AN ATTACHED PROCESSOR FOR MOS-TRANSISTOR MODEL EVALUATION

by

Ronald Steven Gyurcsik

Memorandum No. UCB/ERL M86/82

15 October 1986

AN ATTACHED PROCESSOR FOR MOS-TRANSISTOR MODEL EVALUATION

by

Ronald Steven Gyurcsik

Memorandum No. UCB/ERL M86/82

15 October 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN ATTACHED PROCESSOR FOR MOS-TRANSISTOR MODEL EVALUATION

by

Ronald Steven Gyurcsik

Memorandum No. UCB/ERL M86/82

15 October 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

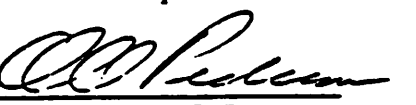
AN ATTACHED PROCESSOR FOR MOS-TRANSISTOR MODEL EVALUATION

Ronald Steven Gyurcsik

Ph.D.

Department of Electrical Engineering
and Computer Sciences

Sponsor: Hewlett-Packard

Signature 

D. O. Pederson
Committee Chairman

ABSTRACT

The design and implementation of an attached processor for MOS-transistor model evaluation is presented. The computational time used in evaluating the MOS-transistor model is a significant amount of the total circuit-simulation time of MOS integrated circuits. A reduction in the total simulation time is achieved by accelerating the MOS-model evaluation with the attached processor. The attached processor is suitable for use with electrical circuit-simulation programs, and a prototype has been constructed and interfaced to an IBM PC-XT computer running the BIASC circuit-simulation program.

The software interaction between the circuit-simulation program and the attached processor and the hardware interaction between the computer-system architecture and the attached processor have both been considered. The attached processor and the host computer work in parallel. The transistor model data is stored in memory local to the attached processor to reduce the communication overhead between the host and attached processor.

Different types of MOS-transistor models have been studied, and an empirical model based on piecewise-cubic polynomials has been developed for use with the attached processor. The model evaluation requires only floating-point addition, subtraction, and multiplication operations and is performed without conditional branching.

An architecture has been developed which exploits the properties of the empirical model. The architecture supports the concurrent evaluation of several transistors and can be expanded to reduce the effective evaluation time.

A prototype attached processor has been developed for the IBM PC-XT. The prototype is a board-level design and is comprised of 101 standard parts assembled on two circuit boards. An order of magnitude decrease in the evaluation time of the MOS-transistor equations, and a maximum of a 30% reduction in circuit-simulation time have been achieved.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Professor D. O. Pederson for his leadership and encouragement.

I gratefully acknowledge the Hewlett-Packard Corporation for their generous funding of the research presented in this dissertation. In particular, I wish to acknowledge Dr. W. McCalla of Hewlett-Packard for his continued interest and support.

The numerous discussions with Jeff Burns which assisted in the formulation of the empirical model is gratefully acknowledged. I wish to thank Kris Pister for wire-wrapping the prototype and writing support programs. The assistance I received from Ferenc Kovac, Joao Wentzcovich and Alex Para of the Elelctronics Shop is gratefully acknowledged. I am also very grateful to Alain Hanover of Viewlogic Systems, Inc. for providing schematic-capture and simulation software.

I wish to thank Karti Mayaram and Jeff Burns for their critique of numerous drafts of this dissertation, and I wish to thank Karti Mayaram and Theo Kelessoglou for their assistance in the preparation of figures for this dissertation. I would also like to thank Jeff Burns, Karti Mayaram, Mark Hofmann, Rick Spickelmeir, Tom Quarles, Theo Kelessoglou, Fabio Romeo, Giorgio Casinovi, Peter Moore, George Jacob, Dave Burnett, Kris Pister, Mike Klein, Ken Kundert, Tom Laideg, Dierdre Ryan, Chris Marino, Jacob White, Wayne Christopher and Res Saleh for being great people to work with. I wish to also thank Professors D. A. Hodges, A. Sangiovanni-Vincentelli and A. R. Newton for their assistance throughout my time at Berkeley.

I wish to thank my parents, Sarah and Steve Gyurcsik, for instilling in me the importance of education and for their continued assistance throughout my education.

Most of all, I wish to thank Peggy Sue Gyurcsik, my wife, for all the love and support she has given me and all the sacrifices she has made for me.

CHAPTER 1

Introduction and an Overview of Circuit Simulation

Electrical circuit simulation is an integral component of the integrated circuit design process, where it is used to verify both the function and performance of the integrated circuit. Electrical circuit simulation is computationally expensive, requiring the numerical solution of the nonlinear differential-algebraic equations(DAEs) modeling the integrated circuit. Increasing size and complexity of integrated circuits have resulted in a superlinear increase in the computation time required for their simulation.

The set of nonlinear DAEs can be solved using either direct[Nag75] or relaxation[New78] methods. Both methods use a numerical integration method to reduce the set of nonlinear DAEs to a set of nonlinear algebraic equations. The solution of the nonlinear algebraic equations requires the evaluation of the nonlinear transistor model equations. A large percentage of the computation time used in simulating integrated circuits is spent in the evaluation of the transistor model equations. A reduction in the time required for transistor model evaluation results in a decrease in computation time for both direct-based and relaxation-based simulation methods.

A MOS-Model Attached Processor(MMAP), a special-purpose attached processor that evaluates the DC-MOS transistor equations for use in electrical circuit simulation, is presented in this dissertation. The motivation for developing the attached processor is to accelerate the calculation of the MOS transistor equations in order to reduce the overall time for electrical simulation. Included in the dissertation are descriptions of the interface between a simulation program and the MMAP, the DC MOS transistor model representation, the MMAP architecture and a prototype MMAP design.

1.1. Dissertation Outline

The remainder of this chapter provides an overview of electrical circuit simulation and the MOS transistor. A qualitative description of the basic structure and electrical behavior of the MOS transistor are given. The companion model of the MOS transistor is then described. Finally, the MOS transistor-evaluation routine of a circuit-simulation program is described.

Chapter 2 describes the computer system interface to the MMAP and the circuit-simulation program interface to the MMAP. Chapter 3 provides an overview of MOS-transistor model representations and the relationship between the MMAP architecture and the transistor model. A new empirical transistor model representation developed for use in the MMAP is presented in Chapter 4. Chapter 5 provides an overview of the MMAP's architecture. This includes descriptions of the organization, the storage and accessing of the model data by the attached processor, and of enhancements made to the architecture to improve performance. A board-level prototype MMAP is described in Chapter 6. The prototype MMAP has been designed and built for use with the IBM PC-XT personal computer. The prototype MMAP's performance, and the performance of the BIASC[1] [Gyu85] circuit simulation program running on the IBM PC-XT with and without the prototype MMAP, are given.

1.2. Description of Circuit Simulation

In the electrical simulation of MOS integrated circuits, both direct and relaxation methods require the evaluation of the MOS transistor's device equations. To illustrate the requirement for transistor model evaluation in electrical circuit simulation, the time-domain transient simulation of a circuit containing nonlinear devices is described. Circuit-simulation programs first assemble the nonlinear DAEs representing the

[1]BIASC - A description of the BIASC electrical circuit-simulation program, including the source-code listing, is given in Appendix A.

integrated circuit. The circuit equations are of the form

$$f_i(x_1, x_2, \cdots x_N, \dot{x}_1, \dot{x}_2, \cdots \dot{x}_N, t) = 0, \quad i = 1, \ldots, N. \tag{1.1}$$

where the x_i are the independent variables, the $\dot{x}_i$ are the time derivatives of the independent variables, and t is the time. The independent variables are the node voltages and branch currents of the circuit.

A method for solving the nonlinear DAEs is given in Figure 1.1. The nonlinear DAEs are first discretized in time. A numerical integration formula is used to represent a variable's time derivative as an algebraic function of that variable and its past time derivatives. For the specific value of time, the nonlinear DAEs are thus reduced to a system of nonlinear equations. This system of equations,

$$f_i(x_1, x_2, \cdots x_N) = 0, \quad i = 1, \ldots, N, \tag{1.2}$$

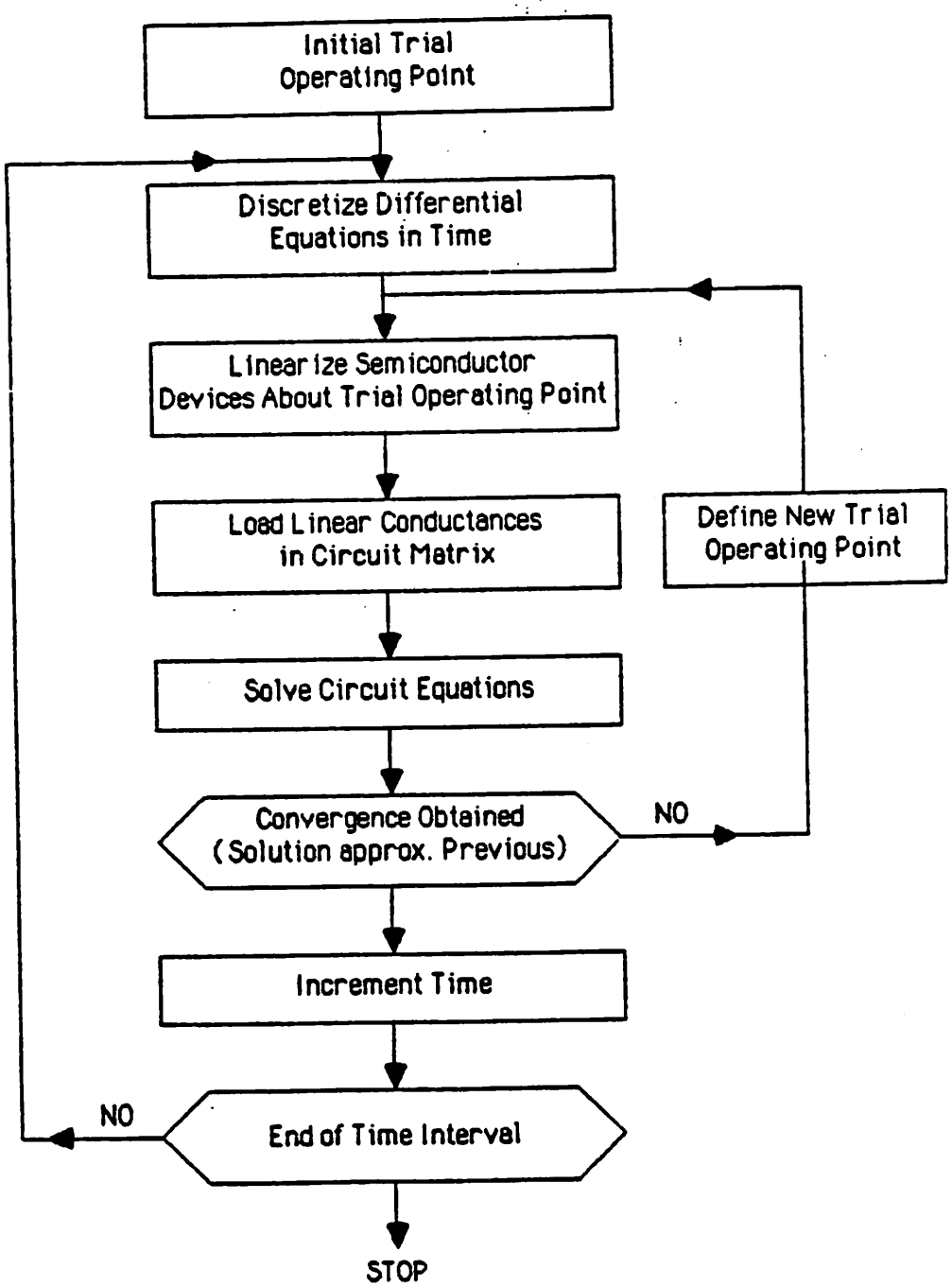
is a function of only the independent variables. The system of nonlinear equations is generally solved using the Newton-Raphson iterative method(NR) [DaB74] [Nag75].

The NR requires the semiconductor device equations to be linearized about an operating point. In general, the terminal currents of an integrated-circuit device are modeled by nonlinear functions of the device's terminal voltages. The terminal currents and the derivatives of the terminal currents with respect the terminal voltages are used in the linearization. The device's linear companion model[ChL75] is formed from the currents and derivatives. As a result, the system of nonlinear circuit equations is reduced to a system of linear equations of the form

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}, \tag{1.3}$$

which are then solved for $\mathbf{x}$. The steps of linearizing the device equations followed by solving the linear circuit equations are repeated until the system of nonlinear equations converges to a solution.

Once the system of equations is solved for the specific time, the time is incremented. The differential equations are then discretized at the new time, and the nonlinear equa-



Transient Simulation
Figure 1.1

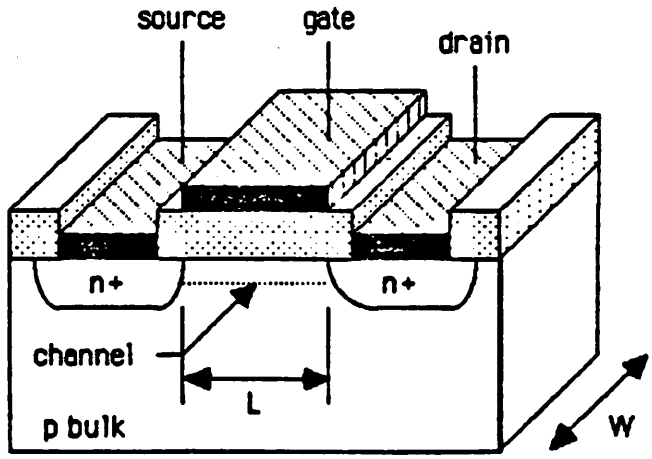
tion solution process is repeated.

1.3. MOS-Transistor Representation for Circuit Simulation

A brief description of the physical characteristics of the MOS transistor is first given in this section. A first-order description of the current-voltage operation of the MOS transistor is then given. Finally, the MOS transistor's companion model is described.

1.3.1. Physical Description of the MOS Transistor

The basic structure of an N-channel enhancement-type[2] MOS transistor is given in Figure 1.2. The MOS transistor is a four terminal device. The device is built on a p-type substrate which is the bulk terminal of the device. Two n+ diffusions[3] form the drain and source terminals of the device. The area between the source and drain diffusions is covered by a thin insulator, which is referred to as the gate oxide. A contact on the gate



MOS Transistor Diagram
Figure 1.2

[2]N-channel depletion type and P-channel devices are discussed later.

[3]Can be formed by either ion-implantation or diffusion.

oxide forms the gate terminal of the device.

In normal operation the source-to-bulk and drain-to-bulk junctions are reverse biased, negligible DC current flows from either the source or drain terminal to the bulk terminal, and the bias between the drain and source terminal is nonnegative. Also, since the gate contact is separated from the other terminals by an insulating material, no DC current flows through the gate terminal.

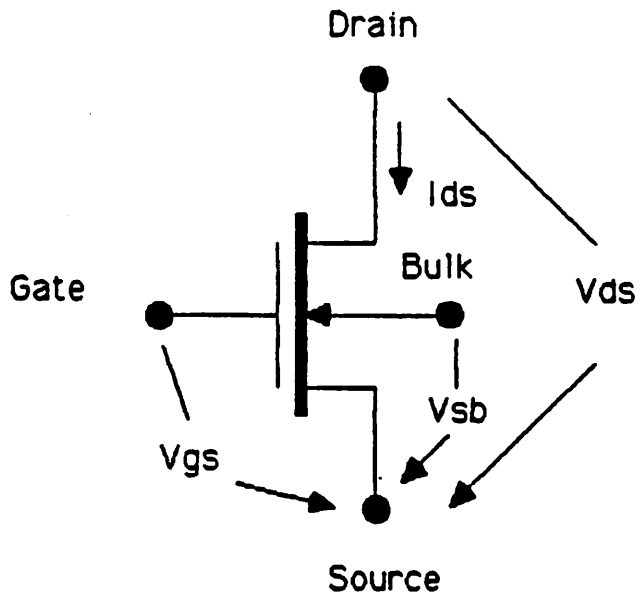
A bias applied to the gate terminal is used to control the flow of current between the source and drain terminals. When there is zero bias applied to the gate terminal relative to the source terminal, no current(neglecting leakage through the reverse-biased p-n junction) flows through either the source or drain terminals. By applying a sufficiently large positive bias to the gate, a channel is formed at the surface below the gate insulator. This channel provides for the conduction of current between the drain and source terminals. Positive current flows from the drain to the source if the bias between the drain and source is positive, and positive current flows from the source to the drain if the bias between the drain and source is negative.

The above discussion can be extended to the p-channel MOS transistor by exchanging the p material with n material and by reversing the polarity of the voltages and currents. The n-channel depletion-type device differs from the enhancement-type device in that the channel exists with zero bias applied to the gate. For nonnegative gate bias the depletion-type device can always conduct current between its drain and source terminals. A sufficiently large negative voltage must be applied to the gate to remove the channel.

1.3.2. First-Order Device Characteristics

A qualitative discussion of the MOS device operation is illustrated by using the Shichman-Hodges(SH) analytic model of the MOS transistor. The description is of a n-channel enhancement-type device, but can be applied to n-channel depletion and p-

channel devices. Throughout this discussion, the drain-to-source voltage is always non-negative, and the source-to-bulk and drain-to-bulk pn junctions are reversed biased. The leakage current in the reverse-biased junctions are small and are neglected in this discussion. The symbol of the n-channel enhancement-type MOS transistor is given in Figure 1.3.



MOS Transistor Symbol
Figure 1.3

The operation of a four-terminal device can be represented as a function of three independent differential-terminal voltages. The MOS transistor, in general, is represented as a function of its drain-to-source(V_{ds}), gate-to-source(V_{gs}) and source-to-bulk(V_{sb}) differential-terminal voltages. Since no DC current flows into the gate terminal and the leakage current from the reverse-biased junctions are neglected, only the current flowing from the drain terminal to the source terminal, drain-to-source current(I_{ds}), is considered. I_{ds}, where

$$I_{ds} = f(V_{ds}, V_{gs}, V_{sb}). \tag{1.4}$$

is a function of the V_{ds}, V_{gs} and V_{sb} voltages.

The SH model equations of I_{ds} are [ShH68]

$$I_{ds} = 0 . \quad \text{for } V_{gs} \leqslant V_t . \tag{1.5}$$

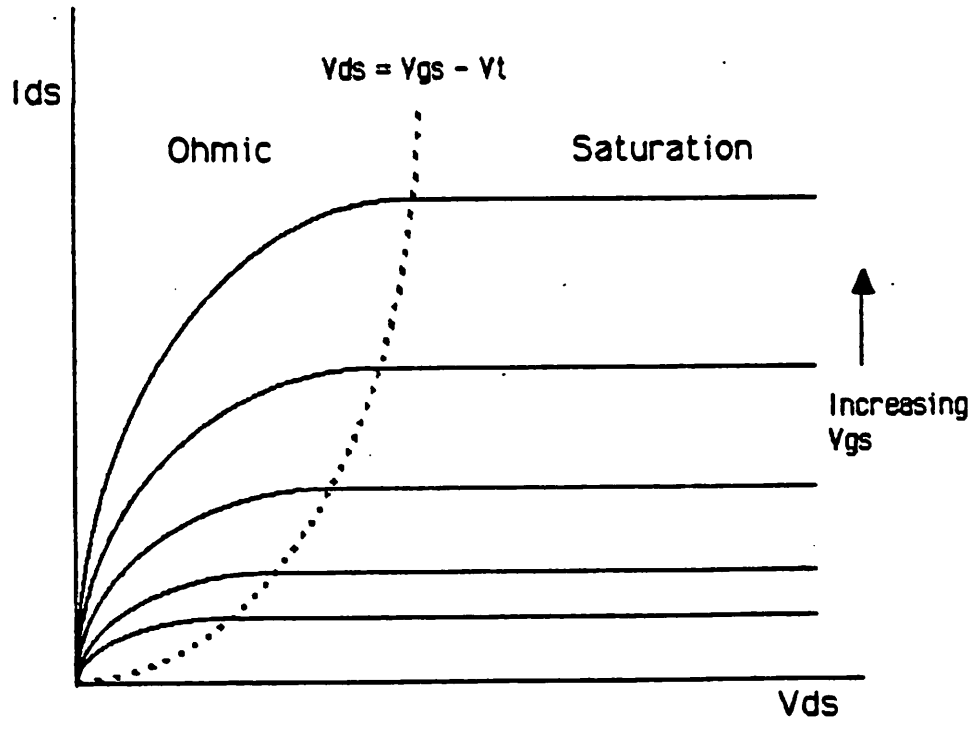
$$I_{ds} = \frac{\mu_n C_{ox}}{2} \frac{W}{L} (V_{gs} - V_t)^2 (1 + \lambda V_{ds}) . \tag{1.6}$$

$$\text{for } V_{gs} > V_t \text{ and } V_{ds} \geqslant V_{gs} - V_t .$$

$$I_{ds} = \mu_n C_{ox} \frac{W}{L} V_{ds} (V_{gs} - V_t - \frac{V_{ds}}{2}) (1 + \lambda V_{ds}) . \tag{1.7}$$

$$\text{for } V_{gs} > V_t \text{ and } V_{ds} < V_{gs} - V_t .$$

As shown in Figure 1.3, V_{ds}, V_{gs} and V_{sb} are the differential terminal voltages of the device. μ_n is the mobility of electrons(holes for p-channel) in the channel. C_{ox} is the capacitance per unit area of the gate oxide. λ is the channel-length modulation parameter. V_t is the Threshold Voltage, and W and L are the channel dimensions as depicted in Figure 1.2. An example of the I_{ds} characteristics is shown in Figure 1.4.



Example I_{ds} Characteristics
Figure 1.4

As mentioned in the previous section, a channel is formed between the MOS transistor's drain and source when a sufficiently large positive voltage is applied to the gate terminal with respect to the source terminal. Provided the V_{ds} is positive, a positive I_{ds} will flow from the drain terminal to the source terminal. The V_{gs} at which the channel is formed is referred to as the Threshold Voltage, V_t. When $V_{gs} < V_t$ no channel is formed and the I_{ds} current is [MuK77]

$$I_{ds} = 0 . \quad \text{for } V_{gs} < V_t. \tag{1.8}$$

A transistor operating under these bias conditions is referred to as cut-off.

The value of V_t is dependent on the V_{sb}. The more negative the bulk voltage is with respect to the source voltage, the greater the V_{gs} voltage that must be applied to form the channel. This is referred to as the body effect. The V_t is [MuK77]

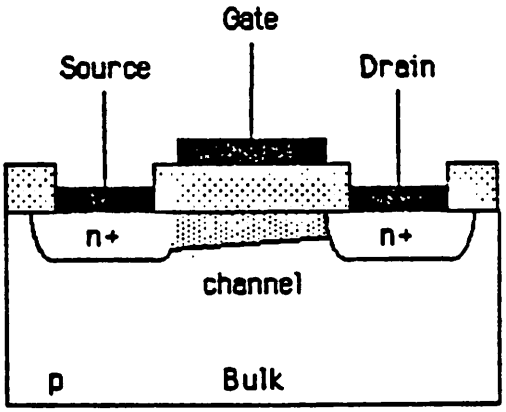
$$V_t = V_{to} + \gamma(\sqrt{V_{sb} + 2|\phi_p|} - \sqrt{2|\phi_p|}). \tag{1.9}$$

V_{to} is the Threshold Voltage at $V_{sb} = 0$. ϕ_p is the Fermi potential and $2|\phi_p|$ is the surface potential of the neutral bulk material. γ is the bulk threshold parameter.

When the V_{gs} is greater than the V_t, and V_{ds} is small, the channel extends from the source to the drain as shown in Figure 1.5. The structure formed by the channel, drain and source is similar to a voltage-dependent resistor, and the behavior of the I_{ds} on the V_{ds} is similar to that of a nonlinear resistor. The I_{ds} is given by [MuK77]

$$I_{ds} = \mu_n C_{ox} \frac{W}{L} V_{ds} (V_{gs} - V_t - \frac{V_{ds}}{2}). \tag{1.10}$$

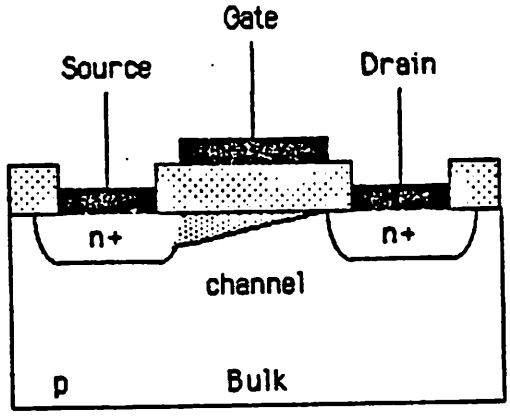
This is referred to as the ohmic or linear region of transistor operation.



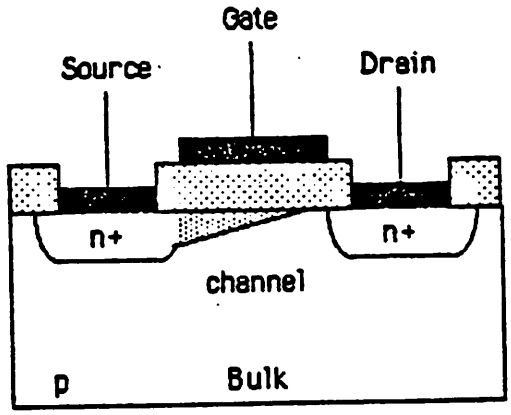
MOS Transistor in the Ohmic Region
Figure 1.5

The channel will extend from the source to the drain until the V_{ds} exceeds $V_{gs} - V_t$, as shown in Figure 1.6. $V_{gs} - V_t$ is referred to as the drain-to-source saturation voltage($V_{ds\,sat}$), and it is the voltage at which the onset of saturation occurs. The device is operating in saturation region of operation when the V_{ds} is greater than the $V_{ds\,sat}$. The I_{ds} of the transistor in saturation is [MuK77]

$$I_{ds} = \frac{\mu_n C_{ox}}{2} \frac{W}{L} (V_{gs} - V_t)^2. \qquad (1.11)$$



MOS Transistor at the Onset of Saturation



MOS Transistor in the Saturation Region
Figure 1.6

The I_{ds} modeled by Equation (1.10) is independent of V_{ds}. In general, MOS transistors demonstrate a slight dependence on V_{ds} when the device is in saturation. To model the linear dependence on V_{ds}, Equation (1.10) is changed to [MuK77]

$$I_{ds} = \frac{\mu_n C_{ox}}{2} \frac{W}{L} (V_{gs} - V_t)^2 (1 + \lambda V_{ds}). \qquad (1.12)$$

To maintain continuous current and partial derivatives, Equation (1.9) is changed to

$$I_{ds} = \mu_n C_{ox} \frac{W}{L} V_{ds} (V_{gs} - V_t - \frac{V_{ds}}{2}) (1 + \lambda V_{ds}). \quad (1.13)$$

1.3.3. Companion Model of the MOS Transistor

The companion model of the MOS transistor is described in this section. A companion model is a linear representation of the nonlinear device about a given bias point. The companion model used with the NR method is derived from the first two terms in the Taylor series expansion [DaB74] of the device equation about the bias point.

The MOS transistor I_{ds} equation is of the form

$$I_{ds} = f(V_{ds}, V_{gs}, V_{sb}). \quad (1.14)$$

The bias-point voltages are $V_{ds\,Q}$, $V_{gs\,Q}$ and $V_{sb\,Q}$. The truncated Taylor series about the bias point is

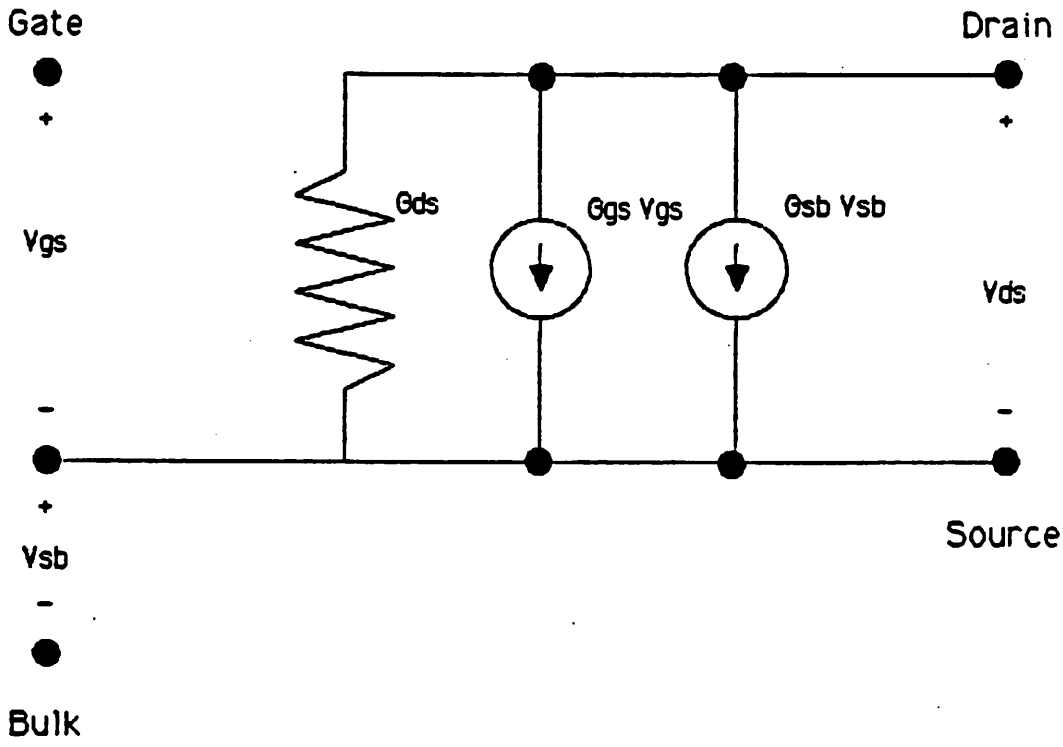
$$I_{ds} = I_Q + \frac{\partial I_{ds}}{\partial V_{ds}}(V_{ds} - V_{ds\,Q}) + \frac{\partial I_{ds}}{\partial V_{gs}}(V_{gs} - V_{gs\,Q}) + \frac{\partial I_{ds}}{\partial V_{sb}}(V_{sb} - V_{sb\,Q}). \quad (1.15)$$

where I_Q is the current at the bias point and $\frac{\partial I_{ds}}{\partial V_{ds}}$, $\frac{\partial I_{ds}}{\partial V_{gs}}$, and $\frac{\partial I_{ds}}{\partial V_{sb}}$ are the partial derivatives of f with respect to V_{ds}, V_{gs} and V_{sb} evaluated at the bias point. The I_{ds} is thus represented as a linear function of V_{ds}, V_{gs}, and V_{sb}. The constant and linear terms in Equation (1.15) are grouped, and Equation (1.15) is rewritten as

$$I_{ds} = \frac{\partial I_{ds}}{\partial V_{ds}} V_{ds} + \frac{\partial I_{ds}}{\partial V_{gs}} V_{gs} + \frac{\partial I_{ds}}{\partial V_{sb}} V_{sb} + I_{EQ}. \quad (1.16)$$

where I_{EQ} is the constant term.

Neglecting any drain-to-bulk and source-to-bulk reverse-biased junction current, Equation (1.15) defines the MOS transistor's DC companion model. The linear model is illustrated in Figure 1.7. The partial-derivative terms have the dimensions of conductance (Ω^{-1}), and their common symbols and names are given in Table 1.1.



MOS Transistor Companion Model
Figure 1.7

Partial Derivative	Symbol	Name
$\frac{\partial I_{ds}}{\partial V_{ds}}$	G_{ds}	Output Conductance
$\frac{\partial I_{ds}}{\partial V_{gs}}$	G_{gs}	Gate-to-Source Transconductance
$\frac{\partial I_{ds}}{\partial V_{sb}}$	G_{sb}	Source-to-Bulk Transconductance

Table 1.1

The components of the companion model are entered into the circuit matrix. matrices A and b in Equation (1.3), representing the linearized circuit. The matrix locations are specified by the circuit template of the transistor. The template of the MOS

transistor used with a nodal-based circuit matrix is given in Figure 1.8. In reference to Equation (1.3), A is the Nodal-Admittance Matrix and b is the Right-Hand-Side Vector. Each row represents a circuit equation, and each column specifies the terminal voltage at that node. For example, the first row in the Nodal-Admittance Matrix template and the Right-Hand-Side Vector template specify the drain current calculated from the companion model.

$$I_d = G_{ds} V_d + G_{gs} V_g - (G_{ds} + G_{gs} - G_{sb}) V_s - G_{sb} V_b + I_{EQ} . \tag{1.17}$$

	Drain	Gate	Source	Bulk
Drain	G_{ds}	G_{gs}	$-G_{ds} - G_{gs} + G_{sb}$	$-G_{sb}$
Gate				
Source	$-G_{ds}$	$-G_{gs}$	$G_{ds} + G_{gs} - G_{sb}$	G_{sb}
Bulk				

Nodal-Admittance Matrix Template

Drain	$-I_{EQ}$
Gate	
Source	I_{EQ}
Bulk	

Right-Hand-Side Vector Template

MOS Matrix Template
Figure 1.8

1.3.4. MOS-Transistor Evaluation Routine

A qualitative description of the model-evaluation routine of the MOS transistor is presented in this section. In Figure 1.1, the simulation steps represented by the boxes "Linearize Semiconductor Devices About Trial Operating Point" and "Load Linear Conductances in Circuit Matrix" are performed by the model-evaluation routines.

For the model-evaluation routine of the MOS transistor, the following procedure is followed in the evaluation of each transistor.

(1) Limit Terminal Voltages

(2) Calculate the Terms of the MOS Companion Model

(3) Load Companion Model

(4) Check For Device Current Convergence

The differential terminal voltages, V_{ds}, V_{gs} and V_{sb}, are calculated from the terminal voltages. Once calculated, the terminal voltages are then limited. The differential terminal voltages of all nonlinear devices are limited to aid in the convergence of the Newton-Raphson algorithm and the prevention of numerical overflow[4].

The terms of the companion model are then calculated. As described earlier, the elements of the MOS transistor's companion model are determined from the values of I_{ds}, G_{ds}, G_{gs} and G_{sb}. The transistor model represents the current and conductances as functions of the V_{ds}, V_{gs} and V_{sb} voltages, namely

$$I_{ds} = I_{ds}(V_{ds}, V_{gs}, V_{sb}). \tag{1.18}$$

$$G_{ds} = G_{ds}(V_{ds}, V_{gs}, V_{sb}). \tag{1.19}$$

$$G_{gs} = G_{gs}(V_{ds}, V_{gs}, V_{sb}). \tag{1.20}$$

$$G_{sb} = G_{sb}(V_{ds}, V_{gs}, V_{sb}). \tag{1.21}$$

The equations representing the current and partial derivatives are evaluated at the given

[4]Numerical overflow is of greatest concern with bipolar transistors and pn junction diodes because of the exponential behavior of their current on terminal voltage.

voltage values. The companion-model entries are then added into the circuit matrix at the positions specified by the device's template.

As shown in Figure 1.1, the results of the "Solve Circuit Equations" step are tested for convergence. If the convergence test fails, the NR process is continued, but if the convergence test passes, the solution of the DAEs continues for next time point. Generally, the solution of the circuit equations provide node voltages and branch currents of voltage sources. As given by [Nag75], the current of every nonlinear device must also be checked for convergence. For the MOS transistor, the accuracy of the I_{ds} is checked in the model-evaluation routine.

CHAPTER 2

System Overview of the MOS-Model Attached Processor

The use of the special-purpose MMAP from the computer-system perspective is presented in this chapter. The MMAP is shown to be a logical partition from both the circuit-simulation and computer-architecture perspective, performing extensive computation with minimal transfer of data. First, a general overview of attached processors is given. The general overview presents three criteria which must be considered when developing a special-purpose attached processor. Next, a detailed description of the function of the MMAP is given, and the three criteria are applied specifically to the MMAP to demonstrate its validity. Finally, the interaction between the circuit simulation program and the MMAP is described.

2.1. General Overview of Special-Purpose Attached Processors

A special-purpose attached processor operating in conjunction with a host processor is used to improve the speed performance of programs. The attached processor performs a small set of functions at a greater speed than the more general host. A floating-point coprocessor is one such example. The floating-point coprocessor performs floating-point operations much faster than they can be done in software by the host processor. In general, the use of an attached processor to increase the speed performance of a function(s) is justified when the following three criteria are met.

(1) The function(s) performed by the attached processor comprise a significant percentage of the overall time of an application program(s) running on the host.

(2) The function to be performed by the attached processor is a logical "break-off" point. The overhead required by the host for data preparation and

communication with the attached processor does not negate the speed advantage of the attached processor.

(3) The time required to execute the function by the attached processor is significantly faster than execution of the function by the host alone.

2.1.1. Definitions

Three definitions, Function-Usage Percentage(FP), Improvement Percentage(IP) and Attached-Processor Efficiency(AE), are given in this section. These definitions are used to characterize the performance of an attached processor and are related to the above-mentioned three criteria.

Function-Usage Percentage

The FP is the percentage of total time that is used in performing a function.

$$FP = \frac{T_{function}}{T_{total}} \times 100\ \% \tag{2.1}$$

The larger the FP, the more valid the reason to develop special hardware to perform that function.

Attached-Processor Efficiency

For a host using an attached processor, the time required to perform the application program(s) is the sum of the time required by the host working independently of the attached processor, the time required for the attached processor to work in conjunction with the host, and the time required by the attached processor working independent of the host.

$$T_{total} = T_{host} + T_{attached\ processor} + T_{host-attached\ processor} \tag{2.2}$$

The effective time to perform the function by the attached processor is equal to the time of the attached processor working independently plus the time required by the host to

communicate with the attached processor. The AE is the percentage of the effective function evaluation time (function evaluation as viewed by the host) that is required by the attached processor to perform the function.

$$AE = \frac{T_{\text{attached processor}}}{T_{\text{attached processor}} + T_{\text{host -attached processor}}} \times 100\ \% \qquad (2.3)$$

The larger the AE, the greater the amount of work that is actually being done by the attached processor relative to the communication time between the host and attached processor.

Improvement Percentage

The IP is the percentage of time that is saved in the execution of application program(s) by using an attached processor.

$$IP = \frac{T_{\text{total}} - T_{\text{total w / attached processor}}}{T_{\text{total}}} \times 100\ \% \qquad (2.4)$$

In the limiting case the function evaluation by the attached processor becomes insignificant compared to the evaluation of that function by the host. The difference between the total times with and without the attached processor is then equal to the time required by the host to evaluate the function. Thus, the maximum IP approaches the FP.

$$\text{Maximum Improvement} = \frac{T_{\text{function}}}{T_{\text{total}}} \times 100\ \% \qquad (2.5)$$

2.1.2. Concurrent Operation of the Host and Attached Processor

The host and attached processor can both be used concurrently, provided a problem can be partitioned properly. The total time is the sum of the host-dependent times, as the attached processor will be operating concurrently with the host processor.

$$T_{\text{total}} = T_{\text{host}} + T_{\text{host -attached processor}} \qquad (2.6)$$

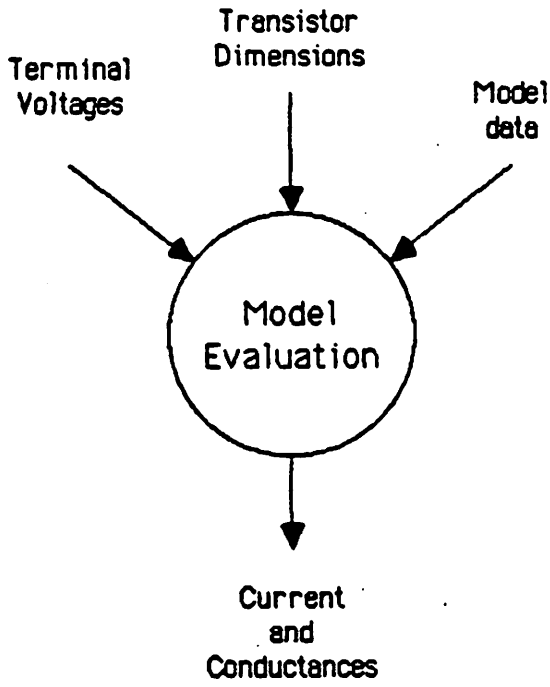
The time required for the interaction of the host and attached processor is primarily due to communication between the host and attached processor. Thus minimizing the amount of data transferred, and thus the communication time, minimizes the amount of total time.

2.2. MMAP System Performance

The use of a MMAP is justified only if it meets the three criteria stated in Section 2.1. In this section, the suitability of using an attached processor for model evaluation is presented, on the basis of the definitions given in section 2.1.

