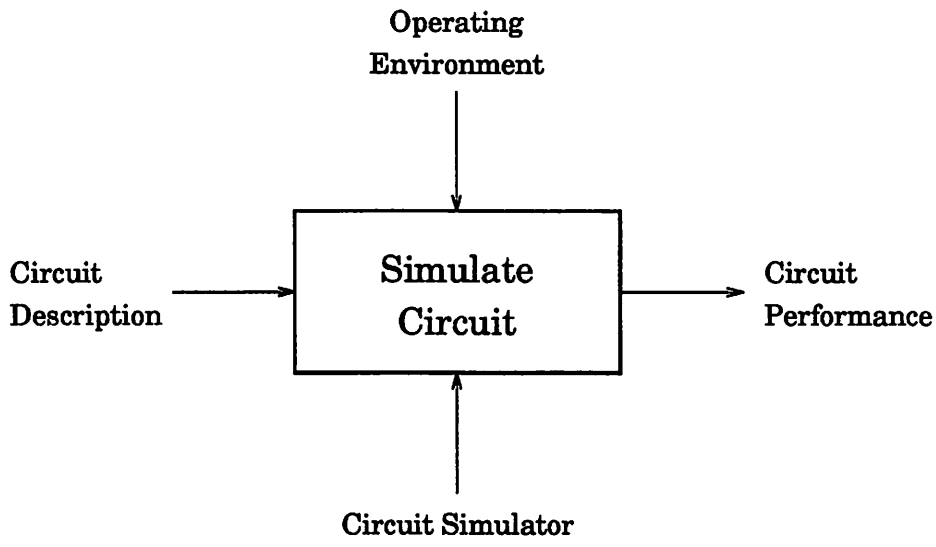


Figure 2.1: Circuit simulation – activity summary

signal AC analysis. Of the three, DC analysis is perhaps the most important, since a DC operating point analysis is a precursor to both transient and AC analyses.

2.2.1 Circuit Description and Equation Formulation

A circuit is most often described in terms of its elements and their interconnections. This description can be developed either in textual form using a circuit specification language or in graphical form using a *schematic-capture* program. In either case, Kirchoff's current and voltage laws (KCL and KVL) are used to translate the connectivity information into equations relating the electrical variables of the circuit [SING86]. In addition, the physical responses of the circuit elements to various forms of electrical stimuli are embodied in the branch constitutive relations (BCRs). Each relation is a mathematical model of the actual physical behavior of an element and is ideally characterized by identifiable physical parameters, such as the emitter area of a bipolar transistor, the gate-oxide thickness of a MOSFET, or a device's operating temperature. However, in many cases an empirical approach must be taken where a stimulus-response curve is instead described by an abstract mathematical function. This function may incorporate parameters, but in general they have no physical significance. If some of the parameter values of a BCR are uncertain or vary statistically, it is possible to describe them using distribution functions.

Sparse Tableau Analysis (STA) [HACH71] uses KCL, KVL and the BCRs to formulate the circuit simulation problem as one large system of equations. For a circuit with $n+1$ nodes and b branches, the resulting system has $n+2b$ equations: n from KVL, b from KCL, and b from the BCRs. This technique allows an arbitrary composition of both voltage- and current-controlled circuit elements. A second approach which still allows arbitrary circuit composition while reducing the overall system size is Modified Nodal Analysis [HO75]. In this approach, Nodal Analysis (NA), which is appropriate for circuits containing only voltage-controlled elements, is supplemented with the equations needed to describe current-controlled elements. The system size of MNA is somewhat larger than the n equations of NA due to these extra equations.

2.2.2 The Circuit Operating Environment

The operating environment of a circuit has two parts. The first describes the type of analysis to performed by the simulator and the analysis parameters. The second part is a binding of any unspecified parameter values to concrete ones. For the most part, this means that the type of analysis being performed is used to determine the values of the independent current and voltage sources in the circuit. However, some circuit simulators [MET90] also allow a device parameter value to be swept in order to determine the value that optimizes circuit performance. The value of the swept parameter is therefore part of the operating environment. Unfortunately, this capability is not available in SPICE3, so it also unavailable in CODECS and CIDER.

2.2.3 DC Analysis

The state of a circuit when none of its electrical variables vary with time is known as its DC or steady state. DC analysis of a circuit has three primary uses. First, it can verify that the circuit has been biased properly to establish a stable operating point. Second, it can be used to generate DC transfer curves by recording the response of the circuit as one or more input sources are stepped through a series of values. Finally, it is used to initialize the state of the circuit prior to a transient or AC analysis. Since the BCRs in steady state contain no time derivatives, the DC simulation problem can be expressed as a system of nonlinear algebraic equations:

$$F(r, s) = O \tag{2.1}$$

where r is the vector of unknown responses (voltages and currents), s is the vector of stimuli and F is a nonlinear vector-valued function obtained from an MNA formulation of the circuit equations. Since Equation 2.1 is a nonlinear system, an iterative procedure, such as the Newton-Raphson algorithm, must be used to find the unknown vector r. At each iteration k, a linear system of equations is formed that relates a solution update, Δr^k, to the current solution, r^k:

$$J(r^k)\Delta r^k = -F(r^k) \tag{2.2}$$

$J(r^k)$ is the Jacobian matrix ($\frac{\partial F}{\partial r}(r^k)$) of the circuit equations. The solution at the next iteration, r^{k+1}, is then computed by solving Equation 2.2 for Δr^k, and adding it to the current solution: $r^{k+1} = r^k + \Delta r^k$. The initial guess r^0 can be constructed in a number of ways, most often by using a previous solution of the circuit equations. In SPICE2 and SPICE3, Equation 2.2 is assembled in a slightly different way so that the next solution can be computed directly:

$$J(r^k)r^{k+1} = J(r^k)r^k - F(r^k) \tag{2.3}$$

In both formulations, a linear system of equations $Ax = b$ must be solved. This system has the following properties:

- Sparseness: The number of non-zero entries in any one row of A is typically small due to low degrees of connectivity between the circuit elements. The sparsity pattern often has no discernible or exploitable structure.

- Nonsymmetry: In general, the matrix is both structurally and numerically nonsymmetric.

- Real-Valuedness: The entries in the matrix A and the right-hand-side (RHS) b are real numbers.

In this situation, a general-purpose sparse matrix analysis package employing L/U decomposition [DUFF86] is commonly used to solve the system of equations. For example, the matrix package SPARSE [KUND86] is employed in the current version of SPICE3.

2.2.4 Transient Analysis

If the input sources of a circuit have time-varying values, the circuit response typically consists of both transient components and steady-state components. Which component is of greatest interest depends on the application being considered. For example, in digital design, the transient switching behavior of a gate may be desired in order to determine rise and fall times or propagation delays. In nonlinear analog design, the sinusoidal steady-state response may be needed to calculate the harmonic distortion of a gain stage. Direct-method simulators such as SPICE2 generally include the capability of computing the time-domain response of a circuit starting from an initial state. While this capability is most useful in transient analysis, it can also be used to find the steady-state, provided that the user is willing to simulate over a period long enough to allow any initial transients to die away.

The dynamic behavior of a circuit is described by a system of nonlinear ordinary differential-algebraic equations:

$$\boldsymbol{F}(\dot{\boldsymbol{q}}(t), \boldsymbol{r}(t), \boldsymbol{s}(t)) = \boldsymbol{O} \tag{2.4}$$

$$\boldsymbol{q}(t) = \boldsymbol{Q}(\boldsymbol{r}(t))$$

The response and stimulus vectors, $\boldsymbol{r}$ and $\boldsymbol{s}$, are supplemented by a vector $\boldsymbol{q}$ that contains the state variables of the energy-storage elements in the circuit. However, the system of equations does not increase in size because the state variables are related to the circuit response by the nonlinear vector-valued function, $\boldsymbol{Q}$.

The initial state of the circuit is obtained by assuming that the circuit is in a DC steady state ($\dot{\boldsymbol{q}}(t) = \boldsymbol{O}$). Substitution of this constraint into Equation 2.4 yields a set of DC operating point equations:

$$\boldsymbol{F}(\boldsymbol{O}, \boldsymbol{r}(0), \boldsymbol{s}(0)) = \boldsymbol{O} \tag{2.5}$$

The solution $\boldsymbol{r}_0 = \boldsymbol{r}(0)$ obtained through DC analysis can then be used to obtain the initial states of the energy-storage elements:

$$\boldsymbol{q}(0) = \boldsymbol{Q}(\boldsymbol{r}_0) \tag{2.6}$$

Once the initial conditions are determined, the solution over the desired time interval $[0, T]$ can be computed. Since it is not generally possible to express the

responses as analytical functions of time, numerical analysis is used to compute the solution. The time interval $[0,T]$ is discretized into a set of time points, $\{t_1, t_2, \ldots, t_N\}$, and the true solution $r(t_n)$ is approximated at each point by r_n. The time derivatives, $\dot{q}(t_n)$, are typically replaced by an implicit integration formula [LINI86] that relates them to the current, unknown values of the state variables q_n and previous, known values, q_{n-i}. The previous states q_{n-i} can either be saved or recomputed based on the previous solutions: $q_{n-i} = Q(r_{n-i})$. The system of equations obtained after time discretization is a nonlinear algebraic system in the unknowns r:

$$F(\dot{q}_n, r_n, s_n) = O \tag{2.7}$$

$$\dot{q}_n = H_k(Q(r_n), ..., Q(r_{n-1-k}))$$

where H_k represents the algebraic integration formula operating on the current state variables and k previous sets of state variables. This nonlinear system is solved using methods similar to those employed in DC analysis.

One complication of transient analysis is that a set of time points adequate to capture changes in all of the circuit responses is not known *a priori*. The usual solution to this problem is to make a reasonable estimate for the value of the next timepoint based on the previous behavior of the circuit. If the (estimated) error introduced into the solution by the time point is unacceptable, a new, smaller time step is chosen, and the solution is recomputed. This procedure is repeated until an acceptable timepoint is found, or a minimum time step is reached and the simulation is aborted. In addition to error constraints, other factors such as natural and induced breakpoints in the circuit waveforms and overall stability of the time discretization formula must be taken into account. Complete descriptions of the timepoint selection methods used in SPICE can be found in [NAGE75] and [QUAR89].

2.2.5 Small-Signal AC Analysis

Computation of the response of a circuit to time-periodic sinusoidal inputs is known as AC analysis. AC analysis is useful in evaluating the frequency response of analog signal processing circuits. If the signal levels of the inputs and outputs are sufficiently small, the response can be assumed to be linear about the DC operating

point. In this case, the general transient behavior, Equation 2.4, can be approximated by a first-order Taylor series expansion about the operating point:

$$F(\dot{q}, r, s) = F(O, r_0, s_0) + \frac{\partial F}{\partial \dot{q}}\dot{q} + \frac{\partial F}{\partial r}(r - r_0) + \frac{\partial F}{\partial s}(s - s_0) \tag{2.8}$$

Because both the final solution and the DC operating point satisfy Equation 2.4, the left-hand-side and the first term on the right-hand-side of Equation 2.8 are zero. The small-signal stimuli $s - s_0$ and the small-signal responses $r - r_0$ can be represented as phasors [STRU85]: $\tilde{s}e^{j\omega t}$ and $\tilde{r}e^{j\omega t}$, where $\tilde{s}$ and $\tilde{r}$ are complex quantities and ω is the input frequency. After substituting these values, the following linear system of equations is obtained:

$$\frac{\partial F}{\partial \dot{q}}\frac{\partial Q}{\partial r}(r_0) \cdot j\omega\tilde{r} + \frac{\partial F}{\partial r}\tilde{r} + \frac{\partial F}{\partial s}\tilde{s} = O \tag{2.9}$$

Notice that state-variable function Q has been used to eliminate the small-signal response of $\dot{q}$. Equation 2.9 can be rearranged to obtain a matrix equation for $\tilde{r}$:

$$\left[\frac{\partial F}{\partial \dot{q}}\frac{\partial Q}{\partial r}(r_0) \cdot j\omega + \frac{\partial F}{\partial r}\right]\tilde{r} = -\frac{\partial F}{\partial s}\tilde{s} \tag{2.10}$$

The matrix in Equation 2.10 has a zero/non-zero structure identical to that obtained from DC analysis. However, the entries in the matrix and RHS are now complex quantities, and a sparse matrix package that can perform L/U decomposition using complex arithmetic is necessary.

2.2.6 Visualization and Representation of Circuit Behavior

The results of a circuit simulation must be presented to the user in an understandable way. Post-processing programs convert the raw data obtained from a simulation into a form that allows a designer to evaluate easily the circuit performance. The most common form of display used in circuit design is the Cartesian plot, where one or more dependent variables is graphed against a single independent variable such as time or frequency. The need to plot higher dimensional data is rarely required.

Early simulators such as SPICE2 often include their own post-processors that are tailored to the needs of circuit simulation. Raw data is held in main memory until it is needed by the post-processor. By way of contrast, SPICE3 is more loosely coupled to its general-purpose post-processor NUTMEG [JOHN92]. Disk files stored in a common

rawfile format are used for persistent storage of simulation results. This arrangement effectively decouples the simulator core from the post-processor. As a result, NUTMEG can serve as a common post-processor for multiple simulation programs.

2.3 Device Simulation

As shown in Figure 2.2, device simulation is a process that transforms a description of a device's structure and its external boundary conditions into a summary of the device's characteristics under those conditions. This process is supported by

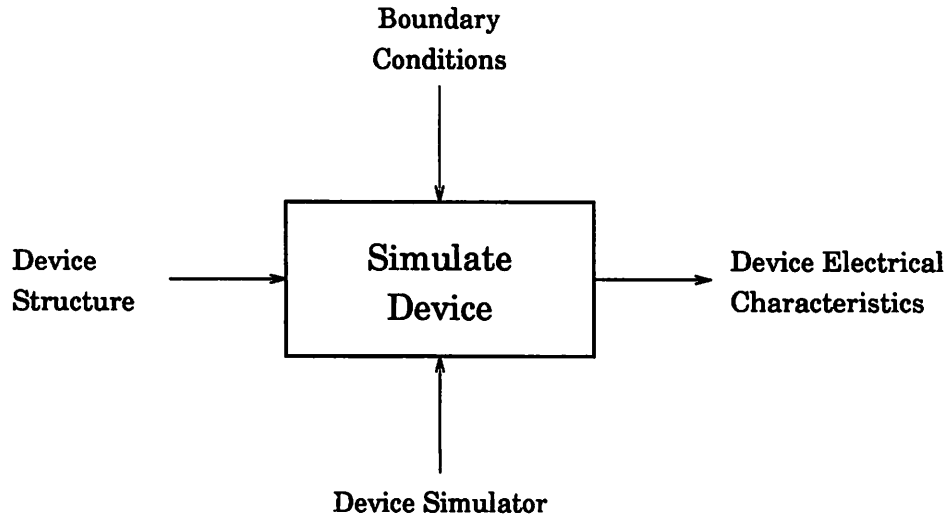


Figure 2.2: Device simulation – activity summary

a general-purpose device simulator such as PISCES [PINT85] or the DSIM program embedded in CODECS. In the following, a general discussion of device simulation is provided, with emphasis placed on the algorithms used in DSIM. For a more comprehensive treatment of semiconductor device simulation, the reader is referred to either [SELB84] or [PINT90].

2.3.1 Device Description

A device is a three-dimensional section of an IC substrate that forms an active component such as a transistor or a passive component such as a resistor. A number of different materials are used in the construction of a device. The description

must provide both the geometries and properties of these materials. In addition, the spatial distributions of dopant atoms, or *doping profile*, must be described. In order to simplify the description and subsequent model of a device, symmetries in a device are exploited whenever possible. In many cases, axial symmetry is used to reduce the full three-dimensional problem to one or two dimensions. Bilateral symmetry allows the behavior of a device to be obtained by simulating only half the device and doubling or halving the results as needed.

While small circuits can be described conveniently in either textual or graphical form, devices represent a more difficult specification problem. Text-only descriptions suffer from the problem that errors are easy to generate and difficult to detect. While this problem is also present in circuit description, it is more important at the device level because of the large amount of spatial/geometric information that must be provided. As a result, graphical *device-capture* programs such as PICASSO [SIMP91] have been developed as aids for this problem. However, for devices described in terms of rectangular geometries, a textual input language can prove to be adequate, if not necessarily ideal.

Because of the difficulties involved in producing device descriptions manually, it is desirable to provide a direct interface to a process simulator such as SUPREMIII [HO83]. Process simulation can then automatically produce geometries and doping profiles based on a process recipe description and mask layout information. This interface is complicated by the fact that the process simulator and device simulator may use different representations of the device structure, and utility routines may need to be used to convert between them.

2.3.2 Semiconductor Device Equations

An IC device typically contains three types of materials: semiconductors, insulators, and metals. Equations governing the electrical behavior of these materials must be provided for the material interiors and at the boundaries between different materials. Semiconductor regions are most often modeled using the drift-diffusion system of equations [ROOS50]:

$$\nabla \cdot (\epsilon \boldsymbol{E}) = q(p - n + N_D^+ - N_A^-) + \rho_F \tag{2.11}$$

$$\frac{1}{q}\nabla \cdot \boldsymbol{J}_n = \frac{\partial n}{\partial t} - (G - R) \tag{2.12}$$

$$\frac{1}{q}\nabla \cdot \boldsymbol{J}_p = -\frac{\partial p}{\partial t} + (G - R) \tag{2.13}$$

where

$$\boldsymbol{E} = -\nabla \Psi \tag{2.14}$$

$$\boldsymbol{J}_n = q\mu_n n \boldsymbol{E}_n + qD_n \nabla n \tag{2.15}$$

$$\boldsymbol{J}_p = q\mu_p p \boldsymbol{E}_p - qD_p \nabla p \tag{2.16}$$

and

ϵ	=	material dielectric constant (F/cm)
q	=	electron charge (C)
Ψ	=	electrostatic potential (V)
$n\,(p)$	=	electron (hole) concentration (/cm^3)
$\boldsymbol{E}$	=	electric field (V/cm)
$N_D^+\,(N_A^-)$	=	ionized donor (acceptor) concentration (/cm^3)
ρ_F	=	fixed charge density (C/cm^3)
$\boldsymbol{J}_n\,(\boldsymbol{J}_p)$	=	electron (hole) current density (A/cm^2)
$\boldsymbol{E}_n\,(\boldsymbol{E}_p)$	=	electron (hole) driving field (V/cm)
G	=	net volume generation rate (/cm^3·s)
R	=	net volume recombination rate (/cm^3·s)
$\mu_n\,(\mu_p)$	=	electron (hole) mobility (cm^2/V·s)
$D_n\,(D_p)$	=	electron (hole) diffusivity (cm^2/s)

Equation 2.11 is Poisson's equation, and Equations 2.12 and 2.13 are, respectively, the electron and hole current-continuity equations. In certain cases, one or both of the continuity equations can be eliminated if it is known that the flow of a particular carrier type is negligible. For example, at thermal equilibrium, no average current flows, and only Poisson's equation needs to be solved. In these equations, Ψ, n and p are the basic variables which characterize the state of the semiconductor. The remaining parameters are functions of these variables and/or physical properties of the semiconductor material. Models for the physical parameters are needed to complete the basic set of semiconductor equations. Ideally these models would be derived from fundamental principles of device physics. However, in most cases, empirical or semi-empirical expressions are used instead. In either situation, it is usually necessary to calibrate the models by comparison to measurements performed under a wide

range of conditions if quantitative accuracy is desired. However, it is useful for these parameters to be user-accessible, so that by varying them their influence on device operation can be determined.

In the insulating regions, it is assumed that there is no flow of charge carriers such as electrons, holes or impurity atoms. In a well-designed device in its normal region of operation, this is a very good assumption. As a result, only Poisson's equation needs to be solved in insulating regions. Insulator-semiconductor interfaces can be easily treated, provided that care is taken to account for the different dielectric constants of the two materials and the possibility of fixed interface charge $\rho_{F,s}$ (C/cm^2) and surface generation-recombination $(G - R)_s$ (cm^2·s).

In metallic or highly conductive regions of the device, a constant potential Ψ is assumed and the current distribution within the conductor is ignored. Therefore, the effect of the metal is only considered along its boundaries, where it makes contact with the other materials in the device. These contacts are invariably made either to insulators or semiconductors, since metal-to-metal contacts are short circuits. However, a single metal region may overlap both insulator and semiconductor regions and this can present some difficulty in determining appropriate boundary conditions.

2.3.3 External Device Boundary Conditions

The operation of a device is controlled by the application of external voltages and/or currents to its metallic contacts. Voltage boundary conditions are more common, where the terminal currents i are defined as functions of the voltages V:

$$i = I(w(x), V) \tag{2.17}$$

where $w(x)$ represents the spatially-varying internal device state ($\Psi(x)$, $n(x)$, $p(x)$). These currents are calculated by integrating the total current density, J_T, over the surface of a contact. The total current density is defined by the relation:

$$J_T = J_n + J_p + J_d \tag{2.18}$$

where $J_d = \epsilon \frac{\partial E}{\partial t}$ is the displacement current density. It arises from the time derivatives of the carrier concentrations in the continuity equations [PINT90] and reflects the buildup or decline of space-charge regions within a device.

Each applied voltage is used to set up Dirichlet or mixed Dirichlet-Neumann boundary conditions along the boundaries of one of the device's contacts. The exact form these boundary conditions take depends on the nature of the contact and the assumptions made to make the equations tractable. For example, the ohmic source/drain contacts of a MOSFET need to be treated differently from the rectifying contact of a Schottky-clamp diode.

In addition to electrical bias conditions, the ambient device operating temperature T_0 (°K) is a key input to the device simulator, since device behavior is very temperature-sensitive. The operating temperature enters directly into the equations used for the physical models. It is usually assumed that the semiconductor lattice is in thermal equilibrium with its environment at T_0. However, the basic device equations can be augmented by a heat-flow equation if the effects of thermal gradients need to be investigated.

2.3.4 Scaling and Space Discretization

The semiconductor device equations are a coupled system of nonlinear partial differential equations (PDEs) in space and time. Because the solution variables Ψ, n, and p have widely varying values it is customary to scale the various physical quantities in order to equilibrate the equations. Of the various scaling approaches that have been used successfully in device simulation, DSIM employs the scaling approach used in SEDAN [YU85].

As in the case of circuit transient analysis, it is not generally possible to solve the device equations analytically, and numerical methods must be employed. The first step in analyzing these equations is space discretization, where the solution is approximated at a finite set of points in space, known as a *mesh*. An example mesh is shown in Figure 2.3. Equations are formulated for each point or *node* in the mesh using either a finite-difference or a finite-element method. In DSIM, the *finite-box* method, a variant of the finite-difference method, is used to discretize the device equations on a rectangular tensor-product mesh. In the box method, fields and currents are approximated along the *edges* of the mesh, flowing perpendicular to the sides of the box. Discretization of the device PDEs produces a system of nonlinear

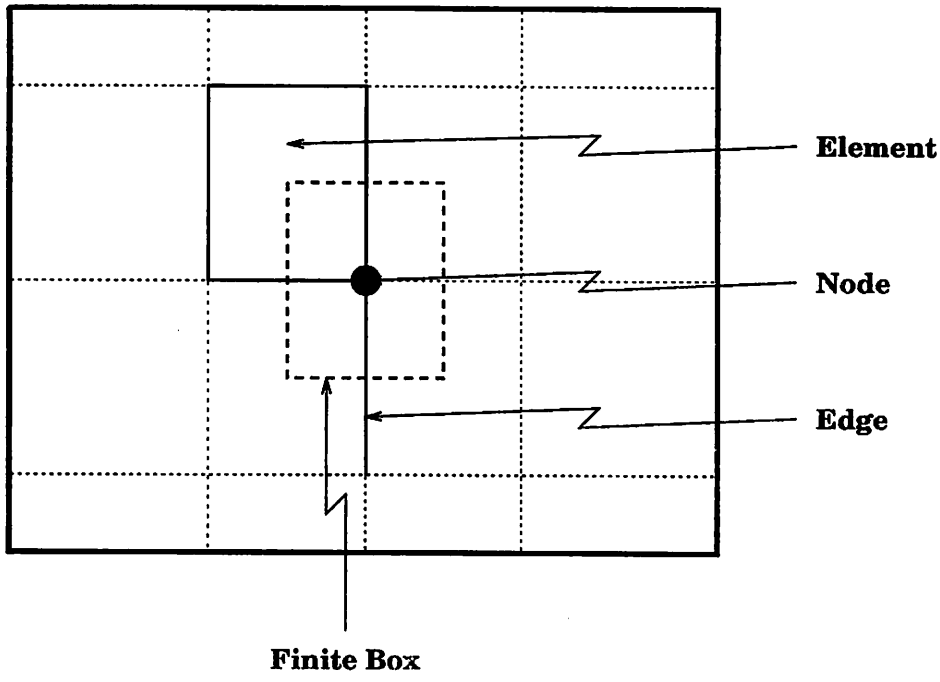


Figure 2.3: Mesh for finite box discretization

ordinary differential-algebraic equations in time, expressed symbolically as:

$$G_\Psi(\Psi, n, p) = O \tag{2.19}$$

$$G_n(\Psi, n, p) - \frac{\partial n}{\partial t} = O \tag{2.20}$$

$$G_p(\Psi, n, p) + \frac{\partial p}{\partial t} = O \tag{2.21}$$

or more compactly as:

$$G(\dot{w}(t), w(t), V(t)) = O \tag{2.22}$$

where $w = (\Psi, n, p)$ is the vector of unknown nodal approximations to the continuous solution, and the dependence on V, the vector of applied voltages, has been explicitly included. This system has three equations for each node lying in a semiconductor region, one for each node in an insulating region, and none for nodes inside metals. At material boundaries, the number of equations used depends on the type of boundary conditions applied. The total number of equations is always less than or equal to $3N$, where N is the number of nodes in the mesh.

The mesh should have high node density in regions where the solution varies rapidly, and lower density where the solution is constant. Since the solution changes with the external boundary conditions, the mesh should ideally adapt itself during an analysis. However, robust adaptive mesh generation is still an open research problem, and in DSIM a fixed mesh is employed. This approach is effective because reasonable meshes for the standard IC devices can be designed *a priori* from a knowledge of the basic device physics. One benefit of this approach is the elimination of the computational burden associated with restructuring the mesh after the solution process has begun. However, this advantage is offset by the fact that a fixed mesh appropriate for a wide range of bias conditions is generally computationally less efficient than one optimized to a particular solution.

Once the solution has been found on the discretized device domain, the terminal currents are calculated based on this solution. An appropriate discretization of the integral equation for each terminal current is defined which is compatible with the domain discretization. This converts the integral to a weighted summation of the total current density around the nodes belonging to the contact. Estimates for the different components of the total current density are then substituted into this summation to arrive at the current for one of the contacts. This process is repeated for all the device contacts.

2.3.5 DC and Transient Analyses

In DC steady state, the time dependence in Equation 2.22 can be eliminated, leaving a system of nonlinear algebraic equations:

$$G(w, V) = O \tag{2.23}$$

This can be solved using the basic Newton-Raphson algorithm as described in Section 2.2.3. However, better convergence is obtained if the solution updates Δw are damped in such a way that the norm of the right-hand-side is reduced. The basic Newton-Raphson step is repeated here:

$$J_w(w^k)\Delta w^k = -G(w^k) \tag{2.24}$$

where $J_w(w^k)$ is the Jacobian matrix of the device equations with respect to the unknowns w:

$$J_w(w^k) = \begin{bmatrix} \frac{\partial G_\psi}{\partial \psi} & \frac{\partial G_\psi}{\partial n} & \frac{\partial G_\psi}{\partial p} \\ \frac{\partial G_n}{\partial \psi} & \frac{\partial G_n}{\partial n} & \frac{\partial G_n}{\partial p} \\ \frac{\partial G_p}{\partial \psi} & \frac{\partial G_p}{\partial n} & \frac{\partial G_p}{\partial p} \end{bmatrix}$$

The derivatives of Equation 2.23 with respect to the applied voltages (J_V) can be ignored for the moment, since the applied voltages are constant. In the damped Newton-Raphson method, the update step is modified to include the damping factor, λ^k:

$$w^{k+1} = w^k + \lambda^k \Delta w^k \tag{2.25}$$

where λ^k is chosen according to the criterion:

$$\left\| G(w^{k+1}) \right\| < \left\| G(w^k) \right\|$$

Since an acceptable value for λ is not known *a priori*, a search procedure must be used to find one [BANK81].

In DSIM, the initial guess w^0 is constructed in a hierarchical fashion. First, the charge-neutral solution (the solution of Poisson's Equation ignoring the left-hand-side) is used as an initial guess to the equilibrium solution, where the continuity equations are ignored. The equilibrium solution is then used to solve the full set of equations at equilibrium. This iteration usually converges in two steps. After this, initial guesses can be obtained by projecting a previous solution using the Jacobian matrices, J_w and J_V, and a set of voltage steps, ΔV.

The linear systems of equations encountered in DC device simulation have a number of interesting properties, some of which are different from those typical of circuit matrices. Each node of the mesh contributes a small set of equations to the overall problem. If the equations for a node are treated as a single block, the Jacobian matrix can be viewed as a collection of coupling blocks, C. A block C_{ij} is structurally non-zero if any of the equations at node i depend on the variables at node j. The occurrence of coupling depends on the exact nature of the discretization scheme, but in general neighboring nodes are coupled and isolated nodes are not. This leads to a great deal of sparsity in the matrix, since most node pairs are not neighbors. In one and two dimensions, L/U decomposition can be used to solve this sparse system

of equations. In three dimensions, the matrices become too large to solve using direct methods and iterative methods are used instead [WEBB91].

In addition to sparsity, it is usually true that if a node is coupled to a neighbor, then the neighbor is coupled back to that node, which implies block symmetry of the Jacobian matrix. This knowledge can be used to develop a special-purpose sparse matrix package that exploits symmetry in order to reduce the amount of pointer overhead. However, one important exception to the above occurs in the simulation of MOSFETs. In order to characterize properly the mobility in the MOSFET inversion layer, an integral equation must be satisfied along each line of nodes that crosses the inversion layer. This integral equation introduces non-local coupling into the device matrix and disrupts the block symmetry. Because DSIM employs this technique, the *general-purpose* package SPARSE is used to solve the device equations.

In transient analysis, the time dependence of w can no longer be ignored in Equation 2.22. The derivatives, $\frac{\partial n}{\partial t}$ and $\frac{\partial p}{\partial t}$, are discretized in time using an implicit integration formula (cf. Section 2.2.4). The initial solution $\{w_0, V_0\} = \{w(0), V(0)\}$ is computed using a DC operating point analysis. After this, the analysis proceeds exactly as in the circuit case, selecting one time step after another until the analysis interval is exhausted. In solving the nonlinear systems that arise, it is not usually necessary to employ damping since continuity in time usually results in a new solution that is not far from the previous one. Time steps are selected using estimates of the local truncation error incurred. This estimate is usually found by computing the L2 norm of a vector of error estimates obtained at each node of the simulation mesh. For more detail on this procedure, consult either [BANK85] or [MAYA88].

2.3.6 Small-Signal AC Analysis

The response of the internal device state to small sinusoidal variations in the voltage and current boundary conditions can be calculated using methods similar to those employed in circuit analysis. The internal state w is represented as the sum of a steady-state component w and a small phasor component $\tilde{w}e^{j\omega t}$ where $\tilde{w}$ is a complex vector. The applied voltages V are treated similarly. After substitution of these values into Equation 2.22 and retention of the linear terms, the following complex-valued

linear system of equations results:

$$[J_{\dot{w}} \cdot j\omega + J_w]\,\tilde{w} = -J_V \tilde{V} \tag{2.26}$$

where J_w is the DC Jacobian matrix, $J_{\dot{w}}$ is the (diagonal) Jacobian matrix for the time derivatives:

$$J_{\dot{w}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{\partial G_n}{\partial \dot{n}} & 0 \\ 0 & 0 & \frac{\partial G_p}{\partial \dot{p}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & I \end{bmatrix}$$

and J_V is the voltage Jacobian matrix:

$$J_V = \begin{bmatrix} \frac{\partial G_\psi}{\partial V} \\ \frac{\partial G_n}{\partial V} \\ \frac{\partial G_p}{\partial V} \end{bmatrix}$$

Equation 2.26 can be solved using complex L/U decomposition to find the small-signal quantities $\tilde{w}$. The decomposed matrix data structure set up during DC analysis can be reused during this process, leading to significant savings in computation overhead. However, faster, iterative methods have been developed that use only real arithmetic [LAUX85], [APTE92]. Time is saved primarily because the previously calculated DC Jacobian L/U factors, in addition to the data structures, are used in these algorithms. Iterative methods are unfortunately limited in their range of applicability, and a scheme must be established to switch to a more robust method upon failure. Ultimately, such a scheme returns to direct methods when all iterative methods fail. This approach is used in CODECS where the successive over-relaxation (SOR) method of [LAUX85] is used until it fails, at which point a switch is method to direct-method solution using the complex arithmetic capabilities of SPARSE.

2.3.7 Visualization and Representation of Device Behavior

Device simulation presents a more difficult visualization problem than circuit simulation because of the need to manage multidimensional data sets. Typically, accuracy constraints require two- or three-dimensional simulations to be performed. Even in cases where one-dimensional simulations are adequate, the addition of time as an independent variable in transient analyses generates a multidimensional data

set. Thus, a need exists to go beyond the simple capabilities required during circuit simulation.

For general mesh structures, the problem of storing internal device variables such as the potential and the electron and hole concentrations is complicated by the irregularity of the mesh. A special-purpose file format accompanied by a procedural interface can be used to handle such problems. An example is the Vset portion of the HDF file format developed by the National Center for Supercomputing Applications [GROU90]. Recent versions of PISCES use this format to store results. A separate *log* file is used to record the terminal voltages and currents. However, in the special case of a tensor product mesh, a much simpler approach can be used. The rawfile format and interface of SPICE3 was extended for version 3F2 [JOHN92] to support multi-dimensional data sets. The only additional information needed is a list of the array dimensions. This format allows CIDER to store one- and two-dimensional device internal states in standard SPICE3 output files. The terminal voltages and currents, as well as small-signal conductances and capacitances, are stored in a separate data set at the beginning of the same file. This is possible because the rawfile format supports multiple data sets per file.

Numerous methods of visualization have been developed to allow exploration of multidimensional data sets arising from scientific computations. Many of these techniques such as three-dimensional projection, animation and the sophisticated used of color graphics can be used in interpreting the results of device simulation. Unfortunately, these techniques generally rely on special-purpose workstations for the necessary computational power and color graphics hardware. Three simpler techniques that can be used on typical general-purpose engineering workstations and black-and-white laser printers are demonstrated below. In Figure 2.4, multiple slices through a two-dimensional data set, taken perpendicular to the Y axis, are plotted on the same set of Cartesian axes. This method is compatible with the existing visualization capabilities of the circuit simulation post-processor NUTMEG. The same data can be viewed from above using a contour plot (multiple Z axis slices), as shown in Figure 2.5. On a workstation, it is often possible to enhance contour definition by using different colors to fill the regions between contours. Finally, variations in the data are probably best exposed using three-dimensional projection techniques as in Figure 2.6. Again, on a workstation, color enhancement can be used to supplement the projection. In addition,

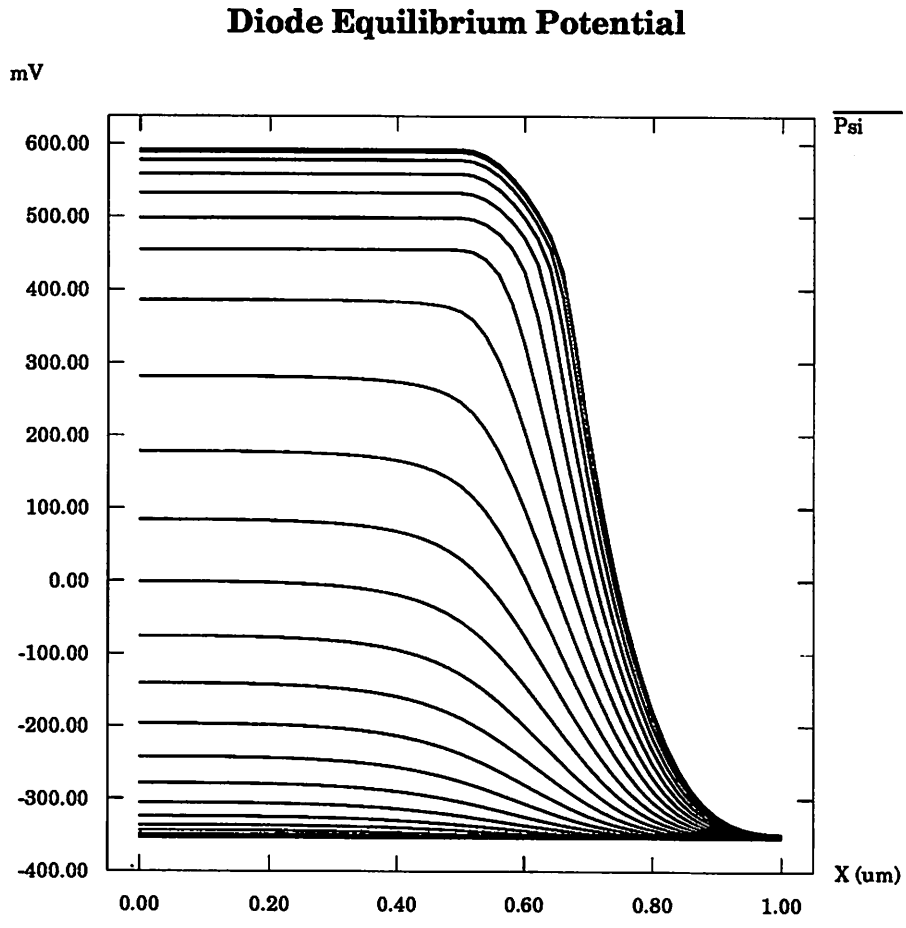


Figure 2.4: Diode potential data set - multiple slices

the data can be viewed from a number of different angles by interactively changing the viewing location.

2.4 Mixed-Level Circuit and Device Simulation

Figure 2.7 is a representation of the mixed-level circuit and device simulation process. This process transforms descriptions of a circuit and of the structures of its critical devices into summaries of both the circuit performance and the internal device behaviors. The operating environment of the circuit must also be provided. This process is supported by a general-purpose circuit and device simulator such as MEDUSA [ENGL82] or CODECS [MAYA88].

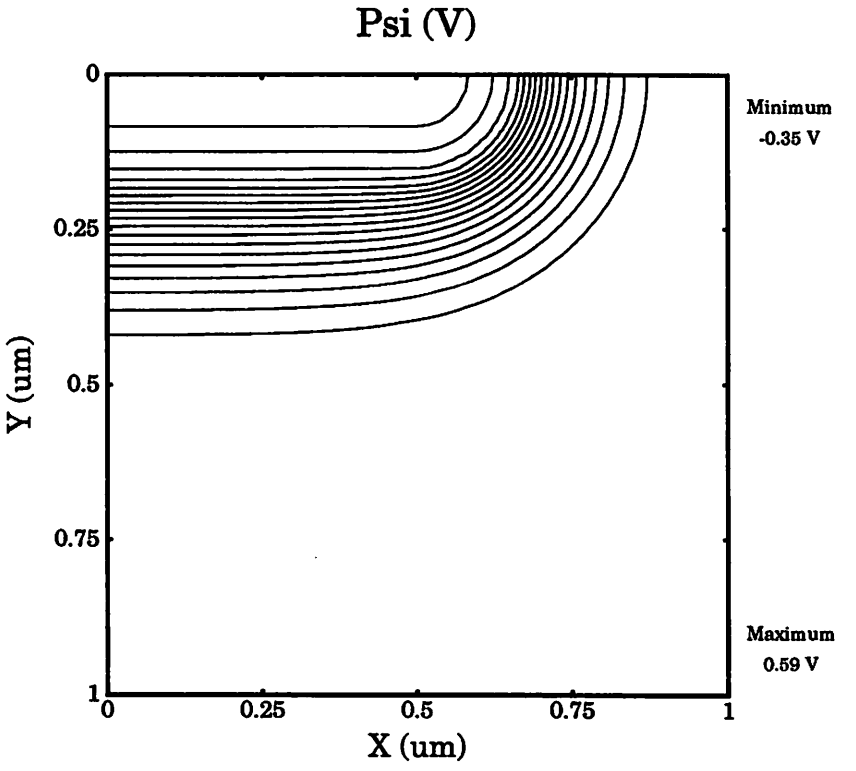


Figure 2.5: Diode potential data set - contour plot

2.4.1 Coupled Circuit and Device Description

In mixed-level circuit and device simulation, a unified means to describe the circuit connectivity, the compact-model parameters, and the numerical-device structures must be supplied. Perhaps the easiest way to accomplish this is by augmenting an existing circuit-specification language with special commands for the numerical devices. In CODECS the existing technique for describing device models to the circuit simulator SPICE3 is also used to describe numerical device models. This approach is simple to implement but lacks flexibility because the model-description features of the SPICE-input syntax were developed with compact models in mind. A better but more difficult approach is to combine a language for circuits and a language for devices, so that both levels can be described efficiently. This approach is taken in CIDER, where the existing parser of SPICE3 has been extended so that device structures can be described using a PISCES-like format. Special provision is made for specification of circuit-level parameters that vary from device to device such as layout areas or widths.

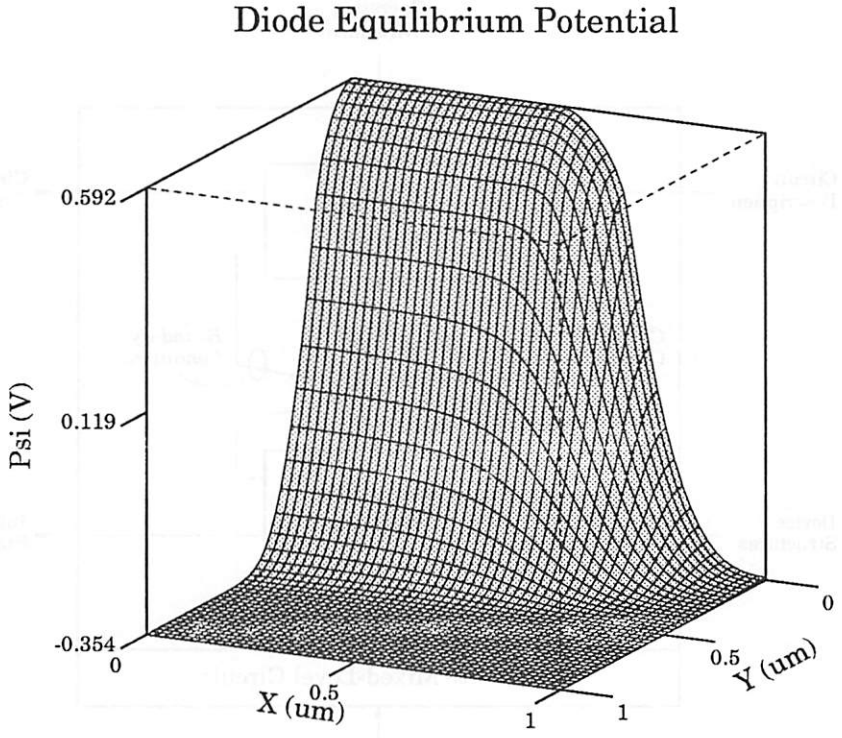


Figure 2.6: Diode potential data set - birdseye view

These provisions are typically unnecessary in pure device simulation where a single unit-size device is analyzed. A detailed description of the CIDER input format is provided as Appendix A. In the future, the details of this input language could be hidden behind a unified graphical user interface (GUI) that supports both schematic capture and device capture. For example, a standard circuit schematic capture program can be extended using special symbols for the numerical devices [TMA91] where a means is provided for linking these symbol instances to device structures created by a device capture program. The unified GUI is then responsible for translating the graphical description into a binary form such as a CAD database, or into a textual form such as a file written in a unified input language. The mixed-level simulator reads the database or the input file to obtain the circuit and device descriptions.

One large system of equations can be formed which describes the complete behavior of a mixed-level circuit. Kirchoff's current and voltage laws and the branch-constitutive relations of the compactly modeled devices are to used to create part of

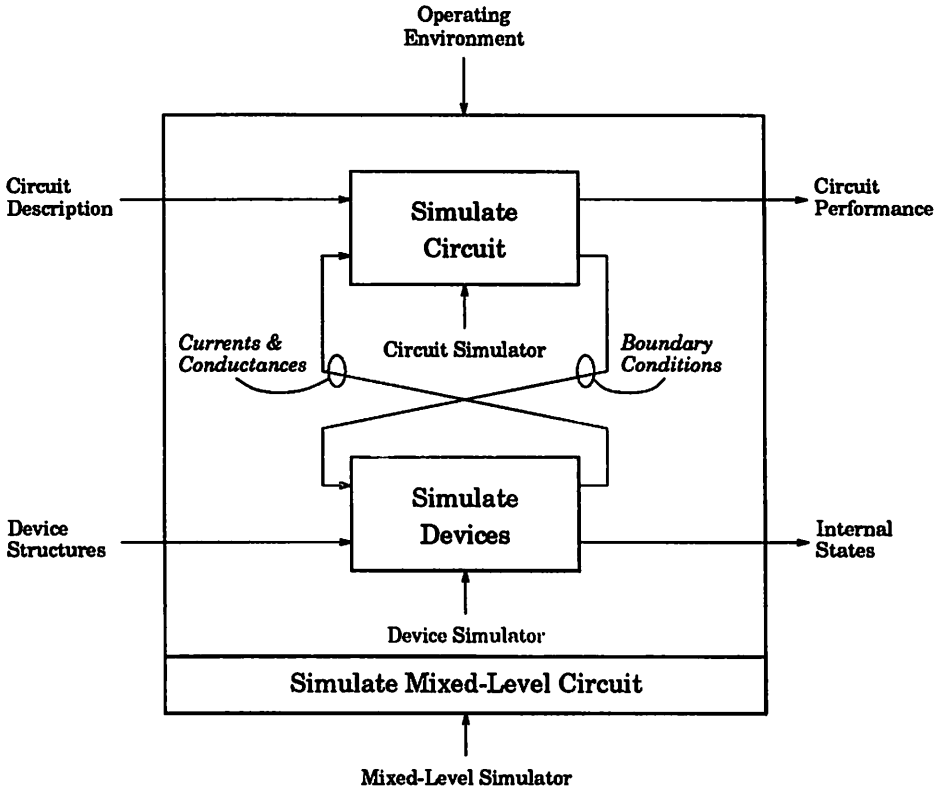


Figure 2.7: Mixed-level simulation – activity summary

this system of equations. Discretized semiconductor device equations are used for the internal states of the numerically modeled devices. Finally, auxiliary branch-constitutive relations are needed for the numerically modeled devices. In CODECS, where only voltage boundary conditions are allowed, these auxiliary equations relate a device's internal state and its branch voltages to its terminal currents (cf. Equation 2.17). Coupling between the circuit-level equations and those of a single device is achieved by establishing a one-to-many correspondence between the circuit nodes and the device's terminals. For example, in Figure 2.8, a single circuit node is coupled to the dual base contacts of a bipolar-transistor device structure. Single contacts are made to the emitter and collector.

2.4.2 Coupled Operating Conditions

The major difference between the operating environment in mixed-level simulation when compared to either circuit or device simulation alone is that the numerical

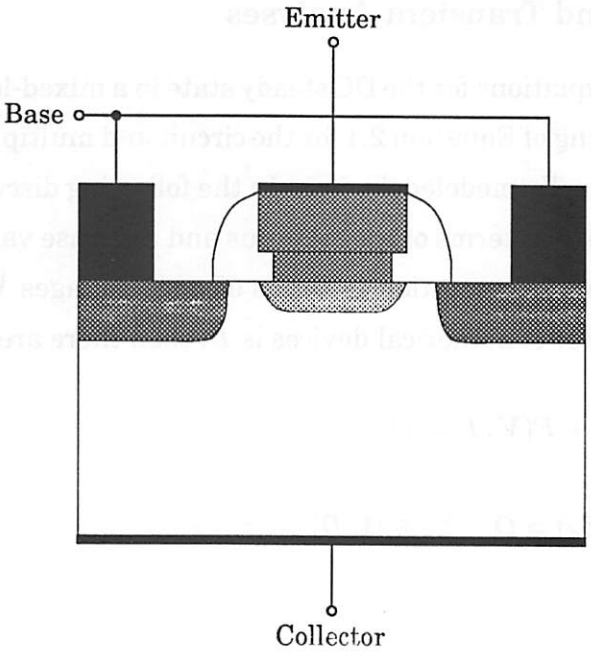


Figure 2.8: Polyemitter bipolar transistor with dual base contacts

devices are no longer operating in isolation. When a numerical device is embedded in a circuit it is not necessary to specify values for the device terminal voltages ahead of time. During the natural course of the solution process, the circuit simulator automatically generates bias conditions for the device. Eventually, a consistent set of circuit node voltages and branch currents and device internal states is obtained.

If the device-structure description is suitably parameterized, preprocessors that support statistical analyses and optimization can be used to modify the device structure prior to simulation. This can be useful in the optimization of a device structure to achieve specified levels of circuit performance. It can also be used to study the direct influence of device physical parameters on circuit performance. In addition, these same preprocessors can be used when optimizing parameters that vary from device to device such as layout widths and areas. Finally, the operating temperature of each device should be separately specifiable so that the effects of non-uniform heat generation in an IC can be simulated. For example, CIDER extends the temperature-dependent sections of CODECS in order to support this ability.

2.4.3 DC and Transient Analyses

The equations for the DC steady state in a mixed-level analysis are obtained by suitable coupling of Equation 2.1 for the circuit and multiple instances of Equation 2.23 for the numerically modeled devices. In the following discussion, representation of the circuit equations in terms of the stimulus and response variables (s,r) is replaced with an alternative representation in terms of node voltages V and branch currents I. If the total number of numerical devices is D, then there are $D+1$ systems of equations:

$$\sum_{d=1}^{D} A_d i_d + F(V, I) = O \tag{2.27}$$

$$G_d(w_d, E_d) = O \quad \forall d \in [1, D]$$

where

$$i_d = I_d(w_d, E_d)$$

$$E_d = A_d^T V$$

Coupling between these systems is completely characterized by the A_d matrices, the node-to-branch incidence matrices of the numerical devices. They determine how to compute the numerical device branch voltages E_d and how to feed the device currents i_d into the rest of the circuit. Note especially that this implies there is no direct coupling between the internal device states, w_d, of different devices.

The mixed-level equations for transient simulation are very similar to those obtained in DC analysis. Once again, time discretization formulae are used to convert the time derivatives into algebraic expressions involving the present and previous states. After discretization, the time-dependent problem can then be treated using methods appropriate for DC analysis. One interesting point is that the discretization method need not be the same at both the circuit and device levels. For example, CODECS employs a mixed method where the trapezoidal rule [NAGE75] is used at the circuit level and Gear's backward differentiation formulae [BRAY72] are used at the device level. In addition to discretization, a time step selection algorithm must be used during transient analysis. The approach taken in CODECS is to calculate maximum allowable time steps for each of the numerical devices using the techniques mentioned

in Section 2.3.5. Then, at the circuit level, these estimates are treated identically to those obtained from compactly modeled devices.

The complete mixed-level simulation problem as described above is a system of nonlinear algebraic equations. As such, one natural technique for solving these equations is the Newton-Raphson method. However, the modular structure of the equations is particularly well suited to the multi-level Newton method [GUY 79], an extension of the basic technique. In [MAYA92], several strategies for solving the DC and transient equations are evaluated. Based on this evaluation, different techniques are recommended for the two types of problems. The standard or full Newton algorithm is sufficient for transient simulation. However, a modified two-level Newton scheme is employed during DC analysis since it has been shown to be more robust [MAYA92]. These two methods are now described as implemented in CODECS.

The mixed-level equations after space and time discretization can be represented more compactly as:

$$\boldsymbol{F}_C^*(\boldsymbol{W}, \boldsymbol{Z}) = \boldsymbol{O} \tag{2.28}$$

$$\boldsymbol{G}_d^*(\boldsymbol{w}_d, \boldsymbol{Z}) = \boldsymbol{O} \quad \forall d \in [1, D]$$

where $\boldsymbol{Z}$ is the complete vector of circuit variables $(\boldsymbol{V}, \boldsymbol{I})$ and $\boldsymbol{W}$ is the complete vector of device internal variables $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_D)$. The linearized equations are first expressed in terms of the solution updates, $\Delta \boldsymbol{w}_d$ and $\Delta \boldsymbol{Z}$, according to:

$$\sum_{d=1}^{D} \boldsymbol{J}_{I,d} \Delta \boldsymbol{w}_d + \boldsymbol{J}_C \Delta \boldsymbol{Z} = -\boldsymbol{F}_C^* \tag{2.29}$$

$$\boldsymbol{J}_{w,d} \Delta \boldsymbol{w}_d + \boldsymbol{J}_{V,d} \Delta \boldsymbol{Z} = -\boldsymbol{G}_d^* \quad \forall d \in [1, D]$$

with

$$\boldsymbol{J}_{I,d} = \frac{\partial \boldsymbol{F}_C^*}{\partial \boldsymbol{w}_d}$$

$$\boldsymbol{J}_C = \frac{\partial \boldsymbol{F}_C^*}{\partial \boldsymbol{Z}}$$

$$\boldsymbol{J}_{w,d} = \frac{\partial \boldsymbol{G}_d^*}{\partial \boldsymbol{w}_d}$$

$$\boldsymbol{J}_{V,d} = \frac{\partial \boldsymbol{G}_d^*}{\partial \boldsymbol{E}_d} \cdot \boldsymbol{A}_d^T$$

Note that J_C, the circuit-level Jacobian matrix, includes terms due to the direct influence of the branch voltages E_d on the terminal currents i_d as well as terms stemming from the compactly modeled devices. In practice, it is more convenient to associate these extra terms with a device's other contributions, so that the first part of Equation 2.29 reads:

$$\sum_{d=1}^{D} (J_{I,d}\Delta w_d + J_{G,d}\Delta Z) + J_C'\Delta Z = -F_C^* \tag{2.30}$$

where

$$J_{G,d} = \frac{\partial F_C^*}{\partial E_d} \cdot A_d^T$$

and J_C' is a modified circuit-level matrix where the numerical device contributions have been removed.

The structure of the resulting linear system is shown in Figure 2.9, where the circuit-level equations are solved after all the device-level equations have been

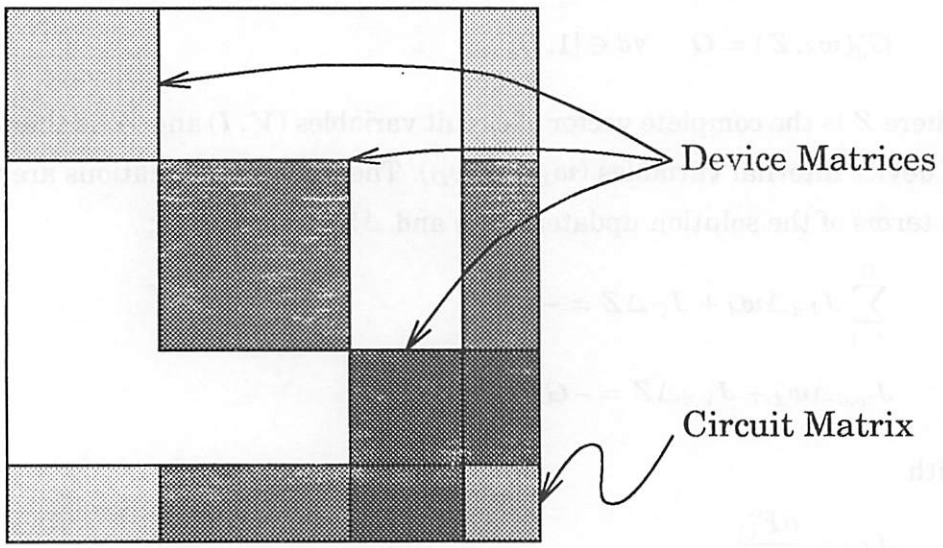


Figure 2.9: Block matrix structure of mixed-level system of equations

solved. The isolated diagonal blocks are the device-level Jacobian matrices $J_{w,d}$ and the circuit-level matrix J_C is in the lower-right corner. The off-diagonal blocks on the right are the matrices $J_{V,d}$ that capture the influence of the circuit voltages on the device internal states. Those on the lower edge are the matrices $J_{I,d}$ that incorporate

the influence of the device internal behavior on the terminal currents. In the full-Newton approach, this system is solved using block L/U decomposition. Each device-level Jacobian matrix is decomposed into L/U factors, and then a modified circuit-level system is assembled:

$$J_C^{**} = J_C' - \sum_{d=1}^{D} \left(J_{I,d} J_{w,d}^{-1} J_{V,d} - J_{G,d} \right) \tag{2.31}$$

$$F_C^{**} = F_C^{*} - \sum_{d=1}^{D} J_{I,d} J_{w,d}^{-1} G_d^{*}$$

so that the circuit-level Newton step becomes:

$$J_C^{**} \Delta Z = -F_C^{**} \tag{2.32}$$

Not that in no case is a true matrix inverse J^{-1} actually calculated. Forward and back substitution using the L/U factors is used instead.

In CODECS consistency with the circuit equation formulation of SPICE demands that additional terms be added to the RHS so that the new solution Z^{k+1} can be computed directly from the previous one, Z^k:

$$J_C^{**} Z^{k+1} = J_C^{**} Z^k - F_C^{**} \tag{2.33}$$

This is the same modification that is performed in the pure circuit simulation problem (cf. Equation 2.3). Solution of these equations results in a new set of circuit-level variables. These must then be back-substituted into each of the device-level systems in order to obtain solution updates for the device internal states:

$$\Delta w_d = -J_{w,d}^{-1} G_d^{*} - J_{w,d}^{-1} J_{V,d} \Delta Z \quad \forall d \in [1, D] \tag{2.34}$$

In order to satisfy the constraint that the device-level evaluation routines are only accessed once per iteration in SPICE, this final step can be deferred to the beginning of the next iteration. In order to eliminate redundant computations, the common subexpressions $-J_{w,d}^{-1} G_d^{*}$ and $J_{w,d}^{-1} J_{V,d}$ in Equations 2.31 and 2.34 are calculated once while assembling the circuit-level system and reused during the update step.

Solution of the DC equations differs from the above full-Newton algorithm in that multiple Newton steps can be taken at the device-level for each step at the circuit-level. During each circuit-level iteration, all device-level systems are solved to

convergence. At the first iteration, the initial circuit variables are used to compute device-level boundary conditions. On subsequent iterations, Equation 2.34 is used to obtain an update to a device's internal state:

$$\Delta w_d = -J_{w,d}^{-1} J_{V,d} \Delta Z \tag{2.35}$$

where the first term in Equation 2.34 has been eliminated since $G_d^* = O$ at the previous solution. This update is added to the previous solution to obtain an initial guess for the new device internal state. Because the circuit state is fixed during these device-level iterations, various techniques to ensure robust convergence can be employed such as damping, voltage-step limiting and voltage-step backtracking [MAYA88].

Once the device-level equations have converged, the circuit-level matrix and RHS are obtained using Equation 2.31:

$$J_C^{**} = J_C' - \sum_{d=1}^{D} \left(J_{I,d} J_{w,d}^{-1} J_{V,d} - J_{G,d} \right) \tag{2.36}$$

$$F_C^{**} = F_C^*$$

where $G_d^* = O$ has been used once again to eliminate unnecessary computation. It is worthwhile noting at this point that the summation on the right-hand side of Equation 2.36 is just an accumulation of the conductance matrices of the numerical devices. That is to say, as shown in [MAYA92], the conductance matrix of a numerical device is identical to $-J_{I,d} J_{w,d}^{-1} J_{V,d} + J_{G,d}$. In addition, equivalent linearized currents for the numerical devices contribute terms to the value of F_C^*. As a result, during implementation, the standard method of adding conductances and currents into the circuit-level matrix known as *stamping* can be used. Only the procedures for calculating these values have changed in the case of numerical devices.

A flowchart of the full-Newton algorithm as used during transient analyses is shown in Figure 2.10. In the following description, labels for the various steps are surrounded by parentheses. The analysis begins by establishing a DC operating point and choosing an initial timestep. The time discretization data structures are then initialized and the main Newton-Raphson loop begins. On each iteration, the compactly modeled devices are linearized and their contributions are loaded into the circuit matrix and RHS (MODcontrib). Then each of the numerical devices is evaluated. The device internal solution is updated appropriately depending on whether

or not this is the first iteration after a timepoint has been accepted (DEVmisc). The physical models (mobility, recombination) are then evaluated and the device's matrix and RHS for the semiconductor PDEs are loaded (DEVload). This system is factored and solved (DEVfactor and DEVsolve) leading to a new internal solution. Convergence of the device internal updates is also checked now (DEVmisc), but the result is not used until later at the circuit-level. Based on this solution, the device's linearized currents and conductances are calculated and loaded into the circuit matrix and RHS (DEVcontrib). Collectively the preceding steps constitute the complete circuit loading phase (CKTload). After the circuit matrix and RHS have been loaded, they are factored and solved (CKTfactor and CKTsolve). This results in a new set of voltages and currents which are checked for convergence using circuit-level tests as well as the results of the device-level checks. If all the updates are small enough, the Newton-Raphson loop terminates and the error in the solution for this timestep is estimated. This invokes calls to the truncation error routines for all time-varying elements including compact devices (MODtrunc) and numerical devices (DEVtrunc). The size of the error is used to determine whether the current timepoint is either accepted or rejected, and then a new timestep is selected. If more time is left in the simulation interval, the algorithm returns to the top of the time loop, and the process begins again.

2.4.4 Small-Signal AC Analysis

The mixed-level equations for small-signal AC analysis can be derived from the equations for the general transient behavior of the circuit:

$$\sum_{d=1}^{D} \boldsymbol{A}_d \boldsymbol{i}_d(t) + \boldsymbol{F}(\dot{\boldsymbol{V}}(t), \boldsymbol{V}(t), \dot{\boldsymbol{I}}(t), \boldsymbol{I}(t)) = \boldsymbol{O} \tag{2.37}$$

$$\boldsymbol{G}_d(\dot{\boldsymbol{w}}_d(t), \boldsymbol{w}_d(t), \boldsymbol{E}_d(t)) = \boldsymbol{O} \quad \forall d \in [1, D]$$

where

$$\boldsymbol{i}_d(t) = \boldsymbol{I}_d(\dot{\boldsymbol{w}}_d(t), \boldsymbol{w}_d(t), \boldsymbol{E}_d(t))$$

$$\boldsymbol{E}_d(t) = \boldsymbol{A}_d^T \boldsymbol{V}(t)$$

All quantities are represented as the sum of DC bias components and small-signal phasor components. Equation 2.37 is linearized about the operating point and the small-signal terms are retained:

$$\sum_{d=1}^{D} \boldsymbol{A}_d \tilde{\boldsymbol{i}}_d + \left[\frac{\partial \boldsymbol{F}}{\partial \dot{\boldsymbol{V}}} \cdot j\omega + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{V}}\right] \tilde{\boldsymbol{V}} + \left[\frac{\partial \boldsymbol{F}}{\partial \dot{\boldsymbol{I}}} \cdot j\omega + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{I}}\right] \tilde{\boldsymbol{I}} = \boldsymbol{O} \tag{2.38}$$

$$\left[\frac{\partial \boldsymbol{G}_d}{\partial \dot{\boldsymbol{w}}_d} \cdot j\omega + \frac{\partial \boldsymbol{G}_d}{\partial \boldsymbol{w}_d}\right] \tilde{\boldsymbol{w}}_d + \frac{\partial \boldsymbol{G}_d}{\partial \boldsymbol{E}_d} \tilde{\boldsymbol{E}}_d = \boldsymbol{O} \quad \forall d \in [1, D]$$

where

$$\tilde{\boldsymbol{i}}_d = \left[\frac{\partial \boldsymbol{I}_d}{\partial \dot{\boldsymbol{w}}_d} \cdot j\omega + \frac{\partial \boldsymbol{I}_d}{\partial \boldsymbol{w}_d}\right] \tilde{\boldsymbol{w}}_d + \frac{\partial \boldsymbol{I}_d}{\partial \boldsymbol{E}_d} \tilde{\boldsymbol{E}}_d$$

$$\tilde{\boldsymbol{E}}_d = \boldsymbol{A}_d^T \tilde{\boldsymbol{V}}$$

The contribution of device d to the circuit-level equation can be represented in a more compact way as:

$$\boldsymbol{A}_d \boldsymbol{Y}_d \boldsymbol{A}_d^T \tilde{\boldsymbol{V}} \tag{2.39}$$

where $\boldsymbol{Y}_d = \tilde{\boldsymbol{i}}_d / \tilde{\boldsymbol{E}}_d$ is the matrix of device admittances:

$$\boldsymbol{Y}_d = \frac{\tilde{\boldsymbol{i}}_d}{\tilde{\boldsymbol{E}}_d} = \left[\frac{\partial \boldsymbol{I}_d}{\partial \dot{\boldsymbol{w}}_d} \cdot j\omega + \frac{\partial \boldsymbol{I}_d}{\partial \boldsymbol{w}_d}\right] \frac{\tilde{\boldsymbol{w}}_d}{\tilde{\boldsymbol{E}}_d} + \frac{\partial \boldsymbol{I}_d}{\partial \boldsymbol{E}_d} \tag{2.40}$$

This matrix is computed by solving the device-level equation for device d for $\tilde{\boldsymbol{w}}_d / \tilde{\boldsymbol{E}}_d$ using the techniques of Section 2.3.6. Note that division by the vector $\tilde{\boldsymbol{E}}_d$ is an ill-defined operation, and is used only as a notational convenience. In practice, the rows of the admittance matrix are computed sequentially. At each step, one of the branch voltages is assumed equal to a unity phasor $\tilde{E} = (1, 0)$ and all other voltages are set to zero. The vector of small-signal currents obtained under these conditions is equal to the row of the admittance matrix corresponding to the perturbed branch voltage. In addition, with these voltages as inputs, $\tilde{\boldsymbol{w}}_d$ and $\tilde{\boldsymbol{i}}_d$ can be computed directly without the need for vector division.

Solution of the circuit level equations proceeds by separating the small-signal voltages and currents, V and I, back into the original two sets: known stimuli s and unknown responses r. Terms involving the stimuli are moved to the right-hand side of Equation 2.38, and the resulting linear system of complex equations is solved for the

responses using L/U decomposition. If desired, the small-signal internal device state $\tilde{w}_d$ can then be found by using the correct small-signal branch voltages $\tilde{E}_d$ that depend upon the now available small-signal voltages $\tilde{V}$. However, this step is not performed in either CODECS or CIDER because it requires additional time and memory to do so.

2.4.5 Visualization and Representation of Mixed-Level Behavior

A flexible output format is needed to store both voltage and current data at the circuit level and the potential and carrier concentrations at the device-level. In addition, other types of data such as conductances, capacitances, mobilities and recombination rates may also be desired at times. Fortunately, the rawfile format supports a generic typing mechanism that can be used to extend the basic set of data types built into the NUTMEG frontend. Using this mechanism, and the multidimensional capabilities mentioned in Section 2.3.7, CIDER can save output from both the circuit and device levels in the same way. Currently, separate files are used for the circuit and device data due to limitations of the implementation. As a result, there is no strong link between a state at the device-level and the corresponding state of the circuit. In the future, a more unified approach relying on a CAD database for output storage could be used instead of the existing ad hoc methods. For example, the OCT object-oriented database [HARR86] provides a general mechanism for data storage that leaves decisions about how that data is used up to a particular application. It was extended to support technology CAD data by the BPIF project [WONG91]. Similar extensions could be defined to store SPICE3 waveforms and to link those waveforms to snapshots of the device state at particular instants of time.

The main additional visualization problem posed by mixed-level simulation is the problem of correlating circuit behavior with the behaviors of multiple numerical devices. For example, such a capability would be useful in analyzing the transient behavior of charge transfer in a switched-capacitor circuit. Animation could be used to step through a standard Cartesian plot of the circuit waveforms, while at the same time multiple color contour plots for the numerical devices could be used to show the evolution of the internal device states. Unfortunately, it would probably be necessary to use a high-performance graphics workstation to adequately perform these operations. In the absence of such a sophisticated system, CIDER uses static methods to achieve

a similar function. In a special version of the code, the NUTMEG commands were extended to support a call to an external contour plot program. Figure 2.11 shows a screendump when two copies of NUTMEG are started from different terminal windows and a contour plot is generated by each copy. The plots show the log contours of the majority carrier concentrations inside the NMOS and PMOS devices of a CMOS inverter. The PMOS device is on top and the NMOS device is on the bottom. The snapshot was taken while the input voltage was being ramped from a low to a high state. The NMOS device turns on during this period, forming a conducting channel of electrons between the source in the upper left and the drain in the upper right. Asymmetry of the contours is due to the large potential on the drain terminal which depletes mobile carriers from the region around the drain. In the upper half of the figure, the PMOS device is being turned off. The holes that formed the channel are no longer confined by the electric field at the surface and spread out into the bulk beneath.

2.5 Summary

Mixed-level simulation combines algorithms from both circuit simulation and device simulation. Both CODECS and CIDER are mixed-level circuit and device simulators based on direct-method solution algorithms. The mixed-level simulation problem is a set of nonlinear ODEs and PDEs. Time and space discretization convert these equations to systems on nonlinear, algebraic equations which are solved using variations of the Newton-Raphson method. Finally, direct solution of large, sparse systems of equations is at the core of a mixed-level simulator.

In addition to the numerical methods that form the core of a simulator, the user interface is also an important part of a simulation program. A simulator should allow circuits and device structures to be specified in a flexible manner. CIDER combines the *de facto* standard circuit-input format of SPICE with a PISCES-like format for device-structure descriptions. Such an approach helps lower barriers to the adoption of mixed-level simulation in the IC design community by providing a familiar frontend interface.

At the backend, a mixed-level simulator should allow both circuit and device behaviors to be visualized. Here again, existing techniques taken from stand-alone

simulators can be applied to the mixed-level problem. Cartesion plots, contour plots, and three-dimensional perspective plots can all be used without the need for special-purpose graphics hardware.

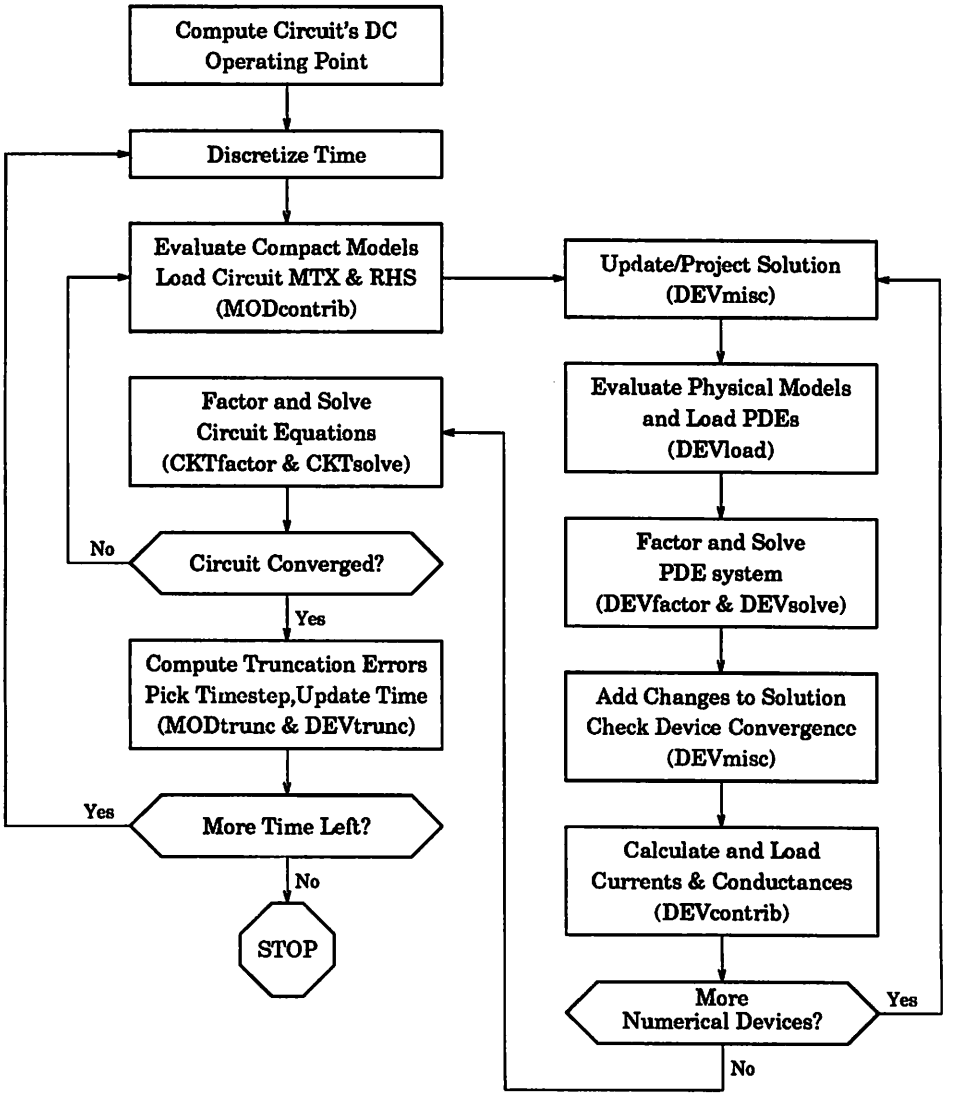


Figure 2.10: Flowchart for mixed-level transient simulation

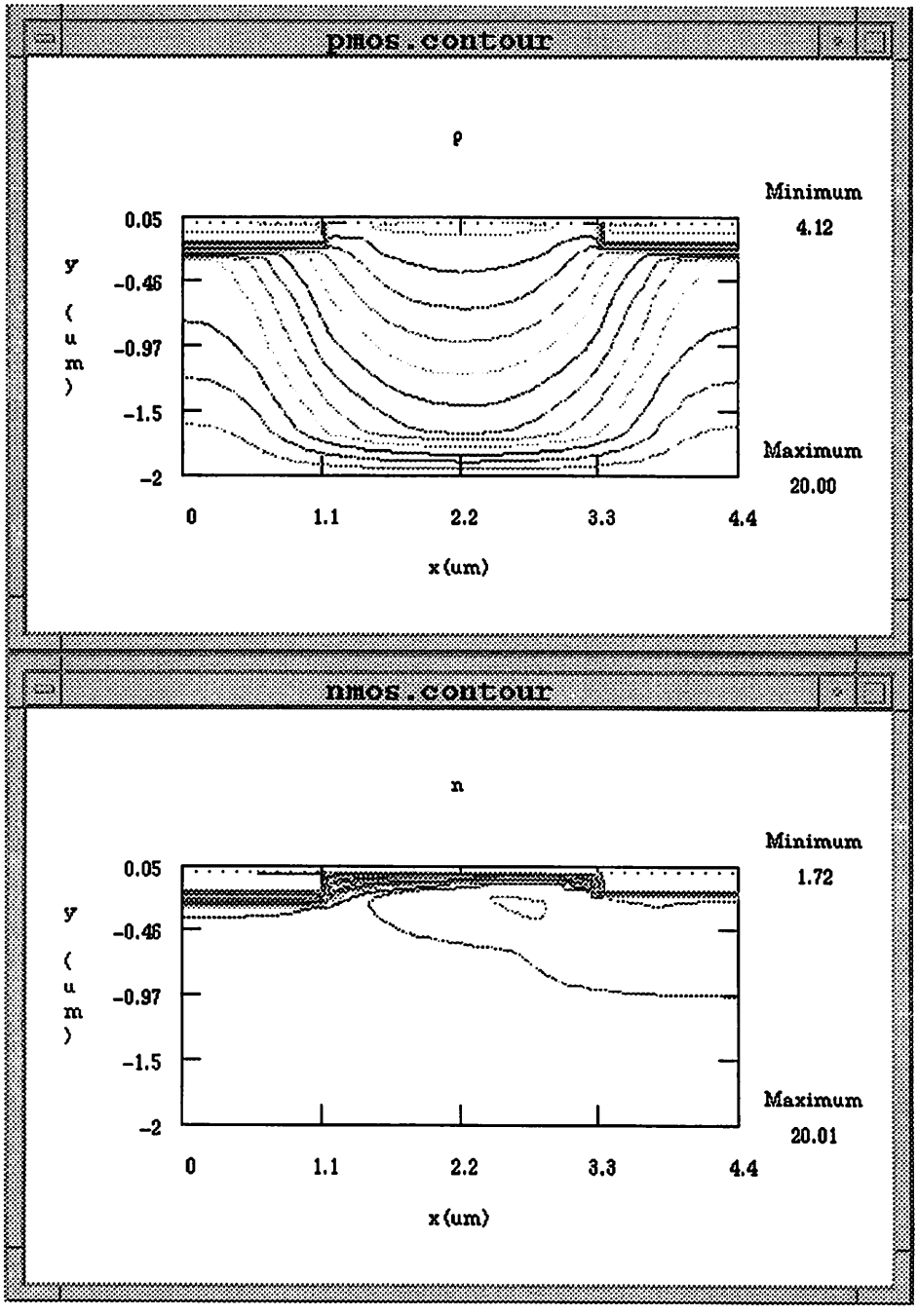


Figure 2.11: MOSFET internal states

Chapter 3

Performance Analysis of CIDER

3.1 Overview

Experience working with a mixed-level circuit and device simulator has demonstrated that the use of numerical device models in place of analytical device models can lengthen simulation execution times by two or more orders of magnitude [MAYA88]. This effect dramatically restricts the class of circuits that can be designed in a reasonable period of time using such a tool. However, the performance of engineering workstations employing Reduced Instruction Set Computer (RISC) architectures has increased rapidly over the past several years. This raises the possibility that mixed-level simulation may become viable on such a system.