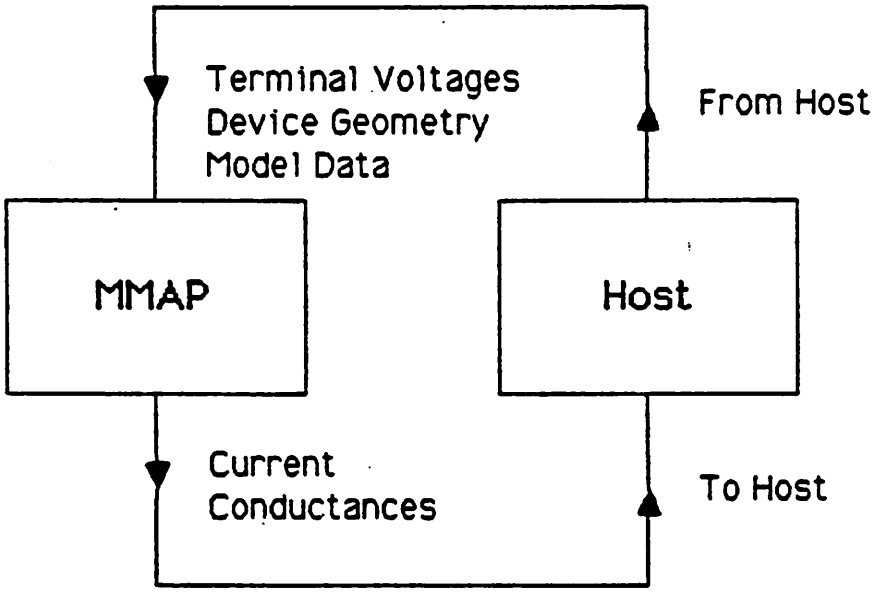
2.2.1. MMAP Function and Organization

The terminal voltages, transistor dimensions, and model data are required for MOS transistor model evaluation. As shown in Figure 2.1, the evaluation of the model equations provides the transistor's current and partial derivatives with respect to the terminal voltages.



Model Evaluation
Figure 2.1

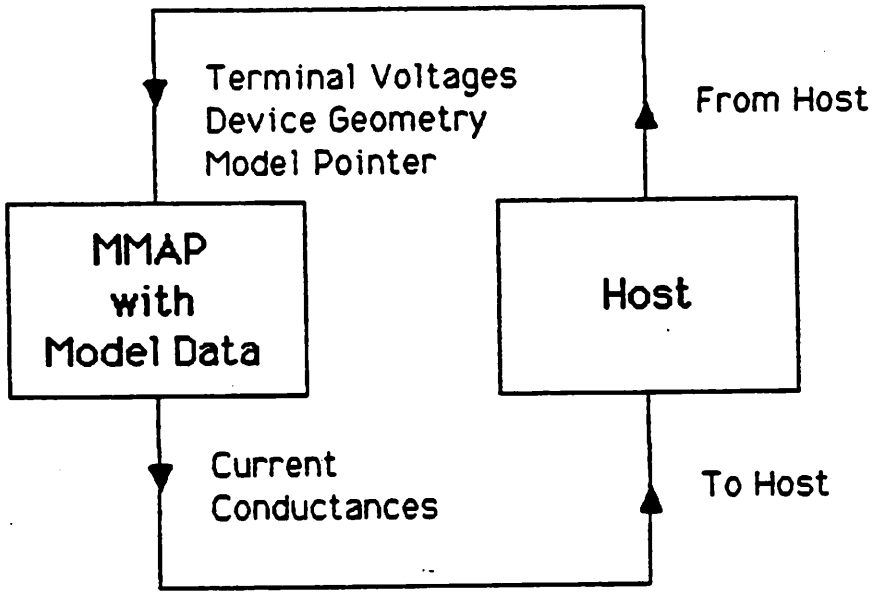
In an electrical circuit simulation program, the transistor evaluation is preformed by a software subroutine. The MMAP replaces the software transistor-evaluation subroutine and is specifically designed to perform the transistor evaluation. The MMAP is a "slave" to the host computer it is connected to. The transistor's terminal voltages, dimensions and model data are input to the MMAP by the host; the model equations are evaluated by the MMAP and the results are then returned to the host as shown in Figure 2.2.



MMAP
Figure 2.2

A transistor's dimensions and model data remain constant throughout the current simulation, whereas the transistor's terminal voltages usually change with each iteration. The model data are the same for all transistors of the same type. The number of different transistor models is dependent on the integrated-circuit process and is independent of the number of transistors in the circuit being simulated. The transistor model data are stored in the MMAP since they remain constant during the simulation. The transistor's dimensions remain constant and thus they can also be stored in the MMAP. But, since the geometric data are unique for each transistor, storage of the geometric data in the MMAP could become prohibitive for large circuits. As a result, only the model data are stored

local to the MMAP, and a model reference pointer is used to associate a transistor with the appropriate model data. The model reference pointer provides the address of the model data stored in the MMAP. As shown in Figure 2.3, the model reference pointer, transistor dimensions and differential terminal voltages are sent to the MMAP.



MMAP With Local Storage of Model Data
Figure 2.3

2.2.2. Function-Usage Percentage of the MMAP

The FP, as defined in Section 2.1, is the percentage of total computation time used in performing a given function. The greater the percentage is, the more valid the develop-

ment of special-purpose hardware to perform that function. In relation to MOS-transistor evaluation in circuit simulation, the FP is,

$$FP = \frac{\text{Total Time Used in Model Evaluation}}{\text{Total Simulation Time}} \times 100\ \%\ . \qquad (2.7)$$

the percentage of the total simulation time that is spent evaluating the MOS transistor equations. The percentage is not a constant, but will vary depending on the size of the circuit, the size of the circuit relative to the number of MOS transistors, the complexity of the MOS-model equations and circuit function.

The FPs for three MOS circuits[1] are given in Table 2.1. The percentages are calculated from the time profile of the BIASC program performing the DC-transfer analysis of three MOS example circuits. The BIASC program is used here since the prototype MMAP, presented in Chapter 6, is used in conjunction with the BIASC program. The time profile contains the evaluation time for the MOS-transistors model equations, the linear-equation solution time and the time required to perform additional operations (e.g., convergence checking). Both the Shichman-Hodges(SH)[ShH68] and SPICE Level-2[VlL80] transistor models[2] are used in the simulations. The profile information is from the BIASC program running on a VAX 11/785 with hardware floating-point under the Berkeley Unix BSD 4.3 operating system.

25 Cascaded NMOS Inverters			
27 Circuit Nodes and 50 MOS Transistors			
MOS Model	% Model Evaluation	% Matrix Solution	% other
Shichman-Hodges	59	30	11
Level 2	63	27	10

Table 2.1a

[1]The input listings for these circuits are given in Appendix B.

[2]Both models are given in Appendix C.

Low-Power CMOS Operational Amplifier			
24 Circuit Nodes and 30 MOS Transistors			
MOS Model	% Model Evaluation	% Matrix Solution	% other
Shichman-Hodges	39	39	22
Level 2	46	35	19

Table 2.1b

Worst-Case Path Through Op-Code PLA			
65 Circuit Nodes and 116 MOS Transistors			
MOS Model	% Model Evaluation	% Matrix Solution	% other
Shichman-Hodges	39	38	23
Level 2	53	43	4

Table 2.1c

For the three circuits simulated. the average FP of SH model evaluation is 44%. and the average FP of SPICE Level-2 model evaluation is 53%.

The matrix-solution time for the three circuits is also significant. Cohen[Coh81] has demonstrated that the use of a "Linear Equation Solution Machine"(LESM) greatly reduces the matrix solution time. Cohen's MOSAMP1 example circuit uses less than 5% of the total time for matrix solution. while 90% of the total time is consumed by MOS transistor model evaluation[Coh81].

The percentage of total time for transistor model evaluation by relaxation-based simulation programs is nearly independent of circuit size. [Sal84] contains the profile information of the SPLICE1.7 program electrically simulating a MOS digital filter circuit containing 698 MOS transistors and 384 circuit nodes. The MOS transistor models used are based on the SH model. The program used 37% of the computation time for transistor model evaluation.

2.2.3. Attached-Processor Efficiency Percentage of the MMAP

The AE of the MMAP is the time required to perform the transistor model evaluation relative to the effective MOS transistor evaluation time, which includes the communication time between the host and MMAP. In relation to the MMAP, Equation 2.3 can be rewritten as

$$\text{MMAP AE} = \frac{\text{MMAP Evaluation Time}}{\text{MMAP Evaluation Time + Communication Time}} \times 100\%. \quad (2.8)$$

The MMAP Evaluation Time is dependent on the number of floating-point operations required, and the Communication Time is dependent on the number of floating-point words transferred between the host and MMAP. The AE of an attached processor performing floating-point operations, such as the MMAP, can be characterized by the ratio of the number of floating-point operations performed to the total number of floating-point words transferred. This ratio is referred to as the Performance Ratio(PR), and the PR applied to the MMAP is

$$\text{Performance Ratio} = \frac{\text{\# Floating–Point Operations}}{\text{\# Floating–Point words Transferred}}. \quad (2.9)$$

The larger the PR, the higher the AE of the MMAP.

The PR of a floating-point attached processor performing a single two-operand floating-point operation is $\frac{1}{3}$ since one floating-point operation is performed, two operands are required and one floating-point result is returned.

The PR of the MMAP is dependent on the MOS transistor model used by the MMAP. As listed in Table 2.1, a total of 9 floating-point numbers plus the model-reference pointer are transferred for each transistor evaluation.

Model	Model Reference Pointer
Geometry	Width Length
Terminal Voltages	V_{ds} V_{gs} V_{sb}
Returned Results	I_{ds} G_{ds} G_{gs} G_{sb}

Table 2.1

If the commonly used SH model is used in the MMAP, the number of floating-point operations and PR is given in Table 2.2. The PR is calculated assuming a total of 10 floating-point numbers are transferred. The model data (C_{ox}, μ, V_{to}, λ, γ and ϕ) are stored in memory local to the MMAP.

Performance Ratio - Current and Partial Derivative Calculation			
Transistor Operating Region	fpt. Operations +, −, × & /	$\sqrt{}$	Performance Ratio
Cutoff	4	2	0.6
Saturation	18	2	2.0
Ohmic	22	2	2.4

Table 2.2

The PR varies from 0.6 to 2.4 with the SH equations. The PR is dependent on the complexity of the equations, which vary depending on the transistor's operating region. The average PR, provided the number of evaluations in each operating region is equal, is 1.6. This is nearly 5 times greater than the PR of a single two-operand floating-point evaluation.

2.2.4. Improvement Percentage Due to the MMAP

IP, applied to a circuit-simulation, is the percentage reduction in simulation time achieved by using a MMAP. Ideally, if the time required by the MMAP to perform the

transistor evaluation is negligible when compared to the host's time, the maximum IP approaches the FP. Under these assumptions, for the data given in Tables 2.1a, 2.1b and 2.1c, the average maximum improvement is 44% for the SH model and 53% for the SPICE Level-2 model.

To achieve the maximum IP, the MMAP must perform the transistor evaluation in much less time than the host. The architecture of the MMAP and transistor-model representation used by the MMAP must be optimized to achieve the largest possible IP. The MOS transistor model used by the MMAP is presented in Chapter 4, and the architecture of the MMAP is presented in Chapter 5.

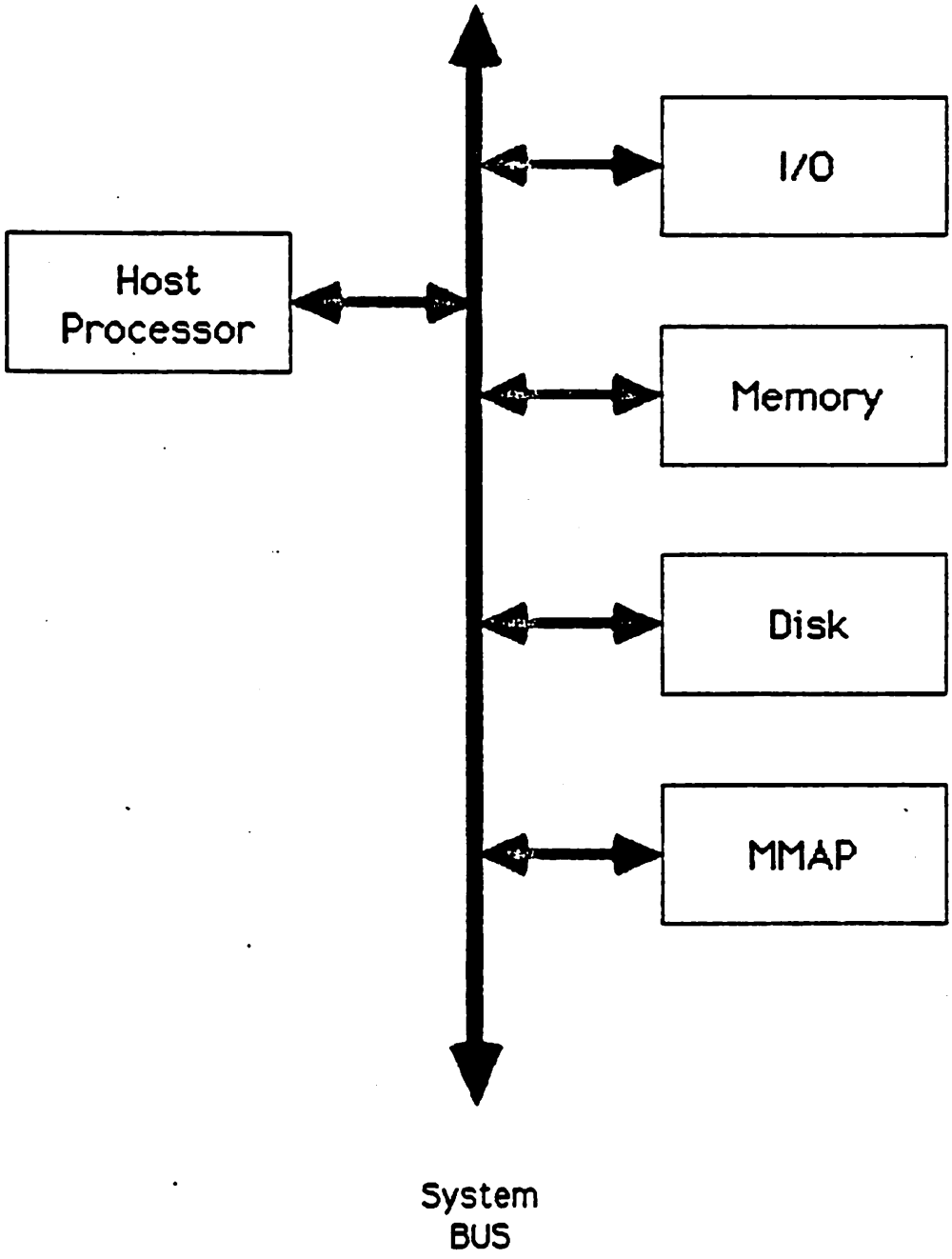
2.3. Using the MMAP in Conjunction with a Circuit-Simulation Program

The interaction between the host and the MMAP required during circuit simulation and the changes made to the circuit-simulation program's transistor-evaluation routine are described in this section.

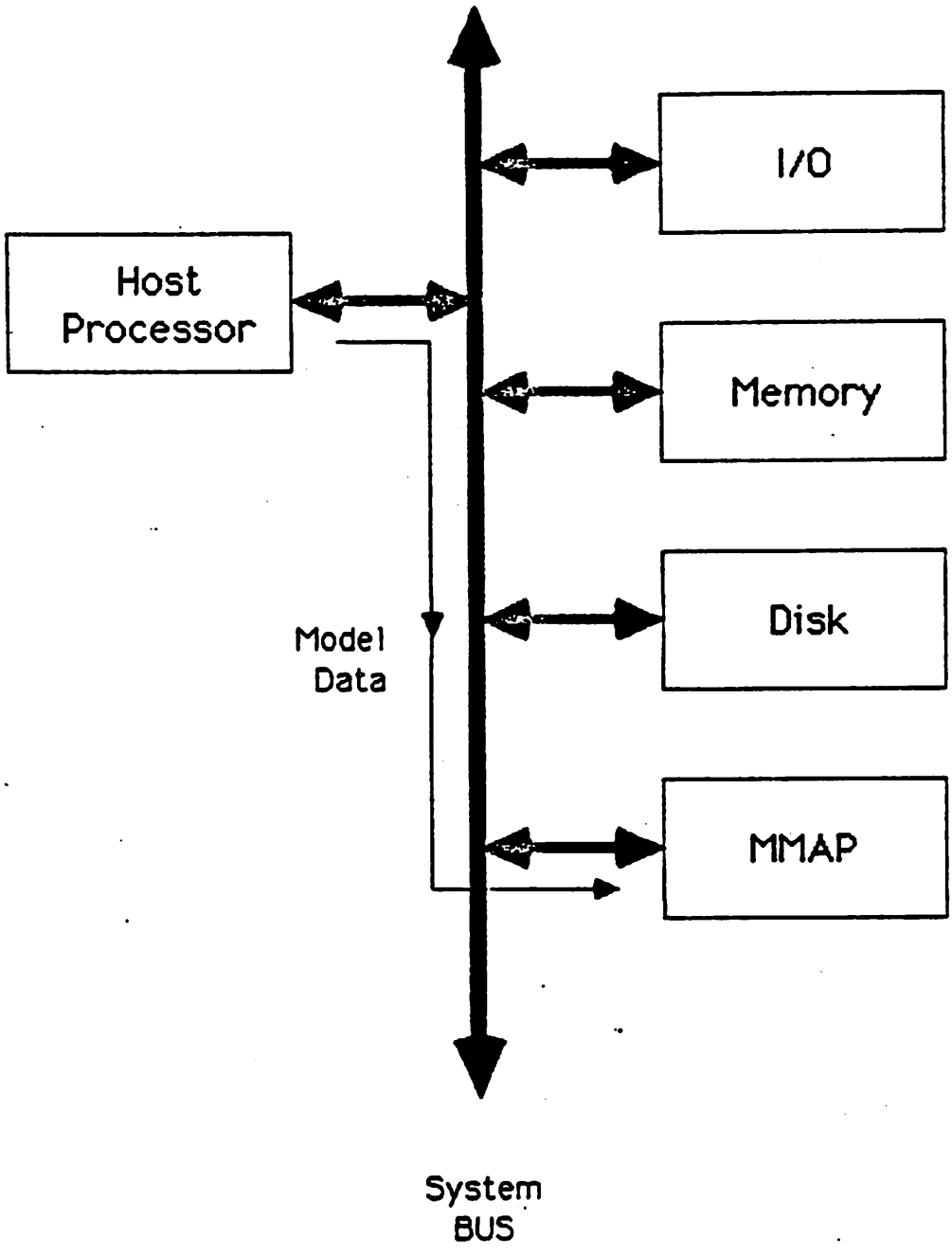
2.3.1. Interaction Between the Host and MMAP

Throughout the discussion of the MMAP's operation, the host is assumed to be a simple single-bus architecture as shown in Figure 2.4. The host's RAM memory, disk, I/O, and MMAP are connected to the host's bus.

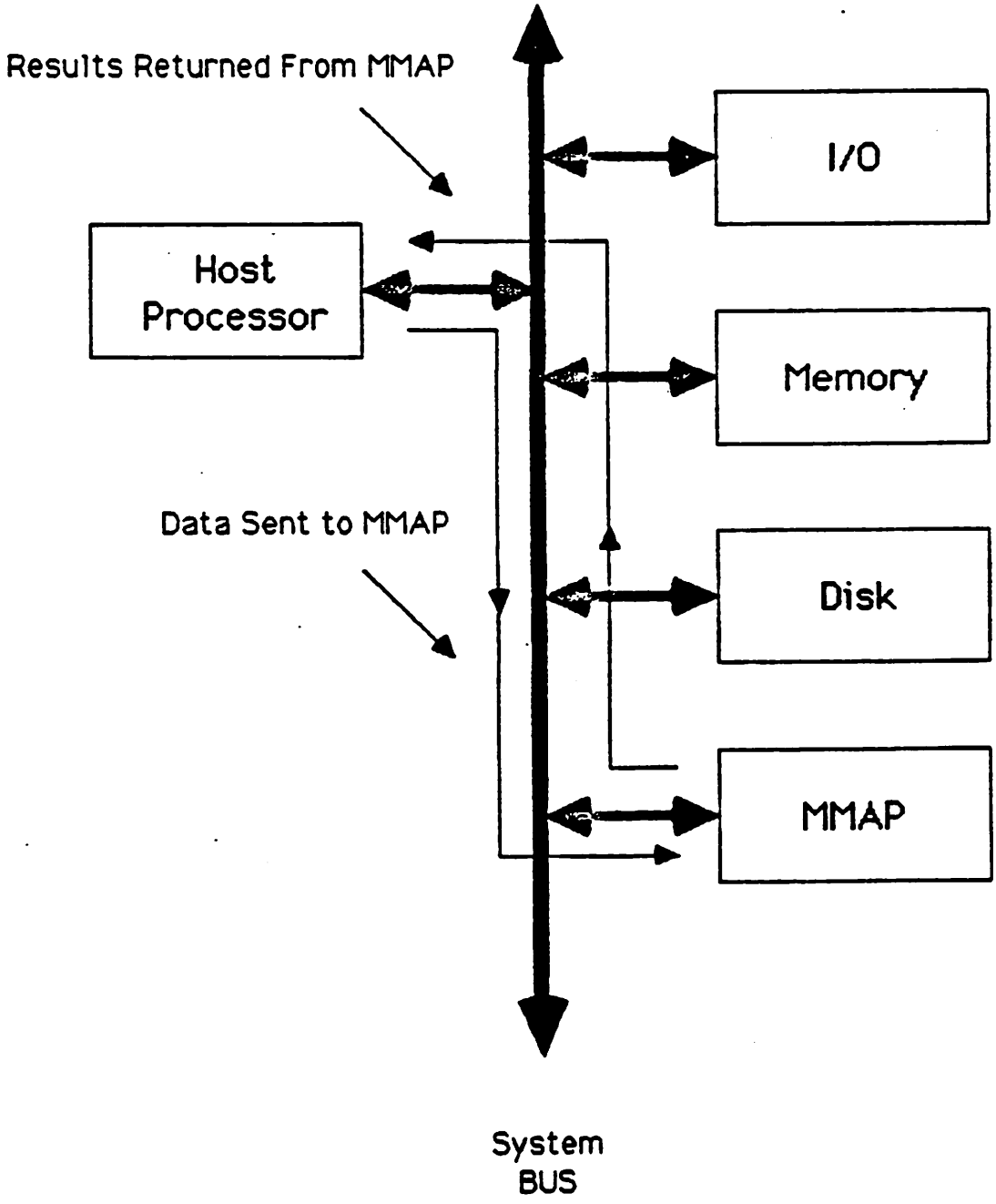
During the initialization phase of the electrical circuit-simulation program, the circuit description is parsed, the sparse-matrix structure is built up, and initial "one-time-only" calculations are performed. In addition, the data for the models required by for the simulation are sent to the MMAP as shown in Figure 2.5. From Figure 2.6, it is seen that when the simulation program requires the evaluation of a MOS transistor, the appropriate data are sent to the MMAP, the MMAP performs the computations, and the results are returned for use by the host processor.



Host Architecture
Figure 2.4



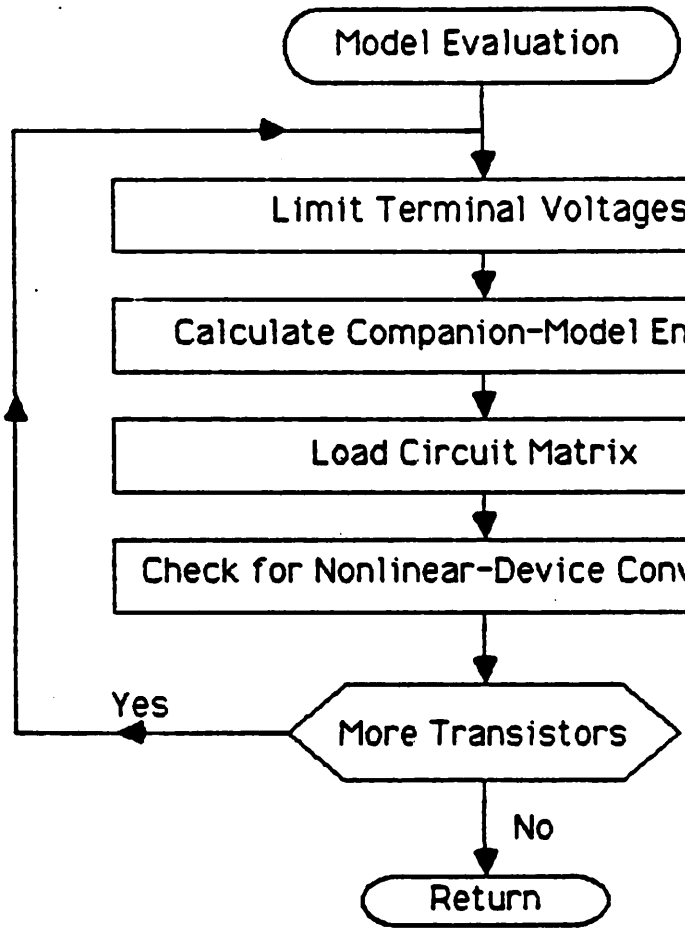
One-Time-Only Loading of Model Data
Figure 2.5



Transistor Evaluation
Figure 2.6

2.3.2. Accessing the MMAP from a Model-Evaluation Routine

A qualitative description of a transistor model-evaluation routine is given in Chapter 1. As shown in Figure 2.7, the steps performed in the evaluation routine are executed sequentially and repeated for every transistor.



Model-Evaluation Routine
Figure 2.7

The MMAP's function is to perform the "Calculate Companion-Model Entries" step in the model evaluation routine. Replacing this step by a call to the MMAP is illustrated in Figure 2.8. The "Calculate Companion Model Entries" step is replaced by three steps: "Send Data to MMAP and Signal Execution", "Calculate Companion Model Entries -

MMAP", and "Access Results From MMAP When Ready". The host remains idle while the MMAP performs the model calculations, and, neglecting the communication time, only the transistor-evaluation time is reduced.

A more efficient model-evaluation routine, shown in Figure 2.9, allows the host to perform work in parallel to the MMAP. The core of the model-evaluation proceeds as follows: while the MMAP evaluates transistor N; the host loads the results of the evaluation of transistor $N-1$ in the circuit matrix, checks for convergence of transistor $N-1$, and limits the terminal voltages of transistor $N+1$. The time required by the host(T_{host}) to limit the terminal voltages(T_{limit}), load the circuit matrix($T_{matrix-load}$), and check for device convergence($T_{converge}$) is

$$T_{host} = T_{limit} + T_{matrix-load} + T_{converge} . \tag{2.10}$$

The total MOS transistor evaluation time(T_{eval}) is equal to the MMAP's model-equation evaluation time(T_{MMAP}) plus the communication time between the MMAP and host(T_{com}).

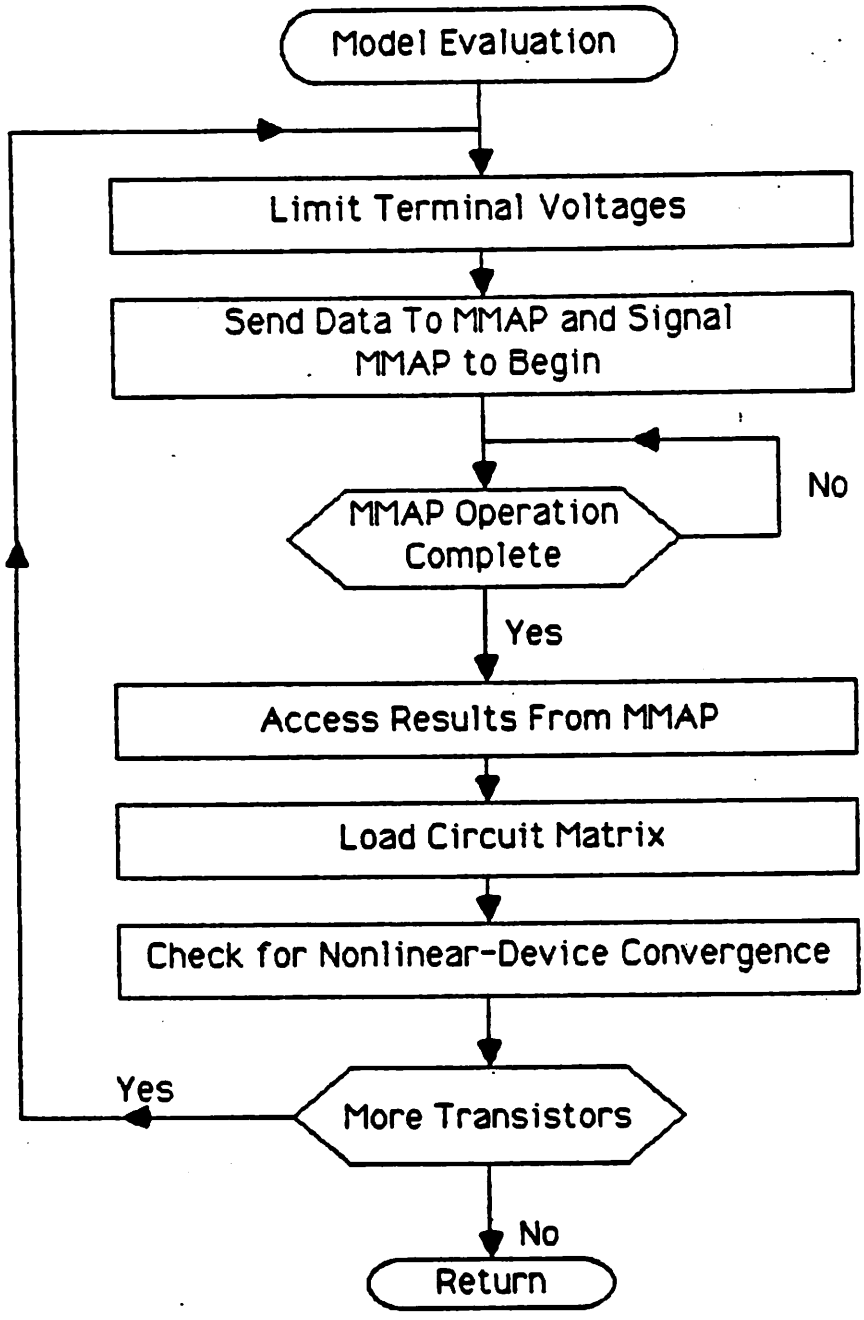
$$T_{eval} = T_{MMAP} + T_{com} . \tag{2.11}$$

provided

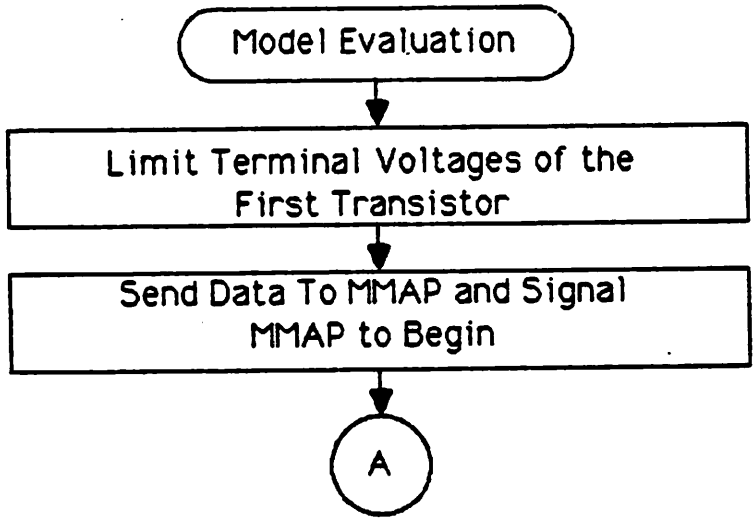
$$T_{host} \leqslant T_{MMAP} . \tag{2.12}$$

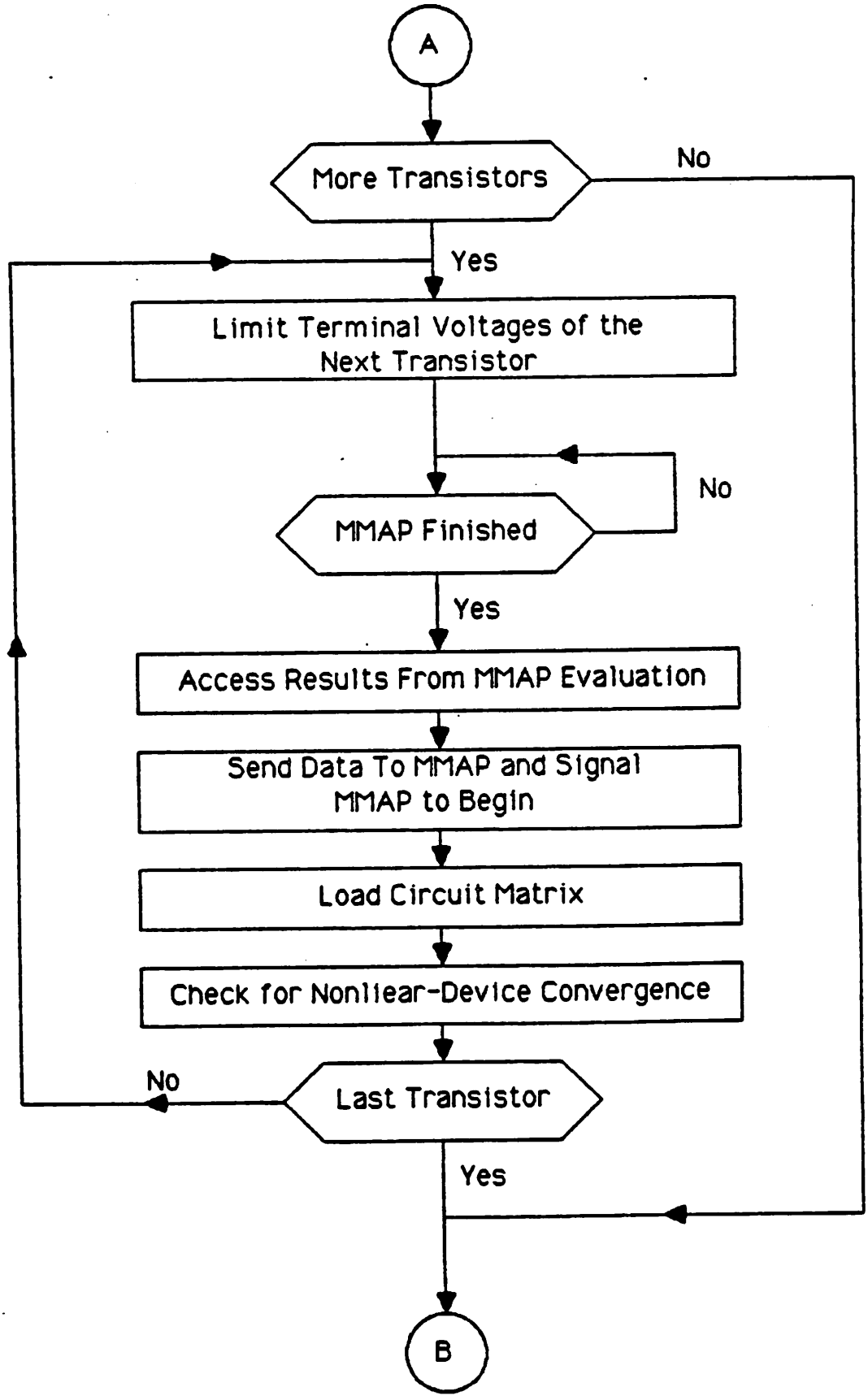
Otherwise, the evaluation time is

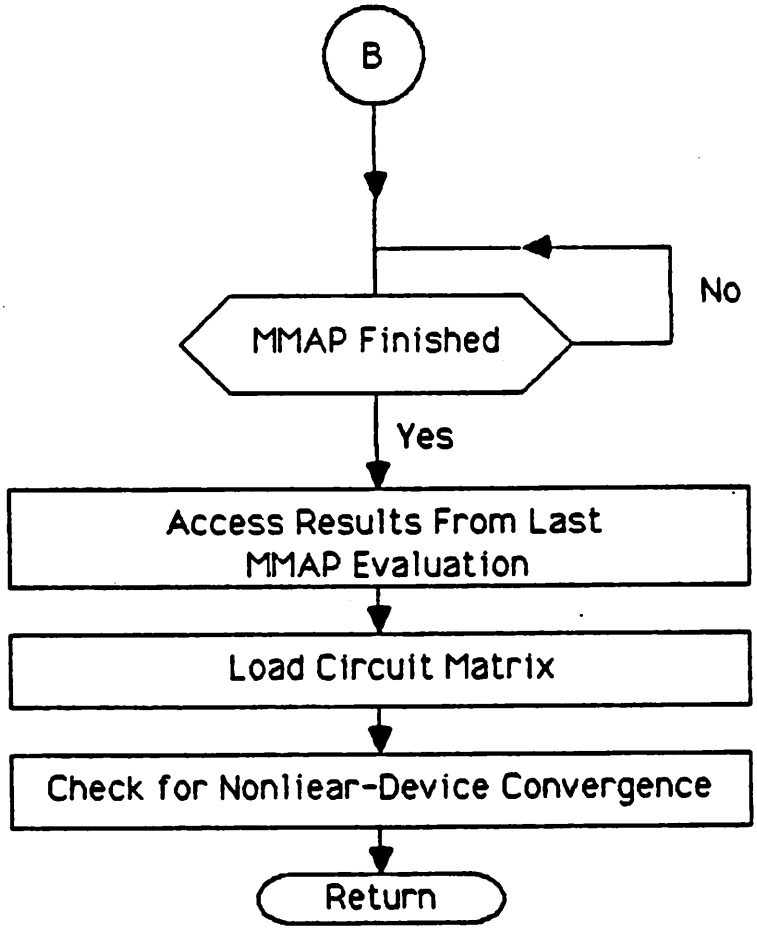
$$T_{eval} = T_{host} + T_{com} . \tag{2.13}$$



Model Evaluation With MMAP
Figure 2.8


Model Evaluation
Limit Terminal Voltages of the First Transistor
Send Data To MMAP and Signal MMAP to Begin
A


A
More Transistors
No
Yes
Limit Terminal Voltages of the Next Transistor
MMAP Finished
No
Yes
Access Results From MMAP Evaluation
Send Data To MMAP and Signal MMAP to Begin
Load Circuit Matrix
Check for Nonlinear-Device Convergence
Last Transistor
No
Yes
B



Improved Model Evaluation With MMAP
Figure 2.9

2.4. Chapter Summary

The use of a MOS-Model Attached Processor(MMAP) in conjunction with a circuit-simulation program is shown to be a logical partition from both the circuit-simulation and computer-architecture perspective. From the simulation outlook, transistor-model evaluation is shown to comprise a significant percentage of the total circuit-simulation time. In lessening the model-evaluation time by using a MMAP, the total simulation time can be reduced. From the architecture view, the MMAP performs a large amount of work with only a minimal transfer of data to and from the host. With a small communication overhead, the MMAP can be efficiently used by the host.

The software model-evaluation routine in the simulation program is changed to include a call to the MMAP. The MMAP performs the transistor equation evaluation and the host performs the remaining model-evaluation tasks. The MMAP and host operate concurrently. The host processes results from the previous MMAP evaluation and prepares data for the next MMAP evaluation, while the MMAP calculates the data for the current MOS transistor.

CHAPTER 3

MOS-Transistor Model Representations

In Chapter 2 it is demonstrated that the use of a MOS-Model Attached Processor(MMAP) is feasible provided the transistor evaluation can be performed by the MMAP at a much greater speed than the host processor and provided the communication between the MMAP and host is fast. On a given computer, the time required for a transistor evaluation in software is dependent on the type of transistor model. The more complicated the transistor model, the greater the time used in solving the model equations. The time required for a transistor evaluation by the MMAP is also dependent on the type of transistor model, but, as stated in Chapter 2, the speed of the model evaluation is primarily dependent on efficiently realizing the transistor model in the architecture of the MMAP.

The different types of MOS transistor models are presented in this chapter. The criteria for comparing the different types of MOS transistor models are first presented. Then an overview of both analytic and empirical MOS transistor model representations is given.

3.1. Criteria Used in Choosing a Transistor-Model Representation

In this section, the criteria used in choosing a transistor model for the MMAP are first given, and then each is described.

As described in Chapter 1, the transistor model represents the current and conductances as functions of the V_{ds}, V_{gs} and V_{sb} voltages, namely

$$I_{ds} = I_{ds}(V_{ds}, V_{gs}, V_{sb}) . \tag{3.1}$$

$$G_{ds} = G_{ds}(V_{ds}, V_{gs}, V_{sb}) . \tag{3.2}$$

$$G_{gs} = G_{gs}(V_{ds}, V_{gs}, V_{sb}) . \tag{3.3}$$

$$G_{sb} = G_{sb}(V_{ds}, V_{gs}, V_{sb}) . \tag{3.4}$$

The current and conductances are used to represent the companion model of the MOS transistor, and are calculated at every Newton-Raphson iteration.

In general, the choice of a MOS-transistor model is based on the model accurately representing the transistor's characteristics while also meeting the numerical requirements of the circuit-simulation program. There are additional concerns for the transistor model used by the MMAP. The model must be efficiently realized in the architecture of the MMAP, and the model should be unaffected by changes in MOS-transistor process technology. The criteria used in choosing a MOS model are summarized in the following list.

(1) Accurately model currents and conductances

(2) Meet the requirements of the circuit-simulation program

(3) Efficiently realized in a hardware architecture

(4) Unaffected by changes in MOS-transistor process technology

These four points are further discussed in the remainder of this section.

3.1.1. Accurate Modeling of Currents and Conductances

In electrical circuit simulation, the MOS transistor model must accurately represent the transistor currents and conductances(partial derivatives of current with respect to voltage). The accurate modeling of the transistor currents is required in the simulation of both analog and digital integrated circuits. In addition, the simulation of analog integrated circuits requires the accurate modeling of transistor conductances.

3.1.2. Meet the Requirements of the Circuit-Simulation Program

As described in Chapter 1, the Newton-Raphson(NR) method is used by circuit-simulation programs to solve the nonlinear circuit equations. To meet the minimum requirements of NR, the I_{ds}, given by Equation (3.1), must be a continuous, monotonic function of V_{ds}, V_{gs} and $(-V_{sb})$. In addition, the first derivatives with respect to voltage, given by Equations (3.2), (3.3) and (3.4), must be nonzero.

3.1.3. Efficiently Realized in a Hardware Architecture

As emphasized in Chapter 2, it is essential that the MMAP perform the transistor evaluation in much less time than the host processor. In general, the model-evaluation time of a given transistor model is limited by the data-access time and the floating-point operation time. As described in Chapter 2, the model data are stored in memory local to the MMAP. The storage of model data in the MMAP reduces the communication overhead between the MMAP and host, and provides direct access to the model data by the MMAP. Circuit-simulation programs are floating-point intensive and are primarily run on computers which have a fast floating-point unit. The MMAP may not be able to perform the floating-point operations much faster than the host, and, under these circumstances, the MMAP may not achieve a significant speed advantage due to floating-point speed alone. In order for the MMAP to achieve any additional speed-up over the host, the transistor model must be well suited for its realization in the MMAP's architecture.

There are two primary considerations in specifying the form of the transistor model from the architecture perspective. First, the number of different logical and numeric operations must be kept to a minimum, and, secondly, the number of control branches required in the model evaluation must be kept to a minimum. Minimizing the number of operations reduces the complexity of the hardware, which allows the performance of the remaining functions to be better optimized, and simplifies the control of the arithmetic and logic unit. Minimizing the number of control branches simplifies the MMAP's

control logic and allows the MMAP's architecture to be further optimized.

3.1.4. Unaffected by Changes in MOS-Transistor Process Technology

The MOS transistor's model equations should not be affected by changes in MOS transistor processing technology. Without the MMAP, the model-evaluation routine is executed solely by the host. To change the transistor equations, in general, requires only the revision of the software model-evaluation routine. With the MMAP, a revision in the model equations can require a change in the MMAP's programming and, in addition, may even require a change in the design of the MMAP.

3.2. Types of MOS-Transistor Models

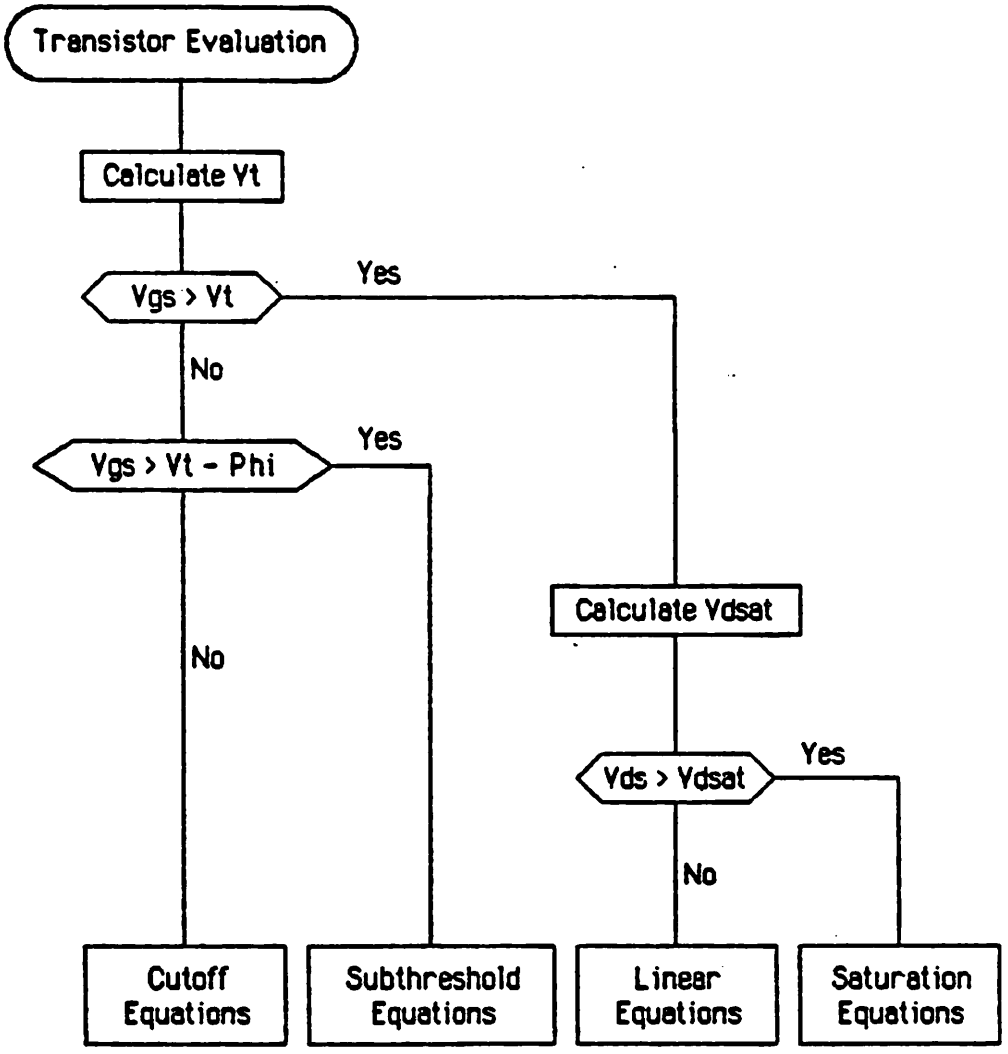
MOS-transistor models may be partitioned into two general groups, analytic and empirical. Analytic-model equations can be derived directly from the physical properties of the MOS device. Empirical MOS transistor models calculate the transistor's current and conductance from values of device current data. Discrete values of data can be stored, or the data can be represented by numerical functions which "curve-fit" the data. In this section, a comparison of the two model types is made relative to the criteria stated in Section 3.1 of this chapter.

3.2.1. Analytic MOS-Transistor Models

The Shichman-Hodges[ShH68] and SPICE Level-2[VIL80] MOS transistor models are examples of analytic models. The two analytic models are of differing complexities but both are derived from the MOS transistor's physical properties. A purely analytic MOS transistor model may not adequately represent the current-voltage relationship of the device. Analytic MOS transistor models usually are augmented with empirical terms, allowing the models to fit more readily to device data. The SPICE Level-3 model[Liu81] and the BSIM model[She85] are examples of analytic transistor models augmented by empirical terms. The accuracy of a given analytic model is dependent on obtaining the

values of model parameters which produce a "best fit" of the analytic equations to measured data.

Analytic MOS transistor models use separate nonlinear equations to represent each of the transistor's regions of operation. In the case of the SPICE Level-2 model, four different sets of equations are required to represent the four different transistor operating regions(cutoff, subthreshold, saturation and ohmic). The control flow for the evaluation of the Level-2 MOS transistor model is given in Figure 3.1. The evaluation of the MOS transistor's model equations requires the solution of several conditional branches, and the number of computational steps is dependent on the transistor's operating region. The nonlinear equations are, in general, very complicated and require many floating-point computations for their evaluation. Floating-point logarithm, exponential and square-root operations, in addition to floating-point addition, subtraction, multiplication and division, are necessary.



Level 2 MOS Model Evaluation Control Flow
Figure 3.1

3.2.2. Empirical MOS-Transistor Models

Several approaches have been used in the development of empirical MOS transistor models [New78] [SSM82] [BNP83] [BVS83] [BVS84] [Bur84] [BVS86]. Empirical MOS transistor models use discrete values of device current directly in the calculation of the currents and partial derivatives for use by the circuit-simulation program. The discrete values of current can be generated from measurements, analytic models, or device simulation. The current and partial derivative values are interpolated as needed from the discrete data. Empirical models can be compared by the approach in which they store the empirical data and by the method of interpolation they use.

The most direct approach to storing the empirical data is to store the discrete values of I_{ds}, G_{ds}, G_{gs} and G_{sb} from every data point. Since the current and conductances are functions of 3 independent variables(refer to Equations (3.1) through (3.4)), the storage of enough discrete values to represent the device can be quite large [New81]. The amount of data stored can be reduced by taking advantage of the first-order behavior of the MOS transistor [New78] [SSM82] [Bur84]. For example, [Bur84] reduced the data stored to two dimensions, V_{ds} and V_{gse}, where V_{gse} is a function of V_{gs} and V_{sb}. The I_{ds} is represented as

$$I_{ds} = I_{ds}(V_{ds}, V_{gse}). \tag{3.5}$$

To a first order, the V_{sb} voltage effects only the Threshold Voltage (V_t). As demonstrated by the Shichman-Hodges [ShH68] equations, given in Chapter 1, V_{gse} can be written as

$$V_{gse} = V_{gs} - V_t(V_{sb}). \tag{3.6}$$

The dependence on the V_{sb} is embedded in Equation 3.5.

Interpolation between the stored data points is used to generate the specific values of current and conductances from the discrete device data. The most straightforward approach is to linearly interpolate between the discrete data. For example, [SSM82] calculates the current and conductances by linearly interpolating in three dimensions from the discrete current data. Higher-order interpolation methods can also be used, but their

representation of I_{ds} may not always be monotonic. [Bur84] combined quadratic and linear interpolation: quadratic interpolation is used to represent $I_{ds}(V_{ds})$ when the device is in the ohmic region of operation, linear interpolation is used to represent $I_{ds}(V_{ds})$ in the saturation region, and linear interpolation in V_{gs} is used for all voltages.

[BVS83] represents I_{ds} by both triquadratic and tricubic splines. The device conductances, G_{ds}, G_{gs} and G_{sb}, are generated by taking the partial derivatives of the polynomial equations. [BVS83] uses both triquadratic and tricubic splines to represent I_{ds} as a function of V_{ds}, V_{gs} and V_{sb}. A negative aspect of splines is that they are not guaranteed to be monotonic. The spline method has been improved to guarantee monotonic behavior at the expense of altering the discrete values of I_{ds} [BVS84] [BVS86].

The accuracy of an empirical model is primarily dependent on the number of discrete data points and the order of the interpolation method. For a given interpolation method, increasing the number of data points can provide a more accurate representation of the device. In general, a distinct model is stored for each device type with a specific channel length. In most cases an empirical model can be scaled by channel width. The scaling by channel length is avoided because of the device's nonlinear dependence on channel length. Empirical models generally, though not always, require less computation time for their evaluation than analytic models. Empirical models store only the data points, and, thus, they do not contain any process-dependent parameters.

3.2.3. Comparison Between Analytic and Empirical Models

There are advantages and disadvantages to using either an analytic or an empirical MOS transistor model. A comparison of MOS transistor models is generally based upon the accuracy of the models and the time required to evaluate their model equations. A further concern when considering empirical models is the storage and access of the usually large amount of empirical data necessary for each model. There are additional concerns when considering transistor models for use with the MMAP. Both the sensitivity of the model relative to changes in process technology, and the adaptation of the model

to the architecture of the MMAP must be considered.

3.2.3.1. Accuracy, Speed and Storage

Analytical and empirical models can offer comparable accuracy. The accuracy of an analytical model is primarily dependent on the complexity of the model equations. and the accuracy of the empirical model is primarily dependent on the number of data points and the order of the interpolation method.

Model accuracy and model-evaluation speed are closely coupled for analytic transistor models. Analytic models become more complicated as they are changed to improve their accuracy. An increase in the complexity of the transistor model results in an increase in their evaluation time. For example. [BNP83] shows the typical evaluation time of the SPICE Level-2 model to be 16 times that of the SPICE Level-1 model. The amount of data stored also increases with increased model complexity. but the amount of storage is minimal when compared to an empirical model.

The accuracy of an empirical model is dependent on the number of data points stored and the type of interpolation method used. An increase in either the order of the interpolation method or the number of data points can increase the accuracy of the empirical model. More storage is required as the number of data points increases. Also. more storage may be required as the order of the interpolation method increases.

The accuracy of the derivatives of current with respect to voltage is also of concern. [TsM84] demonstrated that large errors in output conductance of the MOS transistor. G_{ds}, can occur even when the current is accurate. The analytic model parameters can be calculated such that accuracy of both current and output conductance are optimized [DAR85]. Empirical models. in general. do not directly address derivative accuracy.

3.2.3.2. Dependence on Process Technology

Analytic transistor models are revised to include physical effects that become prominent as the integrated-circuit process changes. Analytical models augmented with empirical terms are less effected by the changes since their empirical components provide them with more degrees of freedom. Ideally, empirical models depend only on the transistor data. The form of the empirical model does not change, only the new device data must be obtained.

3.2.3.3. Minimizing the Number of Functions and Control Branches

Empirical models require fewer types of numerical functions in their evaluation in comparison to analytic models. Both analytic and empirical models require floating-point addition, subtraction, multiplication and division, but analytic models may also require logarithm, exponential and square-root calculation.

Many empirical models can be executed without any branching, since their evaluation is independent of the transistor's operating region. For example, the empirical models described in [SSM82] and [BVS83] can be evaluated without any branching. As described previously and illustrated by Figure 3.1, analytic models use separate equations to represent the transistor's operation in each of the transistor's operating regions.

3.3. Model Choice

Both analytic and empirical models can accurately represent the I_{ds}. For an analytic model, the model equations can be improved to better represent the device output characteristics. The improved analytic-model equations are usually more complex, resulting in an increase in their computation time. The accuracy of an empirical model can be improved by increasing the density of data points. By increasing the density of data points the error from the interpolation is reduced, but the amount of data stored is increased.

The use of empirical models with electrical circuit simulators has been hindered because of the large amount of memory used in storing the model data. Electrical circuit-simulation programs operating on computers with physical memory-size and address-size constraints are unable to use empirical models because of their large storage requirements. Even computers supporting a large virtual-address space and cache memory are hindered. In the normally "tight-looped" model evaluation, the evaluation may be slowed down by repeated memory accesses since the data for all the empirical models used may not fit concurrently in the computer's cache memory. As described in Chapter 2, the model data is stored in the MMAP during the simulation, and thus the system using the MMAP is not hindered by the empirical model data storage.

Empirical MOS transistor models are generally better adapted for an architecture specifically designed for MOS transistor model evaluation. The empirical model is less sensitive to changes in process technology, requires fewer types of arithmetic and logic operations, and can operate without the execution of conditional branches. The large amount of data that must be stored for an empirical model does not tax the host processor and is therefore no longer a primary concern.

CHAPTER 4

Empirical MOS-Transistor Model Based on Piecewise-Cubic Polynomials

This chapter describes the empirical model that has been developed for use with the MOS-Model Attached Processor(MMAP). Chapter 3 provides an overview of various MOS transistor models, concluding that an empirical model is a suitable choice for the MMAP. The empirical model described in this chapter is based on piecewise-cubic polynomials. This model meets the requirements of the numerical methods used in circuit-simulation programs, while accurately representing the transistor's current and conductances. In addition, the empirical model is evaluated without conditional branching and uses only floating-point addition, subtraction, multiplication and division.

Some first-order MOS transistor characteristics, which have been used in the derivation of the empirical model, are first described. The empirical model is then described. Several practical considerations that must be considered before the model can be used with a circuit-simulation program are described. Finally, several examples of the MOS-transistor model are presented.

4.1. First-Order MOS-Transistor Dependences

The empirical representation of the MOS transistor exploits the first-order behavior of the transistor. The first-order device behavior can be used both to reduce the amount of data stored and to simplify the interpolation method. This section presents the first-order behavior of I_{ds} and the conductances, G_{ds} and G_{gs}. The application of this first-order MOS transistor behavior to the different interpolation methods is described in Section 4.2 of this chapter.

4.1.1. The Dependence of I_{ds} on V_{sb}

As given in Chapter 1, I_{ds} is a function of the V_{ds}, V_{gs} and V_{sb} voltages.

$$I_{ds} = I_{ds}(V_{ds}, V_{gs}, V_{sb}) \tag{4.1}$$

Of the three terminal voltages, the I_{ds} is most "weakly" dependent on the V_{sb}. To a first order, V_{sb} affects only the Threshold Voltage, V_t, and changes in V_t can, to a first order, be represented as a change in V_{gs}.

This dependence has been used to reduce the amount of empirical data stored for each model [New78] [SSM82] [Bur84] [SSM85]. [Bur84] represented I_{ds} as

$$I_{ds} = f(V_{ds}, V_{gse}) \tag{4.2}$$

The effective gate-to-source voltage, V_{gse}, includes the weak dependence of the transistor's current on V_{sb}. As shown in Chapter 1, the Threshold Voltage is the only term in the Shichman-Hodges equations dependent on the V_{sb}. The quantity V_{gse} is defined as

$$V_{gse} = V_{gs} - V_t(V_{sb}). \tag{4.3}$$

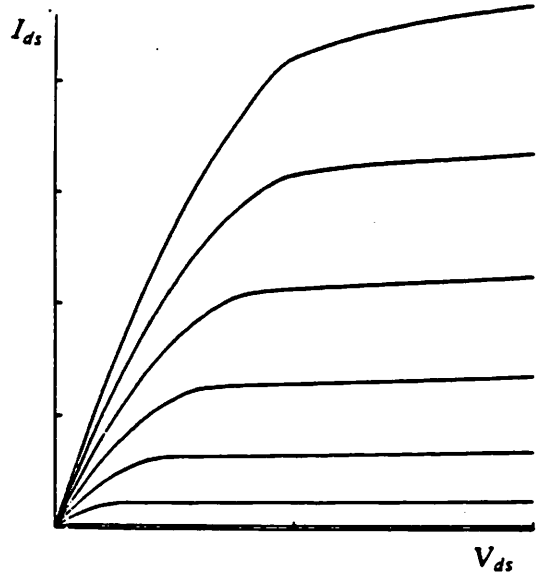
4.1.2. The Dependence of I_{ds} on V_{ds} and V_{gs}

The behavior of the I_{ds} is dependent on the transistor's operating region. Typical I_{ds} vs. V_{ds} and I_{ds} vs. V_{gs} curves are given in Figure 4.1. With respect to V_{ds}, I_{ds} displays a nonlinear behavior when the MOS transistor is operating in the ohmic region, and a linear behavior when the MOS transistor is operating in the saturation region. With respect to V_{gs}, I_{ds} displays a linear behavior when the MOS transistor is operating in the ohmic region, and a nonlinear behavior when the MOS transistor is operating in the saturation region. The behavior of I_{ds} is summarized in Table 4.1.

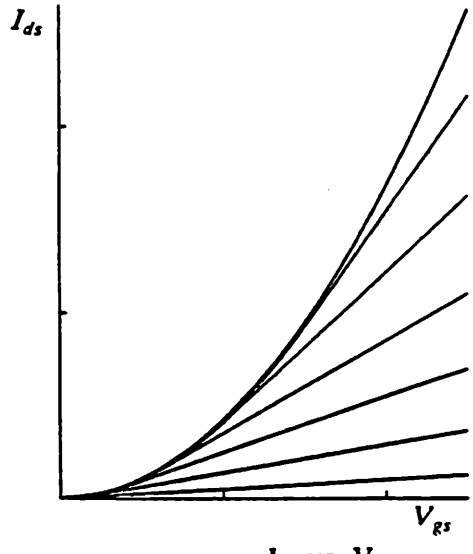
Behavior of I_{ds}		
Differential Voltage	Ohmic	Saturation
V_{ds}	Nonlinear	Linear
V_{gs}	Linear	Nonlinear

Table 4.1

This characteristic behavior has been exploited in many empirical transistor models [New78][Bur84][Sub85]. The form of order of the interpolation method used is dependent on the operating region: For example, in the saturation region only linear interpolation of I_{ds} in V_{ds} is needed, but in the ohmic region a higher-order interpolation and/or a more dense storage of data is required to accurately model the device.



I_{ds} vs. V_{ds}



I_{ds} vs. V_{gs}

Figure 4.1

4.1.3. First-Order Behavior of G_{ds} and G_{gs}

The first-order behavior of the conductances, G_{ds} and G_{gs}, can be derived from the Shichman-Hodges equations. G_{ds} and G_{gs} for a device in the saturation region are

$$G_{ds} = \lambda \frac{\mu C_{ox}}{2} \frac{W}{L} (V_{gs} - V_t)^2 . \tag{4.4}$$

$$G_{gs} = \mu C_{ox} \frac{W}{L} (V_{gs} - V_t)(1 + \lambda V_{ds}). \tag{4.5}$$

In the ohmic region, G_{ds} and G_{gs} are

$$G_{ds} = \mu C_{ox} \frac{W}{L} ((V_{gs} - V_t - V_{ds})(1 + \lambda V_{ds}) + \lambda V_{ds} (V_{gs} - V_t - \frac{V_{ds}}{2})) . \tag{4.6}$$

$$G_{gs} = \mu C_{ox} \frac{W}{L} V_{ds} (1 + \lambda V_{ds}) . \tag{4.7}$$

In the saturation region G_{ds} is independent of V_{ds} and displays a quadratic dependence on V_{gs}, and in the ohmic region G_{ds} displays a quadratic dependence on V_{ds} and a linear dependence on V_{gs}. In general,

$$\lambda << 1 , \tag{4.8}$$

and, with this approximation, G_{ds} displays a linear dependence on both V_{ds} and V_{gs} for the device operating in the ohmic region. The behavior of G_{ds} is summarized in Tables 4.2a and 4.2b.

Behavior of G_{ds}		
Differential Voltage	Ohmic	Saturation
V_{ds}	Quadratic	Constant
V_{gs}	Linear	Quadratic[1]

Table 4.2a

Behavior of G_{ds} $\lambda << 1$		
Differential Voltage	Ohmic	Saturation
V_{ds}	Linear	Constant
V_{gs}	Linear	Quadratic[1]

Table 4.2b

In the saturation region G_{gs} displays a linear dependence on both V_{ds} and V_{gs}, and in the ohmic region G_{gs} displays a quadratic dependence on V_{ds} and is independence of V_{gs}. Using the approximation given by (4.8), G_{gs} displays only a linear dependence on V_{gs} in the saturation region, and a linear dependence on V_{ds} in the ohmic region. The behavior of G_{gs} is summarized in Tables 4.3a and 4.3b.

Behavior of G_{gs}		
Differential Voltage	Ohmic	Saturation
V_{ds}	Quadratic	Constant
V_{gs}	Constant	Quadratic

Table 4.3a

Behavior of G_{gs} $\lambda << 1$		
Differential Voltage	Ohmic	Saturation
V_{ds}	Linear	Constant
V_{gs}	Constant	Linear

Table 4.3b

4.2. Description of the Empirical MOS Model

This section describes the empirical model that has been developed for use with the MMAP. The description is of an N-channel device. The expansion of the model to P-channel devices is presented in Section 4.3. An overview of the empirical method is first given. The representation of the one-dimensional I_{ds} curves is then given. Next, the generation of the family of I_{ds} curves is presented. Two interpolation methods, linear and cubic, are then described. Finally, the function representing the dependence of I_{ds} on V_{sb} is presented. Several examples of the model described in this section are given in Section 4.4.

[1]This behavior of this derivative is observed to be nearly linear for short-channel transistors operating in the saturation region [May86].

4.2.1. Overview of the Empirical Model

As described in Chapter 1. the evaluation of the transistor model is required in circuit-simulation programs. The model evaluation provides the values of I_{ds}. G_{ds}. G_{gs} and G_{sb} for given V_{ds}. V_{gs} and V_{sb} terminal voltages. The current and conductances can be represented as functions of the three terminal voltages. where

$$I_{ds} = F_{I_{ds}}(V_{ds}, V_{gs}, V_{sb}) . \tag{4.9}$$

$$G_{ds} = F_{G_{ds}}(V_{ds}, V_{gs}, V_{sb}) . \tag{4.10}$$

$$G_{gs} = F_{G_{gs}}(V_{ds}, V_{gs}, V_{sb}) . \tag{4.11}$$

$$G_{sb} = F_{G_{sb}}(V_{ds}, V_{gs}, V_{sb}) . \tag{4.12}$$

The basis for the empirical model is 2 families of I_{ds} curves. similar to those illustrated in Figure 4.1. One family represents I_{ds} as a function of V_{ds}. where each curve is for a constant value of V_{gs}. The other family represents I_{ds} as a function of V_{gs}. where each curve is for a constant value of V_{ds}. Both families are derived for a constant value of V_{sb}.

During circuit simulation. operating points arise which do not coincide with either of the two families of curves. The values of current and conductances are calculated by interpolating between curves which bound the operating point. The value of G_{sb} and the dependence of I_{ds}. G_{ds} and G_{gs} on V_{sb} are accounted for by the introduction of V_{gse}. as previously described in Section 4.1. V_{gse} is the actual V_{gs} altered to include a dependence on V_{sb}. The interpolation is performed for $V_{gs} = V_{gse}$.

4.2.2. Piecewise-Cubic Polynomial Equations

The use of piecewise-cubic polynomials to represent the one-dimensional I_{ds} curves is presented in this section.

A single. one-dimensional I_{ds} curve is represented by a piecewise-cubic. polynomial equation. The curves are of either I_{ds} as a function of V_{ds} for a constant V_{gs} or I_{ds} as a function of V_{gs} for constant V_{ds}. For example. I_{ds} as a function of V_{ds} for a constant V_{gs}

is

$$I_{ds} = f_k(V_{ds}) . \quad V_{gs} = V_{gs\,k} . \tag{4.13}$$

The function f_k is valid over the range $[V_{ds\,1}, V_{ds\,J}]$ and is composed of several cubic polynomials, each of which is valid over a unique part of the range.

$$\begin{aligned} I_{ds} &= p_{k\,1}(V_{ds}) . \quad V_{ds\,1} \leqslant V_{ds} < V_{ds\,2} \\ &= p_{k\,2}(V_{ds}) . \quad V_{ds\,2} \leqslant V_{ds} < V_{ds\,3} \\ &\quad . \\ &\quad . \\ &= p_{k\,J-1}(V_{ds}) . \quad V_{ds\,J-1} \leqslant V_{ds} < V_{ds\,J} \end{aligned} \tag{4.14}$$

The cubic polynomial is of the form

$$p_{k\,j} = a_j + b_j \delta V_{ds} + c_j (\delta V_{ds})^2 + d_j (\delta V_{ds})^3 . \tag{4.15}$$

where

$$\delta V_{ds} = V_{ds} - V_{ds\,j} . \tag{4.16}$$

The voltages at which the discrete values of I_{ds} and G_{ds} are known, such as $V_{ds\,j}$, are referred to as *measured* voltages[2]. The polynomial's coefficients, a_j, b_j, c_j, and d_j can be calculated from the values of I_{ds} and G_{ds} at the two end points of the polynomial. The coefficients calculated from the values of currents and conductances at the endpoints are

$$a_j = I_{ds}(V_{ds\,j}) . \tag{4.17}$$

$$b_j = G_{ds}(V_{ds\,j}) . \tag{4.18}$$

$$c_j = \frac{3 \dfrac{I_{ds}(V_{ds\,j+1}) - I_{ds}(V_{ds\,j})}{V_{ds\,j+1} - V_{ds\,j}} - 2\, G_{ds}(V_{ds\,j}) - G_{ds}(V_{ds\,j+1})}{V_{ds\,j+1} - V_{ds\,j}} . \tag{4.19}$$

[2]The term *measured voltages* applies to both drain-to-source and gate-to-source voltages.

$$d_j = \frac{G_{ds}(V_{ds\,j}) + G_{ds}(V_{ds\,j+1}) - 2\,\dfrac{I_{ds}(V_{ds\,j+1}) - I_{ds}(V_{ds\,j})}{V_{ds\,j+1} - V_{ds\,j}}}{(V_{ds\,j+1} - V_{ds\,j})^2} \,. \tag{4.20}$$

where $I_{ds}(V_{ds\,j})$ and $G_{ds}(V_{ds\,j})$ are the current and derivative at $V_{ds\,j}$, and $I_{ds}(V_{ds\,j+1})$ and $G_{ds}(V_{ds\,j+1})$ are the current and derivative at $V_{ds\,j+1}$.

The cubic polynomial given by Equation (4.15) is continuous with continuous derivatives. At the boundary between two adjacent cubic polynomials, the value of I_{ds} and G_{ds} for both polynomials are the same, and their values are equal to the discrete data at the measured voltages used to calculate the polynomial's coefficients. For example, the polynomial $p_{k\,j}$ evaluated at the point $V_{ds\,j}$ is

$$p_{k\,j}(V_{ds\,j}) = I_{ds}(V_{ds\,j}) \,. \tag{4.21}$$

which is equal to the polynomial $p_{k\,j-1}$ evaluated at $V_{ds\,j}$.

$$p_{k\,j-1}(V_{ds\,j}) = I_{ds}(V_{ds\,j}) \,. \tag{4.22}$$

Since the cubic polynomials are continuous and the boundary of adjacent polynomials uses the same data point, the piecewise-cubic equation given by (4.13) is continuous.

The derivative of the piecewise-cubic function is represented by the derivative of the cubic polynomial over the specific range. For example, the derivative of (4.13) is

$$G_{ds} = \frac{d\,f_k}{d\,V_{ds}}(V_{ds}) \,. \quad V_{gs} = V_{gs\,k} \,. \tag{4.23}$$

The function $\frac{d\,f_k}{d\,V_{ds}}$ is valid over the range $[V_{ds\,1}, V_{ds\,J}]$ and is composed of the derivatives of the cubic-polynomial equations, each of which is valid for a specific range.

$$G_{ds} = \frac{d\, p_{k\,1}}{d\, V_{ds}}(V_{ds}) . \quad V_{ds\,1} \leqslant V_{ds} < V_{ds\,2} \tag{4.24}$$

$$= \frac{d\, p_{k\,2}}{d\, V_{ds}}(V_{ds}) . \quad V_{ds\,2} \leqslant V_{ds} < V_{ds\,3}$$

$$= \frac{d\, p_{k\,J-1}}{d\, V_{ds}}(V_{ds}) . \quad V_{ds\,J-1} \leqslant V_{ds} < V_{ds\,J}$$

The derivative of the cubic polynomial is a quadratic polynomial of the form

$$\frac{d\, p_{k\,j}}{V_{ds}} = b_j + 2\, c_j\, \delta V_{ds} + 3\, d_j\, (\delta V_{ds})^2 . \tag{4.25}$$

where δV_{ds} and the coefficients are given in Equations (4.16) through (4.20). At the boundary between two adjacent quadratic polynomials, given by (4.25), the value of G_{ds} is the same for both polynomials, and that value is equal to the value of the discrete derivative for that voltage. For example, the quadratic $\frac{d\, p_{k\,j}}{d\, V_{ds}}$ evaluated at the point $V_{ds\,j}$ is

$$\frac{d\, p_{k\,j}}{V_{ds}}(V_{ds\,j}) = G_{ds}(V_{ds\,j}) . \tag{4.26}$$

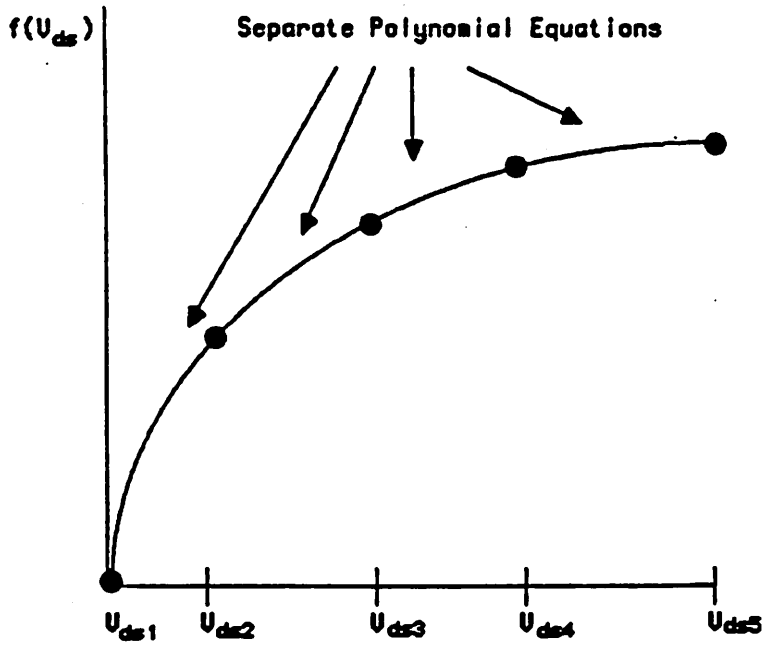
and $\frac{d\, p_{k\,j-1}}{V_{ds}}$ evaluated at $V_{ds\,j}$ is

$$\frac{d\, p_{k\,j-1}}{V_{ds}}(V_{ds\,j}) = G_{ds}(V_{ds\,j}) . \tag{4.27}$$

Since the quadratic polynomials are continuous, and the value of two adjacent polynomials at their common boundary is the same value, the piecewise-cubic equation given by (4.13) is continuous in the first derivative.

An example of a single, piecewise-cubic polynomial is illustrated in Figure 4.2. There are 5 data points, and the piecewise function is composed of 4 cubic polynomials as

shown.



Piecewise-Cubic Polynomial
Figure 4.2

The discrete values of I_{ds} can be obtained from actual device measurements, the results of a device simulation or the solution of analytic model equations [Bur84]. The values of the derivatives at the measured data points, G_{ds} (G_{gs} for curves of I_{ds} as a function of V_{gs}), are calculated using the Enhanced-Monotonic, Piecewise-Cubic Interpolation method (EMPCI)[3]. The values of derivatives are calculated only once and stored for further use.

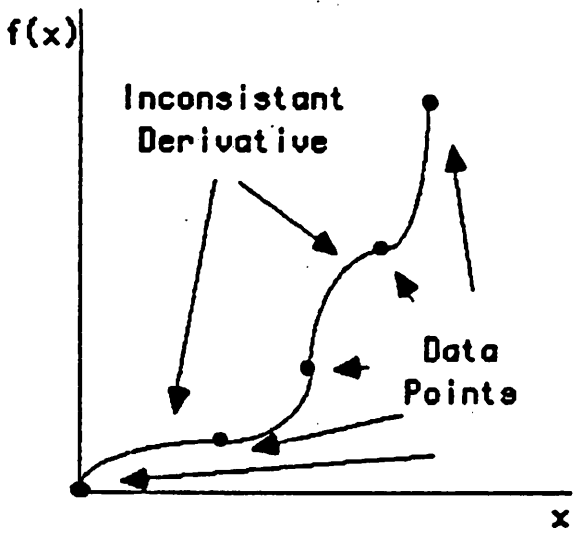
The EMPCI method is similar to a cubic spline in that piecewise-cubic polynomials are fitted to data points. The coefficients of the polynomials comprising the cubic spline

[3]The EMPCI method is presented in detail in Appendix D.

are evaluated based on conditions set on the piecewise-cubic function. For a cubic spline, the piecewise-cubic function and its first two derivatives are everywhere continuous, and the set of derivatives at the data points can be represented by a system of n independent equations in n unknown derivatives. The solution of the system of equations produces unique values of derivatives. The spline provides a smooth fit to the data, however the cubic spline fit to monotone data is not necessarily monotonic.

The EMPCI method is limitted to monotonic data. The EMPCI-derived, piecewise-cubic function is continuous with continuous first derivative. However, unlike the spline method, the EMPCI method does not constrain the second derivative to be continuous. As a result, the system of equations defining the discrete derivatives is underdetermined, resulting in an infinite number of possible values for them. The derivative values calculated by using the EMPCI method ensure that the piecewise-cubic polynomial is monotonic. But, guaranteeing the monotonic behavior of I_{ds} may not accurately represent the behavior of the derivative(G_{ds} and G_{gs}). As illustrated by Figure 4.3, even though the piecewise-cubic polynomial fit through the monotonic data is monotone, the behavior of the derivative does not resemble the discrete data. The incremental slopes of the discrete data are increasing, but the second derivative of the piecewise-cubic function is not always positive. The EMPCI method, in addition to guaranteeing monotonic behavior, also ensures that the shape of the piecewise-cubic polynomial's derivative is correct relative to the discrete data.

The EMPCI method is developed from the Monotone Piecewise Cubic Interpolation[FrC80] method developed by Fritsch et al.. Shima et al. [SYD83] applied Monotone Piecewise Cubic Interpolation to the interpolation of device simulator data for generating data points for use with their table look-up empirical model [SSM82].



Monotone Fit with Inconsistent Derivative
Figure 4.3

4.2.3. Family of Piecewise-Cubic Polynomial Equations

A set of measured data points is used to generate the family of I_{ds} curves, similar to those illustrated in Figure 4.1, where each curve is represented by a piecewise-cubic polynomial. The curves representing I_{ds} as a function of V_{ds} and I_{ds} as a function of V_{gs} are generated from the same measured data. The V_{sb} is constant. The derivatives at the data points for the piecewise-cubic polynomials are derived from using the EMPCI method. The piecewise-cubic polynomials of $I_{ds}(V_{ds})$ intersect the piecewise-cubic polynomials of $I_{ds}(V_{gs})$ at the data points. The values of I_{ds}, G_{ds} and G_{gs} are stored for each data point, where G_{ds} and G_{gs} are the values of derivatives at the endpoints of the cubic polynomials.

As described earlier, the current and partial derivatives at the point (V_{ds}, V_{gs}) are calculated by interpolating between the I_{ds} curves, where each I_{ds} curve is represented by

piecewise-cubic polynomial. As illustrated in Figure 4.4, four piecewise-cubic polynomials.

$$I_{ds} = f_k(V_{ds}) . \quad \text{for } V_{gs} = V_{gs\,k} . \tag{4.28}$$

$$I_{ds} = f_{k+1}(V_{ds}) . \text{ for } V_{gs} = V_{gs\,k+1} . \tag{4.29}$$

$$I_{ds} = g_j(V_{gs}) . \quad \text{for } V_{ds} = V_{ds\,j} . \tag{4.30}$$

$$I_{ds} = g_{j+1}(V_{gs}) . \text{ for } V_{ds} = V_{ds\,j+1} . \tag{4.31}$$

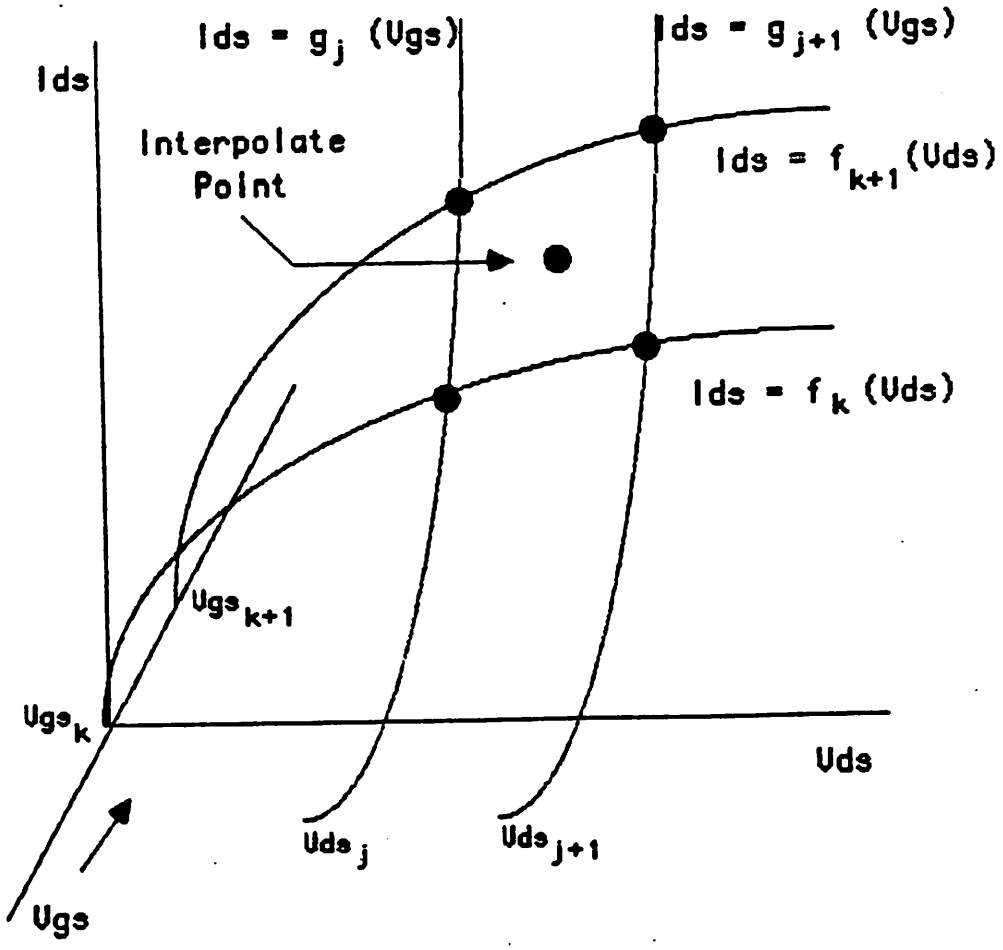
bound the point (V_{ds}, V_{gs}). The value of V_{ds} is

$$V_{ds\,j} \leqslant V_{ds} < V_{ds\,j+1}. \tag{4.32}$$

and the value of V_{gs} is

$$V_{gs\,k} \leqslant V_{gs} < V_{gs\,k+1}. \tag{4.33}$$

A single cubic polynomial from each piecewise-cubic polynomial form a boundary about the point. The four polynomials are calculated from the values of I_{ds}, G_{ds} and G_{gs} at $(V_{ds\,j}, V_{gs\,k})$, $(V_{ds\,j+1}, V_{gs\,k})$, $(V_{ds\,j}, V_{gs\,k+1})$ and $(V_{ds\,j+1}, V_{gs\,k+1})$. The values of current and partial derivatives are calculated by interpolating between the bounding polynomials, as shown in Figure 4.4.