In this chapter, CIDER is characterized in terms of its CPU usage, main memory usage, and I/O (long-term storage) requirements. Evidence is presented demonstrating that most mixed-level simulation time is spent executing device-level code. As a result, the performance of CIDER acting as a device simulator has been examined in detail. One- and two-dimensional numerical diodes with parameterized mesh specifications are used to investigate resource usage as a function of problem size. Simple models for CPU and memory usage are derived that can be used for usage prediction.

The performance of CIDER on a set of benchmark circuits has been measured on a number of RISC-architecture UNIX systems. The systems tested are described, and an examination of the differences among the results is provided. Finally, the performances obtained are evaluated in view of the requirements of effective IC design. Predictions are made for the system capabilities needed to design small analog and

Task	Benchmark Circuit					
	MECLGATE		DBRIDGE		NMOSINV	
	Time (S)	%	Time (S)	%	Time (S)	%
CKTload	286.5	96.3	1930.2	97.8	544.2	99.7
DEVload	145.3	48.8	921.7	46.7	67.4	12.3
DEVfactor	49.8	16.7	410.1	20.8	373.4	68.2
DEVsolve	20.5	6.9	283.9	14.4	32.3	5.9
DEVmisc	7.5	2.5	35.2	1.8	4.8	0.9
DEVcontrib	62.0	20.9	277.1	14.0	67.2	12.3
MODcontrib	1.4	0.5	2.0	0.1	0.1	0.0
CKTfactor	0.7	0.2	0.8	0.0	0.0	0.0
CKTsolve	0.3	0.1	0.3	0.0	0.0	0.0
CKTtrunc	7.7	2.6	38.3	1.9	0.8	0.1
DEVtrunc	7.4	2.5	37.3	1.9	0.8	0.1
MODtrunc	0.3	0.1	0.8	0.0	0.0	0.0
Other	2.2	0.7	4.0	0.2	1.0	0.2
Total	297.0	100.0	1973.7	100.0	546.1	100.0

Table 3.1: Execution profiles for several benchmarks on a DECstation 5000/125

digital standard cells.

3.2 Runtime Breakdown

In Table 3.1, execution profiles are shown for three of the benchmark circuits described in Section 3.4. The system used was a DECstation 5000/125. The first circuit, MECLGATE, is a bipolar ECL inverter containing 11 small-mesh, one-dimensional numerical bipolar transistors. The second circuit, DBRIDGE, is a diode bridge with 4 medium-sized-mesh, one-dimensional numerical diodes. The third circuit, NMOSINV, is a resistively loaded NMOS inverter employing a large-mesh, two-dimensional numerical MOSFET. All three simulations calculate a DC operating point followed by a transient analysis. As can be seen the percentages of time spent in the various sections change. However, although the circuit type, size and technology all change from example to example, an average of 99% of the total time is spent executing device-level code no matter which is considered. Device-level code consists of: evaluating physical models and loading the device-level equations (DEVload); direct solution of the resulting

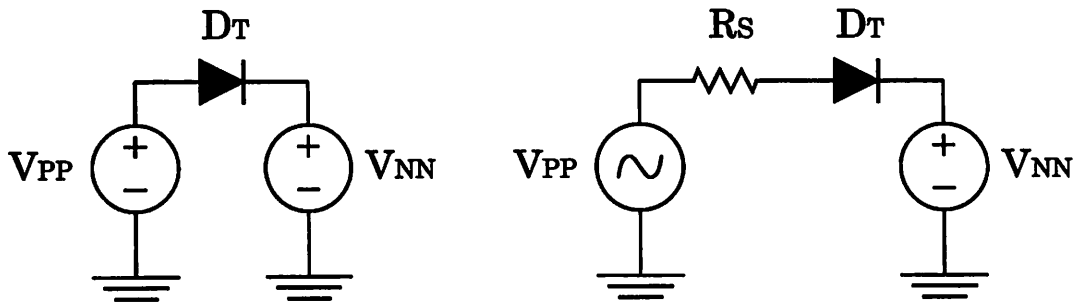
system of equations (DEVfactor and DEVsolve); miscellaneous overhead such as solution updating and convergence checking (DEVmisc), evaluating the equivalent current and conductance contributions to the circuit-level system of equations (DEVcontrib), and computing the numerical-device truncation errors (DEVtrunc). The remaining 1% of the time is divided between loading the compact model contributions (MODcontrib), factoring and solving the circuit-level matrix (CKTfactor and CKTsolve), computing compact-device truncation errors and the next timestep (MODtrunc), and other items such as reading and parsing the input file, setting up the circuit-level data structures, and writing the results to an output file. Note that only the MECLGATE circuit contains a significant number of compactly modeled and simple circuit elements, and thus is perhaps the best model for applications where only the critical devices are modeled numerically.

3.3 Device-Level Resource Usage

In the previous section, the importance of the device-level performance of CIDER is established. This section takes a closer look at this performance by examining data gathered from simulations of two very simple diode test circuits. The numerical diode models are based on a range of different meshes, varying from very coarse meshes to very fine meshes. The test circuits use numerical diodes because it is possible to produce qualitatively correct results using extremely coarse meshes. Both one- and two-dimensional diode models were tested. The schematic for the first circuit is shown in Figure 3.1(a). This circuit is used to measure the DC and AC small-signal analysis performance of CIDER. The second circuit, Figure 3.1(b), simulates the response of the diode to a sinusoidal input, and is used to measure the performance of transient analysis. In all cases, the simulations were performed on a DECsystem 5000/240 with 128Mb of real memory.

3.3.1 One-Dimensional Simulations

The input file for the one-dimensional diode DC/AC test circuit is shown in Figure 3.2. The corresponding input file for transient analysis is shown in Figure 3.3. The parameter `${XMESH_ELEMS}` is varied from 4 to 499 in 5 element increments. This



(a) DC/AC circuit (b) Transient circuit

Figure 3.1: Test circuits for device-level performance characterization

produces meshes with 5 to 500 nodes. The diode doping is uniform on each side of the PN junction with a concentration of $1.0 \times 10^{17}\text{cm}^{-3}$. Several physical models have also been enabled.

The DC analysis samples the diode current at 21 points between 0.0 V and 1.0 V. The AC analysis computes 51 frequency points logarithmically spaced between 100 KHz and 10 GHz. Although shown together in the input file, the two analyses were run independently in order to gather separate data sets. The transient analysis computes the response to a 1 MHz sinusoidal input voltage over one period. The number of timepoints computed depends very weakly on the mesh density with 194 points being about average.

The time per device-level iteration for DC and transient analysis can be broken down into a number of components. In both cases, the three most important components are the load, factor, and solve times. The remaining half dozen or so components take different amounts of time in the two analyses. Individually these remaining components do not contribute significantly to the total per-iteration time, but collectively their contribution is noticeable. In addition, there is one component, the matrix order time, that contributes only during the first iteration, but in that one iteration it dominates all others.

In Figure 3.4, the load, factor, and solve times per iteration are graphed against the total number of device-level equations. The number of equations, E, is equal to $3(n-2)$ where $(n-2)$ is the number of mesh nodes excluding the two contact

```
One-Dimensional Test Diode

VPP 1 0 0.6V AC 1V
VNN 2 0 0.0V
DT  1 2 M_PND AREA=1

.MODEL M_PND NUMD LEVEL=1
+ options defa=1u
+ x.mesh w=2.0 n=${XMESH_ELEMS}
+
+ domain num=1 mat=1
+ material num=1 silicon
+
+ doping unif n.type conc=1e17
+ doping unif p.type conc=2e17 x.h=1.0
+
+ models bgn srh auger conctau concmob fieldmob
+ output statistics

.OPTION ACCT BYPASS=1 TEMP=27
.DC VPP 0.0v 1.001v 0.05v
.AC DEC 10 100K 10G
.PRINT I(VNN)

.END
```

Figure 3.2: Input file – one-dimensional diode DC/AC simulation

nodes. Since the maximum number of nodes is 500, it follows that the maximum value of E is $3(500-2) = 1494$. Also graphed is the order time taken in the first iteration[1]. Note that it is larger than the other three combined. A graph of the time per transient iteration for the three main components would look similar to Figure 3.4. However, the load time is slightly higher because the additional time-dependent terms in the device-level PDEs must be calculated and loaded into the matrix and RHS.

As implemented the load time dominates the factorization and solve times even for the largest problem sizes. A result from work on circuit simulation predicts that this should only be true for relatively small sparse systems of equations. The time to load the system (which dominates initially) grows linearly with problem size, while the times to factor and solve are expected to grow superlinearly. However, in this

[1]Because the timer interval is comparable to the individual component times per iteration, they cannot be measured completely accurately. This problem is most noticeable for the order-time curve because the timing errors are not averaged out over many iterations.

```
One-Dimensional Test Diode

VPP 1 0 0.6V SIN 0.6V 0.1V 1MegHz
VNN 2 0 0.0V
RS  1 3 1.0
DT  3 2 M_PND AREA=1

.MODEL M_PND NUMD LEVEL=1
+ options defa=1u
+ x.mesh w=2.0 n=${XMESH_ELEMS}
+
+ domain num=1 mat=1
+ material num=1 silicon
+
+ doping unif n.type conc=1e17
+ doping unif p.type conc=2e17 x.h=1.0
+
+ models bgn srh auger conctau concmob fieldmob
+ output statistics

.OPTION ACCT BYPASS=1 TEMP=27 RELTOL=1E-6
.TRAN 0.01us 1.0us
.PRINT I(VNN)

.END
```

Figure 3.3: Input file – one-dimensional diode transient simulation

situation, the one-dimensional simulations give rise to block tridiagonal systems at the device-level. It can be shown that the work involved in decomposing such systems grows only linearly with problem size, so the factor and solve times never overtake the load time. On the other hand, the time to order the system of equations does grow superlinearly. This is a result of the general-purpose sparse matrix packages's inability to exploit the special structure of tridiagonal systems.

The AC analysis breakdown of the time per iteration is very different from that of DC and transient analyses. This is a result of the iterative method [LAUX85] that is used to solve the device-level equations in DSIM. Use of the iterative method allows the L/U factors calculated at the DC operating point to be reused until at some high frequency, the iteration fails to converge. A switch to direct methods must then be made [MAYA88]. The iterative method is dominated by the time to perform forward and back solves, whereas the direct method requires reloading and refactoring

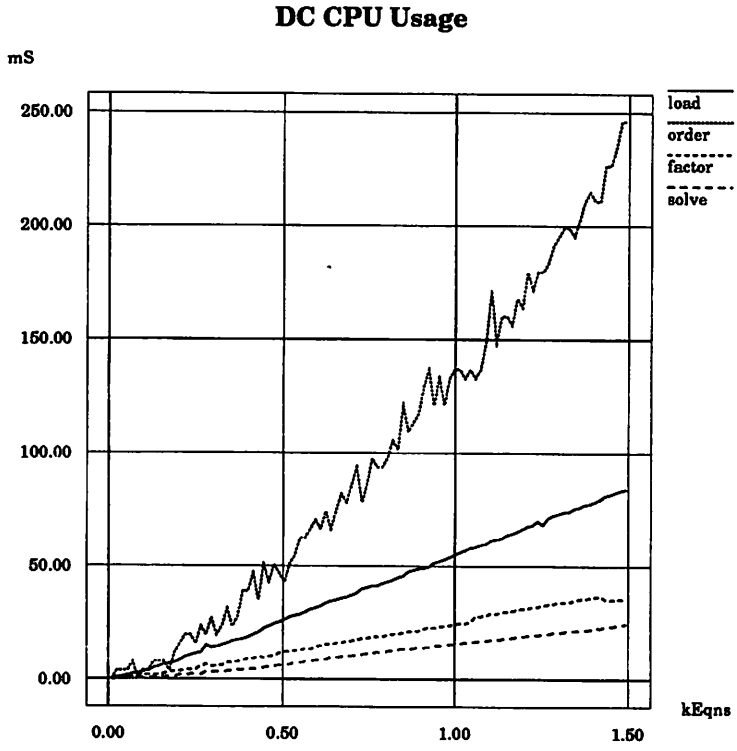


Figure 3.4: Major components of per iteration DC time for 1^D device

of the matrix at each frequency. Thus, the time per iteration for the iterative method is dominated by solve time, whereas the time for the direct method consists of the same components as DC analysis: load, factor and solve times. Because the method switchover does take place during the AC test, the time per iteration is an average of the iterative and direct method per-iteration times. Figure 3.5 shows a breakdown of the simulation time per iteration for the AC test[2]. Solve time dominates since most of the frequency points are calculated using the iterative method. The point of switchover was monitored and roughly 75% of the frequency points used the iterative method successfully. The remaining 25% of the points used the direct method. Thus, the times for loading and factoring are about one-fourth as long as used when the entire run is calculated using only the direct method. This has been verified by rerunning the AC test with the iterative method disabled, thereby forcing all points to use the

[2]The cause of the periodic increases and decreases in the measured times is unknown. Further investigation is needed to determine if this behavior is related to some real property of the machine, or if it is simply an artifact of the measurement process. For the moment, the accuracy of the timing is adequate for modeling purposes.

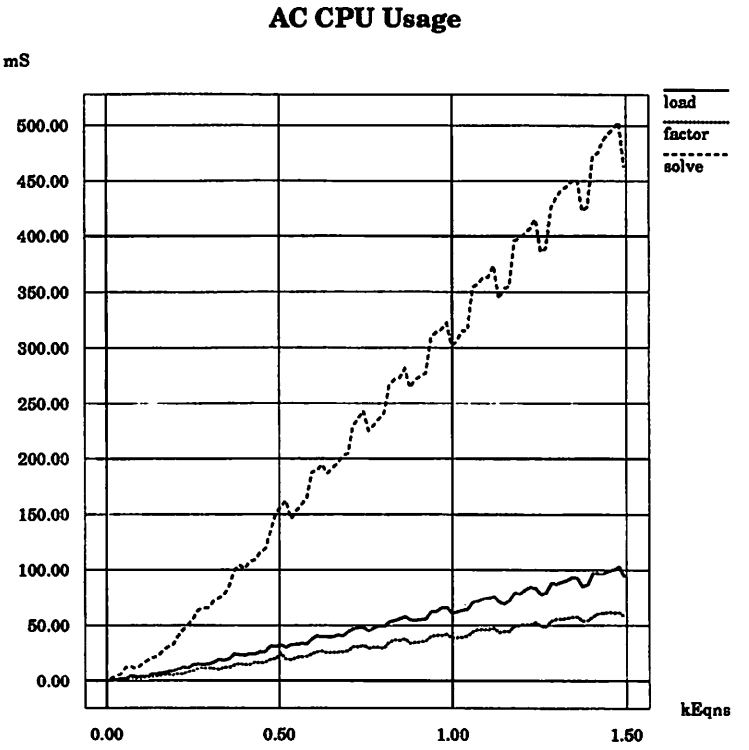


Figure 3.5: Major components of per iteration AC time for 1^D device

direct method.

For each of the three analyses, the total time per iteration was fit using linear regression to an equation of the form:

$$T = \alpha \cdot E^{\beta} \tag{3.1}$$

where α and β are empirically determined constants. Table 3.2 presents the results of this fitting procedure. The table shows that the growth is slightly superlinear. With care, these coefficients can be used to predict the amount of CPU time needed in

Analysis	α	β
DC	$47.5 \times 1.06^{\pm 1}$ μs	1.164 ± 0.009
AC	$22.1 \times 1.13^{\pm 1}$ μs	1.207 ± 0.018
TRAN	$46.9 \times 1.16^{\pm 1}$ μs	1.133 ± 0.012

Table 3.2: Average per iteration time as a function of 1^D problem size

other simulation circumstances. For a typical mesh of 100 nodes and 294 equations, the model gives 35.5 ms per DC iteration, 21.1 ms per AC iteration, and 29.4 ms per transient iteration.

In addition to consuming more CPU time, as problem size grows CIDER also requires more memory and long-term storage. In Figure 3.6 the total memory allocated by CIDER during the DC test is graphed. Comparison to the memory usage of the AC and transient analysis tests shows virtually identical results. The Y intercept is the

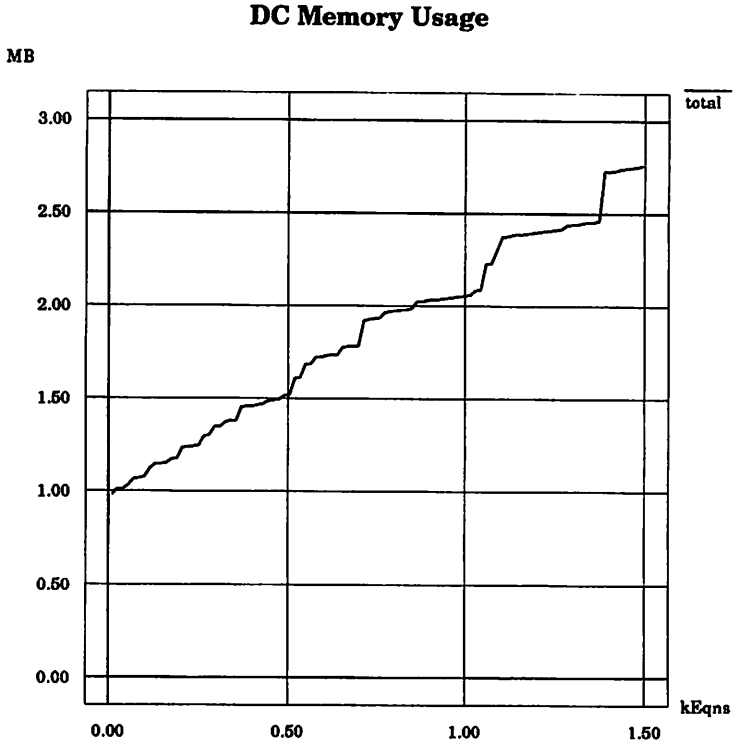


Figure 3.6: Total memory usage of the 1^D DC test

amount of memory used by CIDER for static data such as parameter tables. Jumps in the curve are caused by the memory manager, which requests memory from the system in large blocks that are multiples of 4KB. On the right side of the graph, the jumps are larger because larger blocks are being requested. The main components of the dynamically allocated memory are the memory to store the mesh, which scales with the number of nodes in the mesh, and the memory for the matrix and various RHS vectors, which scales more directly with the number of equations. The amount of disk storage needed scales almost directly with the number of nodes, with only a

slight overhead needed to initialize the output file. The constant of proportionality depends on the number of device internal variables being stored. By default, CIDER stores 9 double precision values for each node in the mesh, so this constant is normally 72 bytes/node. The minimum needed to be able to restore the device to a saved state is 4 double precision values: the node coordinate, the potential and the electron and hole concentrations. For the largest mesh considered in the 1^D tests (500 nodes), one internal state therefore requires a minimum of 16,000 bytes.

3.3.2 Two-Dimensional Simulations

The input file for the two-dimensional DC/AC test, shown in Figure 3.7, is very similar to the file used for the one-dimensional tests. Only the model definition has changed. For the transient analysis file (not shown) this is also the only required change. Two parameters, `${XMESH_ELEMS}` and `${YMESH_ELEMS}`, are now varied to change the size of the simulation mesh. The parameter `${XMESH_ELEMS}` is varied from 4 to 49 in 5 element increments and `${YMESH_ELEMS}` varies from 4 to 24 in 5 element increments. This results in a minimum mesh size of 5×5 nodes and a maximum of 50×25 nodes. The number of equations, E, is equal to $3(nx - 2)(ny)$, where nx and ny are the number of mesh lines in the X and Y dimensions, respectively. Two of the X mesh lines do not contribute equations because they belong to the contacts. Because it is not obvious where the contacts go in the 2^D case, `electrode` statements are needed to define their locations. The doping profile in the X dimension is the same in both the 1^D and 2^D cases. In the Y dimension, the doping is constant for a given value of X. As a result, the one- and two-dimensional files model the same diode. For nx fixed, the calculated diode current is essentially identical for all values of ny and for the 1^D case when $n = nx$. For different values of nx, the current calculated varies somewhat since the solution accuracy does depend on the mesh density in the X dimension.

The circuit analyses performed were identical to those performed in the 1^D tests. However, because different numerical tolerances are used for 2^D simulation, the transient simulation only calculates an average of 60 timepoints, much fewer than the 194 points for the 1^D case. While in a real application this discrepancy would need to be addressed, for the purpose of measuring the time per iteration 60 points is more than adequate.

```
Two-Dimensional Test Diode

VPP 1 0 0.6V AC 1V
VNN 2 0 0.0V
DT  1 2 M_PND AREA=1

.MODEL M_PND NUMD LEVEL=2
+ x.mesh w=2.0 n=${XMESH_ELEMS}
+ y.mesh w=1.0 n=${YMESH_ELEMS}
+
+ domain num=1 mat=1
+ material num=1 silicon
+
+ electrode num=1 x.l=0.0 x.h=0.0 y.l=0.0 y.h=1.0
+ electrode num=2 x.l=2.0 x.h=2.0 y.l=0.0 y.h=1.0
+
+ doping unif n.type conc=1e17
+ doping unif p.type conc=2e17 x.h=1.0
+
+ models bgn srh auger conctau concmob fieldmob
+ output statistics

.OPTION ACCT BYPASS=1 TEMP=27
.DC VPP 0.0v 1.001v 0.05v
.AC DEC 10 100K 10G
.PRINT I(VNN)

.END
```

Figure 3.7: Input file – two-dimensional diode DC/AC simulation

Figure 3.8 shows a breakdown of the DC simulation time into the same components used in the one-dimensional case: load, order, factor, and solve times. It is necessary to present the data using log-log scales because the order, factor and solve times are all growing superlinearly. Because of this the factor time quickly dominates the per-iteration times in the 2^D case. For large problems, the factor time takes between 10 and 100 seconds per iteration. (The ordering time is still only incurred once per simulation. However, for the largest mesh, the order time is significant in absolute terms: over seven and a half minutes.) Note that the load time remains larger than solve time even for the largest problem. It should also be noted that the time per iteration does not grow uniformly with problem size. Nonuniform growth is caused by variation in the percentage of fillin created by the L/U decomposition process. This

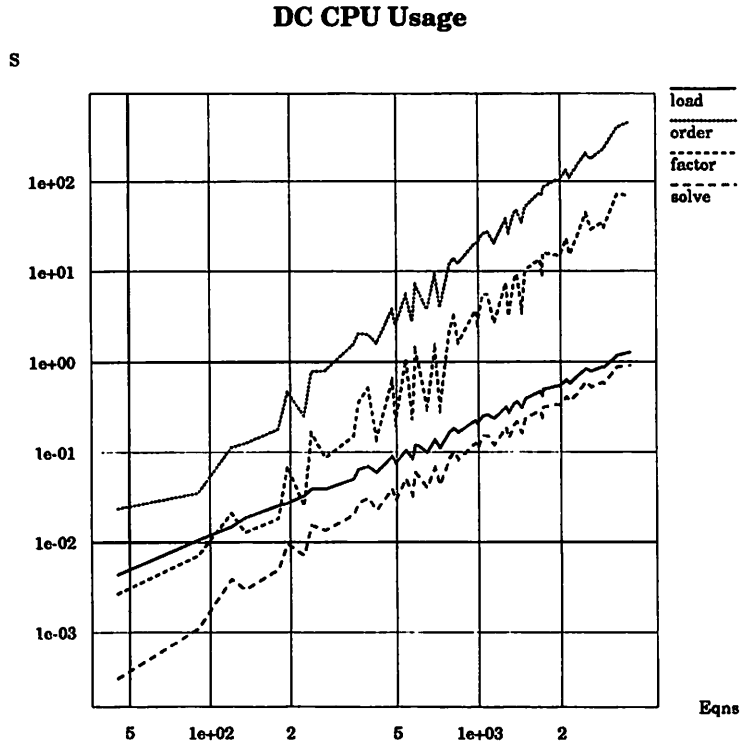


Figure 3.8: Major components of per iteration DC time for 2^D device

affects even the load phase, because part of the load time is used clearing all the matrix entries at the beginning of the load process. Some of the variation is essentially random, but the rest is correlated with the shape of the simulation mesh. The test set contains meshes that are relatively square as well as others that are long and narrow. The long, narrow meshes are essentially one-dimensional in nature. Thus, for $nx \gg ny$, the time grows almost linearly as nx is increased. For $nx \approx ny$, the time grows superlinearly as *both* are increased. So even if two meshes give rise to almost equal numbers of equations, they may not take equal time to factor if one is longer and more narrow than the other. For example, the 50×5 mesh problem takes less than half the time to factor as does the 25×10 mesh, even though there is only a 4% difference in the number of equations in the two cases. Additional experiments factoring test matrices that were derived from a similar mesh-based problem confirmed the above interpretation of the data.

The results for AC analysis are also affected by the increased importance of the factorization process. Figure 3.9 shows the load, factor and solve times per

Analysis	α	β
DC	$1.09 \times 2.30^{\pm 1}$ μs	2.181 ± 0.124
AC	$1.50 \times 1.92^{\pm 1}$ μs	2.007 ± 0.097
TRAN	$0.87 \times 2.34^{\pm 1}$ μs	2.201 ± 0.126

Table 3.3: Average per iteration time as a function of 2^D problem size

iteration for the 2^D AC test. Even though iterative methods are still used 75% of the

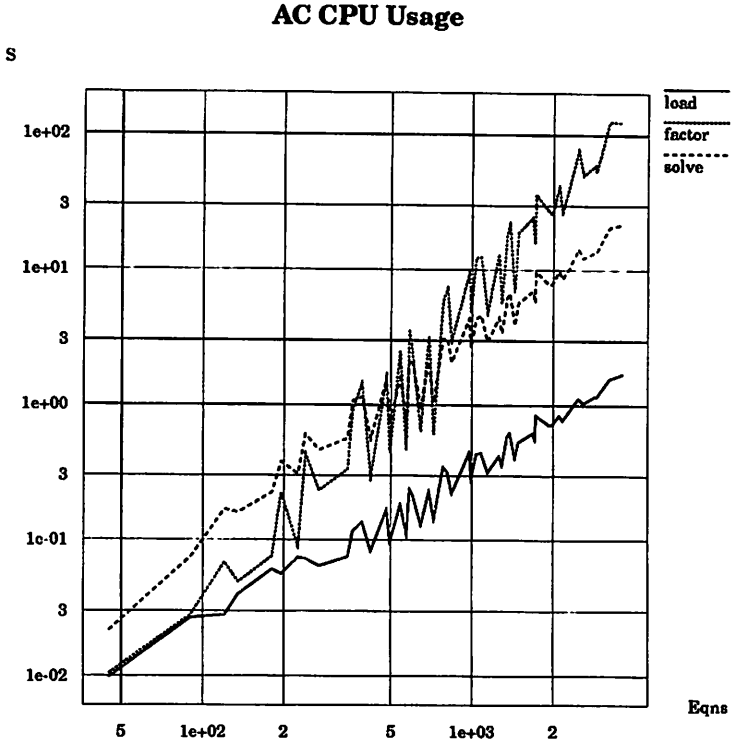


Figure 3.9: Major components of per iteration AC time for 2^D device

time, the factor time becomes the dominant component for large E because it grows so rapidly.

In Table 3.3 the time per iteration for the three types of analyses is fit to Equation 3.1. Superlinear growth is reflected by the fact that the β values are all much greater than one. In fact, these values are all significantly worse than the results presented in [PINT90], suggesting that development of a better sparse matrix package might be in order. In addition, the parameter spreads are larger than in

the 1^D case because of the irregularity in the underlying data. Thus, these formulas are only good for order-of-magnitude calculations of the CPU time. If more accuracy is required, a formula that accounts for the shape of the simulation mesh could be developed. For a typical 20 × 20 mesh with 400 nodes and 1080 equations, this model gives 4.5 s per DC iteration, 1.8 s per AC iteration, and 4.1 s per transient iteration.

The memory usage in the 2^D case is dominated by the memory used for the sparse matrix L/U factors. Figure 3.10 shows the total memory allocated by CIDER along with just the memory used to store the non-zero entries in the factored sparse matrix. Total memory usage is clearly being driven by the L/U factor storage. A fit

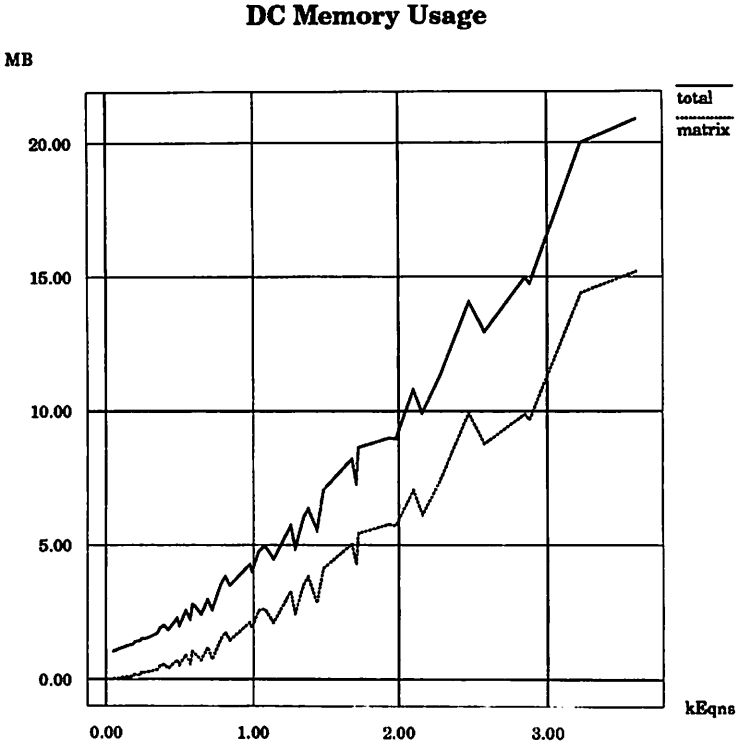


Figure 3.10: Total memory usage of the 2^D DC test

of the dynamic memory used[3] by both the 1^D and 2^D DC tests to Equation 3.1 gives the results in Table 3.4. In the 1^D case, the model predicts sublinear memory growth which is caused by the large jumps in the data. Hand calculations easily show that the memory growth is at least linear for this problem. Thus, extrapolation of this model

[3]The static memory usage was determined by running CIDER with no circuit loaded, and then it was subtracted from the total memory in Figures 3.6 and 3.10 to obtain the dynamic memory usage.

	α	β
OneD	$3.76 \times 1.16^{\pm 1}$ KB	0.830 ± 0.024
TwoD	$0.54 \times 1.30^{\pm 1}$ KB	1.265 ± 0.040

Table 3.4: Average memory used as a function problem size

beyond the range measured is dangerous.

Disk storage per mesh node is larger in the two-dimensional case. First, a pair of double-precision values is now required to save the the coordinates of each mesh node. Second, four of the remaining variables (the electric field and the three current densities) are vector quantities that also use two double-precision values. So the default disk usage is 112 bytes/node. Disk usage can be minimized by excluding nonessential quantities but 40 bytes/node are still required. This is larger than the minimum in the 1^D case because the Y coordinates must be saved.

3.4 Benchmark Circuit Performance

As noted in Chapter 2, CIDER evolved from CODECS. CODECS performance results are given in [MAYA88] for two DEC VAX architecture machines: the VAX 8650 and VAX 8800. In this section, new performance results for a number of RISC-based UNIX systems are provided. These performance results are for CIDER, not the original CODECS code. Due to various incompatibilities between CIDER and CODECS, direct comparison of these new measurements to those reported in [MAYA88] is discouraged. The goal here is not so much to help decide which of the particular machines considered is best for this problem, but rather to help in understanding how much performance is available now and how much is likely to be needed in the future.

Table 3.5 summarizes the system configurations used in this performance test. Each system uses floating-point arithmetic hardware based on the IEEE floating-point standard. All except for the iPSC/860 are available locally at UC Berkeley. The iPSC/860 system tested is installed in the Applied Electronics Laboratory at Stanford. Only 1 of its 32 compute nodes was used for this test. The DECstation 5000/125 is similar in design to the DECsystem 5000/240 used to run the device-level benchmarks except that the clock rate is 60% higher. This has been verified by running the one-

Maker	DEC	HP	Intel	IBM	SUN
Model	5000/125	9000/720	iPSC/860	RS6000/530	4/370
CPU	MIPS R3000	PA/RISC1.1	i860 XR	RS/6000	SPARC
Clock Rate	25 MHz	50 MHz	40 Mhz	25MHz	25MHz
Memory	32MB	16MB	16MB	32MB	56MB
OS	Ultrix 4.2a	HPUX 8.07	NX/2 3.3.2	AIX 3.2	SunOS 4.1.1
C Compiler	MIPS 2.1	HP C 8.71	PGC 2.0a	XLC 1.02	SUN 4.1.1
Identifier	DEC	HPUX	IPSC	RS6K	SUN4

Table 3.5: RISC machine configurations used in test

dimensional DC benchmark test on both and comparing execution times.

It has been noted in [PATT90] that RISC architectures tend to be remarkably similar. In that book, a comparative study including three of the five architectures used here (the R3000, i860 and SPARC) is presented. The primary difference to note here is the clock speed variations of the CPUs, which have a direct impact on the performance. The minimum memory size of 16MB is sufficient to hold the benchmark data sets in real memory, eliminating the performance of virtual memory management as a concern. However, the size of data caches is important in determining overall performance, since data not held in fast cache memory must be accessed from the slower main memory. In each case, the standard C compiler shipped with the operating system is used to compile the source code using an optimization level of *-O2*. In the following tables, the Identifier field of Table 3.5 is used to differentiate between the various systems.

A set of 12 benchmark circuits has been used to exercise CIDER on the systems tested. Input listings written in the CIDER circuit description format are provided in Appendix B. The benchmark circuits include the 9 circuits used to test CODECS in [MAYA88] as well as 3 new circuits that round out the benchmark set. All five of the numerical model types supported by CIDER are represented. Table 3.6 briefly summarizes the circuits used. Of the circuits that employ one-dimensional numerical models, several have multiple numerical devices, whereas a single two-dimensional numerical device is used in the remaining circuits. This limits the execution time of the benchmark set to a reasonable level on all the machines. Larger, more time-consuming circuits are considered in Chapter 5. Even for these relatively small circuits, the number of circuit-level equations is always negligible compared to the number of

Circuit	# Ckt Elts	# Num Devs (Type)	# Ckt Eqns	# Dev Eqns	MFLOP
ASTABLE	8	2 (1D BJT)	9	354	379.5
CHARGE	7	1 (2D MOS)	13	724	1650.0
COLPOSC	8	1 (1D BJT)	9	177	462.7
DBRIDGE	3	4 (1D DIO)	7	2388	1714.8
INVCHAIN	10	4 (1D BJT)	13	708	127.2
MECLGATE	24	11 (1D BJT)	29	1947	371.5
NMOSINV	6	1 (2D MOS)	10	921	380.5
PASS	7	1 (2D MOS)	11	921	198.5
PULLUP	7	1 (2D BJT)	13	1081	3405.4
RECOVERY	4	1 (2D DIO)	8	1269	3946.9
RTLINV	4	1 (1D BJT)	7	177	22.7
VCO	10	6 (1D BJT)	9	1062	731.3

Table 3.6: Serial benchmark circuit characteristics

device-level equations. The floating-point operation counts[4] , measured in millions (MFLOP), were obtained using PIXIE on the DEC-MIPS machine [PIX89]. Although actual operation counts may be different on the other machines, the DEC-MIPS results are used as the standard for comparison purposes.

Despite efforts to ensure that the same computation is performed on each machine, differences still arise in the results. These may be due to variations in the instruction sequences generated by the compiler and the floating-point hardware implementations. In addition, for a large code such as CIDER, it is difficult to rule out undiscovered bugs as a potential cause of these variations. In any event, the benchmark outputs are not identical on each machine. In Table 3.7, several measures of the amount of computation performed are given. In order, the numbers presented are: the number of transient iterations, the total number of timepoints, and the number of timepoints accepted/rejected. An entry of — indicates that the number is identical to the results for the DEC-MIPS architecture. This allows differing results to stand out more clearly. It should be pointed out that the timepoint numbers for the PULLUP and RECOVERY circuit are *identical*, simply the result of a remarkable coincidence. Overall, nearly identical results are obtained on all the benchmarks except for the VCO circuit. This circuit, a voltage-controlled relaxation oscillator, is apparently very

[4]Since double-precision arithmetic is used throughout CIDER these are double-precision floating-point operation counts.

Circuit	System				
	DEC	HPUX	IPSC	RS6K	SUN4
ASTABLE	7157	—	—	—	—
	1766	—	—	—	—
	1457/309	—	—	—	—
CHARGE	1857	1826	—	—	—
	456	454	—	—	—
	382/74	383/71	—	—	—
COLPOSC	13775	—	—	—	—
	3617	—	—	—	—
	3044/573	—	—	—	—
DBRIDGE	5144	—	—	—	—
	1714	—	—	—	—
	1627/87	—	—	—	—
INVCHAIN	1378	—	—	—	—
	354	—	—	—	—
	313/41	—	—	—	—
MECLGATE	1587	1641	—	—	—
	400	411	—	—	—
	350/50	353/58	—	—	—
NMOSINV	289	265	—	—	—
	86	81	—	—	—
	81/5	78/3	—	—	—
PASS	136	135	—	—	—
	34	—	—	—	—
	31/3	—	—	—	—
PULLUP	560	—	—	—	—
	151	—	—	—	—
	128/23	—	—	—	—
RECOVERY	489	—	—	—	—
	151	—	—	—	—
	128/23	—	—	—	—
RTLINV	711	—	—	—	—
	199	—	—	—	—
	175/24	—	—	—	—
VCO	5408	5346	5427	5311	5361
	1239	1224	1242	1222	1232
	1036/203	1027/197	1040/202	1025/197	1032/200

Table 3.7: Iteration and timepoint counts on the various machines. Entries are: total transient iterations, total timepoints computed, and number of timepoints accepted/rejected. An entry of — means that the number is equal to that found in the DEC column.

sensitive to variations in the machine architecture. This may be due to the particular behavior of this circuit, which is characterized by fast switching transients followed by long periods of slow decay. Positive feedback during the switching would be particularly capable of amplifying small differences in the machine precision, resulting in differences in the timepoint counts. However, to the naked eye, the circuit waveforms are indistinguishable. For the other circuits, only the HPUX machine gives results that are different from the others. The reason for this is unknown.

Timing the execution of a program is not a simple task. For this test, execution times are measured in terms of the amount of time spent executing user-level code. The time spent executing system calls is not included, but typically represents a small fraction of the total CPU time. Wall clock time is generally an ineffective measure of performance since UNIX is a time-shared operating system. This means that multiple programs may be competing for the CPU at the same time, thereby lengthening wall clock times. However, on the IPSC system, the only time measurement available is the wall clock time. Fortunately, the NX/2 node operating system does not allow time-sharing of compute nodes so that using wall clock time is reasonable. Even after taking these steps to obtain reproducible timing results, the execution times still vary from one run to another for a given benchmark. This is caused by other programs interfering with the execution of CIDER. For example, on the time-shared machines, all running programs must share the instruction and data caches. If other programs use the caches heavily, CIDER takes extra time refilling the cache with its own data. In order to reduce the importance of such problems, each benchmark was timed over 5 runs and the results were averaged. The spread in the total execution times is typically 1-2% of the mean execution time. However, it is larger for the short running benchmarks such as the RTLINV circuit. In the worst case, on the HP 9000/720 the spread is 11% of the mean.

Table 3.8 gives timing results for each benchmark, as well as for the suite taken as a whole. The numbers are: the total analysis time, the time for just the transient analysis, and the time per transient iteration. Total execution time depends strongly on the number and size of numerical devices in the circuit as well as the total number of iterations performed. A simple model for this is given by:

$$T = N \cdot D \cdot T_{iter} \tag{3.2}$$

Circuit	System				
	DEC	HPUX	IPSC	RS6K	SUN4
ASTABLE	312	77	255	128	480
	310	76	253	127	477
	.043	.011	.035	.018	.067
CHARGE	2295	490	1026	561	2358
	2257	481	1001	550	2316
	1.215	.263	.539	.296	1.247
COLPOSC	366	95	313	160	562
	365	95	311	160	561
	.026	.007	.023	.012	.041
DBRIDGE	1991	474	1149	612	2313
	1976	471	1135	606	2296
	.384	.092	.221	.118	.446
INVCHAIN	102	30	85	42	157
	95	28	77	39	147
	.069	.020	.056	.028	.107
MECLGATE	297	91	248	122	461
	280	85	227	114	434
	.176	.052	.143	.072	.273
NMOSINV	549	112	233	136	559
	471	95	189	115	475
	1.630	.358	.654	.398	1.644
PASS	288	63	128	73	294
	228	49	92	56	232
	1.676	.363	.676	.412	1.706
PULLUP	5945	1412	1857	1273	5743
	5479	1296	1688	1165	5292
	9.784	2.314	3.014	2.080	9.450
RECOVERY	7015	1375	2150	1455	6767
	6467	1260	1952	1327	6241
	13.225	2.577	3.992	2.714	12.763
RTLINV	18	5	16	8	27
	16	4	14	7	25
	.023	.006	.020	.010	.035
VCO	593	176	484	241	908
	588	175	477	238	900
	.109	.033	.088	.045	.168
TOTAL	19771	4400	7944	4811	20629
	18532	4115	7416	4504	19396
	.481	.107	.193	.117	.505

Table 3.8: Benchmark execution times on various machines in seconds. Entries are: total analysis time, transient analysis time and time per transient iteration.

Circuit	# Dev.	Active	System				
			DEC	HPUX	IPSC	RS6K	SUN4
ASTABLE	2	1.5	1.7	1.6	1.5	1.5	1.6
COLPOSC	1	1.0	1.0	1.0	1.0	1.0	1.0
INVCHAIN	4	2.4	2.7	2.9	2.4	2.3	2.6
MECLGATE	11	6.2	6.8	7.4	6.2	6.0	6.7
RTLINV	1	0.9	0.9	0.9	0.9	0.8	0.9
VCO	6	3.9	4.2	4.7	3.8	3.8	4.1

Table 3.9: Relative time per iteration per device for 1^D bipolar circuits

where T is the total analysis time, N is the number of iterations performed, D is the number of numerical devices, and T_{iter} is the time per iteration per device. The difference between the total analysis time and the transient analysis time is primarily contributed by the DC operating point analysis. For some of the circuits this operating point time is negligible, but for others, especially the ones using 2^D models, the operating point time is a significant portion of the overall execution time.

For a given type of numerical model, one might expect the time per iteration per device to be independent of the number of devices as suggested by Equation 3.2. In Table 3.9 the relative time per iteration per device is shown for the 6 one-dimensional bipolar circuits. The time per iteration for the COLPOSC circuit is used as the reference. Also included are the number of numerical devices and the average number of active numerical devices as described below. The time per iteration does not scale linearly with the number of devices. The primary cause of this behavior is the bypass algorithm of CODECS [MAYA88]. Bypassing reuses old values of the currents and conductances for *latent* devices, devices whose terminal voltages have not changed much from the previous iteration. This allows the expensive operations on latent numerical devices to be skipped, thereby producing significant computational savings. As can be seen the time per iteration more closely follows the average number of active (non-latent) devices rather than the total number. Equation 3.2 can be modified to account for this effect as follows:

$$T = N \cdot \overline{D} \cdot T^{iter} \tag{3.3}$$

where $\overline{D}$ is the average number of active devices. In this model, the time per iteration per device is constant; the number of active devices is what changes from circuit to

circuit. Another simple extension of Equation 3.2 can be defined for the case when different types of numerical devices are mixed in a circuit (e.g. in a BiCMOS circuit) where the time per iteration per numerical device is not constant:

$$T = \sum_{k=1}^{K} N_k \cdot \overline{D}_k \cdot T_k^{iter} \tag{3.4}$$

where K is the number of different models used for numerical devices.

Considering that the benchmarks were chosen to complete in a reasonable amount of time it is not surprising that the longest run is 7015 seconds, a little under two hours, for the RECOVERY circuit running on the DEC-MIPS machine. However, this leads to an important observation: given realistic time constraints, the amount of progress made at the circuit level (as measured by the number of timepoints computed) is limited by the size of the mesh used for the numerical devices. This limits the complexity of the circuits that can be simulated effectively using CIDER.

Example: Suppose the 1^D bipolar transistor in the COLPOSC circuit were replaced by the 2^D bipolar used in the PULLUP circuit. Assume also that using a 2^D model does not change the circuit behavior significantly, so that the iteration count remains approximately the same. In the same amount of time (365 seconds on the DEC-MIPS machine), the 2^D simulation would complete only about 37 iterations, 0.27% of the total of 13775 needed. The entire run would take over a day and a half to complete. While a 1^D model might be suitable to use when designing this circuit, the 2^D model is limited to use during final verification of the design. ■

Another way to measure performance is by the amount of computation done per unit time. Table 3.10 gives MFLOP/S ratings of the computational speed of each machine. The MFLOP/S rating is computed by taking the total analysis time and dividing into the FLOP counts given in Table 3.6. At the bottom of the table the minimum, mean and maximum MFLOP/S ratings are shown. These are compared to the LINPACK benchmark and peak MFLOP/S ratings reported in [DONG93]. Because the DS 5000/125 does not appear in that report, the LINPACK rating for the DS 5000/200, which has an identical clock speed and CPU, was substituted. Of the 5 machines tested, the RS/6000 shows the most consistent performance across all the benchmarks. On average, none of the machines achieves the performance reported for the LINPACK benchmark. The ratings are 4 to 6 times lower for CIDER. This

Circuit	System				
	DEC	HPUX	IPSC	RS6K	SUN4
ASTABLE	1.22	4.93	1.49	2.96	.79
CHARGE	.72	3.37	1.61	2.94	.70
COLPOSC	1.26	4.87	1.48	2.89	.82
DBRIDGE	.86	3.62	1.49	2.80	.74
INVCHAIN	1.25	4.24	1.50	3.03	.81
MECLGATE	1.25	4.08	1.50	3.05	.81
NMOSINV	.69	3.40	1.63	2.80	.68
PASS	.69	3.15	1.55	2.72	.68
PULLUP	.57	2.41	1.83	2.68	.59
RECOVERY	.56	2.87	1.84	2.71	.58
RTLINV	1.26	4.54	1.42	2.84	.84
VCO	1.24	4.16	1.51	3.03	.81
Minimum	.56	2.41	1.48	2.68	.58
Mean	.68	3.04	1.69	2.78	.65
Maximum	1.26	4.93	1.84	3.05	.84
LINPACK	3.7	18	9.7	15	2.7
Peak	—	50	40	50	—

Table 3.10: Benchmark MFLOP/S ratings on various machines

is not surprising since the LINPACK program has more regular data access patterns that are typical of dense matrix calculations. This results in a high ratio of floating-point operations to total instructions executed. In contrast, CIDER is dominated by the loading and solving of sparse matrices, which involves irregular data access and pointer manipulation. PIXIE traces of CIDER on the benchmark set show that only 10 to 12% of the instructions executed by CIDER are floating-point operations.

The DEC-MIPS machine results show most clearly that the performance is most correlated with the size of the numerical model used, and not the number of devices or number of iterations. All the 1^D bipolar circuits achieve virtually identical performance on the DS 5000/125. As the numerical model increases in size, the performance usually degrades, especially on those machines that rely most heavily on fast caches to achieve high performance (DEC-MIPS and HPUX). However, this trend is reversed on the IPSC where the MFLOP/S rating actually improves as model size increases. This can be partially explained by noting that the IPSC timings include I/O overhead that is excluded on the other systems. I/O on the IPSC is particularly slow because it involves communication between the compute node and the IPSC host computer over a slow Ethernet link. As the model size increases, I/O becomes less important compared to the model evaluation time, so performance improves.

3.5 Performance Requirements

Every computing environment places limits on the kinds of problems that can be solved in it. Some of these restrictions are due to the computing hardware; CPU speed, available memory, and disk space are a few. Other restrictions are imposed by the people using the system. In the case of mixed-level simulation, the most important of these is the amount of time an IC designer is willing or able to wait for an answer from the simulator. However, equitable sharing of computing resources amongst all users may also be important. In particular, one user is not generally allowed to use more than her fair share of CPU time if it disrupts the work of others. The central question then for any design tool is whether or not it can provide a useful service to the designer while adhering to the restrictions of his computing environment.

In this section, an attempt is made to answer this question by employing the resource usage and performance data gathered in the previous two sections. Estimates

Item	Estimate	Example
Timepoints (TP)	100 - 5000	1000
Iter/Point (N/TP)	3 - 6	5
Total Iterations (N)	300 - 30000	5000
Devices (D)	1 - 50	10
Device Size (E)	150 - 5000	2000
Total Equations (DE)	150 - 250K	20K
Total FLOPs	17M - 196T	87G

Table 3.11: Estimated size of mixed-level simulation problem

are made of the computation performed when simulating larger circuits than the ones in the serial benchmark set. Both the consequences of using bigger meshes and more numerical devices are considered. Several estimates are made so that reasonable upper and lower bounds on the amount of computation can be determined.

These estimates are used in two ways. First, to determine what kind of environment is needed for mixed-level simulation given unlimited computing resources but limited time for completion of a design. For the largest simulations, it is projected that mixed-level simulation falls into the same class of problems as the grand challenges of science and technology [LEWI92], which require supercomputer-or-better levels of performance to solve. The second use of the resource estimates is to assess which requirements are likely to be the most limiting in a variety of computing environments.

3.5.1 Estimated Problem Specifications

In Table 3.11, the key parameters needed to estimate the total number of floating-point operations performed are given. The number of timepoints and iterations per timepoint needed are estimated using the data in Table 3.7. The upper bounds have been deliberately increased to account for the possibility of relatively long simulation intervals. The example case is intended to result in a reasonable estimate for a medium to large size problem capable of being run on present-day supercomputers. The number of devices was estimated by examining circuit schematics from a variety of published sources and counting the devices by hand. The maximum of 50 devices is sufficient to design a wide range of circuits that typically appear as standard cells in IC designs: digital logic gates, flip-flops, latches, comparators,

opamps, mixers, multipliers, and input and output buffers. In addition, multistage ring oscillators from several technologies can be also simulated. This assumes that all devices need to be modeled with the accuracy afforded by numerical simulation. If the devices critical to the overall performance are few in number and only these devices are numerically modeled, much larger circuits can be simulated. The number of equations is related to the density of the simulation mesh, and is intended to allow reasonable quantitative accuracy to be achieved. However, it is recognized that the meshes used are typically less accurate than those used for pure device simulation where larger mesh sizes can be tolerated. The total number of flops performed is estimated using a multistep procedure. The time to solve a problem with E equations is estimated using the transient analysis iteration-time parameters in Table 3.3. This is then multiplied by the MFLOP/S rating for a DECsystem 5000/240, which is estimated as the average MFLOP/S rating of the DS5000/125 times the ratio of the respective clock speeds (40MHz/25MHz). The result is a rough guess for the number of floating point operations per device per iteration that is multiplied by the number of devices and iterations to obtain the flops estimate. From this calculation, the number of flops ranges over 7 orders of magnitude from the smallest to the largest problem. Using the just the FLOP counts in the table, it seems clear that design using small circuit problems is certainly feasible on present-day workstations. For example, the RTLINV benchmark fits the low-end circuit profile fairly well, and takes only 18 seconds to simulate on an approximately 1 MFLOP/S machine (the DS5000/125). However, the largest problem is well beyond the capabilities of even the fastest computers built today, and is likely to remain so for some time to come. How far beyond is explored in the next section.

3.5.2 Estimated Resource Usage

Table 3.11 estimates the size of the mixed-level simulation problems likely to be encountered during IC design. These numbers can be converted into predictions for the computational resources needed to solve such problems. The performance needed depends on two factors: the amount of computation performed and the amount of time allocated for the task. The time available is subject to the expectations of the IC designer. Generally, small problems are expected to run quickly, whereas some waiting will be anticipated for larger problems. In other words, the time allocated should be

Interval	Length	Seconds
Small Task	5 Mins.	300
Coffee Break	15 Mins.	900
Lunch Break	1 Hr.	3,600
Overnight	12 Hrs.	43,200
Weekend	2 Days	172,800
Vacation	1 Week	1,209,600

Table 3.12: Estimated time for designer idle periods

	Estimate		
Item	Small	Large	Example
Time Limit (S)	500	10000	2500
Performance (FLOP/S)	34K	20G	35M
Dynamic Memory (Bytes)	250K	1.3G	81M
Disk Space (Bytes)	18K	38G	7.7M
I/O Bandwidth (Bytes/S)	3.6K	380M	310K

Table 3.13: Estimated resources needed for mixed-level simulation

scaled with the problem size. However, a designer's work schedule is punctuated with various idle periods that are ideal for performing simulations. The lengths of these intervals are better estimates of the amount of time that can be set aside for simulation. Table 3.12 lists some of these intervals along with their durations in seconds. While at first it may seem optimistic to believe anyone would plan to run simulations while away on vacation, this interval can be used effectively by other users who take over the vacationer's workstation in his absence. However, more realistically, several idle periods of approximately 500 to 1000 seconds are likely to occur during the working day, and the 40,000 to 50,000 second idle overnight period can be very useful in practice. However, the designer is likely to expect multiple simulations to be performed overnight or over a weekend. In Table 3.13, the various estimates made so far are combined to form machine requirements. Based on the above arguments, time limits for the smallest, largest and example simulations are estimated as shown. Note that the performance required must be sustained for the duration of the simulation, so, based on the figures in Table 3.10, the peak machine performance needs to be 10 to 20 times larger than this. The memory needed is estimated using the memory

models in Table 3.4. (The static memory usage of approximately 1MB must be added to these figures.) The lowest disk storage estimate is calculated assuming that 200 bytes per device per timepoint are needed to store the circuit waveforms and no device internal states are saved. The number of timepoints accepted is assumed to be 90% of the total number calculated. The upper limit is established using an estimate of 100 bytes per mesh node (or 33 bytes per equation) to store the device internal state, assuming every device state is saved at each timepoint. The same assumptions are used for the example simulation, except that the additional assumption is imposed that device solutions are saved for only 10% of the timepoints. The I/O bandwidth is computed assuming a 1% time overhead is allowed to store the required data. At its upper bounds, CIDER is estimated to require 20 GFLOP/S processing speed and 1.3GB of main memory from the computing system used. This is comparable to the resources needed by many of the grand challenge problems of science and technology.

3.5.3 Assessment of Limitations

A look at the characteristics of any of the RISC-based computers described in Table 3.5 shows that they all have adequate performance and memory to solve the small simulation problem in the allowed time. In fact, each has excess capacity as far as this problem is concerned. The required disk space and I/O bandwidth are also well within the capabilities of these computers. This suggests that less powerful computers such as older RISC machines or personal computers could be used for such problems.

In contrast, the example problem requires more resources in some areas than is currently possible with these machines. Disk space and I/O bandwidth are both large but still manageable. The memory needed is noticeably greater than that installed in these systems. However, it is possible today to obtain compute servers, such as the DECsystem 500/240, that have more than 81MB of real memory installed. Thus, the critical limit here is the floating-point CPU performance which is 10 to 50 times too low on these machines to provide a sustained performance of 35 MFLOP/S. Even using the fastest RISC-based system available today (a DEC 10000-610 Alpha AXP system running at 200 MHz or 43 MFLOP/S LINPACK [DONG93]), one can optimistically expect only about 10 MFLOP/S sustained performance for CIDER. The extra performance gains must be supplied by either moving to a traditional vector supercomputer such

as a uniprocessor CRAY [DONG86] or a parallel supercomputer such as an Intel iPSC [IPS92b]. In either case, CIDER would need to be partially rewritten to take advantage of the special features of these systems.

For the largest problem, even today's fastest established supercomputers, all of which employ multiple processors, are only rated with *peak* performances in the neighborhood of 20 GFLOP/S [BELL92a]. Achieving sustained performance at this level requires an additional order of magnitude or more of peak performance. Next-generation parallel supercomputers such as the Intel Paragon and Thinking Machines CM5 achieve this performance using massive parallelism (> 1000 processors), but have only been available for a short period of time at a small number of installations. By way of contrast, large, fast memories with multi-gigabyte capacities are available in present high-end supercomputing systems. In addition, I/O bandwidths greater than a GB/S are available. The larger problem is likely to arise when trying to post-process and visualize close to 40GB of output data.

3.6 Summary

A detailed performance analysis of CIDER has been undertaken. Execution-time profiles show that 99executing device-level code. Tests of the device-simulator performance of CIDER show that the time per iteration is on the order of 10's of milliseconds for 1^D numerical models and on the order of seconds for 2^D numerical models. One to twenty MB of memory are used per simulation.

A set of 12 benchmark circuits have been run on 5 different RISC-based computers. Total run time for the benchmark set is between 4000 and 20000 seconds. Converting to execution speeds, the 5 machines have sustained performance of between 0.7 and 3.0 MFLOP/S.

Using the preceding performance and memory usage measurements, estimates of the resource requirements needed to enable mixed-level-simulation-based IC design have been made. Existing RISC-based machines are adequate for small design problems. However, the largest problems, estimated to need sustained performance in the 10's of GFLOP/S range, are beyond the capabilities of even the fastest computers built today. Such very fast computers all use parallel computing to achieve high levels of performance.

Given this assessment of the needs of CIDER and the importance of parallel computing in meeting those needs now and in the future, one obvious research direction is to explore the potential for exploiting parallelism in CIDER. This is the subject of the next chapter.

Chapter 4

Parallel Algorithms for Mixed-Level Circuit and Device Simulation

4.1 Overview

In the previous chapter, it is demonstrated that the performance and memory capacity of present-day engineering workstations strongly limit the size and number of mixed-level simulations that can be used to design a particular circuit. In addition, projections indicate that this will remain true even as individual workstations become more powerful. In the absence of more efficient simulation algorithms, it becomes necessary to employ larger, more powerful computing systems to reduce simulation time or increase problem size. In particular, scalable, high-performance computing (SHPC) systems such as the Intel iPSC/860 are a promising alternative for expanding the domain of applicability of mixed-level simulation. These systems employ parallel processing technology to increase performance beyond that achievable by a uniprocessor system. The number of processing elements can number into the 1000's; however, a system with 10 to 100 elements would be more typical.

In this chapter, parallel algorithms for mixed-level circuit and device simulation are presented. Such algorithms are needed to exploit the multiple processing elements in an SHPC system. First, the parallelism available when using direct-method

circuit and device simulation is exposed. Algorithms are drawn from previous research which exploit parallelism at either the circuit or device level. Each algorithm is assessed in terms of its appropriateness for mixed-level simulation. Emphasis is placed on determining each algorithm's strengths and weaknesses in a distributed-memory computing environment. Based on this analysis, an architecture and algorithms for a parallel mixed-level circuit and device simulator are proposed.

4.2 Terminology for Parallel Computer Architectures

Over the years, a wide variety of computers have been developed that are based on parallel architectures. In this work, attention is restricted to those computers that are capable of executing separate instruction streams with separate data sets on each processor. Such multiple-instruction multiple-data (MIMD) machines [FLYN66] are readily available commercially, largely because they are sufficiently general in capability to be applied to a number of interesting problems. Within this class of machines, there is still considerable variation in the architectural details. The machine taxonomy used in [BELL92a] is used in this work to broadly classify these machines.

Two attributes related to memory organization distinguish between the different types of MIMD computers. First, the physical memory can be either centralized or distributed throughout the machine. In either case, delays are incurred if two or more processors need access to the same piece of data. Central memories are difficult to scale to large numbers of processors since a connection must be provided for each one and sufficient memory bandwidth must be available to avoid having memory access time become a bottleneck in the computation. On the other hand, distributed memories *can* be scaled to large sizes since only a few (often just one) processor-memory connections are needed for each portion of the memory. SHPCs are therefore distributed-memory machines. Because there are no direct connections between distributed memories, a separate communication network must be provided to allow processors to exchange information.

The second distinguishing attribute is the program's view of the memory. A *multiprocessor* employs a single address space that can be supported in hardware using either central or distributed memory. A *multicomputer* has multiple address spaces and either the programmer or the compiler must generate explicit message-passing

commands to communicate data between computers. While there are many examples of distributed-memory multicomputers, no examples of central-memory multicomputers are identified in [BELL92a].

The distinctions between central or *shared* memory and distributed memory at the hardware and software levels have not been well resolved in the parallel-processing community. Part of the reason for this is that one class of hardware can often be emulated using a different type of hardware. As a result, it is easy to become confused when discussing differences between the various architectures. In the following, the convention has been adopted that when used alone the terms *shared* and *distributed* memory always refer to the programmer's view of the memory. This is appropriate since most of what follows is concerned with how to program parallel machines. When necessary, implementations on specific types of hardware are identified using the full name of the hardware type (e.g. X was implemented on Y, a distributed-memory multicomputer). The term *processor* by itself refers to a single processing element of either a multiprocessor or a multicomputer.

4.3 Obtaining High Parallel Efficiency

In a parallel algorithm, the problem to be solved is subdivided into smaller tasks in the hope that overall execution time can be reduced by executing multiple tasks at the same time. The degree to which an algorithm is successful in achieving this goal is determined by several factors:

- *Problem Parallelism* : the number of tasks that can be executed independently at any given time.
- *Problem Granularity* : the ratio between the number of computations in a task and the average amount of data transferred between tasks.
- *Machine Parallelism* : the number of processing elements used to solve the problem.
- *Machine Granularity* : the ratio between the computation rate and the communication/synchronization rate of the machine.

- *Load Balance* : the degree to which the tasks are evenly distributed amongst the processors.

The best measure of the effectiveness of a parallel algorithm is its speedup S:

$$S(P) = \frac{T_s}{T_p(P)} \tag{4.1}$$

where T_s is the execution time of the best serial algorithm and $T_p(P)$ is the execution time of the parallel algorithm on P processors. This measure reflects the actual improvement that the end user will see. However, it is often more convenient to talk in terms of the algorithmic speedup S^a, the ratio of the execution time of the parallel algorithm on one processor to the time on P processors:

$$S^a(P) = \frac{T_p(1)}{T_p(P)} \tag{4.2}$$

The ratio between these two, S^a/S, is a constant that reflects the performance lost when moving from a serial algorithm to a parallel one. This efficiency ratio, η^a, is always less than or equal to one for three main reasons. First, the parallel algorithm may be inherently less efficient on a uniprocessor, but still be chosen because it exhibits better speedup than the best serial algorithm. Second, even if the two algorithms are essentially the same, there is generally some overhead in the parallel algorithm in order to manipulate the task structure. Finally, by definition, the serial algorithm used has to be the best known. If the parallel algorithm is faster on a single processor than the previous best serial algorithm, then it becomes the best serial algorithm.

The following equation is very useful in understanding the performance gains offered by parallel processing [PATT90]:

$$S^a = \frac{1}{((1-F) + \frac{F}{P})} \tag{4.3}$$

where F is the fraction of the execution time of the original problem that can be run in parallel on P processors. This formula, known as *Amdahl's Law*, implies that there is a limit to achievable speedup determined by the sections of code that are essentially serial in nature. However, in practical cases, the serial portion can be made small enough that large speedups can still be attained. One situation where this happens is when the parallel fraction grows with problem size faster than the serial fraction. For example, in mixed-level simulation, the device-level execution time can be scaled up by using larger meshes, while the circuit-level time remains constant.

Amdahl's Law as shown assumes that perfect speedup is obtained on the parallelized section of code. An alternative version of Amdahl's Law that takes into account nonideal speedup of the parallel fraction is:

$$S^a = \frac{1}{((1-F) + \frac{F}{S^a_{par}})} \tag{4.4}$$

where S^a_{par} is the speedup of the algorithm used for the parallel section of code.

Machine parallelism and granularity are determined by the architecture of the target system and are relatively independent of the problem solved. (The total number of processors available is fixed, but less may be used if it is known they will be ineffective.) The problem parallelism is determined both by the total number of tasks and the dependencies between the tasks. Since the total execution time is fixed, as the number of tasks increases, the computation size per task decreases. In addition, the amount of communication and synchronization needed generally increases as the size of the tasks decrease. Thus, problem parallelism and problem granularity are intricately related, and attempts to improve one at the expense of the other may not lead to reduced run times. Finally, load balance is affected by how well the problem "fits" the machine. A multiplicity of factors affecting load balance complicates the problem of *scheduling*, the mapping of tasks onto specific processors. In general, any nonhomogeneity in the problem, the machine, or the mapping of the problem onto the machine leads to load imbalance. Problem nonhomogeneities include differing task sizes, differing amounts of communication, and different numbers of tasks at different stages of the algorithm. Machine irregularities include heterogeneous machine architectures, different clock speeds for processors with the same architecture, and different amounts of memory on different processors. Mapping imbalances occur when the problem parallelism is not evenly divisible by the machine parallelism.

One way to visualize the dependencies between tasks is a *task graph*, as shown in Figure 4.1. Each node in the graph corresponds to one task in the overall computation. Each arc corresponds to a dependency between the node at the beginning of the arc and at the end of the arc. The beginning node must execute to completion before the end node can begin execution. Nodes are labeled with a task identifier and the estimated number of operations needed to perform the task. Arcs are labeled with the amount of communication between the two tasks. A lower bound on the execution time can be obtained by assuming that an unlimited number of processing elements

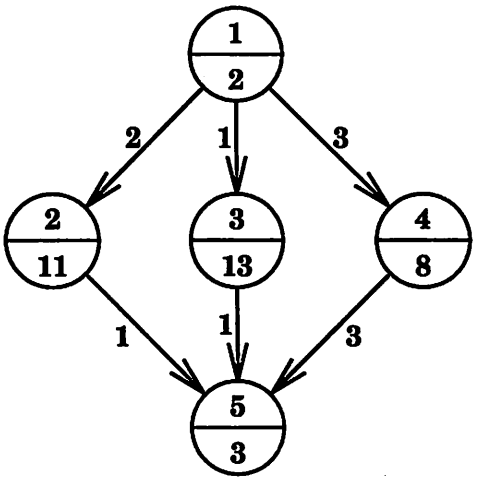


Figure 4.1: Example task graph

are available and that communication is instantaneous. In this case, total execution time is equal to the maximum sum of the task times on any path that traverses the graph from beginning to end [LEWI92].

Inevitably, some of the tasks must be executed serially on a single processor. How the results of such serial tasks are accessed depends on the memory model of the machine. In a shared-memory model, serial tasks can be performed by a single processor and the results left in the shared-memory where all processors can gain access to them. On a distributed-memory machine, the task is still executed on one processor with the results being passed to the other processors via explicit message-passing. In some cases, it may be advantageous to avoid message-passing by distributing the task input data instead and letting each processor compute the results on its own. This can conceivably save time if the volume of input is less than the volume of output. This optimization is known as *task duplication*.

4.4 Available Parallelism

In the previous section factors affecting parallel efficiency are identified. One of these is the amount of parallelism inherent in the problem. Maximum parallelism is achieved when the computational tasks can no longer be subdivided. However, since parallelism and granularity are interdependent, it is typical to approach the problem by discussing different *levels* of parallelism. Figure 4.2 shows the three

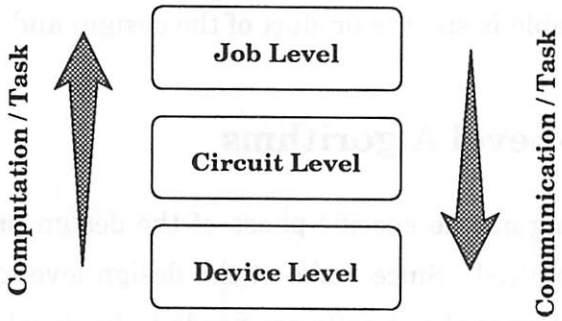


Figure 4.2: Levels of available parallelism

main levels of parallelism that are observed in circuit design based on mixed-level simulations. At the design level, the overall design flow is divided into a number of phases (for example, worst-case analysis or optimization), where each phase may require a number of simulations. Each task at this level then corresponds to a complete analysis of the circuit. At the circuit-level, an analysis is decomposed into the steps in the direct-method circuit-simulation algorithm: CKTload, CKTfactor, CKTsolve, and CKTtrunc. Each task operates on some part of one of the main circuit data structures: the per-device, per-model and per-instance lists or the circuit's sparse matrix and RHS. At the device level, tasks are created by subdividing the work associated with each numerical device: DEVload, DEVfactor, DEVsolve, DEVmisc, DEVcontrib, and DEVtrunc. Parallelism is achieved by partitioning either the mesh or the device's matrix and RHS.

In theory it is possible to exploit parallelism at more than one level at the same time. For example, design-level parallelism and circuit-level parallelism can be exploited simultaneously by running multiple simulations and distributing each simulation across a subset of the total pool of available processors. By using such techniques, problem parallelism can be increased in situations where there are insufficient tasks to keep all the processors busy. However, more sophisticated data structures and scheduling algorithms will be needed to manage this multi-level parallelism.

Example: Suppose a worst-case analysis of a circuit requires simulations at low, nominal, and high temperatures, a total of three simulations. In addition, suppose that the circuit contains two identically modeled numerical devices. Assume that each analysis takes two units of computation time, one for each numerical device. The

parallelism available is six, the product of the design- and circuit-level parallelisms. ■

4.5 Design-Level Algorithms

Depending on the specific phase of the design process, tens to hundreds of tasks may be produced. Since tasks at the design level consist of individual circuit simulations, little specialized software needs to be developed to manage these jobs. Instead, the existing batch processing facilities of the operating system can be used for job control. In a UNIX environment, Bourne and C Shell scripts can be used to manage the necessary files and start the jobs. On a multiprocessor, job control software is generally built into or layered on top of the operating system. For example, the DYNIX operating system used by Sequent central-memory multiprocessors automatically distributes jobs from a shared run-queue. In a networked IC design environment, the RSH command can be used to start jobs asynchronously on remote machines in the network. Alternatively, a simulation server process can be used on each workstation that accepts and executes jobs that are submitted in the form of text input files. Load balancing can be achieved by having the user choose where to submit jobs. However, this can become a bottleneck if many jobs are needed, since each one requires time-consuming and tedious direct user intervention. A network queuing system such as DQS [GREE93] that runs on top of the existing workstation operating system may be preferred. Such a system generally handles jobs from multiple sources and is preferable in a multiuser environment. In the longer term, a true, distributed operating system such as SPRITE [OUST88] may become standard for managing processes in a networked environment.

Ideally, whatever technique is used to distribute the jobs, the details of the implementation should be hidden from the user. One way to accomplish this is by integrating file and job management facilities into an IC design framework such as is done in NECTAR [KELL90]. If the user-interface is well-planned, it will not need to change when the underlying job distribution mechanism does. The design of such a framework, specifically one that supports traditional IC design tasks as well as technology CAD activities is beyond the scope of this thesis. For more information, the reader is referred to [CHIN92].