Interpolation Between Piecewise-Cubic Polynomials
Figure 4.4

4.2.4. Linear Interpolation

I_{ds}, G_{ds} and G_{gs} can be calculated by linearly interpolating between adjacent piecewise-cubic equations of I_{ds}. As given by Equations (4.32) and (4.33), V_{ds} and V_{gs} are bounded by the nearest measured voltages. The values of I_{ds}, G_{ds} and G_{gs} at $(V_{ds\,j}, V_{gs\,k})$, $(V_{ds\,j+1}, V_{gs\,k})$, $(V_{ds\,j}, V_{gs\,k+1})$ and $(V_{ds\,j+1}, V_{gs\,k+1})$ are used to calculate the coefficients for the two cubic polynomials of $I_{ds} = f(V_{ds})$ and the two cubic polynomials of $I_{ds} = g(V_{gs})$. The function representing I_{ds}, which is derived from the linear interpolation of piecewise-cubic polynomials, maintains the requirements of monotonicity and derivative behavior that are ensured by the piecewise-cubic polynomials. In addition, I_{ds}, G_{ds} and G_{gs} are continuous.

Calculation of I_{ds}

I_{ds} can be calculated by linearly interpolating between the two polynomials of $I_{ds} = f(V_{ds})$, which is illustrated by Figure 4.5a. First, the values of $I_{ds}(V_{ds}, V_{gs\,k})$ and $I_{ds}(V_{ds}, V_{gs\,k+1})$ are calculated. The value of I_{ds} is then calculated by linearly interpolating in V_{gs}, yielding

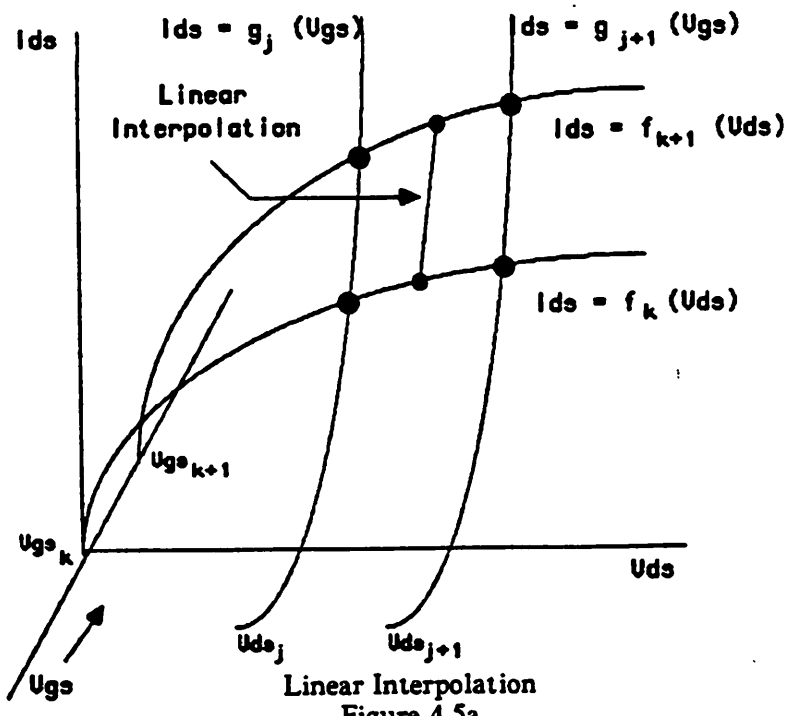
$$I_{ds}(V_{ds}, V_{gs}) = I_{ds}(V_{ds}, V_{gs\,k}) + \left(I_{ds}(V_{ds}, V_{gs\,k+1}) - I_{ds}(V_{ds}, V_{gs\,k}) \right) \frac{V_{gs} - V_{gs\,k}}{V_{gs\,k+1} - V_{gs\,k}}. \tag{4.34}$$

In a similar manner, I_{ds} can also be calculated by linearly interpolating between the two polynomials of $I_{ds} = g(V_{gs})$, which is illustrated by Figure 4.5b. In this case, the values of $I_{ds}(V_{ds\,j}, V_{gs})$ and $I_{ds}(V_{ds\,j+1}, V_{gs})$ are first calculated. The value of I_{ds} is then calculated by linearly interpolating in V_{ds}, producing

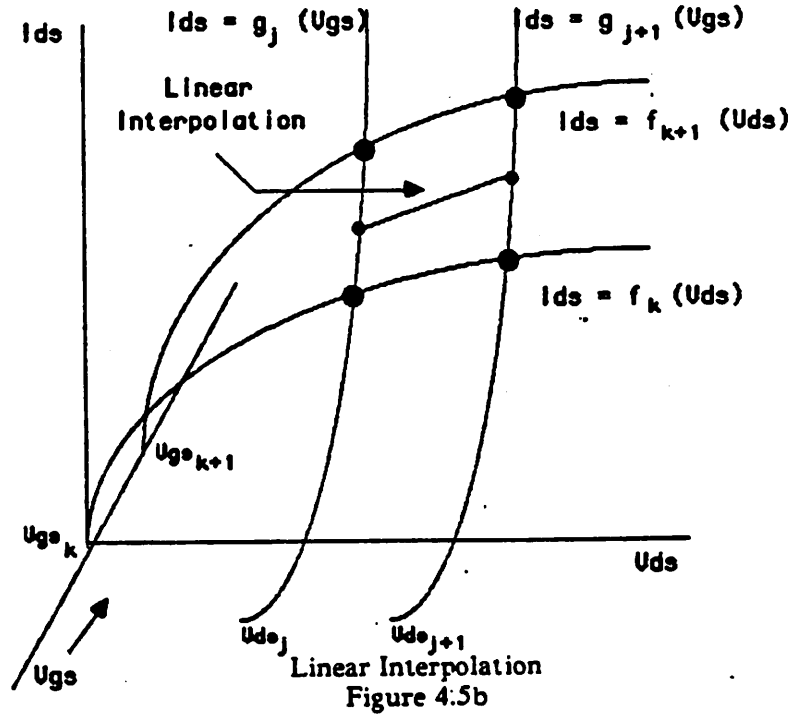
$$I_{ds}(V_{ds}, V_{gs}) = I_{ds}(V_{ds\,j}, V_{gs}) + \left(I_{ds}(V_{ds\,j+1}, V_{gs}) - I_{ds}(V_{ds\,j}, V_{gs}) \right) \frac{V_{ds} - V_{ds\,j}}{V_{ds\,j+1} - V_{ds\,j}}. \tag{4.35}$$

Both (4.34) and (4.35) can be used to calculate I_{ds}. Since the cubic polynomials are monotonic, linear interpolation between them results in a monotonic representation of I_{ds}. As given in Table 4.1, linear interpolation of current in V_{ds} is accurate only if the device is in the saturation region, and linear interpolation in V_{gs} is accurate only in the

ohmic region. Therefore, one representation does not have an advantage over the other.



Linear Interpolation
Figure 4.5a



Linear Interpolation
Figure 4.5b

Continuity of I_{ds}

I_{ds} is required to be continuous for all values of V_{ds} and V_{gs}. The equations representing I_{ds} are continuous, but they are only valid over the range specified by (4.32) and (4.33). The border between two adjacent regions is the cubic polynomial. Since the cubic polynomials are continuous, I_{ds} is continuous for all values of V_{ds} and V_{gs}.

Calculation of G_{ds} and G_{gs}

G_{ds} is calculated by linearly interpolating between the derivatives of the two polynomials of $I_{ds} = f(V_{ds})$. First, the values of $G_{ds}(V_{ds},V_{gs\,k})$ and $G_{ds}(V_{ds},V_{gs\,k+1})$ are calculated. The value of G_{ds} is then calculated by linearly interpolating in V_{gs}, resulting in

$$G_{ds}(V_{ds},V_{gs}) = G_{ds}(V_{ds},V_{gs\,k}) + (G_{ds}(V_{ds},V_{gs\,k+1}) - G_{ds}(V_{ds},V_{gs\,k}))\frac{V_{gs}-V_{gs\,k}}{V_{gs\,k+1}-V_{gs\,k}}. \qquad (4.36)$$

The first-order behavior of G_{ds} is given in Table 4.2. G_{ds} is shown to vary quadratically[4] with respect to V_{gs} when the device is operating in saturation and is otherwise linear. Linear interpolation of G_{ds}, as given by (4.36), does not represent the quadratic dependence of G_{ds} on V_{gs}.

G_{gs} is calculated by linearly interpolating between the derivatives of the two polynomials of $I_{ds} = f(V_{gs})$. First, the values of $G_{gs}(V_{ds\,j},V_{gs})$ and $G_{gs}(V_{ds\,j+1},V_{gs})$ are calculated. The value of G_{gs} is then calculated by linearly interpolating in V_{ds}, resulting in

$$G_{gs}(V_{ds},V_{gs}) = G_{gs}(V_{ds\,j},V_{gs}) + (G_{gs}(V_{ds\,j+1},V_{gs}) - G_{gs}(V_{ds\,j},V_{gs}))\frac{V_{ds}-V_{ds\,j}}{V_{ds\,j+1}-V_{ds\,j}} \qquad (4.37)$$

The first-order behavior of G_{ds} is given in Table 4.3. G_{gs} is shown to be only linearly dependent on V_{ds}. Linear interpolation of G_{gs}, as given by (4.37), does represent the behavior of G_{gs}.

---

[4]Near linear dependence is observed in small-channel devices [May86].

Continuity of G_{ds} and G_{gs}

G_{ds} and G_{gs} are continuous for all V_{ds} and V_{gs}. Equations (4.36) and (4.37) are continuous, but they are only valid over the range specified by Equations (4.32) and (4.33). The border between two adjacent regions is the cubic polynomial. Since the first derivatives of the cubic polynomials are continuous, G_{ds} and G_{gs} are continuous for all values of V_{ds} and V_{gs}.

Summary of Linear Interpolation

Linear interpolation is a straightforward method of representing the MOS transistor's current and partial derivatives as functions of V_{ds} and V_{gs}. The method is executed without any conditional branches. The current obtained from linear interpolation is monotonic, but linear interpolation is not always accurate (refer to Table 4.1). The derivatives produced by linear interpolation are continuous.

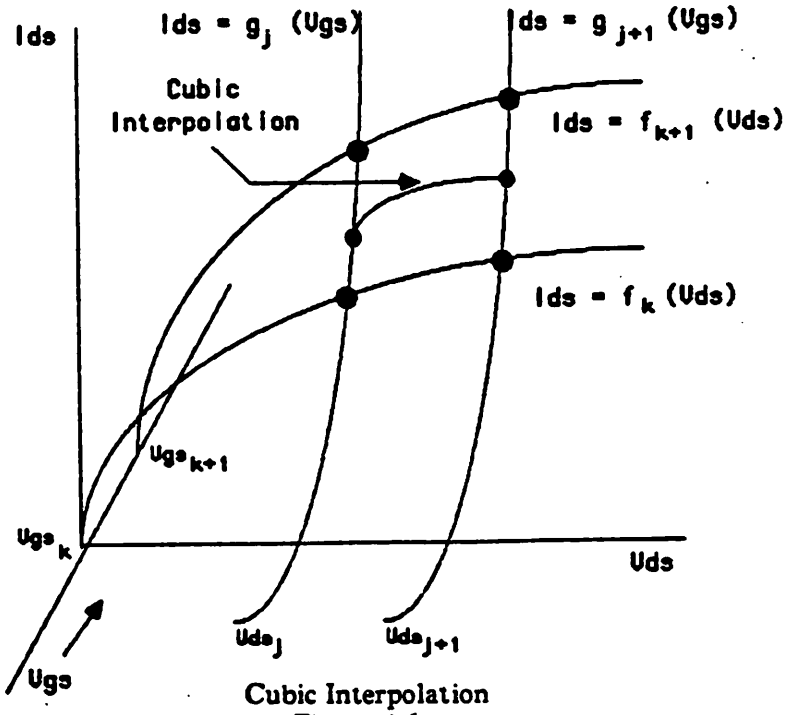
4.2.5. Cubic Interpolation

The use of cubic interpolation between piecewise-cubic polynomials is described in this section.

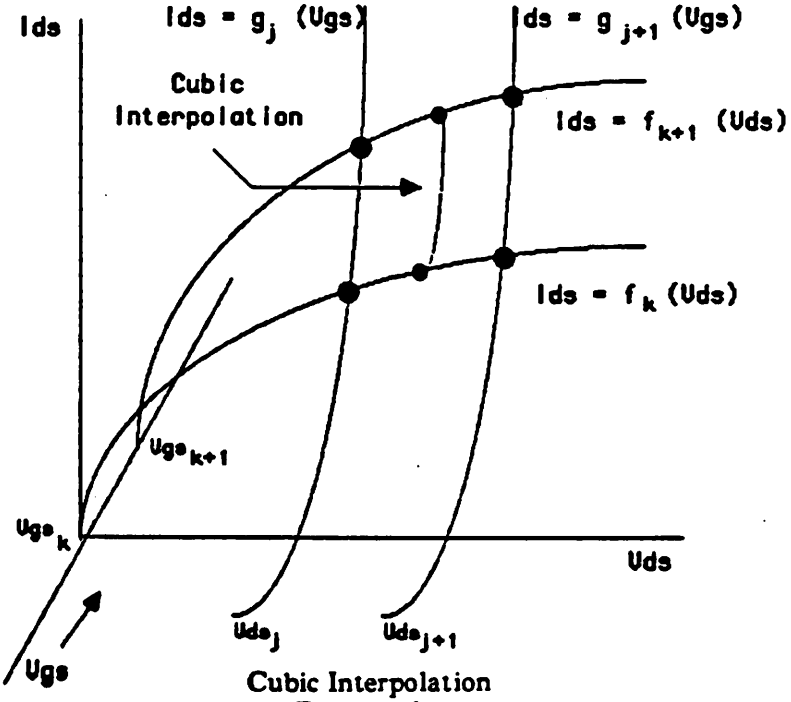
Calculation of I_{ds}

A cubic polynomial, instead of a linear equation, may be used to interpolate between adjacent cubic polynomials. The cubic interpolation, compared to linear interpolation, can better represent the nonlinear behavior of the MOS transistor. Figure 4.6 depicts the two possible ways to perform the cubic interpolation between adjacent cubic polynomials. The first case is depicted in Figure 4.6a. In this case the two polynomials representing I_{ds} as a function of V_{gs} for a constant value of V_{ds} are first solved. The two points are the end points of a cubic polynomial in V_{ds} which interpolates the value of I_{ds}. The second case is depicted in Figure 4.6b. Here the two polynomials representing I_{ds} as a function of V_{ds} for a constant V_{gs} are first solved. The two points are the end points of a cubic polynomial in V_{gs} which interpolates the value of I_{ds}.

A cubic polynomial interpolating between the two data points also requires the values of the derivatives at the end points. For the interpolation illustrated in Figure 4.6a, the values of G_{ds} at the end points are required, and for the interpolation illustrated in Figure 4.6b, the values of G_{gs} at the end points are required. The information in table 4.3 showed that G_{gs} is, to a first order, at most linearly dependent on V_{ds}. As previously stated, representing G_{gs} using linear interpolation of G_{gs} in V_{ds} does represent the behavior of G_{gs}, and, therefore, the interpolation depicted in Figure 4.6b is used.



Cubic Interpolation
Figure 4.6a



Cubic Interpolation
Figure 4.6b

Assume that the values for V_{ds} and V_{gs} are in the subintervals $[V_{ds\,j}, V_{ds\,j+1}]$ and $[V_{gs\,k}, V_{gs\,k+1}]$. The values for I_{ds}, G_{ds} and G_{gs} are known for the data points $(V_{ds\,j}, V_{gs\,k})$, $(V_{ds\,j}, V_{gs\,k+1})$, $(V_{ds\,j+1}, V_{gs\,k})$ and $(V_{ds\,j+1}, V_{gs\,k+1})$, and, they define the polynomials that border the region to be interpolated. As demonstrated in Figure 4.7, I_{ds} and G_{gs} are first obtained at $(V_{ds}, V_{gs\,k})$ and $(V_{ds}, V_{gs\,k+1})$. The values for I_{ds} are obtained by solving the cubic polynomials at V_{ds}. The values for G_{gs} are calculated by linear interpolation.

$$G_{gs}(V_{ds}, V_{gs\,k}) = G_{gs}(V_{ds\,j}, V_{gs\,k}) + (G_{gs}(V_{ds\,j+1}, V_{gs\,k}) - G_{gs}(V_{ds\,j}, V_{gs\,k}))\frac{V_{ds} - V_{ds\,j}}{V_{ds\,j+1} - V_{ds\,j}} \tag{4.38}$$

$$G_{gs}(V_{ds}, V_{gs\,k+1}) = G_{gs}(V_{ds\,j}, V_{gs\,k+1}) + (G_{gs}(V_{ds\,j+1}, V_{gs\,k+1}) - G_{gs}(V_{ds\,j}, V_{gs\,k+1}))\frac{V_{ds} - V_{ds\,j}}{V_{ds\,j+1} - V_{ds\,j}} \tag{4.39}$$

The cubic polynomial representing I_{ds} is

$$I_{ds} = C_0 + C_1\,\delta V_{gs} + C_2\,\delta V^2_{gs} + C_3\,\delta V^3_{gs}, \tag{4.40}$$

where

$$\delta V_{gs} = V_{gs} - V_{gs\,k}. \tag{4.41}$$

The polynomial coefficients are calculated using the values of I_{ds} and G_{gs} at the two end-points, and they are

$$C_0(V_{ds}) = I_{ds}(V_{ds}, V_{gs\,k}), \tag{4.42}$$

$$C_1(V_{ds}) = G_{gs}(V_{ds}, V_{gs\,k}), \tag{4.43}$$

$$C_2(V_{ds}) = 3\frac{I_{ds}(V_{ds}, V_{gs\,k+1}) - I_{ds}(V_{ds}, V_{gs\,k})}{(\Delta V_{gs})^2} - 2\frac{G_{gs}(V_{ds}, V_{gs\,k})}{\Delta V_{gs}} - \frac{G_{gs}(V_{ds}, V_{gs\,k+1})}{\Delta V_{gs}}. \tag{4.44}$$

$$C_3(V_{ds}) = \frac{G_{gs}(V_{ds},V_{gs\,k})}{\Delta V_{gs}} + \frac{G_{gs}(V_{ds},V_{gs\,k+1})}{\Delta V_{gs}} - 2\frac{I_{ds}(V_{ds},V_{gs\,k+1}) - I_{ds}(V_{ds},V_{gs\,k})}{(\Delta V_{gs})^2}. \quad (4.45)$$

$$\Delta V_{gs} = V_{gs\,k+1} - V_{gs\,k}. \quad (4.46)$$

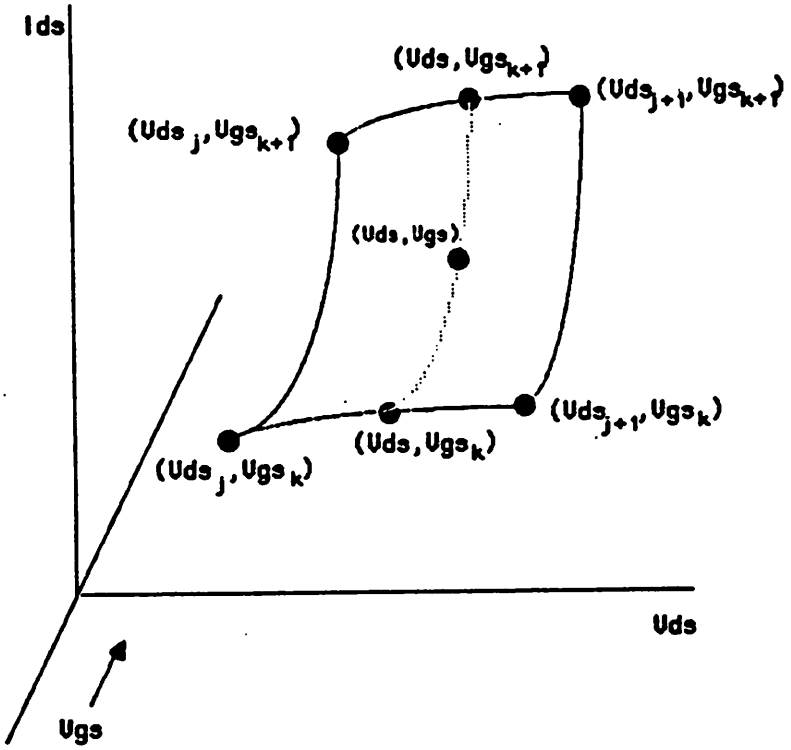


Figure 4.7

Monotonic Behavior of I_{ds}

The condition for the interpolating function to be monotonic is

$$\frac{\partial^2 I_{ds}}{\partial V_{ds}\partial V_{gs}} > 0. \quad (4.47)$$

The second derivative is expanded, yielding a quadratic function in V_{gs}.

$$\frac{d C_1}{d V_{ds}} + 2 \frac{d C_2}{d V_{ds}}\left(\frac{\delta V_{gs}}{\Delta V_{gs}}\right) + 3 \frac{d C_3}{d V_{ds}}\left(\frac{\delta V_{gs}}{\Delta V_{gs}}\right)^2 > 0. \tag{4.48}$$

For the interpolation to be monotonic, the above inequality must be true over the range of $V_{ds} \in [V_{ds\,j}, V_{ds\,j+1}]$, and $V_{gs} \in [V_{gs\,k}, V_{gs\,k+1}]$.

Continuity of I_{ds}

The interpolating function representing I_{ds} is continuous over $V_{ds} \in [V_{ds\,j}, V_{ds\,j+1}]$ and $V_{gs} \in [V_{gs\,k}, V_{gs\,k+1}]$ The function evaluated at a boundary is the cubic polynomial defining that boundary. Therefore, the cubic polynomial that defines the boundary between adjacent areas also provides for a continuous functional representation of I_{ds} over all V_{ds} and V_{gs}.

Calculation of of G_{ds} and G_{gs}

The partial derivatives are calculated directly from I_{ds}. The coefficients C_0, C_1, C_2 and C_3 are functions of V_{ds} only. Taking the partial derivative of I_{ds} with respect to V_{ds} yields

$$G_{ds} = \frac{d C_0}{d V_{ds}} + \frac{d C_1}{d V_{ds}} \delta V_{gs} + \frac{d C_2}{d V_{ds}} \delta V^2_{gs} + \frac{d C_3}{d V_{ds}} \delta V^2_{gs}. \tag{4.49}$$

The partial derivative of I_{ds} with respect to V_{gs} is

$$G_{gs} = C_1 + 2 C_2 \delta V_{gs} + 3 C_3 \delta V^2_{gs}. \tag{4.50}$$

The partial derivative functions are continuous over the range of V_{ds} and V_{gs}.

G_{gs}, represented by Equation (4.50), is continuous at all of the polynomial boundaries. At the boundaries defined by constant V_{gs}, $V_{gs} = V_{gs\,k}$ and $V_{gs} = V_{gs\,k+1}$, G_{gs} is the derivative of the polynomial. At the boundaries defined by constant V_{ds}, $V_{ds} = V_{ds\,j}$ or $V_{ds} = V_{ds\,j+1}$, the derivative is the linear interpolation of the discrete values of G_{gs} which are used in the cubic interpolation.

G_{ds}, represented by Equation (4.49), is continuous only at the boundaries defined by constant V_{gs}, where G_{ds} is defined by the derivative of the polynomial. At the boundary defined by constant V_{ds}, G_{ds} is not continuous, but differs by a constant.

G_{ds} is calculated by linear interpolation, where G_{ds} is

$$G_{ds} = G_{ds}(V_{ds}, V_{gs\,k}) + (G_{ds}(V_{ds}, V_{gs\,k+1}) - G_{ds}(V_{ds}, V_{gs\,k}))\frac{\delta V_{gs}}{\Delta V_{gs}}. \qquad (4.51)$$

which is continuous over all voltage.

4.2.6. Modeling the Source-to-Bulk Voltage Dependence

The empirical interpolation methods described in this chapter represent I_{ds} as a function of V_{ds} and V_{gs} for a constant value of V_{sb}. This relationship can be written as

$$I_{ds} = F_{Int}(V_{ds}, V_{gs}), \quad \text{for } V_{sb} = \text{Constant}. \qquad (4.52)$$

where F_{Int} represents the interpolation of the piecewise-cubic polynomials. During circuit simulation, operating points occur which do not coincide with (4.52). As previously described, I_{ds}'s dependence on V_{sb} is accounted for by augmenting the value of V_{gs}. A general representation of the dependence on V_{sb} is

$$V_{gse} = V_{gs} + F_{V_{sb}}(V_{sb}, V_{ds}). \qquad (4.53)$$

The function $F_{V_{sb}}$ is dependent on V_{ds} and V_{sb}. Equation (4.52) is evaluated using $V_{gs} = V_{gse}$. As previously mentioned, V_{gse} is the effective gate-to-source voltage and is used in the calculation of I_{ds}, G_{ds}, G_{gs} and G_{sb}.

To represent the function $F_{V_{sb}}$, a cubic polynomial is defined for each measured value of V_{ds}.

$$V_{gse\,j} = V_{gs} + A_j + B_j V_{sb} + C_j V^2_{sb} + D_j V^3_{sb}, \quad \text{for } V_{ds} = V_{ds\,j} \qquad (4.54)$$

The coefficients A_j, B_j, C_j and D_j are solved using a least-squares[LaH74][Str76] fit of the data. V_{gse} is calculated by a linear interpolation in V_{ds}.

$$V_{gse} = V_{gse\,j} + (V_{gse\,j+1} - V_{gse\,j}) \frac{V_{ds} - V_{ds\,j}}{V_{ds\,j+1} - V_{ds\,j}} \tag{4.55}$$

Equation (4.52) evaluated at $V_{ds} = V_{ds\,j}$, where $V_{ds\,j}$ is a measured voltage, is the piecewise-cubic equations of I_{ds} as a function of V_{gs} for $V_{ds} = V_{ds\,j}$. The I_{ds} can be written as

$$I_{ds} = g_j(V_{gs}) . \quad \text{for } V_{ds} = V_{ds\,j} . \tag{4.56}$$

where g_j is the piecewise-cubic polynomial equation for $V_{ds} = V_{ds\,j}$. Since Equation (4.52) is evaluated at $V_{gs} = V_{gse}$, Equation (4.56) is evaluated at $V_{gs} = V_{gse\,j}$.

$$I_{ds} = g_j(V_{gse\,j}) . \quad \text{for } V_{ds} = V_{ds\,j} . \tag{4.57}$$

Equation (4.57) can also be represented as

$$V_{gse\,j} = g^{-1}{}_j(I_{ds}) . \quad \text{for } V_{ds} = V_{ds\,j} . \tag{4.58}$$

where g^{-1} is the inverse of function g. As given by Equation (4.9), I_{ds} can be written in terms of its three independent terminal voltages. Substituting Equation (4.9) into Equation (4.58) and setting $V_{ds} = V_{ds\,j}$, $V_{gse\,j}$ is

$$V_{gse\,j} = g^{-1}{}_j(F_{I_{ds}}(V_{ds\,j}, V_{gs}, V_{sb})) . \tag{4.59}$$

The combination of Equation (4.54) and Equation (4.59) is

$$V_{gs} + A_j + B_j V_{sb} + C_j V^2{}_{sb} + D_j V^3{}_{sb} = g^{-1}{}_j(F_{I_{ds}}(V_{ds\,j}, V_{gs}, V_{sb})) , \quad \text{for } V_{ds} = V_{ds\,j} \tag{4.60}$$

Equation (4.60) can then be used to solve for A_j, B_j, C_j, and D_j. Given I_{ds} data for several different combinations of V_{gs} and V_{sb} at $V_{ds} = V_{ds\,j}$, the coefficients can be solved using a least-squares fitting of data. For M combinations of V_{gs} and V_{sb}, the matrix equation is

$$\begin{bmatrix} 1 & V_{sb\,0} & V^2_{sb\,0} & V^3_{sb\,0} \\ 1 & V_{sb\,1} & V^2_{sb\,1} & V^3_{sb\,1} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 1 & V_{sb\,M} & V^2_{sb\,M} & V^3_{sb\,M} \end{bmatrix} \begin{bmatrix} A_j \\ B_j \\ C_j \\ D_j \end{bmatrix} + \tag{4.61}$$

$$\begin{bmatrix} V_{gs\,0} - g^{-1}(F_{ids}(V_{ds\,j}, V_{gs\,0}, V_{sb\,0})) \\ V_{gs\,1} - g^{-1}(F_{ids}(V_{ds\,j}, V_{gs\,1}, V_{sb\,1})) \\ \vdots \\ V_{gs\,M} - g^{-1}(F_{ids}(V_{ds\,j}, V_{gs\,M}, V_{sb\,M})) \end{bmatrix} = 0$$

Equation (4.61) is then in the form to solve for the coefficients, and must be applied for all measured V_{ds}.

The calculation of the partial derivatives include V_{gse}'s dependence on the differential terminal voltages.

$$G_{ds} = \frac{\partial I_{ds}}{\partial V_{ds}} + \frac{\partial I_{ds}}{\partial V_{gse}} \frac{\partial V_{gse}}{\partial V_{ds}} \tag{4.62}$$

$$G_{gs} = \frac{\partial I_{ds}}{\partial V_{gse}} \tag{4.63}$$

$$G_{sb} = \frac{\partial I_{ds}}{\partial V_{gse}} \frac{\partial V_{gse}}{\partial V_{sb}} \tag{4.64}$$

4.3. Empirical Model - Practical Considerations

Several practical considerations must be addressed before the empirical model can be used with a circuit simulation program. The description of the empirical model given in the previous section only addressed a N-channel MOS transistor with specific channel dimensions. In this section, the modeling of P-channel enhancement transistors using the empirical model is first described. Then, the scaling of currents and conductances of model transistors with channel dimensions different from the stored model is presented. The "out-of-range" evaluation of a transistor operating point which is outside of the limit defined by the families of curves is described next. The issue of numerical precision

necessary for the model evaluation is then presented. Finally, the data storage requirements for the polynomial-based model are summarized.

4.3.1. P-Channel Transistors

The description of the empirical model given in Section 4.2 is based on an n-channel enhancement transistor. P-channel transistors are evaluated in the same manner as N-channel transistors. As a circuit device, the two transistor types operate the same, except that the the signs of the V_{ds}, the V_{gs}, the V_{sb} and the I_{ds} of the P-channel transistor are opposite those of the N-channel transistor. For the P-channel transistor, the V_{ds} is negative and the I_{ds} becomes increasingly negative as the V_{gs} becomes more negative.

The empirical data of the P-channel device are generated in the same manner as an N-channel transistor by changing the sign of the I_{ds}, the V_{ds}, the V_{gs} and the V_{sb} from the P-channel data to correspond to an N-channel transistor. The data are then stored as if the P-channel transistor is an N-channel transistor. When a P-channel transistor is evaluated, the signs of the voltages are changed and the evaluation continues as if it were an N-channel device. The current and conductances calculated correspond in sign to an N-channel device, and the signs of the current and conductances are changed to represent the P-channel transistor.

4.3.2. Scaling of Channel Dimensions

Data for an empirical MOS transistor model are derived from a MOS transistor with specific channel dimensions. The data for several empirical models of different dimensions are stored. To model transistors with channel dimensions that differ from those of the measured measured device, the current and conductances are multiplied by the scale factor S_c.

$$I_{ds} = S_c \times I_{ds} \tag{4.65}$$

$$G_{ds} = S_c \times G_{ds} \tag{4.66}$$

$$G_{gs} = S_c \times G_{gs} \tag{4.67}$$

$$G_{sb} = S_c \times G_{sb} \tag{4.68}$$

The scaling of currents and derivatives in this manner is valid for variations in channel width and, in some cases, for small changes in channel length.

4.3.3. Out-of-Range Evaluation

The family of piecewise-cubic polynomials used in both linear and cubic interpolation represent the normal operating range of the transistor. However, a transistor evaluation outside of the range defined by the polynomials may be required by the circuit-simulation program. As shown in Figure 4.8, the two families of polynomials are valid for

$$V_{ds\,min} \leqslant V_{ds} < V_{ds\,max}. \tag{4.69}$$

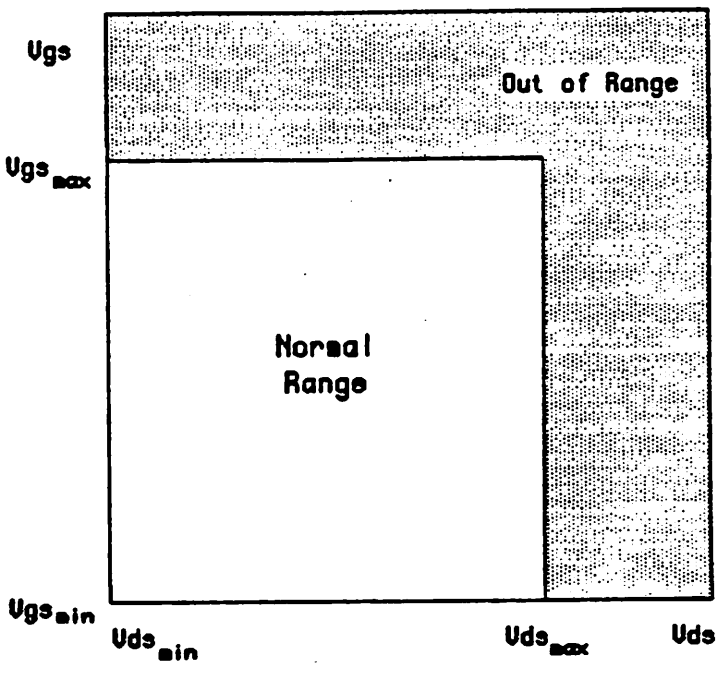
$$V_{gs\,min} \leqslant V_{gs} < V_{gs\,max}. \tag{4.70}$$

In this "out-of-range" case, one or both of the variables V_{ds} and V_{gs} are outside of the range defined by the cubic polynomials. The "out-of-range" case is solved by calculating the function at the nearest boundary point. Therefore, V_{ds} and V_{gs} are represented by the following two equations.

$$V_{ds} = \mathrm{Min}(V_{ds\,max}, \mathrm{Max}(V_{ds}, V_{ds\,min})) \tag{4.71}$$

$$V_{gs} = \mathrm{Min}(V_{gs\,max}, \mathrm{Max}(V_{gs}, V_{gs\,min})) \tag{4.72}$$

and the same evaluation as is required in the "in-range" case is performed.



Out-of-Range
Figure 4.8

4.3.4. Numerical Precision

Single precision and double precision are the two accepted floating-point word sizes. As defined by the IEEE. the single-precision word size is 32 bits including a 24-bit mantissa. and the double-precision word size is 64 bits including a 54-bit mantissa. Double-precision operations require a greater amount of time to compute and/or more hardware to implement compared to single precision.

The calculated value of I_{ds} may include an error due to the finite precision of the floating-point calculation. I_{ds} can be written as

$$I_{ds} = I_{ds\,\infty} + \delta I_{ds}. \tag{4.73}$$

where $I_{ds\,\infty}$ is the answer computed with infinite precision and δI_{ds} is the error due to the inaccuracy of the mantissa as a result of the calculation. Provided the order in which the

calculation occurs does not change. the error is constant for the same inputs. δI_{ds} can effect an electrical simulation program in two ways-as an error in the simulation results and as a discontinuity in the representation of I_{ds}.

δI_{ds} can be directly reflected as an error in the simulation results. If the linearized-circuit equations are calculated with infinite precision. the fraction of error in the circuit variables is of the same order as the fraction of error in I_{ds}. But. δI_{ds} is generally a small fraction of the actual value. A floating-point number whose mantissa is accurate to only 12 bits still has less than 0.025% percent error. Based on this concern. single-precision. floating-point arithmetic is adequate for computing I_{ds}.

δI_{ds} can produce a discontinuity in I_{ds} at the boundaries of polynomials. Even though the error due to the discontinuity is only a small fraction of the actual value. a discontinuity in I_{ds} can prohibit the circuit from being simulated. To illustrate the cause of the discontinuity. consider the example shown in Figure 4.9. The function is composed of two cubic polynomials. p_0 and p_1. where

$$p_0(x) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 \quad x_0 \leq x < x_1 \qquad (4.74)$$

and

$$p_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 \quad x_1 \leq x < x_2. \qquad (4.75)$$

With infinite-precision calculation the composite function is continuous at x_1. the boundary between the two polynomials.

$$p_0(x_1) = p_1(x_1) = a_1 \qquad (4.76)$$

With finite precision calculation. the piecewise function can be discontinuous at x_1. $p_1(x_1)$ remains equal to a_1. but $p_0(x_1)$ contains an error due to the finite precision of its computation. The piecewise function is then discontinuous at x_1. As shown in Appendix D. the piecewise-cubic function is always continuous provided

$$(x_1 - x_0) \times (x_1 - x_0)^{-1} = 1 \qquad (4.77)$$

is exact. By applying this to the calculation of I_{ds}. the measured values of V_{ds} and V_{gs}

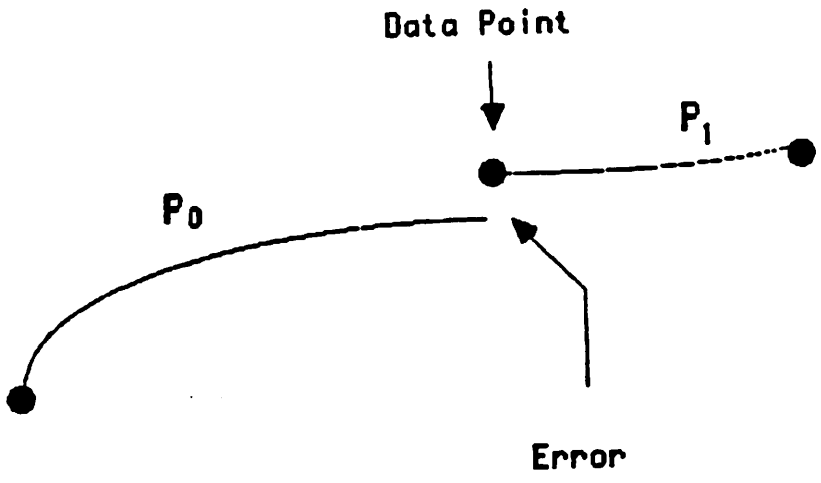
are restricted such that

$$(V_{ds\,j+1} - V_{ds\,j}) \times (V_{ds\,j+1} - V_{ds\,j})^{-1} = 1 \tag{4.78}$$

and

$$(V_{gs\,k+1} - V_{gs\,k}) \times (V_{gs\,k+1} - V_{gs\,k})^{-1} = 1 \tag{4.79}$$

are exact. If Equations (4.78) and (4.79) are met using single-precision calculation. I_{ds} can be calculated using only single-precision, floating-point arithmetic.



Piecewise-Cubic Polynomial Discontinuity
Figure 4.9

4.3.5. Data Storage

The amount of data that is stored for each empirical model is dependent on the number of measured voltages. I_{ds}, G_{ds} and G_{gs} must be stored for all of the combinations of V_{ds} and V_{gs} measured voltages. The V_{sb}-dependence coefficients are also stored, where there are 4 coefficients for each measured V_{ds} voltage. In addition, the measured voltages must also be stored. If J is equal to the number of V_{ds} measured voltages, and K is equal to the number of V_{gs} measured voltages, then the amount of data that must be stored is

equal to

$$3JK + 5J + K. \tag{4.80}$$

Typically J and K will both be greater than 10. and thus the first term in Equation (4.80) dominates. For J = K = 10. 360 floating-point numbers must be stored.

There are only two division operations required by the transistor evaluation for both interpolation methods. In both cases. the inverse of the difference between adjacent measured voltages is calculated. These values have been represented as

$$\frac{1}{V_{ds\,j+1} - V_{ds\,j}} \tag{4.81}$$

and

$$\frac{1}{V_{gs\,k+1} - V_{gs\,k}} \tag{4.82}$$

in the preceding discussions. For both general-purpose processing units and specifically designed floating-point units. floating-point division typically requires a larger amount of computation time in comparison to floating-point addition. subtraction and multiplication [Man82]. For example. the Intel 8087 Numeric Data Processor chip requires 41μs to perform a floating-point division. compared to 28μs for multiplication and 18μs for addition [Sta83]. and the Cray-1 supercomputer requires 14 clock cycles to perform an inversion[5]. compared to 7 cycles for a multiplication and 6 for an addition.

The inverse of the difference between adjacent measured voltages is precalculated and stored with the other transistor model information instead of being calculated during the transistor evaluation. An additional J-1 and K-1 terms are stored. and including these terms. the amount of data stored is

$$3JK + 6J + 2K - 2. \tag{4.83}$$

[5]The Cray-1 does not directly support division. The Cray-1 inverts the divisor and then multiplies that result with the dividend [SBN82].

The first term still remains dominant. For use with the MMAP, the primary advantage is not speed, since there are only two division operations. The primary advantage is that the MMAP does not need to support the division operation, thus simplifying the design of the MMAP. For J = K = 10, the amount of storage increases by 5% to 378 floating-point numbers.

4.4. Examples

Three examples of the empirical model are given in this section. The operating-point data for the first two examples are generated from the Shichman-Hodges and SPICE Level-2 transistor model respectively. The data for the third example is from a measured device.

The empirical model parameters are generated by the POLY_MOS program, which is listed in Appendix E. The input to the program are the discrete current data. The program calculates the values of G_{ds} and G_{gs} to produce the family of curves, and the program also calculates the V_{sb}-dependence parameters. The data for the three examples are given in Appendix F.

4.4.1. Example 1: Data Derived from the Shichman-Hodges Model

The I_{ds} current data used in this example are generated from the Shichman-Hodges(SH) model. The model parameters and current data are listed in Appendix F.

The two families of I-V curves are shown in Figures 4.10 and 4.11. In both figures, the solid lines are the piecewise-cubic polynomials, and the dotted lines are generated from the SH model. The family of curves represented by piecewise-cubic polynomials coincides with the corresponding SH curves. The difference is only visible for the curve of I_{ds} as a function of V_{ds} for $V_{gs} = 6.0$.

Linear interpolation between two curves of constant V_{gs} is given in Figure 4.12. The four curves, $V_{gs} = 3.6$, $V_{gs} = 3.7$, $V_{gs} = 3.8$, and $V_{gs} = 3.9$, are derived by linearly

interpolating between the piecewise-cubic curves at V_{gs} = 3.5 and V_{gs} = 4.0. The dotted lines are the output of the SH model. The difference between the SH curves and the linearly interpolated curves is a result of the linear interpolation. The linear interpolation between curves of constant V_{gs} provides for only a linear dependence on V_{gs}, while, as given by the SH equations, the SH model is quadratically dependent on V_{gs} in the saturation region.

Figure 4.13 displays the results of applying cubic interpolation between the curves, V_{gs} = 3.5 and V_{gs} = 4.0. The cubic interpolation reproduces the behavior of the SH model, with the only variation evident at the border between the saturation and ohmic regions of operation. The agreement between the result of cubic interpolation and the SH model is expected since the basis of the cubic-interpolation method is derived from first-order device behavior represented by the SH equations.

4.4.2. Example 2: Data Derived from the SPICE Level-2 Model

The I_{ds} current data used in this example is generated from the SPICE Level-2 model, and the model parameters and current data are listed in Appendix F.

The two families of I-V curves are shown in Figures 4.14 and 4.15. In both figures, the solid lines are the piecewise-cubic polynomials, and the dotted lines are generated from the Level-2 model. The family of curves represented by piecewise-cubic polynomials coincide with the corresponding Level-2 curves.

Linear interpolation between two curves of constant V_{gs} is given in Figure 4.16. The four curves, V_{gs} = 3.6, V_{gs} = 3.7, V_{gs} = 3.8, and V_{gs} = 3.9, are derived by linearly interpolating between the piecewise-cubic curves at V_{gs} = 3.5 and V_{gs} = 4.0. The dotted lines are the output of the Level-2 model. As with the SH model given in the previous example, the difference between the SH curves and the linearly interpolated curves is a result of the linear interpolation. The linear interpolation between curves of constant V_{gs} provides for only a linear dependence on V_{gs}, while the Level-2 equations demonstrate a

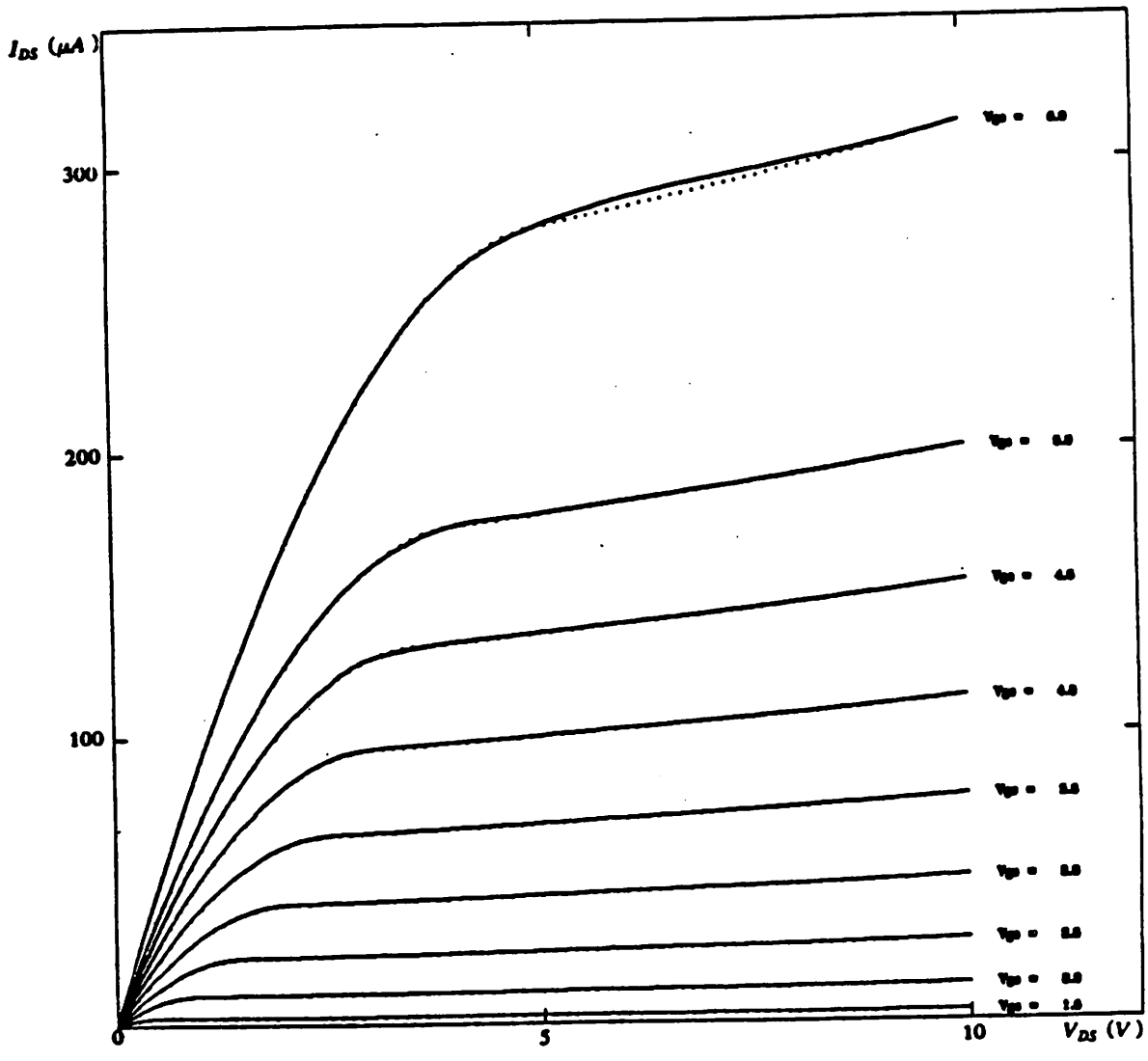
nonlinear (quadratic to a first order) variation in I_{ds} due to varying V_{gs} in the saturation region.

Figure 4.17 displays the results of applying cubic interpolation between the curves. V_{gs} = 3.5 and V_{gs} = 4.0. The cubic interpolation reproduces the behavior of the Level-2 model, with the only variation evident at the border between the saturation and ohmic regions of operation.

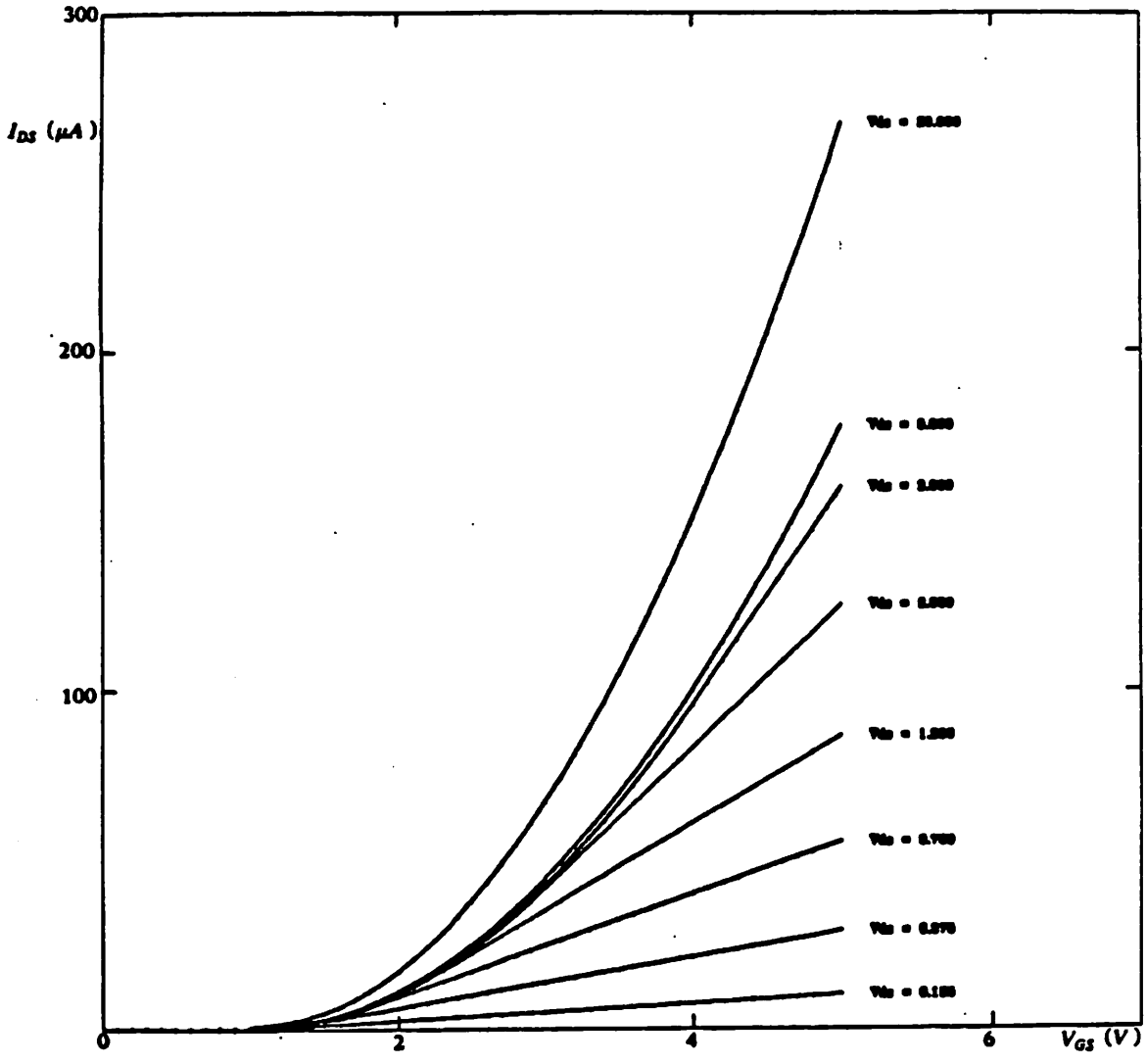
4.4.3. Example 3: Data Generated from Device Measurement

The I_{ds} current data used in this example is from measurement of an N-channel MOS transistor with a channel width of 1.4 microns and a length of 1.4 microns [Dec84]. The I_{ds} data for this example are listed in Appendix F.

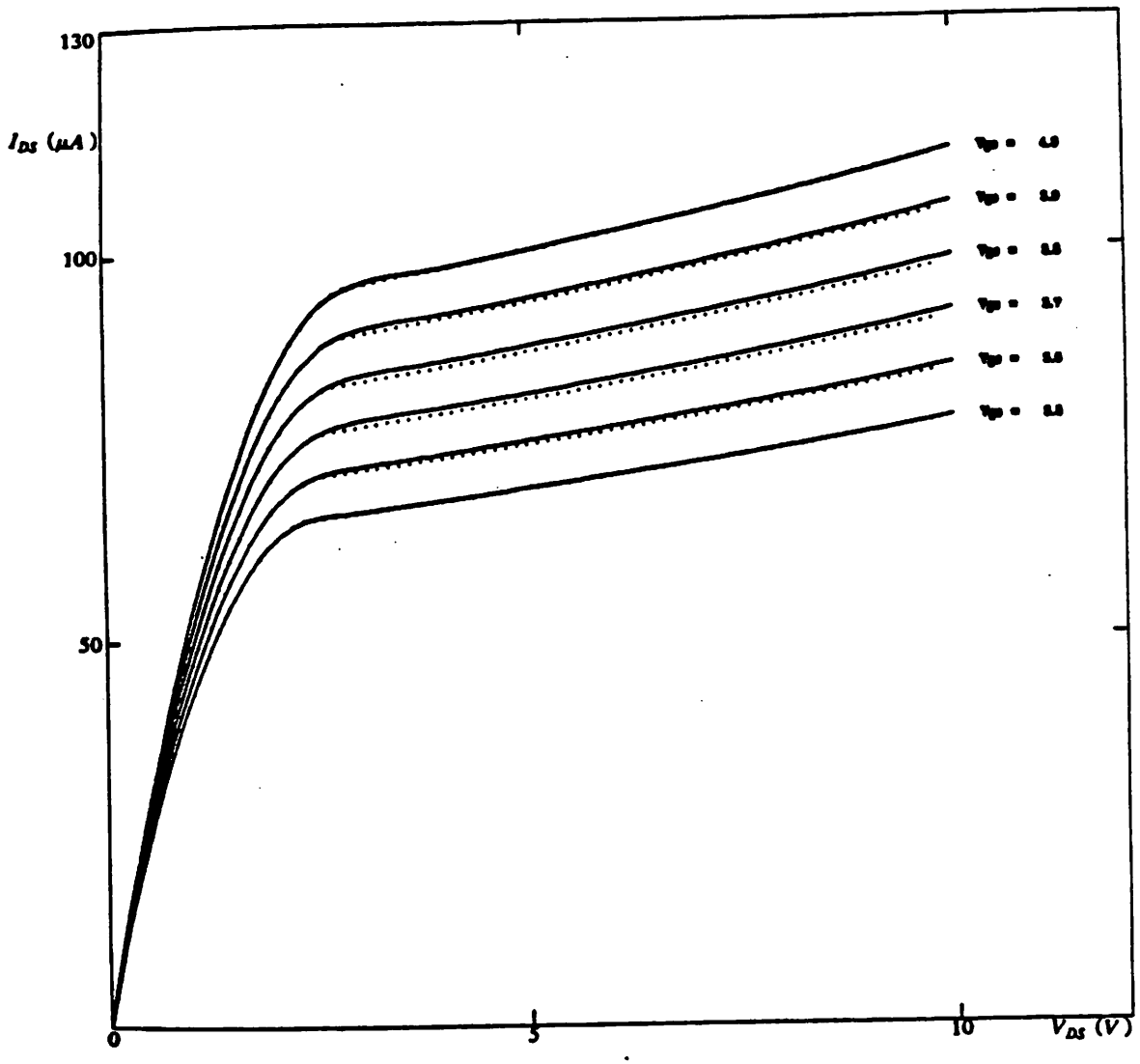
The application of linear interpolation to the measured data is illustrated in Figure 4.18, and the application of cubic interpolation to the measured data is illustrated in Figure 4.19. The curves for V_{gs} = 2.0, 3.0 and 5.0 volts comprise the family of piecewise-cubic polynomial curves of I_{ds} as a function of V_{ds}. The fourth curve, V_{gs} = 4.0, is derived from interpolating between the curves V_{gs} = 3.0 and V_{gs} = 5.0.



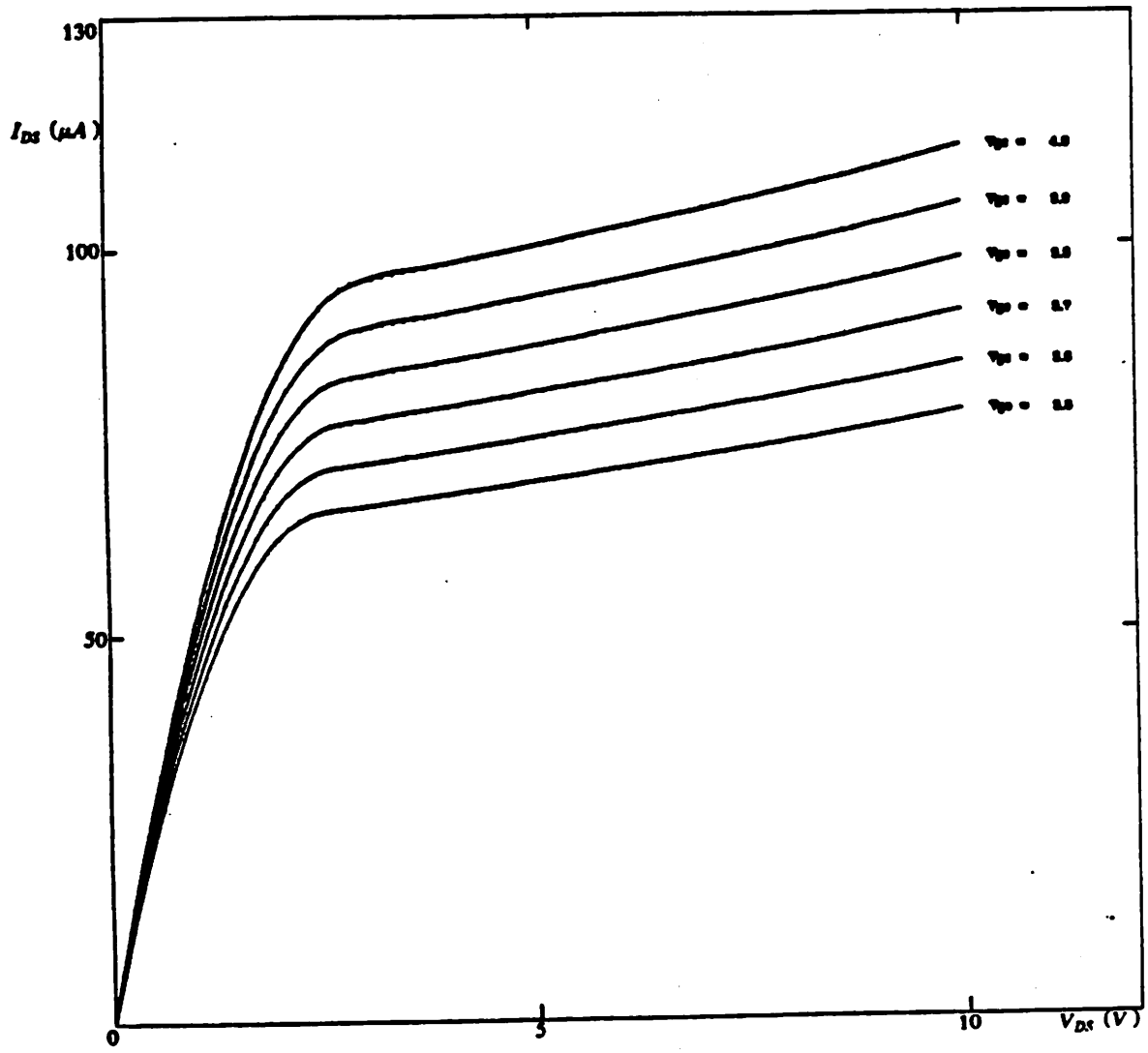
Example 1: Family of I_{ds} vs. V_{ds}
Figure 4.10



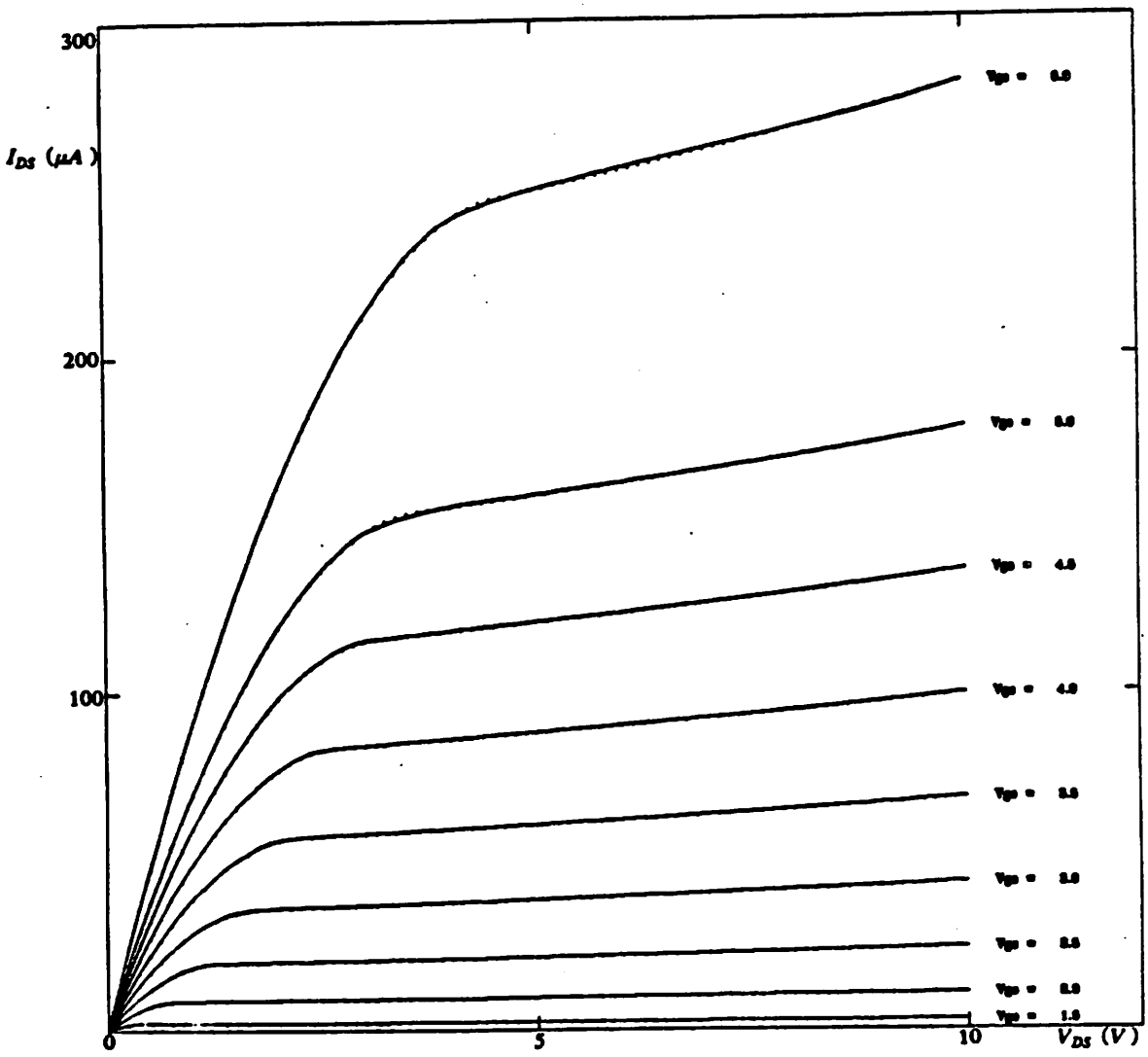
Example 1: Family of I_{ds} vs. V_{gs} Curves
Figure 4.11



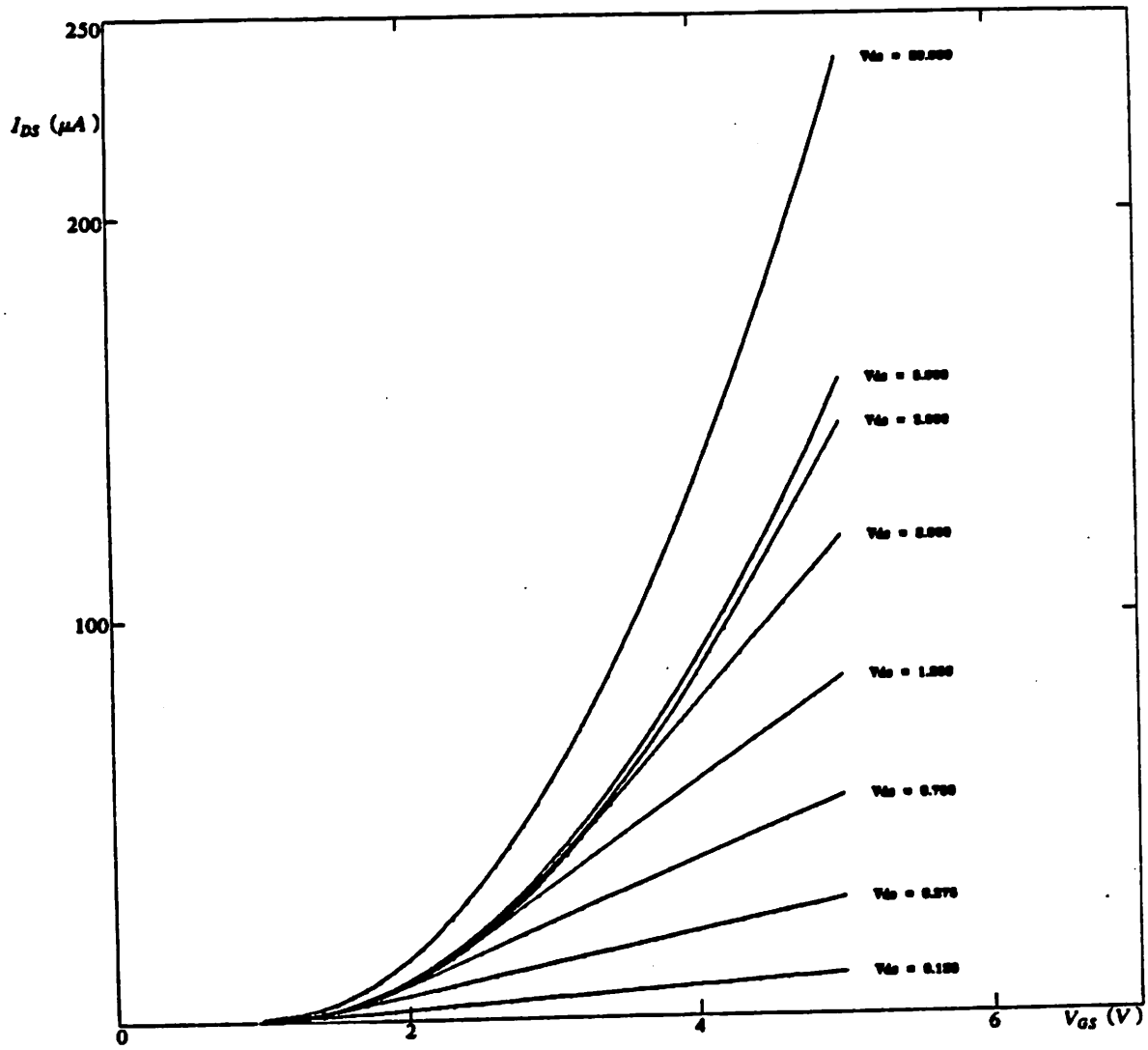
Example 1: Linear Interpolation - I_{ds} vs. V_{ds}
Figure 4.12



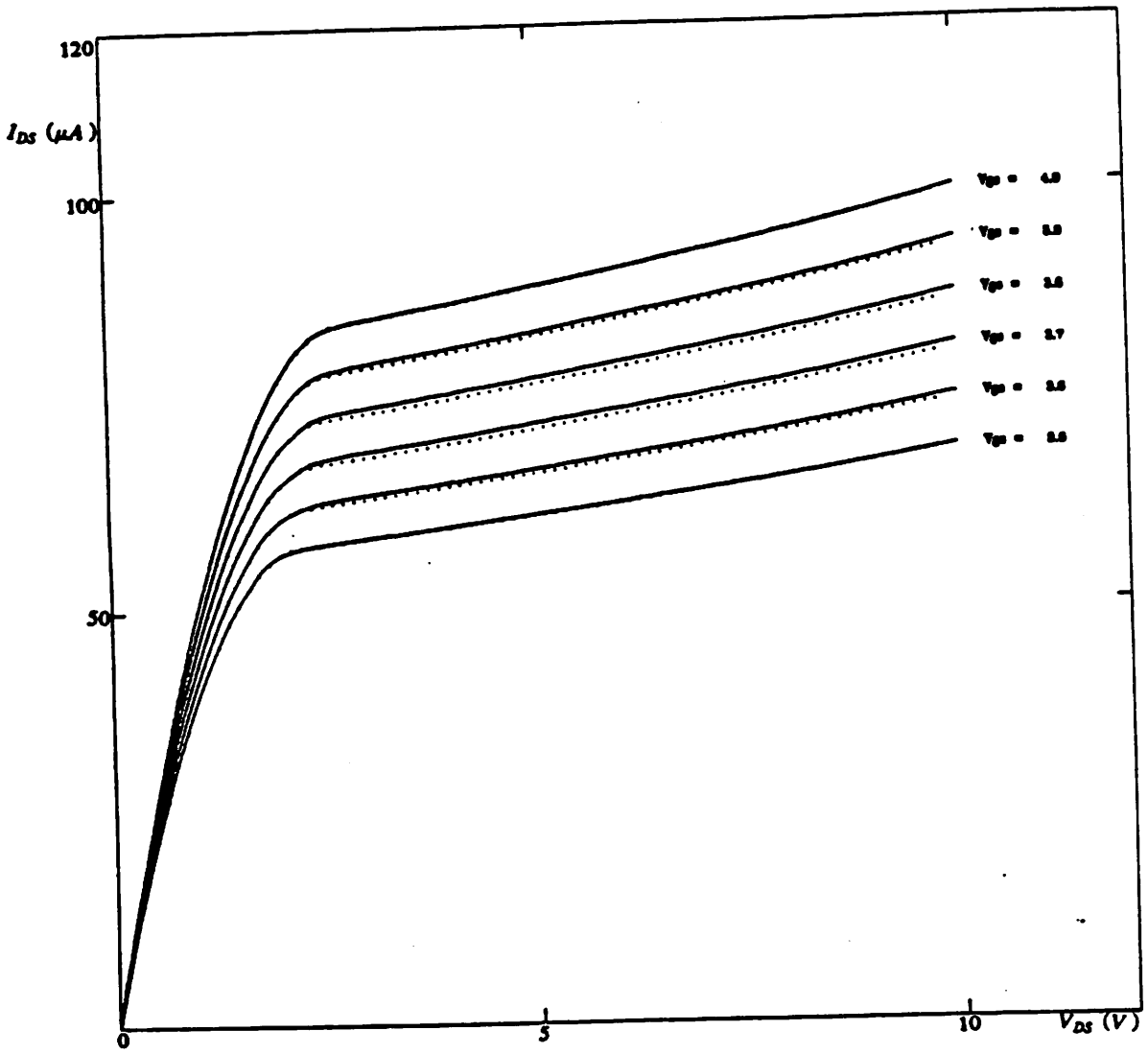
Example 1: Cubic Interpolation - I_{ds} vs. V_{ds}
Figure 4.13



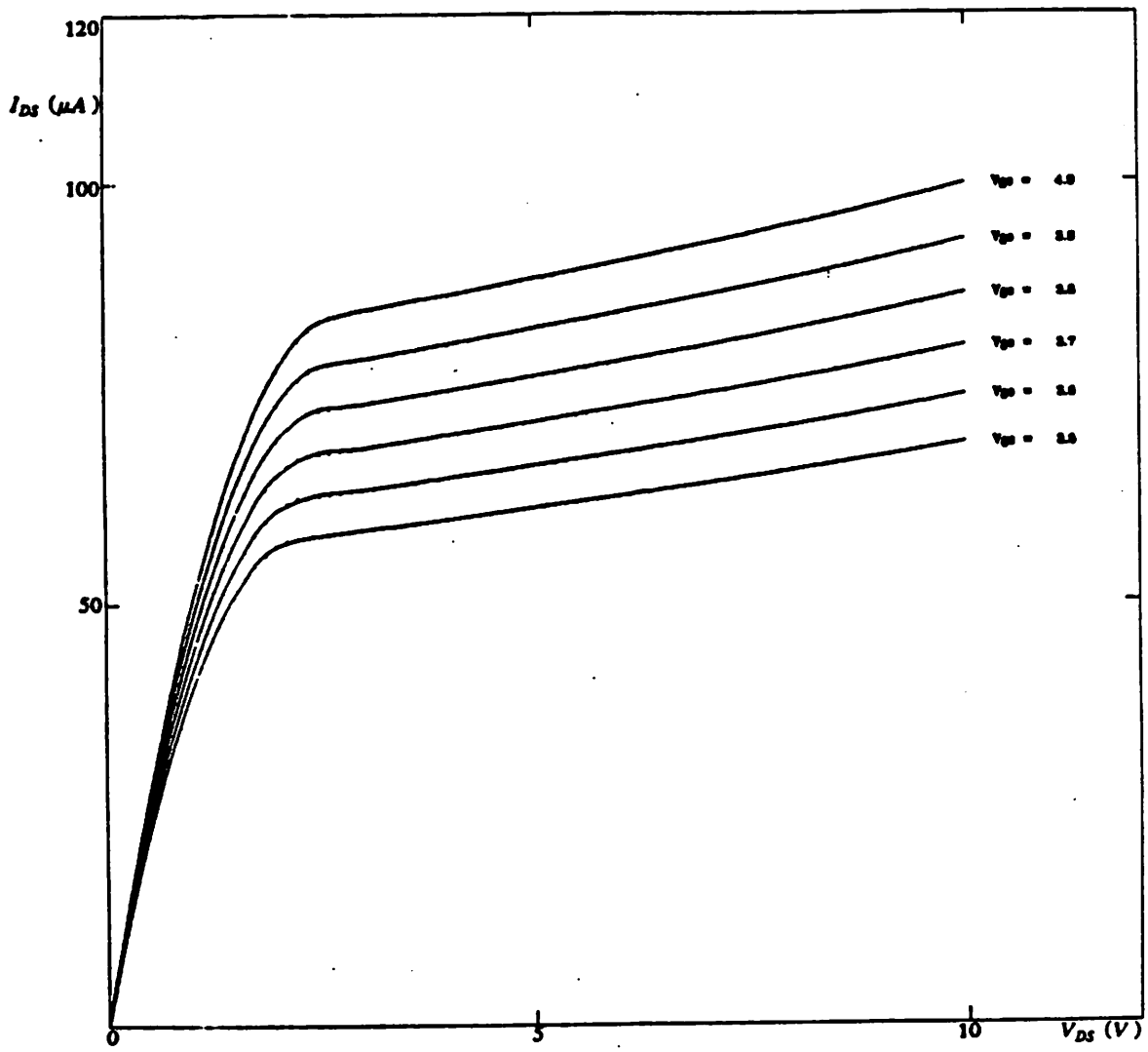
Example 2: Family of I_{ds} vs. V_{ds} Curves
Figure 4.14



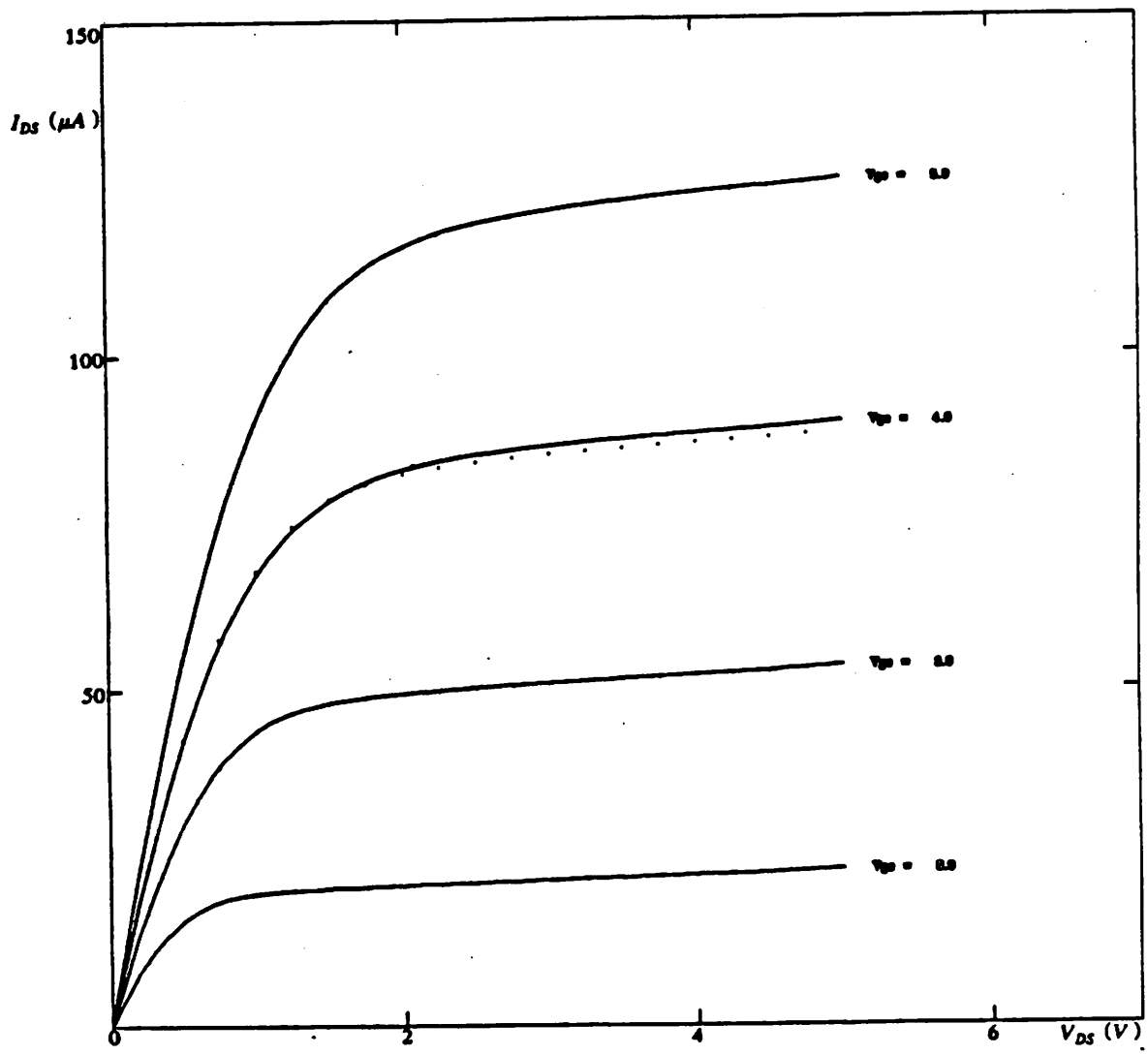
Example 2: Family of I_{ds} vs. V_{gs} Curves
Figure 4.15



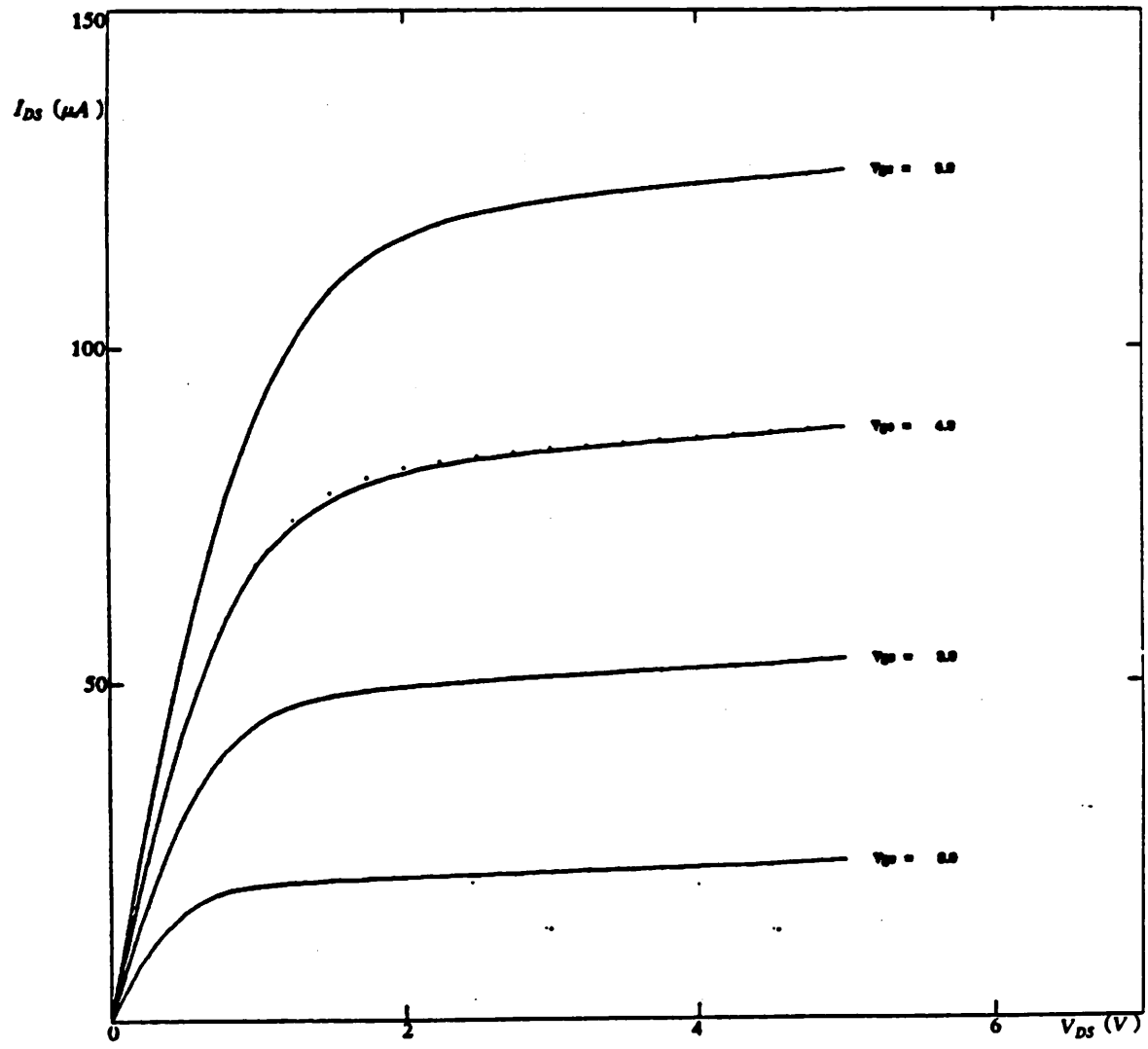
Example 2: Linear Interpolation - I_{ds} vs. V_{ds}
Figure 4.16



Example 2: Cubic Interpolation - I_{ds} vs. V_{ds}
Figure 4.17



Example 3: Linear Interpolation - I_{ds} vs. V_{ds}
Figure 4.18



Example 3: Cubic Interpolation - I_{ds} vs. V_{ds}
Figure 4.19

4.5. Chapter Summary

An Empirical MOS transistor model based on piecewise-cubic polynomials is described in this chapter. The modeling of an N-channel MOS transistor is first presented, and is then later expanded to include a P-channel MOS transistor. Two variations on the empirical method are presented, one using linear interpolation and the other using cubic interpolation, are presented. Both of the interpolation methods developed make use of first-order MOS transistor behavior, and provide a continuous and monotonic representation of the transistor's drain-to-source current.