The primary advantage of design-level parallelism is that an optimized se-

rial mixed-level simulator can be used to run the jobs. Programmer productivity is enhanced because software development tools for serial machines are currently better than those for parallel machines. Updates and enhancements to the serial code are immediately available in the parallel environment. The performance of this approach is limited by a number of factors. First among these is the overhead associated with starting a job on a machine other than the user's personal workstation. Time is required to send both the executable and input files to the destination processor, and to return the output files to the user. Typically, this will take on the order of seconds to accomplish. A design phase consisting of many very short jobs will not achieve appreciable speedup using this form of parallelism. Fortunately (or unfortunately depending on one's perspective), as shown in Chapter 3, mixed-level simulations typically require minutes to hours of CPU time, so startup overhead is not a major problem. However the original motivation for employing parallel processing is that the performance and capacity of existing workstations is inadequate. Design-level parallel processing implicitly assumes the opposite of this belief: that mixed-level simulations *can* be run on a single processor effectively. Thus, design-level parallelism is likely to be limited to circuits containing very few numerical devices. This limitation can be overcome by mixing design-level parallelism with circuit-level parallelism as suggested earlier.

Example: Suppose that it is desired to obtain parameters for a compact MOSFET model so that it closely approximates the behavior of a numerical model. Two-dimensional MOSFET simulations are required at a number of channel lengths and bias conditions in order to obtain the geometry and voltage dependences of the model parameters. A mixed-level simulator used as a device simulator can obtain the necessary data. Since only one numerical device is simulated in each job, the job-per-processor approach will work well. Overhead is not a concern since 2^D simulations are being used. ■

4.6 Circuit-Level Algorithms

If the abnormally large computational needs of numerically modeled devices are ignored for the present, then parallel mixed-level simulation is essentially identical to parallel circuit simulation. Parallel algorithms for direct-method circuit simulation

have been the subject of intensive research in the past decade, and the available literature has become extensive. Techniques have emerged for each of the major steps in the circuit simulation algorithm. Of these, CKTload and CKTfactor/CKTsolve are the most important. The time for sparse-matrix loading grows linearly with circuit size, while sparse-matrix solution time grows as $O(N^{1.1})$ to $O(N^{1.5})$, where N is the number of equations in the system [NEWT83]. The time for timestep calculation and convergence checking also grow linearly with problem size but the constants of proportionality are much lower.

Example: Figure 4.3 shows the time per iteration taken for both circuit loading and

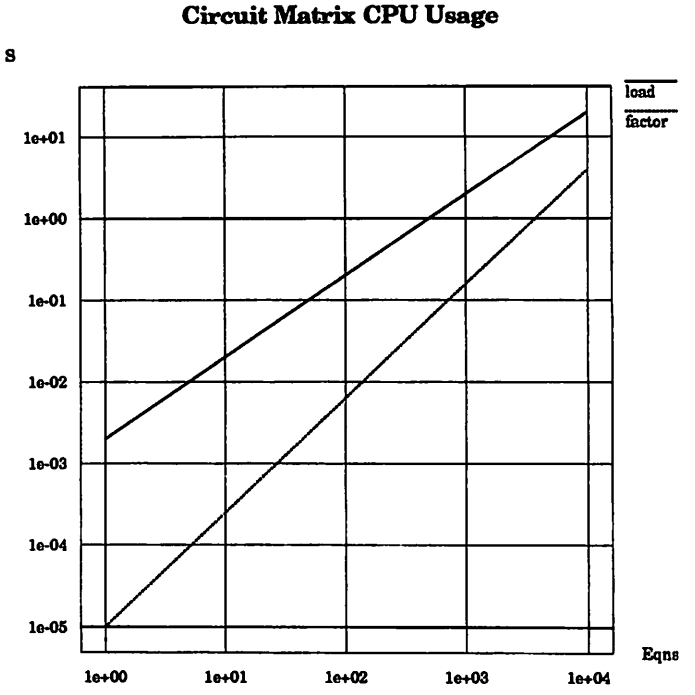


Figure 4.3: Time per iteration to load and factor circuit matrices

solving under the following assumptions:

- The time to evaluate a compact device model is 1.0 ms/iteration.
- The time to factor the matrix goes as $10.0\mu s \cdot N^{1.4}$.
- There is one equation for every two devices in the circuit.

These assumptions are consistent with the results presented in [QUAR89]. In this situation, the problem size must be at least 565,000 equations before CKTfactor begins to dominate the overall runtime. Since the data in [QUAR89] only cover the range up to about 2,000 equations, in realistic situations the load time will always dominate. ■

The parallel techniques for matrix loading and solution are described next. In 1989, the problem had been sufficiently well studied for a review paper to appear in the Proceedings of the IEEE [SALE89]. To a large degree, what follows summarizes the information found therein. Algorithms have been developed for a variety of high-performance computer architectures. Early research focused on the use of vector processors [VLAD82], [YAMA85] and special-purpose hardware [KO86]. Shared-memory multiprocessors have been used in a large number of studies: [JACO87], [BISC86], [COX91], [SADA87], [CHEN88], [CHAN88], [YANG90]. In addition, algorithms and implementations for distributed-memory multicomputers have been reported [YUAN88], [TROT90], [PACH91]. To date, algorithms for scalable, distributed-memory multiprocessors like the Kendall Square Research KSR 1 have not appeared.

4.6.1 Parallel Model Evaluation

It is well known that the task of linearizing and evaluating device models is highly parallel [JACO87]. Difficulties arise when the individual device contributions must be loaded into the circuit matrix and RHS. Some kind of synchronization or communication is needed to ensure that the final system accurately reflects the state of the Newton-Raphson iteration. Two basic approaches are available for controlling access to the matrix and RHS: lock-based methods and barrier-based methods. Barrier-based methods are applicable to both multiprocessors and multicomputers; locks can only be employed on multiprocessors.

Lock-Based Methods

In a lock-based algorithm, access to certain *critical sections* of code is restricted to the one process which that has acquired rights to the lock. If another process attempts to acquire the lock, it blocks until the current owner releases the lock. In this way, it is guaranteed that no two processes are executing the critical section at the same time. If the portion of each model-evaluation routine that accesses the circuit matrix

and RHS is protected by a lock, the final system will be correct [JACO87]. However, this approach suffers from the drawback that the time to acquire and release the lock is not always negligible compared to the time spent executing in the critical section. This is especially true of distributed-memory multiprocessors, where the memory for the lock will not necessarily reside locally on the processor, necessitating a non-local memory access to acquire the lock. One way to circumvent this problem is to group several model evaluations together into a single task and to load all the contributions for the task after acquiring the lock. This decreases the importance of locking and unlocking, but increases the length of the critical section. The inherent problem granularity has not changed; only the overhead in the algorithm has been reduced. A better approach is to exploit the observation that most of the time processors will not be trying to access the same locations in the shared memory. By providing multiple locks, one for each row of the matrix, the likelihood of simultaneous locking is diminished. However, the number of locking operations is increased because each device must access multiple rows in order to load all its contributions. It will also be necessary to find groups of devices that access the same rows in the matrix in order to form larger tasks.

Barrier-Based Methods

A barrier is used to prevent any one process from continuing onto the next phase of a computation until all processes have finished the current phase, thereby synchronizing the processes. In the context of circuit matrix loading, a barrier is used to prevent any process from computing the final entries of the circuit matrix until all contributions have been calculated. This approach thus requires considerable extra memory while temporarily storing these contributions. (A factor of 30% is reported in [YANG90].) The ultimate matrix entries are computed by summing the contributions to each entry in parallel. On a multiprocessor, the entries are collected into a groups of equal size, and each group is assigned to a different processor. Theoretically, the time for this step goes as $O(NC/P)$, where N is the number of matrix entries, C is an upper bound on the number of contributions per entry, and P is the number of processors. However, this result ignores the effects of contention in a bus-based multiprocessor and non-uniform memory-access time in a distributed-memory multiprocessor. As a result, actual performance is likely to be worse than this result suggests.

On a multicomputer, the individual contributions will reside in the separate, local memories of the processors. Message-passing is then needed to transmit the partial entries between processors. In order to simplify the coding of this step, it is desirable to provide a copy of the entire sparse matrix on each processor on the system. All processors then have a uniform view of the problem. In the first half of the matrix load operation, each processor loads the contributions for the devices assigned to it into its local copy of the matrix as they are computed. If necessary, the elements are copied from the linked-list sparse-matrix data structure to a buffer array prior to message passing. Efficient global reduction operations can be used to sum the arrays in $O(\log P)$ time per entry, or $O(N \log P)$ in total. This assumes that the time to perform each step of the global reduction is constant. On mesh-based multicomputers such as the Intel Paragon, the necessary data will not immediately be available in adjacent processors, and extra time will be needed for data transfer.

Special Considerations

The special characteristics of the matrix load problem arising from mixed-level simulation have been ignored. In particular, it has been assumed that the time per model evaluation is relatively constant for all devices, and that the time to update the sparse matrix is comparable to the model-evaluation time. However, numerical devices require the expensive solution of partial differential equations in order to compute their contributions to the circuit matrix. In Chapter 3, it is shown that this step dominates the overall computation time and that the time per device is dependent on the size of the mesh used to discretize the device, or the related measure of the size of the device-level sparse system. In light of this, the time to evaluate the compactly modeled elements of the circuit is likely to be negligible. Only when the number of numerical devices is a very small fraction of the total will this cease to be true.

Example: Assume that the CPU time to evaluate a compactly modeled semiconductor device is 300.0μs per transient iteration. Then, when using the execution time models of Chapter 3, a typical 1^D numerical diode with a 100 node mesh that takes about 30 ms per transient iteration is equivalent to 100 compact devices. A typical 2^D numerical diode with a 20×20 mesh that takes roughly 4s per transient iteration is equivalent to about 4s/300μs $\approx 13,000$ devices. ■

If the compact devices and matrix access are ignored and the numerical devices are each assumed to take the same time to evaluate, an upper bound on the speedup can be obtained using the following equation:

$$S(D,P) = \frac{D}{\left\lceil \frac{D}{P} \right\rceil} \tag{4.5}$$

where D is the number of numerical devices. Two limiting cases are easily identified: $S \approx P$ when $D \gg P$, and $S = D$ when $D \leq P$. For small circuits, the latter limit is the more important one. (In the extreme case of a circuit with only one numerical device, apparently no parallelism is available using this technique. However, tiny speedups may be observed due distribution of the compact devices.) The various assumptions leading to this result need to be examined. The assumption that the compact-model evaluations are negligible is generally true, but special cases exist where the time per iteration can be considerable (for example, lossy transmission-line models [ROYC91]). In addition, the assumption of uniform evaluation time for numerical models is violated when different mesh sizes are used for the various devices. This can occur in real circuits, as shown later in Chapter 5. In such cases, the speedup is given by:

$$S(P) = \frac{\sum_p T^p}{\max_p T^p} \tag{4.6}$$

where T^p is the time to evaluate the devices assigned to processor p:

$$T^p = \sum_{d \text{on} p} T_d$$

and T_d is the non-unit time needed for device d. In this equation, no distinction is made between numerical and compact devices; this provides a way to incorporate the effects of computationally expensive compact devices. The numerator is simply the total time to evaluate all the devices. The denominator is the maximum time spent by any one processor evaluating the devices assigned to it. Speedup is best when the maximum time for a processor is minimized. This is a minmax optimization problem [LEWI92] and is developed further in Section 4.9.

The assumption that the matrix access time is negligible is now examined. For a multiprocessor architecture with uniform-memory access time, the time to load ten to twenty entries in the matrix and RHS is far outweighed by the time to calculate those entries. Even the most inefficient locking methods are likely to achieve reasonable

performance. However, on systems where memory is distributed, the time to access the data is critically dependent on the speed of the underlying interconnect layer. The sparse-matrix access time will be relatively low only if the volume of communication is low and the bandwidth is high. For mixed-level simulations, the circuits simulated are relatively small. Specifically, if a small circuit is defined as one where the matrix loading time is less than the matrix solution time, then even if ordinary compact models were used, the circuits would be considered small. Adding the orders of magnitude increase in load time due to numerical devices solidifies this observation, and also increases the range of circuits that are considered small. Another definition of a small mixed-level circuit is one where the average size of the device-level matrices is greater than the size of the circuit matrix. By this definition, the circuits also would be considered small. Thus, the amount of data accessed remotely is likely to be fairly low. However, in certain distributed-memory multicomputers, the speed of the interconnect has not been scaled with the speed of the processors, and special care must be taken to ensure that an absolute minimum of message traffic is generated. Practical examples of this problem are demonstrated in the next chapter.

4.6.2 Parallel Sparse System Solution

Compared to parallelizing matrix loading, solving sparse systems of equations is much more difficult. The algorithms required are more complex, and the speedup is not as good. This is due in part to the complexity of optimized serial algorithms for L/U decomposition, and forward and back substitution. In the class of all sparse-matrix problems, circuit-matrix problems are particularly difficult because they typically involve nonsymmetric, unstructured, indefinite sparse matrices with potentially complex-valued entries. For a survey of the available parallel algorithms for symmetric, positive-definite systems, the reader is referred to [HEAT91]. Some of the methods described there can be adapted to nonsymmetric systems, although specialized algorithms for circuit simulation have also been developed [JACO87]. Respectable speedup and efficiency have been achieved on small numbers of processors. For example, efficiency between 70% and 80% is about average for up to 8 processors. No good speedups on large numbers of processors have been reported, primarily attributable to the fact that the researchers seem not to have had access to large parallel

machines. Nonetheless, as the following example demonstrates, it is clear that the existing methods will not perform well on larger machines.

Example: A parallel efficiency of 75% is achieved on 8 processors. When Amdahl's Law (Equation 4.3) is used in reverse, the fraction of the solution parallelized is approximately 95%. The maximum speedup is about 20, and the efficiency drops to 50% at 20 processors. ■

Fortunately, as described in the Section 4.6.1, the matrices encountered at the circuit-level in mixed-level simulations are typically small. Recall from Chapter 3 that less than 0.5% of the total time is spent in the CKTfactor and CKTsolve phases for the three example execution profiles. Thus, although sparse-matrix solution is of great concern in stand-alone circuit simulation, it is much less important in mixed-level simulations. As a result, respectable overall speedup can be achieved even when the sparse-matrix solution is performed serially. Only for atypically large circuits that are run on scalable parallel machines will the circuit-level matrix solution time become a problem. Then it is unclear whether the existing methods will be able to provide even small speedups on such a large number of processors due to communication overheads.

4.7 Device-Level Algorithms

At the circuit-level, mixed-level simulation creates an abnormal balance between load and solve times. At the device-level this is not the case. In almost all ways, the device-level problem is identical to the problem of parallelizing direct-method device simulation. However, comparatively speaking parallel device simulation is much less well studied than parallel circuit simulation. The most notable example of a parallel device simulator is PARALLEL PISCES [LUCA87a]. Other applications of parallel machines to device simulation have focused on the iterative methods used in three-dimensional device simulation [WU91], [WEBB91].

For the most part the operations needed for device simulation have direct analogs at the circuit-level. The device-level matrix is loaded, factored and solved, convergence is checked, and a new timestep is chosen. The two major additional steps are norm-reduction during DC analyses, and numerical calculation of currents and conductances. However, for direct-method device simulation, these extra steps

can be formulated as additional load and solve steps, reusing the already computed L/U factors of the device matrix. Thus, the loading and solving algorithms for circuit simulation are reconsidered in light of the different tradeoffs involved in device simulation. In particular, parallel sparse-matrix factorization becomes a necessity at the device-level. Consequently, a summary of the matrix factorization technique used in PARALLEL PISCES is provided.

4.7.1 Parallel Element Evaluation

In the context of mixed-level simulation, there are two major differences between loading at the circuit-level and loading at the device-level. First, whereas the number of devices, especially numerical devices, is small at the circuit-level, this is not necessarily true of the number of elements in a device's mesh. As noted in Chapter 3, typical device meshes contain hundreds to thousands of elements. At the same time, the workload per element is more uniform than at the circuit-level. In general, the same physical models are evaluated across the entire mesh, and each element is the same shape as all the others. For example, the diode simulations in Chapter 3 result in nearly uniform element loads. A major exception to this occurs when different material domains are contained in one device, as in a MOSFET. In this case, the carrier-continuity equations do not need to be solved in the insulating regions, and the physical models for current terms need not be evaluated. Another exception, also important in MOSFETs, is the physical model for the inversion-layer mobility. This model is only applied in parts of the device and takes longer to evaluate than the standard field-dependent mobility model used in the rest of device. As a result of these factors, inversion-layer elements are more expensive to compute than other semiconductor elements, which are more expensive than insulator elements. Differences in execution time for different element types should be taken into account when assigning the element evaluations to the processors. However, other concerns may limit the ability to distribute expensive regions of a device across the system. Foremost among these is the problem that the time to access the sparse matrix can no longer be neglected. In addition, the larger size of the device-level matrix demands an approach which is more memory efficient than the näive replication approach used for the smaller circuit matrix. A loading technique that is memory efficient and does not

require the use of locks is now presented. A second method closely tied to the factorization method used in PARALLEL PISCES is presented later. Both rely on the fact that the device-level matrix is derived from a spatial discretization with only local coupling. As such, they both would need to be modified in order to accommodate the non-local coupling introduced by the inversion-layer mobility model. It is not immediately apparent how this might be accomplished. This local coupling requirement explains why this technique is not used more in circuit simulation, since feedback loops, clocks and supply lines all introduce non-local coupling.

Element Coloring Method

Coloring is a technique used to divide up the device's elements by assigning them to groups such that no two groups have elements which share nodes or edges in common. It was originally developed for use on vector machines like the CRAY computers [PINT90]. In Figure 4.4 a rectangular mesh has been divided using four

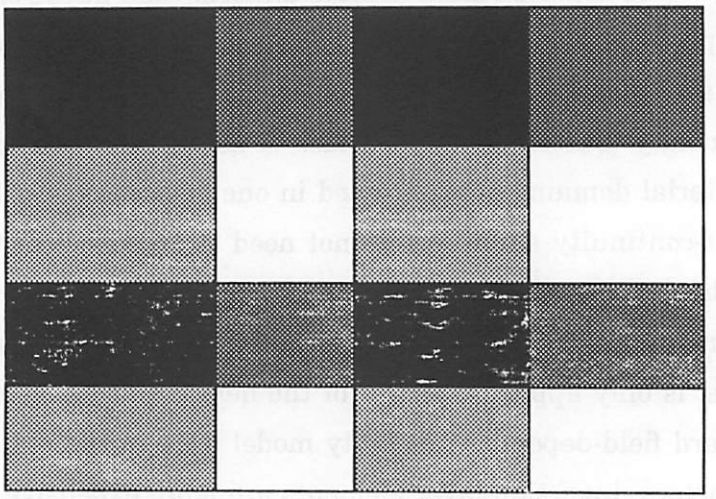


Figure 4.4: Coloring of a rectangular mesh using four colors

colors, the minimum necessary. Because there is no coupling between elements in the different groups, there is no need to worry about two processors accessing the same location in the matrix as long as the processors are synchronized between each group. If the number of elements in each group is large enough, the barrier overhead will be negligible. (On a vector processor, this technique reduces the cost of vector startup operations.)

Unless special efforts are made, the need for two processors to access the same memory location restricts this method to use on multiprocessor architectures. Presumably if elements are known to have different computation times, the groups can be created with this in mind. However, on a distributed-memory multiprocessor if elements of different colors that are spatially adjacent are not assigned to the same processor, many non-local memory accesses will be generated, degrading the performance of the algorithm. This suggests a different method for creating groups based on a spatial decomposition of the mesh. This method is described next.

4.7.2 Distributed Multifrontal Factorization

PARALLEL PISCES uses methods for matrix loading and factorization that are based on the spatial coherence of the underlying problem. Because the same underlying conceptual framework is used for both tasks, the algorithms end up being fairly well-matched. However, the load balancing problem is not identical for both parts, and conflicts may arise when partitioning the device.

The basic idea behind the technique is nested dissection of the problem mesh [GEOR73], a divide-and-conquer method of computing. Nested dissection is a well-known technique for ordering the equations in a sparse matrix in order to minimize matrix fill-in. It proceeds by finding a set of nodes in the mesh which divide the mesh into two halves. The equations are ordered by first considering all the nodes in one half, then the nodes in the other half, and finally the nodes in the separator. Ordering of the nodes in the two halves is performed by recursively applying the dissection procedure to each half. The result is a nested, bordered block-diagonal (BBD) matrix structure as shown in Figure 4.5. The generation and solution of such matrices on a distributed-memory multicomputer is described in [LUCA87b]. This same matrix structure has been used at the circuit level on both a central-memory multiprocessor [CHEN88] and a distributed-memory multicomputer [YUAN88]. A related circuit-level implementation is described in [COX91].

The advantage of the nested BBD structure is that it can be decomposed using a task graph similar to the one shown in Figure 4.6, where each task represents a portion of the overall device matrix. At each step of the algorithm, one level of tasks is factorized using L/U decomposition. Partially decomposed factors are then passed

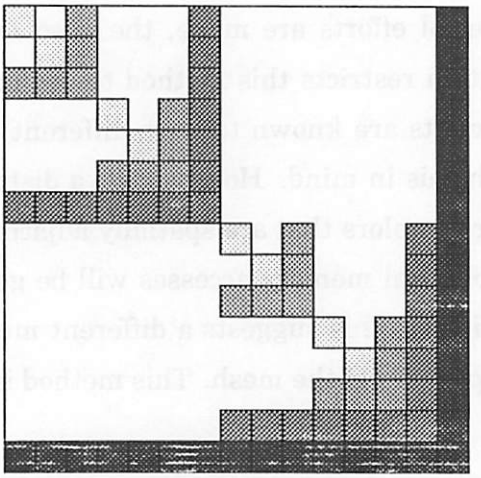


Figure 4.5: Nested, bordered block-diagonal matrix

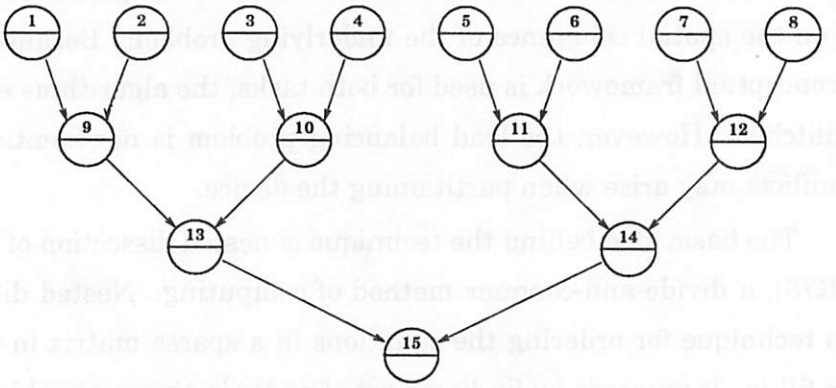


Figure 4.6: Task graph for NBBD matrix

to the next lower level, where they are combined and the process is repeated until at the lowest level the entire matrix has been factored. Forward substituition follows a similar procedure, while back substitution follows a reversed procedure where the processors start out working together and end up working alone. The task graph starts out with fairly high parallelism, but it is reduced by a factor of two at each level. If the task sizes are assumed uniform, then the speedup goes as $O(\frac{N}{\log N})$ on N processors, where N is the number of tasks at the highest level. Typically, however, task sizes will be larger at the lower levels. In this case, all the processors assigned to ancestors of a task can be grouped together to work in parallel. The tree-structure of the task graph guarantees that no processor will be assigned to more than one task using this criterion. If less than N processors are used, tasks in the highest levels can

be grouped together into larger tasks before being assigned to the processors. This fact is typically used to generate just enough levels so that each processor has one task at the highest level. If the problem is sufficiently large, the initial tasks will be much larger than the lower level tasks, and speedup closer to the number of initial tasks will be achieved. This can be useful when only small amounts of parallelism are needed at the device-level.

Because of the way the algorithm isolates various sets of nodes using separators, only part of the sparse matrix is held on each processor. This local matrix is similar to an ordinary sparse matrix except that equations exist only for the local set of mesh nodes and all the sets used to separate them. Thus, equations for the local nodes reside on one processor; for the last separator, each set resides on two processors, and so on, until the equations for the first separator are represented on all the processors. It is therefore desirable to keep the separator sizes small. Unfortunately, it is also a good idea to keep the number of nodes on each side of the separator balanced, which conflicts with the previous goal. Several automated methods which trade-off between the two objectives have been developed. These include traditional heuristic techniques such as the Kernighan-Lin algorithm [KERN70] as well as the more recently applied spectral bisection approach [POTH90]. Nonetheless, in real situations it is often difficult at each step to find small separators that bisect the nodes. This is especially true when the number of bisections is high compared to the total number of nodes so that very small sets of nodes need to be divided. As a result, task sizes are not uniform, load balance is degraded, and speedup falls off.

Once the nodes of the graph have been distributed across the processors, an implicit partitioning of the mesh elements has been defined as well. Since loading of the device matrix is based on a loop over the device elements and would be complicated if nodes belonging to different processors were part of the same element, the partitioning step must take this into account. Figure 4.7(a) shows a small mesh where this constraint is violated and it is unclear which processor should be assigned the element marked with '?'. In Figure 4.7(b) the problem has been corrected by repartitioning the nodes. Thus it can be seen that the partitioning problem is one of dividing the elements into two sets where the nodes on edges common to the two sets become the separator. The loading of the device matrix then takes place by having each processor evaluate all elements that have been assigned to it. For all nodes in the separators,

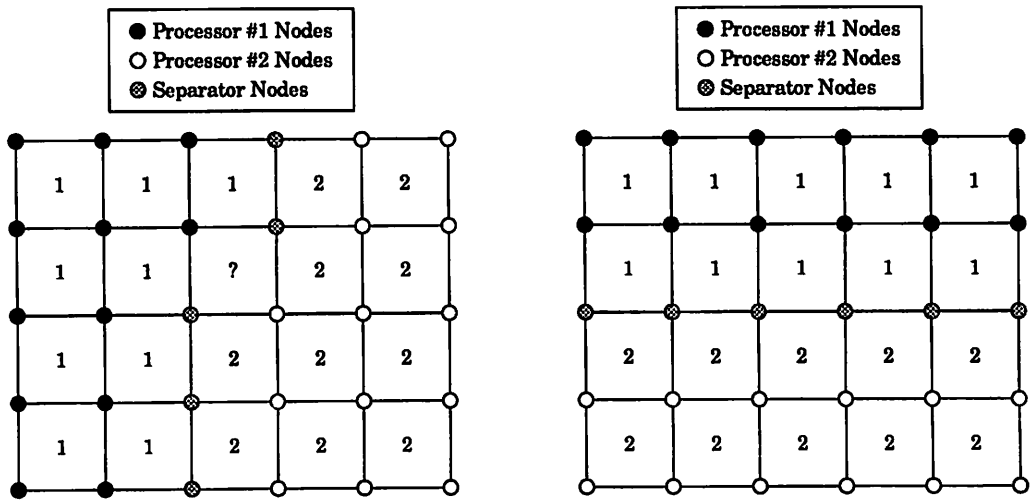


Figure 4.7: Two element partitions of a small mesh

partial contributions are calculated by two or more processors. These partial contributions are summed together during the course of the matrix factorization process. At times, equipartitions of the element blocks may not equally divide the remaining non-separator nodes because the nodes on the block boundaries have already been assigned to separators. Since the device-level load phase is balanced when equal numbers of elements are assigned to each processor while the device-level solve phase requires balanced partitions of the nodes at each step, a trade-off must be made. The partitioner should be able to optimize one or the other type of load balance depending on which phase is likely to dominate the overall computation time. For small problems, element balance will be better, but for larger problems, node balance will give better results.

4.8 Mixed-Level Algorithms

In the preceding sections the various alternatives for exploiting parallelism in mixed-level simulation have been reviewed. An architecture for a parallel mixed-level circuit and device simulator is now proposed. Knowledge of the special characteris-

tics of mixed-level simulation is used to guide the choice of algorithms and levels of parallelism exploited. Although algorithms are available for several architectures, the architecture proposed here has been specifically designed for distributed-memory multicomputers. It is also believed to be applicable to distributed-memory multiprocessors, although this has not been fully investigated.

4.8.1 Previous Work

The idea of using parallel computing to speed up mixed-level simulations is not new. In [MAYA88], the use of parallel model-evaluation is proposed where an assignment of one numerical device to each processor is made. The same basic idea is presented in [SCHR91] where a master-slave process arrangement is proposed for a network of workstations. The master job manages the circuit simulation and distributes the devices to the workstations which act as slaves. Messages containing the matrix updates are sent from the slaves back to the master, where the entries are loaded into the master's circuit matrix. This computational model is equivalent to the client-server model that is used in an actual implementation on a network of UNIX workstations [MEIN90]. Small device-level servers permanently reside on each workstation in the network. The circuit simulation half of a mixed-level simulator acts as a client to these servers sending requests for numerical device simulations to be initiated on particular processors as needed. All communication between the circuit simulator and the device simulators is passed through the device-level servers.

4.8.2 Proposed Architecture

Of the three levels of parallelism available, only circuit- and device-level parallelism must be exploited by the simulator itself. As a result, design-level parallelism has been excluded from consideration in the proposed architecture. Management of this form of parallelism is better left to a general-purpose CAD framework that can provide such facilities for other related tasks such as layout synthesis, design-rule checking, circuit extraction or design optimization.

Figure 4.8 shows the main software components of the algorithm architecture and the calling relationships between them. Each processor receives the same code for all components. However, the contents of the data structures vary from one processor

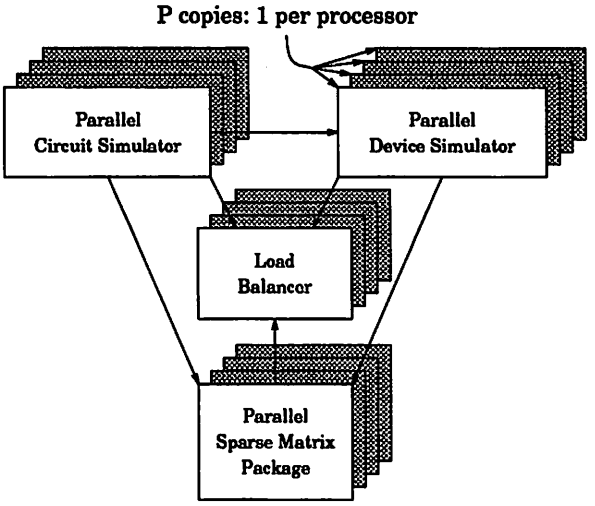


Figure 4.8: Components and call structure of proposed algorithm

to another. This programming approach is called the single program - multiple data (SPMD) model.

The parallel circuit simulator core is capable of performing parallel model-evaluation based on the assignment of devices to *processor groups*. The processor groups are divided into different levels where each processor belongs to exactly one group at each level. This is the only constraint on the group definitions. However, typically groups will be arranged so that one level contains many, small groups and the other levels contain fewer and fewer groups which combine groups at the previous level. In Figure 4.9 a four-processor hypercube is divided into three levels of groups using

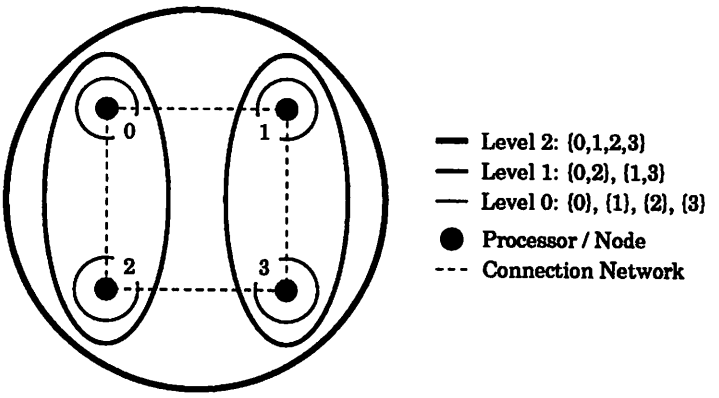


Figure 4.9: Processor groups for four node hypercube

this technique. At the highest level all the processors are in one group and at the lowest level each processor is in a group by itself. Compactly modeled devices are assigned to one-processor groups; numerical devices are assigned to either single or multiple processor groups. Every processor has a local copy of the circuit sparse-matrix data structure, so that each may perform loading without contention. The overall circuit matrix eventually resides in this local copy as well, after a barrier-based method is used to compute it. The parallel device simulator is used to compute the contributions of the numerical devices. For best performance this simulator should exploit both load and solve phase parallelism at the device-level. The device simulators on the different processors coordinate work on the numerical devices that are assigned to multiple processor groups. The different groups at a given level work on separate devices, thus exploiting parallelism at both the circuit- and device-levels simultaneously. Both the circuit and device simulator call on the parallel sparse-matrix package for direct-method matrix solution. A serial mode must included in the sparse-matrix package, since parallel matrix solution may not be effective for the small circuit matrices. The load balancer supports all three of the other components by statically dividing tasks among the processors.

A high-level description of the *multi-level model-evaluation* algorithm for transient analysis executed by every processor in the system is shown in Figure 4.10. It is an extended version of the algorithm in [PACH91] for parallel circuit simulation. The main difference is an additional level of looping needed to access the processor groups. Each processor has access to the complete circuit as described by the input file. Either independently or in concert, a partitioning step is performed that assigns devices to processor groups in a way that attempts to minimize load imbalances. Once computed, the partition is held fixed, since the cost of redistributing numerical devices dynamically is prohibitive on distributed-memory machines. A loop over the group levels is used to setup the necessary preliminary data structures (such as the mesh) for the numerical devices. In addition, all remaining initialization prior to entering the main loop is performed. On the first pass through the main loop the device-level sparse-matrix structures are allocated.

The main loop consists of a Newton-Raphson iteration followed by the computation of the next time step. The Newton-Raphson iteration consists of loading, solving and testing for convergence. The load phase consists of a nested loop over the proces-

sor groups and the devices in each group. For compact devices and numerical devices assigned to single processors, the contributions are directly loaded into the local copy of the circuit matrix structure. For distributed numerical devices, one processor in each group is designated to receive the matrix contributions and load them into its copy of the circuit matrix. Then a reduction operation is performed to accumulate the information in the local matrix copies into a global circuit matrix residing on a single processor. If necessary, this global version is then broadcast to all processors. If the global circuit matrix is only available on one processor, then it must factor and solve the system and distribute the solution to the other processors. Otherwise, each processor solves its local copy of the global system to obtain the solution. Convergence is tested using another loop through the groups and devices, followed by a test to make sure all processors have converged solutions. Once the Newton-Raphson iteration has converged, the solution is saved by a designated processor, and then the next time step is calculated. Timesteps based on local truncation errors are computed in each processor group, breakpoint limiting is applied and then the global minimum timestep is used as the next timestep.

4.8.3 Advantages and Disadvantages

The proposed architecture has two primary advantages over a simpler architecture that only exploits circuit-level parallelism. (This corresponds to a one-level grouping in the proposed architecture where each group contains a single processor. For future purposes this is referred to as the *one-level model-evaluation* algorithm.) First, better speedup can be achieved in situations where there is a mismatch between the number of numerical devices and the number of processors. For example, if D/P is not an integer, the leftover devices can be solved in parallel at a higher grouping level. This technique of dividing up only some of the iterations of a loop is known as loop spreading [LEWI92]. Although normally applied to much finer grained loops, the technique is just as applicable to very large grains as done here. The second advantage is the ability to solve larger device-level problems when necessary. Parallel processing at the device level distributes the memory usage of a numerical device across all the processors in its group. On many parallel machines the amount of memory available to a processor is restricted to the amount of real memory and cannot be extended by

using virtual memory techniques. In other words, a device that will not fit on a single processor can be solved using the additional memory of multiple processors. By the same argument, the capability of assigning numerical devices to different processors is also an advantage of the use of circuit-level parallelism. Thus, the additional memory that comes with parallel machines is very important.

The primary disadvantage of this approach is the extra complexity involved in its implementation. The algorithm requires not only a parallel circuit simulator but also a parallel device simulator. Modifying a modular circuit simulator such as SPICE3 to exploit parallel model-evaluation is relatively easy; redesigning a device simulator to exploit parallelism at each stage of the computation as in PARALLEL PISCES is a much greater undertaking. In addition, unless the device simulator is designed with mixed-level simulation in mind, it may be difficult to integrate into the above scheme. The proposed algorithm demands considerable flexibility on the part of the device simulator to support simulations in multiple processing groups. Such flexibility would not generally be required of a stand-alone device simulator where a single group containing all the processors would be normal.

One advantage of both approaches is the modularity of the basic mixed-level algorithm itself, as noted in [MAYA88]. The device simulator can be easily replaced with another one as long as the basic operations required by the circuit-device interface are still supported.

4.8.4 Software Requirements

The proposed architecture requires four major software modules: a parallel circuit simulator, parallel device simulator, parallel sparse-matrix package, and a flexible partitioner / load balancer. If parallelism at the device-level is ignored (as in the one-level model-evaluation algorithm), then a serial device simulator and sparse-matrix package can be substituted, and the load balancer can be simplified substantially. Certain capabilities must be incorporated in each of the components before they can be merged into a mixed-level simulator. It is assumed that the simulator will support the three basic types of circuit analyses: DC, transient, and AC small-signal analyses. Substantial savings in development time can be realized if a serial mixed-level simulator is available as a starting point.

For this application, developing a parallel circuit simulator from a serial one is a relatively easy job because the algorithm has been designed to have minimal impact at the circuit level. This topic is covered in detail in Chapter 5.

The parallel device simulator has several requirements that are inherited from the serial environment, as well as new ones due to the parallel algorithms. The device simulator DSIM used in CIDER was custom developed for CODECS because a sufficiently general simulator was not already available [MAYA88]. Much of DSIM was written from scratch; however, it was not necessary to write a sparse-matrix package because SPARSE was available. The parallel situation is worse than that originally observed in the serial case. General-purpose dense matrix software for central-memory multiprocessors has recently become available [DONG91], but similar software for distributed-memory machines and sparse matrices has not been publically distributed. Development of an appropriate parallel sparse-matrix package is thus the major obstacle in the implementation of a parallel device simulator. As demonstrated in [LUCA87a], parallelizing the remainder of the device simulator is fairly straightforward once a parallel sparse-matrix package has been defined. The main complication arises in the dividing the mesh and equations among the processors in order to achieve load balance.

The various requirements for the sparse-matrix package are summarized as follows:

- It must be able to handle both the real and complex matrices that arise during DC and AC analyses.
- Asymmetric matrices need to be supported.
- Multiple matrices arising from the different numerical devices need to be stored.
- Parallel solution of multiple matrices in separate processor groups is necessary to allow mixed-level parallelism to be exploited.
- Since memory-storage requirements are unlikely to be known accurately until runtime, dynamic-memory management should be incorporated.

Given that no up-to-date package for a single, real-valued, asymmetric sparse matrix is currently available for distributed-memory machines, the above requirements seem

impossibly overambitious for a single sparse-matrix package. However, once such a package does become available, the remaining features should be relatively easy, although perhaps tedious, to implement. The trickiest problem will be support for group-based processing, something that is presently an active area of research in the parallel-processing community. However, by the time the basic matrix package becomes available, group-based processing will be presumably better understood.

4.9 Mixed-Level Partitioner

The need to partition the workload to achieve load balance is a central part of any parallel algorithm. This problem has already been encountered a number of times in this chapter. The focus here is on a problem that arises in the multi-level model-evaluation algorithm. This is the problem of assigning the devices in a circuit to the multiple levels of processing groups. The goal is to find an assignment that minimizes the maximum completion time of all the device evaluations. In the absence of an exact solution to this problem, a heuristic approach based on simulated annealing [KIRK83], has been prototyped and several conclusions are drawn from observing its performance.

4.9.1 Multi-Level Partitioning Problem

The multi-level partitioning problem is expressed as follows:

Problem 4.1 Given:

- *D tasks $\{d_1, d_2, \ldots, d_D\}$, where each task represents the evaluation of a device model.*
- *P processors labeled from 1 to P.*
- *N_G processor-groups where the gth group, G_g, is a set of P_g processors. For example, group $\{0,1,2,3\}$ has $P_g = 4$ processors.*
- *N_L levels of processor-groups where the lth level, L_l, is a set of $N_{G,l}$ groups and where each processor $p \in [1, P]$ appears in no more than one group at each level.*

- *A function $T(d_n, g)$ that models the amount of time to execute task d_n on the processors in group g. Models for non-uniform device evaluation time and imperfect speedup on multiprocessor groups are incorporated in this function.*

Find: *a mapping function M that assigns each device d_n to group number $M(d_n)$ such that the estimated time to execute all the tasks on the P processors is minimized. The total time taken is calculated using:*

$$T = \sum_{l=1}^{N_L} \max_{G_g} T_{G_g} : G_g \in L_l$$

where the time for group G_g is computed as:

$$T_{G_g} = \sum_{d_n : M(d_n)=g} T(d_n, g)$$

The total time taken is simply the sum over all levels of the maximum time taken by any group in that level.

Two special cases of this problem have already been encountered in Section 4.6.1. Both cases assume that the number of levels N_L is equal to 1 and each group contains 1 processor. That is to say, they both deal with a one-level partitioning problem. In the first case, the time per task is assumed uniform or $T(d_n, g) = K$. In the second case, the time per task varies from device to device, which results in the most general one-level partitioning problem.

4.9.2 Solution Methods

Finding an exact solution to Problem 4.1 in the general case is an essentially impossible proposition. There are $(N_G)^D$ possible permutations to consider so an exhaustive search would take exponential time in the number of devices to perform. In fact this problem is NP-complete [GARE79], which means no polynomial time algorithm to find an exact solution is ever likely to be found. NP-completeness is demonstrated by noting that the general one-level partitioning problem, which is known to be NP-complete from [GARE79], is a special case of the multi-level partitioning problem. Given this, alternative heuristic methods must be used to obtain near optimum solutions.

Even though the general problem is very hard to solve, at least one special case has a trivial solution. For example, the unit time per task, one-level problem can be solved exactly by assigning devices to the processors in round-robin fashion so that each processor receives at most $\lceil \frac{D}{P} \rceil$ devices. This leads to the speedup result in Equation 4.5. The round-robin algorithm has practical significance since real circuits often fit this model reasonably well.

The best known heuristic for partitioning problems is the Kernighan-Lin algorithm [KERN70] which is used to bisect the nodes of a graph such that the number of edges connecting the two halves is minimized. Extensions that involve multi-way partitions and non-uniform node sizes are also discussed in [KERN70]. However, this algorithm is difficult to extend to the current situation because the node/task sizes can vary over a wide range of values and because the cost of a task depends on which group it is assigned to. An alternative approach to partitioning based on stochastic methods is known as simulated annealing [KIRK83]. In this approach a partition is randomly changed and the impact of the change on the partition's cost is assessed. If the cost is decreased, the change is always accepted. If the cost increases, the change is accepted with a probability that depends on how far the algorithm has progressed. Initially the probability is relatively high, so most changes are accepted; later, the probability is decreased. The acceptance probability, P_A, is determined using the following equation:

$$P_A = \exp(\frac{-\Delta C}{T}) \tag{4.7}$$

where ΔC is the cost increase and T is a *temperature* parameter that is slowly decreased over time. Simulated annealing thus has four main components: a concise description of a problem configuration, a set of moves to apply randomly to change the configuration, a cost function that assesses the quality of the configuration, and an annealing schedule that determines how the temperature should be varied and how many random moves should be made at each temperature. For the current problem, the main advantages of simulated annealing are the flexibility that it allows in defining the cost function, and the ease with which it can be adapted to new problems. For example, realistic estimates of the time per iteration for each device in the circuit can be used when constructing the cost function. However, the main drawback is that it is difficult to determine a good set of moves to make and a good annealing schedule. As a result, implementations of simulated annealing generally require considerable tuning

before good results can be obtained consistently.

4.9.3 Trial Implementation

For this work, an experimental implementation of simulated annealing for the multi-level partitioning problem has been implemented based on the generic approach described in [JOHN89]. For information on obtaining the source code to this implementation refer to Appendix E. Since the results from this implementation are inconsistent, the details are only sketched out here. The implementation needs to be tuned and improved before it can be incorporated in a working mixed-level simulator. The configuration manipulated by the simulated annealer is the mapping function M. A move consists of changing the mapping by reassigning a device to a new group. Several strategies for selecting the device to be moved and its new group have been tried. In addition, the annealing schedule has been varied to improve the performance. The cost function attempts to model the execution time of the device evaluation realistically. For simple circuit elements and compactly modeled devices, a constant, independent of the group size, is used which depends on the model complexity. For numerical devices, the execution-time models developed in Chapter 3 are used. To estimate the potential of parallel device simulation, a speedup model is incorporated based on the results in [LUCA87b]. The time to execute a task on a multiprocessor group is then given by:

$$T(d_n, g) = T^{est}(E_n, |G_g|) = T^{meas}(E_n)/S^{est}(E_n, |G_g|) \tag{4.8}$$

where T^{est} is the estimation function, E_n is the number of equations for device d_n, $|G_g|$ is the number of processors in gth group, T^{meas} is an execution-time model calibrated with uniprocessor measurements, and S^{est} is the estimated speedup which depends on both the number of device equations and the number of processors. In a true implementation of the multi-level model-evaluation algorithm, the speedup model should be calibrated using measurements of the parallel device simulator's performance. The final cost function is computed by taking the estimated total execution time and normalizing with the cost of running the tasks on a uniprocessor machine.

The main problem with the current implementation is that it fails to find obvious optimum solutions when confronted with simple problems. For example, when equal time tasks are used and the number of tasks is evenly divisible by the number of

processors, the annealer finds solutions that assign some of the devices to multiprocessor groups even though the obvious solution is to evenly divide the devices among the one-processor groups. In addition, because the algorithm is probabilistic it does not reach the same solution when started from different points. As a consequence, better results can often be obtained by running the annealer multiple times and selecting the best result encountered. This process would need to be automated before the annealer could be incorporated as part of a parallel mixed-level simulator.

Despite the problems of the simulated annealer, some insight into the underlying problem can be gained by examining the solutions obtained. First, in cases where different types of devices (numerical and compact) are mixed in the same circuit, the annealer places the numerical devices as best as it can and then places all of the compact devices on the least loaded processor. Thus, one optimization might be to ignore the compact devices during annealing and fix them in place on one processor beforehand. Second, for small circuits and when the number of devices is not evenly divisible by the number of processors, the best performance is achieved by first assigning devices evenly to the one-processor groups. Then the remaining devices are assigned evenly to the next level of groups, and so on until no devices remain. For example, with 9 devices and 8 processors, 8 devices will be assigned to the one-processor groups and 1 device will be assigned to the 8-processor group. The higher parallel efficiency of smaller groups of processors accounts for this behavior. Third, another effect of having non-ideal speedup of device evaluation on multiprocessor groups is that the annealer will avoid using extra processors if the extra cost of communication overhead increases the execution time for the task. In the above example, if the leftover device obtains no speedup on the 8-processor group, the annealer will place it in a 4, 2 or even 1 processor group instead.

The conclusion to be drawn in this section is that there is no simple solution to the multi-level partitioning problem. However, in light of the fact that a multi-level partitioner is not needed until a parallel device simulator is also available, this is not currently a major concern. In the next chapter, a mixed-level simulator implementing the one-level model-evaluation algorithm is introduced that successfully employs a simple round-robin scheduler to achieve reasonable speedups.

4.10 Summary

The computational bottleneck of mixed-level simulation has been addressed by investigating the possibilities for the use of scalable, high-performance distributed-memory multicomputers. Three levels of parallelism are identified that can be exploited by the multiple processors in such a system. At the design-level, tasks consist of individual simulation jobs. At the circuit-level, the major tasks are the evaluations of the numerically modeled elements of a circuit. At the device-level, each processor is assigned a task that roughly corresponds to a portion of the semiconductor device being simulated. Existing techniques for exploiting each of these levels of parallelism have been reviewed, and extensions that combine parallelism from more than one level have been introduced. In particular, an algorithm is proposed for combining parallelism at the circuit and device levels in a single program. This algorithm employs the concept of dividing a multiprocessor machine into multiple levels of processor groups. Tasks are then assigned to these groups rather than to individual processors. This approach adds a dimension of flexibility that can be used to achieve greater speedup than a task-per-processor approach. Experiments with a simulated-annealing-based partitioning program indicate that in some cases additional speedups may indeed be achievable compared to a simpler implementation based solely on exploiting parallelism at the circuit level. However, the proposed algorithm requires several software components that are not readily available on present parallel computing systems.

```
Read and parse the input file.
Generate the circuit data structure, the lists of devices
    and the task/job structure.
Assign device instances to the processor groups.
Foreach (processor group level) {
  Foreach (device in my group at this level) {
    Setup data structures.
  }
}
Establish a DC operating point.
Foreach (timepoint) {
  Foreach (iteration) {
    Foreach (processor group level) {
      If (I am group leader) {
        Calculate and stamp currents and conductances onto
            my local Matrix and RHS for each normal element.
      }
      Foreach (numerical device) {
        Coordinate with other group members to solve device equations
            and calculate currents and conductances.
        If (I am group leader) Stamp contributions onto Matrix and RHS.
      }
    }
    Combine local Matrix and RHS to get global Matrix and RHS.
    Factor and solve circuit-level equations.
    Check locally for convergence.
    Exchange convergence information with other processors.
    If (Convergence reached) {
      If (I am machine leader) {
        Save the current circuit solution.
      }
      Foreach (processor group level) {
        If (I am group leader) Save numerical device internal states.
      }
      Go to Next timepoint.
    }
Next timepoint:
    Foreach (processor group level) {
      If (I am group leader) {
        Calculate compact device instance truncation errors.
      }
      Foreach (numerical device) {
        Coordinate with group members to compute truncation error
            and maximum timestep for this device.
        If (I am group leader) Update local maximum allowed timestep.
      }
    }
    Find minimum allowed timestep across all processors.
  }
}
```

Figure 4.10: Description of proposed algorithm

Chapter 5

Implementation on Distributed-Memory Multicomputers

5.1 Overview

In Section 4.8, two algorithms for mixed-level simulation on a distributed memory multicomputer are outlined. In this chapter, two implementations of one of those algorithms, the one-level model-evaluation algorithm, on different distributed memory multicomputers are described. The first system considered is an Intel iPSC/860 hypercube, a scalable high-performance computer with a specially designed node architecture and communication system. The second system is a cluster of standard engineering workstations communicating via Ethernet connections. Both of these types of systems are candidates for meeting the processing needs of mixed-level simulation in an IC design environment.

Each system is described from both hardware and software points of view, and basic performance measures are provided. The iPSC/860 is described first, and the necessary modifications to the mixed-level simulator CIDER to support parallel processing are presented. Global reduction of the circuit matrix is identified as a potential performance bottleneck, and three different alternatives for implementing this step are considered.

The performance of both implementations is investigated in Section 5.5. A set of 17 benchmark circuits is used to test the performance. These circuits all contain more than one numerical device, so that one-level model-evaluation offers at least some hope of performance improvement. Using these benchmarks as examples, the problems associated with one-level numerical-model evaluation are described. Several limitations are identified, and solutions that work around the problems are given where possible.

5.2 Description of the Hypercube

The one-level model-evaluation algorithm for multicomputers described in Section 4.8 has been implemented on an Intel iPSC/860 hypercube. The iPSC is a distributed-memory multicomputer: each compute node has its own address space and data is shared via an explicit message-passing mechanism [BELL92a]. Figure 5.1 is a diagram of the system used for parallel-code development on the hypercube. Serial-

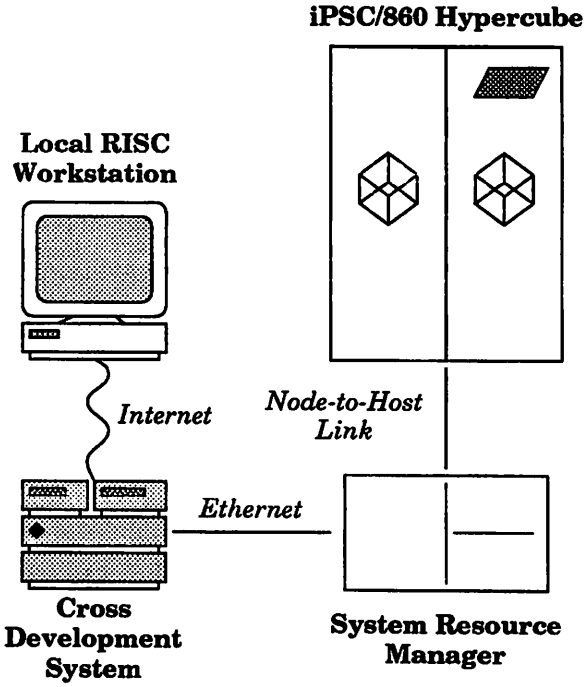


Figure 5.1: Hypercube software development system

code enhancement is performed on the local RISC workstation. The code is then

modified for parallel execution and compiled for the hypercube on the remote cross-development system (XDS). After an initial investment in parallelizing the circuit simulator (SPICE3), updates to the numerical device models can be performed rapidly due to the simplicity and modularity of the algorithm. Simulation jobs are loaded onto the cube by a host computer called the System Resource Manager (SRM), or alternatively by the XDS acting as a remote host. From the SRM or XDS, users can space-share the iPSC by allocating sets of nodes called *cubes* for individual problems. Cubes are actually full smaller-dimensional hypercubes, so that code developed using small cubes can easily be scaled up to larger systems.

5.2.1 Architecture of the iPSC/860

The iPSC/860 hypercube is the third generation of multicomputing systems manufactured by Intel. At the time of its introduction in 1990, the iPSC/860 was the fastest scalable high-performance computer in the world. While previous generations were based on Intel's x86 architecture CPUs, the iPSC/860 uses processing nodes based on the i860XR CPU. The i860XR is a 40 MHz RISC microprocessor fabricated using a 1.0 μm CMOS technology, specially developed for high-performance computing applications. Each compute node can support up to 16MB of physical memory. Both program and data must fit into this space. Since virtual memory is not supported, this represents a hard limit on the size of applications.

The processing nodes use a hypercube-connected communication network to pass messages among themselves. A hypercube of dimension d contains $P = 2^d$ processors and each processor connects to each of its d nearest neighbors. Two nodes are neighbors if their node addresses differ by a single bit when expressed as binary numbers. A message can be passed from any node to any other node in at most d hops, the number of hops needed being equal to the number of differing address bits. As a result, many standard operations that access all of the nodes run in $O(\log P)$ time.

Example: Figure 5.2 shows a 4-d hypercube network. Nodes 3 (0011) and 11 (1011) are connected since their addresses differ in the fourth bit. Nodes 0 (0000) and 15 (1111) are 4 hops apart, since all 4 address bits differ. ■

The network hardware used is the same as that used in the previous generation iPSC/2 system. As is shown later, this results in a large imbalance between the

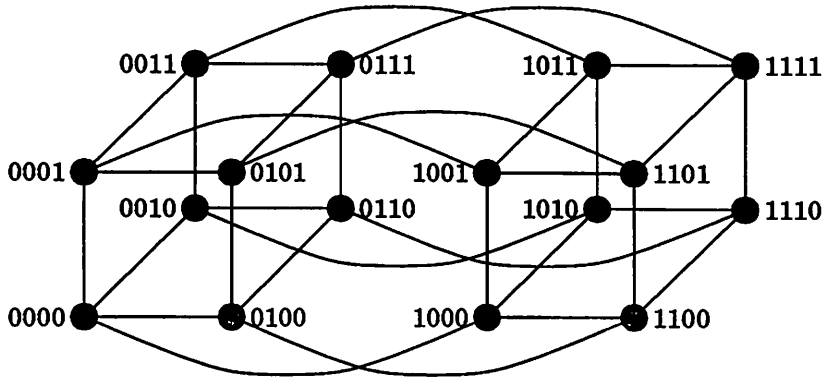


Figure 5.2: Four-dimensional hypercube

compute and communicate speeds of the system. Seven of the eight 2.75 MB/s asynchronous, bidirectional channels in a node's Direct-Connect Module (DCM) are used to connect up to 128 nodes in a 7-d hypercube. However, smaller systems are available that use only some of these links. Unless a node is actually sending or receiving a message, it is free to continuing computing while the DCM performs all the necessary message routing functions. The eighth connection is used to attach compute nodes to optional i386-based I/O nodes. These provide access to a large-capacity Concurrent File System (CFS) and an Ethernet network. The CFS can provide fast I/O for storage of output results and the Ethernet connection can be used for interactive graphics. However, neither of these features is particularly useful in the current situation, since data analysis and visualization take place on the local workstation which is isolated from the iPSC/860 by the Internet. Also, since files stored in the CFS are not directly accessible from the SRM, a special shell (*nsh*), which runs only on the nodes, must be started to gather the results and return them to the SRM. As a result, the actual time needed to access the results may not be improved greatly by using the CFS.

5.2.2 iPSC Software Environment

There are two main categories of software supplied for the iPSC/860: development software and cube-management software. These are in addition to the UNIX operating system that runs on the SRM or the XDS.

Development software consists of compilers, system libraries, debuggers and performance analysis tools. Since CIDER is written completely in the C programming

language, only the *icc* C compiler was needed for this work. The system libraries contain the standard C libraries with extensions for message-passing, parallel file access and graphics, and for controlling and finding out about the user's cube. The NX/2 operating system is a small-kernel OS that runs on each compute node. It provides the services contained in the system libraries. The NX/2 kernel is deliberately kept small since it must share the node physical memory with the application. As a result, some of the features available in a standard UNIX environment are not available on the nodes. The message-passing library contains routines for both point-to-point communication as well as global operations that all processors participate in. The global reduction routines are the primary communication mechanism used in CIDER. Efficient routines are available for both integer and floating-point vectors to find the global minimum and maximum, and for adding and multiplying the vector elements. In each case, the results are automatically distributed to all processors via a broadcast at the end of the routine. Because the one-level model-evaluation algorithm is so simple, debugging of the parallel version is possible without the need for a parallel debugger. Instead, standard debugging techniques such as insertion of diagnostic output calls and comparing the output from a test run to a reference copy are employed. Application tuning has also been performed simply by making program modifications and observing the effect on the overall run time and the time taken by certain critical sections.

Cube-management software consists of extensions to the SRM or XDS operating system. Since NX/2 only supports a single process running on each node, the cube-management software provides the commands needed to space-share the iPSC. Because these commands are layered on top of the host OS, running jobs on the iPSC is more complicated than users are typically accustomed to. Generally, the user must intervene to allocate a cube, run the job and then release the cube. If the job hangs, the user must issue a kill command to the processes on his cube, which automatically releases the cube as well. Since this can quickly become tedious, a C-shell script was originally written to perform these actions automatically for CIDER. Use of a script simplifies the task of running jobs for users who are unfamiliar with the details of the space-sharing mechanism. For example, additional resource management facilities are provided by the NQS network queuing system that is supplied with the iPSC [IPS92a]. A more sophisticated version of the script that submits jobs to the NQS

system has recently been written. However, from the user's point of view the job submission process remains substantially unchanged.

5.3 Description of the Workstation Cluster

A typical IC design environment is likely to contain a number of engineering workstations which are used primarily during normal working hours for computer-aided design. These workstations represent a computational resource that is under-utilized on nights and over weekends. By harnessing the power of otherwise idle workstations, speedups can be achieved at essentially no additional cost to the organization.

The potential of this approach for the current problem has been investigated by implementing one-level model evaluation on a cluster of DEC workstations connected together via an Ethernet communications network. A portable message-passing package implemented on top of the basic operating system is used to provide high-level operations equivalent to those available on the iPSC. However, the resulting parallel system, which conforms to the distributed-memory multicomputer model, is not scalable. The network has a fixed bandwidth that is approximately three times lower than the node-to-node bandwidth of the iPSC. In addition, the network is shared by all computers on the network, even those not actively participating in the parallel computation. As a result, to avoid disturbing normal network operation and to prevent communication from becoming the performance bottleneck, every effort must be made to minimize communication in this environment.

5.3.1 Layered Distributed Computing Systems

Many systems have been developed for distributing computation across a network. In this work, attention was restricted to systems that satisfied the following criteria:

Publically Available The source code is readily available for free via the Internet.

Portable The package has been ported to a wide variety of serial as well as parallel platforms.

Numerically Oriented The system was designed for use in large, numerically intensive computations.

The above requirements are motivated by the desires that the resulting implementation be available to as wide an audience as possible and that it be relatively easy to support both a cluster and a hypercube implementation. Three different systems satisfying these criteria were obtained: P4 [BUTL92] and TCGMSG [HARR91] developed by researchers at Argonne National Laboratory and PVM [SUND90] originating from Oak Ridge National Laboratory. Both P4 and TCGMSG are successors to the earlier PARMACS [BOYL87] portable parallel-programming macros. Each system provides a low-level message-passing application-programmer interface to allow the individual computers to communicate. However, only the first two have built-in operations for global reduction, so PVM is at a disadvantage in this respect. Since the primary goal is to develop a working prototype quickly, no attempt has been made to obtain comparative performance results for each system to determine which system is best. In an Ethernet environment, more fundamental limits are imposed by the network, so an efficient implementation of the message-passing software is not of major concern. As a result no claim is made that the system chosen is in general the best system to use for distributed computing, but only that it is best for this application.

After installing and working with each system on test programs it has been decided that the TCGMSG package is the simplest to use, to modify if needed, and to support on both the iPSC and the workstation cluster. In part this is because TCGMSG is basically an abstraction of the message passing interface of the iPSC. A thin layer of software allows the native iPSC message-passing routines to be used in the iPSC port of TCGMSG. The extra layer of software adds a small amount of time to the overhead when starting a new message. No noticeable deterioration in performance has been detected when using the TCGMSG interface instead of the native routines in the iPSC implementation of the one-level model-evaluation algorithm.

5.3.2 Network Hardware Environment

The iPSC/860 hardware is composed of a few well-defined components; variation is restricted primarily to the number of nodes in the system and the amount of memory installed per node. In contrast, there are limitless variations possible in a

Machine	iPSC/860	DEC cluster
Number of Nodes	32	60
Node CPU	i860 XR	MIPS R3000/3010
Clock Rate	40 MHz	25 MHz
Memory/Node	16MB	32MB
Nominal Data Rate	2.8 MB/s	1.25 MB/s
Connectivity	Hypercube	Bus
Node OS	NX/2 3.3.2	Ultrix 4.2a
C Compiler	PGC Sun4/4.0 2.0a	MIPS 2.1
Optimization Level	-O2	-O2

Table 5.1: Comparison of Parallel Machine Configurations

distributed computing environment. Node instruction-set architectures, clock speeds and operating systems can all vary from one machine to another. In addition, the network bandwidth, which is very important in distributed computing, can vary by an order of magnitude or more from one installation to another. This sort of heterogeneous computing environment makes distributed computing challenging since care must be taken when using different types of machines that the information passed does not become corrupted. Taking this into consideration, it is important whenever discussing performance to identify as well as possible the system being tested.

In this work, the cluster consists entirely of DECstation 5000/125 workstations connected by a 10Mb/s Ethernet. Although other computers are available on this net and TCGMSG does support heterogeneous computing, it has been decided that debugging, performance evaluation and load balancing are greatly simplified if the cluster is restricted to a homogeneous configuration. Distributed computation is enabled by layering the TCGMSG message passing system on top of the native Ultrix 4.2a operating system of the DECstation. Table 5.3.2 shows a comparison between the iPSC/860 and the DEC cluster configurations.

The nodes of these two configurations have been included in the CIDER serial benchmark tests of Chapter 3. Those tests establish the computational performance of the nodes. The general communication performance of the iPSC has been reported in [DUNI91]. However, the specific performance for global reduction is not included there. A test program has been used to specifically exercise double-precision global reduction when adding together vectors of various lengths. This program has been run on both the iPSC and the DEC cluster in order to allow comparison between the two.

In both cases, measurements are made using wall clock time. On the DEC cluster, this causes the times to vary significantly from one run to another due to interference from the other machines on the network. However, this seems to be the fairest way to measure time in such an environment, since idle time can account for a significant portion of the runtime in mixed-level simulations for machines that are assigned fewer or less computationally demanding numerical devices. If CPU time were to be used instead, the idle time would not be reported on the lightly loaded machines.

Figure 5.3 shows the time taken for global reduction on the iPSC for cube sizes of 2, 4, 8, 16 and 32 processors and vectors up to 2000 entries in length. The

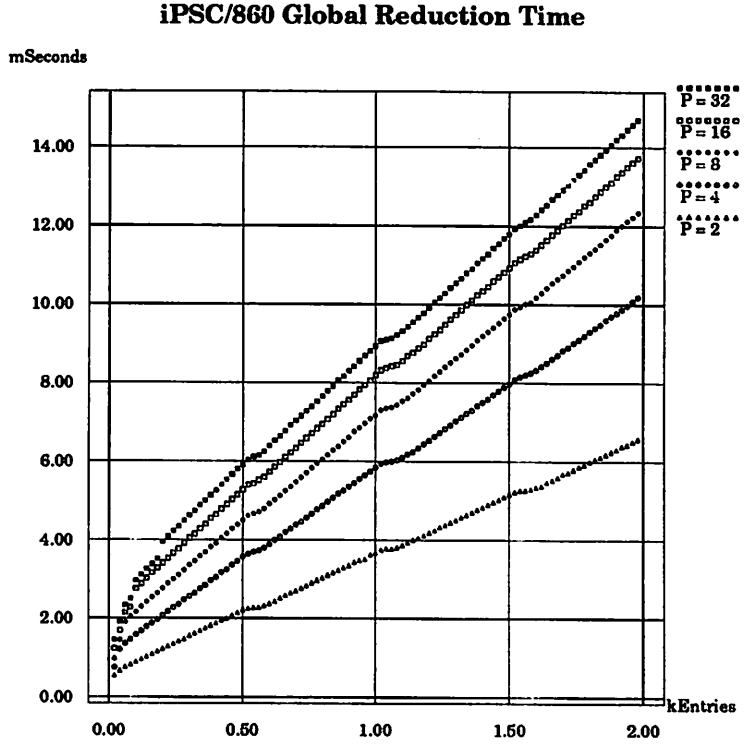


Figure 5.3: Global reduction execution time on the iPSC/860 for different numbers of processors and vector lengths

maximum time is just under 15 milliseconds. In Figure 5.4 the time taken on the DEC cluster under the same conditions is given. Five runs have been performed to accumulate enough data that statistical variation is observed. Here the maximum time is measured in seconds. As the number of processors is increased, the likelihood of interference also increases, so the data are more scattered for $P = 16$ and $P = 32$.

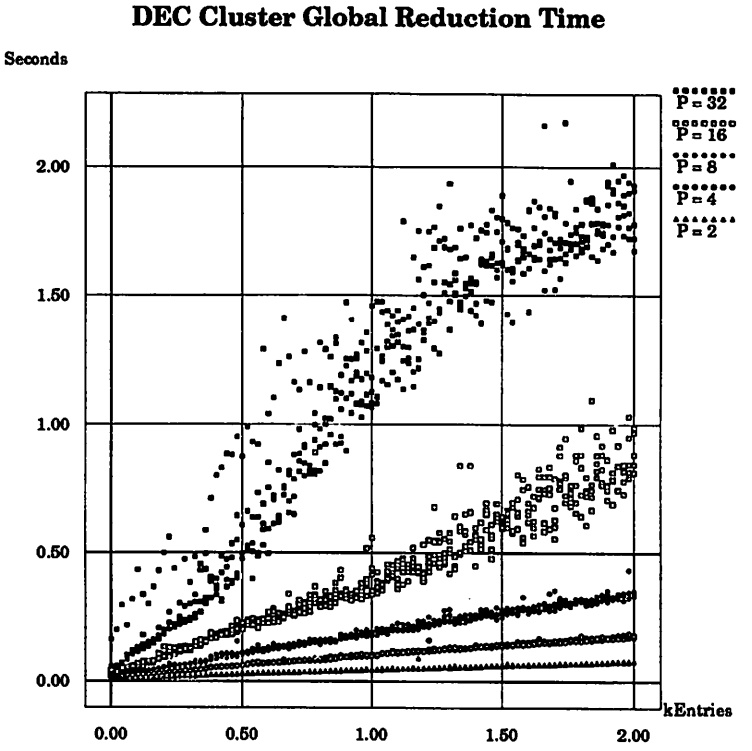


Figure 5.4: Global reduction execution time on the DEC cluster for different numbers of processors and vector lengths

Except for the 32 processor DEC cluster and for the larger iPSC cubes operating on short vectors, the execution time is reasonably modeled with an equation of the form:

$$T = \alpha + \beta L \tag{5.1}$$

where α, β are parameters that depend on the number of processors and L is the length of the vectors. It is expected that both α and β should grow as $O(\log P)$ since that is how many message-passing steps are needed for global reduction. Table 5.2 shows the parameters obtained by fitting the previous expression to the data in the two figures. For the DEC cluster, the minimum time taken at each vector length was used in order to obtain a lower-bound estimate for the time. Theoretically, there is no upper bound since any machine in the cluster may be arbitrarily loaded down during the course of the reduction operation. These extracted coefficients quantify what is already apparent from the figures: that the iPSC time grows slowly as additional processors are added, whereas the cluster time grows almost linearly with the number of processors. This is

# of Processors	Number of Processors			
	iPSC/860		DEC Cluster	
	α	β	α	β
2	0.6 ms	3.0 μs	5 ms	36 μs
4	1.2 ms	4.5 μs	11 ms	83 μs
8	1.7 ms	5.4 μs	16 ms	158 μs
16	2.2 ms	5.8 μs	26 ms	352 μs
32	2.6 ms	6.1 μs	46 ms	940 μs

Table 5.2: Extracted global-reduction-time coefficients for different numbers of processors and machine architectures

due to competition among the cluster nodes for access to the Ethernet. Both machines have high message startup times in the millisecond range. For a typical vector length of 500, the models predict that a 16-node iPSC subcube would take about 5 milliseconds while the DEC cluster would take around 0.2 seconds, a factor of 40 higher. This indicates that communication performance is a more important factor in determining overall performance on the DEC cluster.

5.4 Implementing Parallel Model Evaluation

The implementation of one-level model evaluation is based on the SPMD programming model[1]. Each compute node runs the CIDER executable, but is responsible for a different set of circuit elements. The implementation is hostless; there is no need to run a control process on the SRM of the iPSC to execute the non-parallel sections of the code. Nodes exchange information when necessary and duplicate tasks so that certain key data structures such as the circuit sparse matrix remain consistent across all processors.

The modifications to the source code needed to parallelize CIDER are very minor. One additional member is added to the circuit-element data structure that stores the address of the node that owns that element. The owner node is responsible for all updates to the circuit matrix for the element. The owner field is used to bypass

[1]The description that follows is oriented towards the iPSC implementation. On the workstation cluster, compute node refers to one of the computers being used and the host and host process should be interpreted as references to the user's own workstation and a separate process running there.

elements that are not owned by a node. Figure 5.5 shows the main loop of the resistor

```
{
    register RESmodel *model = (RESmodel *)inModel;
    register RESinstance *here;

    /*  loop through all the resistor models */
    for( ; model != NULL; model = model->RESnextModel ) {

        /* loop through all the instances of the model */
        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {
            if (here->RESowner != ARCHme) continue;

            *(here->RESposPosptr) += here->RESconduct;
            *(here->RESnegNegptr) += here->RESconduct;
            *(here->RESposNegptr) -= here->RESconduct;
            *(here->RESnegPosptr) -= here->RESconduct;
        }
    }
    return(OK);
}
```

Figure 5.5: Main loop of resistor loading code

loading code demonstrating this bypass step. The line indicated is the *only* line needed to parallelize this particular loop. Similar small changes are needed in the other files associated with a particular type of device, amounting to a total of less than 20 lines of code for each type of device. Since certain information about the state of a device is needed only by that device, memory for this information is only allocated by the owner node. This can provide a significant savings in memory usage when numerical devices are used, since the device-dependent information for a numerical device is significant. For example, the L/U factors of a device-level matrix are include in this category. The disadvantage of this approach is that dynamic load balancing is essentially useless in this system, because the cost of transmitting all this information to another processor is prohibitive.

Although the changes to the model libraries of CIDER are minimal, this is less true for the main simulation driving routines. Trivial changes are needed to the frontend to initialize the architecture variables (node id and size), open various output files (one diagnostic output for each node, one rawfile for the simulation output) and

ensure that the simulator output file is only written by one node.

The more interesting changes are in the simulator core. The first core routine encountered in an analysis is the circuit setup routine. All processors receive a copy of the input file which they use to setup a circuit's data structures. Once this is done, the devices are divided among the processors by the partitioner. It is impossible to automatically parallelize this step at compile time because the number of circuit elements depends on the circuit being simulated. The iPSC implementation uses a much simpler approach to partitioning than the one described in Section 4.9. Using a loop, circuit elements are assigned to the processing nodes in round-robin fashion. This has the effect of evenly distributing the numerical devices among the nodes as best as is possible since they are all grouped together at the end of the loop. Although simple, this approach seems to work fairly well in practice. However, it has obvious limitations that are exposed in Section 5.5.4.

Because there is no process running on the host computer available to execute the serial sections of code, one or more compute nodes must be used to do this while the remaining nodes sit idle. Two options for this problem have been considered: run the serial code on one processor and broadcast the results as needed, or alternatively, run the serial code on all processors and make sure that all data structures accessed by a serial section of code are identical at the beginning of the section. The second approach is the more attractive of the two. Its disadvantage is that it takes roughly twice as long to gather data together since extra communication is needed to broadcast the necessary data. However, since this functionality is built in to the iPSC global reduction routines, separate less efficient routines would have to be written to avoid this step. One advantage is that less communication may be needed later in the algorithm. For example, if solution of the circuit sparse matrix is done on one processor, a broadcast is needed to send that solution to the rest of the processors which need it so that devices can properly load the circuit matrix on the next iteration. Another advantage is that it is easier to program because each node is free to execute any section of code without risk that the necessary data is not present. This allows the elimination of tests that would be need to be scattered through the code to prevent the use of uninitialized or incorrect variable values.

5.4.1 Global Combining

The decision to use duplication of serial tasks on all processors is based strongly on the availability of efficient global reduction operations on the iPSC. These operations are needed at three points during the circuit Newton-Raphson loop: after each processor computes its entries in the circuit (during CKTload), after each processor checks convergence of the circuit variables (during CKTconvTest), and while computing the next time step in a transient analysis (during CKTtrunc and DCtran[2]). Summing the local circuit matrices is the most difficult of the three; the other two are very simple. Convergence testing requires an integer global sum to find the number of nonconverged devices. Timestep control uses floating-point global-minimum operations to find the minimum allowed timestep.

Figure 5.6 shows the flow of data during this step of the one-level model-evaluation algorithm. After the individual processors load their local matrix copies they are combined via message-passing floating-point global-sum operations into a global matrix that is then broadcast so that each processor has a copy. Each processor then proceeds by computing its own local circuit L/U factors and continuing until it is necessary to test for convergence. In addition to combining the circuit matrix, extra combines are needed to check for errors during the load step (an integer operation) and to combine the RHS vector (a floating-point sum operation). Error checking occurs prior to combining the circuit matrix, while the RHS vector is passed along with the circuit matrix and combined at the same time.

Three different strategies for combining the local circuit matrices have been considered. These strategies differ in the number of communications needed and the amount of memory needed. The amount of memory varies because an extra communication buffer is needed to hold incoming messages prior to actually performing the requested operation. The amount of memory needed is thus proportional to the length of the messages sent. Combining the local matrices is complicated by the fact that the sparse matrix package uses a linked-list data structure for storing and accessing the matrix elements. Since global reduction operates on an array of values, the matrix elements must be copied out of the matrix data structure into a buffer. Only the original non-zero elements need to be copied, since the fillin values are all zero prior

[2]DCtran is the SPICE3 subroutine that contains the transient analysis driving loop.

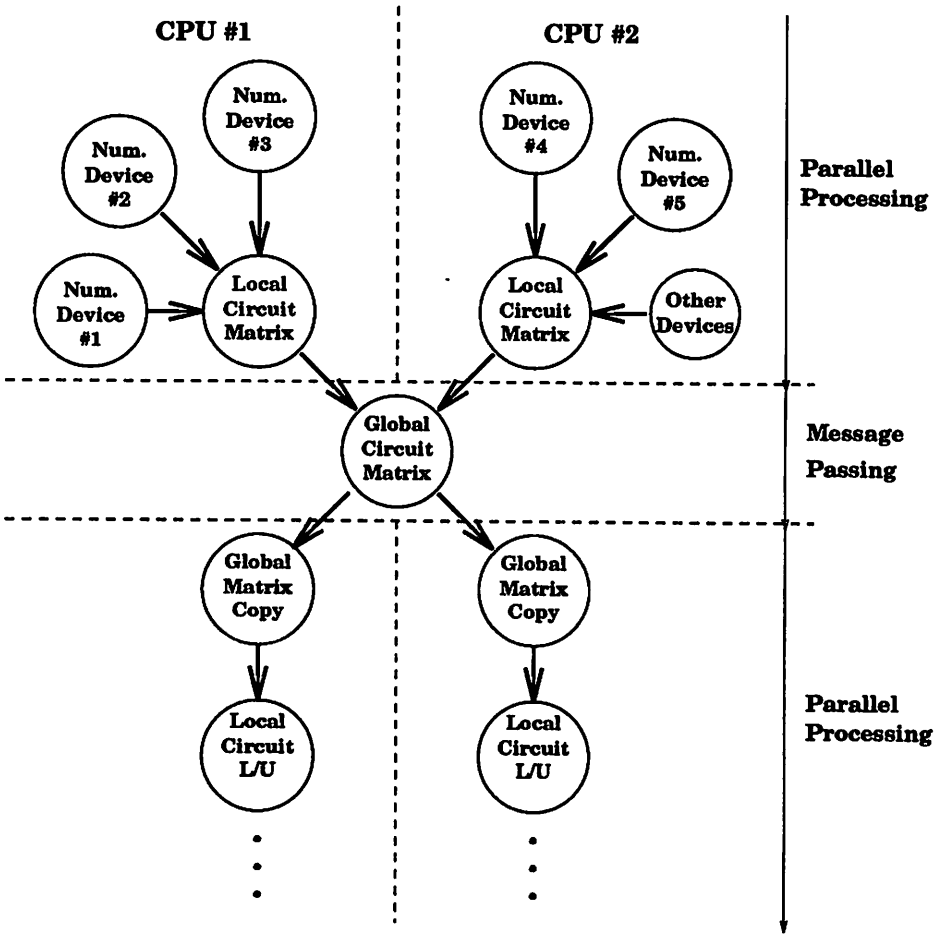


Figure 5.6: Flow of data during CKTload and CKTsolve

to factoring the matrix. However, the existing data structures in the serial version of SPARSE do not allow easy identification of the original sparse-matrix elements. By extending the data structures with a list that keeps track of these elements, the parallel version allows the original non-zeroes to be marked during a preprocessing step by scanning the list. Only the marked entries are copied into the buffer. The row index of an element is used as a marker flag, thereby eliminating the need for extra memory to hold it. The row index is restored in a subsequent post-processing step when the data is copied back into the matrix from the buffer. To avoid the gathering step entirely, one might consider initially allocating the matrix elements from a common pool that is already stored as an array. However, this is not feasible in the current implementation because the matrix element data structure contains other members besides the

actual data. Although it would be possible to change the data structure, the necessary modifications would permeate the sparse matrix package and would likely degrade the serial performance. Alternatively, more sophisticated global-combine operations that use a non-unit stride as they scan the array would make it possible to skip the intervening data.

The three approaches considered differ in the amount of the sparse matrix buffered before the matrix is combined. Each procedure makes three passes through the entire matrix: one to mark the original non-zeroes, one to buffer the elements before combining and one to unbuffer afterwards[3]. The methods are:

Row-by-Row One row (or column) of the sparse matrix is buffered for each combine operation.

All-Rows The entire matrix is buffered in one step before it is combined.

Fixed-Length A buffer that can store a fixed number of elements is set aside initially. Elements are added to the buffer until it becomes full or there are no elements left. The buffer is then combined.

The row-by-row method is the simplest to implement and requires no extra memory since the circuit solution vector and an intermediate vector can be used for the element and communication buffers. The all-rows approach is also straightforward to implement, but requires twice the memory needed to store the matrix data since no suitable element and communication buffers are available. This could be a problem if a large circuit matrix is encountered. The fixed-length approach is the most difficult to implement because it is necessary to periodically interrupt the buffering process, possibly in the middle of a row, to combine and unbuffer. The amount of memory used can be varied over a wide range. Two possibilities that limit this quantity are to use the same buffers as used in the row-by-row approach or to allocate fixed length buffers at compile time. Both approaches handle the occurrence of a large matrix gracefully.

The volume of communication is equal in all three cases to the number of original non-zero elements in the matrix. If the overhead to send a message were

[3]An attempt to save time by combining the first two passes by buffering the non-zeroes in an order unrelated to their location in the matrix produced incorrect results. Further investigation is needed to clarify the source of this problem, but for the present, no real harm is done because the time for the additional pass is not significant compared to the communication time.

negligible, they would all take time proportional to the number of elements. However, message startup time on the iPSC is significant, and the approach that sends the fewest messages runs fastest. Since the all-rows approach uses the minimum of one combine operation, it is guaranteed to send the fewest messages as well. The row-by-row approach uses one combine for each row in the matrix, and each operation is very short (< 10 elements per row is typical of sparse matrices), the message overhead is very high and this approach has the poorest performance. The fixed-length approach uses a number of combine operations equal to $\left\lceil \frac{N}{L} \right\rceil$, where N is the number of matrix elements and L is the length of the buffer. If L is greater than N, only one combine is needed in the fixed-length approach and its performance is equivalent to the all-rows approach. If L does not scale with problem size, as the matrix size grows, more and more combine operations will be needed. However, the performance will not degrade significantly if L is chosen so that the message startup time is a small fraction of the total message time. If L is scaled with problem size by using spare RHS vectors, the number of combine operations will be equal to the average number of elements per row, which should be relatively constant as problem size grows. (This assumes that fillin elements are skipped. If fillin elements are also combined, the average number of elements per row will grow with problem size.) The obvious disadvantage of this approach is that the common case of a small mixed-level circuit matrix will need multiple messages rather than a single one if a longer buffer were used.

Based on the above considerations and the results of tests of implementations of all three methods, the fixed buffer approach is used in CIDER. Its main advantage is that it minimizes the number of messages sent compared to the row-by-row approach. This is important on both the iPSC and the DEC cluster since message startup time is not negligible, as shown in Section 5.3.2.

5.4.2 An Alternative Programming Approach

The one-level model-evaluation algorithm lends itself naturally to a client-server (or host-node) view of the computation [MEIN90]. The circuit simulator is the client and the device evaluator (which includes or communicates with the device simulator) is the server. On the iPSC, the host program typically would run on the SRM while the node program runs on the compute nodes. The host-node model is not used

on the iPSC for two reasons. First, two separate programs need to be maintained in the host-node approach. This increases both initial development costs and later maintenance costs. Special interfaces must be written to pass information such as model parameters between the host and node programs. These interfaces are complicated by the fact that SPICE3 relies heavily on pointer-based data structures which are difficult to pass in messages[4]. Also, communication must take place across the host-to-node link which has limited bandwidth [DUNI91]. This could become a performance bottleneck if many nodes are trying to communicate with the host simultaneously. This would also be true in a workstation cluster.

The second reason that the host-node model is not used is peculiar to the design of SPICE3. It has been mentioned earlier that state information is saved for many devices in the circuit. The primary examples of this are the previous solutions at the circuit and device levels that are needed during a transient analysis. For SPICE3 which was designed to simulate large circuits, it would be inefficient to let each device exchange its few state variables each time a new timestep was accepted. Thus, this variable state information is collected into one long array and each device stores the indices of its entries. The simulator core is then able to allocate, deallocate and swap state information for the entire circuit simply by dealing with the state array as a whole. In effect, information that could be hidden in each device is exposed to the circuit in order to improve efficiency. However, this whole approach is a problem when the circuit and the devices reside on different processors because they must now communicate the state information back and forth, or the simulator core must be modified to allow devices to manage their own states. The second method would need to be restricted to the numerical devices in the circuit where the overhead would be small. In that case, the compactly modeled devices would be assigned directly to the host. Unfortunately, either method would require more extensive modification of the program to implement.

The above two concerns are not as important when linking separate stand-alone circuit and device simulators together. In such a situation, the programs are already being maintained and the extra development cost of linking them using the

[4]The main difficulty in passing pointers in messages is that the address spaces of the individual nodes are not shared. Thus, the pointer on one machine is not likely to point to the same item on another node, but is in fact very likely to point to something random. Special care must then be taken when pointer-based structures need to be communicated.

Circuit	# Ckt Elts	# Num Devs (Type)	# Ckt Eqns	# Dev Eqns	Analysis
ASTABLE	8	2 (1D NPN)	9	354	TRAN
BICMPD	9	2 (2D NMOS+NPN)	19	3745	TRAN
BICMPU	6	2 (2D PMOS+NPN)	13	3745	TRAN
CLKFEED	16	3 (2D NMOS)	22	8889	TRAN
CMOSAMP	5	8 (2D CMOS)	14	21312	DC
DBRIDGE	3	4 (1D DIO)	7	2388	TRAN
ECLINV	9	4 (2D NPN)	14	4324	DC
ECPAL	9	4 (2D NPN+PNP)	12	4948	AC
GMAMP	13	5 (2D NMOS+NPN)	11	8571	AC
INVCHAIN	10	4 (1D NPN)	13	708	TRAN
MECLGATE	24	11 (1D NPN)	29	1947	TRAN
LATCH	14	14 (1D NPN)	24	17010	TRAN
PPEF.1D	5	4 (1D NPN+PNP)	16	4854	TRAN
PPEF.2D	5	4 (2D NPN+PNP)	16	4948	TRAN
RINGOSC.1U	58	14 (2D CMOS)	124	37296	TRAN
RINGOSC.2U	58	14 (2D CMOS)	124	12894	TRAN
VCO	10	6 (1D NPN)	9	1062	TRAN

Table 5.3: Parallel benchmark-circuit characteristics

host-node approach is less significant. However, the performance would still be degraded by the node-to-host link.

5.5 Parallel Performance Assessment

In this section the performances of both the iPSC and the DEC cluster implementations are presented. Several limitations of the one-level model-evaluation algorithm are also identified and examined.

5.5.1 The Parallel Benchmark Inputs

A set of 17 circuits is used to test the parallel performance of CIDER. All circuits contain multiple numerical devices so that one-level model evaluation is at least potentially effective in producing performance improvement. In Table 5.3, the characteristics of the circuits and the analyses performed are summarized. A wide range of circuits are represented in the benchmark set. This demonstrates the general

applicability of CIDER to circuit design and is also useful in testing the program[5]. The circuits are drawn from several sources and include digital, analog and nonlinear analog designs. Five circuits are taken from the serial benchmark set, and the input listings are identical to those found in Appendix B. Listings for the remainder are found in Appendix C. In two cases (PPEF,RINGOSC) the same circuit has been run more than once using different models for the numerical devices. The circuits range in size from the smallest, 2 numerical device, 354 device-level equation ASTABLE circuit to the largest, 14 numerical device, 37296 device-level equation RINGOSC.1U circuit.

5.5.2 Results for the IPSC/860

On the iPSC, each circuit has been run on every subcube of the system where adding processors reduces the number of numerical devices per processor. Since the maximum number of numerical devices is 14, the full 32-node hypercube is never needed in these tests. In many cases, the limited memory of a single hypercube node prevents large circuits from being run on small subcubes and this is indicated in the tables of results. In each case, the subcube must be allocated before the run, loaded with the executable, and then deallocated after the run. This time typically takes about 30 seconds, and has been excluded from the measurements. Execution time is measured using wall clock time because it is not possible to obtain the actual CPU time. However, as mentioned in Chapter 3, the CPU time is close to wall clock time because the iPSC nodes only contain one process at a time. Because the individual processors take slightly different amounts of time to complete, the largest processor time is taken as the runtime.

In Table 5.4, several measures of the time taken on the iPSC/860 by the parallel version of CIDER are presented. Each entry contains 4 numbers: the total analysis time followed by the overall speedup, the time for the main analysis only (DC, AC, etc.), and its speedup. For AC and transient analysis, this means the total time includes the time to calculate the initial operating point. For DC analysis, the main analysis is not separable so the entry is omitted. When necessary, the execution time for a single hypercube node is estimated by summing the times spent evaluating the

[5]Several bugs in the serial code were discovered only after running the parallel version on different numbers of processors and obtaining substantially different results.

Circuit	Number of Processors									
	1	2		4		8		16		
ASTABLE	275	168	(1.64)	—		—		—		
	272	166	(1.64)	—		—		—		
BICMPD	6763†	4950	(1.37)	—		—		—		
	6162†	4490	(1.37)	—		—		—		
BICMPU	4600†	3132	(1.47)	—		—		—		
	4038†	2739	(1.47)	—		—		—		
CLKFEED	14781†	—		5423	(2.73)	—		—		
	12554†	—		4585	(2.74)	—		—		
CMOSAMP	22892†	—		—		4497	(5.09)	—		
	—	—		—		—		—		
DBRIDGE	1170	606	(1.93)	369	(3.17)	—		—		
	1156	599	(1.93)	366	(3.16)	—		—		
ECLINV	2903†	1652	(1.76)	975	(2.98)	—		—		
	—	—		—		—		—		
ECPAL	2125†	1072	(1.98)	568	(3.74)	—		—		
	1446†	723	(2.00)	390	(3.71)	—		—		
GMAMP	3565†	—		—		1201	(2.97)	—		
	1853†	—		—		497	(3.73)	—		
INVCHAIN	93	57	(1.63)	41	(2.27)	—		—		
	84	51	(1.65)	33	(2.54)	—		—		
MECLGATE	261	151	(1.73)	95	(2.75)	72	(3.63)	59	(4.42)	
	239	138	(1.73)	85	(2.82)	62	(3.85)	48	(4.98)	
LATCH	5469†	2952	(1.85)	1798	(3.04)	1065	(5.14)	641	(8.53)	
	5312†	2871	(1.85)	1753	(3.03)	1041	(5.10)	624	(8.51)	
PPEF.1D	205	105	(1.95)	57	(3.60)	—		—		
	164	85	(1.93)	46	(3.57)	—		—		
PPEF.2D	3896†	2019	(1.93)	1191	(3.27)	—		—		
	3173†	1650	(1.92)	1001	(3.17)	—		—		
RINGOSC.1U	132043†	—		—		—		11067	(11.93)	
	126625†	—		—		—		10471	(12.09)	
RINGOSC.2U	20370†	—		6270	(3.25)	3397	(6.00)	1869	(10.90)	
	19675†	—		6033	(3.26)	3275	(6.00)	1800	(10.93)	
VCO	502	282	(1.78)	212	(2.37)	129	(3.89)	—		
	495	279	(1.77)	208	(2.38)	126	(3.93)	—		

Table 5.4: Execution time and speedup on the iPSC/860 system. The entries are: total execution time in seconds (overall speedup) and time for the main analysis in seconds (its speedup). Entries marked — could not be run on that cube. Entries marked † are estimated times. See text for details.

numerical models on a larger cube[6]. Since this excludes a small amount of computation these estimates are slightly optimistic for the serial run time, so the actual speedup is slightly higher than reported here.

In all cases, use of one-level model evaluation decreases the overall run time. However, the efficiency and speedup vary from circuit to circuit. In some cases, the speedup is very good (over 1.9 on 2 processors for the PPEF benchmarks, around 12 for the 14 device RINGOSC.1U circuit). In other cases, the speedup is much less substantial (a best improvement of only 37% for the BICMPD example, a speedup of 4.4 on 16 processors for MECLGATE).

5.5.3 Results for the DEC Cluster

On the DEC cluster, different size processor groups are also used to run the benchmarks. However, since the number of processors is not restricted to be a power of two, the minimum number of processors needed is used when scaling the group size. For example, the MECLGATE circuit is run on clusters of 2, 3, 4, 6 and 11 processors. This results, respectively, in at most 6, 4, 3, 2 and 1 devices per processor. The memory per computer in the cluster is higher, so theoretically it should be able to run large circuits on smaller groups than in the hypercube case. However, because the DEC machines are roughly 3 times slower than the iPSC nodes (c.f. Section 3.4) the time taken on a small machine may be infeasibly large. For example, the RINGOSC.1U example takes over 12 hours to complete when running in a 14-DECstation group. Although, it would also fit in main memory on a 7-processor group, it would take over a day to complete one run. For similar reasons it is infeasible to compare run times to those obtained on a single-processor CPU server such as the DECsystem 5000/240. However, execution times for a single workstation have been estimated in the same way as on the iPSC, by summing individual processor model-evaluation times. For simplicity this has been done for all the benchmarks including the ones that fit easily on one workstation.

Another difficulty in benchmarking network applications has already been

[6]The size of the larger cube does not matter since no parallelism is exploited during the individual model evaluations. Thus, comparable results are obtained if any larger cube is used. However, for consistency, the smallest cube that successfully completes the simulation is used to provide the model-evaluation times.

encountered in the context of measuring the communication performance of the cluster. This is the problem of randomly varying run times caused by differing amounts of load on the individual workstations and of network traffic. These variations can significantly increase the run time above that achievable in a quiet environment. To combat this problem, the network tests have been run on nights and weekends. Workstations are selected by a program that chooses the most lightly loaded machines from the available pool of processors. This time can be significant[7]; in some cases it is longer than the time required to perform the actual simulation. In addition, when practical, multiple runs are performed and the lowest time encountered for a single run is reported. Unfortunately, the outcome of this is that the most reliable timings are obtained for the circuits with the poorest parallel performance. However, the timings of the long running examples are sufficiently accurate to give a general idea of the performance being achieved. In Figure 5.7, the results on the DEC cluster for the LATCH example are plotted. The times usually cluster about a minimum value for a given cluster size, but in several cases one or two of the runs take much longer.

In Tables 5.5 and 5.6, the minimum observed times for the parallel version of CIDER running on the DEC cluster are given. The speedup quoted is the best observed for any run, where the speedup is calculated by dividing the measured run time by the estimated serial run time.

Based on these results, the benchmarks can be divided into three groups: those whose best speedup is roughly equal to that obtained on the iPSC (BICMPD, BICMPU, CLKFEED, CMOSAMP, ECLINV, ECPAL, GMAMP, PPEF.2D) , those that speedup, but not as well as on the iPSC (DBRIDGE, LATCH, PPEF.1D, RINGOSC.1U, RINGOSC.2U), and those that start to slow down when more processors are used (ASTABLE, INVCHAIN, MECLGATE, VCO). A look at the characteristics of the circuits in each class confirms the supposition that the performance difference between the iPSC and DEC cluster generally decreases as the per-iteration model-evaluation time increases[8]. This is not surprising since the major difference between the two

[7]It takes about 3 minutes to obtain the CPU load for each of the 40 machines used in these tests. Of the 60 machines available, the other 20 were left out for several reasons, the most common being a request from the machine's primary user.

[8]The one obvious exception is the RINGOSC.1U example, which uses the same models as the other 2^D benchmarks. This may be simply be an artifact of the limited number of runs (2) made for this benchmark. It could also be due to the large number of processors and the long run time which make it more likely that other jobs will interfere with a parallel job, thereby disrupting load balance.

# Proc.	Circuit		
	ASTABLE	BICMPD	BICMPU
1	325†	20417†	14359†
	323†	18956†	12991†
2	409 (0.81)	14590 (1.42)	9711 (1.51)
	405 (0.81)	13483 (1.42)	8784 (1.51)
	DBRIDGE	ECLINV	ECPAL
1	1972†	8747†	5607†
	1956†	—	4000†
2	1140 (1.73)	5145 (1.75)	2834 (1.98)
	1130 (1.73)	—	2003 (2.00)
4	867 (2.28)	3002 (2.91)	1542 (3.69)
	861 (2.27)	—	1096 (3.70)
	CLKFEED	CMOSAMP	GMAMP
1	46450†	68373†	9544†
	40598†	—	5092†
3	17384 (2.67)	—	—
	15140 (2.68)	—	—
5	—	—	3207 (3.01)
	—	—	1359 (3.75)
8	—	13819 (4.95)	—
	—	—	—

Table 5.5: Execution times on the DEC cluster in seconds. The entries are: minimum observed total execution time (best overall speedup) and minimum observed time for the main analysis (best analysis speedup). Entries marked † are estimated times.

# Proc.	Circuit		
	PPEF.1D	PPEF.2D	
1	367†	12051†	
	318†	10322†	
2	199 (1.85)	6300 (1.91)	
	174 (1.84)	5781 (1.92)	
4	122 (3.04)	3753 (3.21)	
	107 (2.99)	3279 (3.15)	
	INVCHAIN	MECLGATE	VCO
1	103†	305†	603†
	94†	288†	599†
2	101 (1.02)	234 (1.32)	482 (1.25)
	96 (0.98)	221 (1.32)	478 (1.25)
3	—	201 (1.56)	435 (1.41)
	—	190 (1.55)	432 (1.41)
4	111 (0.93)	209 (1.49)	—
	106 (0.89)	197 (1.48)	—
6	—	210 (1.49)	485 (1.25)
	—	199 (1.47)	482 (1.24)
11	—	257 (1.29)	—
	—	243 (1.22)	—
	LATCH	RINGOSC.1U	RINGOSC.2U
1	10562†	425769†	49328†
	10382†	413406†	48385†
2	5850 (1.81)	—	—
	5757 (1.80)	—	—
3	4414 (2.41)	—	—
	4344 (2.40)	—	—
4	3825 (2.80)	—	—
	3770 (2.79)	—	—
5	3179 (3.35)	—	—
	3134 (3.34)	—	—
7	2571 (4.19)	—	8977 (5.55)
	2532 (4.12)	—	8809 (5.55)
14	1994 (5.45)	42522 (10.22)	5688 (8.67)
	1971 (5.41)	41030 (10.27)	5584 (8.67)

Table 5.6: Execution times on the DEC cluster in seconds. The entries are: minimum observed total execution time (best overall speedup) and minimum observed time for the main analysis (best analysis speedup). Entries marked † are estimated times.

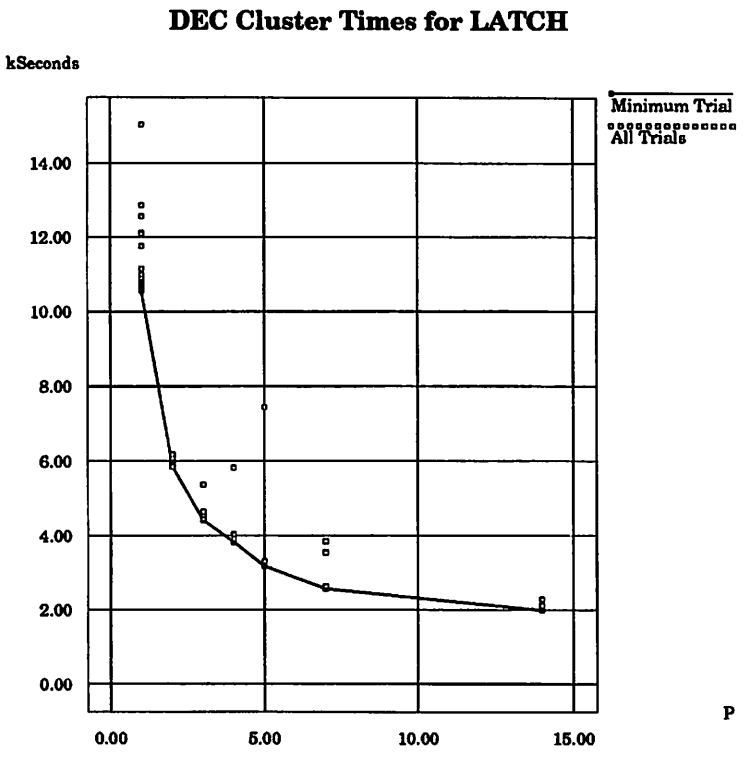


Figure 5.7: Total execution time for LATCH on the DEC cluster. Single processor times are the sums of the device-level times from each multiprocessor run.

parallel machines is the performance of the communication network which is less important when relatively more time is spent at the device-level than at the circuit-level.

5.5.4 Observed Limitations

The one-level model-evaluation algorithm has several limitations that become apparent in these two implementations. As a result, in many cases the ideal speedup of P on P processors has not been achieved. However, in some cases nearly ideal speedup *is* obtained. In this section, the limitations are identified and examples from the benchmark set are used to illustrate how the limitations can be of practical importance. In addition, strategies for overcoming these limitations by exploiting special cases are introduced.

Limited Number of Devices

The first and most important source of difficulty often is the limited parallelism available to exploit in one-level model evaluation. The maximum number of devices in the benchmark circuits is 14, so even though more processors are available on both the hypercube and the cluster, they can not be used. This limitation comes into play on every circuit, although it is more obvious when the number of numerical devices is very small as in the ASTABLE, BICMPD and BICMPU circuits. The main way to overcome this problem is by running multiple simulations at once; i.e. use design-level parallelism. By using the extra processors as a way to increase system throughput, the time to finish a design task is greatly reduced. This approach has been used often when running the simulations described in Chapter 6.

Example: On one weekend, while performing a device characterization application, 2 MOSFET device designs were simulated at 7 channel lengths with 3 sets of bias conditions applied to the devices. In total 42 simulations were run. By using the large number of DS5000/125 workstations and one DS5000/240 compute server in the DEC cluster, the simulations were all completed in less than a day. The shortest simulation took 8 hours 20 minutes to finish, the longest took 18 hours 53 minutes and the average job took 12 hours 49 minutes. The faster compute server was able to finish 2 of the jobs during this period. Overall, the 42 jobs would have taken roughly 538 hours or 22 days to complete on a single workstation. The speedup was therefore about 28 and the efficiency was about 70%. ■

Processor - Device Count Mismatch

The next limitation is the mismatch between the number of processors available and the number of numerical devices in the circuit. The speedup in this case is limited by Equation 4.5 as repeated here:

$$S(D,P) = \frac{D}{\left\lceil \frac{D}{P} \right\rceil} \tag{5.2}$$

where D is the number of numerical devices. This limit is a bigger problem on the hypercube, where the number of processors must be a power of two. Only some of the circuits have a device count That is also a power of two (e.g. DBRIDGE, CMOSAMP). Even when the number of processors can be tailored to the problem, few good choices

may be available that exactly divide the number of processors. (Consider the case where D is prime. Only when $P = D$ will no performance loss be caused by this problem.) In actual practice, this limitation has thus far been overcome by taking $P \geq D$ whenever possible and living with the resultant waste of some of the processors when $P > D$.

Task Size Imbalance

Many of the circuits use a different numerical model for each kind of device in the circuit. For example, BICMPD, BICMPU, and GMAMP are BiCMOS circuits that contain both bipolar and MOS devices. Since different meshes are needed to accurately simulate each kind of device, the time per iteration per device varies from one to the next. This effect is accounted for in the speedup model of Equation 4.6 and in the annealing load balancer of Section 4.9. Another example of this problem is when one-carrier simulation is used for the MOS devices as in all of the benchmarks. Then the time per iteration is different for bipolar and MOS devices even if the mesh sizes are similar because only 2 of the 3 semiconductor device equations are solved for the MOS devices. A third example of this problem accounts for the some of the differences in job execution time in the MOS device characterization example because devices with different channel lengths need different mesh sizes. A final case that is not represented in the benchmark set is when one- and two-dimensional numerical device models are mixed in the same circuit.

Many of the benchmark circuits suffer from this problem, however the two PPEF examples are the most interesting. These two circuits use numerical bipolar models and have 2 NPN devices and 2 PNP devices. (PPEF.1D uses 1^D models and PPEF.2D uses 2^D models.) On two processors, the round-robin partitioner puts 1 NPN and 1 PNP on each processor leading to almost perfect load balance; the efficiency on the iPSC/860 is greater than 95%. However, on 4 processors the difference between the NPN and PNP device meshes is exposed and the efficiency drops to 90% and 80%, respectively for the 1^D and 2^D cases. The drop is less pronounced in the 1^D case because the mesh sizes are closer to one another.

The simplest way to overcome this problem is by trying to match the mesh sizes for the numerical devices as much as possible. However, this will in general

result in either compromised accuracy for devices whose mesh size are reduced or extra unneeded accuracy for devices whose mesh sizes are increased.

Latent Devices

Even for circuits with identically modeled devices (e.g. DBRIDGE, ECLINV, MECLGATE), the speedup predicted by Equation 4.5 may not be achieved. This is caused by a fourth nonideality: differences in the activity of the numerical devices in the circuit. During DC and transient analysis, some of the devices become latent and are bypassed during the model-evaluation phase. Also during DC analysis, different iterations counts may be needed to obtain convergence of the device-level Newton-Raphson iteration. For AC analysis, latency is not as big a problem because at least one device-level solution is needed for every device in the circuit at each frequency point. Some imbalance may result because an iterative AC solver is used in DSIM. However, the iterative solver usually converges in 2 iterations, and if it fails a single direct-method solution is performed instead. As a result, if the numerical device models are identical, most of the time the same amount of work is done for each numerical device.

The effects of latency are accounted for in the serial execution time models of Equations 3.3 and 3.4. An equation for the parallel execution time in the presence of latency is now derived. For simplicity, the numerical models are assumed to be identical. In the serial case, one time unit is taken for each active device so the time taken is proportional to the number of active devices. In the parallel case, the same holds true for each processor, so the time taken is proportional to the maximum number of active devices on any one processor. Assume at this point that each processor is assigned the same number of devices (i.e. P divides D evenly). Assume also that each device has a probability f of being active on any given iteration, and that the devices become inactive independently of one another[9]. The probability that any one processor has less than or equal to A active devices is given by:

$$\mathcal{P}(A,d,f) = \sum_{i=1}^{A} \binom{d}{i} (f)^i (1-f)^{d-i} \tag{5.3}$$

[9]This last assumption may not be true of actual circuits, especially digital ones.

where $d = D/P$ is the number of devices on a processor, $\binom{d}{i}$ is the number of ways to choose i active devices from the d total, and $(f)^i(1-f)^{d-i}$ is the probability that exactly i devices are active. Therefore, the expected value of the maximum number of active devices, $\langle D_{active} \rangle$, is given by:

$$\langle D_{active} \rangle = \sum_{a=1}^{d} a \cdot \left[(\mathcal{P}(a, D/P, f))^P - (\mathcal{P}(a-1, D/P, f))^P \right] \tag{5.4}$$

where a is a number of active devices, and the expression in brackets is the probability that the maximum number of active devices on the P processors is exactly equal to a.

As shown, Equation 5.4 is valid for any values of D, P and f where D/P is an integer. However, it can be simplified considerably in special cases. In particular, if the number of devices per processor is equal to 1, as suggested in Section 5.5.4, then the expected number of active devices is 1 minus the probability that all the devices are inactive:

$$\langle D_{active} \rangle = 1 - (1-f)^P \tag{5.5}$$

If instead these same P devices are placed on a single processor, the expected value is simply:

$$\langle D_{active} \rangle = P \cdot f \tag{5.6}$$

So the speedup in this special case is given by:

$$S = \frac{P \cdot f}{1 - (1-f)^P} \tag{5.7}$$

Figure 5.8 is a plot of speedup using Equation 5.7 for several different values of P. As can be seen, for most values of P and f, the probability that all devices become latent at the same time is very low, so the efficiency $\eta = S/P$ is limited by the fraction of active devices f. Essentially what this means is that there is less overall latency when executing the model evaluations in parallel.

Example: An examination of event traces taken from a run of the RINGOSC.2U benchmark verified that only a very small percentage of iterations had all 14 devices inactive at the same time. ■

In Table 5.7 the measured speedups (S_{meas}) obtained on the iPSC for cases

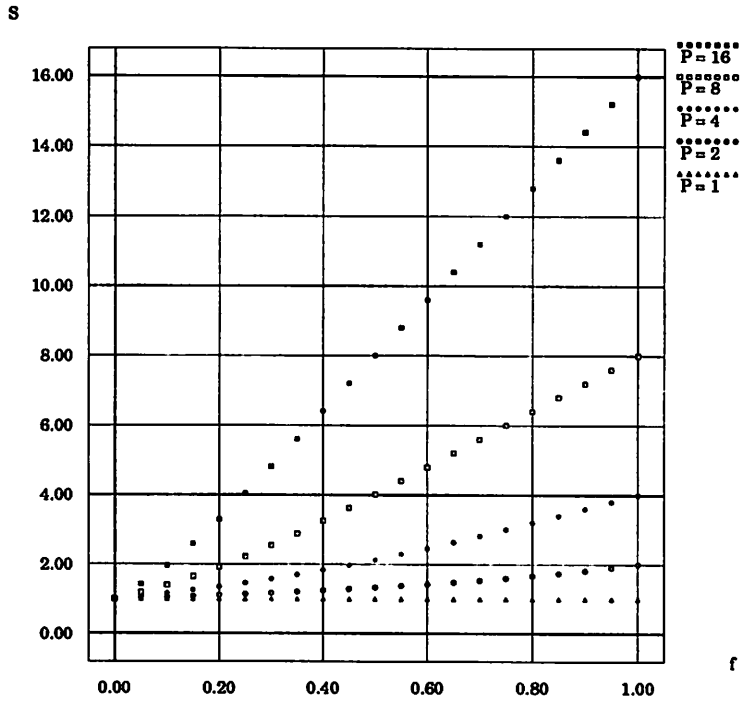


Figure 5.8: Speedup predicted in the presence of latency

where each processor has at most one device are compared to the speedup modeled using Equation 5.7 (S_{prcd}) and to the average number of active devices $\overline{D} = D \cdot f$. The value of P used is set equal to D, the number of numerical devices. The equation remains valid in this case because it is known ahead of time that the extra processors will not limit the execution time in the parallel case. The fraction of time a device is active is computed by taking the total number of transient device-level iterations and dividing by the number of circuit iterations and the number of devices. For the most part, the average number of active devices provides a reasonable estimate of the speedup that can be obtained. The difference between $\overline{D}$ and S_{model} is also generally negligible.

There appears to be little that can be done to prevent latency effects from degrading the speedup of the one-level model-evaluation algorithm. Since it is difficult if not impossible to predict the activity of the devices before running the simulation, a static load balancer cannot compensate for this effect in the device assignment step. Dynamic load balancing during the simulation is also likely to be ineffective because

Circuit	D	f	S_{meas}	S_{model}	$\overline{D}$
ASTABLE	2	0.77	1.64	1.63	1.54
CLKFEED	3	0.91	2.74	2.73	2.73
DBRIDGE	4	0.85	3.16	3.40	3.40
INVCHAIN	4	0.61	2.54	2.50	2.44
MECLGATE	11	0.56	4.98	6.16	6.16
LATCH	14	0.54	8.51	7.56	7.56
VCO	6	0.66	3.93	3.96	3.96

Table 5.7: Comparison of iPSC speedup with average number of active devices

latent devices can reactivate at any time thereby disrupting attempts to rebalance only the currently active devices. As noted in [GATE93], turning off the bypass capability altogether does no good because it increases the percentage of active devices at the expense of computing more model evaluations so that the overall execution time actually increases. One possible solution is to move to the multi-level model-evaluation algorithm where the number of groups in the upper levels is small so that the probability that all groups become inactive at the same time is increased. For example, at the highest level where there is only one group, all the latency of the serial algorithm can be exploited. However, the gain from increased latency exploitation is likely to be offset by the decreased efficiency of parallel device simulation, so the overall performance improvement would be less substantial.

Communication and I/O Overheads

The fifth limitation of the one-level model-evaluation algorithm is heavily influenced by the specific performance characteristics of the machine being used. Unless communication and I/O overheads are decreased in the same proportion as the main computation, they can become factors that limit overall speedup. Communication time increases when processors are added to the one-level model-evaluation algorithm because the global circuit matrix and RHS must be distributed to more processors. This is more of a problem on the DEC cluster, as demonstrated in Section 5.3.2, where the time for global reduction scales greater than logarithmically with the number of processors. In addition, the relative speed of communication to computation, or machine granularity, is worse on the cluster than on the iPSC.

Because the iPSC was specifically designed for high-performance message-passing applications, its communication performance usually does not severely degrade the speedup below what is predicted by the latency-dependent model of the previous section. Only in cases where the task size is low and the number of processors is high (e.g. the MECLGATE benchmark) does it begin to disrupt performance. Of more concern is the I/O performance of the node-to-host link which is used to return circuit- and device-level output to the System Resource Manager. The limited bandwidth of this link and the fact that all results go through it can turn it into a serial bottleneck. While CPU time for the main computation decreases, the I/O time remains constant and significant. As shown in [GATE93], for the MECLGATE example, the time to save one numerical device's internal state accounts for about 40% of the total time on 1 processor and about 70% of the total on 16 processors because it does not scale down. The degradation would increase if more than one device state were saved, and the overall computation rate would be I/O bound. One alternative that avoids the node-to-host link is to use the concurrent file system. This would free up the compute nodes more quickly for other jobs at the expense of having to retrieve the results using the slow node shell.

On the DEC cluster, which was not originally designed to support message-passing parallel processing, the network performance has a dramatic impact on the overall performance. Thc worst case behavior is shown in Figure 5.9 for the MECLGATE benchmark, where the total analysis time is actually *increasing* as more processors are added to the problem. The reason for this behavior is the relatively short time per iteration spent evaluating the numerical models in this circuit. Similar behavior is exhibited by the 3 other circuits (ASTABLE, INVCHAIN, and VCO) that employ the same one-dimensional numerical bipolar model.

Startup Overhead

The final limitation of the current implementations is the time to start a parallel job. On a single workstation, a simulation can be started without any noticeable delay. However, in a parallel environment, the time to set up the parallel machine can be significant. On the iPSC, the cube used must be allocated initially, loaded with the executable at the start of the run, and deallocated at the run's conclusion. On the

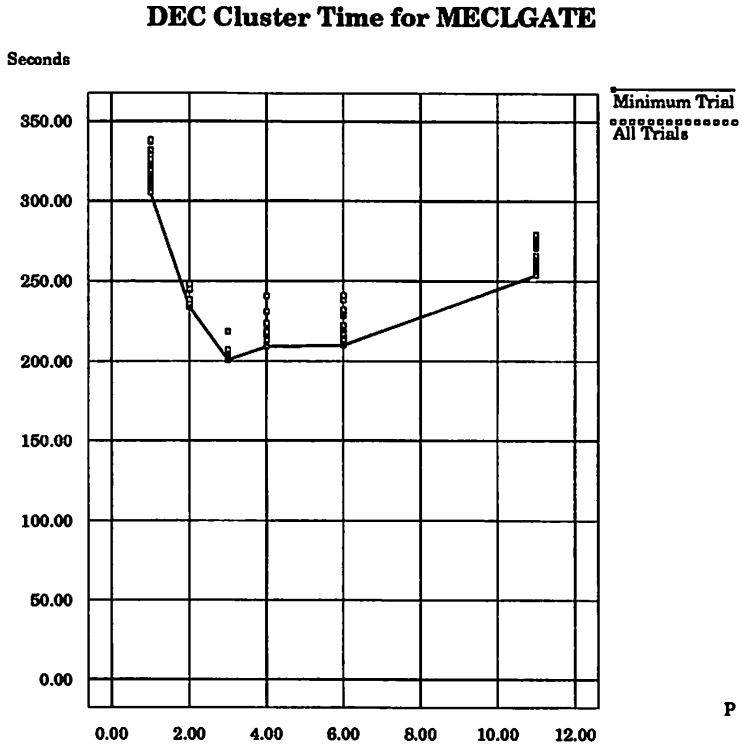


Figure 5.9: Total execution time in seconds for MECLGATE on the DEC cluster. Single processor times are the sums of the device-level times from each multiprocessor run.

DEC cluster, a script is run to determine which machines on the network will perform the job. If two or more jobs need to be run, they must be started serially to prevent them from selecting the same unloaded machines to use. A serial loop then spawns the simulation processes on the various machines. Each spawning requires the CIDER executable residing on a centralized host machine to be transferred across the network to the chosen remote machine. This step is sometimes so time-consuming that built-in alarm routines occasionally time out waiting for large clusters to initialize, and the startup process is terminated.

Table 5.8 shows the times needed to run a null job on an 8-processor machine in four different cases. The first two cases are the preferred methods for starting jobs on the iPSC and the DEC cluster, respectively. On the iPSC, the job is submitted to NQS which takes care of allocating a cube for the job. On the DEC cluster, the network is first scanned for unloaded machines and then the TCGMSG `parallel` command is used to bring up the machine. For this test, the pool of workstations scanned

Startup Method	Time
iPSC - Queue	41
DEC - Scan	175
iPSC - Direct	40
DEC - Reuse	28

Table 5.8: Job startup times in seconds on the iPSC/860 and DEC cluster

contained 40 processors. The second two methods save time by bypassing steps in the preferred methods. On the iPSC, a cube can be allocated directly, bypassing the queueing system. However, such jobs are subject to preemption by jobs submitted to NQS. The one second improvement does not seem to justify the risk. On the cluster, a previous cluster configuration file can be reused, bypassing the network scan. This saves almost two and a half minutes in startup time. Unfortunately, this approach is dangerous because the machines listed in an old file may not be unloaded at a later time[10].

Since the startup time increases the fraction of time executing serially, Amdahl's Law dictates that the job itself must be many times the startup time before noticeable speedup is achieved. The system may be executing efficiently because the parallel resources are not tied up during most of the setup time. However, from the user's point of view, the high startup cost decreases productivity and reduces the ability to use mixed-level simulation for short 5 to 10 minute jobs.

5.6 Summary

The one-level model-evaluation algorithm has been implemented on two different distributed-memory multicomputers. The first system is an Intel iPSC/860 hypercube, a scalable, high-performance computer. The second system is a cluster of DEC engineering workstations. The TCGMSG portable message-passing package provides communication services for both implementations. Fast message-passing is available on the iPSC; the DEC cluster is slower due to the limited bandwidth of its Ethernet connectivity. The programming effort involved in parallelizing CIDER is min-

[10]If the primary user of a workstation has returned in the meantime, he might be somewhat upset to find a large mixed-level simulation running on his machine.

imized by the simplicity of the one-level model-evaluation algorithm. Parallelism is exploited primarily through the use of efficient global-reduction routines.

The performances of the implementations have been measured by running a set of 17 benchmark circuits. The iPSC version shows good speedup and efficiency on several of the benchmarks, but poor efficiency on some of the others. A best speedup of 12 on 16 processors has been observed. In certain cases, the DEC cluster version can match the speedup of the iPSC version. However, the slower performance of the DEC cluster nodes makes the iPSC version the faster of the two. In other cases, the poor communication performance of the DEC cluster causes runtime to increase when large numbers of processors are used.

The reasons for the varied performances of the two implementations have been traced to several limitations of the one-level model-evaluation algorithm. The speedup available is limited by the number of devices modeled numerically and the number of processors. In the extreme case of a circuit with only one numerical device, no parallelism is available using this technique. In practice, several other factors make it difficult to achieve the best speedup predicted by Equation 4.5. First, in situations where different meshes are used for the numerical devices in a circuit, the workload per numerical device can become unbalanced. An example of this is a BiCMOS circuit, where the bipolar and MOS devices require different mesh specifications to achieve accurate solutions. Second, in DC and transient analyses, devices can become latent, reducing their workloads to nearly nothing. If numerical devices go latent, it is possible for a processor to sit idle for lengthy periods while the other processors work on the nonlatent devices. This is another source of load imbalance that limits speedup. Third, even though the amount of information passed between compute nodes is minimal, the time taken by this step can become significant if there is a large imbalance between the computation and communication speeds of the multicomputer. Since all communication is overhead, it reduces the overall speedup. This effect is minimized when the numerical device models are made larger (more mesh points are used). Finally, the time to start a parallel job reduces the effectiveness of parallel computing for short-running simulations.

Chapter 6

Applications of CIDER

6.1 Overview

Mixed-level circuit and device simulation has been used previously in a number of situations where compact device models fail to provide accurate results. The list of such applications has grown steadily since the introduction of the earliest mixed-level simulator MEDUSA [ENGL82]. However, in the past, applications have been limited to circuits containing only a few numerical devices, because of the enormous computational burden imposed by mixed-level simulation. In this chapter, several applications of CIDER are presented that demonstrate how high-performance engineering workstations and parallel computing can be used together to expand the range of problems that can be approached using mixed-level simulation. In addition, the extended capabilities of CIDER compared to its predecessor CODECS are used to provide more realistic numerical modeling of physical effects that are important in present-day IC devices.

The chapter begins by introducing a hypothetical 1.0 μm complementary bipolar - complementary MOS (CBiCMOS) process. CIDER is used as a device simulator to characterize the electrical performance of the various devices available in the process. The results of device simulations are used in the remaining sections of the chapter to obtain SPICE model parameters.

The first circuit application is a study of output resistance and gain modeling in a variety of analog IC amplifier stages. Existing SPICE models provide only a crude first-order modeling of the output resistance of transistors. As a result, large

qualitative and quantitative differences are observed when SPICE simulations are compared to the results obtained from CIDER.

In a second application, the performance of a compound-device push-pull emitter-follower (PPEF) output stage is examined. Results of simulations from both SPICE and CIDER are compared and substantial differences are observed in the large-signal performance of the output stage.

6.2 Hypothetical 1.0 μm CBiCMOS Technology

Before studying the performance of complete circuits, models must be developed for each type of device available to the circuit designer. In a traditional approach based solely on circuit simulation, parameters for the devices' compact models must be determined either by measuring actual devices or from hand calculations guided by a knowledge of the device structure. For advanced technologies, where two-dimensional effects are important, usable hand calculations are generally difficult if not impossible to perform. As a result, a parameter extractor has become an essential adjunct to any new compact model developed [JENG90]. However, parameter extraction cannot always be relied on to produce physically correct values for the parameters [LIN93]. A review of some of the deficiencies of existing compact models, and the difficulties that arise in developing parameters for these models is presented in [MAYA88].

In mixed-level simulation, numerically modeled devices are described directly using parameters of the underlying technology: critical dimensions, doping profiles and material parameters. Detailed descriptions of production technologies are rarely, if ever, published, since the information provided would give critical insight into the process optimizations introduced by the manufacturer. Consequently, in this work, a set of hypothetical device designs has been developed based on information drawn from a number of sources: published descriptions of actual technologies [IRAN91], [KAPO89], other simulation studies [BELL92b],[NAKA91] and informal discussions with a device physicist [KO93]. Four different device types are available: vertical NPN and PNP polysilicon emitter bipolar transistors [KAPO89] and NMOS and PMOS surface-channel lightly-doped drain (LDD) field-effect transistors [OGUR80]. Using these devices 4 different technologies can be created: a high-speed complementary bipolar process, a CMOS process, a BiCMOS process with fast, vertical NPN devices,

Process Parameter	Device Type	
	NPN	PNP
Minimum Emitter Width (μm)	1.0	1.0
Emitter-External Base Separation (μm)	1.5	1.5
Poly Emitter Thickness (μm)	0.2	0.2
B-E Junction Depth (μm)	0.11	0.12
Peak Emitter Doping (cm^{-3})	3.0e20	3.0e20
B-C Junction Depth (μm)	0.25	0.40
Base Peak Doping (cm^{-3})	7.8e17	2.6e17
C-Buried Layer Depth (μm)	1.0	1.0
Epi Layer Doping (cm^{-3})	1.0e16	1.0e16
Buried Layer Peak Doping (cm^{-3})	5.0e19	5.0e19

Table 6.1: Key process parameters for bipolar devices

and a full CBiCMOS process. Issues of manufacturability of the various process options have not been addressed. As a result, it may not be technically or economically feasible to use these device designs in an actual production process.

Because the devices are not based on a specific technology and the simulated performance has not been compared to measurements on actual devices, conclusions based on strictly quantitative analysis cannot be made. However, the underlying physical basis of the numerical modeling approach results in qualitatively correct behavior of the simulated devices, as demonstrated next. In addition, the observed behavior is also quantitatively reasonable, if not necessarily strictly accurate.

6.2.1 Bipolar Devices

Typical low-voltage high-speed bipolar technologies employ a number of advanced processing techniques to meet the necessary performance specifications. A thin heavily doped epitaxial layer and oxide isolation are used to reduce parasitic capacitances. A buried layer is used to minimize collector series-resistance, and polysilicon emitters are used to increase the forward current gain by a factor of 3-10 over that achieved using an aluminum emitter contact. A description of the process flow for a modern low-voltage process can be found in [GRAY93].

One- and two-dimensional numerical models have been developed for both NPN and PNP transistors. A summary of the key dimensions and doping concentrations is provided in Table 6.1. A two-dimensional cross-section of the NPN device is

shown in Figure 6.1. The device structure is assumed bilaterally symmetric about the

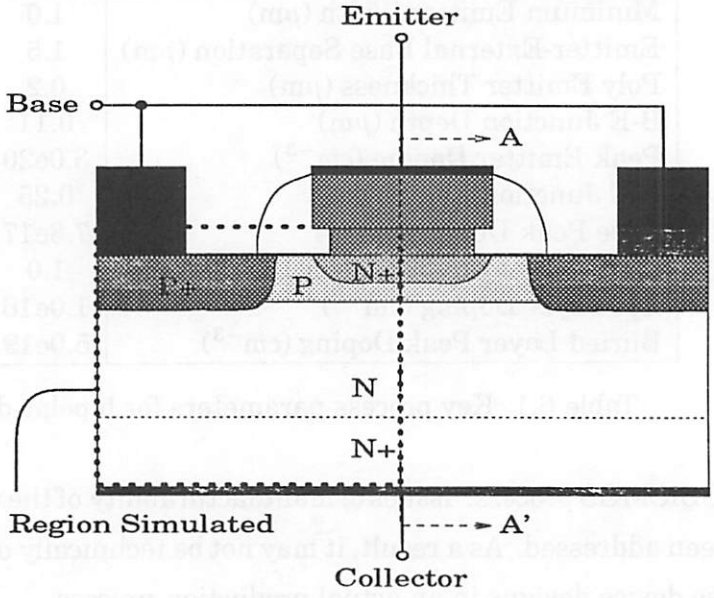


Figure 6.1: Cross section of NPN transistor

line A-A', so only half of the device is simulated. This corresponds to the assumption that dual base contacts are provided for the device. Under the emitter, the doping is uniform in the X direction, so the one-dimensional doping profile is taken along the line A-A'. In Figures 6.2 and 6.3 the 1^D profiles for the NPN and PNP devices are shown. Lower peak doping and a deeper base-collector junction are used in the PNP to increase the base Gummel number [SZE81] and current gain while still preventing base punchthrough under normal operating conditions.

Accuracy in bipolar simulation requires good physical models for the intrinsic carrier concentration n_i, the minority-carrier mobility in the base region and the transport properties of the polysilicon emitter. CIDER extends the abilities of CODECS to support bipolar simulation by incorporating many of the models and model parameters described in [SOLL90]. However, the polysilicon portion of the emitter is modeled differently from how it is done in [SOLL90] by using a separate semiconductor region that extends the underlying silicon emitter. In this region, all the material parameters are the same as for similarly doped monosilicon except for the mobility, which is reduced by a factor of 0.07 in accordance with the results presented in [ASHB87]. This capability is not available in CODECS because only one type of semiconductor

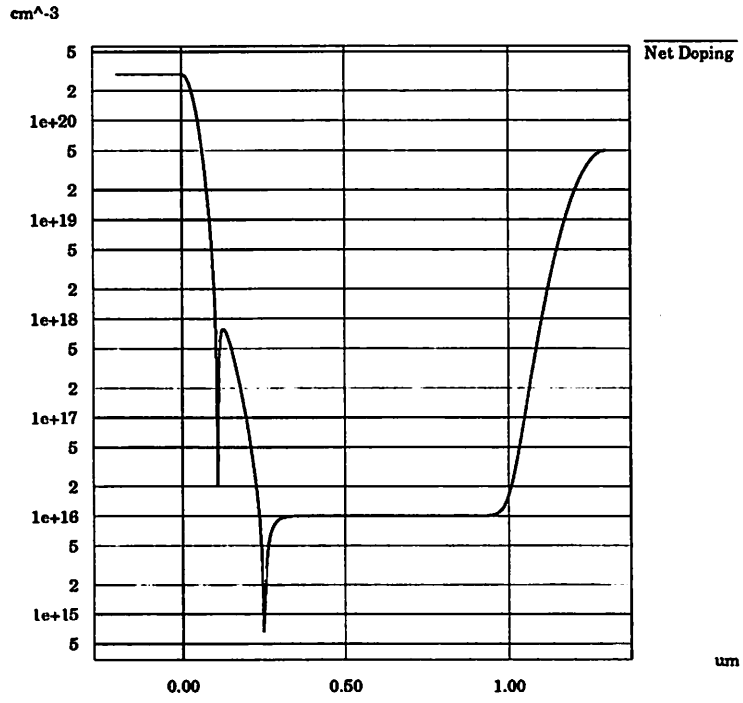


Figure 6.2: 1D NPN Doping Profile

material is supported. Simulations using normal silicon underestimate the current gain, because the higher mobility allows more current to flow in the emitter, thereby increasing the base current.

CIDER model descriptions for the devices are listed in Appendix D. Two different two-dimensional models are available for each device. One uses relatively fine mesh in the X direction while the other is coarser and therefore closer to a one-dimensional model with only approximate modeling of two-dimensional effects. However, the per-iteration time and memory use for the coarse-mesh model are significantly smaller than that of the fine-mesh model. These characteristics make it better for use in mixed-level simulations.

One-dimensional simulations have been performed to characterize the DC and AC electrical performance of these devices. The main device parameters are summarized in Table 6.2. Briefly, the NPN (PNP) device has a maximum current gain $\beta = \frac{I_C}{I_B}$ of 205 (82), a knee current density J_K of 0.2 (0.09) mA/μm^2, and a peak transition frequency f_t of 17.5 (5.5) GHz. These parameters are all measured at

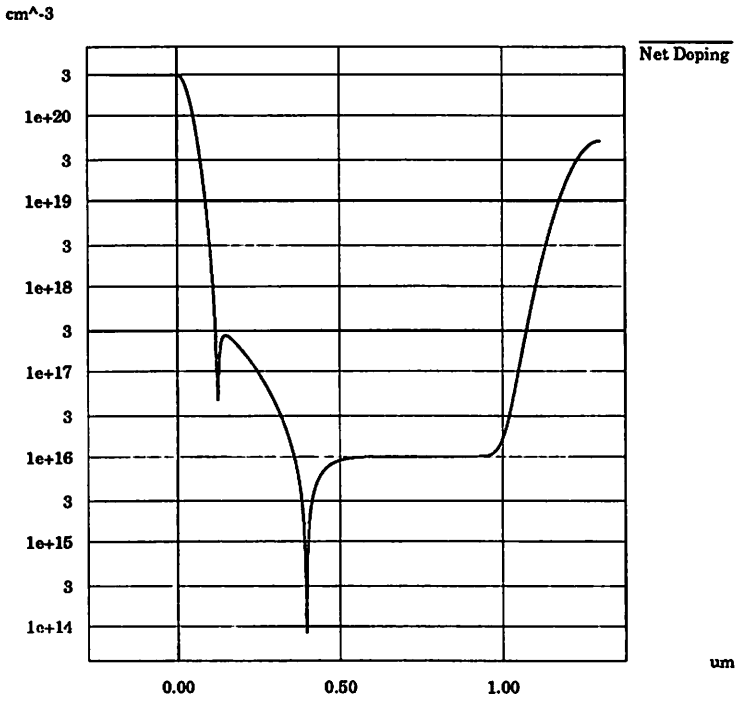


Figure 6.3: 1^D PNP Doping Profile

at a collector-base voltage $|V_{CB}|$ of 2.0V. The knee current density is defined as the point where β drops to 1/2 its peak value. The Early voltage [EARL52] is calculated from the change in the collector current for two different values of V_{CB} [GETR76]. Collector resistance is computed from the slope of the $I_C - V_{CE}$ curves in the saturation region, and base resistance is computed using a variation on the input-impedance circle method [NAKA91]. For the base resistance, the device was biased at a moderate collector current density 1/10th the knee current density. The emitter resistance is found by simulating the device with a high emitter current and measuring the voltage drop across the polysilicon layer using a plot of the internal potential of the device. The collector and emitter zero-bias capacitances are obtained directly from the small-signal admittances of the device when driven by a low-frequency input. Since one-dimensional simulations are used, the collector-base capacitance excludes the parasitic capacitance between the remote base contact and the buried layer. In addition, the parasitic collector-substrate capacitance is not modeled by either the 1^D or 2^D structures. If necessary, both parasitic elements could be reasonably modeled

Electrical Parameter	Device Type	
	NPN	PNP
Maximum Current Gain, β	205	82
Knee Current Density, J_K (mA/μm^2)	0.2	0.09
Forward Early Voltage, V_{Af} (V)	24	23
Collector Resistance, r_c, (Ω)	330	750
Base Resistance, r_b (Ω)	180	310
Emitter Resistance, r_e (Ω)	0.67	0.46
E-B capacitance, C_{je0} (fF/μm^2)	2.3	1.4
C-B capacitance, C_{jc0} (fF/μm^2)	0.3	0.3
Maximum Transition Frequency, f_t (GHz)	17.5	5.5

Table 6.2: Key electrical parameters for 1.0 μm $\times$ 10.0 μm BJT devices

using standard compact diode models with appropriately determined parameters.

Figure 6.4 shows a plot of the NPN collector and base currents, I_C and I_B, versus the base-emitter voltage V_{BE} for $V_{CB} = 2.0V$. For large V_{BE}, both collector and base current roll off due to a variety of high current effects. Among these are high-level injection at the emitter-base junction, base pushout, and voltage drops across the various parasitic resistances of the device. In addition, two-dimensional effects such as current crowding and lateral base pushout can also be important. This multiplicity of factors makes it difficult to model the behavior analytically in the high-current regime. In Section 6.4, a circuit application is presented where the devices operate at such high current levels.

6.2.2 MOS Devices

In a 1.0 μm channel-length MOSFET design, carefully engineered doping profiles must be used to maintain reasonable current drive and threshold voltage while still minimizing parasitic series resistance, and punchthrough and hot-electron effects [JENG90]. A thin gate oxide gives rise to high transconductance and it combines with shallow source and drain junctions to suppress drain-induced barrier lowering (DIBL). Unfortunately, the power supply voltage for 1.0 μm technologies has remained at 5V, which leads to increased electric fields within the device compared to previous MOSFET generations. In order to avoid avalanche generation at the drain junction during saturated MOSFET operation, lightly doped pockets are added at each end of the channel in order to smooth out the peak in the electric field that occurs near the drain

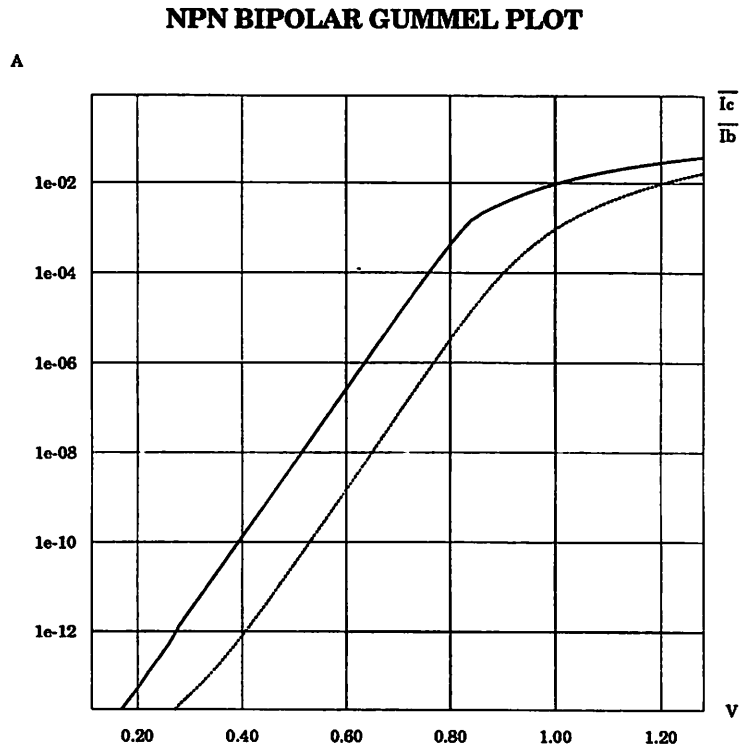


Figure 6.4: NPN Gummel plot for $V_{CB} = 2.0V$. Emitter area = 1 μm$\times$10 μm

side of the channel. LDD designs therefore reduce hot-electron device degradation but also degrade current drivability and gain due to increased source-drain resistance.

Because a MOSFET's operation is inherently two-dimensional, only 2^D numerical MOSFET models are available in CIDER. Figure 6.5 shows a device cross-section for a 1.0 μm NMOS device. The 1.0 μm PMOS device has an identical cross-section except for a change in polarity of all the doping impurities. This includes the polysilicon gate layer, so the PMOS device has a P^+ poly gate which leads to surface-channel operation of the PMOS device. The key process parameters and dimensions of the MOS devices are provided in Table 6.3. A three-dimensional view of the final doping profile is shown in Figure 6.6. The substrate doping tapers off from a peak doping of $1.0 \times 10^{17}\text{cm}^{-3}$ at the surface to a uniform concentration of $5.0 \times 10^{15}\text{cm}^{-3}$ at a depth of about 0.7 μm. The LDD implant is visible as a slight bump on the side of the drain junction.

The primary physical parameter that should be modeled accurately for short-channel MOSFETs is the mobility in the surface inversion layer. CIDER uses the

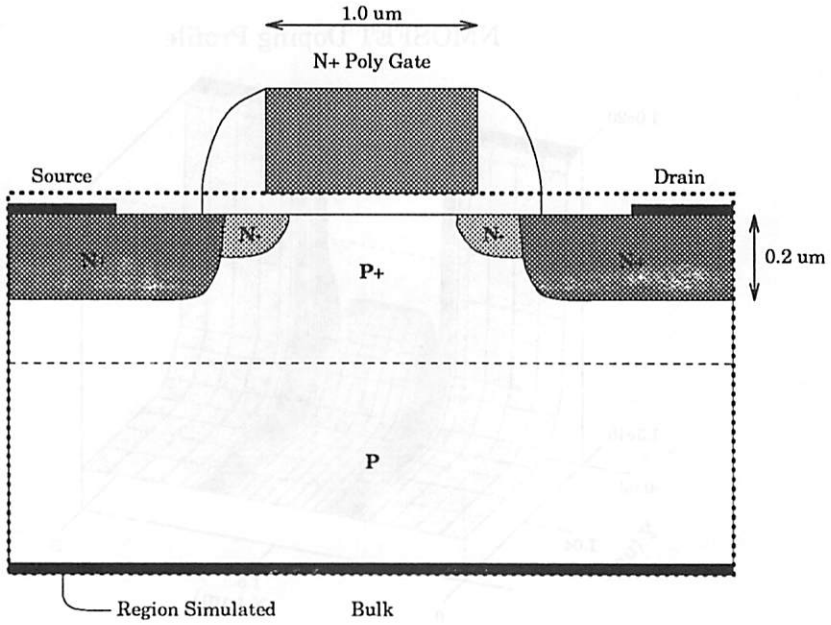


Figure 6.5: Cross section of NMOS transistor

model described in [GATE90] to account for normal-field and lateral-field mobility degradation[1]. This model was originally implemented in an updated version of CODECS that eventually evolved into CIDER. However, until the release of CIDER it has not been publically distributed to the world at large. Direct solution of Poisson's equation by the numerical model accounts for other short-channel effects such as channel-length modulation and DIBL. Hot electron effects (avalanche generation) are not accounted for

[1]This effect is also known as velocity saturation.

	Device Type	
Process Parameter	NMOS	PMOS
Supply Voltage (V)	5.0	5.0
Minimum Gate Length (μm)	1.0	1.0
Type of Gate	N^+ poly	P^+ poly
Oxide Thickness (Å)	200	200
Junction Depth (μm)	0.2	0.2
Substrate Doping (cm^{-3})	5.0e15	5.0e15
Implant Dose (cm^{-2})	1.6e13	1.6e13
LDD Length (μm)	0.1	0.1
LDD Doping (cm^{-3})	4.0e17	4.0e17

Table 6.3: Key process parameters for MOS devices

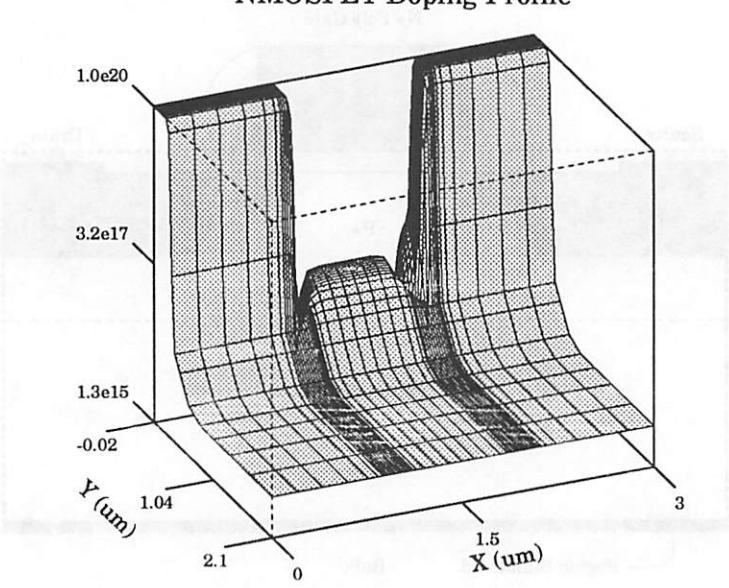


Figure 6.6: 2^D NMOS Doping Profile

in the model for two reasons. First, the two-dimensional avalanche generation model has never been implemented with the complete set of derivatives added to the device-level Jacobian matrix. This has resulted in noticeable convergence difficulties when using the model. Second, impact ionization current is carried by both electrons and holes and therefore requires full two-carrier device simulations. When it is omitted, current flow consists almost exclusively of majority carrier current (by electrons in NMOS, by holes in PMOS). Thus, one-carrier simulation can be used which results in considerable savings in CPU time.

CIDER model descriptions for the MOS devices are listed in Appendix D along with the bipolar descriptions. Because the distances in a two-dimensional MOSFET cross section depend on the gate length of the device, a different model is needed for each device length used in a circuit. Models are provided for gate lengths of 1, 2, 3, 4, 5, 10 and 50 μm. Models for other lengths can be obtained by adjusting the model distances that depend on the channel length.

Each of the devices was simulated for three sets of conditions: in the linear region with low V_{DS} of 50 mV and V_{GS} and V_{BS} swept, in the "square-law" region with $V_{DS} = V_{DD}$ and V_{GS} and V_{BS} swept, and in the saturation region with V_{GS} stepped

Electrical Parameter	Device Type	
	NMOS	PMOS
Effective Channel Length, L_{eff} (μm)	0.8	0.8
Threshold Voltage, V_t (V)	0.8	-0.7
Saturation Drain Current, I_{dsat} (mA/μm)	0.42	0.13
Subthreshold Swing, S (mV/decade)	100	100
Early Voltage, $1/\lambda$ (V)	38	30

Table 6.4: Key electrical parameters for 1.0 μm L_{drawn} MOS devices

in 1 V increments and V_{DS} swept from 0 V to V_{DD}. In total 42 simulations were distributed across the workstations of the DEC cluster described in Chapter 5. The DC characteristics of the 1.0 μm devices are summarized in Table 6.4. In short, the NMOS (PMOS) device has a threshold voltage of 0.8 (-0.7) volts and maximum current drive of 0.42 (0.13) mA/μm of width. The effective channel length listed is only approximate since it varies significantly with both applied gate bias and drain bias in LDD devices [HU87]. The Early voltage is also an approximation since a constant value is not accurate for characterizing the output resistance of short-channel MOSFETs. This topic is covered more thoroughly in Section 6.3

In Figure 6.7, the saturation curves for NMOS and PMOS devices with W/L = 10.0μm/1.0μm are shown. In the flat region of the curves, the increase in I_{DS} with increasing V_{GS} is limited by velocity saturation. For a fixed value of V_{GS} the current increases slightly wth increasing V_{DS} due to channel-length modulation [FROH69] and drain-induced barrier lowering [JENG90]. In the linear region, shown for the NMOS device in Figure 6.8, the current actually increases sublinearly with increasing V_{GS} due to normal-field mobility degradation, a voltage drop across the parasitic source resistance, and a gate-voltage-dependent increase in the channel length that is typical of LDD devices.

A key circuit used to measure the speed performance of CMOS circuits is the ring oscillator. The parallel version of CIDER for the iPSC/860 is capable of simulating this circuit in a reasonable amount of time. This capability allows the technology performance to be directly characterized without the need to extract compact model parameters in an intervening step. In Figure 6.9, the per-stage delay of a 7-stage unloaded CMOS ring oscillator is shown as a function of the supply voltage. The predicted performance is somewhat high because several capacitive parasitics have

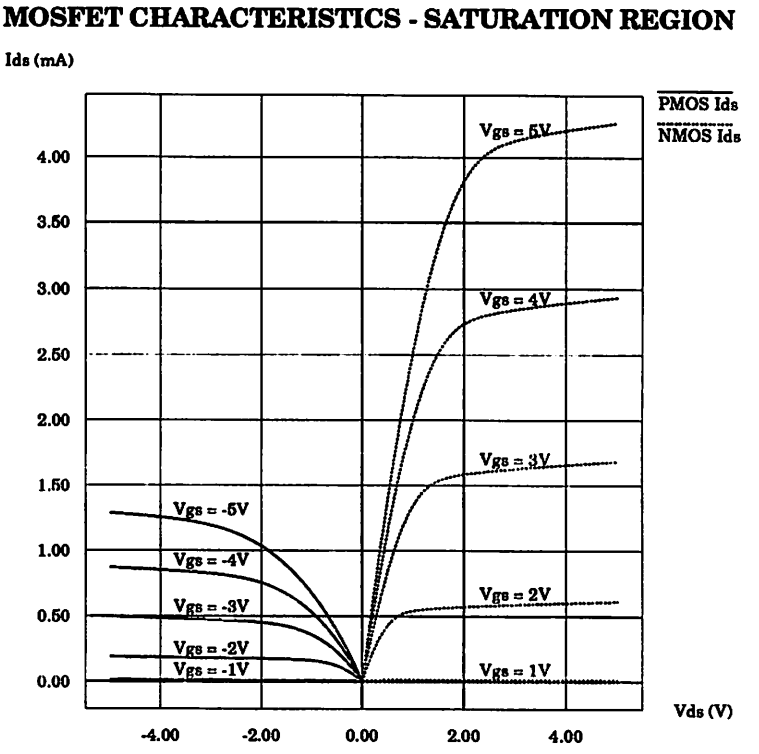


Figure 6.7: MOS saturation region characteristics for 10.0μm/1.0μm devices. Linear increase of maximum current with Vgs is caused by velocity saturation.

been ignored. As the supply voltage decreases, the gate delay increases because less current is available to charge and discharge the fixed capacitances of the circuit. The simulations were performed on a 16 node subcube of the iPSC/860 and took between 2 and 4 hours of CPU time each to execute[2].

6.3 Gain of Various Amplifier Cells

It would be nearly impossible to find an analog circuit that does not contain some kind of amplification stage. As a result, the small- and large-signal gains of amplifiers are key parameters in an analog IC design. Good device models are needed to establish the value of gain for a given circuit. Especially for very aggressive high-speed analog circuits where low-gain, wide-bandwidth amplifiers are used, designers

[2]Due to a bug in CIDER these simulations computed unnecessary circuit iterations and thus took longer to perform. For a better measure of the CPU time with the bug fixed, consult the results for the RINGOSC.1U benchmark in Chapter 5.