Three examples are given to illustrate the empirical model. The data for the first two examples are generated from the Shichman-Hodges and SPICE Level-2 models respectively. The data for the third example is from direct measurement of a 1.4μ m-channel device. As demonstrated by the examples, the empirical model reproduced the behavior of the Shichman-Hodges and SPICE Level-2 models. In addition, the third example demonstrated that the empirical model can represent the behavior of an actual device from measured data. As illustrated by the three examples, cubic interpolation more accurately models the behavior of the device, however, as earlier stated, the cubic interpolation is not guaranteed to be monotonic.

The evaluation of the empirical model equations requires only single-precision floating-point addition, subtraction, multiplication and division, provided the restrictions presented in Section 4.3 are met. For a small increase in data storage, the division operation can be precalculated and stored along with the other model data, and, thus, no division operation is used during the evaluation. The equations and interpolation are consistent over all transistor operating regions, allowing the model equations to be executed without conditional branching.

CHAPTER 5

Architecture of the MOS-Model Attached Processor

The MOS-Model Attached Processor(MMAP) is used in conjunction with a host computer running an electrical circuit-simulation program. The MOS transistor's equations are evaluated by the MMAP instead of the host computer. For the MMAP to be an effective attached processor, it must evaluate the transistor equations at a much greater speed than the host processor could do by itself.

The system-level description of the MMAP is given in Chapter 2. It is shown that the data for the transistor models are stored in the MMAP in order to minimize the communication between the host and MMAP. In addition, the host does not remain idle while the MMAP is evaluating the transistor equations but performs additional operations in parallel with the MMAP. The empirical MOS-transistor model developed for use with the MMAP is described in Chapter 4. This empirical model uses only single-precision floating-point addition/subtraction[1] and multiplication, and the equations are evaluated without conditional branching.

The architecture of the MMAP is described in this chapter. The architecture is designed to efficiently evaluate the empirical model described in Chapter 4. The components of the MMAP are first described. An example of a transistor evaluation is then given, illustrating the operation of the MMAP's components. Next, a further description of the organization and accessing of transistor-model data within the MMAP is given. The MMAP architecture is shown to support the pipeline evaluation of several transistors since the evaluation of the MOS model does not require any conditional branching.

[1]For the remainder of this chapter, floating-point addition refers to both addition and subtraction.

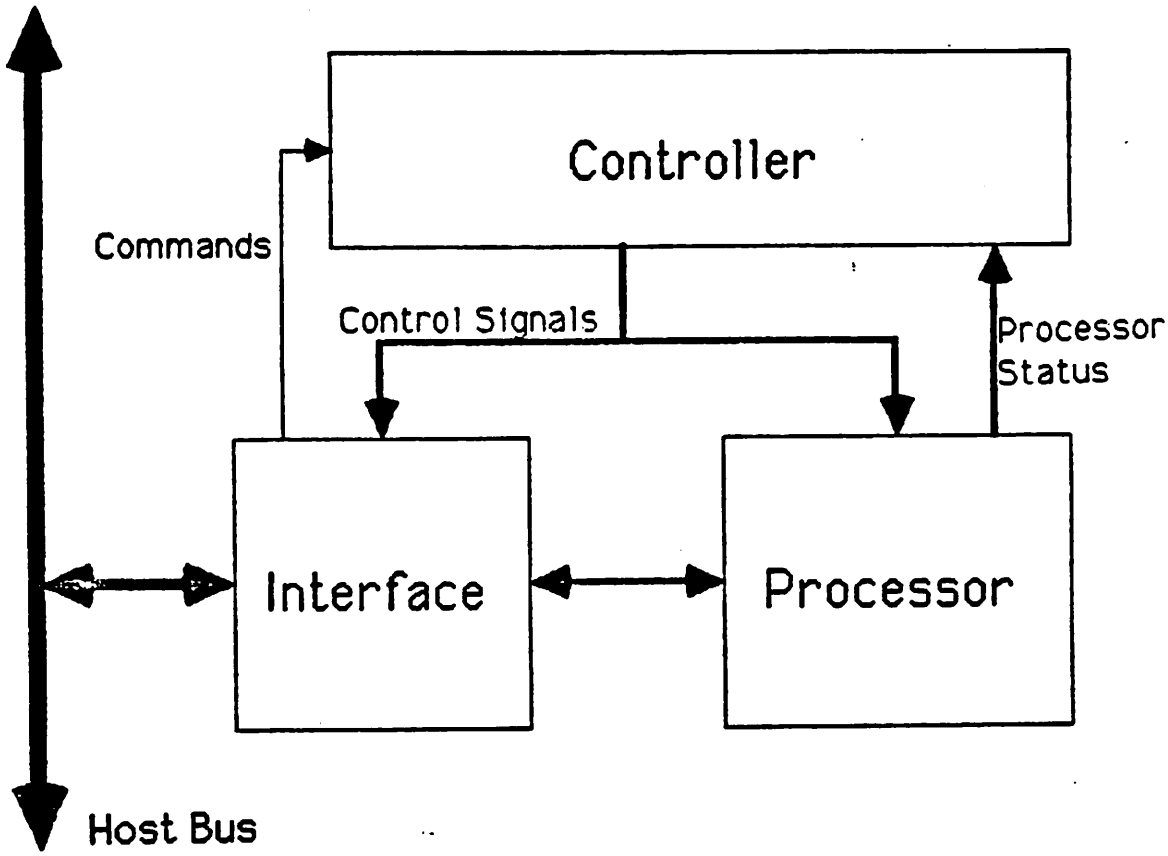
Finally, the MMAP's architecture is shown to be expandable. The expansion provides an increase in the number of transistors that can be evaluated at the same time.

5.1. Components of the MMAP Architecture

The organization of the MMAP's architecture and a description of the different components are given in this section.

As described in Chapter 2, the MMAP is a "slave" to the host processor, performing a transistor evaluation when requested by the host. The host sends the appropriate data to the MMAP and then signals the MMAP to begin a transistor evaluation. While the MMAP performs the transistor evaluation the host is able to perform other operations. Once the MMAP has completed the evaluation, the host can accesses the results of the evaluation and initiate another transistor evaluation.

The basic structure of the MMAP is shown in Figure 5.1. The MMAP consists of the Controller, Interface and Processor components. The communication between the MMAP and host is done through the Interface component. The MOS transistor equations are evaluated by the MMAP's Processor component. The program defining the MMAP's operation is stored in the Controller, and the program defines the operation of the Interface and Processor components necessary to perform the transistor evaluation.



MMAP Organization
Figure 5.1

5.1.1. Processor

The Processor component contains the data path used in evaluating the transistor equations. As described in Chapter 2, the model data are stored in the MMAP, and, as a result, the memory used to store the data is also a part of the Processor component.

The Processor component, shown in Figure 5.2, is composed of the Coefficient Memory(CM) and the Model-Processing Unit(MPU), where the MPU is composed of the Floating-Point Unit(FPU) and the Coefficient Cache(CC). Three data buses connect the functional units. The A and B bus lines connect the data outputs of the dual-output CC to the inputs of the FPU, providing the FPU with input operands. The C bus connects the output of the CM and FPU with the input of the CC. From the C bus, data from the CM or results from the FPU can be written into the CC.

The transistor model data are stored in the CM. The CM is large since it must store the data for all models required by the simulation. For example, if the MMAP is to store a maximum of 16 different models, and each model required 4 K-bytes[2] of storage, the CM must be at least 64 K-bytes in size.

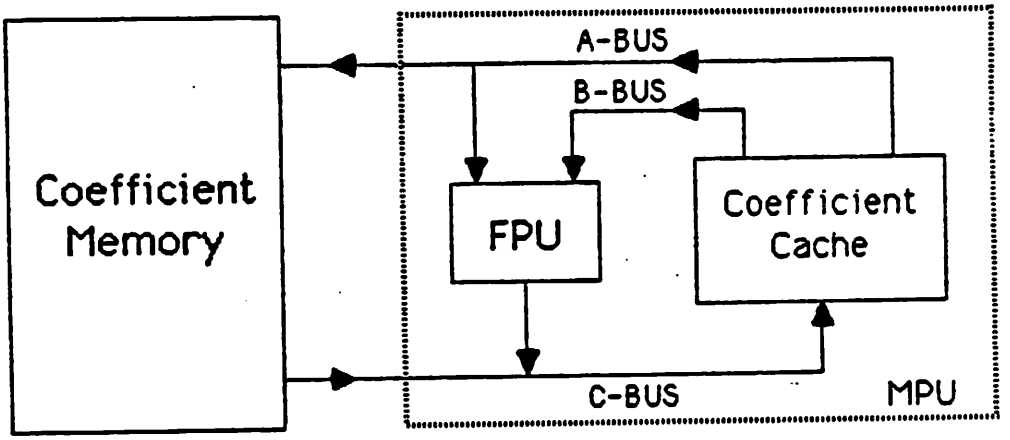
The transistor evaluation requires the access of data from the CM, but the amount of data required for the evaluation is only a small fraction of the total transistor-model data. The necessary data are accessed from the CM for each transistor simulation and are stored in the CC for the calculation.

The CC is a set of registers which store the necessary data for the current evaluation and the intermediate results from the calculations. The CC's two outputs are addressed independently. The CC provides both input operands for the FPU and stores the results of the FPU's floating-point calculations. Once the required model data have been read from the CM and stored in the CC, the CM is no longer accessed for the remainder of the calculation. The data path, comprised of the FPU and CC, is used in the floating-point calculations. The CC operates at the same clock speed as the FPU.

The FPU performs the floating-point operations of the input operands presented on the A and B buses. The result is placed on the C bus for storage in the CC. As described in Chapter 4, only single-precision floating-point addition and multiplication are needed

[2]1 K-byte is equal to 1024 bytes.

in the transistor evaluation. Therefore, the FPU supports only these floating-point operations. The FPU is described in further detail in Section 5.4, "Pipelined Operation of the MMAP".



MMAP Processor[3]
Figure 5.2

5.1.2. Controller

In general, a processor can be divided into a data part and a control part [SBN82]. A computer program is translated into a sequence of machine-language instructions, where the instructions specify the operations that have to be performed on the data part. For example, the instruction may specify the moving of data between internal registers or the addition of two numbers. In order to perform a machine-language instruction, the control part executes an ordered sequence of control signals which dictate the operation of the data part necessary to perform the instruction. The control of the data part of the MMAP can be viewed in the same manner as the general case. The Controller executes an

[3]Control signals and interface connection are not shown.

ordered sequence of control signals which direct the Processor and Interface components to perform the transistor evaluation.

The Controller component of the MMAP can be organized in either of two ways:

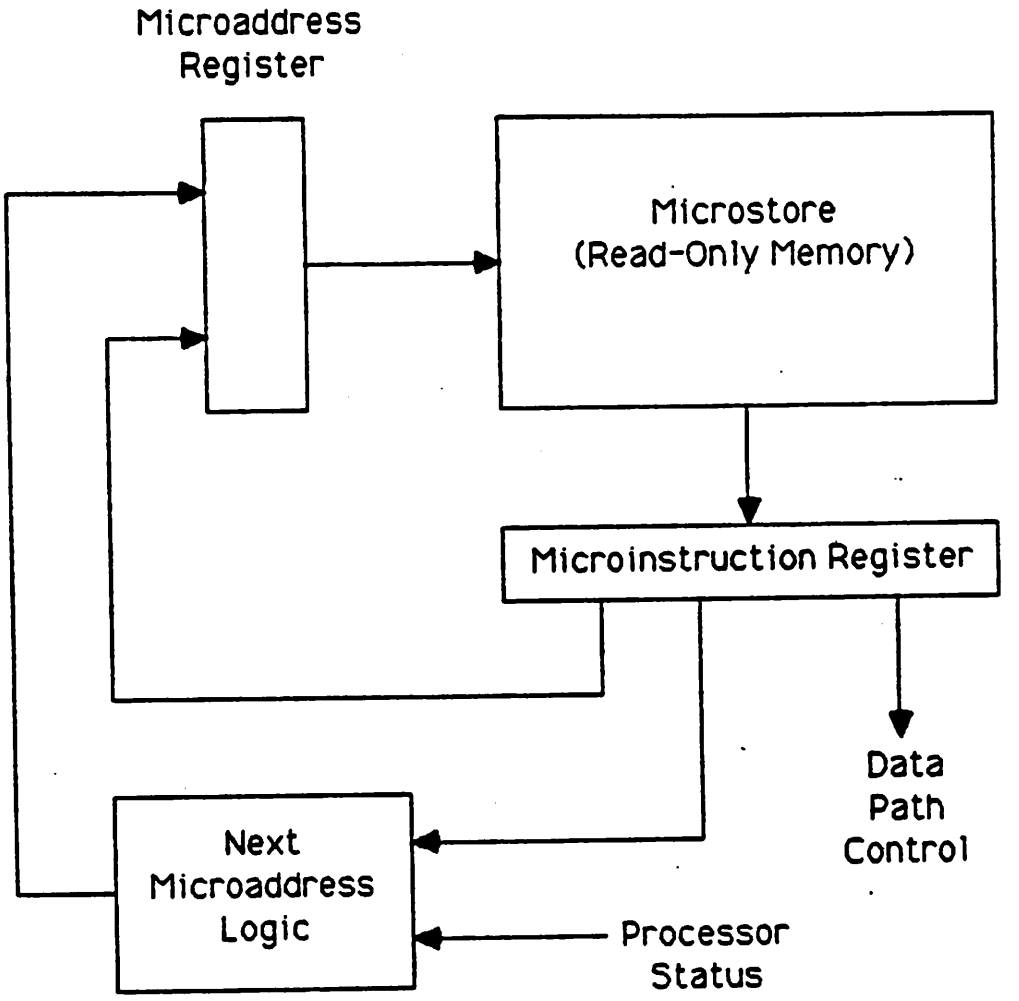
(1) A machine-language instruction set for the MMAP can be developed, and the Controller executes a sequence of machine-language instructions representing the transistor model.

(2) The transistor model can be embedded directly into the Controller as a single machine-language instruction.

(1) is the more flexible than (2). For (1), if the transistor model changes, only the sequence of machine-language instructions must change, but for (2), the machine-language instruction must change, necessitating a change in the Controller component. The transistor model developed for use with the MMAP is empirical. As described in Chapter 3, empirical models are generally unaffected by changes in process technology. Therefore, the advantage of greater flexibility that (1) has over (2) is negated. The implementation given in (2) is generally more efficient than (1) since each step in the machine-language instruction of (2) is tailored to a step in the model evaluation. As previously stated, for the MMAP to be an effective attached processor, it must perform the evaluation at as great a speed as possible. Therefore, the transistor model is embedded directly into the Controller as a single machine-language instruction.

The Controller is microprogram based, composed of a microprogram sequencer and control memory[4] as shown in Figure 5.3. The instructions defining the MOS-transistor model are stored directly in the control memory. The microprogram sequencer is used to calculate the instruction address of the Controller's control memory. All operations performed by the Interface and Processor components are controlled by the Controller component.

[4]Control memory is also referred to as microstore.



MMAP Controller
Figure 5.3

5.1.3. Interface

The Interface component is connected to the host processor's bus and is used to communicate between the MMAP and the host processor. Model coefficients are sent from the host processor to the MMAP for storage in the CM. For each transistor evalua-

tion. the transistor data is sent to the MMAP and the Controller is signaled to begin computation. Once the evaluation is completed. the results from the evaluation are returned to the host through the Interface.

5.2. Single Transistor Model Calculation: An Example

In this section. the operation of the MMAP is demonstrated by the evaluation of a piecewise-cubic polynomial of one independent variable.

$$I_{ds} = f(V_{ds}). \tag{5.1}$$

The piecewise-cubic polynomial represents the one dimensional behavior of the MOS transistor's I_{ds} as a function of its V_{ds}. The function $f(V_{ds})$ is composed of piecewise-cubic polynomials as shown in Figure 5.4. Each cubic polynomial interpolates between two consecutive data points. The function of the MMAP is to calculate I_{ds} and G_{ds} at a specified value of V_{ds}. For this example. the value of V_{ds} is within the range

$$V_{ds\,2} \leqslant V_{ds} < V_{ds\,3} . \tag{5.2}$$

The cubic polynomial. valid for the region specified by (5.2). is

$$I_{ds} = a_2 + b_2 \delta V_{ds} + c_2 \delta V_{ds}^2 + d_2 \delta V_{ds}^3. \tag{5.3}$$

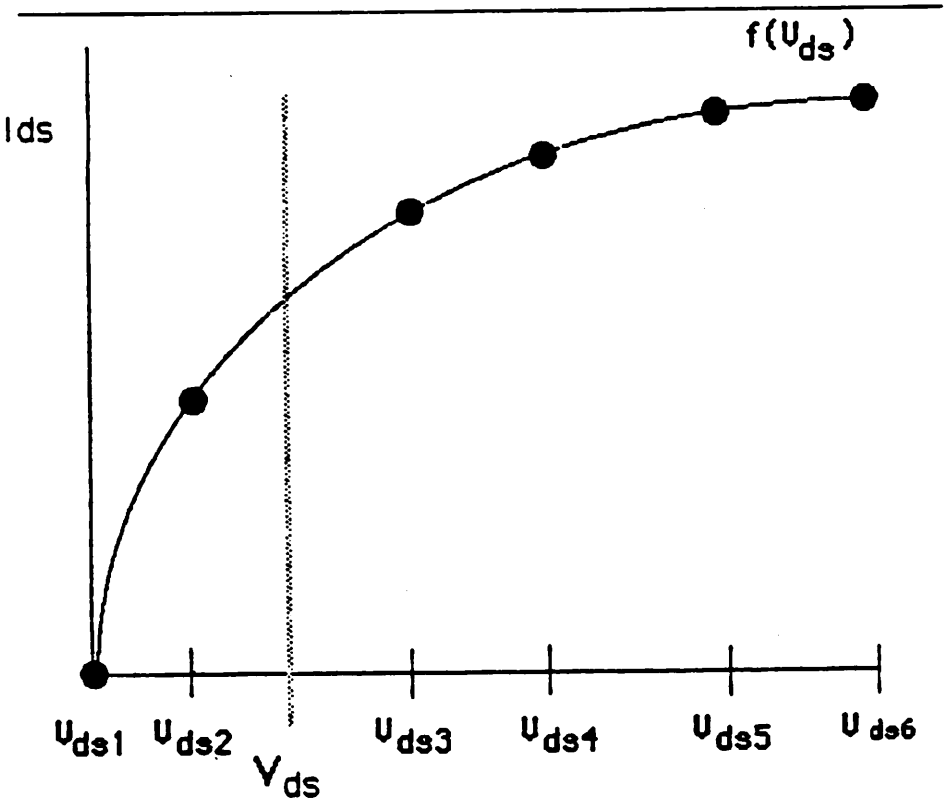
where

$$\delta V_{ds} = V_{ds} - V_{ds\,2} . \tag{5.4}$$

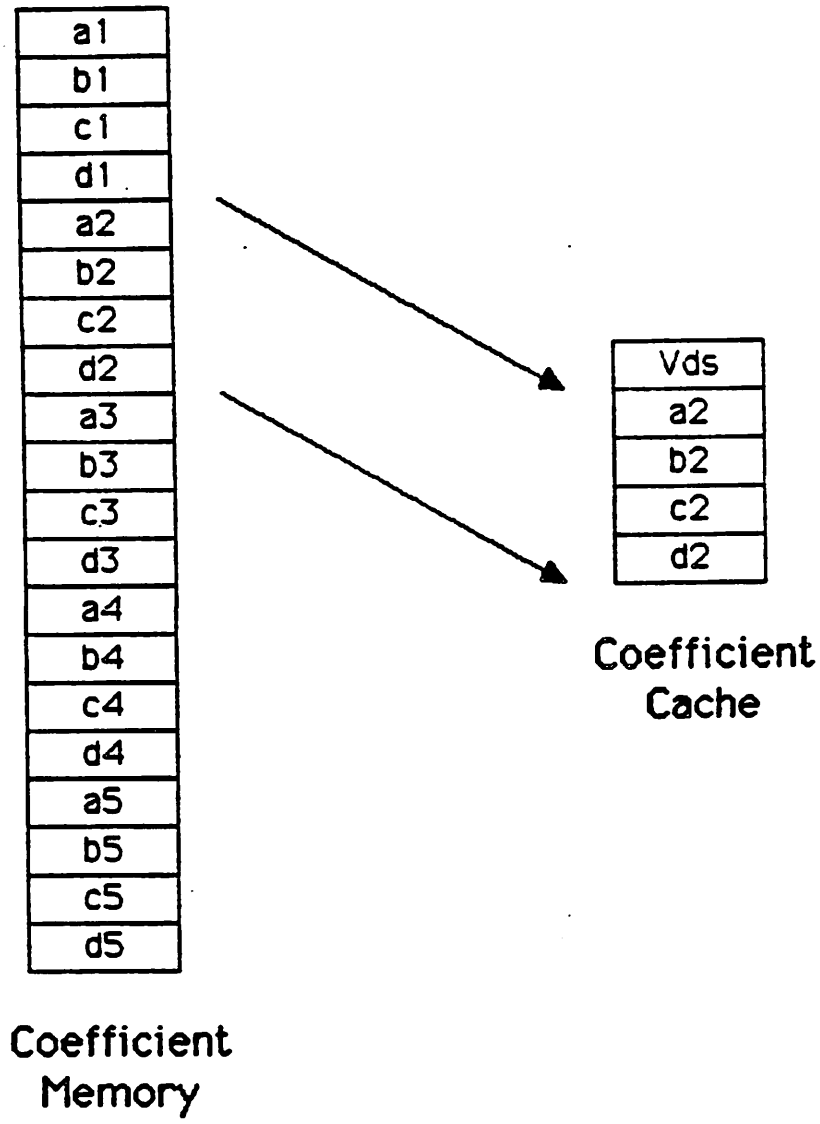
As described in Chapter 4. the polynomial's coefficients(a_2. b_2. c_2 and d_2) are algebraic functions of the values of current and derivative at the polynomial's endpoints. However. for this example the polynomial's coefficients are assumed to be stored in the CM. To calculate the value of I_{ds} at a given V_{ds}. only the coefficients of the appropriate polynomial are necessary. As shown in Figure 5.5. the values of a_2. b_2. c_2 and d_2 are read from the CM and stored in the CC. The polynomial equation is then evaluated. Horner's rule[HoS84] is used to minimize the number of floating-point operations.

$$I_{ds} = a_2 + \delta V_{ds}\,(b_2 + \delta V_{ds}\,(c_2 + d_2\,\delta V_{ds})) \qquad (5.5)$$

First d_2 and V_{ds} are input to the FPU from the CC. The FPU multiplies the two floating-point numbers and returns the intermediate result to the CC. Next, the intermediate result and the coefficient c_2 are sent to the FPU. The FPU performs the addition of the two operands and returns the result to the CC. This procedure is continued until the calculation of I_{ds} and then G_{ds} are completed.



Piecewise-Cubic Example
Figure 5.4



Access Coefficients
Figure 5.5

5.3. Storage of Transistor-Model Data

The storage and organization of the transistor-model data are described in this section.

The CM is divided into equal-sized sections, where each section stores the data for a single MOS transistor model. As given in Chapter 4, the following data are stored for each model:

(1) I_{ds}, G_{ds} and G_{gs} at the measured terminal voltages.

(2) Measured V_{ds} and V_{gs}.

(3) Values of $(\Delta V_{ds})^{-1}$ and $(\Delta V_{gs})^{-1}$, which are the inverses of the difference between adjacent measured voltages,

$$(\Delta V_{ds\,i})^{-1} = \frac{1}{V_{ds\,i+1} - V_{ds\,i}} \tag{5.7}$$

$$(\Delta V_{gs\,i})^{-1} = \frac{1}{V_{gs\,i+1} - V_{gs\,i}} . \tag{5.8}$$

(4) V_{sb}-dependent parameters.

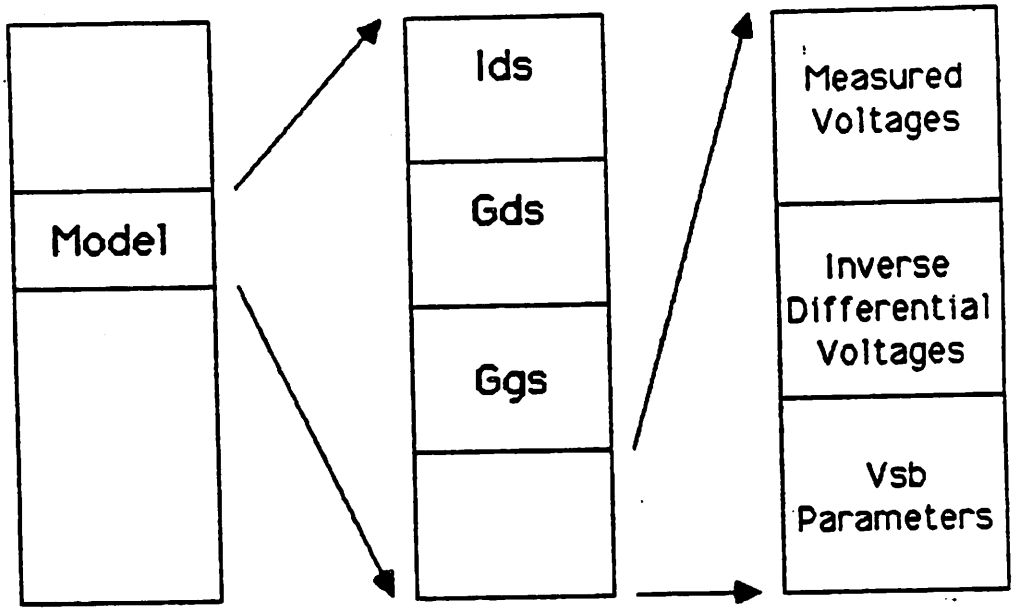
As described in Chapter 2, the information passed to the MMAP for each transistor evaluation includes a unique model pointer. The model pointer is a unique label that corresponds to the transistor model name given in the entered circuit description, and references the model data that is stored in the CM. The model pointer and differential terminal voltages are used to retrieve the required information from the coefficient memory.

The address for the CM is divided into Model, Subsection and Offset fields.

Coefficient Memory Address
Figure 5.6

The Model field specifies the part of CM containing the data for a model. The size of the Model field is dependent on the number of transistor models stored. The Subsection field

is two bits wide, specifying four equal-sized subsections. As depicted in Figure 5.7, one subsection is used to store each of the values of I_{ds}, G_{ds} and G_{gs} at the measured voltages, and one subsection is used for the storage of the V_{sb} dependence parameters, measured voltages, and inverse differential voltages. The Offset field specifies the appropriate data within a subsection.



Data Storage in the Coefficient Memory
Figure 5.7

5.4. Pipelined Operation of the MMAP

As described earlier, the data path formed by the CC and the FPU is used to perform the floating-point addition and multiplication needed for the transistor evaluation. In this section, the concurrent evaluation of several transistors by pipelining the data path is described.

5.4.1. Pipelined Floating-Point Unit

Pipelining is a technique which can be used to increase the concurrency in a floating-point unit. The technique decomposes the floating-point operation into a sequence of sub-operations with each sub-operation executed by a special dedicated stage of the floating-point unit that operates concurrently with all other stages in the floating-point unit. Registers placed between stages are used to buffer between stages which allows each stage to operate on different data simultaneously. For example, the addition of two floating-point numbers can be done using the following three-step procedure [Man82].

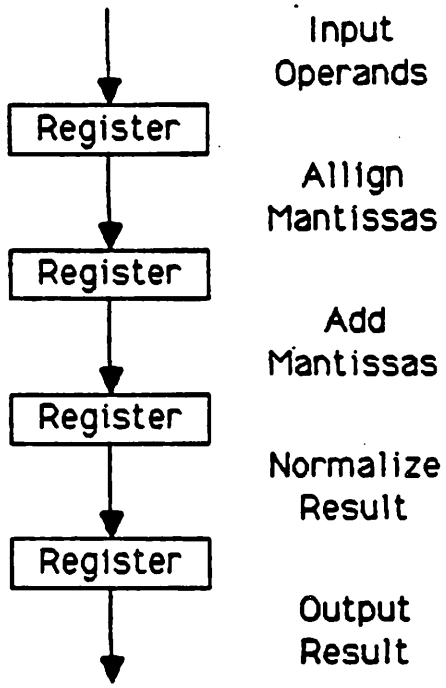
(1) Align the mantissas.

(2) Add the mantissas.

(3) Normalize the result.

As shown in Figure 5.8, the hardware for the floating-point addition can be structured as a three-stage pipeline, where each stage performs one step of the addition procedure. Assuming each stage is executed in a single clock cycle, an addition operation is performed in three clock cycles, and, once the pipeline is full, a result is available every clock cycle.

Most fast floating-point units are pipelined. The floating-point add and floating-point multiply units of the CRAY-1 require 6 and 7 clock cycles, respectively, to perform their operations, and each unit is pipelined into single-clock segments [SBN82]. As a result of pipelining, each unit can return a result with a delay of 6 cycles for addition and 7 cycles for multiplication. A floating-point unit which is not pipelined can generally perform a single floating-point operation faster than a pipelined floating-point unit. But, the calculation of a large number of floating-point operations can be performed more efficiently using a pipelined floating-point unit since the pipelined unit can return a result every clock cycle once all its segments are full.



Pipelined Floating-Point Addition
Figure 5.8

5.4.2. Pipelined Transistor Evaluation

The pipelined FPU must be efficiently utilized to achieve the greatest possible speed for model evaluation. Each empirical transistor model evaluated by the MMAP requires the calculation of several cubic polynomials. But, cubic-polynomials are not well suited for pipelined evaluation. Consider the cubic polynomial

$$y = a + b x + c x^2 + d x^3 . \tag{5.9}$$

Horner's Rule [HoS84], which minimizes the number of floating-point operations, is used to evaluate (5.9). The order of computation is

(1) $d x$

(2) $c + (d\,x)$

(3) $x\,(c + d\,x)$

(4) $b + (c\,x + d\,x^2)$

(5) $x\,(b + c\,x + d\,x^2)$

(6) $a + (x\,b + c\,x^2 + d\,x^3)$

Each step in the procedure cannot begin until the previous step is completed, and, thus, only one segment in the FPU's pipeline is used at any time during the evaluation. For example, $c + (d\,x)$ cannot be computed until $(d\,x)$ is first computed.

(5.11) can be evaluated in an alternative fashion which does not minimize the number of floating-point operations but allows for the most efficient use of the pipelined FPU. The steps used in the computation are given below, where each step contains the set of computations which can be in the pipeline at the same time.

(1) $b\,x\,,\,x^2$ and $d\,x$

(2) $a + (b\,x)\,,\,c\,(x^2)$ and $(d\,x)\,(x^2)$

(3) $(a + b\,x) + (c\,x^2)$

(4) $(a + b\,x + c\,x^2) + (d\,x^3)$

In this case, three operations can be in the pipeline in steps (1) and (2), but only one operation in the pipeline during steps (3) and (4).

The MMAP, instead of evaluating one transistor, performs the evaluation of N transistors concurrently, where N is equal to the number of segments in the FPU's data path. The transistor evaluation can be pipelined in this manner since the transistor equations are executed without any conditional branching. The pipelined evaluation of N transistors is illustrated by the evaluation of the following N polynomial equations.

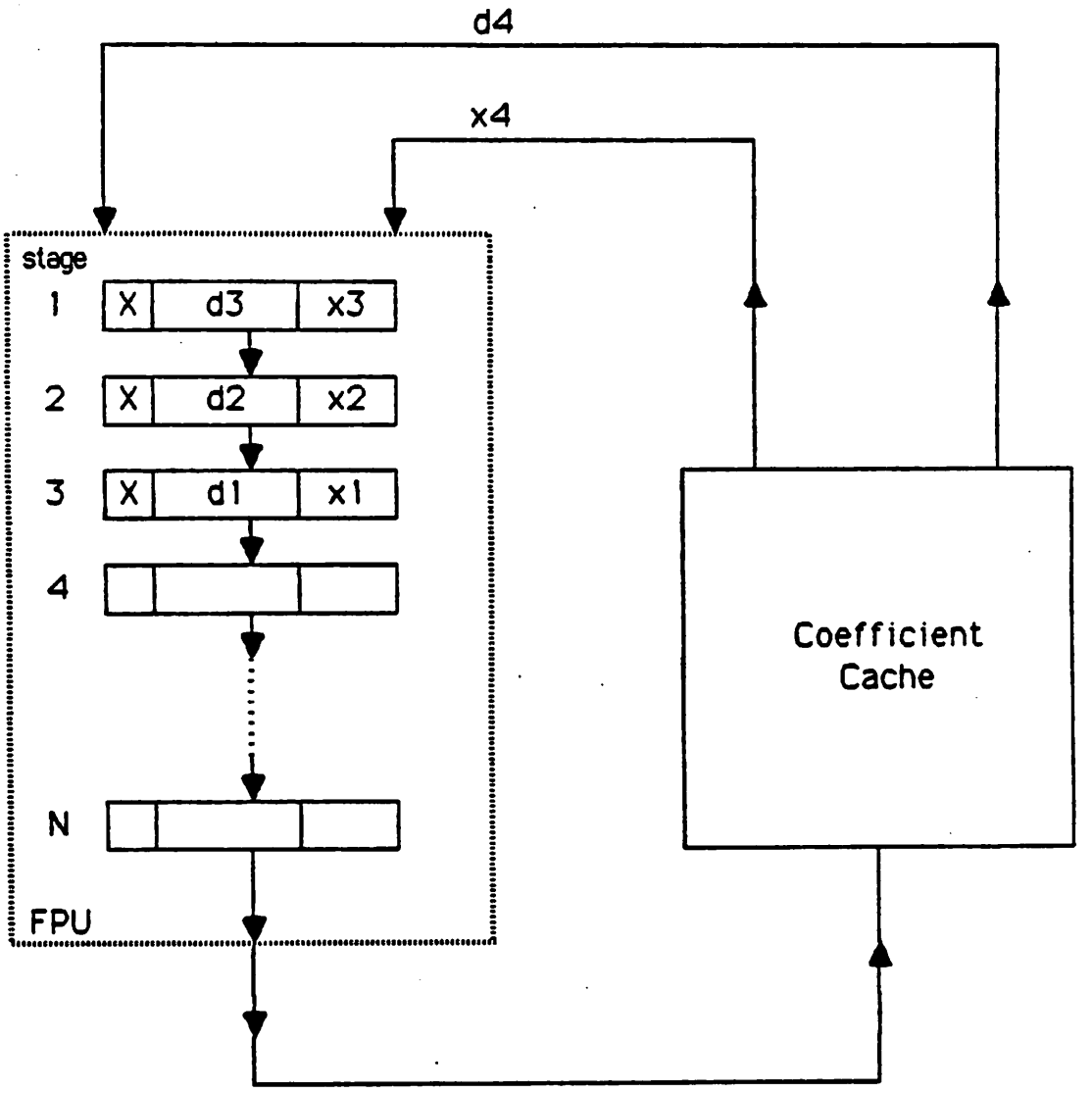
$$y_1 = a_1 + x_1\,(b_1 + x_1\,(c_1 + d_1\,x_1)). \tag{5.10}$$

$$y_2 = a_2 + x_2 (b_2 + x_2 (c_2 + d_2 x_2)).$$

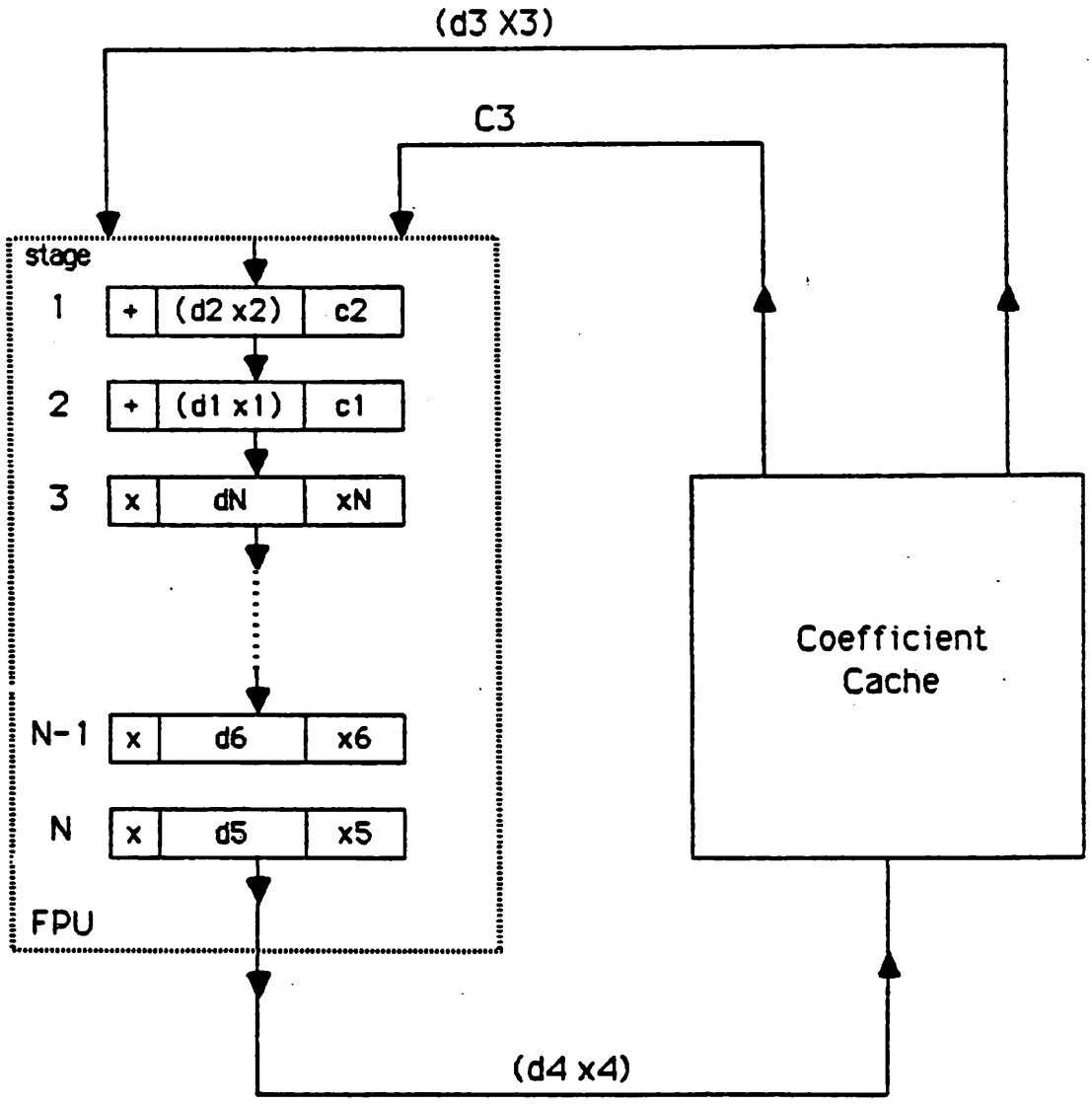
. . .

$$y_N = a_N + x_N (b_N + x_N (c_N + d_N x_N)).$$

As shown in Figure 5.9, d_1 and x are input to the FPU, followed by d_2 and x on the next clock cycle. After N cycles, the pipeline is full, and the first result, $d_1 x_1$, is returned from the the FPU. As illustrated in Figure 5.10, the results of the first step are applied to the next step in the evaluation. Ideally, once the pipeline is full, there are no breaks in the pipeline.



Loading the FPU's Pipeline
Figure 5.9



Cycling Data Through the Pipeline
Figure 5.10

5.5. Parallel MPUs

The amount of communication between the CM and the MPU is a fraction of the MMAP's total MOS transistor model evaluation time. Several MPUs can share the same CM to improve the overall performance of the MMAP. The expanded MMAP can be configured with a single Controller as shown in Figure 5.11. A single Controller is used to control all the MPU's. The total MOS transistor evaluation time for a MMAP with a single MPU(T_{single}) is the sum of the time required for reading data(T_{read}), evaluating the transistor equations(T_{eval}), and writing the results(T_{write}).

$$T_{single} = T_{read} + T_{eval} + T_{write} \tag{5.11}$$

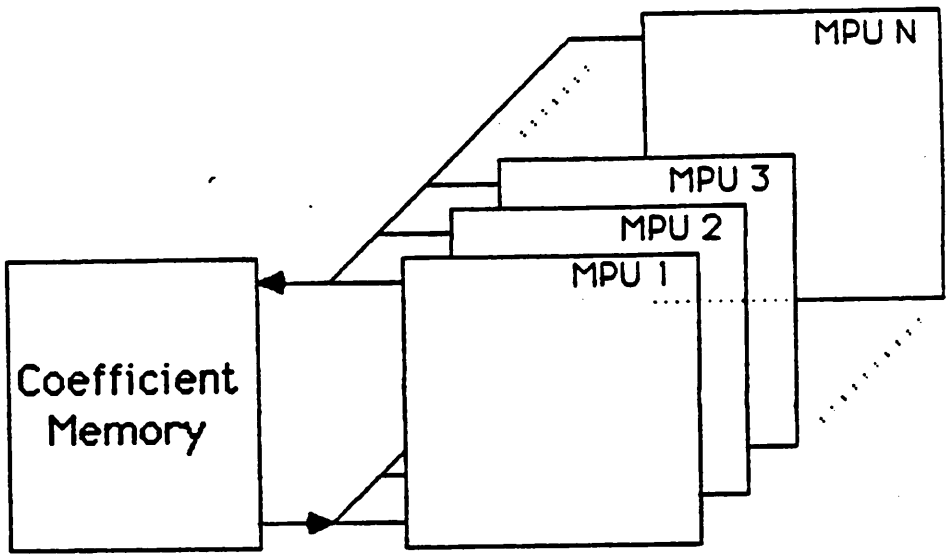
With a single Controller and N MPUs, each MPU evaluates the transistor equations simultaneously, but reading from the coefficient memory and writing results are performed serially. The total evaluation time is the sum of the time to evaluate the transistor equations by a single MPU plus the sum of read and write times for all MPUs. For N MPUs, the total evaluation time($T_{multiple}$) is

$$T_{multiple} = T_{eval} + N \times (T_{read} + T_{write}) . \tag{5.12}$$

The total evaluation time increases linearly with the sum of the read and write time. The evaluation time per MOS transistor(T_{tran}) decreases with an increasing number of MPUs

$$T_{tran} = \frac{T_{eval}}{N \times P} + \frac{T_{read} + T_{write}}{P}, \tag{5.13}$$

where P is the number of transistors evaluated concurrently by a single MPU. As the number of MPUs increases, the T_{tran} becomes limited by the sum of the data read and result write times.

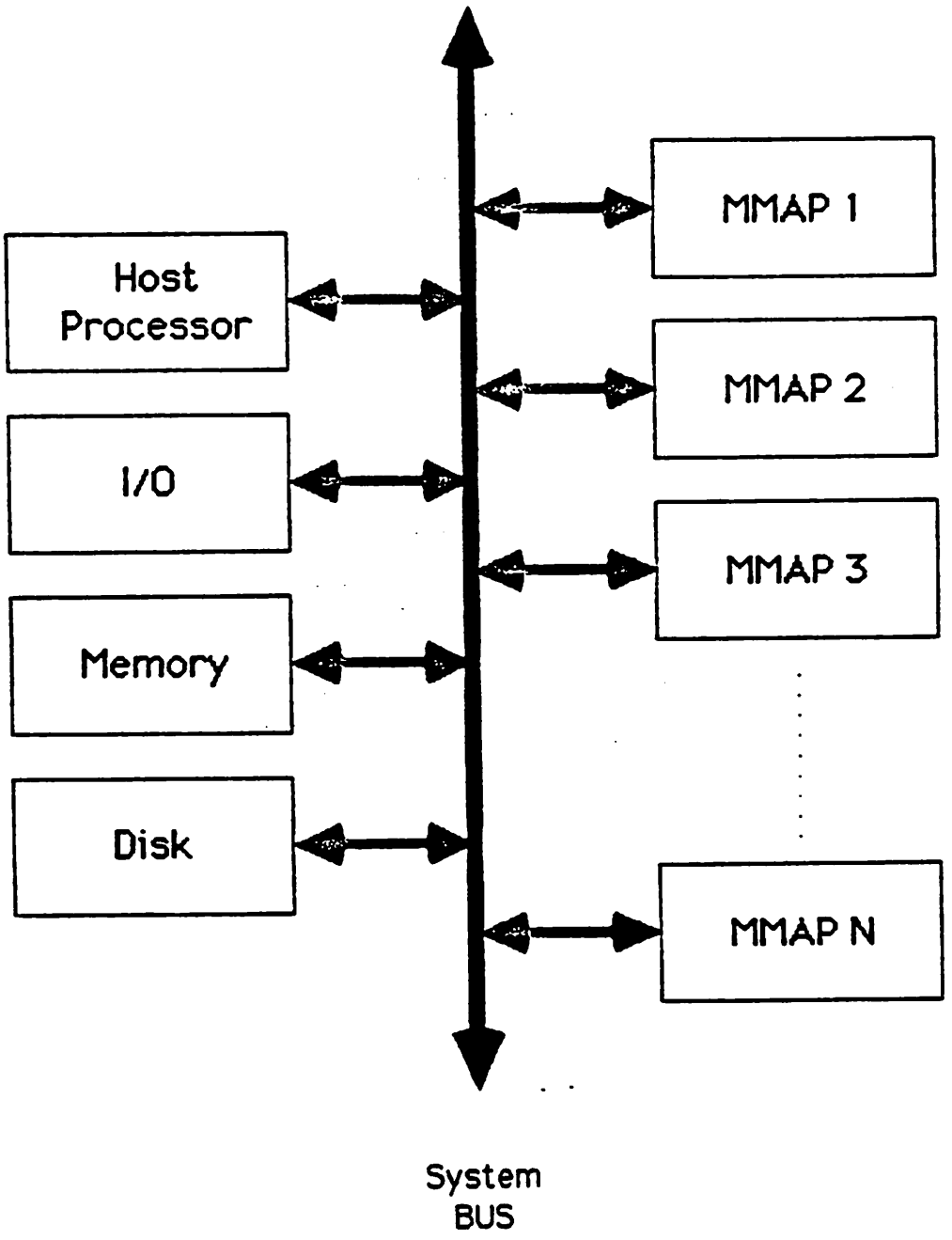


Processor with Multiple MPUs
Figure 5.11

5.6. Parallel MMAPs

The number of different transistor models that can be used in the simulation is limited by the size of a MMAP's CM. The maximum number of MOS transistor models can be increased by using multiple MMAPs as shown in Figure 5.12. The overall evaluation time per MOS transistor model can also be reduced by having multiple MMAPs. The reduction in time is dependent on being able to utilize all additional MMAPs. When all MMAPs are fully utilized and there is no conflict in communication. the time per MOS-transistor evaluation is equal to the time per MOS-transistor evaluation of a single MMAP divided by the number of MMAPs.

$$\text{Total Time per Model Evaluation} = \frac{T_{tran}}{\#\text{ of MMAP s}} \tag{5.14}$$



Multiple MMAPs
Figure 5.12

5.7. Chapter Summary

The MMAP evaluates the MOS transistor model equations. The architecture of the MMAP is designed to evaluate the transistor operation based on the cubic polynomial representation. The MMAP is composed of an Interface unit, a Controller unit and a Processing unit. Data is communicated to and from the host through the MMAP's Interface unit. The Controller unit governs the operation of the Processor and Interface units. The MMAP executes the polynomial equations without branching, allowing for the concurrent pipelined evaluation of several transistors. It is demonstrated that the architecture of the MMAP can be further optimized to allow for several Model Processing Units to be connected to a single Coefficient Memory to increase transistor-evaluation throughput. In addition, several MMAPs can be used in parallel to improve the throughput.

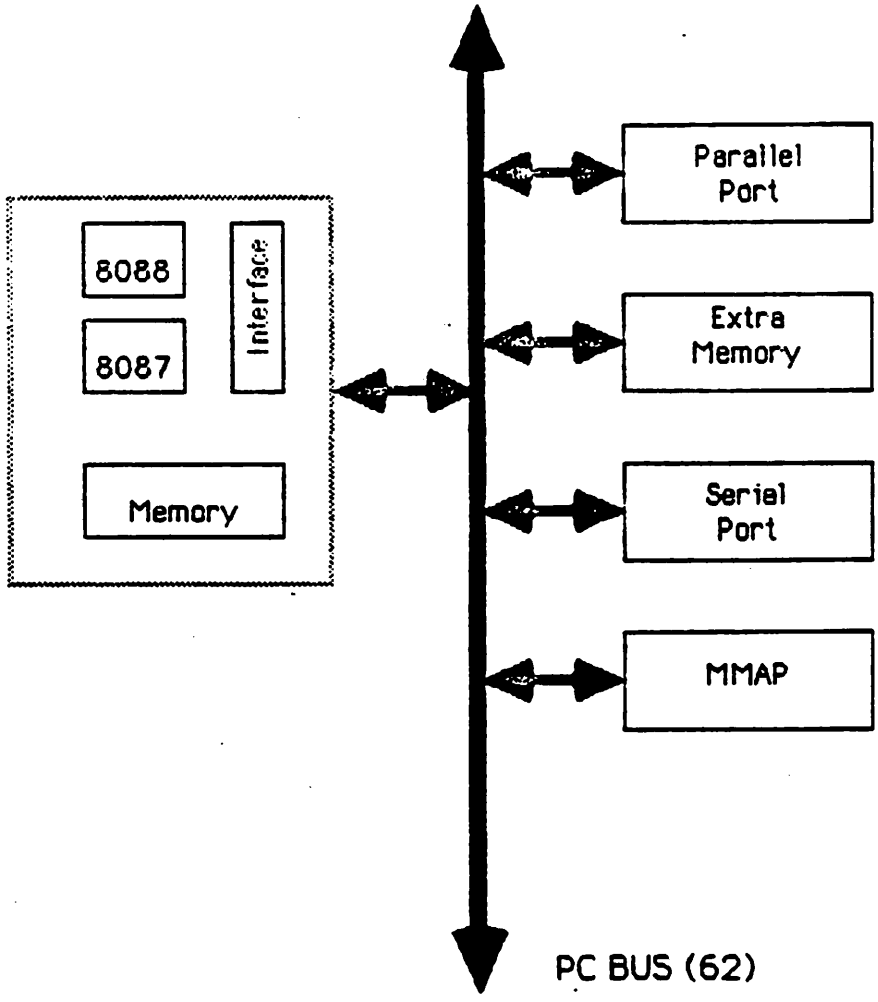
CHAPTER 6

Prototype Implementation of the MMAP on the IBM PC-XT Personal Computer

The design of a board-level prototype of the MOS-Model Attached Processor(MMAP) is presented in this chapter. The prototype's design is a practical realization of the architecture described in Chapter 5 and is optimized for the evaluation of the empirical MOS transistor model described in Chapter 4. As described in Chapter 4, the empirical model is shown to reproduce the behavior of both the Shichman-Hodges and SPICE Level-2 transistor models and data from device measurements.

The prototype MMAP is built using at total of 101 "off-the-shelf" SSI, MSI and LSI components on two wire-wrapped circuit boards which are connected through a common interface. The prototype has a single Model-Processing Unit(MPU) and can evaluate four MOS transistors concurrently. The prototype is interfaced to the IBM PC-XT personal computer through the IBM PC-XT's bus, as shown in Figure 6.1, and is used in conjunction with the BIASC circuit simulation program running on the IBM PC-XT.

A description of the IBM PC-XT personal computer and the reasons for implementing the prototype MMAP on the IBM PC-XT are first given. Next, the prototype's hardware design is described. This includes a description of the MMAP's components and the MMAP's hardware interface to the IBM PC-XT. The schematics and parts listing of the prototype MMAP are given in Appendix G. Next, the performance of the prototype is presented. Included are the measured MOS transistor-evaluation times of the MMAP and IBM PC-XT. Finally, the performance of the BIASC circuit-simulation program running on the IBM PC-XT using the MMAP is presented, and results from two example circuits are given.



IBM PC-XT With Prototype MMAP
Figure 6.1

6.1. IBM PC-XT Personal Computer

The IBM PC-XT personal computer is based on the Intel 8088 16-bit microprocessor[ReA80]. The PC-XT has an 8-bit-wide memory data bus and a 20-bit-wide address bus. The 20-bit address provides a 1M-byte address range. of which a maximum of 640K is available for random-access memory(RAM). The In8088 operates in conjunction with the Intel 8087 Numeric Data Processor(NDP)[Int84]. The In8087 performs all the floating-point computations and conforms to the IEEE floating-point standard[Kah84]. In addition. the In8087 has an 8-word stack. The stack word is 80 bits wide and conforms to the IEEE standard for intermediate storage of double-precision floating-point numbers. The stack provides storage for the intermediate results of floating-point calculations which reduces the time required for the calculations by minimizing the amount of data transferred.

The prototype MMAP is implemented for use with the IBM PC-XT. [GMP84]. [Gyu85] and [Blu85] demonstrated that the IBM PC-XT is effective in performing electrical circuit simulation. The IBM PC-XT's In8087 provides fast evaluation of "in-line" floating-point equations such as the MOS transistor equations. Since the IBM PC-XT is effective in performing circuit simulation and efficient in solving "in-line" floating-point equations. the IBM PC-XT provides a fair test of the MMAP's performance.

6.2. Design of the Prototype MMAP

The design of the prototype MMAP is presented in this section. An overview of the MMAP architecture is first given followed by a description of the board-level organization of the prototype. Next. the design of the MMAP's Controller and Processor components are presented. In the prototype design. the Interface component is part of the Coefficient Memory and is discussed in the section describing the design of the Coefficient Memory. The schematics of the prototype are given in Appendix G.

6.2.1. Design Overview

As described in Chapter 5, the MMAP is composed of the Processor, Interface and Controller components. The Processor component stores the model data and performs the calculations required in the model evaluation. The communication between the host and the MMAP is done through the MMAP's Interface component. A sequence of signals generated by the Controller, direct the operation of the Processor component in performing the transistor evaluation.

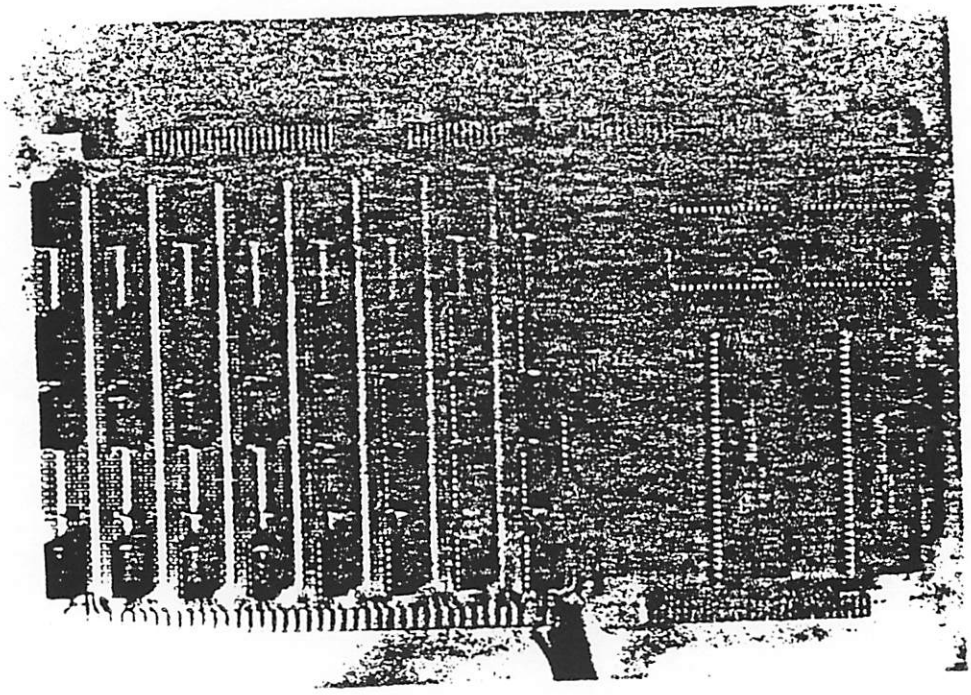
The MMAP's Processor component is composed of the Coefficient Memory(CM) and Model-Processing Unit(MPU). The CM provides storage for the transistor model data, and the MPU contains the data path used in the equation evaluation. The MPU is composed of the Floating-Point Unit(FPU) and the Coefficient Cache(CC). The FPU performs the floating-point operations, and the CC stores the data and intermediate results required for the current evaluation. Both the FPU and CC operate at the same clock speed.

As described in Chapter 5, the MMAP architecture can support the concurrent evaluation of several MOS transistors. The prototype is designed to simultaneously evaluate 4 transistors.

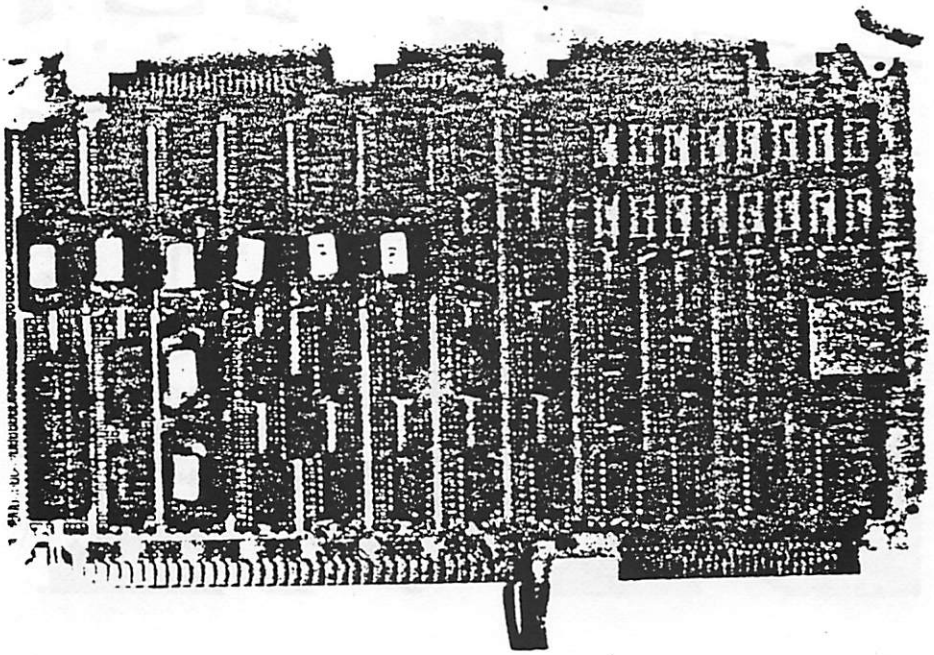
The prototype MMAP is built on two circuit boards, and the components on the circuit boards are wire-wrapped. Board A, shown in Figure 6.2, contains the FPU, the CC, and the CM's data and address registers. Board B, shown in Figure 6.3, contains the CM's memory chips and memory control logic, the MMAP's Controller, and the interface to the IBM PC-XT. The two boards are connected by three buses. The first bus (see Figures 6.2 and 6.3) connects the CM's data input/output and address, the second bus connects the clock and status signals, and the third bus connects the MMAP's Controller on Board B to control points on Board A. Board B, as shown in Figure 6.4, is connected

directly to the IBM PC-XT's bus[1]. The PC-XT's bus is 62-bits wide and includes address, data, control, status, and clocking signals in addition to several supply voltages and ground reference. The bus diagram is shown in Figure 6.5. The signals that are used by the MMAP and referred to later in this chapter are marked with an asterisk. All of the components used by the MMAP are TTL or TTL compatible and require only a +5 V power supply. The MMAP is powered by a single +5 V power supply that is separate from the power supply of the IBM PC-XT.

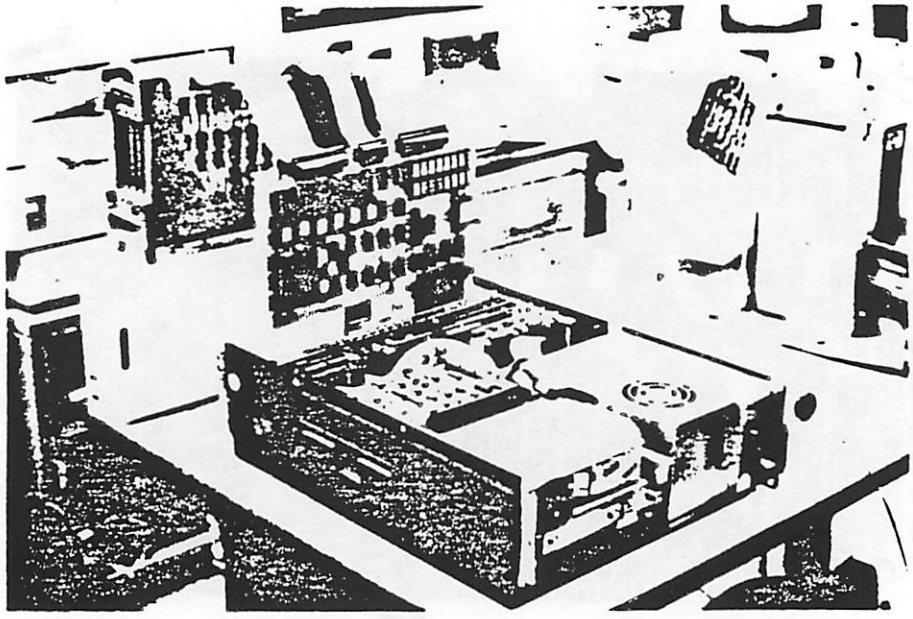
[1]The PC bus is also referred to as the I/O channel.



Board A
Figure 6.2



Board B
Figure 6.3



Connected to the IBM PC-XT
Figure 6.4

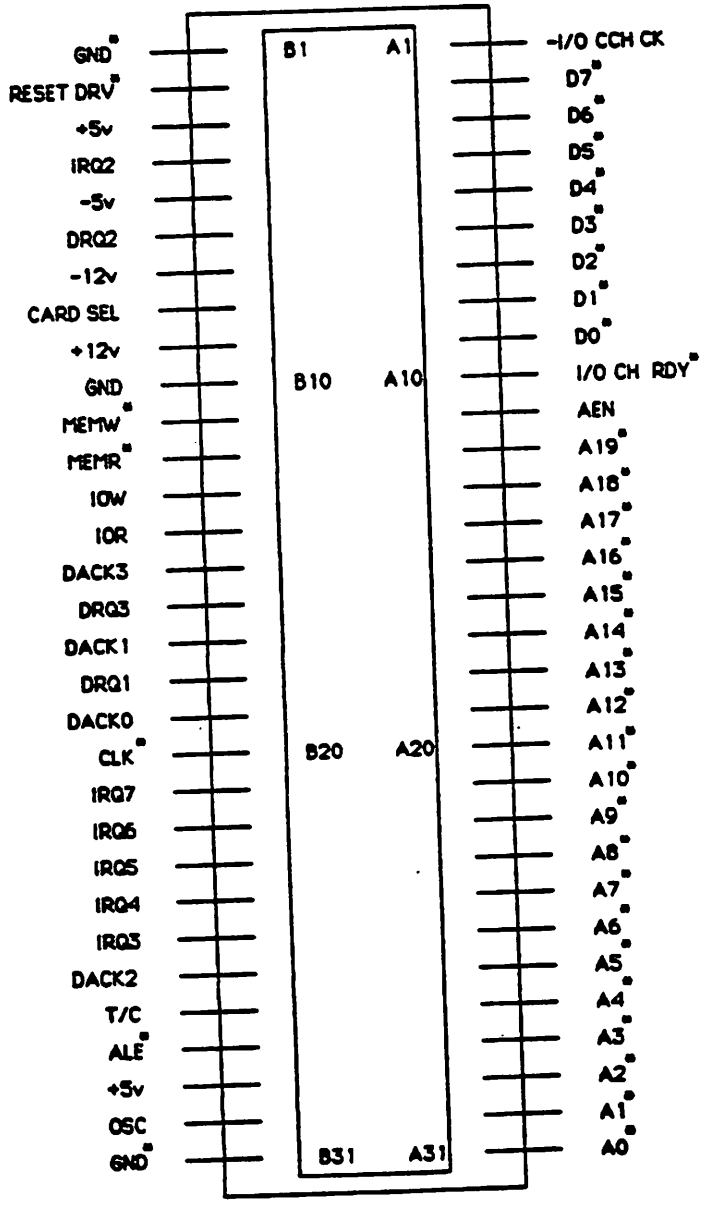


Diagram of the IBM PC Bus [IBM84]
Figure 6.5

6.2.2. Controller

The operation and design of the Controller component of the prototype MMAP is presented in this section.

As described in Chapter 5, the Controller is microprogram based. The Controller consists of control logic used to generate the microaddress, a microstore used to store the microinstructions[2], and a microinstruction register used to store the current microinstruction. The instructions for the transistor evaluation are stored directly as microinstructions in the Controller's microstore. The sequencing of the microinstructions by the Controller define the transistor-evaluation operation.

The block diagram of the Controller is shown in Figure 6.6. The microinstruction address is generated by an AM2910 [AMD83] microcontroller. The AM2910 outputs a 12-bit-wide address which can address up to 4K microinstructions. The microinstructions are stored in a 4K × 64 ROM. A new address is generated every clock cycle, and the corresponding microinstruction is latched into the microinstruction register on the rising clock edge.

The AM2910 can generate an address from an internal source or use the address supplied to its D inputs. Two of the internal sources are a microprogram counter and a five-level deep, last-in, first-out stack. The microprogram counter contains the value of the previous address incremented by one. Sequential access to microinstructions is obtained by repeated use of the address stored in the microprogram counter. Unconditional, conditional and subroutine branching are also supported. Unconditional branching within the control program is accomplished by using the address supplied by the external input as the next microinstruction address. With conditional branching, the address from the external input is used if the condition test passes, otherwise the address is given by the microprogram counter. Subroutine branching is similar to unconditional

[2]Instructions stored in the microstore are referred to as microinstructions.

branching, except the return address is stored in the stack. The subroutine return is thus accomplished by using the address stored on the stack as the next microinstruction address. The stack is five levels deep, allowing up to five levels of subroutine nesting.

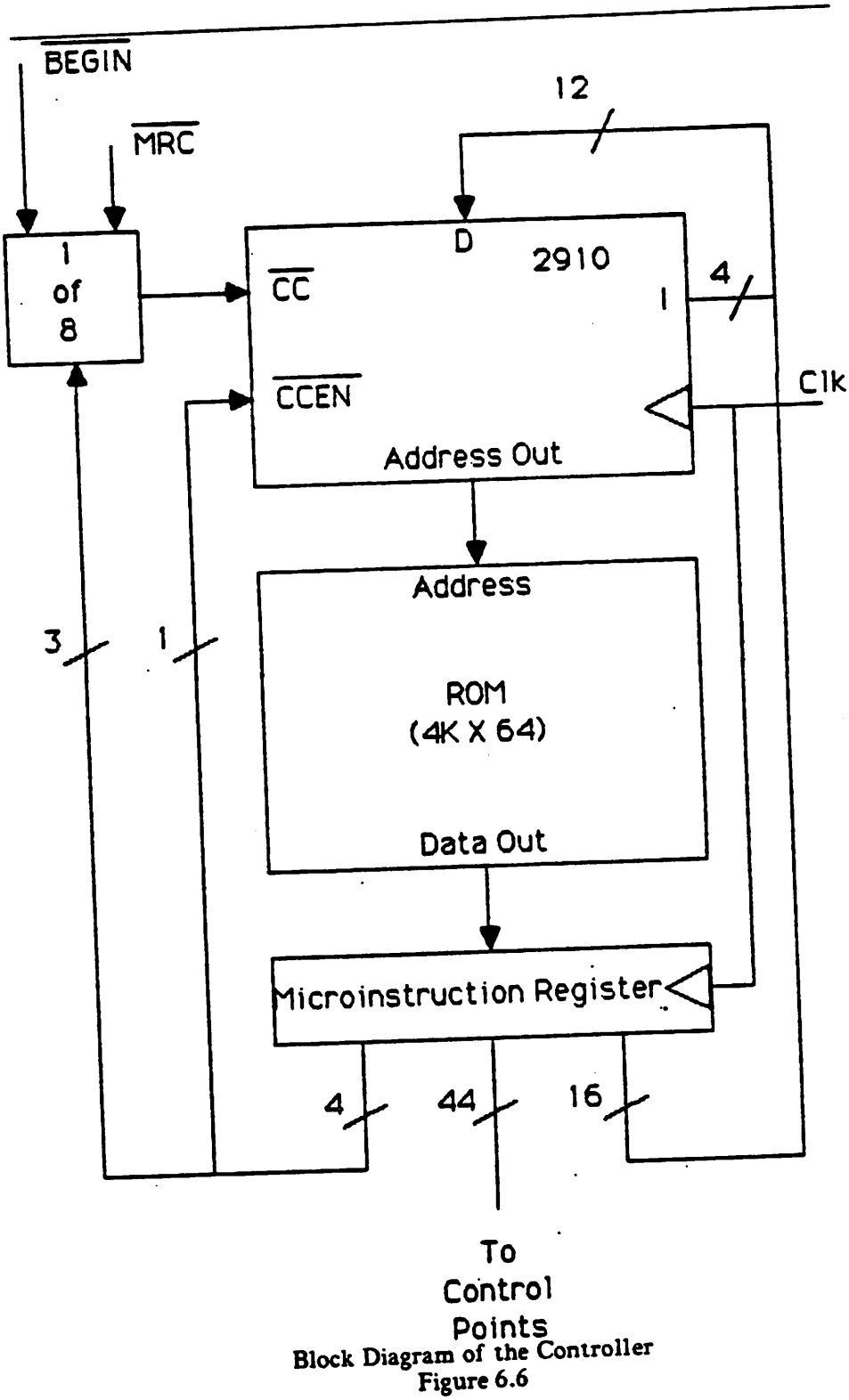
As shown in Figure 6.6, the inputs to the AM2910 are the 12-bit external address (D11-D0), the 4-bit instruction (I3-I0), condition-code ($\overline{CC}$), and condition-code enable ($\overline{CCEN}$) inputs. The I inputs provide the instruction used by the AM2910 in determining the next microinstruction address. For example, an instruction of binary value 0010 instructs the AM2910 to unconditionally branch to the address given by the D inputs. The $\overline{CC}$ and $\overline{CCEN}$ inputs determine the conditional branching. The conditional test always passes, independent of the $\overline{CC}$ input, if the $\overline{CCEN}$ input is set at a logic "1". If the $\overline{CCEN}$ input is set at a logic "0", the conditional test is determined by the $\overline{CC}$ input.

Conditional branching is required in two instances, both of which are illustrated in Figure 6.7. The MMAP remains idle after completing an evaluation, only beginning another operation when signaled by the IBM PC-XT. The IBM PC-XT signals the MMAP to begin evaluation by setting the $\overline{BEGIN}$ signal to a logic "0". Once signaled, the MMAP performs the evaluation until completion. Also, as described in Section 6.2.5, the communication between the CM's internal memory storage and input/output registers is asynchronous. Since the communication is asynchronous, the Controller must wait for a signal from the CM before continuing. The CM signals the Controller that the memory operation is complete by setting the $\overline{MRC}$ (Memory Reference Complete) signal to a logic "0"

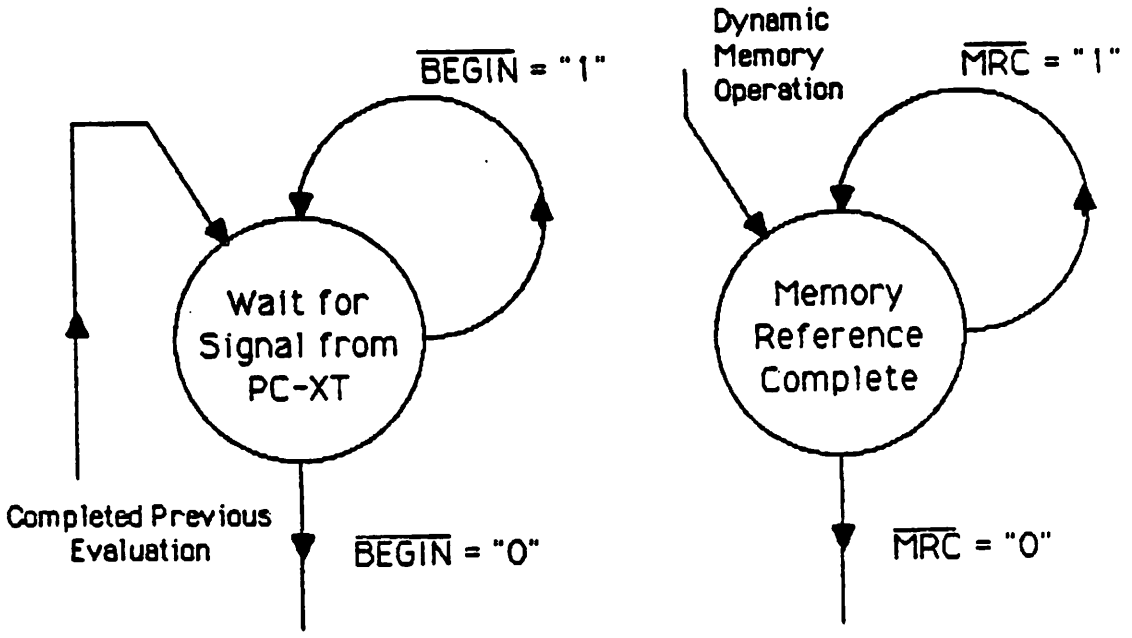
As shown in Figure 6.6, the $\overline{BEGIN}$ and $\overline{MRC}$ signal are connected to 2 inputs of a 74ls151 1-of-8 multiplexor, and the output of the multiplexor is connected to the $\overline{CC}$ input of the AM2910. The three CCS signals specify which one of the eight inputs to the multiplexor is input to the $\overline{CC}$ input of the AM2910. The $\overline{BEGIN}$ signal is selected as the input to the $\overline{CC}$ input when the MMAP is waiting to be begin another evaluation, and the $\overline{MRC}$ signal is selected as the input to the $\overline{CC}$ input when the MMAP is checking for the completion of a memory operation.

The microstore is composed of Read-Only Memory(ROM) which can store up to 4K microinstructions, and the width of the microinstruction is 64 bits. The address generated by the AM2910 is used to address the ROM. The ROM is composed of eight 4K × 8-bit ROM(Read-Only Memory) chips. The eight ROMs share the common address input and provide a combined 64-bit-wide output.

As shown in Figure 6.6, the output of the microstore is directed to the input of the instruction register. One instruction is generated per clock cycle, and the microinstruction that is currently being executed is stored in the instruction register. New microinstructions are loaded into the instruction register on the rising edge of the clock signal. The output of the instruction register is connected to control points throughout the remainder of the MMAP and to control points within the Controller. The signals fed back to the Controller select the appropriate condition code, providing the D, I and $\overline{\text{CCEN}}$ inputs to the AM2910 and the CCS signals to the 74ls151 multiplexor. The instruction register is composed of 8 AM2954 [AMD83] octal register chips.



Block Diagram of the Controller
Figure 6.6



Conditional Branches
Figure 6.7

The format of the microinstruction is shown in Figure 6.8. Each microinstruction is logically divided into 4 fields, one each for the Controller, CM, FPU and CC.

Microinstruction Fields
Figure 6.8

41 of the 64 instruction bits are used for control points throughout the MMAP and are described in the following sections on the FPU, CC and CM and 20 instruction bits are used in the determination of the microcontroller's next address. 3 instruction bits are not used.

A complete list of the microinstruction bits are listed in Table 6.1. The list includes the signal name, the name of the component (eg. FPU) it controls, and a short description

of the signal.

Control Signals			
Micro-instruction	Signal Name	Component name	Description
0-11	D0-D11	Controller	Explicit Address
12-15	I0-I3	Controller	μ-controller Function
16	$\overline{\text{CCEN}}$	Controller	Condition-Code Enable
17-20	DPA0-DPA3	CC	Dual-port Address
21	$\overline{\text{DPW}}$	CC	Dual-port Write
22	$\overline{\text{DPL}}$	CC	Output Low
23	$\overline{\text{CMA}}$	CM	Memory Request
24-29	AFL0-AFL5	CM	Address Formation Logic Control
30-34	LA0-LA4	CM	Load Address Register
35-37	CCS0-CCS2	Controller	Condition Code Select
38	$\overline{\text{CMR}}$	CM	CM Read by MMAP
39	$\overline{\text{CMW}}$	CM	CM Write by MMAP
40	$\overline{\text{LDATA}}$	CM	Load Data-Input Register
41	$\overline{\text{ENBLE}}$	CM	Enable Data-Output Register
42	$\overline{\text{TOP}}$	CM	
43-45	FP_F0-FP_F2	FPU	FPU Function
46	FPALU_U1	FPU	Unload ALU
47	FPMUL_U1	FPU	Unload Multiplier
48	FP_U0	FPU	
49,50	FP_L0,FP_L1	FPU	Load FPU
51	-	unused	-
52	-	unused	-
53	SRW	CC	Static Ram Write
54-61	SRA0-SRA7	CC	Static Ram Address
62	-	unused	-
63	AFL6	CM	Address Formation Logic Control

Table 6.1

6.2.3. Floating-Point Unit

As described in Chapter 5, the FPU performs floating-point addition, subtraction and multiplication. The two input operands are entered at the same time from the MMAP's A bus and B bus. The FPU performs the floating-point operation and outputs the result to the MMAP's C bus. The architecture of the FPU is pipelined. Pairs of operands can be sent continually to the FPU, with the FPU performing the calculation and returning the results in order.

The FPU is built using the Weitek 1032 32-bit multiplier[Wei84] and Weitek 1033 32-bit floating-point ALU[Wei84]. The Weitek chip-set is used since it performs all the

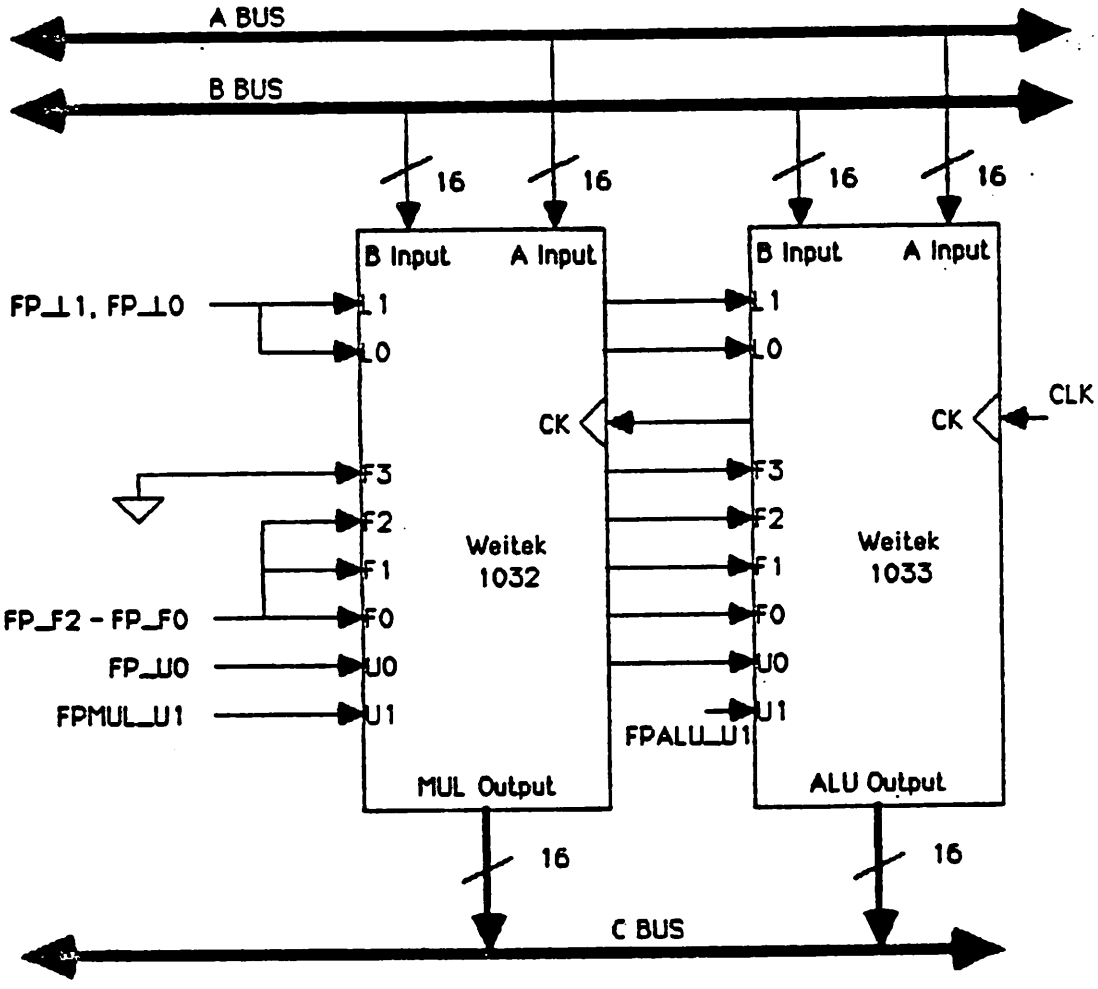
required floating-point operations. has a pipelined data path. and executes the floating-point operations very quickly. The WTL1033 performs floating-point addition and subtraction. and the WTL1032 performs floating-point multiplication. Both the Weitek WTL1032 and WTL1033 are pipelined. New operands can be input to the WTL1032 and WTL1033 every 2 clock cycles. and both the WTL1032 and WTL1033 have a pipeline latency of 10 clock cycles. Operating in pipeline mode at a clock speed of 10Mhz. the Weiteks can execute all floating-point operations in 1 μ-second and output results every 200 n-seconds. As a comparison. the IBM PC's Intel 8087 operating at 4.47Mhz executes a floating-point add on data stored in its own registers in 18 μ-seconds.