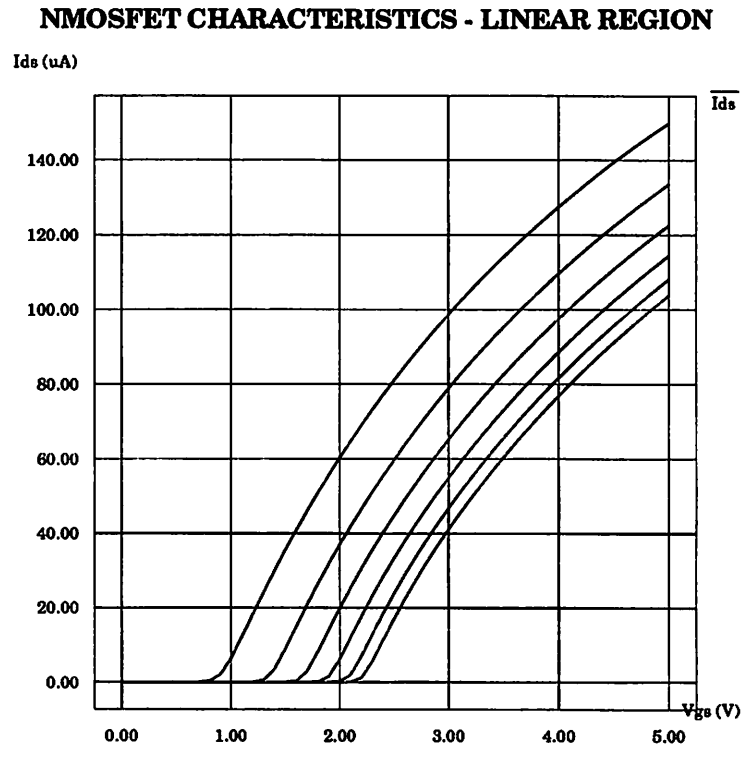


Figure 6.8: NMOS linear region characteristics for a 10.0μm/1.0μm device. Reduction in the transconductance $g_m = \frac{\partial I_{DS}}{\partial V_{GS}}$ is caused by transverse-field mobility degradation, series resistance, and channel-length variation.

are aided by precise modeling of the devices in the circuit.

Given the importance of gain calculations, it is surprising to note that most of the existing compact transistor models in SPICE have difficulty accurately predicting the gain for modern IC circuits. To some degree this is due to analog design techniques that minimize the importance of knowing the exact value of the gain. However, this difficulty is also caused by advances in IC processing that introduce important new physical effects and that invalidate some of the assumptions made for older technologies.

In this section, mixed-level circuit and device simulation is used to study the gain of several IC amplifier circuits. The deficiencies of output resistance modeling in the existing MOSFET models of SPICE are well known [JENG90]; for the sake of the current argument, the BSIM model is used to demonstrate limitations of these models. The BSIM model is chosen because it models the important short-channel

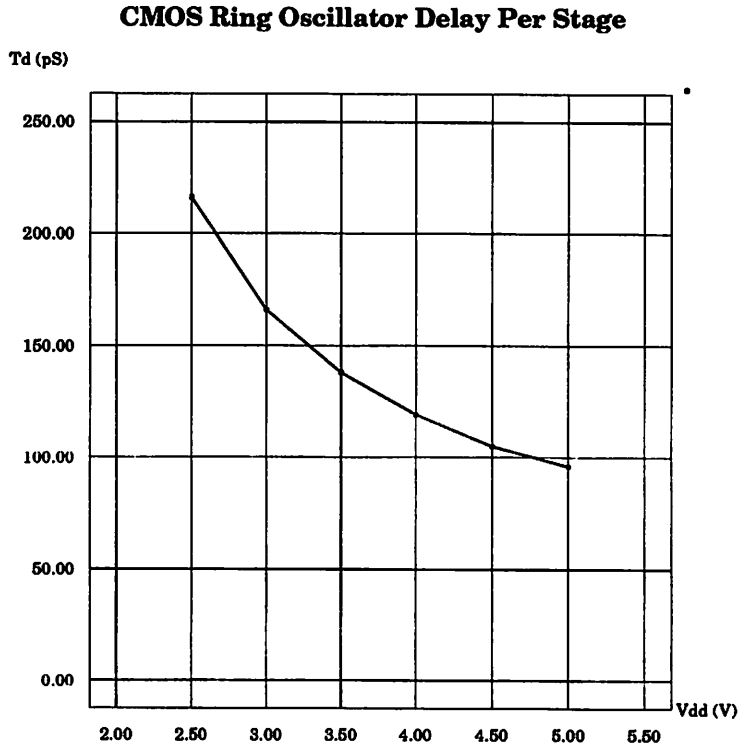


Figure 6.9: CMOS ring oscillator delay per stage. Ring has 7 stages of inverters with NMOS (PMOS) W/L = 3/1 (6/1) and no load capacitance.

effects reasonably well with a limited number of parameters. The BSIM2 model can produce better fits, but requires a much larger number of parameters, many of which are nonphysical. Only recently have compact models begun to appear that address these limitations in a physical way [HUAN93]. In contrast, the physically based numerical models of CIDER produce results that are consistent and physically reasonable. Thus, mixed-level simulation holds the promise that improved amplifier designs might result if CIDER were tuned to an IC process and used to design real amplifier circuits.

6.3.1 Ideal Inverter

In Figure 6.10, a simple circuit is shown that is used to determine the intrinsic gain available from a single MOS transistor. The same circuit can be used for bipolar transistors as well. However, the gain of bipolar circuits has previously been

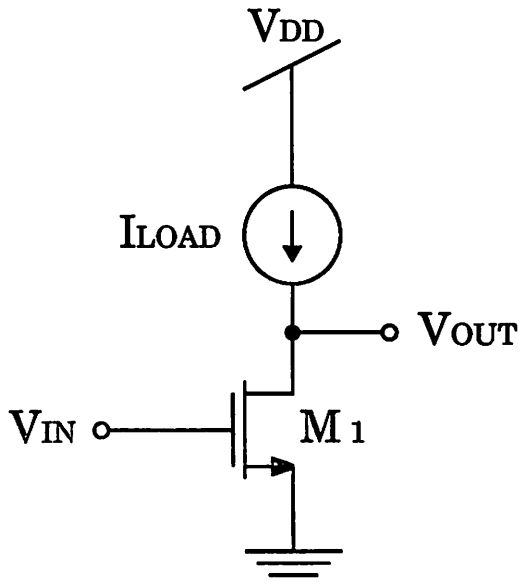


Figure 6.10: NMOS inverter with ideal load

investigated using CODECS, the predecessor to CIDER, in [ZARR89]. The transistor is biased with a constant current source that provides an ideal load for the active device. Figure 6.11 shows an exaggerated load line construction for this circuit. For each value of the input voltage V_{IN}, the output voltage V_{OUT} stabilizes at a point where the current drawn by the transistor is exactly equal to that supplied by the current source[3]. As the input voltage increases, the intersection point moves to the left and the output voltage falls. The rate at which it falls is the gain of the circuit, a_v, which is given by:

$$a_v \equiv \left|\frac{dV_{OUT}}{dV_{IN}}\right| = -\frac{\frac{dI_{DS}}{dV_{IN}}}{\frac{dI_{DS}}{dV_{OUT}}} = \frac{dI_{DS}}{dV_{GS}}\frac{dV_{DS}}{dI_{DS}} = g_m \cdot r_o \tag{6.1}$$

where g_m is the transconductance of the transistor, and r_o is its output resistance. All three of these quantities (a_v, g_m, r_o) are bias dependent. The transconductance can be obtained with reasonable accuracy using compact models as long as the drain current is modeled within a few percent of its actual value [GRAA90]. However, modeling of the output resistance requires that the *slope* of the $I_{DS} - V_{DS}$ curves be well matched in the saturation region.

[3]In practice, the current source tapers off for output voltage values higher than the maximum voltage of interest V_{DD} in order to prevent the output voltage from rising indefinitely when the transistor is off. In simulations, such a current source can be modeled by an ideal current mirror employing very wide transistors to create a very sharp corner in the load line as shown in the figure.

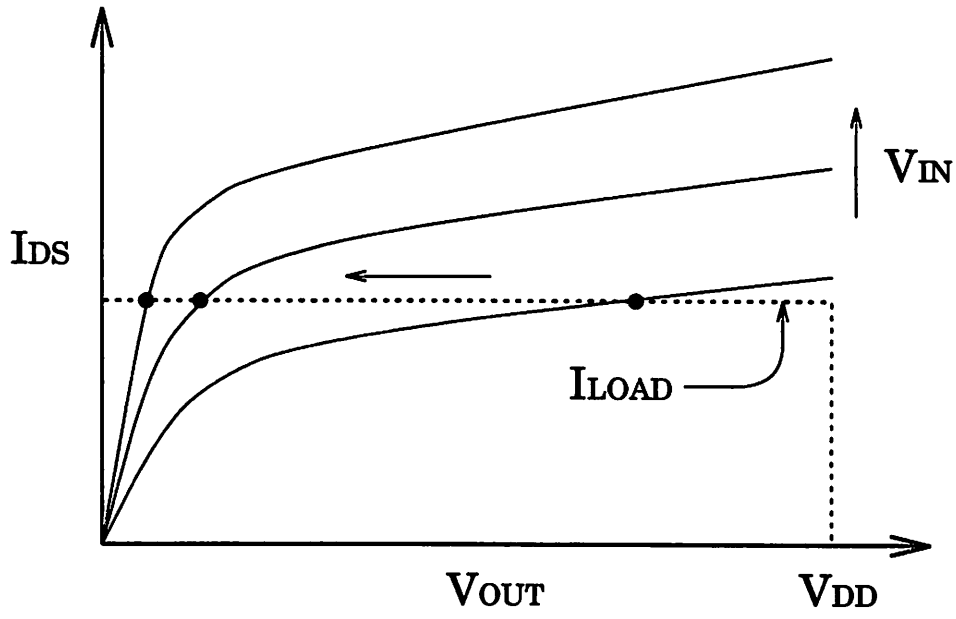


Figure 6.11: Load line construction for ideal NMOS inverter

In Figure 6.12, the gain of the ideal inverter is graphed against the output voltage for a 1 μm NMOS test device. The load current is set to 50 μA, so the device is operating at a current density of 5 μA/μm of width. Two different MOSFET models available in CIDER are used: the BSIM model and the CIDER numerical MOS model. The numerical MOS model is the same as used in Section 6.2.2. The BSIM model parameters have been chosen by trial and error to best match the results from the numerical model. The curve produced by the numerical model is smooth over the entire range of output voltage. Initially the gain is very low because the device is operating in the triode region. As V_{OUT} increases, the device makes a gradual transistion from triode to saturated operation. The output begins to increase roughly proportional to $\sqrt{V_{OUT}}$ due to channel-length modulation. For high values of V_{OUT}, the curve deviates from this dependence and begins to flatten out due to DIBL. In a real device, hot-carrier output-resistance degradation would cause the gain to fall in this same region. However, the numerical model does not include this effect and the effect of DIBL can be isolated. In contrast to the numerical-model results, the BSIM results are clearly non-physical. The gain curve demonstrates a sharp corner where the model equations shift from the triode to the saturation region. In the saturation region, the gain curve is concave up instead of concave down. In analog design, MOSFETs are

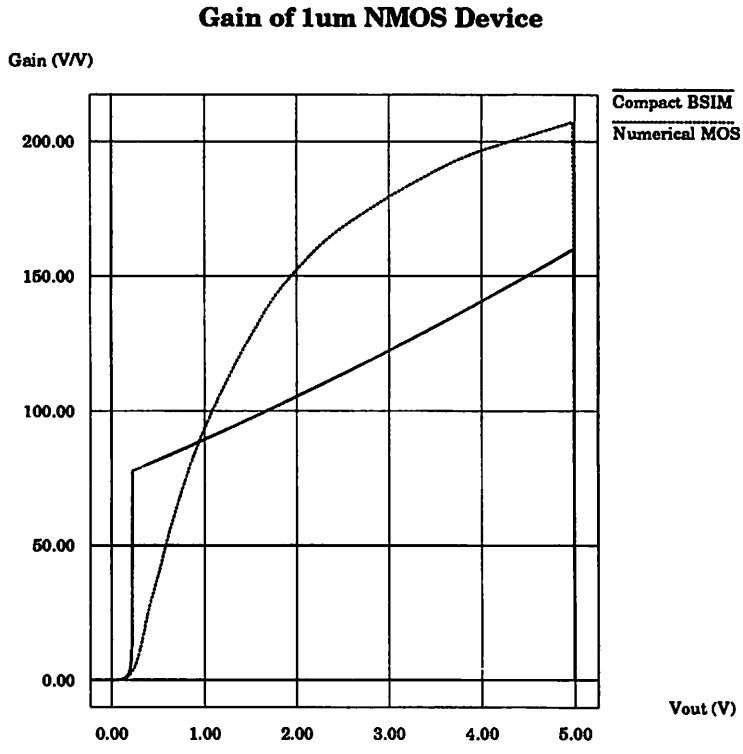


Figure 6.12: Gain of 1.0 μm NMOS transistor at I_{LOAD} of 50 μA. BSIM curve has a sharp corner whereas the numerical-model produces a smooth curve.

sometimes biased with an output voltage just above the triode-saturation transition point. The BSIM model severely overestimates the gain in this region as compared to the numerical-model results[4].

6.3.2 Source-Coupled Pair with Active Load

While a stand-alone device simulator possibly could be used to study the preceding ideal inverter, a full mixed-level simulator is needed for the following circuit. Moving a step up in complexity, Figure 6.13 shows an NMOS source-coupled pair (SCP) with a PMOS current mirror acting as an active load. While a stand-alone device simulator could be used to study the preceding ideal inverter, a full mixed-level simulator is needed for this circuit. This type of circuit is commonly used as the input

[4]The MOS level 3 model also has a sharp transition from triode to saturation and overestimates the gain in this region. However, the transition is so sharp that SPICE is unable to obtain convergence as the output voltage crosses the transition point. As a result, comparative results are not available.

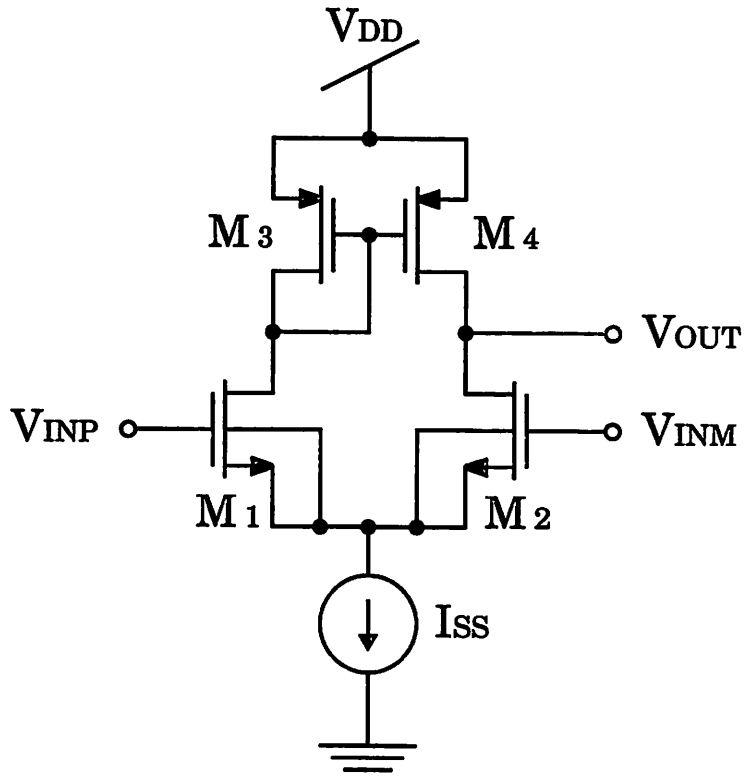


Figure 6.13: Schematic for source-coupled pair with active load

stage for CMOS operational amplifiers. The small-signal gain of this circuit can be shown to be [GRAY93]:

$$a_v = g_{m1,2}(r_{o2} \parallel r_{o4}) \tag{6.2}$$

This assumes the circuit is operating with equal DC bias currents flowing through transistors M_1 and M_2, so that the transconductance $g_{m1,2}$ is the same for both transistors. When determining the large-signal behavior of this circuit, unequal currents flow in each transistor and this formula is no longer applicable. Although closed-form expressions for the large-signal behavior of the SCP exist in simplified cases (such as with resistive loads [GRAY93]), no general expression is available. Therefore, it is necessary to resort to simulation, or direct measurement, to determine the performance of this circuit.

Figure 6.14 shows a typical plot of the output voltage V_{OUT} as the positive input V_{INP} is swept from 50 mV below to 50 mV above the negative input voltage V_{INM}. The bias current is twice that used in the ideal inverter so the NMOS devices

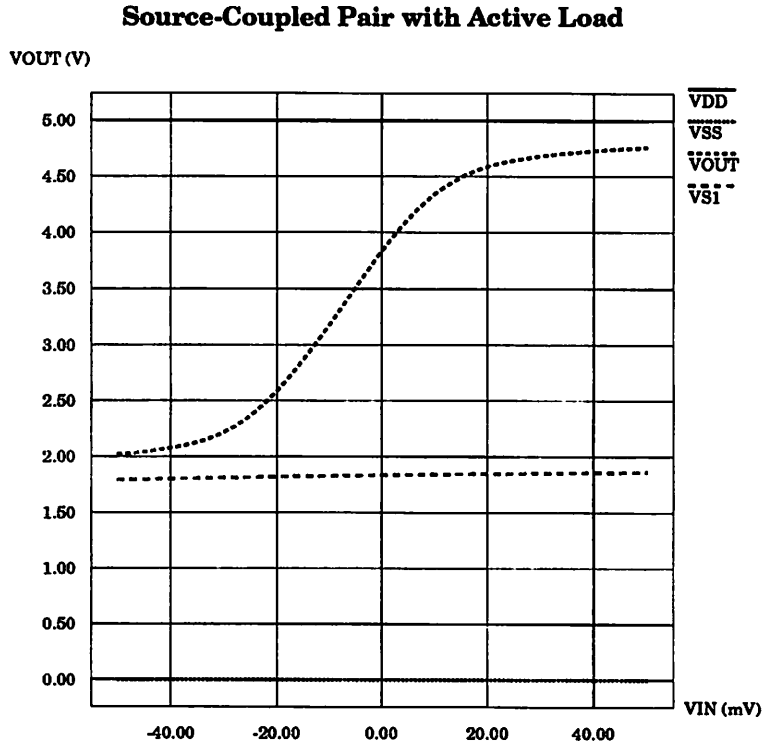


Figure 6.14: Output voltage of source-coupled pair with active load

are operating at a current density of 5.0 mA/μm when both inputs are at the same potential. The low output voltage is limited by the source voltage of transistor M_2 as M_2 goes into the triode region of operation. On the high side, the output is limited by the supply voltage of 5.0 V, and transistor M_4 goes into the triode region. Between these extremes, there is a high gain region where both M_2 and M_4 are operating in the saturation region. However, from the plot it is clear that there are no distinct boundaries to this region. In Figure 6.15, the slope of the output voltage (the gain) is graphed against the value of the output voltage. Results are presented for cases where all the devices are modeled numerically and where all are modeled using BSIM models. The inverted bell shape of the numerical-model results is distinctly different from the mesa shape of the BSIM results. The BSIM curve is shifted downward in voltage because the devices are operating at higher $V_{DS,sat}$. In addition, the BSIM curve displays two corners: one as M_2 switches from triode to saturation on the left side of the figure and the other when M_4 switches from saturation to triode on the right side. No such corners are visible in the numerical-model results; there is a smooth

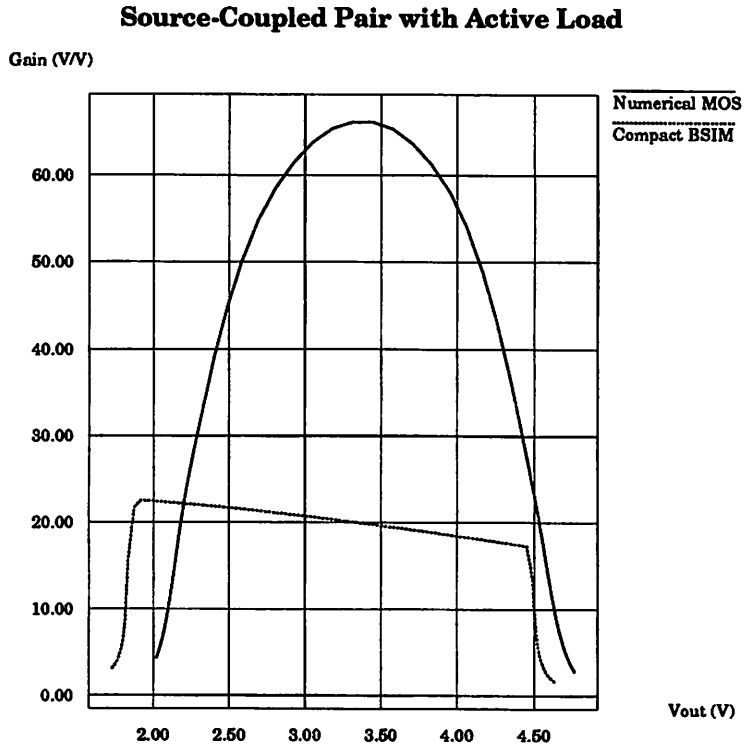


Figure 6.15: Gain of source-coupled pair with active load

transition between regions. The shape of the numerical-model curve is explained by the following argument. The total drain-source voltage for both output transistors is approximately constant. As the output voltage swings from low to high, the drain-source voltage of M_2 increases while that of M_4 decreases. As already mentioned, the gain of the ideal inverter increases as V_{DS} increases due to increased incremental output resistance. The same effect occurs here. However, as the output resistance of M_2 increases, that of M_4 decreases. Near the center of the output range, the two effects balance each other and a peak occurs in the gain.

The qualitative differences in the shapes of the two gain curves have important consequences for other types of simulations. For example, a small-signal analysis of the BSIM-model circuit simulation would show nearly identical results independent of the DC bias value of the output voltage. In contrast, the numerical-model results would vary considerably. In addition, a large-signal sinusoidal analysis would show little distortion when using BSIM models until the output voltage reached the upper and lower transition corners. With numerical models, significant distortion would be

produced independent of the output swing magnitude.

6.3.3 Two-Stage CMOS Opamp

A two-stage CMOS operational amplifier can be created by combining the actively loaded source-coupled pair with an additional inverting stage based on the ideal inverter. A schematic for this two-stage amplifier is shown in Figure 6.16. The input stage is modified by inverting the polarities of the NMOS and PMOS devices.

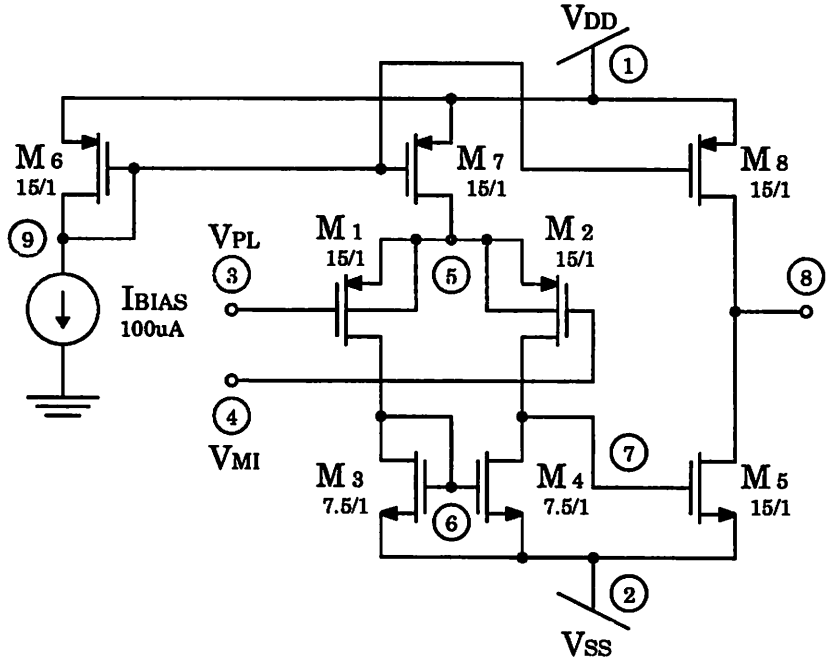


Figure 6.16: Schematic for CMOS two-stage amplifier

The ideal current source load is replaced by a PMOS current mirror as would be done in a real implementation.

To first order, the small-signal differential-mode gain of this amplifier is given by:

$$a_v = (g_{m1,2}(r_{o2} \parallel r_{o4}))\,(g_{m5}(r_{o5} \parallel r_{o8})) \tag{6.3}$$

where once again it is assumed that the input transistors are operating with equal DC bias currents. The gain depends directly on small-signal parameters for 5 of the 8 transistors, so these are the critical elements of the circuit. Due to matching considerations, transistors M_3 and M_6 must use the same models as M_4 and M_8

Configuration	Transistor								CPUs
	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	
A None	B	B	B	B	B	B	B	B	1
B Stage1	N	N	N	N	B	B	B	B	4
C Stage2	B	B	B	B	N	N	N	N	4
D NoInput	B	B	N	N	N	N	N	N	8
E All	N	N	N	N	N	N	N	N	8

Table 6.5: Two-stage CMOS amplifier test configurations. Entry of B denotes transistor modeled with BSIM; entry of N denotes numerical model. CPUs is the number of hypercube nodes used to solve the problem.

respectively. For the simulations that follow, the bias transistor M_7 could be replaced by an additional current source; however, it would need to be a transistor in any case if the value of the common-mode gain were needed. For this reason, it is modeled as a transistor here as well.

One way to reduce the time taken by mixed-level simulation is to model only some of the transistors numerically. However, it may be difficult to determine *a priori* which transistors operate in regions where the existing compact models are inaccurate. To investigate this approach, the two-stage amplifier has been simulated several times with some of the transistors modeled numerically and some modeled using BSIM. Table 6.5 lists the different configurations used to identify which transistors need to be modeled numerically. In configuration A, none of the transistors are modeled numerically, whereas in configuration E, they all are. In configuration B, only the transistors of the first stage are modeled numerically while the others use BSIM models. In configuration C, only the second stage transistors and their matching bias transistors are modeled numerically. In D, only the two input transistors, M_1 and M_2, use BSIM models. This may produce fairly accurate results since BSIM can model the input transconductance $g_{m1,2}$ reasonably well, and an error in r_{o2} is not critical because it is combined in parallel with r_{o4}. In Figure 6.17, the output voltage at node 8 is graphed versus the input voltage difference V_{IN} = V(4,7) for all 5 configurations. The results were obtained on the Intel iPSC/860 using subcube sizes appropriate for the number of numerical devices in the configuration. Four of the five cases are clearly discernible in the figure; however, cases D and E are almost indistinguishable. Ideally, the output voltage should pass through 0.0 V when V_{IN} = 0.0 V. Unfortunately,

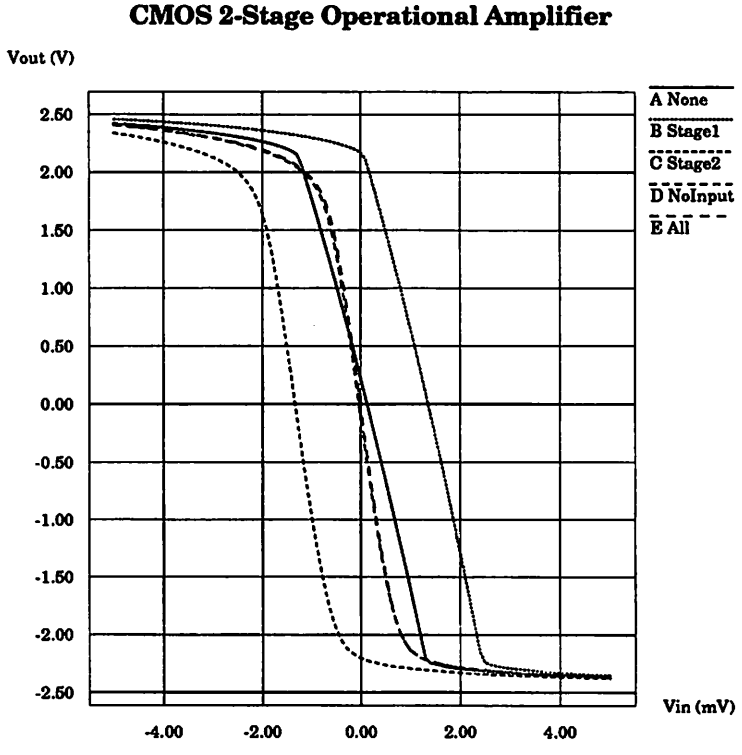


Figure 6.17: Output voltage of two-stage amplifier

cases B and C have approximately 1.5 mV offsets because a matching relationship has been broken. In a real design, if transistors M_3, M_4, and M_5 are not matched, a systematic offset voltage is known to arise [GRAY93]. In this case, mismatched device models produce the same effect. From the figure it is also apparent that certain cases produce smoother output curves than others. A clear view of this phenomenon can be obtained by plotting the gain, as shown in Figure 6.18. The degree of smoothness is primarily dependent on whether or not the two output transistors M_5 and M_8 are modeled numerically. For cases A and B there are visible transitions at the edges of the high gain region due to triode-saturation region border-crossings in the BSIM model. This effect is not apparent for case C where the two output transistors of the first stage are modeled with BSIM because the devices always operate in saturation. In terms of overall accuracy, only case D matches the behavior of the all-numerical-model configuration with an acceptable level of accuracy.

From the above arguments, it should be clear that all but three of the transistors (M_1, M_2, and M_7) are critical in determining the differential-mode gain. Since

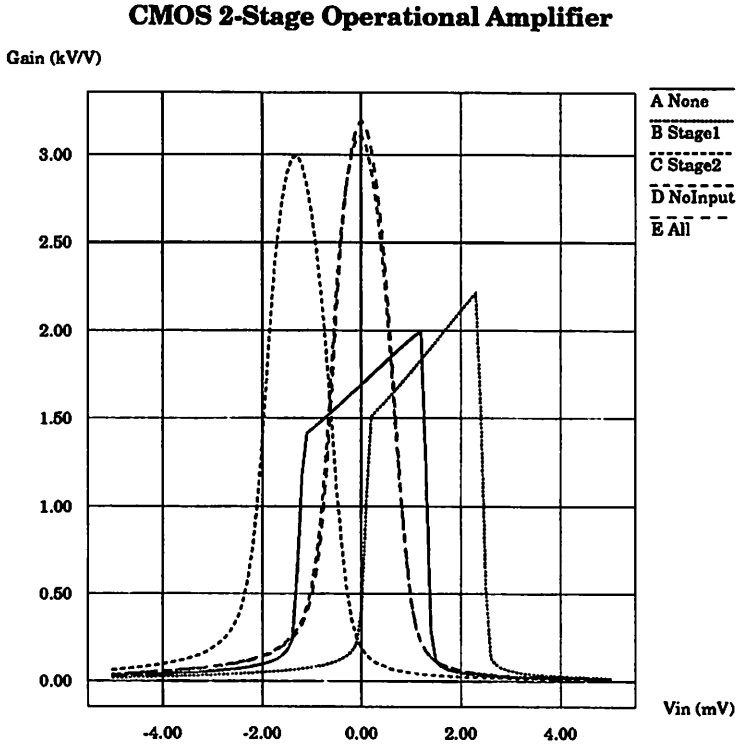


Figure 6.18: Gain of the two-stage CMOS amplifier

a hypercube of at least 8 processors is needed to achieve maximum speedup for the 5 critical devices, it is unfortunately not possible to take advantage of this in this example.

6.4 Push-Pull Emitter-Follower Output Stage

The primary goal of an IC output stage is to supply power to a load device while maintaining an acceptable level of signal distortion [GRAY93]. One commonly used circuit for this purpose is the push-pull complementary emitter-follower (PPEF) circuit shown in Figure 6.19 [PEDE91]. If identical NPN and PNP devices are available, excellent distortion behavior is observed. However, several nonidealities cause higher distortion levels to appear under more realistic circumstances. Power gain is achieved in the form of the combined current gains of the compound NPN-PNP devices. Due to high-level injection effects which reduce the high-current gain, the power gain falls off as large amplitude voltages are applied to the input. Given a set of NPN and PNP

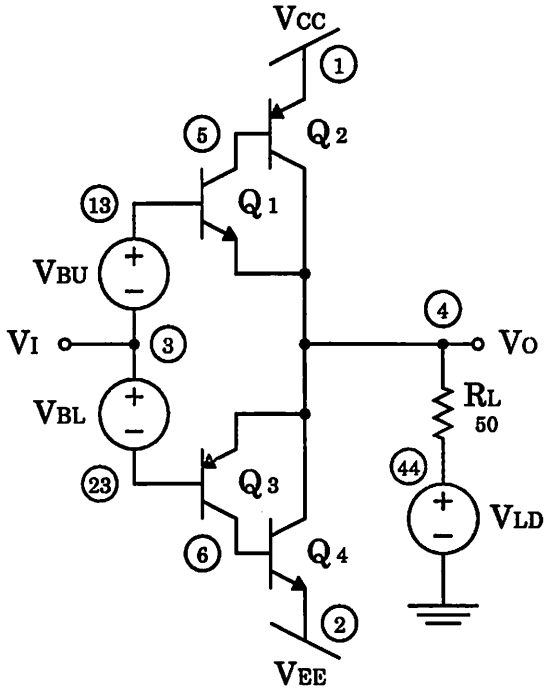


Figure 6.19: Schematic of push-pull complementary emitter follower

device structures, the design of a PPEF proceeds by determining appropriate emitter areas for the 4 devices and DC input bias voltages to meet the requirements of low distortion and high power gain.

PPEF designs based on idealized devices modeled with the Ebers-Moll equations and on nonideal devices modeled using the modified Gummel-Poon equations in SPICE have been investigated previously in [YOUN90]. The main conclusions of that work are summarized here. With identical idealized devices, low distortion is produced due to cancellation of even-order harmonics. However, with nonidentical devices, the even harmonics do not cancel. Low total harmonic distortion (THD) is thus achieved by paralleling many output devices to achieve low effective base resistance and high knee current, I_K. Compound devices reduce the loading of the PPEF stage on the previous voltage-driving stage by supplying large overall DC current gain approximately equal to the product of the β's of the individual transistors. The use of small input devices is suggested as a means to reduce the silicon die area consumed by the stage.

It is known that high-current effects are not well modeled by the modified Gummel-Poon model of SPICE [MAYA88], [ZARR89]. As an alternative, detailed phys-

ical simulations based on CIDER numerical models can be used to provide better modeling in this region of operation. Different conclusions as to the sizing of the various devices and setting of bias voltages and currents may be reached when using such models. In order to determine whether further investigation into the performance of the compound-device PPEF is warranted, CIDER simulations have been performed and the results compared to those obtained using SPICE compact models. Significant qualitative and quantitative differences in the results indicate that design criteria established using SPICE simulations need to be reevaluated using numerical device modeling.

6.4.1 Factors Affecting PPEF Performance

There are three main factors affecting the total harmonic distortion (THD) of the PPEF:

1. Crossover distortion as the upper and lower half circuits turn on and off.
2. Mismatches between the NPN and PNP devices.
3. Clipping.

Crossover distortion [PEDE91] dominates for low values of $\hat{V}_I$[5], and can be minimized by using large idling currents in Q_2 and Q_4. For moderate values of $\hat{V}_I$, THD is limited by nonlinearity in the output curve caused by unequal gain on the positive and negative output swings and bias-dependent resistances. The source of this nonlinearity is the differences in the compound devices in the upper and lower half-circuits. At very high values of $\hat{V}_I$, distortion is generated due to clipping of the output waveform. For example, the upper half circuit will not allow the output voltage to rise any higher than:

$$V_{O,max} = V_{CC} - V_{BE2} - V_{CEsat1} \tag{6.4}$$

before clipping begins. V_{CC} is the upper supply voltage, V_{BE2} is the DC base-emitter voltage of the PNP output transistor, and V_{CEsat1} is the collector-emitter saturation voltage of the NPN input transistor. This upper limit is determined by the requirement that Q_1 remain turned sufficiently on to supply base current to transistor Q_2. Because

[5] $\hat{V}_I$ is the amplitude of the sinusoidal input voltage.

clipping distortion is so severe, it places an upper bound on the magnitude of $\hat{V}_I$ that can be used to drive the PPEF. It also limits the maximum achievable power conversion efficiency.

The power gain of the circuit is the product of the voltage gain and the current gain. The voltage gain and current gains used must be the *large-signal* gains, which vary as a function of the input voltage amplitude. The voltage gain A_V is approximately constant at a value near 1. A rough formula for A_V is given by:

$$A_V = \frac{1}{1 + \frac{R_I^c}{\beta^c R_L}} \tag{6.5}$$

where R_I^c is an effective input resistance of the compound device, β^c is its effective current gain, and R_L is the load resistance. For low load resistance, e.g. $R_L = 50\ \Omega$, large currents flow, and the gain may drop somewhat due to finite input resistance and beta of the compound device. Large output devices should be used to minimize the parasitic base resistance and prevent premature falloff of beta. The current gain A_I of the PPEF is determined by the effective β's of the compound-device pairs, which are approximately equal to the product of the β's of the individual transistors. Typically, the output devices are sized such that they operate in a region where the transistor β drops rapidly with increasing I_C, as shown in Figure 6.20. As a result, total power gain drops as the input voltage amplitude $\hat{V}_I$ increases. This effect can be reduced by using larger output transistors that can supply more current before β begins to roll off. However, larger transistors consume more die area, so a design tradeoff is involved in determining the exact size needed for the output transistors. The input devices should be sized to supply this level of current without degrading current gain themselves. Finally, the input bias voltages should be set to provide sufficient idle current in the output transistors to reduce crossover distortion without unnecessarily increasing the standby power dissipation of the circuit.

6.4.2 Evaluation of PPEF Designs

The PPEF circuit in Figure 6.19 has been simulated using the CIDER 1^D bipolar models characterized in Section 6.2.1 and also using the SPICE modified Gummel-Poon bipolar model. The Gummel-Poon model parameters have been determined from a hand fit to the CIDER device characteristics. Reasonable fit is obtained in

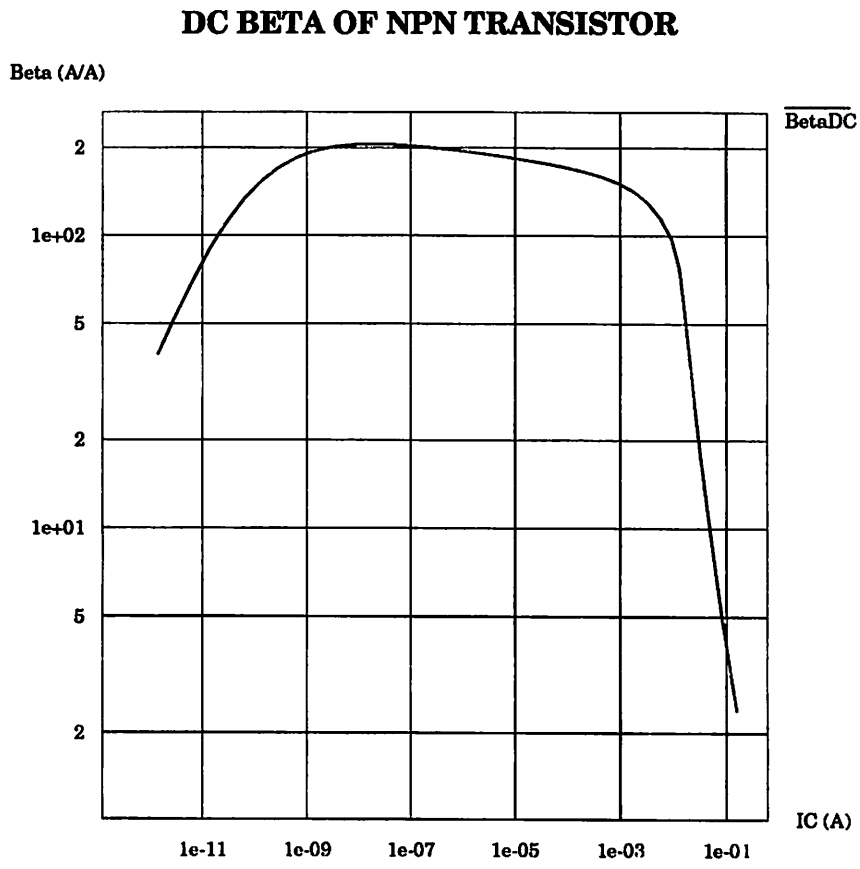


Figure 6.20: DC Beta of an NPN bipolar transistor. Beta rolls off at high I_C due to several high-level injection effects such as base pushout.

the moderate-current region, but large errors appear in the high-current region. The model input descriptions for CIDER and SPICE are listed in Appendix D[6] Note that the SPICE model parameters (especially the parasitic resistances) do not agree with the device parameters listed in Table 6.2, because they have been adjusted to provide a good match of the high-current behavior in the models.

Two PPEF designs have been simulated: one with equal size devices for all 4 transistors and one with adjusted device sizes to remove some of the limitations of the initial design. A unit-size device is set to have an emitter area of 1 μm $\times$ 40 μm which corresponds to the same area used in [YOUN90]. The load resistance is set to 50 Ω, which is a typical off-chip load value. Bias voltages of 0.7 V are applied to the two input transistors to set up standby current in the output transistors. The input voltage is driven with large-signal low-frequency sinusoidal inputs of magnitude up to 4.0 V to determine the THD and power gain of the stage. The peak load-current magnitude is therefore roughly 4.0V/50Ω = 80 mA, assuming an ideal voltage gain of 1. The input voltage has also been swept from V_{EE} to V_{CC} to investigate the linearity of the output characteristic.

For the initial design with equal-sized transistors, the quiescent bias current in Q_2 and Q_4 is determined from an operating point simulation of the circuit to be about 3.5 mA for SPICE and 2.9 mA for CIDER. These values are approximately 4% of the estimated peak load current. Using the values from Table 6.2, the knee current densities of the NPN and PNP devices are 8.0 mA and 3.6 mA, respectively. These values are roughly one-tenth to one-twentieth of the estimated peak load current. As a result, it can be anticipated that nonlinear β effects will result in low power gain at high input amplitude, and that the upper half-circuit containing the PNP device will be the limiting factor for clipping.

Figure 6.21 shows the output voltage of the PPEF as V_I is swept from V_{EE} = -5 V to V_{CC} = 5 V. For the CIDER results, the curve is almost a straight line unless $V_I <$ -3.8 V or $V_I >$ 3.5 V. The output voltage changes slope outside of these bounds due to clipping. The clipping points are determined by the V_{BE}'s of the output transistors

[6]The CIDER 1[D] models characterized in Section 6.2.1 and those used here differ in one respect. The effective base length of the PPEF models is 1.0 μm, rather than the 0.5 μm listed in the appendix. This increases the base resistance of the devices, and diminishes the base current in the high-current region. The SPICE models used here have been fit to the 1.0 μm CIDER models, and therefore do not agree as well as they could with the 0.5 μm models.

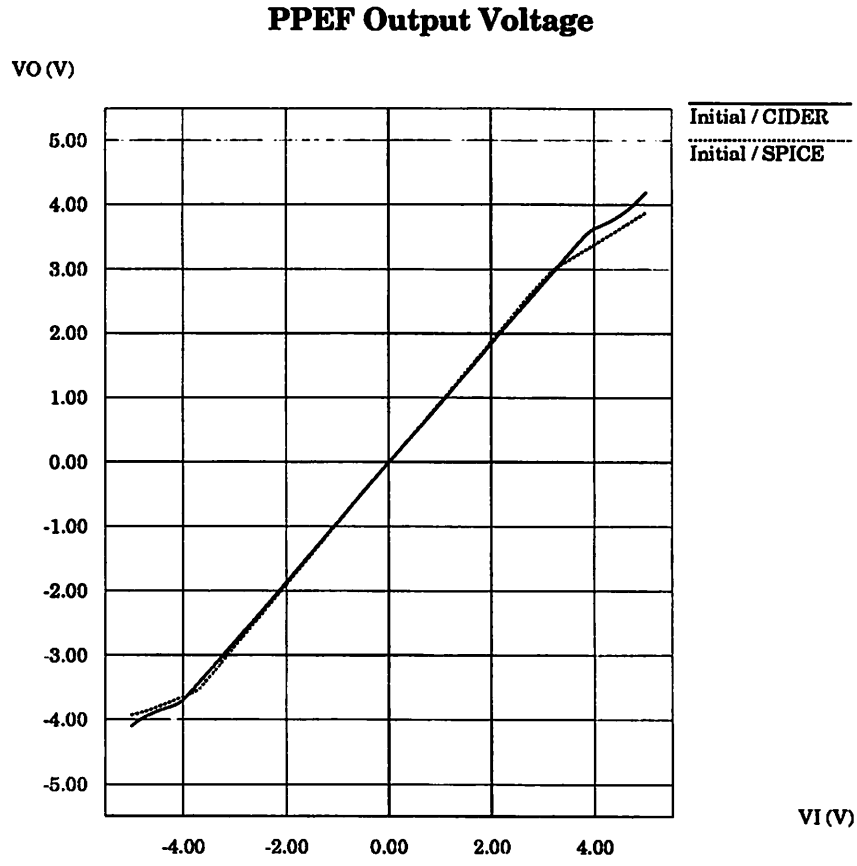


Figure 6.21: Output voltage of PPEF output stage. Clipping occurs for large positive and negative input voltages

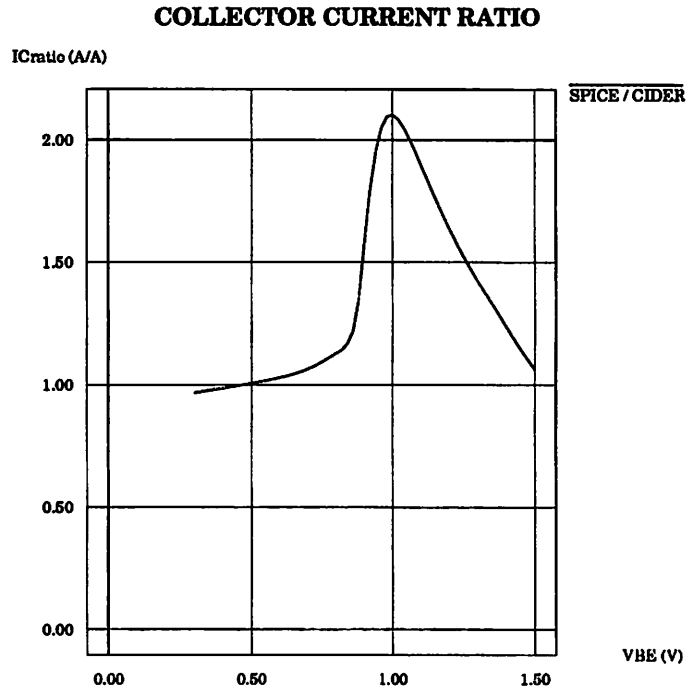


Figure 6.22: Ratio of collector currents from SPICE to CIDER

and the V_{CEsat}'s of the input transistors. Note that the SPICE and CIDER simulations clip at different points. This is because the collector currents of the two models do not agree at the same value of V_{BE} in the high-current region. The graph in Figure 6.22 demonstrates this by showing the ratio of the SPICE NPN collector current to that of CIDER as a function of V_{BE}. At the same level of collector current I_C (determined primarily by V_I and R_L), different V_{BE}'s are obtained from the two models.

In Figure 6.23, the slope (gain) of the output characteristic is plotted versus the input voltage. The average gain from SPICE is about 0.94, while that from the numerical-model simulation is about 0.92. Qualitatively, both curves show clipping at the edges of the central region, and a dip in the gain near the center due to crossover distortion. Note that as expected, clipping occurs sooner for positive input voltage due to the lower knee current of the PNP device. The CIDER results have multiple dips and peaks and an asymmetry about $V_I = 0$ V that should give rise to increased second harmonic distortion for small input amplitudes. In contrast, the SPICE curve has only the one dip due to crossover. The reasons for these differences are not known.

An improved design results if the output devices are made twice as large as

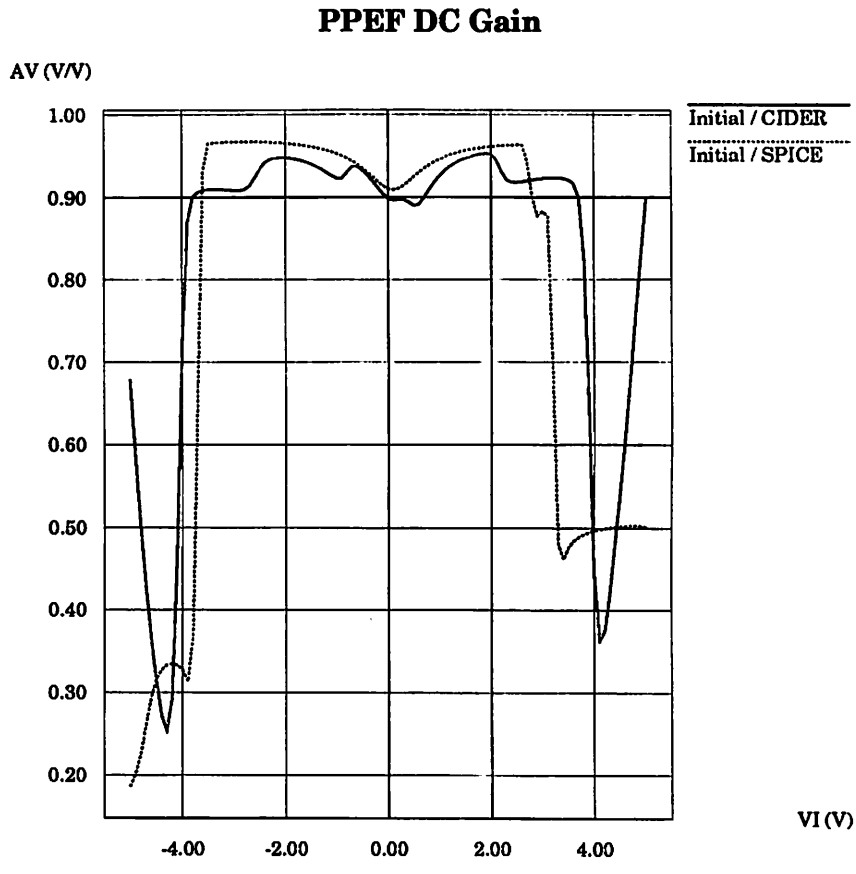


Figure 6.23: Gain of PPEF output stage. Ideal stage would have flat gain. Variations in gain result in harmonic distortion in output waveforms.

the input devices, and the PNP devices are made 2.5 times larger than NPN devices. This improves the current-carrying capacity of the output devices by delaying the onset of high-level injection, and better balances the knee currents of the NPN and PNP devices. As a result, power gain is improved and clipping occurs at a higher positive input voltage. Also, crossover distortion is reduced somewhat, since for fixed bias voltages of 0.7 V, there is more standby current in the larger output devices. Improved linearity and output voltage range have been verified by comparing the new gain curves to the results shown in Figure 6.23.

The total harmonic distortion and power gain of the original and improved designs have been measured from transient simulations of the PPEF. A low-frequency sinusoidal input voltage is applied and the output voltage and input current are stored. The low input frequency eliminates concerns about capacitive transients, so only a single cycle of the input needs to be simulated. The THD and power gain are determined from Fourier analyses of the resulting waveforms. The power gain is defined as:

$$A_P \equiv \frac{\hat{V_O}\hat{I_O}}{\hat{V_I}\hat{I_I}} = \frac{\hat{V_O}^2/R_L}{\hat{V_I}\hat{I_I}} \tag{6.6}$$

where $\hat{V_I}$ and $\hat{V_O}$ are the fundamental magnitudes of the input and output voltages and $\hat{I_I}$ and $\hat{I_O}$ are the fundamental magnitudes of the input and output currents. $\hat{V_O}$ and $\hat{I_I}$ must be measured, whereas $\hat{V_I}$ is known *a priori*.

The THD is shown in Figure 6.24 for the four different cases: SPICE - initial design, CIDER - initial design, SPICE - improved design, and CIDER - improved design. In the original design, both CIDER and SPICE show increased distortion at large $\hat{V_I}$ due to clipping. This effect is less apparent in the improved-design simulations. For the remaining values of $\hat{V_I}$, the improved design reduces the THD by about a factor of two. Qualitatively, the differences in the large-signal output voltage characteristics between SPICE and CIDER show up as differences in the THD. The CIDER simulations reach a peak THD between 0.8 V and 1.4 V $\hat{V_I}$ and then start to diminish. The SPICE simulations show increasing THD until just before the onset of clipping. SPICE therefore appears to overestimate the THD in this region. Comparison to experimental data would be needed to determine which set of simulations (CIDER or SPICE) gives more accurate results.

In Figure 6.25, the power gain of the PPEF is plotted for $\hat{V_I}$ up to 4.0 V. All four simulations give substantially different results. The lower 2 curves are from the

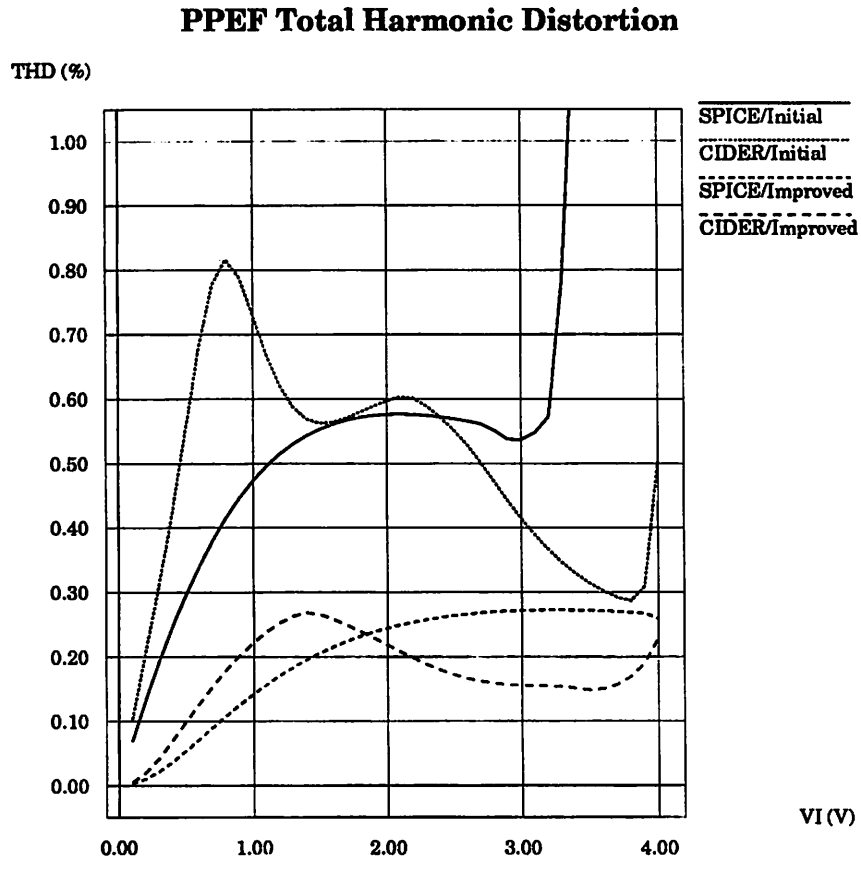


Figure 6.24: Total harmonic distortion (THD) of PPEF designs. Rapid increase in THD for large $\hat{V}_I$ is due to clipping.

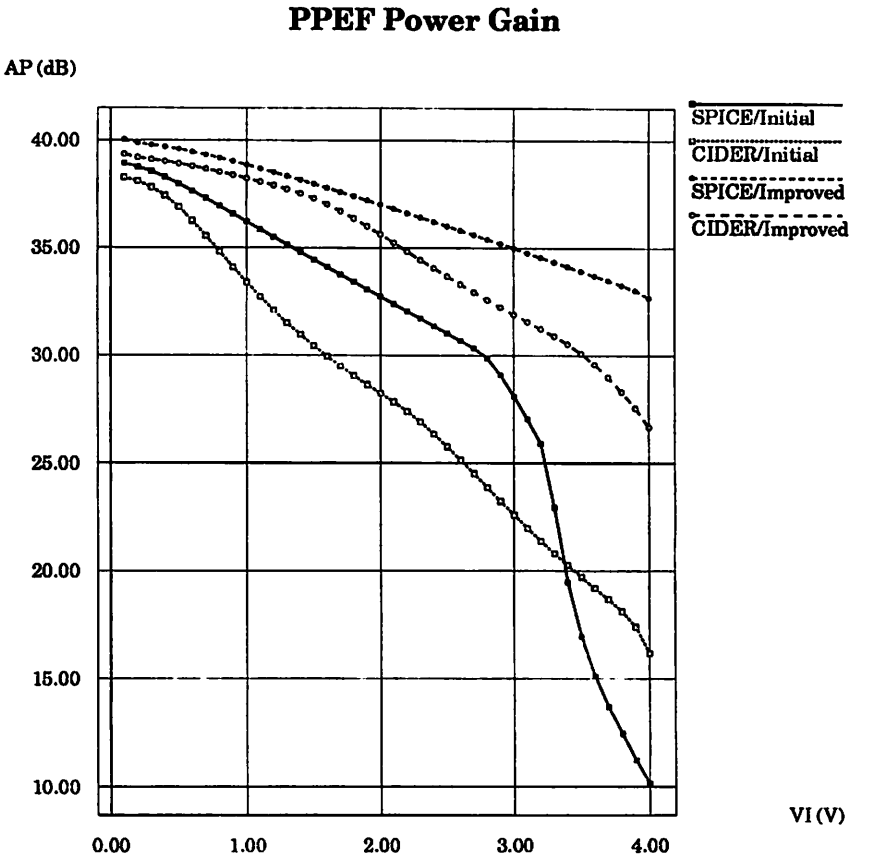


Figure 6.25: Power gain of PPEF designs. Gain falls off due to high-current beta rolloff.

Design/Model	A_V (V/V)	A_I (A/A)	A_P (dB)	THD (%)	$\hat{I}_O$ (mA)	$\hat{I}_I$ (mA)
Initial/SPICE	0.915	11.2	10.1	3.6	73.2	6.5
Initial/CIDER	0.923	44.9	16.2	0.50	73.9	1.6
Improved/SPICE	0.967	1920	32.7	0.26	77.4	0.040
Improved/CIDER	0.957	486	26.7	0.23	76.5	0.16

Table 6.6: Performance summary of PPEF designs. Initial design has area ratios for Q_1 to Q_4 of 1:1:1:1. Improved design has area ratios 1:5:2.5:2. Unit device has emitter area of 1μm $\times$ 40 μm.

original design which has less power gain. The CIDER results are consistently lower than those from the SPICE Gummel-Poon model up until about 3.2 V. At that point, clipping in the SPICE simulation rapidly reduces the power gain. For the improved design, a similar relationship between the numerical and compact model results is also seen. However, in both cases the power gain has increased significantly with respect to the initial design. The falloff of power gain with increasing $\hat{V}_I$ has also been reduced. Another observation is that with numerical modeling the falloff is more rapid than with compact device modeling[7]. This is most likely due to the differences in high-current modeling between the CIDER and SPICE models.

Several measures of the performance of the PPEF designs are assembled in Table 6.6. For $\hat{V}_I$ = 4.0 V and $\pm$ 5.0 V supplies, the CIDER simulations of the improved design predict an output amplitude of 3.83 V, power gain of 26.7 dB, total harmonic distortion of 0.23 %, peak load current of 76.5 mA, and input demand of 160 μA.

6.4.3 Two-Dimensional Simulations of the PPEF

So far only one-dimensional numerical device models have been used in CIDER to evaluate the PPEF designs. However, the high-current behavior of bipolar transistors is affected by inherently two-dimensional phenomena such as emitter current crowding and lateral base pushout. In order to determine whether the simulation results change significantly in the presence of these 2^D effects, simulations employing the coarse-mesh 2^D bipolar transistor models mentioned in Section 6.2.1 have been performed. Because 2^D simulations are computationally intensive, only the improved

[7]The rate of falloff is roughly measured by finding the input voltage where the power gain has dropped to 1/2 its maximum value.

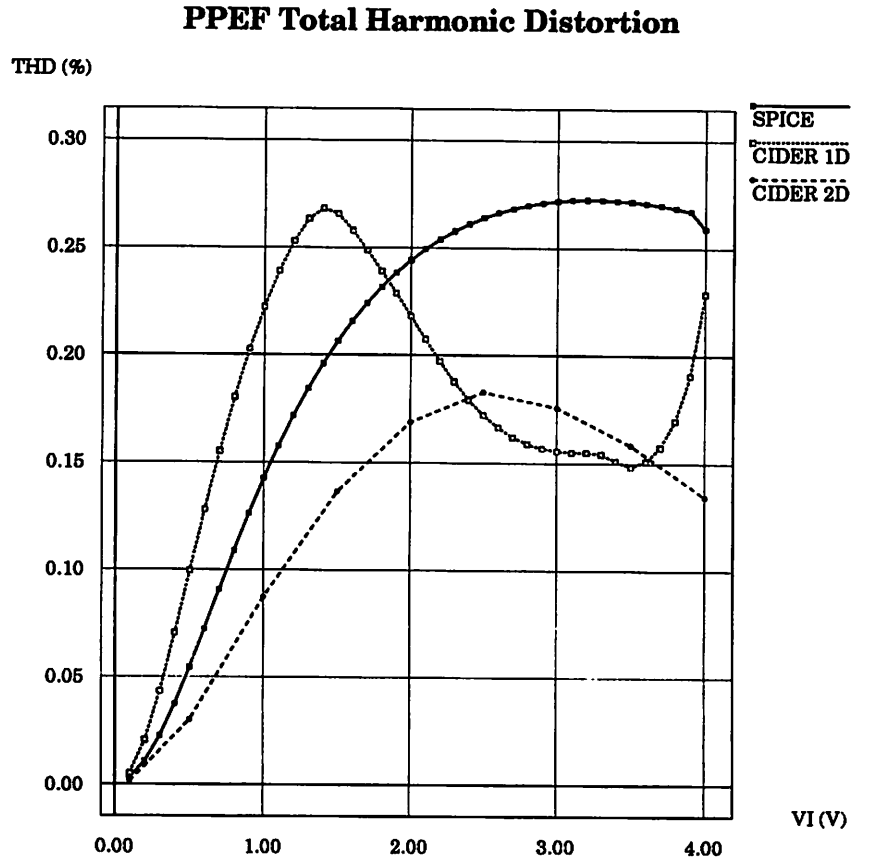


Figure 6.26: Comparison of THD predictions from different models. The 2^D simulations show less distortion.

PPEF design has been simulated, and fewer input voltage amplitudes $\hat{V}_I$ have been sampled. In addition, the simulations have been performed on 4-node subcubes of the iPSC/860 described in Chapter 5 in order to reduce the simulation run times by exploiting parallelism. Simulations of the 2^D model using CIDER in a device-simulation mode indicate differences between the 1^D and 2^D behavior, especially at high-current levels. Differences in the simulated performance of the PPEF circuit can also be expected.

In Figure 6.26, the THD obtained from the 2^D simulations is compared to that obtained for the improved design when using 1^D bipolar models or SPICE compact models. In Figure 6.27, the power gain for these 3 cases is plotted. In both figures, major discrepancies are observed. The 2^D simulations predict lower average THD and higher power gain. Further investigation is needed to identify the source of these

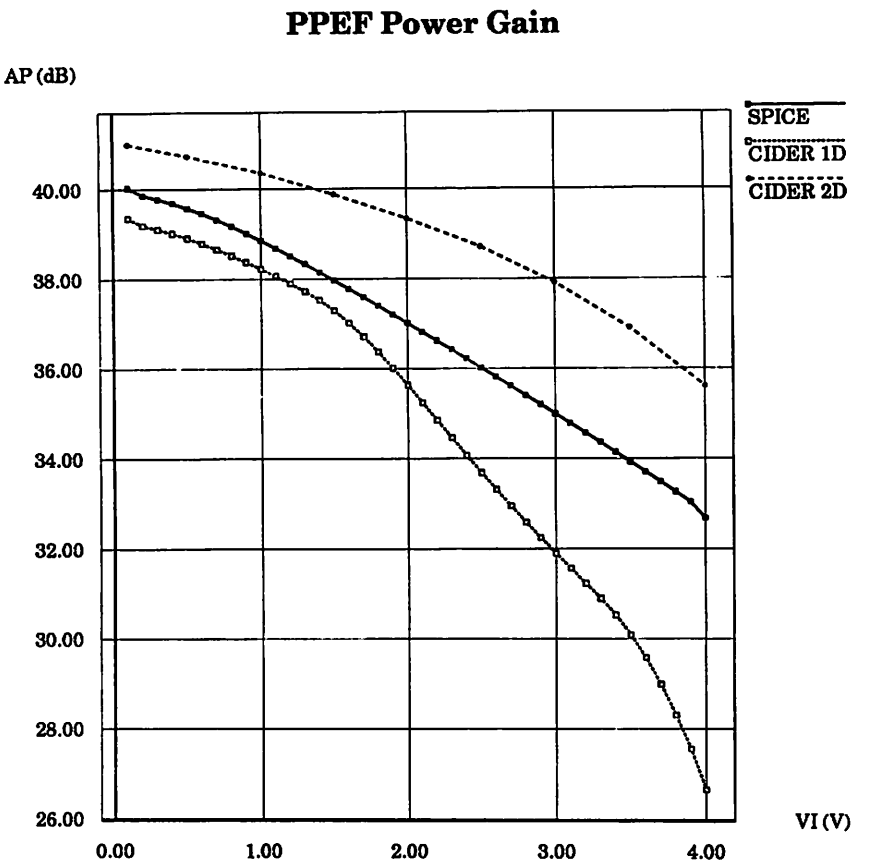


Figure 6.27: Comparison of power gain predictions from different models. The 2^D simulations show greater power gain.

discrepancies. One contributing factor may be the level of standby current in the output transistors. All three models predict different levels of standby current for the given 0.7 V bias voltages, and therefore can be expected to show different THD from crossover distortion. A possible direction for future research would be to adjust the input bias voltages to achieve identical standby current in all 3 simulations. However, since all 3 models are supposedly simulating the same device in the real world, they should produce the same results without the need for such adjustment. At least 2 and perhaps all 3 models are improperly simulating the behavior of real PPEF circuits.

Given that the SPICE Gummel-Poon and CIDER one- and two-dimensional numerical models all predict different levels of THD and power gain, it is difficult to conclude that any of the models provides more accurate simulation results. Although physically motivated reasoning would suggest that the 2^D numerical models are most accurate, this assertion needs to be tested by carefully comparing simulation results to measurements of actual PPEF implementations. In the mean time, PPEF designs based solely on SPICE simulations may fail to meet their specifications due to potentially erroneous simulation results. As a precaution, double-checking of designs using one- or two-dimensional CIDER simulations is warranted.

6.5 Summary

The serial and parallel versions of CIDER have been demonstrated using three examples. In the first application, the expanded capabilities of CIDER compared to its predecessor CODECS are shown. A hypothetical 1.0 μm CBiCMOS process has been characterized. Advanced technology devices such as polysilicon-emitter bipolar transistors and LDD MOSFETs are incorporated into the process. The new physical models of CIDER allow these devices to be accurately modeled. Evaluation of a 7-stage CMOS ring oscillator is possible if the Intel iPSC/860 version of CIDER is used.

The second application studies the gain of three different MOS amplifiers. The simplest circuit is a MOS inverter with an ideal load, the next is an NMOS source-coupled pair with an active PMOS current-mirror load, and the most complex circuit is an 8-transistor two-stage CMOS opamp. The parallel version of CIDER makes analysis of the 8-transistor circuit tractable. Simulation results obtained using the SPICE BSIM model have been compared to results obtained using CIDER numerical models. The

poor modeling of output resistance in the BSIM model gives rise to nonphysical gain predictions, as demonstrated by comparison to the CIDER results.

In the third and final example, the large-signal performance of the push-pull complementary emitter-follower output stage is investigated. In this circuit, compound bipolar devices are used to provide high load-current drivability while maintaining low input demand. For large-amplitude inputs, the output transistors operate in high-level injection. It is well known that the SPICE modified Gummel-Poon bipolar transistor model does not model high-current effects very well. CIDER simulations employing one- and two-dimensional numerical models have been used to study the accuracy of SPICE-model-based simulations. Simulation results show that large qualitative and quantitative differences exist between compact-model results and numerical-model results. These differences call into question design criteria for PPEFs developed using SPICE simulations.

Chapter 7

Conclusions

Used properly, a mixed-level circuit and device simulator can be a valuable addition to the set of tools available to the designer of integrated circuits and devices. However, the degree to which this is true depends on the capabilities and limitations of a particular implementation of mixed-level simulation. This dissertation addresses several topics which all relate to the same central question: how should existing mixed-level circuit and device simulators evolve in the future to become more effective tools for IC design?

Issues of concern for a mixed-level simulator are the same as for any other large, complex piece of software:

Performance An adequate level of performance must be sustained to support the routine tasks of IC design.

Reliability The program should reliably produce accurate results for any well-posed simulation problem. In addition, an electrical IC simulator must contain appropriate models for the physical effects that influence device and circuit behavior.

Utility It should be easy to both generate input for the program and evaluate the simulation results. In addition, a general-purpose simulator should support a wide range of analysis capabilities.

Portability The program must be capable of running on a wide variety of computing systems. In mixed-level simulation, where performance requirements are stringent, multiple parallel computing systems should also be supported.

The recently developed circuit and device simulator CODECS [MAYA88] has been used as the basis for a new, parallel mixed-level simulator CIDER. CIDER extends the capabilities of CODECS in order to address each of the four major issues of concern identified above. In order to improve reliability, the basic proven algorithms of CODECS have been left untouched as much as possible when moving to a parallel computing environment. These algorithms have been reviewed, and several extensions to improve the modeling of important physical effects and the ease-of-use of the user interface have been introduced. A complete manual for CIDER with several examples is provided as Appendix A.

A detailed performance analysis of CIDER has been performed to identify the major computational bottlenecks of mixed-level simulation. In all cases considered, evaluation of the numerically modeled elements accounts for over 99% of the total time used. Experiments have been performed to measure the resources used by a set of simple test problems. These measurements are used to create models for CPU and memory usage that can predict the resource requirements in other situations. A set of benchmark circuits has been run on 5 different RISC architecture computing systems to establish the serial performance of CIDER on actual circuit examples. For these systems with clock rates ranging between 25 and 50 MHz, sustained performance for the benchmark circuits averages from between 0.7 and 3.0 MFLOP/S. Estimates of the resources needed in several simulation scenarios show that this level of performance is well above that needed for small problems, an order of magnitude too slow for a large but still manageable problem, and 4 orders of magnitude too slow for a problem at the upper limits of what might be encountered in IC cell design.

The computational bottleneck of mixed-level simulation has been addressed by investigating the possibilities for the use of scalable, high-performance distributed-memory multicomputers. Three levels of parallelism are identified that can be exploited by the multiple processors in such a system. At the design-level, tasks consist of individual simulation jobs. At the circuit-level, the major tasks are the evaluations of the numerically modeled elements of a circuit. At the device-level, each processor is assigned a task that roughly corresponds to a portion of the semiconductor device being simulated. Existing techniques for exploiting each of these levels of parallelism have been reviewed, and extensions that combine parallelism from more than one level have been introduced. In particular, an algorithm is proposed for combining

parallelism at the circuit and device levels in a single program. Experiments with a simulated-annealing-based partitioning program indicate that in some cases additional speedups may be achievable compared to a simpler implementation based solely on exploiting parallelism at the circuit level. However, the proposed algorithm requires several software components that are not readily available on present parallel computing systems.

In order to meet current needs for performance enhancement, an implementation of a one-level parallel model evaluation algorithm has been developed for distributed-memory multicomputers. This implementation has been successfully tested on two different computing systems: a scalable, hypercube supercomputer and a nonscalable cluster of engineering workstations. The primary advantage of this approach is the relative simplicity with which it can be implemented and maintained. By leaving the cores of the component circuit and device simulators virtually untouched, the parallel mixed-level simulator can be quickly and easily upgraded as modifications are made to the component simulators. Benchmark testing of the simulator on the two systems reveals that it is possible to achieve good speedup and efficiency using this approach. In general, the hypercube implementation outperforms the cluster implementation, even after taking into account the faster compute nodes of the hypercube. However, in certain special cases, the speedups obtained are very similar. Unfortunately, the one-level approach has a number of limitations that prevent the full power of the multicomputer from being used effectively. Chief among these is a ceiling on the available parallelism equal to the number of numerically modeled devices in the circuit. The other limitations all reduce the speedup achieved below this upper bound. While options exist for minimizing the impact of most of these limitations, the unpredictable nature of imbalances caused by latent devices appears at this point to be an insurmountable difficulty.

Despite the limited success of the parallel implementation, it is still an improvement over the best serial algorithm. Examples have been provided that use this improved performance to study circuits containing multiple, two-dimensional numerically modeled devices. A hypothetical 1.0 μm CBiCMOS process has been characterized using CIDER. Parallel simulation allows the stage delay of a 7-stage CMOS ring oscillator to be determined. The gain of several MOS amplifiers has been simulated using both SPICE compact models and CIDER numerical MOS models. Analysis of the

results shows that the SPICE models predict incorrect behavior, whereas the CIDER numerical models result in qualitatively correct behavior. Parallel CIDER simulations allow an 8-transistor two-stage CMOS operational amplifier to be simulated. Finally, a compound-device push-pull complementary emitter-follower IC output stage has been studied. Results from simulations using CIDER one- and two-dimensional numerical bipolar models and from using the SPICE modified Gummel-Poon model do not agree. Design criteria developed based on SPICE simulations need to be reevaluated based on this new information. The expensive 2^D transient simulations have been performed on the hypercube supercomputer.

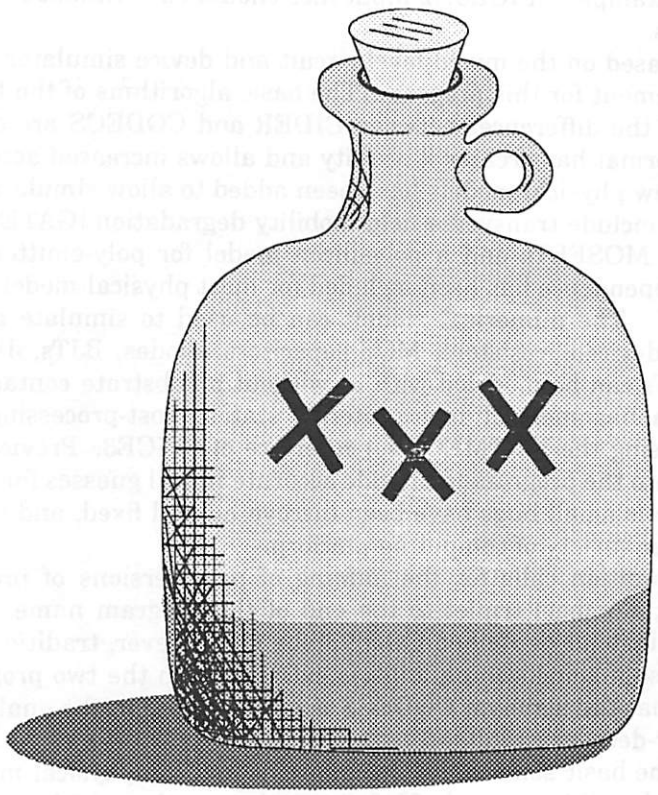
Future work should continue to address the four major issues of concern. The existing implementation is believed to have sufficient accuracy, utility and performance to enable the design of small cells such as opamps, comparators, and logic gates. This belief should be checked by tuning the physical models in CIDER to an actual IC process and then designing, manufacturing and testing several such small circuits.

If problems arise during this process, several options exist for improving the current implementation. Performance may be enhanced by implementing the multi-level model-evaluation algorithm described in Chapter 4. Alternatively, or in concert, the rectangular-mesh-based device simulator could be replaced by one employing a more general and efficient meshing scheme such as used in PISCES [PINT85].

Although mixed-level circuit and device simulation has been available for some time [ENGL82], only recently has it begun to attract significant attention in the TCAD community. TCAD vendors have integrated this capability into their lines of product offerings [TMA91]. However, someday in the future it may also have a place alongside circuit simulators in the ICCAD toolbox. If so, parallel computers will surely have a role in making this happen.

Appendix A

CIDER User's Manual



The CIDER User's Manual that follows is organized as a series of individual UNIX-style manual pages. At the end of the manual, several examples illustrating the use of CIDER are given.

NAME

INTRODUCTION – Overview of CIDER's features / capabilities

DESCRIPTION

CIDER is a mixed-level circuit and device simulator that provides a direct link between technology parameters and circuit performance. A mixed-level circuit and device simulator can provide greater simulation accuracy than a stand-alone circuit or device simulator by numerically modeling the critical devices in a circuit. Compact models can be used for noncritical devices.

CIDER couples the latest version of SPICE3 (version 3F.2) [JOHN92] to an internal C-based device simulator, DSIM. SPICE3 provides circuit analyses, compact models for semiconductor devices, and an interactive user interface. DSIM provides accurate, one- and two-dimensional numerical device models based on the solution of Poisson's equation, and the electron and hole current-continuity equations. DSIM incorporates many of the same basic physical models found in the the Stanford two-dimensional device simulator PISCES [PINT85]. Input to CIDER consists of a SPICE-like description of the circuit and its compact models, and PISCES-like descriptions of the structures of numerically modeled devices. As a result, CIDER should seem familiar to designers already accustomed to these two tools. For example, SPICE3F.2 input files should run without modification, producing identical results.

CIDER is based on the mixed-level circuit and device simulator CODECS [MAYA88], and is a replacement for this program. The basic algorithms of the two programs are the same. Some of the differences between CIDER and CODECS are described below. The CIDER input format has greater flexibility and allows increased access to physical model parameters. New physical models have been added to allow simulation of state-of-the-art devices. These include transverse field mobility degradation [GATE90] that is important in scaled-down MOSFETs and a polysilicon model for poly-emitter bipolar transistors. Temperature dependence has been included for most physical models over the range from -50°C to 150°C. The numerical models can be used to simulate all the basic types of semiconductor devices: resistors, MOS capacitors, diodes, BJTs, JFETs and MOSFETs. BJTs and JFETs can be modeled with or without a substrate contact. Support has been added for the management of device internal states. Post-processing of device states can be performed using the NUTMEG user interface of SPICE3. Previously computed states can be loaded into the program to provide accurate initial guesses for subsequent analyses. Finally, numerous small bugs have been discovered and fixed, and the program has been ported to a wider variety of computing platforms.

Berkeley tradition calls for the naming of new versions of programs by affixing a (number, letter, number) triplet to the end of the program name. Under this scheme, CIDER should instead be named CODECS2A.1. However, tradition has been broken in this case because major incompatibilities exist between the two programs and because it was observed that the acronym CODECS is already used in the analog design community to refer to coder-decoder circuits.

Details of the basic semiconductor equations and the physical models used by CIDER are not provided in this manual. Unfortunately, no other single source exists which describes all of the relevant background material. Comprehensive reviews of device simulation can be found in [PINT90] and the book [SELB84]. CODECS and its inversion-layer mobility model are described in [MAYA88] and [GATE90], respectively. PISCES and its models are described in [PINT85]. Temperature dependences for the PISCES models used by CIDER are available in [SOLL90].

SYSTEM REQUIREMENTS

The program has been run successfully on the following operating system / hardware combinations: (Ultrix 4, RISC), (SunOS 4, Sun4), (AIX 3, RS/6000), (UNIX SVR3, iPSC/860 node), (HPUX 8, 9000/700). Compatibility with other computer systems has not been tested.

NAME

SPECIFICATION – Overview of numerical-device specification

DESCRIPTION

The input to CIDER consists of a SPICE-like description of a circuit, its analyses and its compact device models, and PISCES-like descriptions of numerically analyzed device models. For a description of the SPICE input format, consult the SPICE3 User's Manual [JOHN92].

To simulate devices numerically, two types of input must be added to the input file. The first is a model description in which the common characteristics of a device class are collected. In the case of numerical models, this provides all the information needed to construct a device cross-section, such as, for example, the doping profile. The second type of input consists of one or more element lines that specify instances of a numerical model, describe their connections to the rest of the circuit, and provide additional element-specific information such as device layout dimensions and initial bias information.

The format of a numerical device model description differs from the standard approach used for SPICE3 compact models. It begins the same way with one line containing the .MODEL keyword followed by the name of the model, device type and modeling level. However, instead of providing a single long list of parameters and their values, numerical model parameters are grouped onto **cards**. Each type of card has its own set of valid parameters. In all cases, the relative ordering of different types of cards is unimportant. However, for cards of the same type (such as mesh-specification cards), their order in the input file can be important in determining the device structure.

Each card begins on a separate line of the input file. In order to let CIDER know that card lines are continuations of a numerical model description, each must begin with the continuation character, '+'. If there are too many parameters on a given card to allow it fit on a single line, the card can be continued by adding a second '+' to the beginning of the next line. However, the name and value of a parameter should always appear on the same line.

Several features are provided to make the numerical model format more convenient. Blank space can follow the initial '+' to separate it from the name of a card or the card continuation '+'. Blank lines are also permitted, as long as they also begin with an initial '+'. Parentheses and commas can be used to visually group or separate parameter definitions. In addition, while it is common to add an equal sign between a parameter and its value, this is not strictly necessary.

The name of any card can be abbreviated, provided that the abbreviation is unique. Parameter name abbreviations can also be used if they are unique in the list of a card's parameters. Numeric parameter values are treated identically as in SPICE3, so exponential notation, engineering scale factors and units can be attached to parameter values: `tau=10ns, nc=3.0e19cm^-3`. In SPICE3, the value of a FLAG model parameter is changed to TRUE simply by listing its name on the model line. In CIDER, the value of a numerical model FLAG parameter can be turned back to FALSE by preceding it by a caret '^'. This minimizes the amount of input change needed when features such as debugging are turned on and off. In certain cases it is necessary to include filenames in the input description and these names may contain capital letters. If the filename is part of an element line, the input parser will convert these capitals to lowercase letters. To protect capitalization at any time, simply enclose the string in double quotes '"'.

The remainder of this manual describes how numerically analyzed elements and models can be used in CIDER simulations. The manual consists of three parts. First, all of the

model cards and their parameters are described. This is followed by a section describing the three basic types of numerical models and theier corresponding element lines. In the final section, several complete examples of CIDER simulations are presented.

Several conventions are used in the card descriptions. In the card synopses, the name of a card is followed by a list of parameter classes. Each class is represented by a section in the card parameter table, in the same order as it appears in the synopsis line. Classes which contain optional parameters are surrounded by brackets: [...]. Sometimes it only makes sense for a single parameter to take effect. (For example, a material can not simultaneously be both Si and SiO_2.) In such cases, the various choices are listed sequentially, separated by colons. The same parameter often has a number of different acceptable names, some of which are the listed in the parameter tables.[1] These aliases are separated by vertical bars: '|'. Finally, in the card examples, the model continuation pluses have been removed from the card lines for clarity's sake.

EXAMPLES

The model description for a two-dimensional numerical diode might look something like what follows. This example demonstrates many of the features of the input fromat described above. Notice how the .MODEL line and the leading pluses form a border around the model description:

```
.MODEL M_NUMERICAL NUMD LEVEL=2
+ cardname1 number1=val1 (number2 val2), (number3 = val3)
+ cardname2 number1=val1 string1 = name1
+
+ cardname3 number1=val1, flag1, ^flag2
+ + number2=val2, flag3
```

The element line for an instance of this model might look something like the following. Double quotes are used to protect the filename from decapitalization:

```
d1 1 2 M_NUMERICAL area=100pm^2 ic.file = "diode.IC"
```

[1]Some of the possibilities are not listed in order to shorten the lengths of the parameter tables. This makes the use of parameter abbreviations somewhat troublesome since an unlisted parameter may abbreviate to the same name as one that is listed. CIDER will produce a warning when this occurs. Many of the undocumented parameter names are the PISCES names for the same parameters. The adventurous soul can discover these names by delving through the 'cards' directory of the source code distribution and looking for the C parameter tables.

NAME

BOUNDARY, INTERFACE – Specify properties of a domain boundary or the interface between two boundaries

SYNOPSIS

boundary domain [bounding-box] [properties]
interface domain neighbor [bounding-box] [properties]

DESCRIPTION

The boundary and interface cards are used to set surface physics parameters along the boundary of a specified domain. Normally, the parameters apply to the entire boundary, but there are two ways to restrict the area of interest. If a neighboring domain is also specified, the parameters are only set on the interface between the two domains. In addition, if a bounding box is given, only that portion of the boundary or interface inside the bounding box will be set.

If a semiconductor-insulator interface is specified, then an estimate of the width of any inversion or accumulation layer that may form at the interface can be provided. If the surface mobility model (cf. **models** card) is enabled, then the model will apply to all semiconductor portions of the device within this estimated distance of the interface. If a point lies within the estimated layer width of more than one interface, it is assumed to belong to the interface specified first in the input file. If the layer width given is less than or equal to zero, it is automatically replaced by an estimate calculated from the doping near the interface. As a consequence, if the doping varies so will the layer width estimate.

Each edge of the bounding box can be specified in terms of its location or its mesh-index in the relevant dimension, or defaulted to the respective boundary of the simulation mesh.

PARAMETERS

Name	*Type*	*Description*
Domain	Integer	ID number of primary domain
Neighbor	Integer	ID number of neighboring domain
X.Low	Real	Lowest X location of bounding box, (μm)
: IX.Low	Integer	Lowest X mesh-index of bounding box
X.High	Real	Highest X location of bounding box, (μm)
: IX.High	Integer	Highest X mesh-index of bounding box
Y.Low	Real	Lowest Y location of bounding box, (μm)
: IY.Low	Integer	Lowest Y mesh-index of bounding box
Y.High	Real	Highest Y location of bounding box, (μm)
: IY.High	Integer	Highest Y mesh-index of bounding box
Qf	Real	Fixed interface charge, (C/cm^2)
SN	Real	Surface recombination velocity - electrons, (cm/s)
SP	Real	Surface recombination velocity - holes, (cm/s)
Layer.Width	Real	Width of surface layer, (μm)

EXAMPLES

The following shows how the surface recombination velocities at an Si-SiO_2 interface might be set:

```
interface dom=1 neigh=2 sn=1.0e4 sp=1.0e4
```

In a MOSFET with a 2.0 μm gate width and 0.1 μm source and drain overlap, the surface channel can be restricted to the region between the metallurgical junctions and within 100 Å (0.01 μm) of the interface:

```
interface dom=1 neigh=2 x.l=1.1 x.h=2.9 layer.w=0.01
```

The inversion layer width in the previous example can be automatically determined by setting the estimate to 0.0:

```
interface dom=1 neigh=2 x.l=1.1 x.h=2.9 layer.w=0.0
```

SEE ALSO

domain, contact, mobility, models

NAME

COMMENT – Add explanatory comments to a device definition

SYNOPSIS

comment [text]
* [text]
$ [text]
# [text]

DESCRIPTION

Annotations can be added to a device definition using the comment card. All text on a comment card is ignored. Several popular commenting characters are also supported as aliases: '*' from SPICE, '$' from PISCES, and '#' from UNIX shell scripts.

EXAMPLES

A SPICE-like comment is followed by a PISCES-like comment and shell script comment:

```
* CIDER and SPICE would ignore this input line
$ CIDER and PISCES would ignore this, but SPICE wouldn't
# CIDER and UNIX Shell scripts would ignore this input line
```

NAME

CONTACT – Specify properties of an electrode

SYNOPSIS

contact number [workfunction]

DESCRIPTION

The properties of an electrode can be set using the contact card. The only changeable property is the workfunction of the electrode material and this only affects contacts made to an insulating material. All contacts to semiconductor material are assumed to be ohmic in nature.

PARAMETERS

Name	*Type*	*Description*
Number	Integer	ID number of the electrode
Workfunction	Real	Workfunction of electrode material, (eV)

EXAMPLES

The following shows how the workfunction of the gate contact of a MOSFET might be changed to a value appropriate for a P^+ polysilicon gate:

```
contact num=2 workf=5.29
```

SEE ALSO

electrode, material

NAME

DOMAIN, REGION – Identify material-type for section of a device

SYNOPSIS

domain number material [position]
region number material [position]

DESCRIPTION

A device is divided into one or more rectilinear domains, each of which has a unique identification number and is composed of a particular material.

Domain (aka region) cards are used to build up domains by associating a material type with a box-shaped section of the device. A single domain may be the union of multiple boxes. When multiple domain cards overlap in space, the one occurring last in the input file will determine the ID number and material type of the overlapped region.

Each edge of a domain box can be specified in terms of its location or mesh-index in the relevant dimension, or defaulted to the respective boundary of the simulation mesh.

PARAMETERS

Name	*Type*	*Description*
Number	Integer	ID number of this domain
Material	Integer	ID number of material used by this domain
X.Low	Real	Lowest X location of domain box, (μm)
: IX.Low	Integer	Lowest X mesh-index of domain box
X.High	Real	Highest X location of domain box, (μm)
: IX.High	Integer	Highest X mesh-index of domain box
Y.Low	Real	Lowest Y location of domain box, (μm)
: IY.Low	Integer	Lowest Y mesh-index of domain box
Y.High	Real	Highest Y location of domain box, (μm)
: IY.High	Integer	Highest Y mesh-index of domain box

EXAMPLES

Create a 4.0 μm wide by 2.0 μm high domain out of material #1:

```
domain num=1 material=1 x.l=0.0 x.h=4.0 y.l=0.0 y.h=2.0
```

The next example defines the two domains that would be typical of a planar MOSFET simulation. One occupies all of the mesh below y = 0 and the other occupies the mesh above y = 0. Because the x values are left unspecified, the low and high x boundaries default to the edges of the mesh:

```
domain n=1 m=1 y.l=0.0
domain n=2 m=2 y.h=0.0
```

SEE ALSO

x.mesh, material

NAME

DOPING – Add dopant to regions of a device

SYNOPSIS

doping [domains] profile-type [lateral-profile-type] [axis] [impurity-type] [constant-box] [profile-specifications]

DESCRIPTION

Doping cards are used to add impurities to the various domains of a device. Initially each domain is dopant-free. Each new doping card creates a new doping profile that defines the dopant concentration as a function of position. The doping at a particular location is then the sum over all profiles of the concentration values at that position. Each profile can be restricted to a subset of a device's domains by supplying a list of the desired domains. Otherwise, all domains are doped by each profile.

A profile has uniform concentration inside the constant box. Outside this region, it varies according to the primary and lateral profile shapes. In 1^D devices the lateral shape is unused and in 2^D devices the y-axis is the default axis for the primary profile. Several analytic functions can be used to define the primary profile shape. Alternatively, empirical or simulated profile data can be extracted from a file. For the analytic profiles, the doping is the product of a profile function (e.g. Gaussian) and a reference concentration, which is either the constant concentration of a uniform profile, or the peak concentration for any of the other functions. If concentration data is instead taken from an ASCII file containing a list of location-concentration pairs or a SUPREM3 exported file, the name of the file must be provided. If necessary, the final concentration at a point is then found by multiplying the primary profile concentration by the value of the lateral profile function at that point. Empirical profiles must first be normalized by the value at 0.0 to provide a usable profile function. Alternatively, the second dimension can be included by assigning the same concentration to all points equidistant from the edges of the constant box. The contours of the profile are then circular.

Unless otherwise specified, the added impurities are assumed to be N type. However, the name of a specific dopant species is needed when extracting concentration information for that impurity from a SUPREM3 exported data file.

Several parameters are used to adjust the basic shape of a profile function so that the final, constructed profile matches the doping profile in the real device. The constant box region should coincide with a region of constant concentration in the device. For uniform profiles its boundaries default to the mesh boundaries. For the other profiles the constant box starts as a point and only acquires width or height if both the appropriate edges are specified. The location of the peak of the primary profile can be moved away from the edge of the constant box. A positive location places the peak outside the constant box (cf. Fig. A.1), and a negative value puts it inside the constant box (cf. Fig. A.2). The concentration in the constant box is then equal to the value of the profile when it intersects the edge of the constant box. The argument of the profile function is a distance expressed in terms of the characteristic length (by default equal to 1μm). The longer this length, the more gradually the profile will change. For example, in Fig. A.1 and Fig. A.2, the profiles marked (a) have characteristic lengths twice those of the profiles marked (b). The location and characteristic length for the lateral profile are multiplied by the lateral ratio. This allows the use of different length scales for the primary and lateral profiles. For rotated

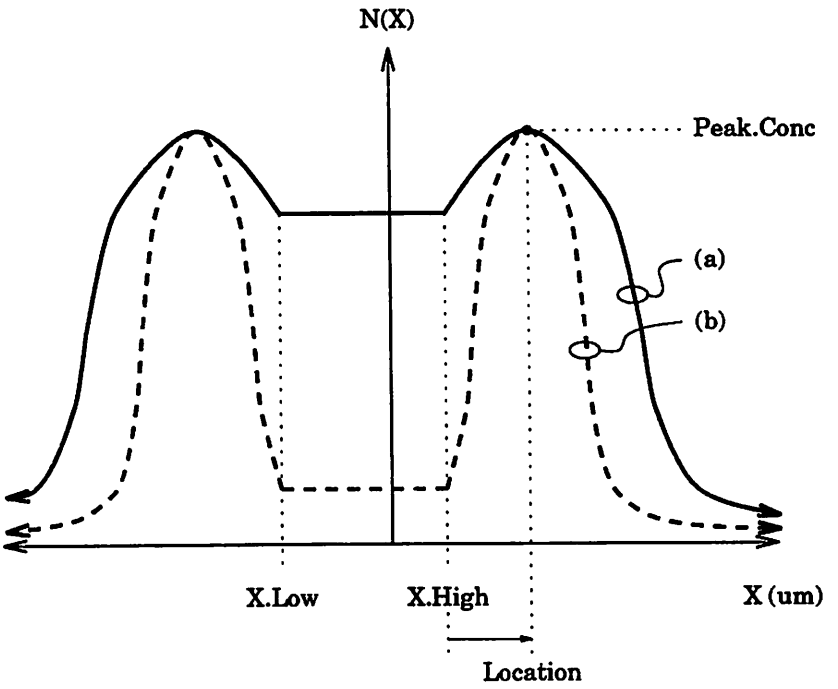


Figure A.1: 1^D doping profiles with location > 0.

profiles, this scaling is taken into account, and the profile contours are elliptical rather than circular.

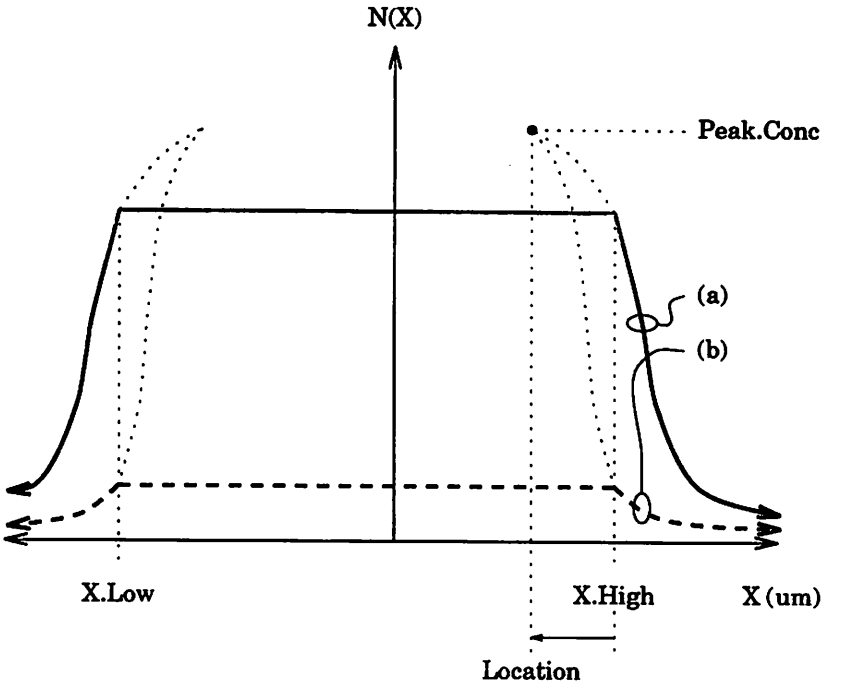


Figure A.2: 1^D doping profiles with location < 0.

PARAMETERS

Name	*Type*	*Description*
Domains	Int List	List of domains to dope
Uniform : Linear : Gaussian : Erfc : Exponential : Suprem3 : Ascii : Ascii Suprem3	Flag	Primary profile type
InFile	String	Name of Suprem3, Ascii or Ascii Suprem3 input file
Lat.Rotate : Lat.Unif : Lat.Lin : Lat.Gauss : Lat.Erfc : Lat.Exp	Flag	Lateral profile type
X.Axis : Y.Axis	Flag	Primary profile direction
N.Type : P.Type : Donor : Acceptor : Phosphorus : Arsenic : Antimony : Boron	Flag	Impurity type
X.Low	Real	Lowest X location of constant box, (μm)
X.High	Real	Highest X location of constant box, (μm)
Y.Low	Real	Lowest Y location of constant box, (μm)
Y.High	Real	Highest Y location of constant box, (μm)
Conc | Peak.Conc	Real	Dopant concentration, (cm^{-3})
Location | Range	Real	Location of profile edge/peak, (μm)
Char.Length	Real	Characteristic length of profile, (μm)
Ratio.Lat	Real	Ratio of lateral to primary distances

EXAMPLES

This first example adds a uniform background P-type doping of $1.0 \times 10^{16} cm^{-3}$ to an entire device:

```
doping uniform p.type conc=1.0e16
```

A gaussian implantation with rotated lateral falloff, such as might be used for a MOSFET source, is then added:

```
doping gauss lat.rotate n.type conc=1.0e19
+ x.l=0.0 x.h=0.5 y.l=0.0 y.h=0.2 ratio=0.7
```

Alternatively, an error-function falloff could be used:

```
doping gauss lat.erfc conc=1.0e19
+ x.l=0.0 x.h=0.5 y.l=0.0 y.h=0.2 ratio=0.7
```

Finally, the MOSFET channel implant is extracted from an ASCII-format SUPREM3 file. The lateral profile is uniform, so that the implant is confined between $X = 1\mu$m and $X = 3\mu$m. The profile begins at $Y = 0\mu$m (the high Y value defaults equal to the low Y value):

```
doping ascii suprem3 infile=implant.s3 lat.unif boron
+ x.l=1.0 x.h=3.0 y.l=0.0
```

SEE ALSO

domain, mobility, contact, boundary

NAME

ELECTRODE – Set location of a contact to the device

SYNOPSIS

electrode [number] [position]

DESCRIPTION

Each device has several electrodes which are used to connect the device to the rest of the circuit. The number of electrodes depends on the type of device. For example, a MOSFET needs 4 electrodes. A particular electrode can be identified by its position in the list of circuit nodes on the device element line. For example, the drain node of a MOSFET is electrode number 1, while the bulk node is electrode number 4. Electrodes for which an ID number has not been specified are assigned values sequentially in the order they appear in the input file.

For 1^D devices, the positions of two of the electrodes are predefined to be at the ends of the simulation mesh. The first electrode is at the low end of the mesh, and the last electrode is at the high end. The position of the special 1^D BJT base contact is set on the **options** card. Thus, electrode cards are used exclusively for 2^D devices.

Each card associates a portion of the simulation mesh with a particular electrode. In contrast to domains, which are specified only in terms of boxes, electrodes can also be specified in terms of line segments. Boxes and segments for the same electrode do not have to overlap. If they don't, it is assumed that the electrode is wired together outside the area covered by the simulation mesh. However, pieces of different electrodes must not overlap, since this would represent a short circuit.

Each electrode box or segment can be specified in terms of the locations or mesh-indices of its boundaries. A missing value defaults to the corresponding mesh boundary.

PARAMETERS

Name	*Type*	*Description*
Number	Integer	ID number of this domain
X.Low	Real	Lowest X location of electrode, (μm)
: IX.Low	Integer	Lowest X mesh-index of electrode
X.High	Real	Highest X location of electrode, (μm)
: IX.High	Integer	Highest X mesh-index of electrode
Y.Low	Real	Lowest Y location of electrode, (μm)
: IY.Low	Integer	Lowest Y mesh-index of electrode
Y.High	Real	Highest Y location of electrode, (μm)
: IY.High	Integer	Highest Y mesh-index of electrode

EXAMPLES

The following shows how the four contacts of a MOSFET might be specified:

```
* DRAIN
electrode x.l=0.0 x.h=0.5 y.l=0.0 y.h=0.0
* GATE
electrode x.l=1.0 x.h=3.0 iy.l=0 iy.h=0
* SOURCE
```

```
electrode x.l=3.0 x.h=4.0 y.l=0.0 y.h=0.0
* BULK
electrode x.l=0.0 x.h=4.0 y.l=2.0 y.h=2.0
```

The numbering option can be used when specifying bipolar transistors with dual base contacts:

```
* EMITTER
electrode num=3 x.l=1.0 x.h=2.0 y.l=0.0 y.h=0.0
* BASE
electrode num=2 x.l=0.0 x.h=0.5 y.l=0.0 y.h=0.0
electrode num=2 x.l=2.5 x.h=3.0 y.l=0.0 y.h=0.0
* COLLECTOR
electrode num=1 x.l=0.0 x.h=3.0 y.l=1.0 y.h=1.0
```

SEE ALSO

domain, contact

NAME

END – Terminate processing of a device definition

SYNOPSIS

end

DESCRIPTION

The end card stops processing of a device definition. It may appear anywhere within a definition. Subsequent continuation lines of the definition will be ignored. If no end card is supplied, all the cards will be processed.

NAME

MATERIAL – Specify physical properties of a material

SYNOPSIS

material number type [physical-constants]

DESCRIPTION

The material card is used to create an entry in the list of materials used in a device. Each entry needs a unique identification number and the type of the material. Default values are assigned to the physical properties of the material. Most material parameters are accessible either here or on the **mobility** or **contact** cards. However, some parameters remain inaccessible (e.g. the ionization coefficient parameters). Parameters for most physical effect models are collected here. Mobility parameters are handled separately by the **mobility** card. Properties of electrode materials are set using the **contact** card.

PARAMETERS

Name	*Type*	*Description*
Number	Integer	ID number of this material
Semiconductor : Silicon : Polysilicon : GaAs : Insulator : Oxide : Nitride	Flag	Type of this material
Affinity	Real	Electron affinity, (eV)
Permittivity	Real	Dielectric permittivity, (F/cm)
Nc	Real	Conduction band density, (cm^{-3})
Nv	Real	Valence band density, (cm^{-3})
Eg	Real	Energy band gap, (eV)
dEg.dT	Real	Bandgap narrowing with temperature, (eV/°K)
Eg.Tref	Real	Bandgap reference temperature, (°K)
dEg.dN	Real	Bandgap narrowing with N doping, (eV/cm^{-3})
Eg.Nref	Real	Bandgap reference concentration - N type, (cm^{-3})
dEg.dP	Real	Bandgap narrowing with P doping, (eV/cm^{-3})
Eg.Pref	Real	Bandgap reference concentration - P type, (cm^{-3})
TN	Real	SRH lifetime - electrons, (s)
SRH.Nref	Real	SRH reference concentration - electrons, (cm^{-3})
TP	Real	SRH lifetime - holes, (s)
SRH.Pref	Real	SRH reference concentration - holes, (cm^{-3})
CN	Real	Auger coefficient - electrons, (cm^6/s)
CP	Real	Auger coefficient - holes, (cm^6/s)
ARichN	Real	Richardson constant - electrons, ($A/cm^2/°K^2$)
ARichP	Real	Richardson constant - holes, ($A/cm^2/°K^2$)

EXAMPLES

Set the type of material #1 to silicon, then adjust the values of the temperature-dependent bandgap model parameters:

```
material num=1 silicon eg=1.12 deg.dt=4.7e-4 eg.tref=640.0
```

The recombination lifetimes can be set to extremely short values to simulate imperfect semiconductor material:

```
material num=2 silicon tn=1ps tp=1ps
```

SEE ALSO

domain, mobility, contact, boundary

NAME

METHOD – Choose types and parameters of numerical methods

SYNOPSIS

method [types] [parameters]

DESCRIPTION

The method card controls which numerical methods are used during a simulation and the parameters of these methods. Most of these methods are optimizations that reduce run time, but may sacrifice accuracy or reliable convergence.

For majority-carrier devices such as MOSFETs, one carrier simulations can be used to save simulation time. The systems of equations in AC analysis may be solved using either direct or successive-over-relaxation techniques. Successive-over-relaxation is faster, but at high frequencies, it may fail to converge or may converge to the wrong answer. In some cases, it is desirable to obtain AC parameters as functions of DC bias conditions. If necessary, a one-point AC analysis is performed at a predefined frequency in order to obtain these small-signal parameters. The default for this frequency is 1 Hz. The Jacobian matrix for DC and transient analyses can be simplified by ignoring the derivatives of the mobility with respect to the solution variables. However, the resulting analysis may have convergence problems. Additionally, if they are ignored during AC analyses, incorrect results may be obtained.

A damped Newton method is used as the primary solution technique for the device-level partial differential equations. This algorithm is based on an iterative loop that terminates when the error in the solution is small enough or the iteration limit is reached. Error tolerances are used when determining if the error is "small enough". The tolerances are expressed in terms of an absolute, solution-independent error and a relative, solution-dependent error. The absolute-error limit can be set on this card. The relative error is computed by multiplying the size of the solution by the circuit-level SPICE paramcter RELTOL.

PARAMETERS

Name	*Type*	*Description*
OneCarrier	Flag	Solve for majority carriers only
AC.Analysis	String	AC analysis method, (either DIRECT or SOR)
NoMobDeriv	Flag	Ignore mobility derivatives
Frequency	Real	AC analysis frequency, (Hz)
ItLim	Integer	Newton iteration limit
DevTol	Real	Maximum residual error in device equations

EXAMPLES

Use one carrier simulation for a MOSFET, and choose direct method AC analysis to ensure accurate, high frequency results:

```
method onec ac.an=direct
```

Tolerate no more than 10^{-10} absolute error in device-level equations, and perform no more than 15 Newton iteration in any one loop:

```
method devtol=1e-10 itlim=15
```

NAME

MOBILITY – Specify types and parameters of mobility models

SYNOPSIS

mobility material [carrier] [parameters] [models] [initialize]

DESCRIPTION

The mobility model is one of the most complicated models of a material's physical properties. As a result, separate cards are needed to set up this model for a given material.

Mobile carriers in a device are divided into a number of different classes, each of which has different mobility modelling. There are three levels of division. First, electrons and holes are obviously handled separately. Second, carriers in surface inversion/accumulation layers are treated differently than carriers in the bulk. Finally, bulk carriers can be either majority or minority carriers.

For surface carriers, the normal-field mobility degradation model has three user-modifiable parameters. For bulk carriers, the ionized impurity scattering model has four controllable parameters. Different sets of parameters are maintained for each of the four bulk carrier types: majority-electron, minority-electron, majority-hole and minority-hole. Velocity saturation modelling can be applied to both surface and bulk carriers. However, only two sets of parameters are maintained: one for electrons and one for holes. These must be changed on a majority carrier card (i.e. when the majority flag is set).

Several models for the physical effects are available, along with appropriate default values. Initially, a universal set of default parameters usable with all models is provided. These can be overridden by defaults specific to a particular model by setting the initialization flag. These can then be changed directly on the card itself. The bulk ionized impurity models are the Caughey-Thomas (CT) model and the Scharfetter-Gummel (SG) model [CAUG67], [SCHA69]. Three alternative sets of defaults are available for the Caughey-Thomas expression. They are the Arora (AR) parameters for Si [AROR82], the University of Florida (UF) parameters for minority carriers in Si [SOLL90], and a set of parameters appropriate for GaAs (GA). The velocity-saturation models are the Caughey-Thomas (CT) and Scharfetter-Gummel (SG) models for Si, and the PISCES model for GaAs (GA). There is also a set of Arora (AR) parameters for the Caughey-Thomas model.

PARAMETERS

Name	*Type*	*Description*
Material	Integer	ID number of material
Electron : Hole	Flag	Mobile carrier
Majority : Minority	Flag	Mobile carrier type
MuS	Real	Maximum surface mobility, ($cm^2/V \cdot s$)
EC.A	Real	Surface mobility 1st-order critical field, (V/cm)
EC.B	Real	Surface mobility 2nd-order critical field, (V^2/cm^2)
MuMax	Real	Maximum bulk mobility, ($cm^2/V \cdot s$)
MuMin	Real	Minimum bulk mobility, ($cm^2/V \cdot s$)
NtRef	Real	Ionized impurity reference concentration, (cm^{-3})
NtExp	Real	Ionized impurity exponent
Vsat	Real	Saturation velocity, (cm/s)
Vwarm	Real	Warm carrier reference velocity, (cm/s)
ConcModel	String	Ionized impurity model, (CT, AR, UF, SG, or GA)
FieldModel	String	Velocity saturation model, (CT, AR, SG, or GA)
Init	Flag	Copy model-specific defaults

EXAMPLES

The following set of cards completely updates the bulk mobility parameters for material #1:

```
mobility mat=1 concmod=sg fieldmod=sg
mobility mat=1 elec major mumax=1000.0 mumin=100.0
+ ntref=1.0e16 ntexp=0.8 vsat=1.0e7 vwarm=3.0e6
mobility mat=1 elec minor mumax=1000.0 mumin=200.0
+ ntref=1.0e17 ntexp=0.9
mobility mat=1 hole major mumax=500.0 mumin=50.0
+ ntref=1.0e16 ntexp=0.7 vsat=8.0e6 vwarm=1.0e6
mobility mat=1 hole minor mumax=500.0 mumin=150.0
+ ntref=1.0e17 ntexp=0.8
```

The electron surface mobility is changed by the following:

```
mobility mat=1 elec mus=800.0 ec.a=3.0e5 ec.b=9.0e5
```

Finally, the default Scharfetter-Gummel parameters can be used in Si with the GaAs velocity-saturation model (even though it doesn't make physical sense!):

```
mobility mat=1 init elec major fieldmodel=sg
mobility mat=1 init hole major fieldmodel=sg
mobility mat=1 fieldmodel=ga
```

SEE ALSO

material

BUGS

The surface mobility model does not include temperature-dependence for the transverse-field parameters. Those parameters will need to be adjusted by hand.

NAME

MODELS – Specify which physical models should be simulated

SYNOPSIS

models [model flags]

DESCRIPTION

The models card indicates which physical effects should be modeled during a simulation. Initially, none of the effects are included. A flag can be set false by preceding it by a caret.

PARAMETERS

Name	*Type*	*Description*
BGN	Flag	Bandgap narrowing
SRH	Flag	Shockley-Reed-Hall recombination
ConcTau	Flag	Concentration-dependent SRH lifetimes
Auger	Flag	Auger recombination
Avalanche	Flag	Local avalanche generation
TempMob	Flag	Temperature-dependent mobility
ConcMob	Flag	Concentration-dependent mobility
FieldMob	Flag	Lateral-field-dependent mobility
TransMob	Flag	Transverse-field-dependent surface mobility
SurfMob	Flag	Activate surface mobility model

EXAMPLES

Turn on bandgap narrowing, and all of the generation-recombination effects:

```
models bgn srh conctau auger aval
```

Amend the first card by turning on lateral- and transverse-field-dependent mobility in surface charge layers, and lateral-field-dependent mobility in the bulk. Also, this line turns avalanche generation modeling off.

```
models surfmob transmob fieldmob ^aval
```

SEE ALSO

material, mobility

BUGS

The local avalanche generation model for 2^D devices does not compute the necessary contributions to the device-level Jacobian matrix. If this model is used, it may cause convergence difficulties and it will cause AC analyses to produce incorrect results.

NAME

OPTIONS – Provide optional device-specific information

SYNOPSIS

options [device-type] [initial-state] [dimensions] [measurement-temperature]

DESCRIPTION

The options card functions as a catch-all for various information related to the circuit-device interface. The type of a device can be specified here, but will be defaulted if none is given. Device type is used primarily to determine how to limit the changes in voltage between the terminals of a device. It also helps determine what kind of boundary conditions are used as defaults for the device electrodes.

A previously calculated state, stored in the named initial-conditions file, can be loaded at the beginning of an analysis. If it is necessary for each instance of a numerical model to start in a different state, then the unique flag can be used to generate unique filenames for each instance by appending the instance name to the given filename. This is the same method used by CIDER to generate unique filenames when the states are originally saved. If a particular state file does not fit this pattern, the filename can be entered directly on the instance line.

Mask dimension defaults can be set so that device sizes can be specified in terms of area or width. Dimensions for the special 1^D BJT base contact can also be controlled.

The measurement temperature of material parameters, normally taken to be the circuit default, can be overridden.

PARAMETERS

Name	*Type*	*Description*
Resistor	Flag	Resistor
: Capacitor	Flag	Capacitor
: Diode	Flag	Diode
: Bipolar | BJT	Flag	Bipolar transistor
: MOSFET	Flag	MOS field-effect transistor
: JFET	Flag	Junction field-effect transistor
: MESFET	Flag	MES field-effect transistor
IC.File	String	Initial-conditions filename
Unique	Flag	Append instance name to filename
DefA	Real	Default Mask Area, (m^2)
DefW	Real	Default Mask Width, (m)
DefL	Real	Default Mask Length, (m)
Base.Area	Real	1^D BJT base area relative to emitter area
Base.Length	Real	1^D BJT base contact length, (μm)
Base.Depth	Real	1^D BJT base contact depth, (μm)
TNom	Real	Nominal measurement temperature, (°C)

EXAMPLES

Normally, a 'numos' device model is used for MOSFET devices. However, it can be changed into a bipolar-with-substrate-contact model, by specifying a bipolar structure using the other cards, and indicating the device-structure type as shown here. The default length is

set to 1.0 μm so that when mask area is specified on the element line it can be divided by this default to obtain the device width.

```
options bipolar defl=1.0
```

Specify that a 1^D BJT has base area 1/10th that of the emitter, has an effective base contact depth of 0.2 μm and a length between the internal and external base contacts of 1.5 μm:

```
options base.area=0.1 base.depth=0.2 base.len=1.5
```

If a circuit contains two instances of a bipolar transistor model named 'q1' and 'q2', then the following line tells the simulator to look for initial conditions in the files 'OP1.q1' and 'OP1.q2', respectively. The period in the middle of the names is added automatically:

```
options unique ic.file="OP1"
```

SEE ALSO

numd, nbjt, numos

NAME

OUTPUT – Identify information to be printed or saved

SYNOPSIS

output [debugging-flags] [general-info] [saved-solutions]

DESCRIPTION

The output card is used to control the amount of information that is either presented to or saved for the user. Three types of information are available. Debugging information is available as a means to monitor program execution. This is useful during long simulations when one is unsure about whether the program has become trapped at some stage of the simulation. General information about a device such as material parameters and resource usage can be obtained. Finally, information about the internal and external states of a device is available. Since this data is best interpreted using a post-processor, a facility is avaliable for saving device solutions in auxiliary output files. Solution filenames are automatically generated by the simulator. If the named file already exists, the file will be overwritten. A filename unique to a particular circuit or run can be generated by providing a root filename. This root name will be added onto the beginning of the automatically generated name. This feature can be used to store solutions in a directory other than the current one by specifying the root filename as the path of the desired directory. Solutions are only saved for those devices that specify the 'save' parameter on their instance lines.

The various physical values that can be saved are named below. By default, the following values are saved: the doping, the electron and hole concentrations, the potential, the electric field, the electron and hole current densities, and the displacement current density. Values can be added to or deleted from this list by turning the appropriate flag on or off. For vector-valued quantities in two dimensions, both the X and Y components are saved. The vector magnitude can be obtained during post-processing.

Saved solutions can be used in conjunction with the **options** card and instance lines to reuse previously calculated solutions as initial guesses for new solutions. For example, it is typical to initialize the device to a known state prior to beginning any DC transfer curve or operating point analysis. This state is an ideal candidate to be saved for later use when it is known that many analyses will be performed on a particular device structure.

PARAMETERS

Name	*Type*	*Description*
All.Debug	Flag	Debug all analyses
OP.Debug	Flag	Debug .OP analyses
DC.Debug	Flag	Debug .DC analyses
TRAN.Debug	Flag	Debug .TRAN analyses
AC.Debug	Flag	Debug .AC analyses
PZ.Debug	Flag	Debug .PZ analyses
Material	Flag	Physical material information
Statistics | Resources	Flag	Resource usage information
RootFile	String	Root of output file names
Psi	Flag	Potential (V)
Equ.Psi	Flag	Equilibrium potential (V)
Vac.Psi	Flag	Vacuum potential (V)
Doping	Flag	Net doping (cm^{-3})
N.Conc	Flag	Electron concentration (cm^{-3})
P.Conc	Flag	Hole concentration (cm^{-3})
PhiN	Flag	Electron quasi-fermi potential (V)
PhiP	Flag	Hole quasi-fermi potential (V)
PhiC	Flag	Conduction band potential (V)
PhiV	Flag	Valence band potential (V)
E.Field	Flag	Electric field (V/cm)
JC	Flag	Conduction current density (A/cm^2)
JD	Flag	Displacement current density (A/cm^2)
JN	Flag	Electron current density (A/cm^2)
JP	Flag	Hole current density (A/cm^2)
JT	Flag	Total current density (A/cm^2)
Unet	Flag	Net recombination (cm^{-3}/s)
MuN	Flag	Electron mobility (low-field) ($cm^2/V{\cdot}s$)
MuP	Flag	Hole mobility (low-field) ($cm^2/V{\cdot}s$)

EXAMPLES

The following example activates all potentially valuable diagnostic output:

```
output all.debug mater stat
```

Energy band diagrams generally contain the potential, the quasi-fermi levels, the band edge energies and the vacuum energy. The following example enables saving of the non-default values needed to make energy band diagrams:

```
output phin phip phic phiv vac.psi
```

Sometimes it is desirable to save certain key solutions, and then reload them for use in subsequent simulations. In such cases only the essential values (Ψ, n, and p) need to be saved. This example turns off the nonessential default values (and indicates the essential ones explicitly):

```
output psi n.conc p.conc ^e.f ^jn ^jp ^jd
```

SEE ALSO

options, numd, nbjt, numos

NAME

TITLE – Provide a label for this device's output

SYNOPSIS

title [text]

DESCRIPTION

The title card provides a label for use as a heading in various output files. The text can be any length, but titles that fit on a single line will produce more aesthetically pleasing output.

EXAMPLES

Set the title for a minimum gate length NMOSFET in a 1.0 μm BiCMOS process:

```
title L=1.0um NMOS Device, 1.0um BiCMOS Process
```

BUGS

The title is currently treated like a comment.

NAME

X.MESH, Y.MESH – Define locations of lines/nodes in a mesh

SYNOPSIS

x.mesh position numbering-method [spacing-parameters]
y.mesh position numbering-method [spacing-parameters]

DESCRIPTION

The domains of a device are discretized onto a rectangular finite-difference mesh using x.mesh cards for 1^D devices, or x.mesh and y.mesh cards for 2^D devices. Both uniform and non-uniform meshes can be specified.

A typical mesh for a 2^D device is shown in Figure A.3. The mesh is divided into intervals

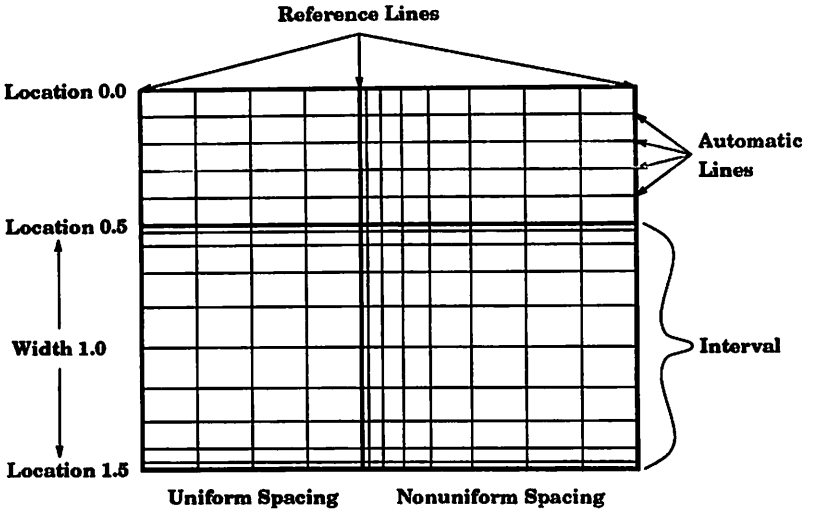


Figure A.3: Typical mesh for 2^D device.

by the *reference* lines. The other lines in each interval are automatically generated by CIDER using the mesh spacing parameters. In general, each new mesh card adds one reference line and multiple automatic lines to the mesh. Conceptually, a 1^D mesh is similar to a 2^D mesh except that there are no reference or automatic lines needed in the second dimension.

The location of a reference line in the mesh must either be given explicitly (using *Location*) or defined implicitly relative to the location of the previous reference line (by using *Width*). (If the first card in either direction is specified using *Width*, an initial reference line is automatically generated at location 0.0.) The line number of the reference line can be given explicitly, in which case the automatic lines are evenly spaced within the interval, and the number of lines is determined from the difference between the current line number and that of the previous reference line. However, if the interval width is given, then the line number is interpreted directly as the number of additional lines to add to the mesh.

For a nonuniformly spaced interval, the number of automatic lines has to be determined using the mesh spacing parameters. Nonuniform spacing is triggered by providing a desired

ratio for the lengths of the spaces between adjacent pairs of lines. This ratio should always be greater than one, indicating the ratio of larger spaces to smaller spaces. In addition to the ratio, one or both of the space widths at the ends of the interval must be provided. If only one is given, it will be the smallest space and the largest space will be at the opposite end of the interval. If both are given, the largest space will be in the middle of the interval. In certain cases it is desirable to limit the growth of space widths in order to control the solution accuracy. This can be accomplished by specifying a maximum space size, but this option is only available when one of the two end lengths is given. Note that once the number of new lines is determined using the desired ratio, the actual spacing ratio may be adjusted so that the spaces exactly fill the interval.

PARAMETERS

Name	*Type*	*Description*
Location	Real	Location of this mesh line, (μm)
: Width	Real	Width between this and previous mesh lines, (μm)
Number | Node	Integer	Number of this mesh line
: Ratio	Real	Ratio of sizes of adjacent spaces
H.Start | H1	Real	Space size at start of interval, (μm)
H.End | H2	Real	Space size at end of interval, (μm)
H.Max | H3	Real	Maximum space size inside interval, (μm)

EXAMPLES

A 50 node, uniform mesh for a 5 μm long semiconductor resistor can be specified as:

```
x.mesh loc=0.0 n=1
x.mesh loc=5.0 n=50
```

An accurate mesh for a 1^D diode needs fine spacing near the junction. In this example, the junction is assumed to be 0.75 μm deep. The spacing near the diode ends is limited to a maximum of 0.1 μm:

```
x.mesh w=0.75 h.e=0.001 h.m=0.1 ratio=1.5
x.mesh w=2.25 h.s=0.001 h.m=0.1 ratio=1.5
```

The vertical mesh spacing of a MOSFET can generally be specified as uniform through the gate oxide, very fine for the surface inversion layer, moderate down to the source/drain junction depth, and then increasing all the way to the bulk contact:

```
y.mesh loc=-.04 node=1
y.mesh loc=0.0 node=6
y.mesh width=0.5 h.start=0.001 h.max=.05 ratio=2.0
y.mesh width=2.5 h.start=0.05 ratio=2.0
```

SEE ALSO

domain

NAME

NUMD – Diode / two-terminal numerical models and elements

SYNOPSIS

Model:
.MODEL model-name NUMD [level]
+ ...

Element:
DXXXXXXX n1 n2 model-name [geometry] [temperature] [initial-conditions]

Output:
.SAVE [small-signal values]

DESCRIPTION

NUMD is the name for a diode numerical model. In addition, this same model can be used to simulate other two-terminal structures such as semiconductor resistors and MOS capacitors. See the **options** card for more information on how to customize the device type.

Both 1^D and 2^D devices are supported. These correspond to the LEVEL=1 and LEVEL=2 models, respectively. If left unspecified, it is assumed that the device is one-dimensional.

All numerical two-terminal element names begin with the letter 'D'. The element name is then followed by the names of the positive (n1) and negative (n2) nodes. After this must come the name of the model used for the element. The remaining information can come in any order. The layout dimensions of an element are specified relative to the geometry of a default device. For 1^D devices, the default device has an area of $1m^2$, and for 2^D devices, the default device has a width of 1m. However, these defaults can be overridden on an **options** card. The operating temperature of a device can be set independently from that of the rest of the circuit in order to simulate non-isothermal circuit operation. Finally, the name of a file containing an initial state for the device can be specified. Remember that if the filename contains capital letters, they must be protected by surrounding the filename with double quotes. Alternatively, the device can be placed in an OFF state (thermal equilibrium) at the beginning of the analysis. For more information on the use of initial conditions, see the SPICE User's Manual.

In addition to the element input parameters, there are output-only parameters that can be shown using the SPICE `show` command or captured using the `save`/`.SAVE` command. These parameters are the elements of the indefinite conductance (G), capacitance (C), and admittance (Y) matrices where $Y = G + j\omega C$. By default, the parameters are computed at 1 Hz. Each element is accessed using the name of the matrix (g, c or y) followed by the node indices of the output terminal and the input terminal (e.g. g11). Beware that parameter names are case-sensitive for `save`/`show`, so lower-case letters must be used.

PARAMETERS

Name	*Type*	*Description*
Level	Integer	Dimensionality of numerical model
Area	Real	Multiplicative area factor
W	Real	Multiplicative width factor
Temp	Real	Element operating temperature
IC.File	String	Initial-conditions filename
Off	Flag	Device initially in OFF state
gIJ	Flag	Conductance element G_{ij}, (Ω)
cIJ	Flag	Capacitance element C_{ij}, (F)
yIJ	Flag	Admittance element Y_{ij}, (Ω)

EXAMPLES

A one-dimensional numerical switching-diode element/model pair with an area twice that of the default device (which has a size of 1μm $\times$1 μm) can be specified using:

```
DSWITCH 1 2 M_SWITCH_DIODE AREA=2
.MODEL M_SWITCH_DIODE NUMD
+ options defa=1p ...
+ ...
```

A two-dimensional two-terminal MOS capacitor with a width of 20μm and an initial condition of 3V is created by:

```
DMOSCAP 11 12 M_MOSCAP W=20um IC=3v
.MODEL M_MOSCAP NUMD LEVEL=2
+ options moscap defw=1m
+ ...
```

The next example shows how both the width and area factors can be used to create a power diode with area twice that of a 6μm-wide device (i.e. a 12μm-wide device). The device is assumed to be operating at a temperature of 100°C:

```
D1 POSN NEGN POWERMOD AREA=2 W=6um TEMP=100.0
.MODEL POWERMOD NUMD LEVEL=2
+ ...
```

This example saves all the small-signal parameters of the previous diode:

```
.SAVE @d1[g11] @d1[g12] @d1[g21] @d1[g22]
.SAVE @d1[c11] @d1[c12] @d1[c21] @d1[c22]
.SAVE @d1[y11] @d1[y12] @d1[y21] @d1[y22]
```

SEE ALSO

options, output

BUGS

Convergence problems may be experienced when simulating MOS capacitors due to singularities in the current-continuity equations.

NAME

NBJT – Bipolar / three-terminal numerical models and elements

SYNOPSIS

Model:
.MODEL model-name NBJT [level]
+ ...

Element:
QXXXXXXX n1 n2 n3 model-name [geometry] [temperature] [initial-conditions]

Output:
.SAVE [small-signal values]

DESCRIPTION

NBJT is the name for a bipolar transistor numerical model. In addition, the 2^D model can be used to simulate other three-terminal structures such as a JFET or MESFET. However, the 1^D model is customized with a special base contact, and cannot be used for other purposes. See the **options** card for more information on how to customize the device type and setup the 1^D base contact.

Both 1^D and 2^D devices are supported. These correspond to the LEVEL=1 and LEVEL=2 models, respectively. If left unspecified, it is assumed that the device is one-dimensional.

All numerical three-terminal element names begin with the letter 'Q'. If the device is a bipolar transistor, then the nodes are specified in the order: collector (n1), base (n2), emitter (n3). For a JFET or MESFET, the node order is: drain (n1), gate (n2), source (n3). After this must come the name of the model used for the element. The remaining information can come in any order. The layout dimensions of an element are specified relative to the geometry of a default device. For the 1^D BJT, the default device has an area of $1m^2$, and for 2^D devices, the default device has a width of 1m. In addition, it is assumed that the default 1^DBJT has a base contact with area equal to the emitter area, length of 1 μm and a depth automatically determined from the device doping profile. However, all these defaults can be overridden on an **options** card.

The operating temperature of a device can be set independently from that of the rest of the circuit in order to simulate non-isothermal circuit operation. Finally, the name of a file containing an initial state for the device can be specified. Remember that if the filename contains capital letters, they must be protected by surrounding the filename with double quotes. Alternatively, the device can be placed in an OFF state (thermal equilibrium) at the beginning of the analysis. For more information on the use of initial conditions, see the SPICE User's Manual.

In addition to the element input parameters, there are output-only parameters that can be shown using the SPICE `show` command or captured using the `save/.SAVE` command. These parameters are the elements of the indefinite conductance (G), capacitance (C), and admittance (Y) matrices where $Y = G + j\omega C$. By default, the parameters are computed at 1 Hz. Each element is accessed using the name of the matrix (g, c or y) followed by the node indices of the output terminal and the input terminal (e.g. g11). Beware that parameter names are case-sensitive for `save/show`, so lower-case letters must be used.

PARAMETERS

Name	*Type*	*Description*
Level	Integer	Dimensionality of numerical model
Area	Real	Multiplicative area factor
W	Real	Multiplicative width factor
Temp	Real	Element operating temperature
IC.File	String	Initial-conditions filename
Off	Flag	Device initially in OFF state
gIJ	Flag	Conductance element G_{ij}, (Ω)
cIJ	Flag	Capacitance element C_{ij}, (F)
yIJ	Flag	Admittance element Y_{ij}, (Ω)

EXAMPLES

A one-dimensional numerical bipolar transistor with an emitter stripe 4 times as wide as the default device is created using:

```
Q2 1 2 3 M_BJT AREA=4
```

This example saves the output conductance (go), transconductance (gm) and input conductance (gpi) of the previous transistor in that order:

```
.SAVE @q2[g11] @q2[g12] @q2[g22]
```

The second example is for a two-dimensional JFET with a width of 5μm and initial conditions obtained from file "IC.jfet":

```
QJ1 11 12 13 M_JFET W=5um IC.FILE="IC.jfet"
.MODEL M_JFET NBJT LEVEL=2
+ options jfet
+ ...
```

A final example shows how to use symmetry to simulate half of a 2^D BJT, avoiding having the user double the area of each instance:

```
Q2 NC2 NB2 NE2 BJTMOD AREA=1
Q3 NC3 NB3 NE3 BJTMOD AREA=1
.MODEL BJTMOD NBJT LEVEL=2
+ options defw=2um
+ * Define half of the device now
+ ...
```

SEE ALSO

options, output

BUGS

MESFETs cannot be simulated properly yet because Schottky contacts have not been implemented.

NAME

NUMOS – MOSFET / four-terminal numerical models and elements

SYNOPSIS

Model:
.MODEL model-name NUMOS [level]
+ ...

Element:
MXXXXXXX n1 n2 n3 n4 model-name [geometry] [temperature] [initial-conditions]

Output:
.SAVE [small-signal values]

DESCRIPTION

NUMOS is the name for a MOSFET numerical model. In addition, the 2^D model can be used to simulate other four-terminal structures such as integrated bipolar and JFET devices with substrate contacts. However, silicon controlled rectifiers (SCRs) cannot be simulated because of the snapback in the transfer characteristic. See the **options** card for more information on how to customize the device type. The LEVEL parameter of two- and three- terminal devices is not needed, because only 2^D devices are supported. However, it will accepted and ignored if provided.

All numerical four-terminal element names begin with the letter 'M'. If the device is a MOSFET, or JFET with a bulk contact, then the nodes are specified in the order: drain (n1), gate (n2), source (n3), bulk (n4). If the device is a BJT, the node order is: collector (n1), base (n2), emitter (n3), substrate (n4). After this must come the name of the model used for the element. The remaining information can come in any order. The layout dimensions of an element are specified relative to the geometry of a default device. The default device has a width of 1m. However, this default can be overridden on an **options** card. In addition, the element line will accept a length parameter, L, but does not use it in any calculations. This is provided to enable somewhat greater compatibility between numerical MOSFET models and the standard SPICE3 compact MOSFET models.

The operating temperature of a device can be set independently from that of the rest of the circuit in order to simulate non-isothermal circuit operation. Finally, the name of a file containing an initial state for the device can be specified. Remember that if the filename contains capital letters, they must be protected by surrounding the filename with double quotes. Alternatively, the device can be placed in an OFF state (thermal equilibrium) at the beginning of the analysis. For more information on the use of initial conditions, see the SPICE

In addition to the element input parameters, there are output-only parameters that can be shown using the SPICE `show` command or captured using the `save`/`.SAVE` command. These parameters are the elements of the indefinite conductance (G), capacitance (C), and admittance (Y) matrices where $Y = G + j\omega C$. By default, the parameters are computed at 1 Hz. Each element is accessed using the name of the matrix (g, c or y) followed by the node indices of the output terminal and the input terminal (e.g. g11). Beware that parameter names are case-sensitive for `save`/`show`, so lower-case letters must be used.

PARAMETERS

Name	*Type*	*Description*
Level	Integer	Dimensionality of numerical model
Area	Real	Multiplicative area factor
W	Real	Multiplicative width factor
L	Real	Unused length factor
Temp	Real	Element operating temperature
IC.File	String	Initial-conditions filename
Off	Flag	Device initially in OFF state
gIJ	Flag	Conductance element G_{ij}, (Ω)
cIJ	Flag	Capacitance element C_{ij}, (F)
yIJ	Flag	Admittance element Y_{ij}, (Ω)

EXAMPLES

A numerical MOSFET with a gate width of 5μm and length of 1μm is described below. However, the model can only be used for 1μm length devices, so the length parameter is redundant. The device is initially biased near its threshhold by taking an initial state from the file "NM1.vth".

```
M1 1 2 3 4 M_NMOS_1UM W=5um L=1um IC.FILE="NM1.vth"
.MODEL M_NMOS_1UM NUMOS
+ * Description of a 1um device
+ ...
```

This example saves the definite admittance matrix of the previous MOSFET where the source terminal (3) is used as the reference. (The definite admittance matrix is formed by deleting the third row and column from the indefinite admittance matrix.)

```
.SAVE @m1[y11] @m1[y12] @m1[y14]
.SAVE @m1[y21] @m1[y22] @m1[y24]
.SAVE @m1[y41] @m1[y42] @m1[y44]
```

Bipolar transistors are usually specified in terms of their area relative to a unit device. The following example creates a unit-sized device:

```
MQ1 NC NB NE NS M_BJT
.MODEL M_BJT NUMOS LEVEL=2
+ options bipolar defw=5um
+ ...
```

SEE ALSO

options, output

NAME

EXAMPLE 1 – One-Dimensional Diode Capacitance

DESCRIPTION

This example demonstrates the use of CIDER as a means to obtain compact model parameters. The junction capacitance of a diode is obtained by recording AC small-signal parameters during a DC transfer curve analysis. The diode voltage is swept from a reverse bias of 3.0V to a forward bias of 0.3V in 50mV steps. The diode capacitance `@d1[c11]` is saved at each bias point. The results are then compared to a fit to the standard diode junction capacitance model:

$$CD = \frac{CJ0}{\left(1 - \frac{V_D}{VJ}\right)^{MJ}}$$

INPUT FILE

```
One-Dimensional Diode Capacitance

Vpp 1 0 0.7v (PWL 0ns 3.0v 0.01ns -6.0v) (AC 1v)
Vnn 2 0 0v
D1 1 2 M_PN AREA=100

.model M_PN numd level=1
+ options defa=1p
+ x.mesh loc=0.0 n=1
+ x.mesh loc=1.3 n=201
+ domain num=1 material=1
+ material num=1 silicon
+ doping gauss p.type conc=1e20 x.l=0.0 x.h=0.0 char.l=0.100
+ doping unif n.type conc=1e16 x.l=0.0 x.h=1.3
+ doping gauss n.type conc=5e19 x.l=1.3 x.h=1.3 char.l=0.100
+ models bgn aval srh auger conctau concmob fieldmob
+ method ac=direct

.OPTION ACCT
.DC Vpp -3.0v 0.3001v 50mv
.PRINT DC I(Vpp)
.SAVE ALL @d1[c11]
.END
```

RESULTS

The doping profile for the simulated diode is shown in Figure A.4. The diode capacitance is shown in Figure A.5 as obtained from CIDER and from the standard SPICE model where `CJ0=32.4fF, VJ=0.68, MJ=0.47`. As can be seen the fit is excellent.

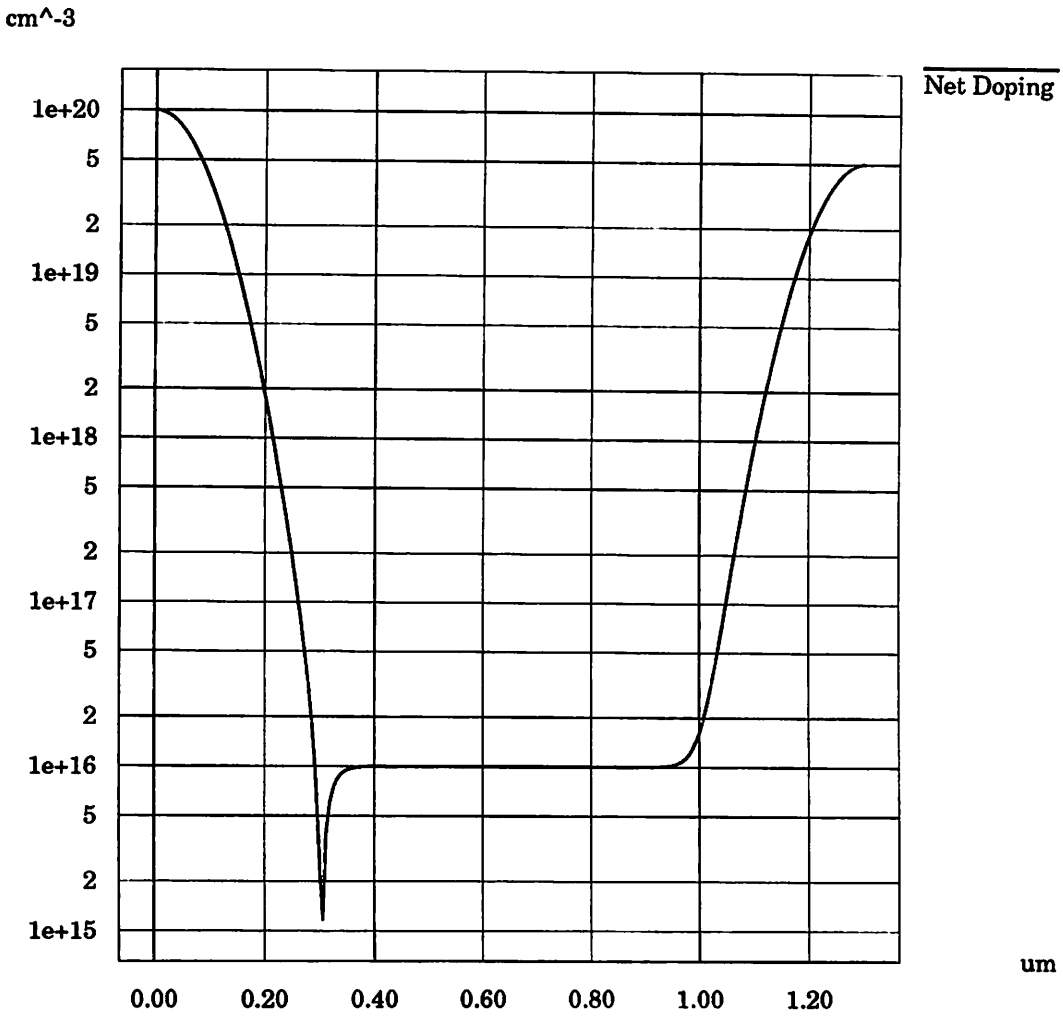


Figure A.4: 1^D Diode Doping Profile

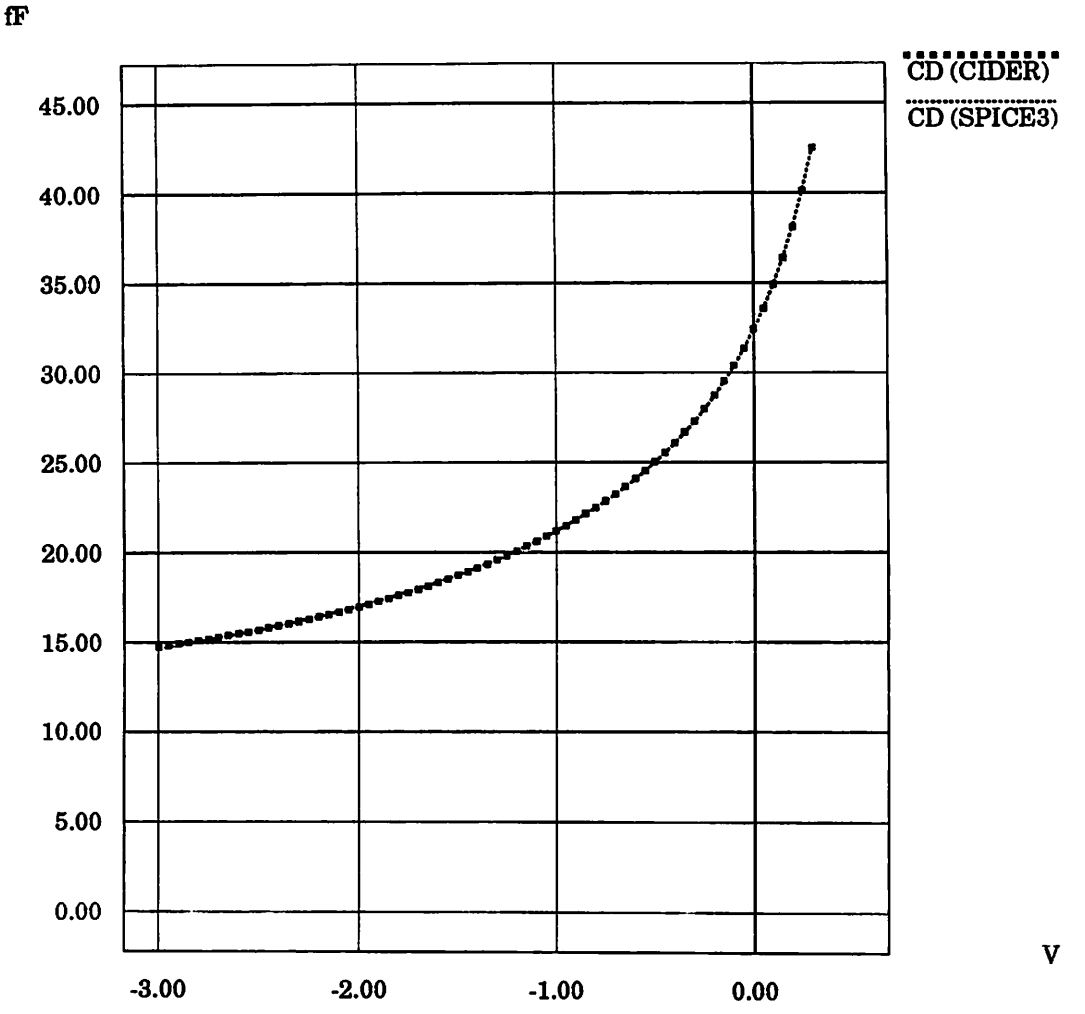


Figure A.5: Diode Capacitance from CIDER and SPICE3

NAME

EXAMPLE 2 – One-Dimensional Bipolar Frequency Response

DESCRIPTION

This example demonstrates the use of AC small-signal analysis in CIDER. The circuit is an NPN emitter-coupled pair with a PNP active load. The gain of this circuit is primarily determined by the transconductance of the Q1-Q2 pair and the output resistances of Q2 and Q4, which are difficult to model accurately using the existing SPICE compact bipolar model. Matching considerations dictate the need for numerical models for all four devices. The doping profiles are representative of a 1.0um complementary poly-emitter bipolar process. A DC offset voltage of -0.5mv is needed to center the operating point at the point of maximum gain.

INPUT FILE

```
Emitter Coupled Pair with Active Load

VCC 1 0 5v
VEE 2 0 0v
VINP 4 0 2.9995v AC 0.5v
VINM 7 0 3v AC 0.5v 180
IEE 5 2 0.1mA
Q1 3 4 5 M_NPN AREA=8
Q2 6 7 5 M_NPN AREA=8
Q3 3 3 1 M_PNP AREA=8
Q4 6 3 1 M_PNP AREA=8

.AC DEC 10 10kHz 100gHz
.PLOT AC VDB(6)

.model M_NPN nbjt level=1
+ options base.depth=0.15 base.area=0.1 base.length=1.0 defa=1p
+ x.mesh loc=-0.2 n=1
+ x.mesh loc=0.0 n=51
+ x.mesh wid=0.15 h.e=0.0001 h.m=.004 r=1.2
+ x.mesh wid=1.15 h.s=0.0001 h.m=.004 r=1.2
+ domain num=1 material=1 x.l=0.0
+ domain num=2 material=2 x.h=0.0
+ material num=1 silicon
+ material num=2 polysilicon
+ doping gauss n.type conc=3e20 x.l=-0.2 x.h=0.0 char.len=0.047
+ doping gauss p.type conc=5e18 x.l=-0.2 x.h=0.0 char.len=0.100
+ doping unif n.type conc=1e16 x.l=0.0 x.h=1.3
+ doping gauss n.type conc=5e19 x.l=1.3 x.h=1.3 char.len=0.100
+ models bgn srh auger conctau concmob fieldmob
+ method devtol=1e-12 ac=direct itlim=15

.model M_PNP nbjt level=1
+ options base.depth=0.2 base.area=0.1 base.length=1.0 defa=1p
```

```
+ x.mesh loc=-0.2 n=1
+ x.mesh loc=0.0 n=51
+ x.mesh wid=0.20 h.e=0.0001 h.m=.004 r=1.2
+ x.mesh wid=1.10 h.s=0.0001 h.m=.004 r=1.2
+ domain num=1 material=1 x.l=0.0
+ domain num=2 material=2 x.h=0.0
+ material num=1 silicon
+ material num=2 polysilicon
+ doping gauss p.type conc=3e20 x.l=-0.2 x.h=0.0 char.len=0.047
+ doping gauss n.type conc=5e17 x.l=-0.2 x.h=0.0 char.len=0.200
+ doping unif p.type conc=1e16 x.l=0.0 x.h=1.3
+ doping gauss p.type conc=5e19 x.l=1.3 x.h=1.3 char.len=0.100
+ models bgn srh auger conctau concmob fieldmob
+ method devtol=1e-12 ac=direct itlim=15

.OPTIONS ACCT RELTOL=1E-6
.END
```

RESULTS

The doping profiles for the NPN and PNP devices are shown in Figure A.6 and Figure A.7, respectively. In order to center the operating point at the point of maximum gain, a DC offset voltage of -0.5mv is needed on the positive input when the operating temperature is 27°C(the default). In Figure A.8, the small-signal gains of the emitter coupled-pair is plotted as a function of frequency. Both the differential-mode gain (calculated by this input file) and the common-mode gain (calculated separately) are presented. In addition, the differential-mode gain has been calculated at -50°C, 27°Cand 150°C. At 27°C, the low-frequency differential-mode gain is 51.7 dB, the unity gain bandwidth is 15.2 GHz but the phase-margin is only 13°. The low-frequency common-mode gain (calculated separately) is -23.0 dB, so the common-mode rejection ratio (CMRR) is 74.7 dB. Notice that differential-mode gain decreases as the temperature increases. This is caused by an increase in the thermal voltage that degrades the transconductance of the input transistors. The output resistances of Q2 and Q4 remain relatively constant with temperature.

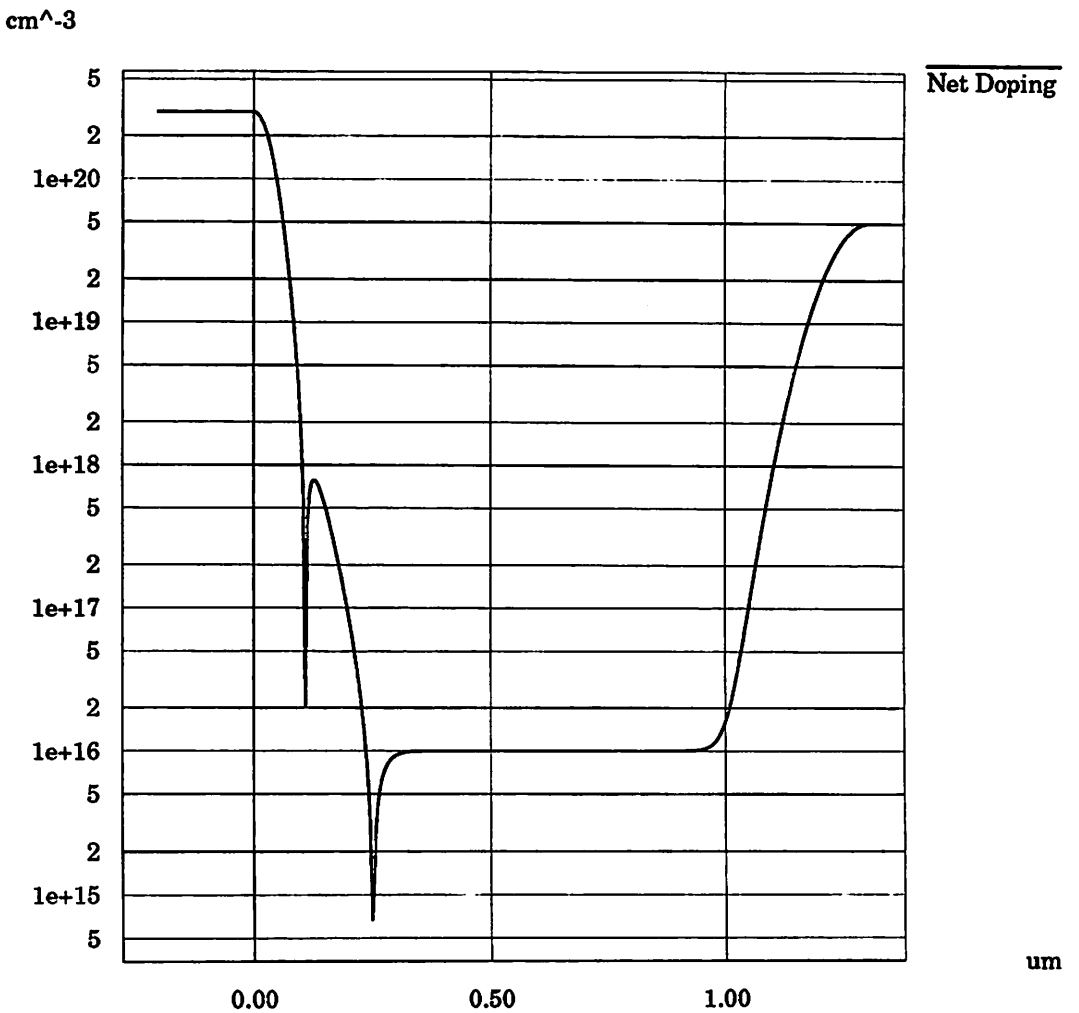


Figure A.6: 1^D NPN Doping Profile

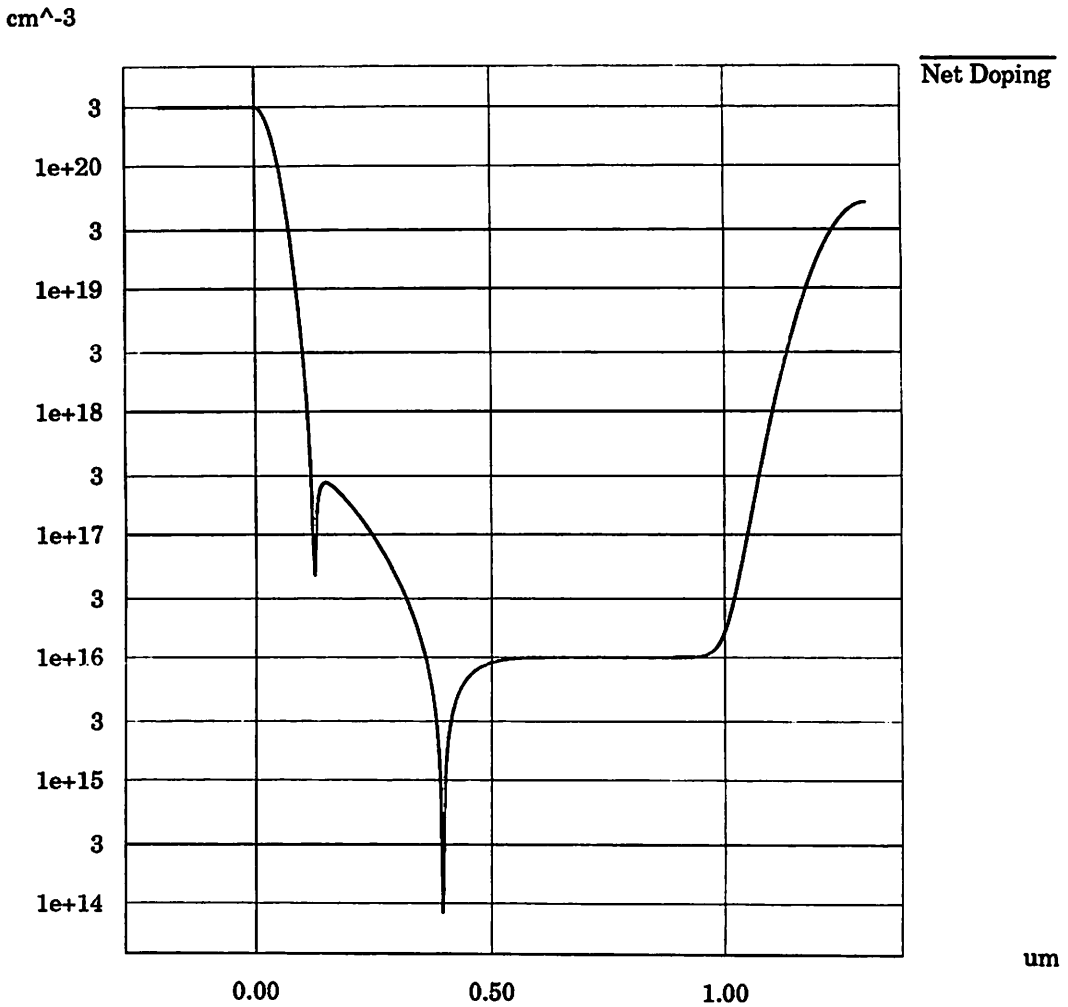


Figure A.7: 1^D PNP Doping Profile

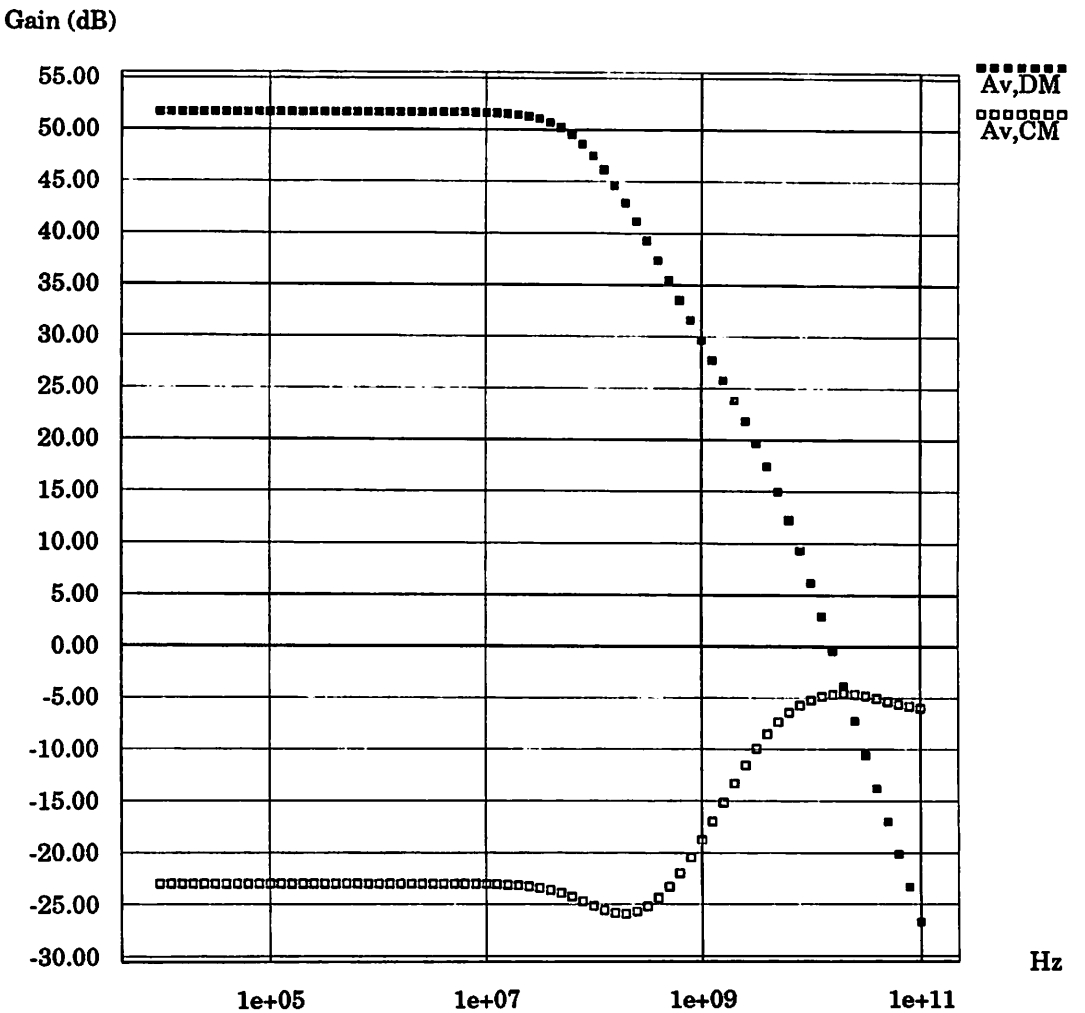


Figure A.8: Small-Signal Gains of Emitter-Coupled Pair

NAME

EXAMPLE 3 – Two-Dimensional MOSFET Transient Response

DESCRIPTION

The third example is an NMOS bootstrapped enhancement-load inverter. The transient response of the circuit is calculated as the input voltage is pulsed from high to low and back again. A schematic of the circuit is shown in Figure A.9. Two-dimensional numerical

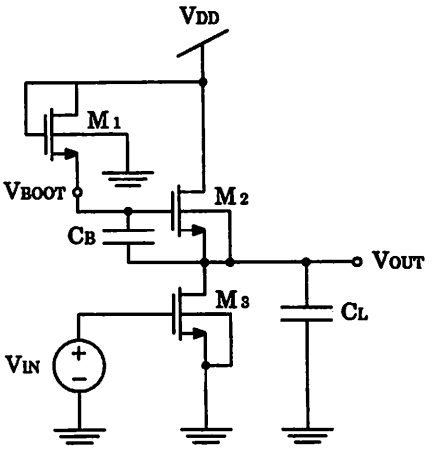


Figure A.9: Bootstrap Inverter Schematic

models are used for the three MOS transistors. The only physical models enabled are the concentration- and field-dependent mobility models. In addition, one-carrier simulation is used in order to save CPU time. The cross-sectional geometry for each of the MOSFETS is shown in Figure A.10.

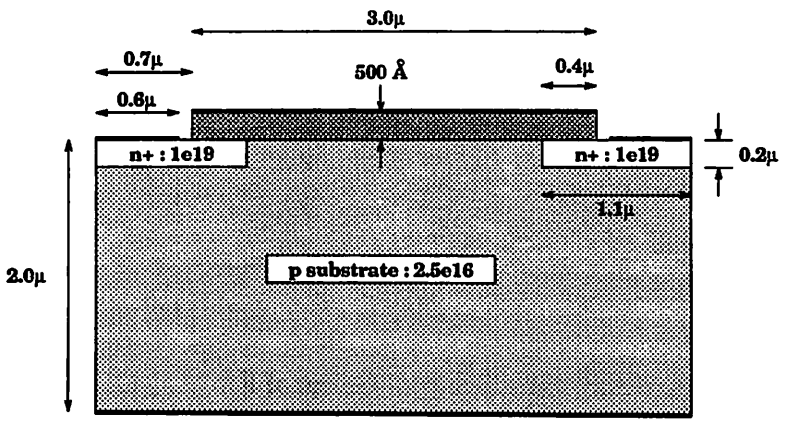


Figure A.10: Geometry of NMOS Transistor

INPUT FILE

```
NMOS Enhancement-Load Bootstrap Inverter

Vdd 1 0 5.0v
Vss 2 0 0.0v
Vin 5 0 0.0v PWL (0.0ns 5.0v) (1ns 0.0v) (10ns 0.0v) (11ns 5.0v)
+ (20ns 5.0v) (21ns 0.0v) (30ns 0.0v) (31ns 5.0v)
M1 1 1 3 2 M_NMOS w=5u
M2 1 3 4 4 M_NMOS w=5u
M3 4 5 2 2 M_NMOS w=5u
CL 4 0 0.1pf
CB 3 4 0.1pf

.model M_NMOS numos
+ x.mesh l=0.0 n=1
+ x.mesh l=0.6 n=4
+ x.mesh l=0.7 n=5
+ x.mesh l=1.0 n=7
+ x.mesh l=1.2 n=11
+ x.mesh l=3.2 n=21
+ x.mesh l=3.4 n=25
+ x.mesh l=3.7 n=27
+ x.mesh l=3.8 n=28
+ x.mesh l=4.4 n=31
+ y.mesh l=-.05 n=1
+ y.mesh l=0.0 n=5
+ y.mesh l=.05 n=9
+ y.mesh l=0.3 n=14
+ y.mesh l=2.0 n=19
+ region num=1 material=1 y.l=0.0
+ material num=1 silicon
+ mobility material=1 concmod=sg fieldmod=sg
+ mobility material=1 init elec major
+ mobility material=1 init elec minor
+ mobility material=1 init hole major
+ mobility material=1 init hole minor
+ region num=2 material=2 y.h=0.0 x.l=0.7 x.h=3.7
+ material num=2 oxide
+ elec num=1 x.l=3.8 x.h=4.4 y.l=0.0 y.h=0.0
+ elec num=2 x.l=0.7 x.h=3.7 iy.l=1 iy.h=1
+ elec num=3 x.l=0.0 x.h=0.6 y.l=0.0 y.h=0.0
+ elec num=4 x.l=0.0 x.h=4.4 y.l=2.0 y.h=2.0
+ doping unif p.type conc=2.5e16 x.l=0.0 x.h=4.4 y.l=0.0 y.h=2.0
+ doping unif p.type conc=1e16 x.l=0.0 x.h=4.4 y.l=0.0 y.h=0.05
+ doping unif n.type conc=1e20 x.l=0.0 x.h=1.1 y.l=0.0 y.h=0.2
+ doping unif n.type conc=1e20 x.l=3.3 x.h=4.4 y.l=0.0 y.h=0.2
+ models concmob fieldmob
+ method ac=direct onec

.TRAN 0.2ns 40ns
```

```
.PRINT TRAN V(4)
.OPTIONS ACCT BYPASS=1 METHOD=GEAR
.END
```

RESULTS

The doping profile of the NMOS transistor is shown in Figure A.11. Figure A.12 shows the important waveforms in the bootstrap inverter. When the input is high, the output is low, the bootstrap capacitor CB is charged to $(V_{DD} - V_{Tn} - V_{ol})$ by transistor M1. When the input voltage drops to 0.0 V, transistor M2 charges the load capacitor CL. Normally, the output voltage would stop rising when it reached $(V_{DD} - V_{Tn})$. However, the stored charge on CB maintains the gate-source voltage of M2 above V_{Tn} and M2 remains on, allowing V_o to reach the full supply voltage V_{DD}. When the input goes high again, the output is quickly discharged by M3, and the voltage across CB is reset to its initial value. Notice that the gate voltage of M2 rises well above the upper supply voltage V_{DD} = 5.0v.

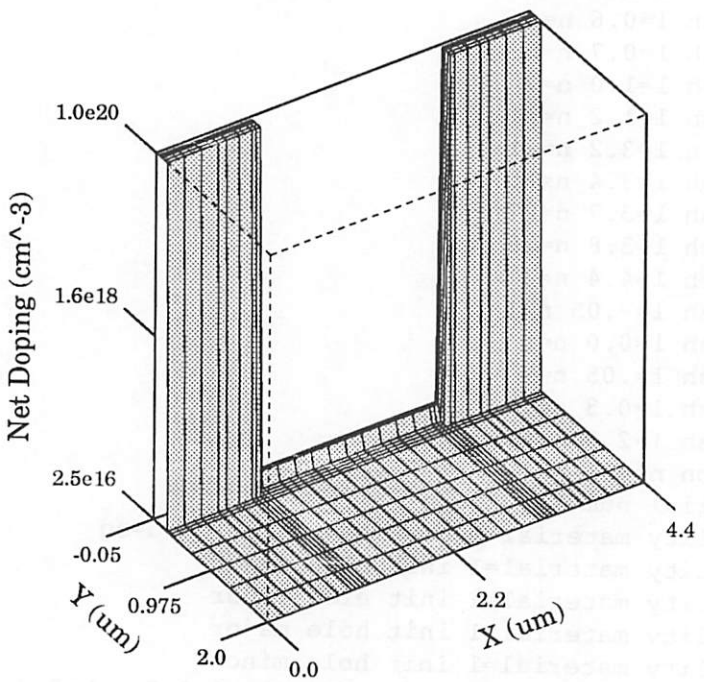


Figure A.11: 2^D NMOSFET Doping Profile

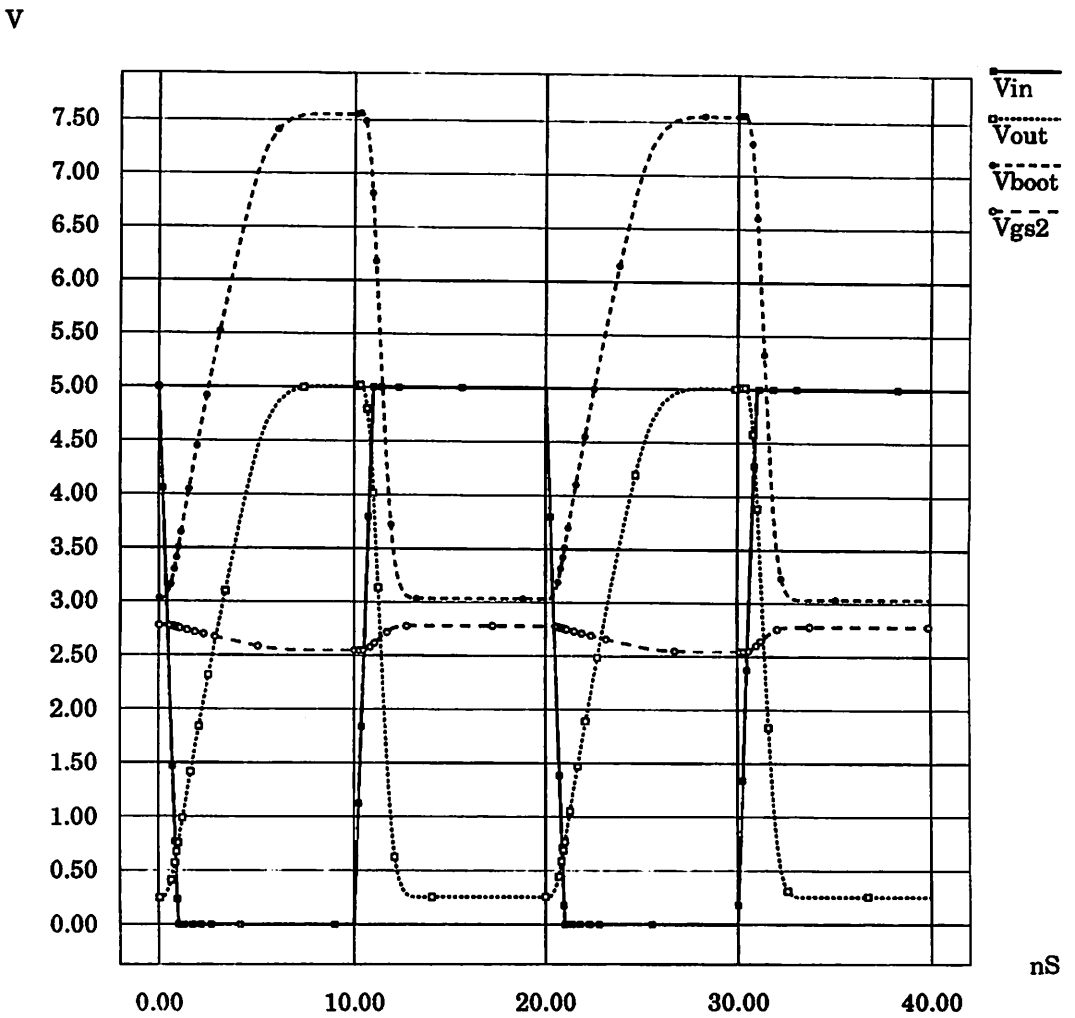


Figure A.12: Output Waveforms of Bootstrap Inverter

NAME

EXAMPLE 4 – Two-Dimensional Doping Profiles

DESCRIPTION

There are several options for specifying two-dimensional doping profiles and it can be confusing to understand how they operate. This example exercises these options by simulating a typical source or drain junction of a MOSFET with a variety of profiles. A contour plot for each of the doping profiles is provided. Since CIDER does not have a self-contained contour-plot capability, these results will be difficult to reproduce unless a separate contouring program is available.

In each case, the bulk is uniformly doped with a P-type concentration of $1.0 \times 10^{16}\text{cm}^{-3}$. The N+ region is varied from case to case. However, the concentration along the upper left surface is always $1.0 \times 10^{20}\text{cm}^{-3}$ from x = 0 to x = 0.5 except for the final two cases. Also, the profile characteristic length has been chosen so that the junction depth is always $0.2\mu\text{m}$.

INPUT FILE

```
TWO-DIMENSIONAL SOURCE DOPING PROFILES

VSS 1 0 0.0v
VBB 2 0 0.6v
D1 1 2 M_SRCJUNC W=10u SAVE

.MODEL M_SRCJUNC NUMD LEVEL=2
+ x.mesh w=1.0 n=50
+ y.mesh w=0.4 n=20
+ domain num=1 material=1
+ material num=1 silicon
+ electrode num=1 x.l=0.0 x.h=0.5 y.h=0.0
+ electrode num=2 y.l=0.4
+ doping unif p.type conc=1.0e16
+ *** (a) Box Uniform ***
+ doping unif n.type conc=1.0e20 x.l=0.0 x.h=0.7 y.h=0.2
+ *** (b) Rounded Uniform
+ * doping unif n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + location=0.2 lat.rotate ratio=1.0
+ *** (c) Linear ***
+ * doping lin n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.2 lat.rotate ratio=1.0
+ *** (d) Exponential ***
+ * doping exp n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0217 lat.rotate ratio=1.0
+ *** (e) Gaussian ***
+ * doping gauss n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0656 lat.rotate ratio=1.0
+ *** (f) Complementary Error-Function ***
+ * doping erfc n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0727 lat.rotate ratio=1.0
+ *** (g) Gaussian - Lateral Ratio 0.5 ***
```

```
+ * doping gauss n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0656 lat.rotate ratio=0.5
+ *** (h) Gaussian - Lateral Erfc ***
+ * doping gauss n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0656 lat.erfc ratio=1.0
+ *** (i) Gaussian - Deep Constant Box ***
+ * doping gauss n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.l=0.1 y.h=0.1
+ * + char.len=0.0328 lat.rotate ratio=1.0
+ *** (j) Gaussian - Deep Peak Location ***
+ * doping gauss n.type conc=1.0e20 x.l=0.0 x.h=0.5 y.h=0.0
+ * + char.len=0.0328 location=0.1 lat.rotate ratio=1.0
+ method onec

.OP
.END
```

RESULTS

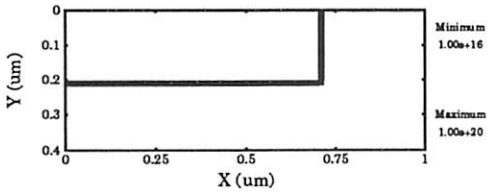
As shown, the input file generates doping profile data for the case when the N+ doping is $1.0 \times 10^{20}cm^{-3}$ down to 0.2 μm and out to 0.7 μm. However, Figure A.13 contains contour plots for each of the ten different profiles defined in the input file. These plots were generated by taking the log of the absolute value of the net doping before generating contours at half decade intervals. This is necessary since the doping varies by orders of magnitude, especially near the N+ - P junction. In some cases a minimum concentration of $1.0 \times 10^{14}cm^{-3}$ was added to prevent the contour program from generating too many contours at the lightly doped junction boundary. Also, in some of the subfigures a single deep contour is generated beneath and to the right of the junction as the doping returns to the substrate concentration. This is due to a slight reduction in the net doping caused by finite N-type impurity concentration coupled with the program generating a contour very near $1.0 \times 10^{16}cm^{-3}$.

Figure A.13(a) shows the profile for the input file as shown. The constant box has been extended by 0.2 μm in each dimension to create the rectangular junction. Since the doping drops abruptly from $1.0 \times 10^{20}cm^{-3}$ to $1.0 \times 10^{16}cm^{-3}$ at the junction, all the contours are right at the junction. Figure A.13(b) shows a similar abrupt junction where the boundary is rounded for x > 0.5. This is accomplished by rotating the primary profile about the x = 0.5, y = 0.0 corner of the constant box. The *location* is set to 0.2 μm so that uniformly doped primary profile is non-zero down to 0.2 μm. In Figure A.13(c), the N+ doping varies linear from $1.0 \times 10^{20}cm^{-3}$ at the surface to $0.0 \times 10^{20}cm^{-3}$ at 0.2 μm. Since the slope of the profile is so large, the junction is almost exactly at that depth as well. Although one might expect the contours to be evenly spaced for a linear profile, they are actually almost all near the junction because the contours are generated at half decade intervals. For the exponential profile (Figure A.13(d)), the contours are evenly spaced. The gaussian and complementary error-function profiles in Figures A.13(e) and A.13(f) both fall off rapidly near the surface like the exponential profile but generate unevenly spaced contours. From the figures it is difficult to differentiate between the two except that fall-off of the gaussian profile is slightly more gradual.

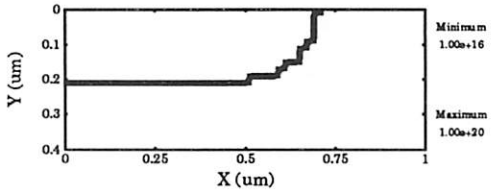
In the remaining portions of the figure, the primary profile is fixed as a gaussian and other parameters are varied. In Figure A.13(g), the lateral ratio is cut in half, so that the contours are no longer circular but are instead elliptical. This can be used to

model the reduced lateral diffusion of dopants relative to vertical diffusion. A similar effect can be created by using a different profile type to control the lateral diffusion as in Figure A.13(h). The contours for $x > 0.5$ are neither circular or elliptical, since the concentration of the N+ profile is now determined by multiplying the falloff factors of the primary and lateral profiles. The junction intersects the surface nearer the left side of the figure because the complementary error-function falls off more rapidly than the gaussian when the characteristic lengths are equal.

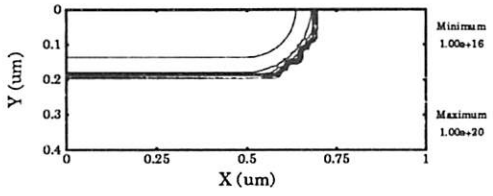
The final two subfigures demonstrate the difference between the constant box and the *location* parameter more clearly than the first two subfigures do. In Figure A.13(i), the constant box (in this case, constant line segment) goes from $x = 0$ to $x = 0.5$ at a depth of 0.1 μm. The concentration there is still $1 \times 10^{20} cm^{-3}$. Since the profile is symmetric about the constant box, the doping drops off both above and below this line; there is now a junction at the surface as well. The two junctions are connected as the profile is rotated about $x = 0.5$, $y = 0.1$. In Figure A.13(j) the constant box is left at the surface and only the peak of the primary profile is moved to a depth of 0.1 μm. Rotation now takes place about $x = 0.5$, $y = 0.0$. Along the left sides of the subfigures, the two profiles are the same. However, the two junctions no longer merge for $x > 0.5$ μm as they do in Figure A.13(i).



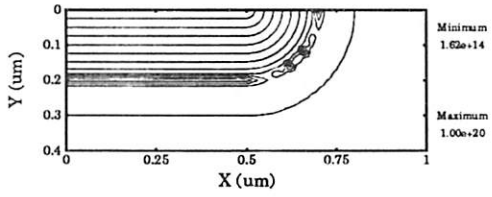
(a) Box Uniform



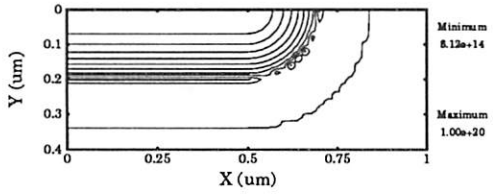
(b) Rounded Uniform



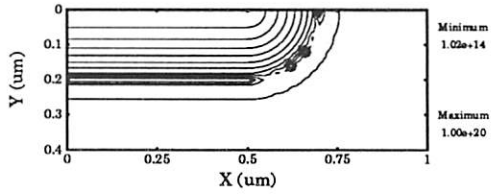
(c) Linear



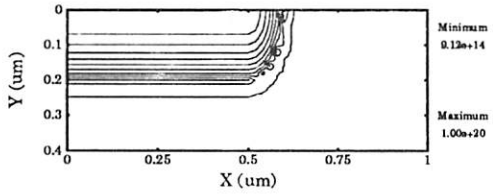
(d) Exponential



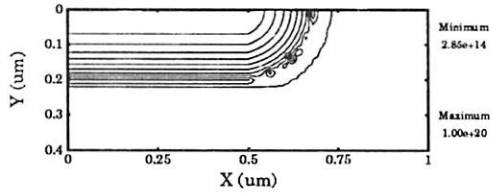
(e) Gaussian



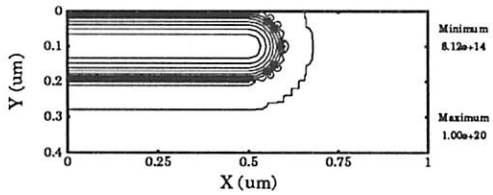
(f) Comp. Error-Function



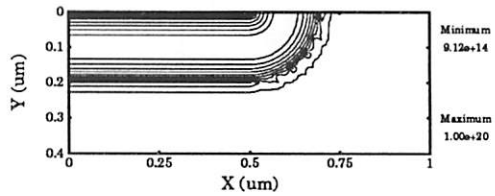
(g) Gaussian - Lateral Ratio 0.5



(h) Gaussian - Lateral Erfc



(i) Gaussian - Deep Constant Box



(j) Gaussian - Deep Peak Location

Figure A.13: Contours of 2^D Doping Profiles

Appendix B

CIDER Serial-Version Benchmarks

ASTABLE Benchmark

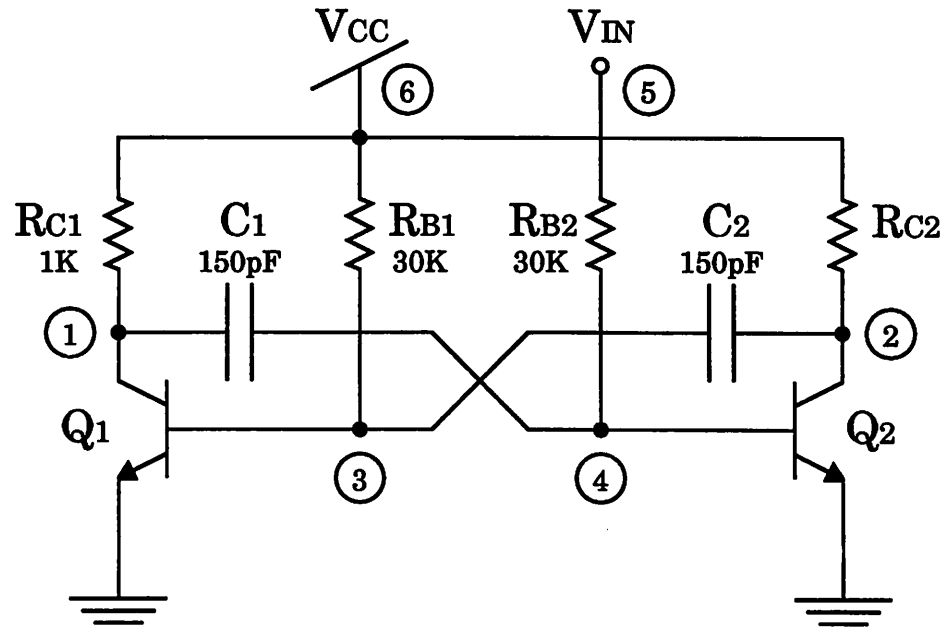


Figure B.1: ASTABLE schematic

```
ASTABLE MULTIVIBRATOR

VIN 5 0 DC 0 PULSE(0 5 0 1US 1US 100US 100US)
VCC 6 0 5.0
RC1 6 1 1K
RC2 6 2 1K
RB1 6 3 30K
RB2 5 4 30K
C1 1 4 150PF
C2 2 3 150PF
Q1 1 3 0 QMOD AREA = 100P
Q2 2 4 0 QMOD AREA = 100P

.OPTION ACCT BYPASS=1
.TRAN 0.05US 8US 0US 0.05US
.PRINT TRAN V(1) V(2) V(3) V(4)

.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
```

```
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.END
```

CHARGE Benchmark

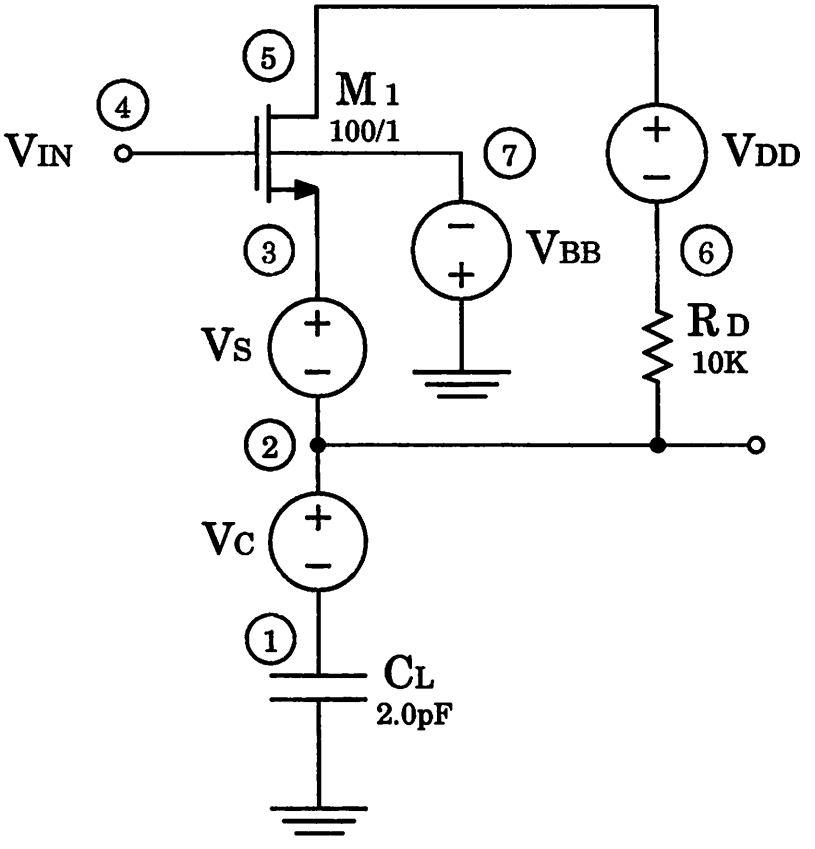


Figure B.2: CHARGE schematic

```
MOS CHARGE PUMP

VIN 4 0 DC 0V PULSE 0 5 15NS 5NS 5NS 50NS 100NS
VDD 5 6 DC 0V PULSE 0 5 25NS 5NS 5NS 50NS 100NS
VBB 0 7 DC 0V PULSE 0 5  0NS 5NS 5NS 50NS 100NS
RD 6 2 10K
M1 5 4 3 7 MMOD W=100UM
VS 3 2 0
VC 2 1 0
C2 1 0 10PF
```

```
.IC V(3)=1.0
.TRAN 2NS 200NS
.OPTIONS ACCT BYPASS=1
.PRINT TRAN V(1) V(2)

.MODEL MMOD NUMOS
+ X.MESH N=1 L=0
+ X.MESH N=3 L=0.4
+ X.MESH N=7 L=0.6
+ X.MESH N=15 L=1.4
+ X.MESH N=19 L=1.6
+ X.MESH N=21 L=2.0
+
+ Y.MESH N=1 L=0
+ Y.MESH N=4 L=0.015
+ Y.MESH N=8 L=0.05
+ Y.MESH N=12 L=0.25
+ Y.MESH N=14 L=0.35
+ Y.MESH N=17 L=0.5
+ Y.MESH N=21 L=1.0
+
+ REGION NUM=1 MATERIAL=1 Y.L=0.015
+ MATERIAL NUM=1 SILICON
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+
+ REGION NUM=2 MATERIAL=2 Y.H=0.015 X.L=0.5 X.H=1.5
+ MATERIAL NUM=2 OXIDE
+
+ ELEC NUM=1 IX.L=18 IX.H=21   IY.L=4  IY.H=4
+ ELEC NUM=2 IX.L=5  IX.H=17   IY.L=1  IY.H=1
+ ELEC NUM=3 IX.L=1  IX.H=4    IY.L=4  IY.H=4
+ ELEC NUM=4 IX.L=1  IX.H=21   IY.L=21 IY.H=21
+
+ DOPING UNIF N.TYPE CONC=1E18   X.L=0.0 X.H=0.5 Y.L=0.015 Y.H=0.25
+ DOPING UNIF N.TYPE CONC=1E18   X.L=1.5 X.H=2.0 Y.L=0.015 Y.H=0.25
+ DOPING UNIF P.TYPE CONC=1E15   X.L=0.0 X.H=2.0 Y.L=0.015 Y.H=1.0
+ DOPING UNIF P.TYPE CONC=1.3E17 X.L=0.5 X.H=1.5 Y.L=0.015 Y.H=0.05
+
+ MODELS CONCMOB FIELDMOB
+ METHOD ONEC

.END
```

COLPOSC Benchmark

```
COLPITT'S OSCILLATOR CIRCUIT
```

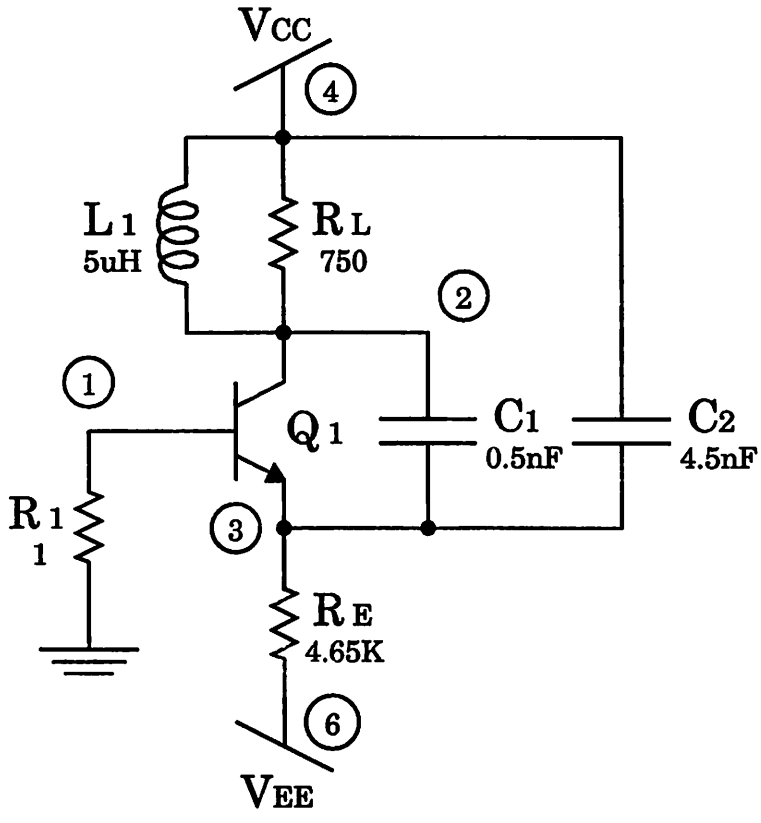


Figure B.3: COLPOSC schematic

```
R1 1 0 1
Q1 2 1 3 QMOD AREA = 100P
VCC 4 0 5
RL 4 2 750
C1 2 3 500P
C2 4 3 4500P
L1 4 2 5UH
RE 3 6 4.65K
VEE 6 0 DC -15 PWL 0 -15 1E-9 -10

.TRAN 30N 12U
.PRINT TRAN V(2)

.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
```

```
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.OPTIONS ACCT BYPASS=1
.END
```

DBRIDGE Benchmark

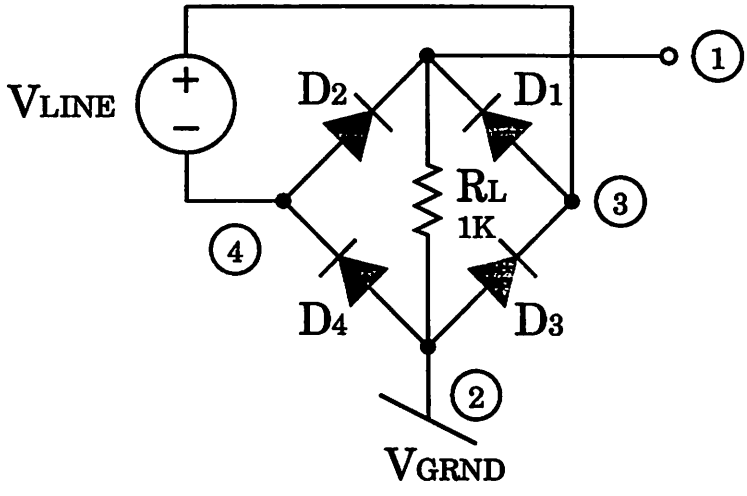


Figure B.4: DBRIDGE schematic

```
DIODE BRIDGE RECTIFIER

VLINE 3 4 0.0V SIN 0V 10V 60HZ
VGRND 2 0 0.0V
D1 3 1 M_PN AREA=100
D2 4 1 M_PN AREA=100
D3 2 3 M_PN AREA=100
D4 2 4 M_PN AREA=100
RL 1 2 1.0K

.MODEL M_PN NUMD LEVEL=1
+ ************************************
+ *** ONE-DIMENSIONAL NUMERICAL DIODE ***
+ ************************************
+ OPTIONS DEFA=1P
+ X.MESH LOC=0.0 N=1
+ X.MESH LOC=30.0 N=201
+ DOMAIN   NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON
+ MOBILITY MAT=1 CONCMOD=CT FIELDMOD=CT
```

```
+ DOPING GAUSS P.TYPE CONC=1E20 X.L=0.0  X.H=0.0 CHAR.L=1.0
+ DOPING UNIF  N.TYPE CONC=1E14 X.L=0.0  X.H=30.0
+ DOPING GAUSS N.TYPE CONC=5E19 X.L=30.0  X.H=30.0 CHAR.L=2.0
+ MODELS BGN ÂVAL SRH AUGER CONCTAU CONCMOB FIELDMOB
+ METHOD AC=DIRECT

.OPTION ACCT BYPASS=1 METHOD=GEAR
.TRAN 0.5MS 50MS
.PRINT I(VLINE)
.END
```

INVCHAIN Benchmark

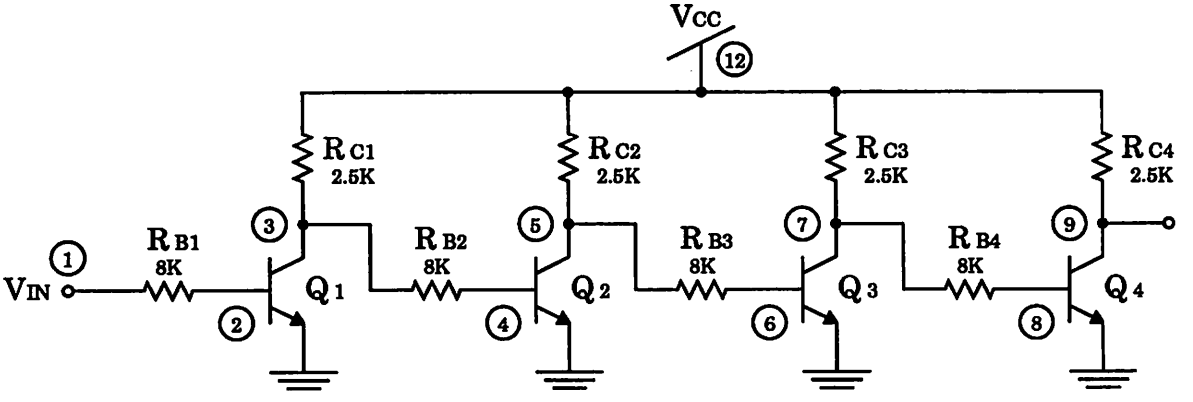


Figure B.5: INVCHAIN schematic

```
4 STAGE RTL INVERTER CHAIN

VIN 1 0 DC 0V PWL 0NS 0V 1NS 5V
VCC 12 0 DC 5.0V
RC1 12 3 2.5K
RB1 1 2 8K
Q1 3 2 0 QMOD AREA = 100P
RB2 3 4 8K
RC2 12 5 2.5K
Q2 5 4 0 QMOD AREA = 100P
RB3 5 6 8K
RC3 12 7 2.5K
Q3 7 6 0 QMOD AREA = 100P
RB4 7 8 8K
RC4 12 9 2.5K
Q4 9 8 0 QMOD AREA = 100P

.PRINT TRAN V(3) V(5) V(9)
```

```
.TRAN 1E-9 10E-9

.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.OPTION ACCT BYPASS=1
.END
```

MECLGATE Benchmark

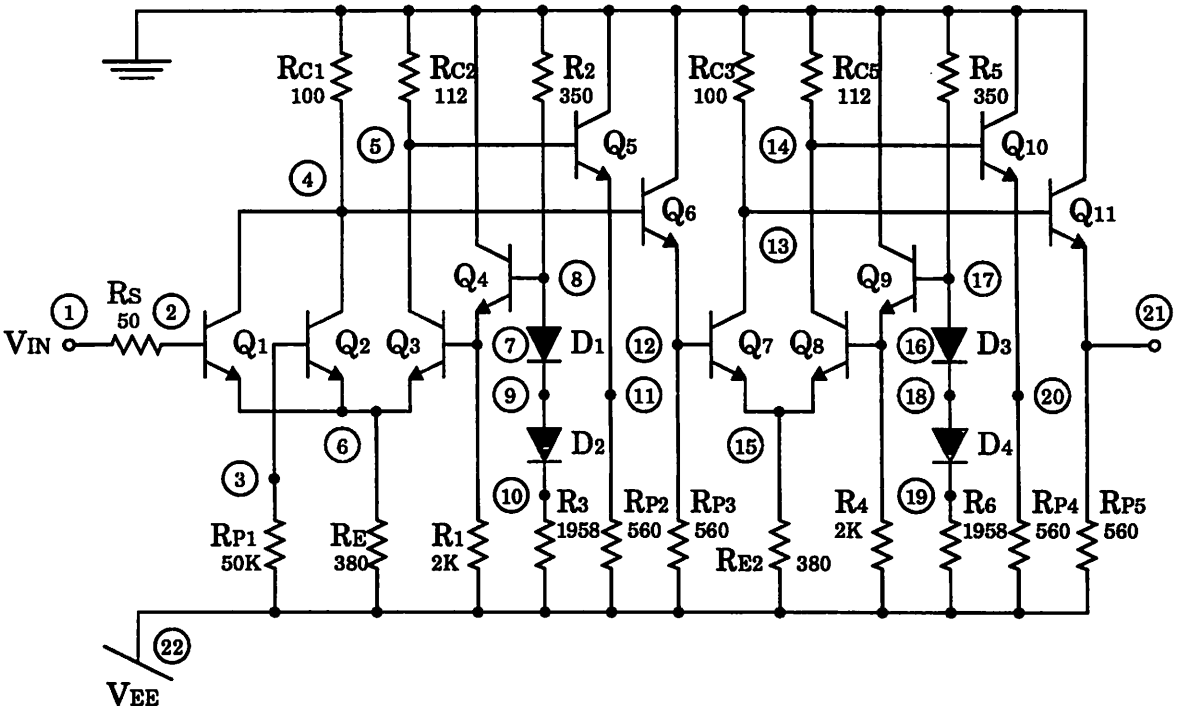


Figure B.6: MECLGATE schematic

```
MOTOROLA MECL III ECL GATE
*.DC VIN -2.0 0 0.02
.TRAN 0.2NS 20NS
```

```
VEE 22 0 -6.0
VIN 1 0 PULSE -0.8 -1.8 0.2NS 0.2NS 0.2NS 10NS 20NS
RS 1 2 50
Q1 4 2 6 QMOD AREA = 100P
Q2 4 3 6 QMOD AREA = 100P
Q3 5 7 6 QMOD AREA = 100P
Q4 0 8 7 QMOD AREA = 100P

D1 8 9 DMOD
D2 9 10 DMOD

RP1 3 22 50K
RC1 0 4 100
RC2 0 5 112
RE 6 22 380
R1 7 22 2K
R2 0 8 350
R3 10 22 1958

Q5 0 5 11 QMOD AREA = 100P
Q6 0 4 12 QMOD AREA = 100P

RP2 11 22 560
RP3 12 22 560

Q7 13 12 15 QMOD AREA = 100P
Q8 14 16 15 QMOD AREA = 100P

RE2 15 22 380
RC3 0 13 100
RC4 0 14 112

Q9 0 17 16 QMOD AREA = 100P

R4 16 22 2K
R5 0 17 350
D3 17 18 DMOD
D4 18 19 DMOD
R6 19 22 1958

Q10 0 14 20 QMOD AREA = 100P
Q11 0 13 21 QMOD AREA = 100P

RP4 20 22 560
RP5 21 22 560

.MODEL DMOD D RS=40 TT=0.1NS CJO=0.9PF N=1 IS=1E-14 EG=1.11 VJ=0.8 M=0.5

.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
```

```
+ X.MESH NODE=10 LOC=0.9
+ X.MESH NODE=20 LOC=1.1
+ X.MESH NODE=30 LOC=1.4
+ X.MESH NODE=40 LOC=1.6
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.OPTIONS ACCT BYPASS=1
.PRINT TRAN V(12) V(21)
.END
```

NMOSINV Benchmark

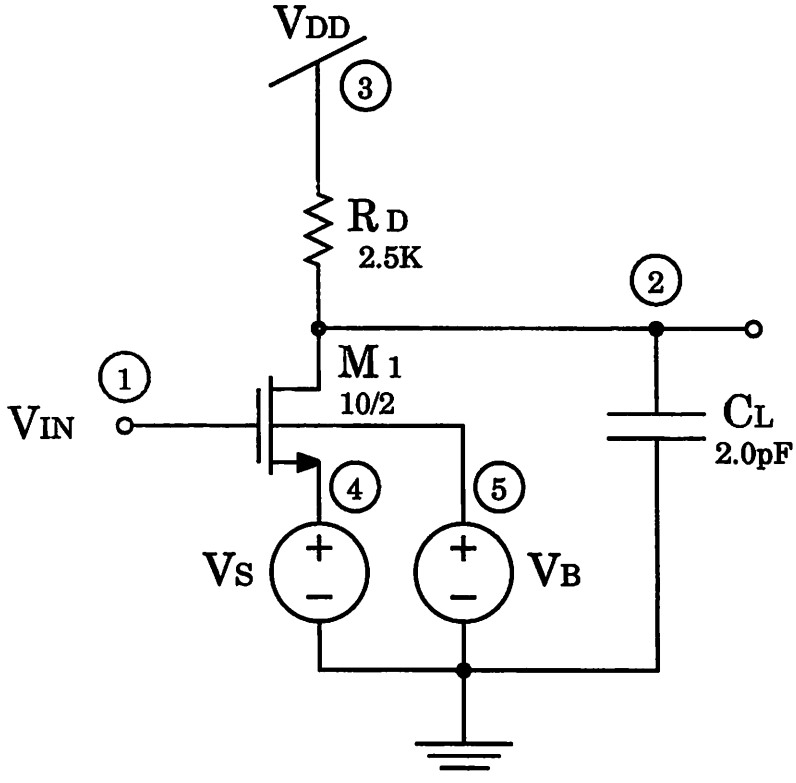


Figure B.7: NMOSINV schematic

```
RESISTIVE LOAD NMOS INVERTER
VIN 1 0 PWL 0 0.0 2NS 5
VDD 3 0 DC 5.0
RD 3 2 2.5K
M1 2 1 4 5 MMOD W=10UM
CL 2 0 2PF
VB 5 0 0
VS 4 0 0

.MODEL MMOD NUMOS
+ X.MESH L=0.0 N=1
+ X.MESH L=0.6 N=4
+ X.MESH L=0.7 N=5
+ X.MESH L=1.0 N=7
+ X.MESH L=1.2 N=11
+ X.MESH L=3.2 N=21
+ X.MESH L=3.4 N=25
+ X.MESH L=3.7 N=27
+ X.MESH L=3.8 N=28
+ X.MESH L=4.4 N=31
+
+ Y.MESH L=-.05 N=1
+ Y.MESH L=0.0  N=5
+ Y.MESH L=.05  N=9
+ Y.MESH L=0.3  N=14
+ Y.MESH L=2.0  N=19
+
+ REGION NUM=1 MATERIAL=1 Y.L=0.0
+ MATERIAL NUM=1 SILICON
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+
+ REGION NUM=2 MATERIAL=2 Y.H=0.0 X.L=0.7 X.H=3.7
+ MATERIAL NUM=2 OXIDE
+
+ ELEC NUM=1 X.L=3.8 X.H=4.4 Y.L=0.0 Y.H=0.0
+ ELEC NUM=2 X.L=0.7 X.H=3.7 IY.L=1  IY.H=1
+ ELEC NUM=3 X.L=0.0 X.H=0.6 Y.L=0.0 Y.H=0.0
+ ELEC NUM=4 X.L=0.0 X.H=4.4 Y.L=2.0 Y.H=2.0
+
+ DOPING UNIF P.TYPE CONC=2.5E16 X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=2.0
+ DOPING UNIF P.TYPE CONC=1E16   X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=0.05
+ DOPING UNIF N.TYPE CONC=1E20   X.L=0.0 X.H=1.1  Y.L=0.0 Y.H=0.2
+ DOPING UNIF N.TYPE CONC=1E20   X.L=3.3 X.H=4.4  Y.L=0.0 Y.H=0.2
+
+ MODELS CONCMOB FIELDMOB
+ METHOD AC=DIRECT ONEC

.TRAN 0.2NS 30NS
.OPTIONS ACCT BYPASS=1
.PRINT TRAN V(1) V(2)
```

```
.END
```

PASS Benchmark

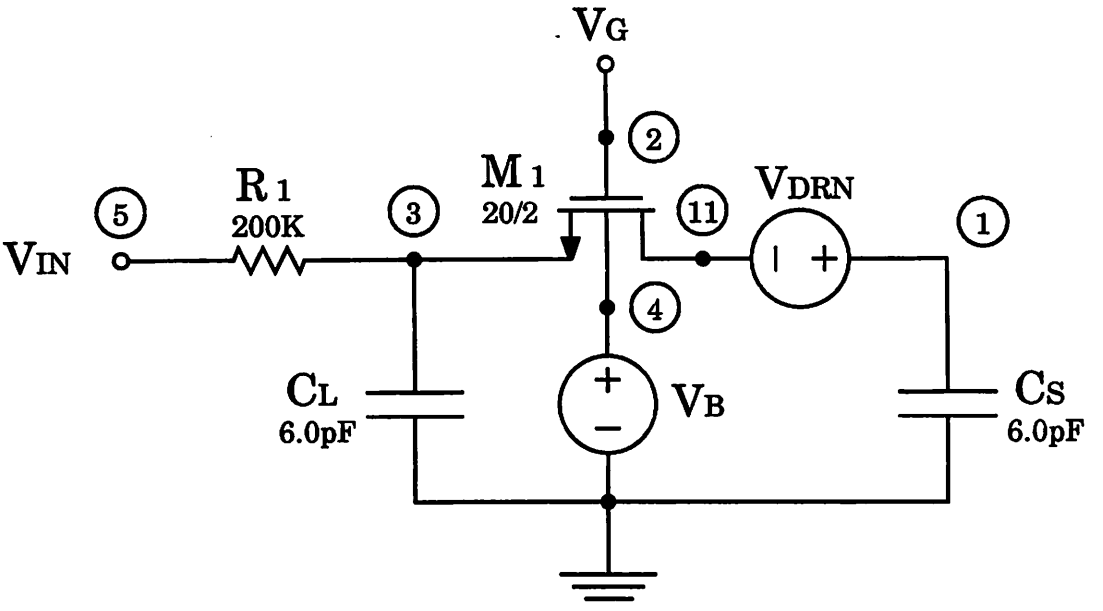


Figure B.8: PASS schematic

```
TURNOFF TRANSIENT OF PASS TRANSISTOR

M1 11 2 3 4 MMOD W=20UM
CS 1 0 6.0PF
CL 3 0 6.0PF
R1 3 6 200K
VIN 6 0 DC 0
VDRN 1 11 DC 0
VG 2 0 DC 5 PWL 0 5 0.1N 0 1 0
VB 4 0 DC 0.0

.TRAN 0.05NS 0.2NS 0.0NS 0.05NS
.PRINT TRAN V(1) I(VDRN)
.IC V(1)=0 V(3)=0
.OPTION ACCT BYPASS=1

.MODEL MMOD NUMOS
+ X.MESH L=0.0 N=1
+ X.MESH L=0.6 N=4
+ X.MESH L=0.7 N=5
+ X.MESH L=1.0 N=7
```

```
+ X.MESH L=1.2 N=11
+ X.MESH L=3.2 N=21
+ X.MESH L=3.4 N=25
+ X.MESH L=3.7 N=27
+ X.MESH L=3.8 N=28
+ X.MESH L=4.4 N=31
+
+ Y.MESH L=-.05 N=1
+ Y.MESH L=0.0  N=5
+ Y.MESH L=.05  N=9
+ Y.MESH L=0.3  N=14
+ Y.MESH L=2.0  N=19
+
+ REGION NUM=1 MATERIAL=1 Y.L=0.0
+ MATERIAL NUM=1 SILICON
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+
+ REGION NUM=2 MATERIAL=2 Y.H=0.0 X.L=0.7 X.H=3.7
+ MATERIAL NUM=2 OXIDE
+
+ ELEC NUM=1 X.L=3.8 X.H=4.4 Y.L=0.0 Y.H=0.0
+ ELEC NUM=2 X.L=0.7 X.H=3.7 IY.L=1  IY.H=1
+ ELEC NUM=3 X.L=0.0 X.H=0.6 Y.L=0.0 Y.H=0.0
+ ELEC NUM=4 X.L=0.0 X.H=4.4 Y.L=2.0 Y.H=2.0
+
+ DOPING UNIF P.TYPE CONC=2.5E16 X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=2.0
+ DOPING UNIF P.TYPE CONC=1E16   X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=0.05
+ DOPING UNIF N.TYPE CONC=1E20   X.L=0.0 X.H=1.1  Y.L=0.0 Y.H=0.2
+ DOPING UNIF N.TYPE CONC=1E20   X.L=3.3 X.H=4.4  Y.L=0.0 Y.H=0.2
+
+ MODELS CONCMOB FIELDMOB
+ METHOD AC=DIRECT ONEC

.END
```

RTLINV Benchmark

```
RTL INVERTER

VIN 1 0 DC 1 PWL 0 4 1NS 0
VCC 12 0 DC 5.0
RC1 12 3 2.5K
RB1 1 2 8K
Q1 3 2 0 QMOD AREA = 100P

.OPTION ACCT BYPASS=1
.TRAN 0.5N 5N
.PRINT TRAN V(2) V(3)
```

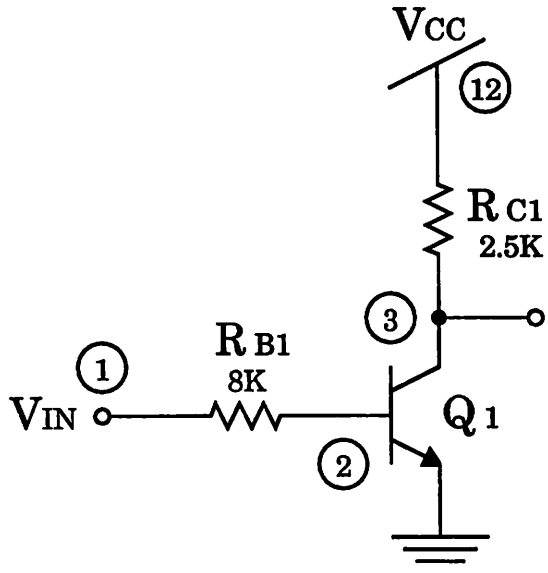


Figure B.9: RTLINV schematic

```
.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.END
```

VCO Benchmark

```
VOLTAGE CONTROLLED OSCILLATOR

RC1 7 5 1K
RC2 7 6 1K

Q5 7 7 5 QMOD AREA = 100P
Q6 7 7 6 QMOD AREA = 100P

Q3 7 5 2 QMOD AREA = 100P
```

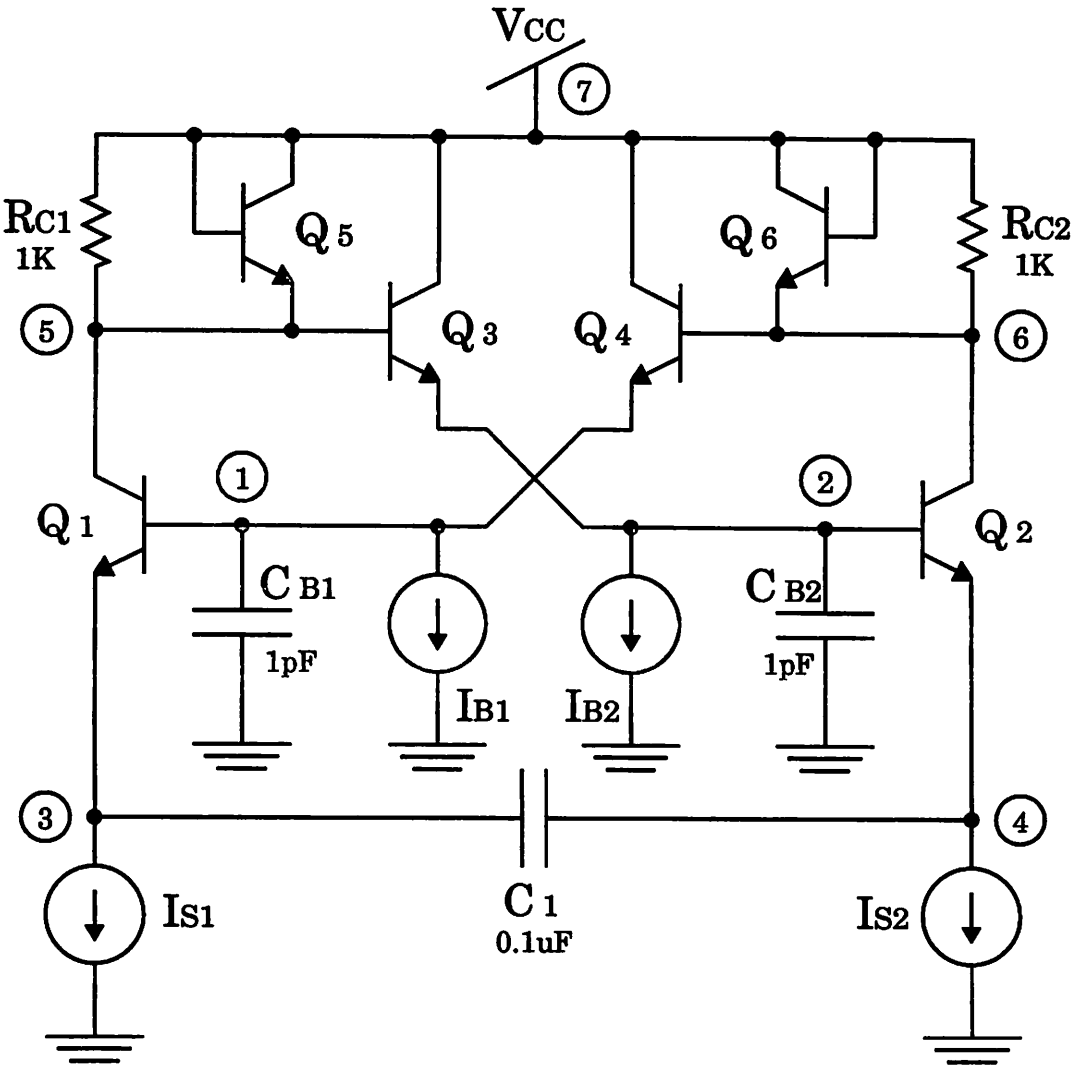


Figure B.10: VCO schematic

```
Q4 7 6 1 QMOD AREA = 100P

IB1 2 0 .5MA
IB2 1 0 .5MA
CB1 2 0 1PF
CB2 1 0 1PF

Q1 5 1 3 QMOD AREA = 100P
Q2 6 2 4 QMOD AREA = 100P

C1 3 4 .1UF

IS1 3 0 DC 2.5MA PULSE 2.5MA 0.5MA 0 1US 1US 50MS
```

```
IS2 4 0 1MA
VCC 7 0 10

.MODEL QMOD NBJT LEVEL=1
+ X.MESH NODE=1  LOC=0.0
+ X.MESH NODE=61 LOC=3.0
+ REGION NUM=1 MATERIAL=1
+ MATERIAL NUM=1 SILICON NBGNN=1E17 NBGNP=1E17
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+ DOPING UNIF N.TYPE CONC=1E17 X.L=0.0 X.H=1.0
+ DOPING UNIF P.TYPE CONC=1E16 X.L=0.0 X.H=1.5
+ DOPING UNIF N.TYPE CONC=1E15 X.L=0.0 X.H=3.0
+ MODELS BGNW SRH CONCTAU AUGER CONCMOB FIELDMOB
+ OPTIONS BASE.LENGTH=1.0 BASE.DEPTH=1.25

.OPTION ACCT BYPASS=1
.TRAN 3US 600US 0 3US
.PRINT TRAN V(4)
.END
```

Appendix C

CIDER Parallel-Version Benchmarks

BICMPD Benchmark

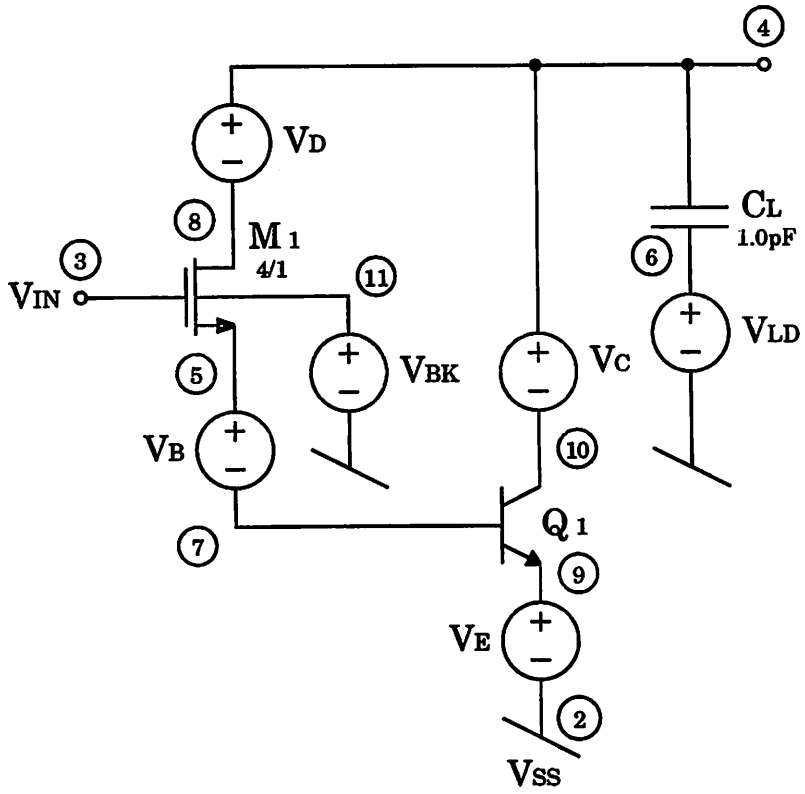


Figure C.1: BICMPD schematic

```
BICMOS INVERTER PULLDOWN CIRCUIT

VSS 2 0 0V

VIN 3 2 0V (PULSE 0.0V 4.2V 0NS 1NS 1NS 9NS 20NS)

M1   8 3 5 11 M_NMOS_1 W=4U L=1U
VD   4 8 0V
VBK 11 2 0V

Q1   10 7 9    M_NPNS AREA=8
VC   4 10 0V
VB   5 7 0V
VE   9 2 0V

CL   4 6 1PF
VL   6 2 0V

.IC V(10)=5.0V V(7)=0.0V
.TRAN 0.1NS 5NS 0NS 0.1NS
```

```
.PLOT TRAN I(VIN)

.INCLUDE BICMOS.LIB

.OPTIONS ACCT BYPASS=1
.END
```

BICMPU Benchmark

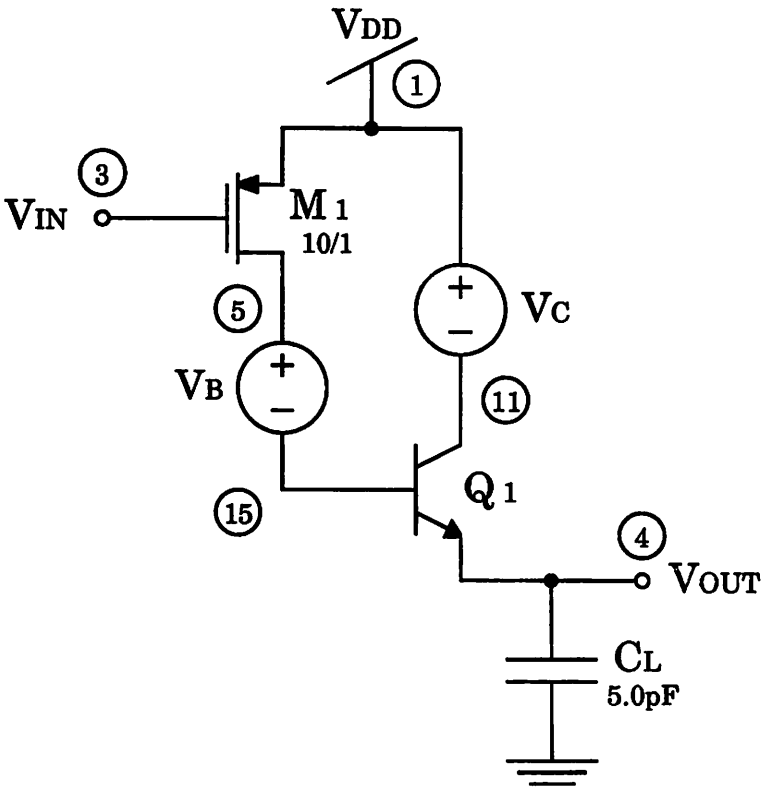


Figure C.2: BICMPU schematic

```
BICMOS INVERTER PULLUP CIRCUIT

VDD 1 0 5.0V
VSS 2 0 0.0V

VIN 3 0 0.75V

VC  1 11 0.0V
VB  5 15 0.0V
```

```
Q1   11 15 4 M_NPNS    AREA=8
M1   5 3 1 1 M_PMOS_1 W=10U L=1U

CL   4 0 5.0PF

.IC V(4)=0.75V V(5)=0.0V

.INCLUDE BICMOS.LIB

.TRAN 0.5NS 4.0NS
.PRINT TRAN V(3) V(4)

.OPTION ACCT BYPASS=1
.END
```

CLKFEED Benchmark

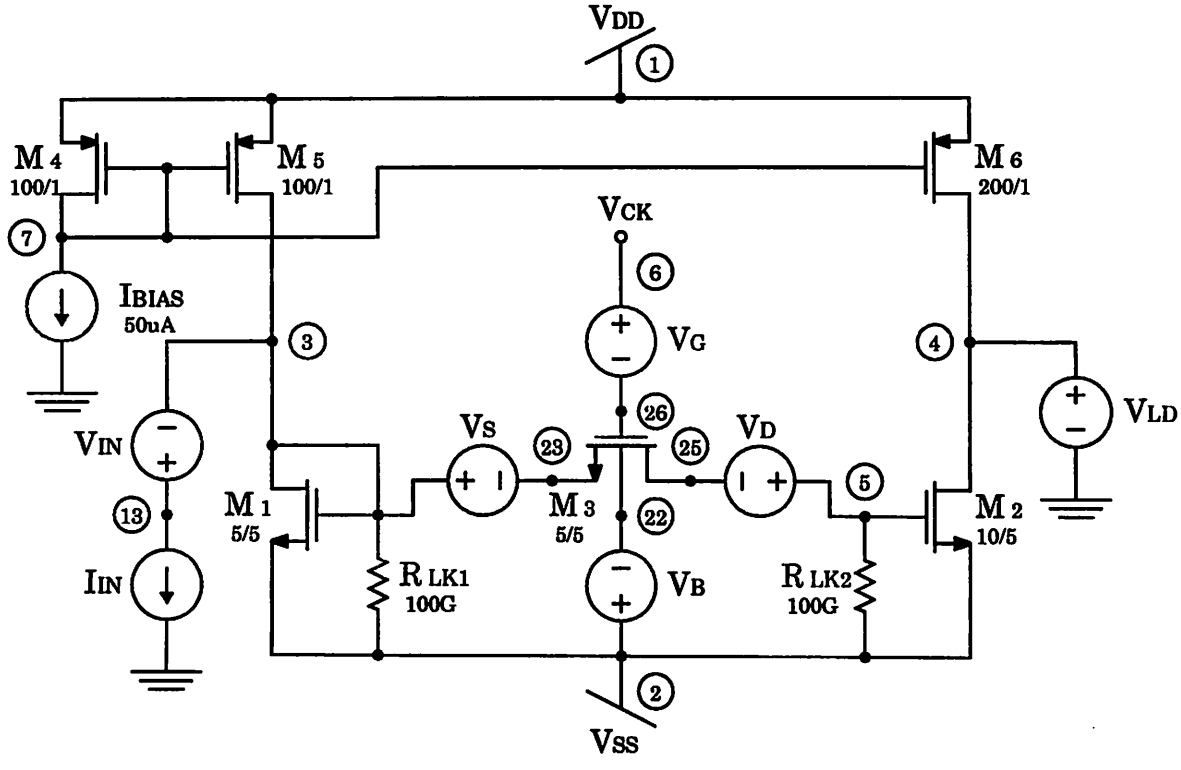


Figure C.3: CLKFEED schematic

```
SWITCHED CURRENT CELL - CLOCK FEEDTHROUGH

VDD 1 0 5.0V
```

```
VSS 2 0 0.0V

IIN 13 0 0.0
VIN 13 3 0.0
VL  4 0 2.5V
VCK 6 0 5.0V PULSE 5.0V 0.0V 5.0NS 5NS 5NS 20NS 50NS

M1   3  3  2  2 M_NMOS_5 W=5U L=5U
M2   4  5  2  2 M_NMOS_5 W=10U L=5U
M3  23 26 25 22 M_NMOS_5 W=5U L=5U
RLK1 3 0 100G
RLK2 5 0 100G
VD  3 23 0.0V
VG  6 26 0.0V
VS  5 25 0.0V
VB  2 22 0.0V

M4  7 7 1 1 M_PMOS_IDEAL  W=100U L=1U
M5  3 7 1 1 M_PMOS_IDEAL  W=100U L=1U
M6  4 7 1 1 M_PMOS_IDEAL  W=200U L=1U
IREF 7 0 50UA

****** MODELS ******
.MODEL M_PMOS_IDEAL PMOS VTO=-1.0V KP=100U

.INCLUDE BICMOS.LIB

.TRAN 0.1NS 50NS

.OPTIONS ACCT BYPASS=1 METHOD=GEAR
.END
```

CMOSAMP Benchmark

```
CMOS 2-STAGE OPERATIONAL AMPLIFIER

VDD 1 0  2.5V
VSS 2 0 -2.5V

IBIAS 9 0 100UA

VPL 3 0 0.0V AC 0.5V
VMI 4 0 0.0V AC 0.5V 180

M1  6 3 5 5 M_PMOS_1 W=15U  L=1U
M2  7 4 5 5 M_PMOS_1 W=15U  L=1U
M3  6 6 2 2 M_NMOS_1 W=7.5U L=1U
M4  7 6 2 2 M_NMOS_1 W=7.5U L=1U
```

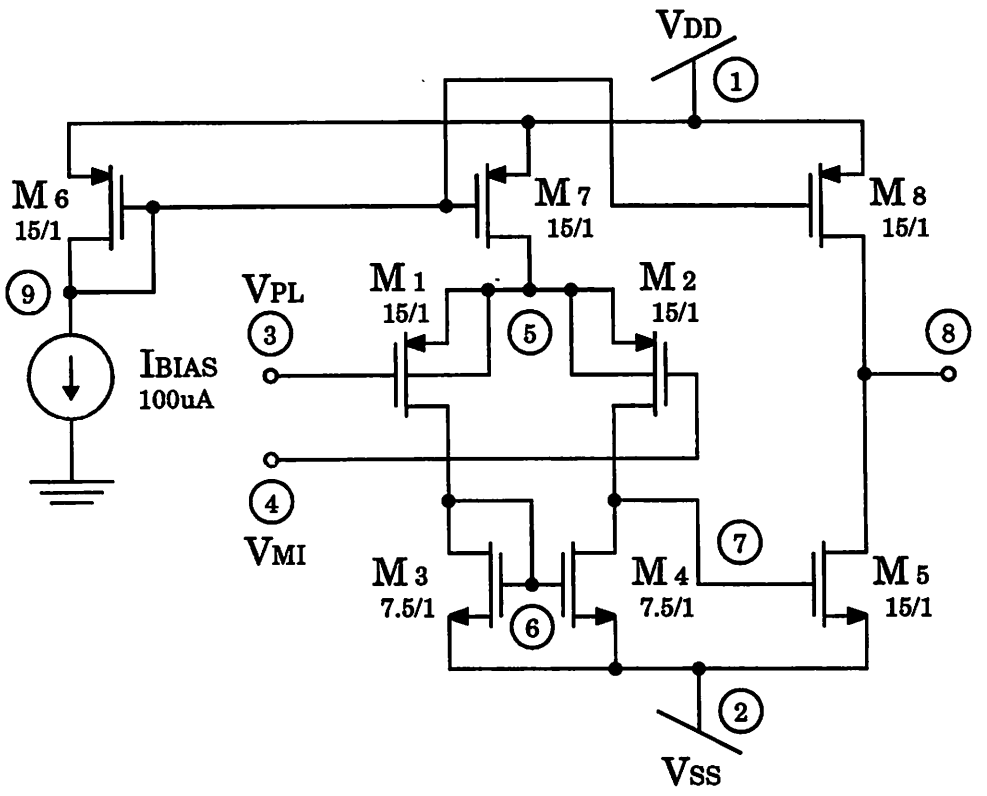


Figure C.4: CMOSAMP schematic

```
M5   8 7 2 2 M_NMOS_1 W=15U  L=1U
M6   9 9 1 1 M_PMOS_1 W=15U  L=1U
M7   5 9 1 1 M_PMOS_1 W=15U  L=1U
M8   8 9 1 1 M_PMOS_1 W=15U  L=1U

*CC  7 8 0.1PF

.INCLUDE BICMOS.LIB

*.OP
*.AC DEC 10 1K 100G
.DC VPL -5MV 5MV 0.1MV

.OPTIONS ACCT BYPASS=1 METHOD=GEAR
.END
```

ECLINV Benchmark

```
ECL INVERTER
```

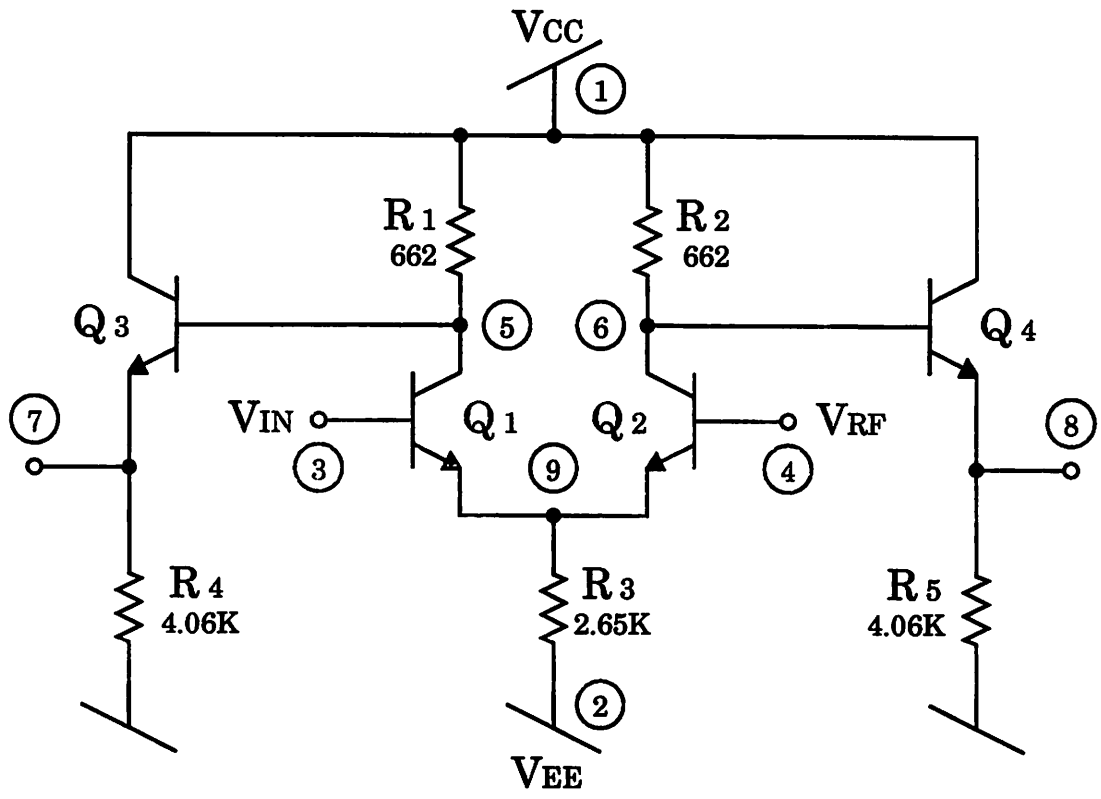


Figure C.5: ECLINV schematic

```
*** (FROM MEINERZHAGEN ET AL.)

VCC 1 0 0.0V
VEE 2 0 -5.2V

VIN 3 0 -1.25V
VRF 4 0 -1.25V

*** INPUT STAGE
Q1 5 3 9 M_NPNS AREA=8
Q2 6 4 9 M_NPNS AREA=8
R1 1 5 662
R2 1 6 662
R3 9 2 2.65K

*** OUTPUT BUFFERS
Q3 1 5 7 M_NPNS AREA=8
Q4 1 6 8 M_NPNS AREA=8
R4 7 2 4.06K
R5 8 2 4.06K

*** MODEL LIBRARY
```

```
.INCLUDE BICMOS.LIB

.DC VIN -2.00 0.001 0.05
.PLOT DC V(7) V(8)

.OPTIONS ACCT BYPASS=1
.END
```

ECPAL Benchmark

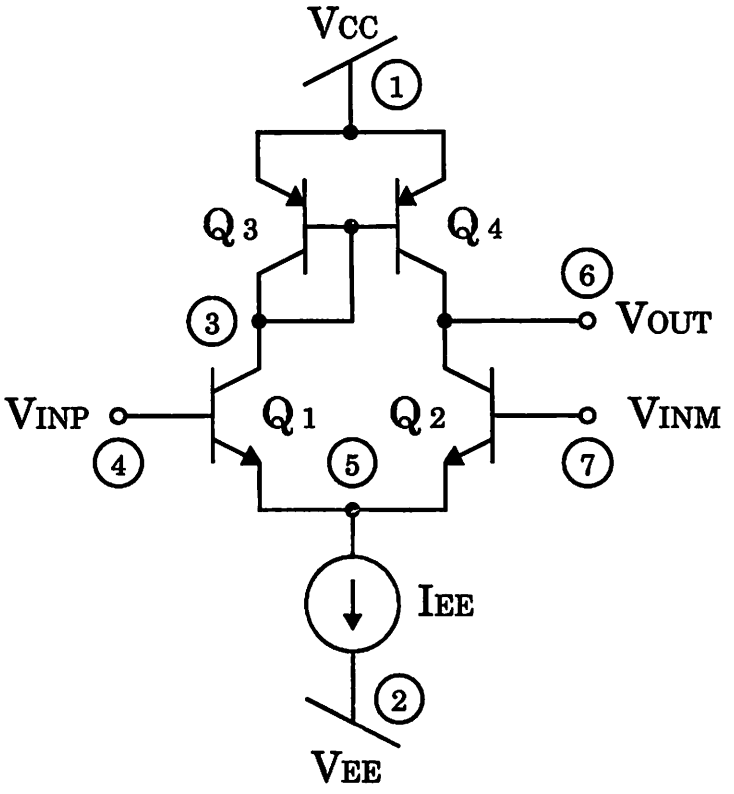


Figure C.6: ECPAL schematic

```
EMITTER COUPLED PAIR WITH ACTIVE LOAD

VCC 1 0 5V
VEE 2 0 0V
VINP 4 0 2.99925V  AC 0.5V
VINM 7 0 3V        AC 0.5V 180
IEE 5 2 0.1MA
Q1 3 4 5 M_NPNS AREA=8
Q2 6 7 5 M_NPNS AREA=8
```

```
Q3 3 3 1 M_PNPS AREA=8
Q4 6 3 1 M_PNPS AREA=8

.AC DEC 10 10K 100G
.PLOT AC VDB(6)

.INCLUDE BICMOS.LIB

.OPTIONS ACCT RELTOL=1E-6
.END
```

GMAMP Benchmark

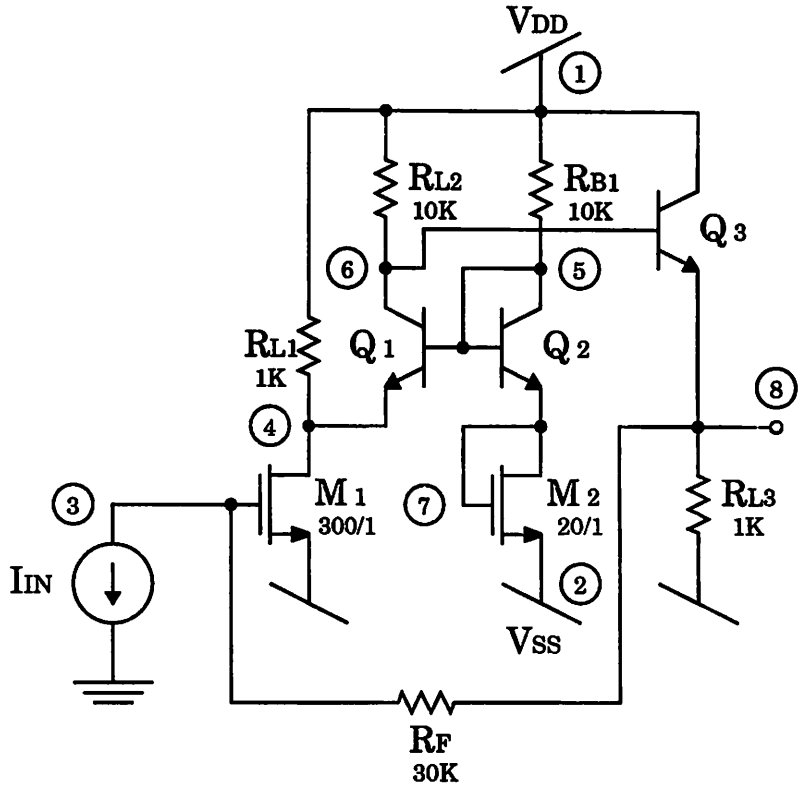


Figure C.7: GMAMP schematic

```
BICMOS 3-STAGE AMPLIFIER
*** IN GRAY & MEYER, 3RD ED. P.266, PROB. 3.12, 8.19

VDD 1 0 5.0V
VSS 2 0 0.0V
```

```
*** VOLTAGE INPUT
*VIN 13 0 0.0V AC 1V
*CIN 13 3 1UF

*** CURRENT INPUT
IIN 3 0 0.0 AC 1.0

M1   4 3 2 2 M_NMOS_1 W=300U L=1U
M2   7 7 2 2 M_NMOS_1 W=20U  L=1U

Q1   6 5 4   M_NPNS AREA=40
Q2   5 5 7   M_NPNS AREA=40
Q3   1 6 8   M_NPNS AREA=40

RL1 1 4      1K
RL2 1 6      10K
RB1 1 5      10K
RL3 8 2      1K
RF1 3 8      30K

*** NUMERICAL MODEL LIBRARY ***
.INCLUDE BICMOS.LIB

.AC DEC 10 100KHZ 100GHZ
.PLOT AC VDB(8)

.OPTIONS ACCT BYPASS=1 KEEPOPINFO
.END
```

LATCH Benchmark

```
STATIC LATCH
***    IC=1MA, RE6=3K
***    SPICE ORIGINAL 1-7-80, CIDER REVISED 4-16-93

***    BIAS CIRCUIT
***    RESISTORS
RCC2 6 8 3.33K
REE2 9 0 200
***    TRANSISTORS
Q1 6 8 4 M_NPN1D AREA=8
Q2 8 4 9 M_NPN1D AREA=8

***    MODELS
.INCLUDE BICMOS.LIB

***    SOURCES
VCC 6 0 5V
```

```
VREF 3 0 2.5V
VRSET 1 0 PULSE(2V 3V 0.1NS 0.1NS 0.1NS 0.9NS 4NS)
VSET  7 0 PULSE(2V 3V 2.1NS 0.1NS 0.1NS 0.9NS 4NS)

***   LATCH
X1 1 2 3 4 5 6 ECLNOR2
X2 5 7 3 4 2 6 ECLNOR2

***   SUBCIRCUITS
.SUBCKT ECLNOR2 1 2 3 4 5 6
**  RESISTORS
RS 6 11 520
RC2 11 10 900
RE4 12 0 200
RE6 5 0 6K
**  TRANSISTORS
Q1 9 1 8 M_NPN1D AREA=8
Q2 9 2 8 M_NPN1D AREA=8
Q3 11 3 8 M_NPN1D AREA=8
Q4 8 4 12 M_NPN1D AREA=8
Q5 10 10 9 M_NPN1D AREA=8
Q6 6 9 5 M_NPN1D AREA=8
.ENDS ECLNOR2

***   CONTROL CARDS
.TRAN 0.01NS 8NS
.PRINT TRAN V(1) V(7) V(5) V(2)
.OPTIONS ACCT BYPASS=1
.END
```

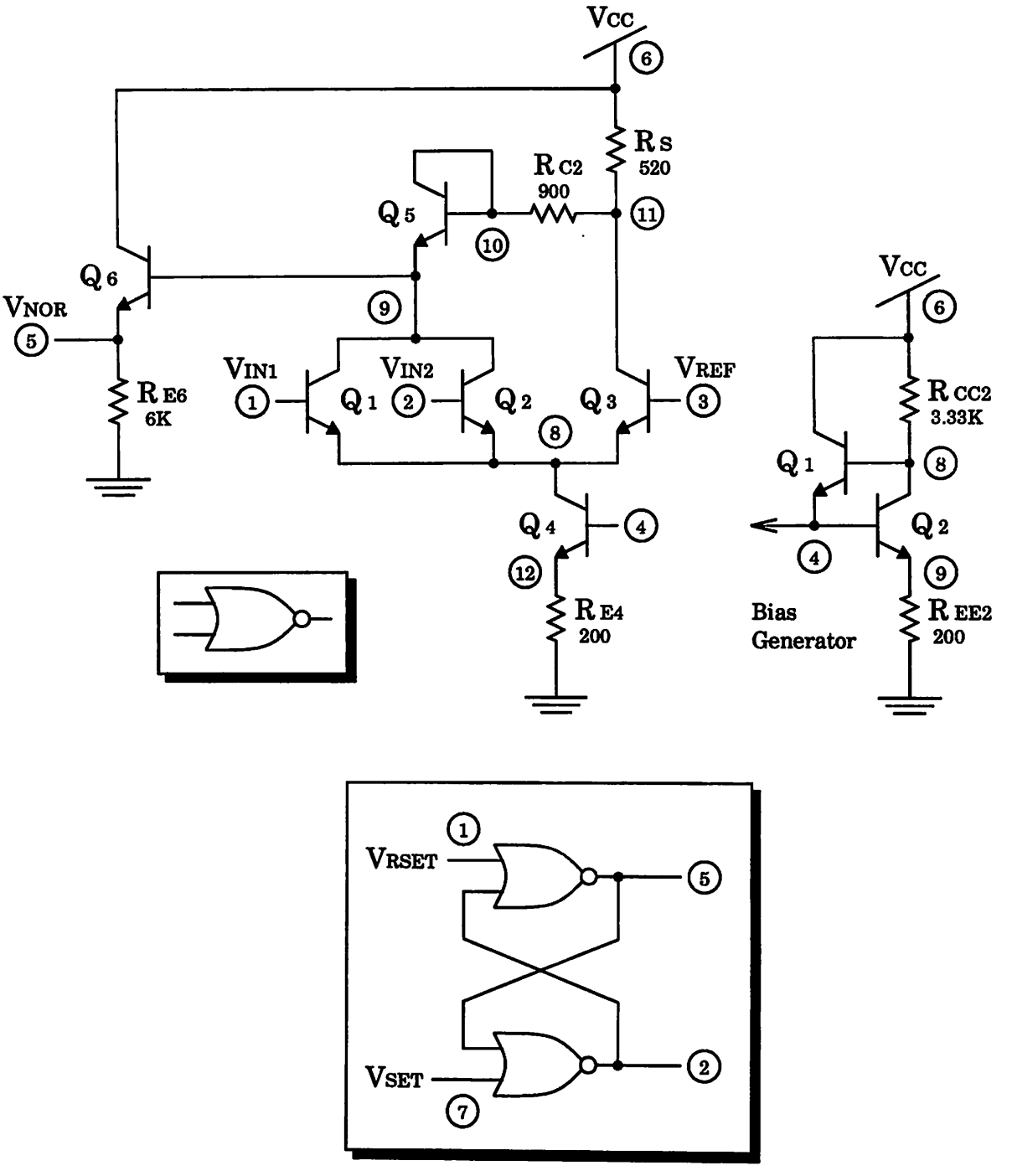


Figure C.8: LATCH schematic

PPEF Benchmarks

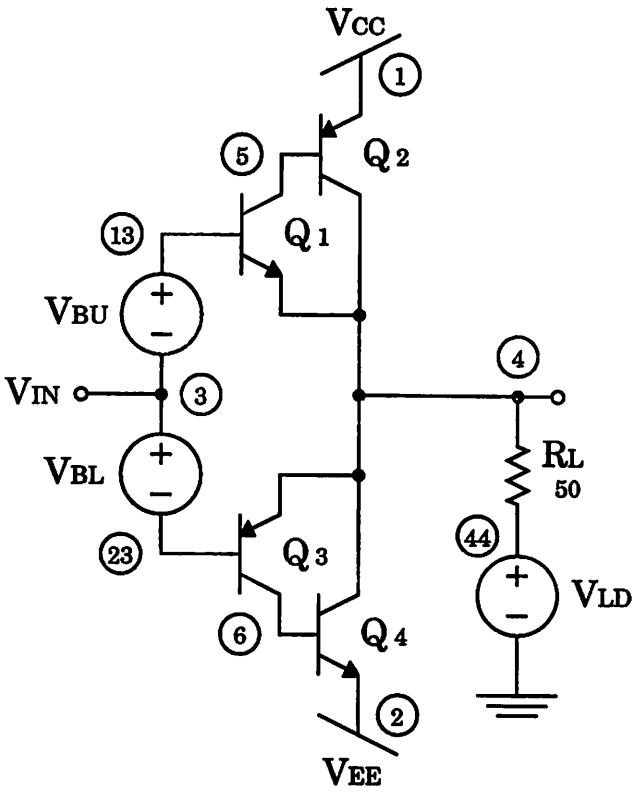


Figure C.9: PPEF.1D and PPEF.2D schematic

```
PUSH-PULL EMITTER FOLLOWER - ONE-DIMENSIONAL MODELS

VCC 1 0  5.0V
VEE 2 0 -5.0V

VIN 3 0  0.0V (SIN 0.0V 0.1V 1KHZ) AC 1
VBU 13 3 0.7V
VBL 3 23 0.7V

RL  4 44  50
VLD 44 0 0V

Q1  5  13 4 M_NPN1D AREA=40
Q2  4  5  1 M_PNP1D AREA=200

Q3  6  23 4 M_PNP1D AREA=100
Q4  4  6  2 M_NPN1D AREA=80

.INCLUDE BICMOS.LIB
```

```
.TRAN 0.01MS 1.00001MS 0US 0.01MS
.PLOT TRAN V(4)

.OPTIONS ACCT BYPASS=1 TEMP=26.85OC RELTOL=1E-5
.END


PUSH-PULL EMITTER FOLLOWER - TWO-DIMENSIONAL MODELS

VCC 1 0  5.0V
VEE 2 0 -5.0V

VIN 3 0  0.0V (SIN 0.0V 0.1V 1KHZ) AC 1
VBU 13 3 0.7V
VBL 3 23 0.7V

RL  4 44  50
VLD 44 0 0V

Q1  5  13 4 M_NPNS AREA=40
Q2  4  5  1 M_PNPS AREA=200

Q3  6  23 4 M_PNPS AREA=100
Q4  4  6  2 M_NPNS AREA=80

.INCLUDE BICMOS.LIB

.TRAN 0.01MS 1.00001MS 0US 0.01MS
.PLOT TRAN V(4)

.OPTIONS ACCT BYPASS=1 TEMP=26.85OC RELTOL=1E-5
.END
```

RINGOSC Benchmarks

```
CMOS RING OSCILLATOR - 1UM DEVICES

VDD 1 0 5.0V
VSS 2 0 0.0V

X1 1 2 3 4 INV
X2 1 2 4 5 INV
X3 1 2 5 6 INV
X4 1 2 6 7 INV
X5 1 2 7 8 INV
X6 1 2 8 9 INV
X7 1 2 9 3 INV
```

```
.IC V(3)=0.0V V(4)=2.5V V(5)=5.0V
+   V(6)=0.0V V(7)=5.0V V(8)=0.0V V(9)=5.0V

.SUBCKT INV 1   2   3   4
*           VDD VSS VIN VOUT
M1  14 13 15 16 M_PMOS_1 W=6.0U
M2  24 23 25 26 M_NMOS_1 W=3.0U

VGP 3 13 0.0V
VDP 4 14 0.0V
VSP 1 15 0.0V
VBP 1 16 0.0V

VGN 3 23 0.0V
VDN 4 24 0.0V
VSN 2 25 0.0V
VBN 2 26 0.0V
.ENDS INV

.INCLUDE BICMOS.LIB

.TRAN 0.1NS 1NS
.PRINT TRAN V(3) V(4) V(5)

.OPTIONS ACCT BYPASS=1 METHOD=GEAR
.END


CMOS RING OSCILLATOR - 2UM DEVICES

VDD 1 0 5.0V
VSS 2 0 0.0V

X1 1 2 3 4 INV
X2 1 2 4 5 INV
X3 1 2 5 6 INV
X4 1 2 6 7 INV
X5 1 2 7 8 INV
X6 1 2 8 9 INV
X7 1 2 9 3 INV

.IC V(3)=0.0V V(4)=2.5V V(5)=5.0V V(6)=0.0V
+ V(7)=5.0V V(8)=0.0V V(9)=5.0V

.SUBCKT INV 1   2   3   4
*           VDD VSS VIN VOUT
M1  14 13 15 16 M_PMOS W=6.0U
M2  24 23 25 26 M_NMOS W=3.0U

VGP 3 13 0.0V
VDP 4 14 0.0V
```

```
VSP 1 15 0.0V
VBP 1 16 0.0V

VGN 3 23 0.0V
VDN 4 24 0.0V
VSN 2 25 0.0V
VBN 2 26 0.0V
.ENDS INV

.MODEL M_NMOS NUMOS
+ X.MESH L=0.0 N=1
+ X.MESH L=0.6 N=4
+ X.MESH L=0.7 N=5
+ X.MESH L=1.0 N=7
+ X.MESH L=1.2 N=11
+ X.MESH L=3.2 N=21
+ X.MESH L=3.4 N=25
+ X.MESH L=3.7 N=27
+ X.MESH L=3.8 N=28
+ X.MESH L=4.4 N=31
+
+ Y.MESH L=-.05 N=1
+ Y.MESH L=0.0  N=5
+ Y.MESH L=.05  N=9
+ Y.MESH L=0.3  N=14
+ Y.MESH L=2.0  N=19
+
+ REGION NUM=1 MATERIAL=1 Y.L=0.0
+ MATERIAL NUM=1 SILICON
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+
+ REGION NUM=2 MATERIAL=2 Y.H=0.0 X.L=0.7 X.H=3.7
+ MATERIAL NUM=2 OXIDE
+
+ ELEC NUM=1 X.L=3.8 X.H=4.4 Y.L=0.0 Y.H=0.0
+ ELEC NUM=2 X.L=0.7 X.H=3.7 IY.L=1  IY.H=1
+ ELEC NUM=3 X.L=0.0 X.H=0.6 Y.L=0.0 Y.H=0.0
+ ELEC NUM=4 X.L=0.0 X.H=4.4 Y.L=2.0 Y.H=2.0
+
+ DOPING UNIF P.TYPE CONC=2.5E16 X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=2.0
+ DOPING UNIF P.TYPE CONC=1E16   X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=0.05
+ DOPING UNIF N.TYPE CONC=1E20   X.L=0.0 X.H=1.1  Y.L=0.0 Y.H=0.2
+ DOPING UNIF N.TYPE CONC=1E20   X.L=3.3 X.H=4.4  Y.L=0.0 Y.H=0.2
+
+ MODELS CONCMOB FIELDMOB BGN SRH CONCTAU
+ METHOD AC=DIRECT ONEC
+ OUTPUT ÂLL.DEBUG

.MODEL M_PMOS NUMOS
+ X.MESH L=0.0 N=1
```

```
+ X.MESH L=0.6 N=4
+ X.MESH L=0.7 N=5
+ X.MESH L=1.0 N=7
+ X.MESH L=1.2 N=11
+ X.MESH L=3.2 N=21
+ X.MESH L=3.4 N=25
+ X.MESH L=3.7 N=27
+ X.MESH L=3.8 N=28
+ X.MESH L=4.4 N=31
+
+ Y.MESH L=-.05 N=1
+ Y.MESH L=0.0  N=5
+ Y.MESH L=.05  N=9
+ Y.MESH L=0.3  N=14
+ Y.MESH L=2.0  N=19
+
+ REGION NUM=1 MATERIAL=1 Y.L=0.0
+ MATERIAL NUM=1 SILICON
+ MOBILITY MATERIAL=1 CONCMOD=SG FIELDMOD=SG
+
+ REGION NUM=2 MATERIAL=2 Y.H=0.0 X.L=0.7 X.H=3.7
+ MATERIAL NUM=2 OXIDE
+
+ ELEC NUM=1 X.L=3.8 X.H=4.4 Y.L=0.0 Y.H=0.0
+ ELEC NUM=2 X.L=0.7 X.H=3.7 IY.L=1  IY.H=1
+ ELEC NUM=3 X.L=0.0 X.H=0.6 Y.L=0.0 Y.H=0.0
+ ELEC NUM=4 X.L=0.0 X.H=4.4 Y.L=2.0 Y.H=2.0
+
+ DOPING UNIF N.TYPE CONC=1E16   X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=2.0
+ DOPING UNIF P.TYPE CONC=3E16   X.L=0.0 X.H=4.4  Y.L=0.0 Y.H=0.05
+ DOPING UNIF P.TYPE CONC=1E20   X.L=0.0 X.H=1.1  Y.L=0.0 Y.H=0.2
+ DOPING UNIF P.TYPE CONC=1E20   X.L=3.3 X.H=4.4  Y.L=0.0 Y.H=0.2
+
+ MODELS CONCMOB FIELDMOB BGN SRH CONCTAU
+ METHOD AC=DIRECT ONEC
+ OUTPUT ÂLL.DEBUG

.TRAN 0.1NS 5.0NS
.PRINT V(4)
.OPTIONS ACCT BYPASS=1 METHOD=GEAR
.END
```

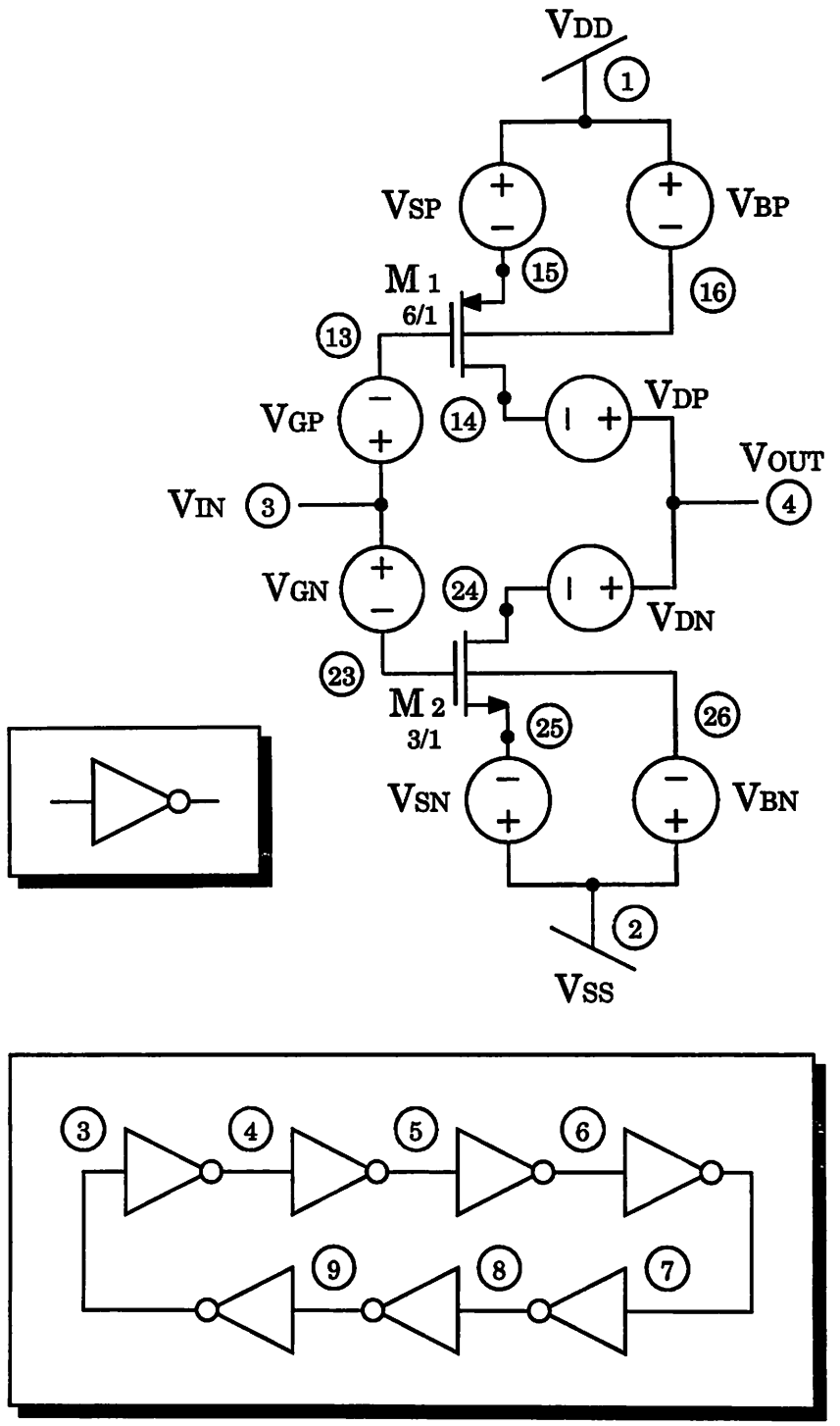


Figure C.10: RINGOSC.1U and RINGOSC.2U schematic

Appendix D

Model Libraries

This appendix contains the model descriptions used in the parallel benchmarks of Chapter 5, and in the examples of Chapter 6. The input listings of Appendix C assume that the remaining contents of this appendix have been placed in a file called 'BICMOS.LIB'.

```
**
* BICMOS.LIB: Library of models used in the 1.0 um CBiCMOS process
*     Contains CIDER input descriptions as well as matching
*     SPICE models for some of the CIDER models.
**

**
* One-dimensional models for a
* polysilicon emitter complementary bipolar process.
* The default device size is 1um by 1um (LxW)
**

.model M_NPN1D nbjt level=1
+ title One-Dimensional Numerical Bipolar
+ options base.depth=0.15 base.area=0.1 base.length=0.5 defa=1p
+ x.mesh loc=-0.2 n=1
+ x.mesh loc=0.0  n=51
+ x.mesh wid=0.15 h.e=0.0001 h.m=.004 r=1.2
+ x.mesh wid=1.15 h.s=0.0001 h.m=.004 r=1.2
+ domain   num=1 material=1 x.l=0.0
+ domain   num=2 material=2 x.h=0.0
+ material num=1 silicon
+ mobility mat=1 concmod=ct fieldmod=ct
+ material num=2 polysilicon
+ mobility mat=2 concmod=ct fieldmod=ct
+ doping gauss n.type conc=3e20  x.l=-0.2 x.h=0.0 char.len=0.047
+ doping gauss p.type conc=5e18  x.l=-0.2 x.h=0.0 char.len=0.100
```

```
+ doping unif  n.type conc=1e16  x.l=0.0 x.h=1.3
+ doping gauss n.type conc=5e19  x.l=1.3 x.h=1.3 char.len=0.100
+ models bgn srh auger conctau concmob fieldmob
+ method devtol=1e-12 ac=direct itlim=15

.model M_PNP1D nbjt level=1
+ title One-Dimensional Numerical Bipolar
+ options base.depth=0.2 base.area=0.1 base.length=0.5 defa=1p
+ x.mesh loc=-0.2 n=1
+ x.mesh loc=0.0  n=51
+ x.mesh wid=0.20 h.e=0.0001 h.m=.004 r=1.2
+ x.mesh wid=1.10 h.s=0.0001 h.m=.004 r=1.2
+ domain   num=1 material=1 x.l=0.0
+ domain   num=2 material=2 x.h=0.0
+ material num=1 silicon
+ mobility mat=1 concmod=ct fieldmod=ct
+ material num=2 polysilicon
+ mobility mat=2 concmod=ct fieldmod=ct
+ doping gauss p.type conc=3e20  x.l=-0.2 x.h=0.0 char.len=0.047
+ doping gauss n.type conc=5e17  x.l=-0.2 x.h=0.0 char.len=0.200
+ doping unif  p.type conc=1e16  x.l=0.0 x.h=1.3
+ doping gauss p.type conc=5e19  x.l=1.3 x.h=1.3 char.len=0.100
+ models bgn srh auger conctau concmob fieldmob
+ method devtol=1e-12 ac=direct itlim=15

**
* Two-dimensional models for a
* polysilicon emitter complementary bipolar process.
* The default device size is 1um by 1um (LxW)
**

.MODEL M_NPNS nbjt level=2
+ title TWO-DIMENSIONAL NUMERICAL POLYSILICON EMITTER BIPOLAR TRANSISTOR
+ * Since half the device is simulated, double the unit width to get
+ * 1.0 um emitter.  Use a small mesh for this model.
+ options defw=2.0u
+ output stat
+
+ x.mesh w=2.0 h.e=0.02 h.m=0.5 r=2.0
+ x.mesh w=0.5 h.s=0.02 h.m=0.2 r=2.0
+
+ y.mesh l=-0.2 n=1
+ y.mesh l= 0.0 n=5
+ y.mesh w=0.10 h.e=0.004 h.m=0.05  r=2.5
+ y.mesh w=0.15 h.s=0.004 h.m=0.02  r=2.5
+ y.mesh w=1.05 h.s=0.02  h.m=0.1   r=2.5
+
+ domain num=1 material=1 x.l=2.0 y.h=0.0
+ domain num=2 material=2 x.h=2.0 y.h=0.0
+ domain num=3 material=3 y.l=0.0
```

```
+ material num=1 polysilicon
+ material num=2 oxide
+ material num=3 silicon
+
+ elec num=1 x.l=0.0  x.h=0.0  y.l=1.1  y.h=1.3
+ elec num=2 x.l=0.0  x.h=0.5  y.l=0.0  y.h=0.0
+ elec num=3 x.l=2.0  x.h=3.0  y.l=-0.2 y.h=-0.2
+
+ doping gauss n.type conc=3e20 x.l=2.0 x.h=3.0 y.l=-0.2 y.h=0.0
+ + char.l=0.047 lat.rotate
+ doping gauss p.type conc=5e18 x.l=0.0 x.h=5.0 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate
+ doping gauss p.type conc=1e20 x.l=0.0 x.h=0.5 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate ratio=0.7
+ doping unif  n.type conc=1e16 x.l=0.0 x.h=5.0 y.l=0.0 y.h=1.3
+ doping gauss n.type conc=5e19 x.l=0.0 x.h=5.0 y.l=1.3 y.h=1.3
+ + char.l=0.100 lat.rotate
+
+ method ac=direct itlim=10
+ models bgn srh auger conctau concmob fieldmob

.MODEL M_NPN nbjt level=2
+ title TWO-DIMENSIONAL NUMERICAL POLYSILICON EMITTER BIPOLAR TRANSISTOR
+ * Since half the device is simulated, double the unit width to get
+ * 1.0 um emitter length. Uses a finer mesh in the X direction.
+ options defw=2.0u
+ output stat
+
+ x.mesh w=0.5  h.e=0.075 h.m=0.2 r=2.0
+ x.mesh w=0.75 h.s=0.075 h.m=0.2 r=2.0
+ x.mesh w=0.75 h.e=0.05 h.m=0.2 r=1.5
+ x.mesh w=0.5  h.s=0.05 h.m=0.1 r=1.5
+
+ y.mesh l=-0.2 n=1
+ y.mesh l= 0.0 n=5
+ y.mesh w=0.10 h.e=0.003 h.m=0.01  r=1.5
+ y.mesh w=0.15 h.s=0.003 h.m=0.02  r=1.5
+ y.mesh w=0.35 h.s=0.02  h.m=0.2   r=1.5
+ y.mesh w=0.40 h.e=0.05  h.m=0.2   r=1.5
+ y.mesh w=0.30 h.s=0.05  h.m=0.1   r=1.5
+
+ domain num=1 material=1 x.l=2.0 y.h=0.0
+ domain num=2 material=2 x.h=2.0 y.h=0.0
+ domain num=3 material=3 y.l=0.0
+ material num=1 polysilicon
+ material num=2 oxide
+ material num=3 silicon
+
+ elec num=1 x.l=0.0  x.h=0.0  y.l=1.1  y.h=1.3
+ elec num=2 x.l=0.0  x.h=0.5  y.l=0.0  y.h=0.0
```

```
+ elec num=3 x.l=2.0  x.h=3.0  y.l=-0.2 y.h=-0.2
+
+ doping gauss n.type conc=3e20 x.l=2.0 x.h=3.0 y.l=-0.2 y.h=0.0
+ + char.l=0.047 lat.rotate
+ doping gauss p.type conc=5e18 x.l=0.0 x.h=5.0 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate
+ doping gauss p.type conc=1e20 x.l=0.0 x.h=0.5 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate ratio=0.7
+ doping unif  n.type conc=1e16 x.l=0.0 x.h=5.0 y.l=0.0 y.h=1.3
+ doping gauss n.type conc=5e19 x.l=0.0 x.h=5.0 y.l=1.3 y.h=1.3
+ + char.l=0.100 lat.rotate
+
+ method ac=direct itlim=10
+ models bgn srh auger conctau concmob fieldmob

.MODEL M_PNPS nbjt level=2
+ title TWO-DIMENSIONAL NUMERICAL POLYSILICON EMITTER BIPOLAR TRANSISTOR
+ * Since half the device is simulated, double the unit width to get
+ * 1.0 um emitter length.  Use a small mesh for this model.
+ options defw=2.0u
+ output stat
+
+ x.mesh w=2.0 h.e=0.02 h.m=0.5 r=2.0
+ x.mesh w=0.5 h.s=0.02 h.m=0.2 r=2.0
+
+ y.mesh l=-0.2 n=1
+ y.mesh l= 0.0 n=5
+ y.mesh w=0.12 h.e=0.004 h.m=0.05  r=2.5
+ y.mesh w=0.28 h.s=0.004 h.m=0.02  r=2.5
+ y.mesh w=1.05 h.s=0.02  h.m=0.1   r=2.5
+
+ domain num=1 material=1 x.l=2.0 y.h=0.0
+ domain num=2 material=2 x.h=2.0 y.h=0.0
+ domain num=3 material=3 y.l=0.0
+ material num=1 polysilicon
+ material num=2 oxide
+ material num=3 silicon
+
+ elec num=1 x.l=0.0  x.h=0.0  y.l=1.1  y.h=1.3
+ elec num=2 x.l=0.0  x.h=0.5  y.l=0.0  y.h=0.0
+ elec num=3 x.l=2.0  x.h=3.0  y.l=-0.2 y.h=-0.2
+
+ doping gauss p.type conc=3e20 x.l=2.0 x.h=3.0 y.l=-0.2 y.h=0.0
+ + char.l=0.047 lat.rotate
+ doping gauss n.type conc=5e17 x.l=0.0 x.h=5.0 y.l=-0.2 y.h=0.0
+ + char.l=0.200 lat.rotate
+ doping gauss n.type conc=1e20 x.l=0.0 x.h=0.5 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate ratio=0.7
+ doping unif  p.type conc=1e16 x.l=0.0 x.h=5.0 y.l=0.0 y.h=1.3
+ doping gauss p.type conc=5e19 x.l=0.0 x.h=5.0 y.l=1.3 y.h=1.3
```

```
+ + char.l=0.100 lat.rotate
+
+ method ac=direct itlim=10
+ models bgn srh auger conctau concmob fieldmob

.MODEL M_PNP nbjt level=2
+ title TWO-DIMENSIONAL NUMERICAL POLYSILICON EMITTER BIPOLAR TRANSISTOR
+ * Since half the device is simulated, double the unit width to get
+ * 1.0 um emitter length. Uses a finer mesh in the X direction.
+ options defw=2.0u
+ output stat
+
+ x.mesh w=0.5  h.e=0.075 h.m=0.2 r=2.0
+ x.mesh w=0.75 h.s=0.075 h.m=0.2 r=2.0
+ x.mesh w=0.75 h.e=0.05 h.m=0.2 r=1.5
+ x.mesh w=0.5  h.s=0.05 h.m=0.1 r=1.5
+
+ y.mesh l=-0.2 n=1
+ y.mesh l= 0.0 n=5
+ y.mesh w=0.12 h.e=0.003 h.m=0.01  r=1.5
+ y.mesh w=0.28 h.s=0.003 h.m=0.02  r=1.5
+ y.mesh w=0.20 h.s=0.02  h.m=0.2   r=1.5
+ y.mesh w=0.40 h.e=0.05  h.m=0.2   r=1.5
+ y.mesh w=0.30 h.s=0.05  h.m=0.1   r=1.5
+
+ domain num=1 material=1 x.l=2.0 y.h=0.0
+ domain num=2 material=2 x.h=2.0 y.h=0.0
+ domain num=3 material=3 y.l=0.0
+ material num=1 polysilicon
+ material num=2 oxide
+ material num=3 silicon
+
+ elec num=1 x.l=0.0  x.h=0.0  y.l=1.1  y.h=1.3
+ elec num=2 x.l=0.0  x.h=0.5  y.l=0.0  y.h=0.0
+ elec num=3 x.l=2.0  x.h=3.0  y.l=-0.2 y.h=-0.2
+
+ doping gauss p.type conc=3e20 x.l=2.0 x.h=3.0 y.l=-0.2 y.h=0.0
+ + char.l=0.047 lat.rotate
+ doping gauss n.type conc=5e17 x.l=0.0 x.h=5.0 y.l=-0.2 y.h=0.0
+ + char.l=0.200 lat.rotate
+ doping gauss n.type conc=1e20 x.l=0.0 x.h=0.5 y.l=-0.2 y.h=0.0
+ + char.l=0.100 lat.rotate ratio=0.7
+ doping unif  p.type conc=1e16 x.l=0.0 x.h=5.0 y.l=0.0 y.h=1.3
+ doping gauss p.type conc=5e19 x.l=0.0 x.h=5.0 y.l=1.3 y.h=1.3
+ + char.l=0.100 lat.rotate
+
+ method ac=direct itlim=10
+ models bgn srh auger conctau concmob fieldmob

**
```

```
* Two-dimensional models for a
* complementary MOS process.
* Device models for 1um, 2um, 3um, 4um, 5um, 10um and 50um are provided.
**

.MODEL M_NMOS_1 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.4 h.s=0.005 h.m=0.1 r=2.0
+ x.mesh w=0.4 h.e=0.005 h.m=0.1 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=2 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=2.5  x.h=3.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=2  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=3.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=3.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=3.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=2 x.h=3.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=2.05 x.h=3.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_2 numos
+ output stat
```

```
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.005 h.m=0.2 r=2.0
+ x.mesh w=0.9 h.e=0.005 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=3 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=3.5  x.h=4.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=3  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=4.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=4.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=4.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=3 x.h=4.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=3.05 x.h=4.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_3 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=1.4 h.s=0.005 h.m=0.3 r=2.0
+ x.mesh w=1.4 h.e=0.005 h.m=0.3 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
```

```
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=4 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=4.5  x.h=5.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=4  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=5.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=5.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=5.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=4 x.h=5.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=4.05 x.h=5.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_4 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=1.9 h.s=0.005 h.m=0.4 r=2.0
+ x.mesh w=1.9 h.e=0.005 h.m=0.4 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
```

```
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=5 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=5.5  x.h=6.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=5  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=6.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=6.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=6.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=5 x.h=6.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=5.05 x.h=6.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_5 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=2.4 h.s=0.005 h.m=0.5 r=2.0
+ x.mesh w=2.4 h.e=0.005 h.m=0.5 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=6 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=6.5  x.h=7.1  y.l=0.0 y.h=0.0
```

```
+ elec num=2 x.l=1 x.h=6  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=7.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=7.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=7.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=6 x.h=7.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=6.05 x.h=7.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_10 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=4.9 h.s=0.005 h.m=1 r=2.0
+ x.mesh w=4.9 h.e=0.005 h.m=1 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=11 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=11.5  x.h=12.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=11  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=12.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=12.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=12.1 y.l=0.0 y.h=2.1
```

```
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=11 x.h=12.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=11.05 x.h=12.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_NMOS_50 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=24.9 h.s=0.005 h.m=5 r=2.0
+ x.mesh w=24.9 h.e=0.005 h.m=5 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=51 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=51.5  x.h=52.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=51  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=52.1 y.l=2.0 y.h=2.0
+
+ doping gauss p.type conc=1.0e17 x.l=-0.1 x.h=52.1 y.l=0.0
+ + char.l=0.30
+ doping unif p.type conc=5.0e15 x.l=-0.1 x.h=52.1 y.l=0.0 y.h=2.1
+ doping gauss n.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss n.type conc=4e17  x.l=51 x.h=52.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss n.type conc=1e20  x.l=51.05 x.h=52.1 y.l=0.0 y.h=0.08
```

```
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=4.10
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_1 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.4 h.s=0.005 h.m=0.1 r=2.0
+ x.mesh w=0.4 h.e=0.005 h.m=0.1 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=2 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=2.5  x.h=3.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=2  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=3.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=3.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=3.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=2 x.h=3.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=2.05 x.h=3.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_2 numos
```

```
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.005 h.m=0.2 r=2.0
+ x.mesh w=0.9 h.e=0.005 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=3 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=3.5  x.h=4.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=3  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=4.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=4.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=4.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=3 x.h=4.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=3.05 x.h=4.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_3 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=1.4 h.s=0.005 h.m=0.3 r=2.0
+ x.mesh w=1.4 h.e=0.005 h.m=0.3 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
```

```
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=4 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=4.5  x.h=5.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=4  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=5.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=5.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=5.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=4 x.h=5.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=4.05 x.h=5.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_4 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=1.9 h.s=0.005 h.m=0.4 r=2.0
+ x.mesh w=1.9 h.e=0.005 h.m=0.4 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
```

```
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=5 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=5.5  x.h=6.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=5  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=6.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=6.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=6.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=5 x.h=6.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=5.05 x.h=6.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_5 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=2.4 h.s=0.005 h.m=0.5 r=2.0
+ x.mesh w=2.4 h.e=0.005 h.m=0.5 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=6 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
```

```
+ elec num=1 x.l=6.5  x.h=7.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=6  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=7.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=7.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=7.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=6 x.h=7.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=6.05 x.h=7.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_10 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=4.9 h.s=0.005 h.m=1 r=2.0
+ x.mesh w=4.9 h.e=0.005 h.m=1 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=11 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=11.5  x.h=12.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=11  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=12.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=12.1 y.l=0.0
+ + char.l=0.30
```

```
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=12.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=11 x.h=12.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=11.05 x.h=12.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

.MODEL M_PMOS_50 numos
+ output stat
+
+ x.mesh w=0.9 h.e=0.020 h.m=0.2 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=24.9 h.s=0.005 h.m=5 r=2.0
+ x.mesh w=24.9 h.e=0.005 h.m=5 r=2.0
+ x.mesh w=0.2 h.e=0.005 h.m=0.02 r=2.0
+ x.mesh w=0.9 h.s=0.020 h.m=0.2 r=2.0
+
+ y.mesh l=-.0200 n=1
+ y.mesh l=0.0 n=6
+ y.mesh w=0.15 h.s=0.0001 h.max=.02 r=2.0
+ y.mesh w=0.45 h.s=0.02 h.max=0.2 r=2.0
+ y.mesh w=1.40 h.s=0.20 h.max=0.4 r=2.0
+
+ region num=1 material=1 y.h=0.0
+ region num=2 material=2 y.l=0.0
+ interface dom=2 nei=1 x.l=1 x.h=51 layer.width=0.0
+ material num=1 oxide
+ material num=2 silicon
+
+ elec num=1 x.l=51.5  x.h=52.1  y.l=0.0 y.h=0.0
+ elec num=2 x.l=1 x.h=51  iy.l=1  iy.h=1
+ elec num=3 x.l=-0.1  x.h=0.5  y.l=0.0 y.h=0.0
+ elec num=4 x.l=-0.1  x.h=52.1 y.l=2.0 y.h=2.0
+
+ doping gauss n.type conc=1.0e17 x.l=-0.1 x.h=52.1 y.l=0.0
+ + char.l=0.30
+ doping unif n.type conc=5.0e15 x.l=-0.1 x.h=52.1 y.l=0.0 y.h=2.1
+ doping gauss p.type conc=4e17  x.l=-0.1 x.h=1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
+ doping gauss p.type conc=1e20  x.l=-0.1 x.h=0.95 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+ doping gauss p.type conc=4e17  x.l=51 x.h=52.1 y.l=0.0 y.h=0.0
+ + char.l=0.16 lat.rotate ratio=0.65
```

```
+ doping gauss p.type conc=1e20  x.l=51.05 x.h=52.1 y.l=0.0 y.h=0.08
+ + char.l=0.03 lat.rotate ratio=0.65
+
+ contact num=2 workf=5.29
+ models concmob surfmob transmob fieldmob srh auger conctau bgn
+ method ac=direct itlim=10 onec

**
* BSIM1 NMOS and PMOS 1.0 μm models.
* Gummel-Poon bipolar models.
**
.model M_NSIM_1 nmos level=4
+vfb= -1.1908
+phi= .8399
+k1= 1.5329
+k2= 193.7322m
+eta= 2m
+muz= 746.0
+u0= 90.0m
+x2mz= 10.1429
+x2e= -2.5m
+x3e= 0.2m
+x2u0= -10.0m
+mus= 975.0
+u1= .20
+x2ms= 0.0
+x2u1= 0.0
+x3ms= 10
+x3u1= 5.0m
+tox=2.00000e-02
+cgdo=2.0e-10
+cgso=2.0e-10
+cgbo=0.0
+temp= 27
+vdd= 7.0
+xpart
+n0= 1.5686
+nb= 94.6392m
+nd=0.00000e+00
+rsh=30.0          cj=7.000e-004     cjsw=4.20e-010
+js=1.00e-008      pb=0.700e000
+pbsw=0.8000e000   mj=0.5            mjsw=0.33
+wdf=0             dell=0.20u

.model M_PSIM_1 pmos level=4
+vfb= -1.3674
+phi= .8414
+k1= 1.5686
+k2= 203m
+eta= 2m
```

```
+muz= 340.0
+u0= 35.0m
+x2mz= 6.0
+x2e= 0.0
+x3e= -0.2m
+x2u0= -15.0m
+mus= 440.0
+u1=  .38
+x2ms= 0.0
+x2u1= 0.0
+x3ms= -20
+x3u1= -10.0m
+tox=2.00000e-02
+cgdo=2.0e-10
+cgso=2.0e-10
+cgbo=0.0
+temp= 27
+vdd= 5.0
+xpart
+n0= 1.5686
+nb= 94.6392m
+nd=0.00000e+00
+rsh=80.0          cj=7.000e-004     cjsw=4.20e-010
+js=1.00e-008      pb=0.700e000
+pbsw=0.8000e000   mj=0.5            mjsw=0.33
+wdf=0             dell=0.17u

.model M_GNPN npn
+ is=1.3e-16
+ nf=1.00 bf=262.5 ikf=25mA  vaf=20v
+ nr=1.00 br=97.5 ikr=0.5mA var=1.8v
+ rc=20.0
+ re=0.09
+ rb=15.0
+ ise=4.0e-16 ne=2.1
+ isc=7.2e-17 nc=2.0
+ tf=9.4ps itf=26uA xtf=0.5
+ tr=10ns
+ cje=89.44fF vje=0.95 mje=0.5
+ cjc=12.82fF vjc=0.73 mjc=0.49

.model M_GPNP pnp
+ is=5.8e-17
+ nf=1.001 bf=96.4 ikf=12mA  vaf=29v
+ nr=1.0   br=17.3 ikr=0.2mA var=2.0v
+ rc=50.0
+ re=0.17
+ rb=20.0
+ ise=6.8e-17 ne=2.0
+ isc=9.0e-17 nc=2.1
```

```
+ tf=27.4ps itf=26uA xtf=0.5
+ tr=10ns
+ cje=55.36fF vje=0.95 mje=0.58
+ cjc=11.80fF vjc=0.72 mjc=0.46
```

Appendix E

CIDER Source Code Listing

The source-code listings for the programs used in this dissertation can be obtained from the following address:

Software Distribution Office
Industrial Liaison Program
Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720

All programs (the serial and parallel versions of CIDER and the experimental simulated annealer) are contained in the one CIDER source distribution.

Bibliography

[APTE92] D. R. Apte and M. E. Law. Comparison of iterative methods for AC analysis in PISCES-IIB. *IEEE Transactions on Computer-Aided Design*, 11(5):671–673, May 1992.

[AROR82] N. D. Arora, J. R. Hauser, and D. J. Roulston. Electron and hole mobilities in silicon as a function of concentration and temperature. *IEEE Transactions on Electron Devices*, ED-29:292–295, February 1982.

[ASHB87] P. Ashburn, D. J. Roulston, and C. R. Selvakumar. Comparison of experimental and computed results on arsenic- and phosphorus-doped polysilicon emitter bipolar transistors. *IEEE Transactions on Electron Devices*, ED-34:1346–1353, June 1987.

[BANK81] R. E. Bank and D. J. Rose. Global approximate Newton methods. *Numerische Mathematik*, 37:279–295, 1981.

[BANK85] R. E. Bank, W. M. Coughran, Jr., W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith. Transient simulation of silicon devices and circuits. *IEEE Transactions on Computer-Aided Design*, CAD-4(4):436–451, October 1985.

[BELL92a] G. Bell. Ultracomputers: a teraflop before its time. *Communications of the ACM*, 35(8):27–47, August 1992.

[BELL92b] A. Bellaouar, S. H. K. Embabi, and M. I. Elmasry. Low-voltage scaled CMOS and BiCMOS digital circuits. *IEEE Transactions on Electron Devices*, 39:1005–1009, April 1992.

[BISC86] G. Bischoff and S. Greenberg. CAYENNE: a parallel implementation of the circuit simulator SPICE. In *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pages 182–185, November 1986.

[BOYL87] J. Boyle, R. Butler, T. Disz, Glickfeld B., R. Lusk, R. Overbeek, J. Patterson, and R. Stevens. *Portable Programs for Parallel Processors*. Holt, Rinehart, and Winston, New York, 1987.

[BRAY72] R. K. Brayton, F. G. Gustavson, and G. D. Hachtel. A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulae. *Proceedings of the IEEE*, 60(1):98–108, January 1972.

[BUTL92] R. Butler and E. Lusk. User's guide to the p4 programming system. Technical Report ANL-92/17, Argonne National Laboratory, October 1992.

[CAUG67] D. M. Caughey and R. E. Thomas. Carrier mobilities in silicon empirically related to doping and field. *Proceedings of the IEEE*, 55(12):1292–1293, December 1967.

[CHAN88] M.-C. Chang and I. N. Hajj. iPRIDE: a parallel integrated circuit simulator using direct method. In *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pages 304–307, November 1988.

[CHEN88] C.-C. Chen and Y.-H. Hu. Parallel LU factorization for circuit simulation on a MIMD computer. In *Proceedings, 1988 IEEE International Conference on Computer Design*, pages 129–132, October 1988.

[CHIN92] G. Chin and R. W. Dutton. A tool towards integration of IC process, device, and circuit simulation. *IEEE Journal of Solid-State Circuits*, 27(3):265–273, March 1992.

[COX91] P. F. Cox, R. G. Burch, D. E. Hocevar, P. Yang, and B. D. Epler. Direct circuit simulation algorithms for parallel processing. *IEEE Transactions on Computer-Aided Design*, 10(6):714–725, June 1991.

[DONG86] J. J. Dongarra. A survey of high performance computers. In *IEEE COMPCON*, pages 8–11, March 1986.

[DONG91] J. Dongarra and J. Demmel. LAPACK – a portable high-performance numerical library for linear algebra. *Supercomputer*, 8(6):33–38, November 1991.

[DONG93] Jack J. Dongarra. Performance of various computers using standard linear equation software. Technical Report CS-89-85, Oak Ridge National Laboratory, March 11 1993. Available from `netlib@ornl.gov`.

[DUFF86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[DUNI91] T. H. Dunigan. Performance of the Intel iPSC/860 and Ncube 6400 hypercubes. *Parallel Computing*, 17(10-11):1285–1302, December 1991.

[EARL52] J. M. Early. Effects of space-charge layer widening in junction transistors. *Proceedings, IRE*, 40:1401–1406, November 1952.

[ENGL82] W. L. Engl, R. Laur, and H. K. Dirks. MEDUSA - a simulator for modular circuits. *IEEE Transactions on Computer-Aided Design*, CAD-1(2):85–93, April 1982.

[FLYN66] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, December 1966.

[FROH69] D. Frohman-Bentchkowsky and A. S. Grove. Conductance of MOS transistors in saturation. *IEEE Transactions on Electron Devices*, ED-16(1):108–113, January 1969.

[GARE79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[GATE90] D. A. Gates. An inversion-layer mobility model for CODECS. Memorandum No. UCB/ERL M90/96, Electronics Research Laboratory, University of California, Berkeley, October 1990.

[GATE93] D. A. Gates, P. K. Ko, and D. O. Pederson. Mixed-level circuit and device simulation on a distributed-memory multicomputer. In *Proceedings of the IEEE 1993 Custom Integrated Circuits Conference*, May 1993.

[GEOR73] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal of Numerical Analysis*, 10:345–363, 1973.

[GETR76] I. Getreu. *Modeling the bipolar transistor.* Tektronix, Beaverton, OR, 1976.

[GRAA90] H. C. de Graaff and F. M. Klaasen. *Compact Transistor Modelling for Circuit Design.* Springer-Verlag, Wien, 1990.

[GRAY93] P. R. Gray and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits, Third Edition.* John Wiley & Sons, New York, 1993.

[GREE93] T. Green, R. Pennington, and D. Reynolds. *Distributed queuing system version 2.1 release notes*, March 22 1993. Available from `ftp.scri.fsu.edu`.

[GROU90] NCSA Software Tools Group. *NCSA HDF Vset version 2.0.* University of Illinois at Urbana-Champaign, November 1990. Available from `ftp.ncsa.uiuc.edu`.

[GUY 79] N. B. Guy Rabbat, A. L. Sangiovanni-Vincentelli, and H. Y. Hsieh. A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain. *IEEE Transactions on Circuits and Systems*, CAS-26(9):733–740, September 1979.

[HACH71] G. D. Hachtel, R. K. Brayton, and F. G. Gustavson. The sparse tableau approach to network analysis and design. *IEEE Transactions on Circuit Theory*, CT-18(1):101–113, January 1971.

[HARR86] D. S. Harrison, P. Moore, R. L. Spickelmier, and A. R. Newton. Data management and graphics editing in the Berkeley design environment. In *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pages 20–24, November 1986.

[HARR91] R. J. Harrison. Portable tools and applications for parallel computers. *International Journal of Quantum Chemistry*, 40:847–863, December 1991.

[HEAT91] M. T Heath, E. Ng, and B. W. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33(3):420–460, September 1991.

[HO75] C. W. Ho, A. E. Ruehli, and P. A. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, CAS-22(6):504–509, June 1975.

[HO83] C. Ho, J. D. Plummer, S. Hansen, and R. W. Dutton. VLSI process modeling – Suprem III. *IEEE Transactions on Electron Devices*, ED-30(11):1438–1453, November 1983.

[HU87] G. J. Hu, C. Chang, and Y.-T. Chia. Gate-voltage-dependent effective channel length and series resistance of LDD MOSFET's. *IEEE Transactions on Electron Devices*, ED-34(12):2469–2475, December 1987.

[HUAN93] J.H. Huang, Z. H. Liu, P. K. Ko, C. Hu, and M. C. Jeng. A robust physical and predictive model for deep-submicrometer MOS circuit simulation. In *Proceedings of the IEEE 1993 Custom Integrated Circuits Conference*, May 1993.

[IPS92a] Intel Corporation, Beaverton, OR. *iPSC/860 Network Queueing System Manual*, March 1992.

[IPS92b] Intel Corporation, Beaverton, OR. *iPSC/860 System User's Guide*, March 1992.

[IRAN91] A. A. Iranmanesh, V. Ilderem, M. Biswal, and B. Bastani. A 0.8 μm advanced sing-poly BiCMOS technology for high-density and high-performance applications. *IEEE Journal of Solid-State Circuits*, 26:422–426, March 1991.

[JACO87] G. K. Jacob. Direct methods in circuit simulation using multiprocessors. Memorandum No. UCB/ERL M87/67, Electronics Research Laboratory, University of California, Berkeley, October 1987.

[JENG90] M.-C. Jeng. Design and modeling of deep-submicrometer MOSFETs. Memorandum No. UCB/ERL M90/90, Electronics Research Laboratory, University of California, Berkeley, October 1990.

[JOHN89] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing; part I, graph partitioning. *Operations Research*, 37(6):865–892, November-December 1989.

[JOHN92] B. Johnson, T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli. SPICE3 version 3f user's manual. Technical report, Department of Electrical Engineering and computer Science, University of California, Berkeley, October 1992.

[KAPO89] A. K. Kapoor and D. J. Roulston, editors. *Polysilicon Emitter Bipolar Transistors*. IEEE Press, 1989.

[KELL90] T. M. Kellesoglou. NECTAR: A knowledge-based framework for analog circuit verification. Memorandum No. UCB/ERL M90/112, Electronics Research Laboratory, University of California, Berkeley, December 1990.

[KERN70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[KIRK83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[KO86] H. F.-H. Ko. *A special-purpose architecture and parallel algorithms on a multiprocessor system for the solution of large scale linear systems of equations*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1986.

[KO93] P. K. Ko. Private communication, 1993.

[KUND86] K. S. Kundert. Sparse matrix techniques and their application to circuit simulation. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pages 281–324. North-Holland, New York, 1986.

[LAUX85] S. E. Laux. Techniques for small-signal analysis of semiconductor devices. *IEEE Transactions on Computer-Aided Design*, CAD-4(4):472–481, October 1985.

[LEWI92] T. G. Lewis and H. El-Rewini. *Introduction to Parallel Computing*. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[LIN93] W. W. Lin and P. C. Chan. Fix to negative output conductance problem in BSIM2 model. *IEEE Transactions on Electron Devices*, 40(5):1024–1028, May 1993.

[LINI86] W. Liniger, F. Odeh, and A. Ruehli. Integration methods for the solution of circuit equations. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pages 235–279. North-Holland, New York, 1986.

[LUCA87a] R. Lucas and T. Blank. Parallel PISCES. In *Proceedings of the IEEE 1987 Custom Integrated Circuits Conference*, pages 119–123, May 1987.

[LUCA87b] R. F. Lucas, T. Blank, and J. J. Tiemann. A parallel solution method for large sparse systems of equations. *IEEE Transactions on Computer-Aided Design*, CAD-6(6):981–991, November 1987.

[MAYA88] K. Mayaram. CODECS: a mixed-level circuit and device simulator. Memorandum No. UCB/ERL M88/71, Electronics Research Laboratory, University of California, Berkeley, December 1988.

[MAYA92] K. Mayaram. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer-Aided Design*, 11(8):1003–1012, August 1992.

[MEIN90] B. Meinerzhagen, J. M. J. Krücken, K. H. Bach, F. M. Stecher, and W. L. Engl. A modular approach to parallel mixed level device/circuit simulation. In *Proceedings, 1990 VLSI Process/Device Modeling Workshop (VPAD)*, pages 170–172, 1990.

[MET90] Meta-Software, Inc., Campbell, CA. *HSPICE User's Manual*, h9001 edition, 1990.

[NAGE75] L. W. Nagel. SPICE2: A computer program to simulate semiconductor circuits. Memorandum No. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, May 1975.

[NAKA91] T. Nakadai and K. Hashimoto. Measuring the base resistance of bipolar transistors. In *Proceedings, 1991 IEEE Bipolar Circuits and Technology Meeting*, pages 200–203, September 1991.

[NEWT83] A. R. Newton and A. L. Sangiovanni-Vincentelli. Relaxation-based electrical simulation. *IEEE Transactions on Electron Devices*, ED-30(9):1184–1206, September 1983.

[OGUR80] S. Ogura, P. J. Tsang, W. W. Walker, D. L. Critchlow, and J. F. Shepard. Design and characteristics of the lightly-doped drain-source (LDD) insulated gate field-effect transistor. *IEEE Transactions on Electron Devices*, ED-27:1359–1367, August 1980.

[OUST88] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The Sprite operating system. *IEEE Computer*, 21:23–36, February 1988.

[PACH91] P. S. Pacheco, J. M. del Rosario, and T. Rashid. Parallel SPICE on distributed memory multiprocessors. *Supercomputer*, 8(6):119–126, November 1991.

[PATT90] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[PEDE91] D. O. Pederson and K. Mayaram. *Analog integrated circuits for communication: principles, simulation, and design*. Kluwer Academic Publishers, Boston, 1991.

[PINT85] M. R. Pinto, C. S. Rafferty, H. R. Yeager, and R. W. Dutton. PISCES-II user's guide and supplementary report. Technical report, Stanford Electronics Lab., Stanford University, 1985.

[PINT90] M. R. Pinto. *Comprehensive semiconductor device simulation for silicon ULSI*. PhD thesis, Stanford University, 1990.

[PIX89] Digital Equipment Corporation, Maynard, MA. *Pixie – Ultrix 4.2a Manual Page*, 1989.

[POTH90] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[QUAR89] T. L. Quarles. Analysis of performance and convergence issues for circuit simulation. Memorandum No. UCB/ERL M89/42, Electronics Research Laboratory, University of California, Berkeley, April 1989.

[ROOS50] W. van Roosbroeck. Theory of flow of electrons and holes in germanium and other semiconductors. *Bell System Technical Journal*, 29:560–607, 1950.

[ROYC91] J. S. Roychowdhury, A. R. Newton, and D. O. Pederson. An impulse-response based linear time-complexity algorithm for lossy interconnect simulation. In *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pages 62–65, November 1991.

[SADA87] P. Sadayappan and V. Visvanathan. Circuit simulation on a multiprocessor. In *Proceedings of the IEEE 1987 Custom Integrated Circuits Conference*, pages 124–128, May 1987.

[SALE89] R. A. Saleh, K. A. Gallivan, M.-C. Chang, I. N. Hajj, D. Smart, and T. N. Trick. Parallel circuit simulation on supercomputers. *Proceedings of the IEEE*, 77(12):1915–1931, December 1989.

[SCHA69] D. L. Scharfetter and H. K. Gummel. Large-signal analysis of a silicon Read diode oscillator. *IEEE Transactions on Electron Devices*, ED-16:64, January 1969.

[SCHR91] M. Schröter. Transient and small-signal high-frequency simulation of numerical device models embedded in an external circuit. *COMPEL*, 10(4):377–387, December 1991. NASECODE VII Transactions.

[SELB84] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer-Verlag, Wien, 1984.

[SIMP91] M. R. Simpson. PRIDE: An integrated design environment for semiconductor device simulation. *IEEE Transactions on Computer-Aided Design*, 10(9):1163–1174, September 1991.

[SING86] K. Singhal and J. Vlach. Formulation of circuit equations. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pages 45–70. North-Holland, New York, 1986.

[SOLL90] E. G. Solley, Jr. Temperature dependence of physical parameters for improved bipolar device simulation. Master's thesis, University of Florida, 1990.

[STRU85] R. D. Strum and J. R. Ward. *Electric Circuits and Networks, Second Edition*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[SUND90] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice & Experience*, 2(4):315–339, December 1990.

[SZE81] S. M. Sze. *Physics of Semiconductor Devices, Second Edition*. John Wiley & Sons, New York, 1981.

[TMA91] TMA PISCES-2B circuit analysis advanced application module. Technology Modeling Associates, Inc. product announcement, 1991.

[TROT90] J. A. Trotter and P. Agrawal. Circuit simulation algorithms on a distributed memory multiprocessor system. In *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pages 438–441, November 1990.

[VLAD82] A. Vladimirescu. *LSI circuit simulation on vector computers*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1982.

[WEBB91] D. M. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, and A. Sangiovanni-Vincentelli. A massively parallel algorithm for three-dimensional device simulation. *IEEE Transactions on Computer-Aided Design*, 10(9):1201–1209, September 1991.

[WONG91] A. S. Wong and A. R. Neureuther. The intertool profile interchange format: a technology CAD environment approach. *IEEE Transactions on Computer-Aided Design*, 10(9):1157–1162, September 1991.

[WU91] K.-C. Wu, G. R. Chin, and R. W. Dutton. A STRIDE towards practical 3-D device simulation – numerical and visualization considerations. *IEEE Transactions on Computer-Aided Design*, 10(9):1132–1140, September 1991.

[YAMA85] F. Yamamoto and S. Takahashi. Vectorized LU decomposition algorithms for large-scale circuit simulation. *IEEE Transactions on Computer-Aided Design*, CAD-4(3):232–239, July 1985.

[YANG90] G.-C. Yang. PARASPICE: a parallel circuit simulator for shared-memory multiprocessors. In *Proceedings, 27th ACM/IEEE Design Automation Conference*, pages 400–405, June 1990.

[YOUN90] D. Young. Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1990. EE199 Report.

[YU85] Z. Yu and R. W. Dutton. SEDAN III – a general purpose, one-dimensional semiconductor analysis program. Technical report, Integrated Circuits Lab., Stanford University, Stanford, CA, July 1985.

[YUAN88] C.-P. Yuan, R. Lucas, P. Chan, and R. Dutton. Parallel electronic circuit simulation on the iPSC system. In *Proceedings of the IEEE 1988 Custom Integrated Circuits Conference*, May 1988.

[ZARR89] M. Zarrabian. Evaluation of SPICE modeling of bipolar transistors with CODECS. Master's thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.