The data inputs and outputs of the WTL1032 and WTL1033 are connected in parallel. as shown in Figure 6.9. One input of each Weitek is connected to the MMAP's A bus and the other to the MMAP's B bus. The outputs of the WTL1032 and WTL1033 are connected to the C bus. The Weitek's two data inputs and one data output are each 16 bits wide. requiring 2 clock cycles to load the 32-bit input operands and two clock cycles to unload the 32-bit result of the floating-point operation. In both loading and unloading of data. the 16 most-significant bits of the floating-point number are entered first followed by the 16 least-significant bits. The 16-bit-wide data inputs define the width of the MMAP's A. B and C buses to be 16 bits.

The WTL1032 and WTL1033 each have 8 control inputs[Wei84]. On each Weitek chip. the L1 and L0 inputs control the loading of input data. the U1 and U0 inputs control the unloading of results. and the F3-F0 inputs select the appropriate floating-point function. The data and control inputs are latched into internal registers on positive going clock transitions. but only the function entered along with the most-significant portion of the input operands is used.

As shown in Figure 6.9. a common set of control signals from the Controller is shared by the two Weitek chips. The load inputs are both connected to the FP_L0 and FP_L1 control lines. The F0-F2 function inputs are both connected to the FP_F0-FP_F2 control lines. and the U0 unload input of both chips are connected to the FP_U0 control

line. The MMAP uses only a subset of the floating-point functions available from the Weiteks, and, for these operations, the F3 signal is always a logic "0". The F3 input to both Weitek chips is connected to ground. The same data is loaded into both Weiteks at the same time, but only the result from the appropriate output is accessed. The U1 unload input for each chip is connected to separate signals from the Controller, FPMUL_U1 for the WTL1032 and FPALU_U1 for the WTL1033. U1 enables the tri-state output of a Weitek chip. Since only one output can be enabled at a time, there are separate U1 signals to each Weitek.



Block Diagram of the Floating-Point Unit
Figure 6.9

6.2.4. Coefficient Cache

The CC is used to store data for use with the FPU. The CC has 2 output ports and a single input port. Separate data can be accessed from two locations within the CC and output to the A and B buses at the same time. The CC's input port is connected to the C bus. The CC and the FPU operate synchronously, with the CC performing read and write operations in a single clock cycle.

The CC, shown in Figure 6.10, stores 256 16-bit words. Since four transistors are evaluated simultaneously, 64 16-bit words are available for each transistor. The CC is composed of a 256 × 16 static memory and a 16 × 16 dual-ported memory register. The output of the static memory is input to the dual-ported memory register, which allows data to be read from the C bus and written to the A and B buses concurrently.

The 16 × 16 dual-port register can store eight 32-bit floating-point numbers, providing direct access to four pairs of operands by the FPU. In general, the same floating-point operation is performed on the four pairs of operands, and each pair is specific to one of the four transistors being evaluated.

The dual-port register has a separate 4-bit address for each of the two data outputs. During a read operation, the A-address port addresses the data-output port connected to the A bus, and the B-address port addresses the data-output port connected to the B bus. During a write operation, the A-address port specifies the location within the dual-port register in which the data from the static memory is written.

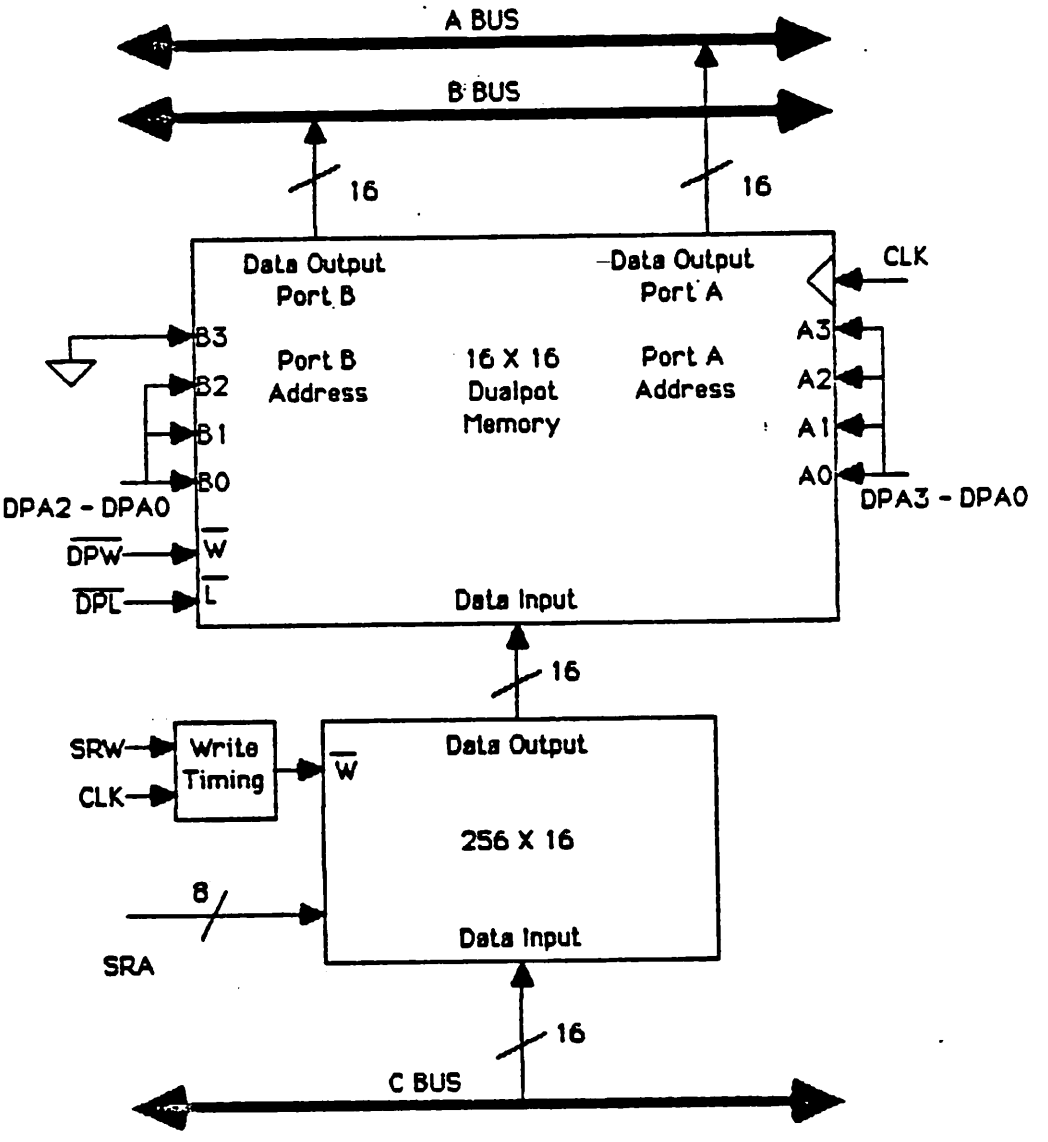
The four DPA signals are the address of both ports. As shown in Figure 6.10, the lowest three A-address and B-address inputs are connected to the DPA2-DPA0 control signals. The fourth A-address input is connected to the DPA3, and the fourth B-address input is connected to ground(logic "0"). The four DPA signals provide the address in a write operation since only the A-address port is used. In a read operation, all 16 words can be output to the A bus, but only the lowest 8 words can be output to the B bus. $\overline{\text{DPW}}$ is the write signal to the dual-port memory and is active low. $\overline{\text{DPL}}$ signal is active low and forces the outputs of the dual-port register to a logic "0".

For floating-point operations, operands for the B bus are stored in the lower 8 locations, and operands for the A bus are stored in the upper 8 locations. The two operands for a floating-point operation are stored in locations with the same lowest three address bits, but with a different fourth address bit. For example, the A operand is stored in the dual-port location specified by binary address 1001, and the B operand is stored in the

dual-port address specified by binary address 0001. Setting DPA to 1001 and reading the data stored in location 1001 is output to the A bus and the data stored in location 0001 is output to the B bus.

The eight SRA control signals specify the static memory address. The SRW control signal is the static memory write signal and is active high. With SRW set to a logic 1, the data present on the C bus is written into the static memory location specified by the SRA.

The 16 × 16 dual-port register consists of 4 16 × 4 AM29705 Dual-port Registers[AMD83]. The 256 × 16 static memory consists of 4 256 × 4 AM91122-60 static RAM[AMD83].



Block Diagram of the Coefficient Cache
Figure 6.10

6.2.5. Coefficient Memory

The CM contains 128K bytes of dynamic, random-access memory(DRAM). As shown in Figure 6.11, the CM is dual-ported, accessed by both the MMAP's Processor component and the IBM PC-XT. The CM is part of the IBM PC-XT's memory address

space, and also acts as the interface to the IBM PC-XT. The PC's data and address bus are connected to the CM, and the PC performs memory-read and memory-write operations to the CM. Logic within the CM controls the reading and writing of data since the CM can only perform memory operations from one port at a time.

The IBM PC-XT has a 1 M-byte(2^{20}) address space, of which 640K-bytes are available for random-access memory. The remainder of the address space is reserved. As shown by the memory map in Figure 6.12, the CM is located in the last 128K bytes of the IBM PC-XT's 640K memory address space. The first 512K-bytes are available only to the PC, and the remaining 128K-bytes are shared between the PC and MMAP.

The block diagram of the CM is given in Figure 6.13. In addition to the dynamic memory, the CM is composed of a memory controller, data-input and data-output registers for the two ports, address registers for both ports, and combinational logic used by the MMAP to generate an address in the CM. The PC's 8-bit data bus is connected to the CM's data input and data output through 2 8-bit latches. The 16-bit A and C buses of the MPU are connected to two 16-bit registers. Of the PC's 20-bit address, the three most-significant bits specify the CM and the lower 17 bits are the address within the CM. The CM's address from the MMAP is stored in a 17-bit register. The PC's and MMAP's read and write signals are connected to the memory controller. The memory controller selects the data and address from the requesting port and performs the memory operation.

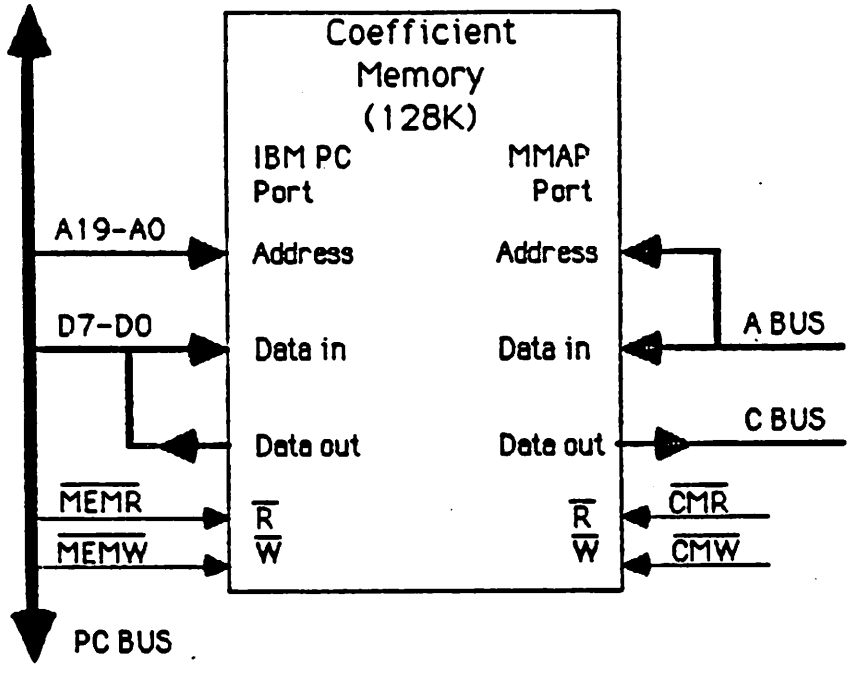
The CM detects a memory request from the PC by detecting if the address is valid and within the range of the the CM's memory space(512K to 640K). The three most significant bits of the PC's 20-bit address, if at binary 100, initiate a memory operation with the CM. For all CM memory read and write operations requested by the PC, the CM sets the PC's I/O Channel Ready($\overline{IOCR}$) signal to a logic "0", causing the PC to idle while maintaining valid data(in write operations) and address signals. In a read operation, the data is read from the dynamic memory location specified by the PC's address and stored in the data-output register connected to the PC's data bus. The tri-state

output of the register is enabled, allowing the data stored in the register to be present on the PC's data bus. The memory controller then sets the $\overline{IOCR}$ signal to a logic "1", causing the PC to latch the data from the data bus. In a write operation, the data on the PC's data bus is written to the memory location in the CM specified by the lower 17 bits of the PC's address. When the write operation is complete, the CM sets the $\overline{IOCR}$ signal to a logic "1" allowing the PC to continue with its next operation.

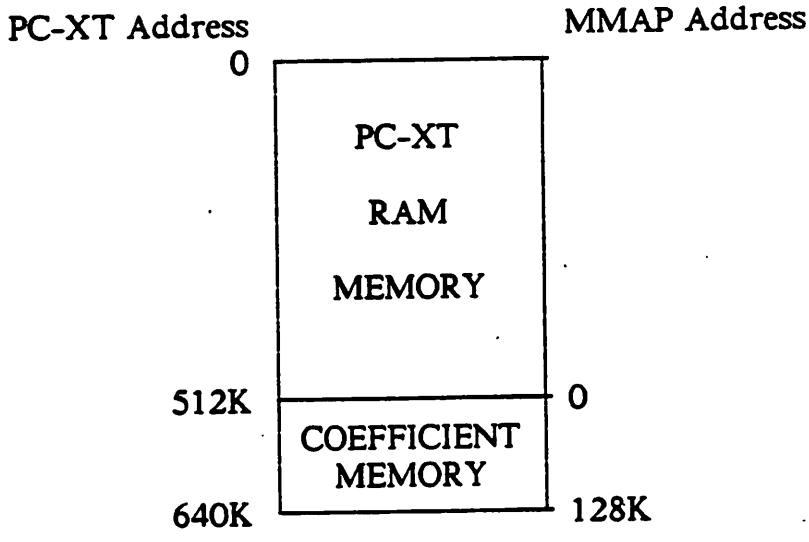
The two 16-bit wide data registers buffer the transfer of data between the CM and the MPU. The Processor's data paths are 16 bits wide, but the dynamic memory is byte oriented. Data is written from the MPU to the CM by storing the data in the 16-bit data-input register, and then writing the two bytes of data into the dynamic memory one byte at a time. The MPU reads the data from the CM by first performing two separate memory reads from the byte-oriented dynamic memory, storing the two bytes in the 16-bit data-output register, and then writing the 16-bit data into the CC. The CM signals the Controller that the memory operation is complete by setting the Memory Reference Complete(MRC) signal to a logic "0". The MRC signal is reset to a logic "1" prior to the next memory reference.

The 128K-bytes of the CM's RAM are composed of 16 64K-bit dynamic memory chips. The memory is structured into two 64K-byte memory banks, with each bank composed of 8 64K-bit chips. The Intel 8207 Dual-Port Memory Controller[Int83] and additional logic control the operation of the CM. The In8207 provides all the timing signals for memory reads, writes and refreshes. The lowest 17 bits of the PC's address is buffered using three 74ls244[Tex75] octal buffers. The MMAP's address register is 17-bits wide and is built using 6 74ls173[Tex75] 4-bit registers. The outputs of the MMAP's address register and the PC's address buffer are connected, forming the internal address bus used by the CM. Only one output is enabled at a given time. The 16-bit data registers connected to the MMAP's internal buses are each made of 4 74ls173 4-bit registers. The data-input and the data-output registers that are connected to the PC's data bus are made of 2 74ls373[Tex75] octal registers.

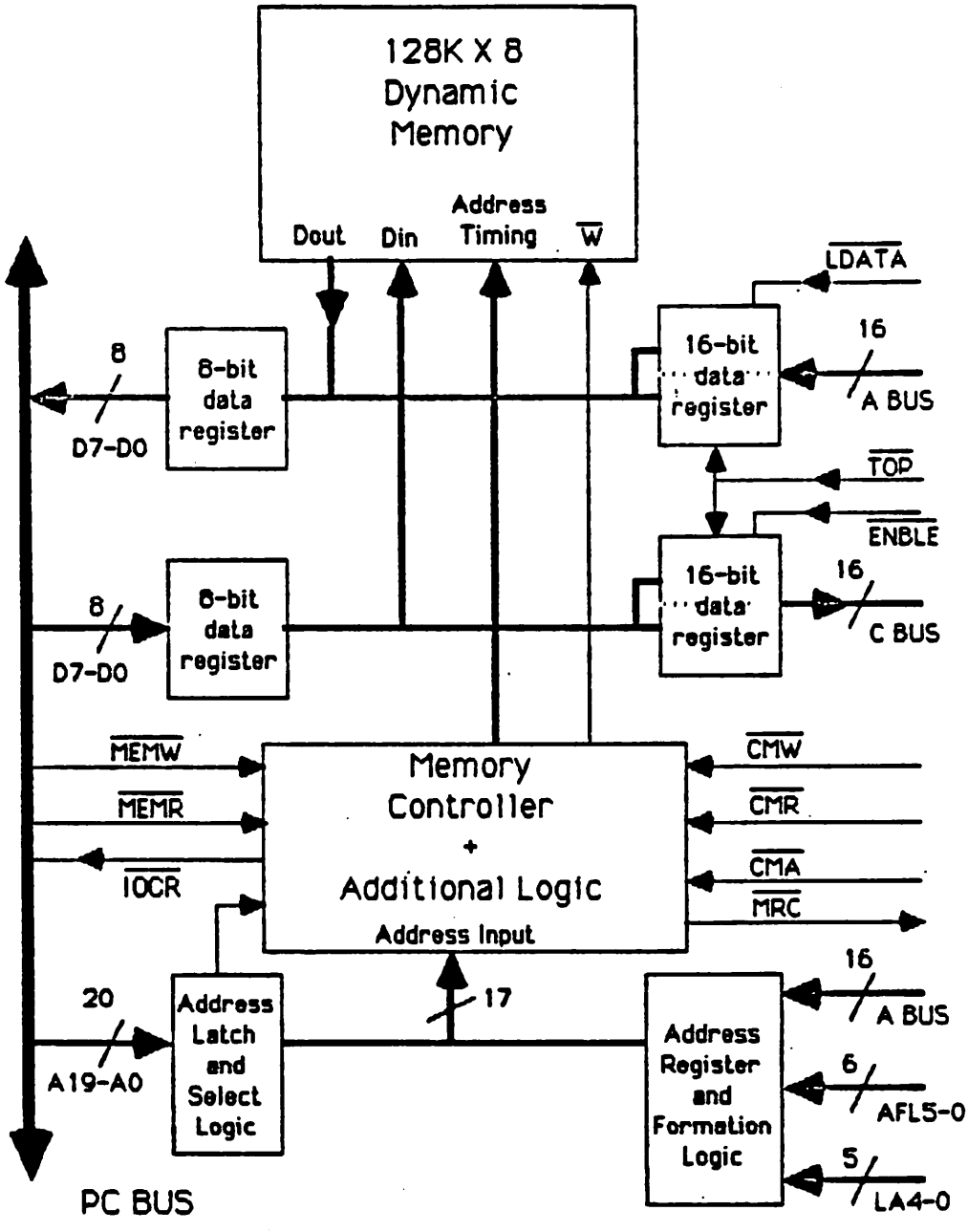
Eighteen control points within the CM are operated by the Controller. The seven AFL signals control the formation of the MMAP's address from the combinational logic and the five LA signals enable the loading of the address in the MMAP's address register. The $\overline{\text{CMA}}$, $\overline{\text{CMW}}$ and $\overline{\text{CMR}}$ are connected to the In8207. All three are active low. The $\overline{\text{CMA}}$ requests a memory operation. The $\overline{\text{CMR}}$ and $\overline{\text{CMW}}$ are the read and write request signals respectively The $\overline{\text{TOP}}$ signal differentiates between the high and low 8-bits of the 16-bit data-input and data-output registers. The $\overline{\text{TOP}}$ signal specifies which byte is to be read from the data-input register during a memory write, and which byte of the data-output register is to receive the data during a memory read. The $\overline{\text{LDATA}}$ signal enables the loading of the data-input register, and the $\overline{\text{ENBLE}}$ signal enables the output of the data-output register. Both signals are active low.



Dual-Access Coefficient Memory
Figure 6.11



IBM PC-XT Memory Configuration
Figure 6.12



Block Diagram of the Coefficient Memory
Figure 6.13

6.2.6. MMAP Clocking

The MMAP's Controller, FPU and CC operate synchronously using a common clock signal and all operations are executed on the rising clock edge. The maximum designed frequency of the MMAP's clock is 10MHz, which can only be achieved when high-speed($\leq$ 50ns delay time) fuse-linked Programmable ROMs(PROMs) are used in the Controller. The frequency of the MMAP's clock is then limited by the speed of the Weitek ALU and multiplier. If PROMs with access times $>$ 50ns are used, the maximum clock frequency is limited by the Controller's microinstruction access time.

Erasable PROMs (EPROMs) are used instead of the fuse-linked PROMs. EPROMs are more practical for use in the development of a prototype. EPROMs can be erased and reprogrammed, while fuse-linked PROMs can only be programmed once, and, in general, the cost of fuse-linked PROMs are prohibitive in a development project. 450ns access-time EPROMs are used.

The 450ns EPROMs restrict the maximum clock frequency to 2Mhz. Instead of using a clock signal generated external to both the IBM PC-XT and MMAP, the MMAP's clock signal is derived from the IBM PC's clock signal present on the PC's bus. The clock frequency of the IBM PC-XT used with the prototype is 4.77 Mhz. The PC's clock signal is divided by 4, providing a 1.2Mhz clock frequency for the MMAP (reducing the PC's clock by 2 would result in a frequency of 2.38Mhz, which is greater than the maximum allowed clock frequency).

Communication between the CM and both the IBM PC-XT and remainder of the MMAP is performed asynchronously. The CM uses a clock separate from the remainder of the MMAP, and the CM clock frequency is equal to the frequency of the IBM PC-XT clock.

6.3. Organization and Access of Data in the Coefficient Memory

The organization of data within the CM, and the access of the data stored in the CM by both the MMAP and IBM PC-XT are described.

6.3.1. The Storage and Access of Model Data Within the Coefficient Memory

The data that is stored in the CM is listed in Chapter 5. All floating-point numbers are stored in single-precision format, requiring 4 bytes of storage per floating-point number. In this design there are a maximum of 16 measured values of V_{ds}, $V_{ds\,meas}$, and 16 measured values of V_{gs}, $V_{gs\,meas}$, for each model. 3K bytes are used to store the values of I_{ds}, G_{ds} and G_{gs} for the 256 combinations of measured voltages. The 32 measured voltages require 128 bytes of memory, the 30 values of the inverse of the difference between adjacent measured voltages require 248 bytes of memory, and the 64 (4 coefficients × 16 measure V_{ds}) V_{sb}-dependent parameters require 256 bytes of memory. In addition, 256 1-byte voltage-dependent pointers, 128 each for V_{ds} and V_{gs}, and 3 floating-point constants are stored. The function of the pointers is described in Section 6.3.3. The following table summarizes the data storage requirements.

Data Stored	# bytes
I_{ds}	1024
G_{ds}	1024
G_{gs}	1024
V_{ds}	64
V_{gs}	64
$(\Delta V_{ds})^{-1}$	60
$(\Delta V_{gs})^{-1}$	60
V_{sb}-Dependence Parameters	256
V_{ds} Pointers	128
V_{gs} Pointers	128
Fpt. Constants	12

Table 6.2

As described in Chapter 5, the CM is partitioned into equal sections, each of which stores the data for a single transistor model. Within one section the memory is further partitioned into four subsections: three subsections store the values of I_{ds}, G_{ds} and G_{gs}

respectively, and the fourth subsection stores the remainder of the model data. The I_{ds}, G_{ds} and G_{gs} are stored in sequential memory locations within their subsection. The storage order of the data is illustrated by Figures 6.14 and 6.15. In Figure 6.14, the data are represented by points in the plane with V_{ds} as the x-axis and V_{gs} as the y-axis. The points correspond to the data at the different combinations of $V_{ds\ meas}$ and $V_{gs\ meas}$. The storage of the data illustrated in Figure 6.14 is given in Figure 6.15. The first values stored of I_{ds}, G_{ds}, and G_{gs} correspond to the combination of the first (lowest value) $V_{ds\ meas\ 0}$ and the first (lowest value), $V_{gs\ meas\ 0}$. The next value stored corresponds to the combination of the second $V_{ds\ meas\ 1}$ and the first $V_{gs\ meas\ 0}$. The ordering continues through to the combination of the last $V_{ds\ meas\ 15}$ and the first $V_{gs\ meas\ 0}$. The above is repeated using the remaining values of $V_{gs\ meas}$ in order of smallest to largest value.

With 16 measured values of both V_{ds} and V_{gs}, the offset from relative to the first location within the subsection of either I_{ds}, G_{ds} of G_{gs} can be given as

$$\text{offset} = 16\,k + j, \text{ for } 0 \leqslant j < 16 \text{ and } 0 \leqslant k < 16. \qquad (6.1)$$

where both j and k are integers. $V_{ds\ meas\ j}$ and $V_{gs\ meas\ k}$ are ordered in increasing value. For example, the I_{ds}, G_{ds}, and G_{gs} corresponding to $(V_{ds\ meas\ 2}, V_{gs\ meas\ 1})$ is accessed from the appropriate subsection using an offset of value 18.

(6.1) can be represented in an alternative fashion when the offset term is used directly as part of the address. Since j and k are within the limits given in (6.1) , they can each be represented by a 4 bit binary number. Multiplying k by 16 is the same as adding four zeros to the right of the binary representation of k. The offset can then be represented as a 8-bit binary number, where the four most-significant bits are the binary representation of k and the four least-significant bits are the binary representation of j. With j=2 and k=1, the offset, in binary, is 00010010, which is 18 decimal. The binary representations of j and k are referred to as the V_{ds} and V_{gs} pointers respectively.

As described in Chapter 4, the model evaluation uses the values of I_{ds}, G_{ds} and G_{gs} from 4 data points to calculate the device's operating-point information. For the given

values of V_{ds} and V_{gs}, each voltage is bound by two measured voltages.

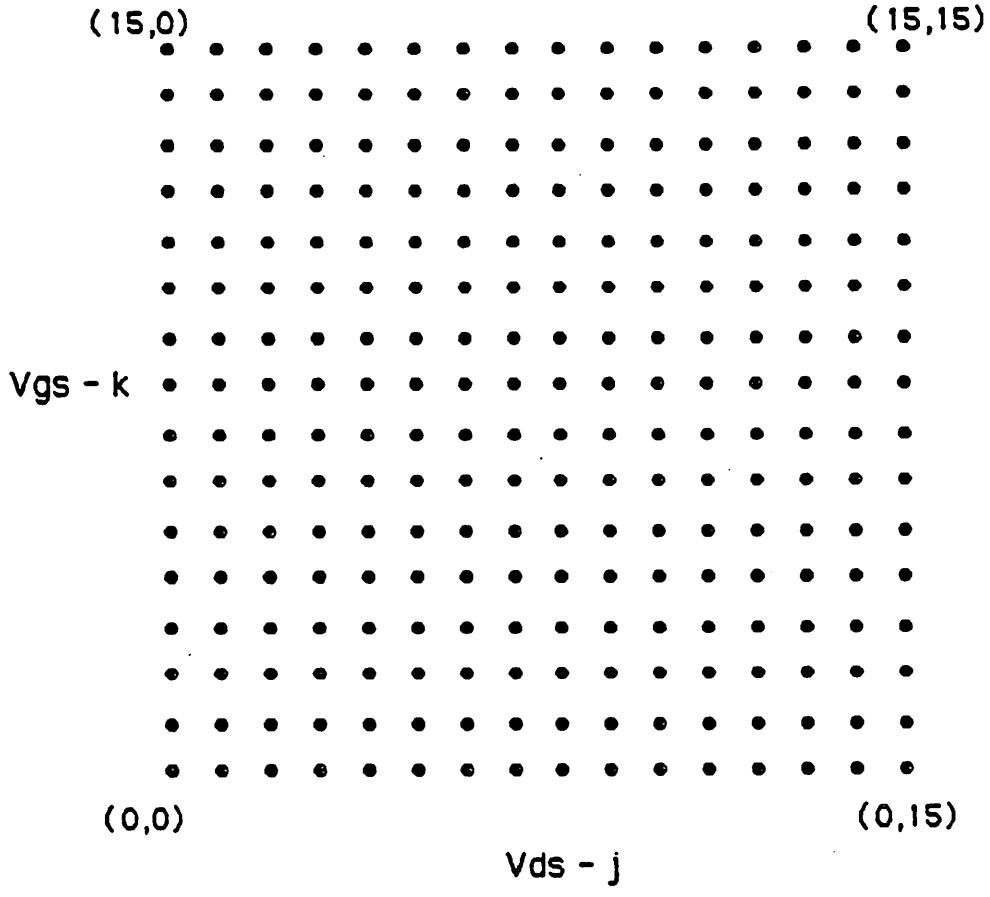
$$V_{ds\,j} \leqslant V_{ds} < V_{ds\,j+1} . \tag{6.2}$$

and

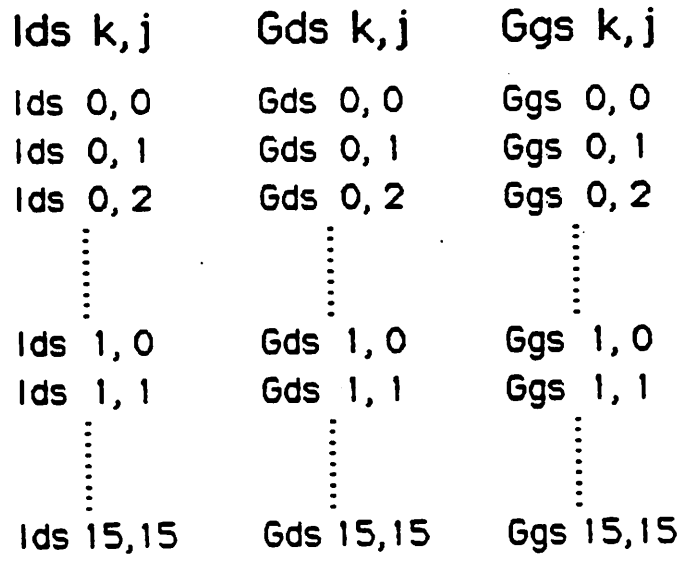
$$V_{gs\,k} \leqslant V_{gs} < V_{gs\,k+1} . \tag{6.3}$$

j and k are restricted to the range given in Equation (6.1), and their binary values are the V_{ds} and V_{gs} pointers. The values of I_{ds}, G_{ds} and G_{gs} for the data points $(V_{ds\,j}, V_{gs\,k})$, $(V_{ds\,j+1}, V_{gs\,k})$, $(V_{ds\,j}, V_{gs\,k+1})$, and $(V_{ds\,j+1}, V_{gs\,k+1})$ are accessed. The offset for the four points are:

$(V_{ds\,j}, V_{gs\,k})$	k	j
$(V_{ds\,j}, V_{gs\,k+1})$	k+1	j
$(V_{ds\,j+1}, V_{gs\,k})$	k	j+1
$(V_{ds\,j+1}, V_{gs\,k+1})$	k+1	j+1



Storage of I_{ds}, G_{ds} and G_{gs}
Figure 6.14



Storage of I_{ds}, G_{ds} and G_{gs}
Figure 6.15

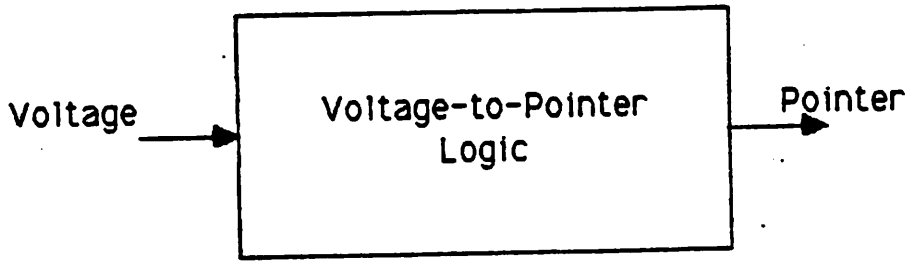
6.3.2. Generation of the Voltage-Dependent Pointers

As described in Chapter 4, the data required for a transistor evaluation is dependent on the values of V_{ds} and V_{gs}. This dependence is represented by the V_{ds} and V_{gs} pointers introduced in the previous section. This section describes the generation of these pointers from the given voltage values.

6.3.2.1. Background

There are several methods by which the pointers can be generated. A straightforward approach is to directly map the voltage into a pointer using combinational logic, as shown in Figure 6.16. The advantage of this approach is that the operation can be performed in one clock cycle. The primary disadvantage to this approach is that the values of the measured voltages, 16 in the case of the prototype, are restricted to always being the same value. A change in the value of a measured voltages would require a change in the combinational logic generating the pointer.

A more general approach is to derive the pointer value by doing repeated comparisons of the voltage value with the measured voltages. The voltage value would be compared to the different measured voltages until the valid range is determined. For example,



Direct Transformation of Voltage into a Pointer
Figure 6.16

if

$$V_{ds} \geq V_{ds\ meas\ 3} \text{ and } V_{ds} < V_{ds\ meas\ 4}. \tag{6.4}$$

the pointer would be 3. This approach is very general. but would require additional hardware to perform the floating-point comparison and use many additional memory references in accessing the values of measured voltage from the CM to use in the comparison.

6.3.2.2. Overview of Two-Step Procedure

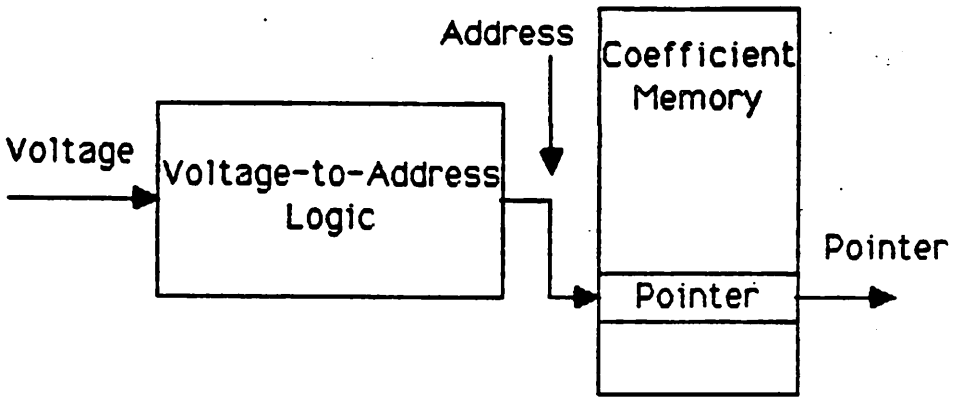
A two-step procedure. illustrated in Figure 6.17. is used to generate the voltage-dependent pointers in the prototype. The floating-point number representing the voltage is first transformed into a unique offset address. This transformation is performed the combinational logic within the CM. The offset is then used to access the appropriate pointer for the transistor model stored in the CM.

The first step in the procedure. the transformation of a voltage into a unique address. is similar to the direct approach since the transformation is "hardwired" directly into the combinational logic within the CM. Unlike the direct approach. the result is not a pointer. but an address in the CM containing a pointer. The intermediate address represents a possible measured voltage. referred to as an allowed measured voltage. There are 128 allowed measured voltages. and they comprise the set of voltages from which the 16 measured voltages can be taken.

$$V_{ds\ measured} \in V_{ds\ allowed} \tag{6.5}$$

$$V_{gs\ measured} \in V_{gs\ allowed} \tag{6.6}$$

The pointer stored at this address references the nearest $V_{ds\ meas}(V_{gs\ meas})$ that does not exceed V_{ds} (V_{gs}).



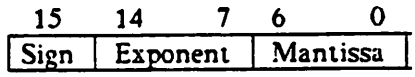
Generate Voltage-Dependent Pointers
Figure 6.17

6.3.2.3. Allowed Measured Voltages

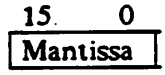
The single-precision, floating-point number representing either V_{ds} or V_{gs} is mapped into an address which represents the allowed measured voltage value that is nearest to the floating-point number without exceeding it. The format of the floating-point number is given below.

31	30 — 23	22 — 0
Sign	Exponent	Mantissa

The most significant bit of the floating-point number is the sign bit, the following 8 bits store the exponent and the remaining 23 bits store the mantissa. A leading 1 is implicit in the mantissa, giving the mantissa 24 bits of accuracy. Since the data paths and storage of the MPU is 16-bits wide, the 32-bit wide floating-point numbers are represented by 2 16-bit words.



Most-Significant Word



Least-Significant Word

The 16 most-significant bits of the floating-pint number, sign bit, exponent field and 7 leading bits of the mantissa are stored in one word, and the 16 least-significant bits of the mantissa are stored in the other.

The empirical model described in Chapter 4 represents the MOS transistor only over the range of measured voltages. The set of allowed measured voltages must span a wide range, allowing a transistor to be represented over a large voltage range. In addition, exponential variations in transistor current can occur in the device's subthreshold region of operation. To better represent the subthreshold behavior, the differences between adjacent measured voltages must be small in this region to better approximate the exponential behavior by a cubic polynomial.

A further restriction is imposed on the values of measured voltages. As described in Chapter 4, the inverse of the difference between adjacent measured voltages must be calculated exactly since single-precision, floating-point arithmetic is used. As shown in Chapter 4, an error in this computation can produce discontinuities in the transistor characteristics which can result in the nonconvergence of the circuit-simulation program.

The floating-point number's exponent field and the first 4 bits of the mantissa, not including the implicit leading 1, are used to derive the voltage-dependent component of the CM's address. The allowed measured values of V_{ds} and V_{gs} are restricted to be between 0 and 256 volts.

$$0.0 \leqslant \text{Measured Voltages} < 256.0 \tag{6.7}$$

The total range is divided into eight sub-ranges, and in each sub-range there are 16 allowed voltages. The sub-ranges are given in Table 6.3.

Voltage Sub-Range(volts)					Increment(volts)	Address(Voltage)
0.0	$\leqslant$	v	<	2.0	0.125	111VVVV
2.0	$\leqslant$	v	<	4.0	0.125	000VVVV
4.0	$\leqslant$	v	<	8.0	0.25	001VVVV
8.0	$\leqslant$	v	<	16.0	0.5	010VVVV
16.0	$\leqslant$	v	<	32.0	1.0	011VVVV
32.0	$\leqslant$	v	<	64.0	2.0	100VVVV
64.0	$\leqslant$	v	<	128.0	4.0	101VVVV
128.0	$\leqslant$	v	<	256.0	8.0	110VVVV

Table 6.3

The Address(Voltage) is the lowest 7 bits of the Offset field. The first 3 bits of the Address(Voltage) specify the range, and the remaining 4 bits(VVVV as used in Table 6.3) represent one of the 16 allowed voltages within that range. The size of the increments increase as the voltage increases. This results in a fine granularity of allowed measured voltages at small voltages, and increasing granularity as the voltage increases. The minimum value of allowed measured voltage is 0.0 and the maximum value of allowed measured voltage is 248.0. For voltages that are less than 0.0 the combinational logic generates the Address(Voltage) representing 0.0 volts, and for voltages that are greater than 248.0 the combinational logic generates the Address(Voltage) representing 248.0 volts.

For example, over the sub-range

$$2.0 \leqslant v < 4.0 . \tag{6.8}$$

the allowed voltages are in increments of 0.125 volts. The allowed voltages and corresponding Address(Voltage) are given in the following table.

Allowed Measured Voltage	Address(Voltage)
2.0	0000000
2.125	0000001
2.25	0000010
2.375	0000011
2.5	0000100
2.625	0000101
2.75	0000110
2.875	0000111
3.0	0001000
3.125	0001001
3.25	0001010
3.375	0001011
3.5	0001100
3.625	0001101
3.75	0001110
3.875	0001111

Table 6.4

In this sub-range the allowed voltages begin with 2.0 volts and continue to 3.875 volts in 0.125 volt increments. The first 3 bits of the Address(Voltage) component of the Offset field is 000, the value for the sub-range given in Table 6.3. The remaining 4 bits begin at a value of 0000, corresponding to a voltage of 2.0. For each 0.125 increase in voltage, the remaining 4 bits are incremented by 1. For a voltage value of 2.7, the combinational logic generates an Address(Voltage) of 0000101. This address corresponds to the allowed measured voltage of 2.625 volts, the allowed voltage that is nearest to 2.7 without exceeding it.

The inverse of the difference between adjacent measured voltages, $(\Delta V)^{-1}$, are precalculated and stored as model data. As stated in Chapter 4, it is necessary for these values to be exact in order to guarantee continuity of the transistor output characteristics. If the floating-point number representing (ΔV) is an integral power of 2,

$$\Delta V = 1 \times 2^{n}. \tag{6.9}$$

then its inverse, $(\Delta V)^{-1}$, is

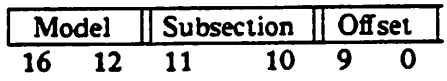
$$(\Delta V)^{-1} = 1 \times 2^{-n}. \tag{6.10}$$

which can be stored exactly by a single-precision, floating-point number. The allowed

measured voltages, as represented in Table 6.3, allow for the difference between adjacent measured voltages to be equal to an integral power of 2, and, therefore, $(\Delta V)^{-1}$ can be represented exactly. For example, if the first measured voltage is 0.0 and the second is 0.25, their difference is 0.25 (2^{-2}). The inverse of 0.25 is 4 (2^{2}), which can be stored exactly.

6.3.3. Coefficient-Memory Address

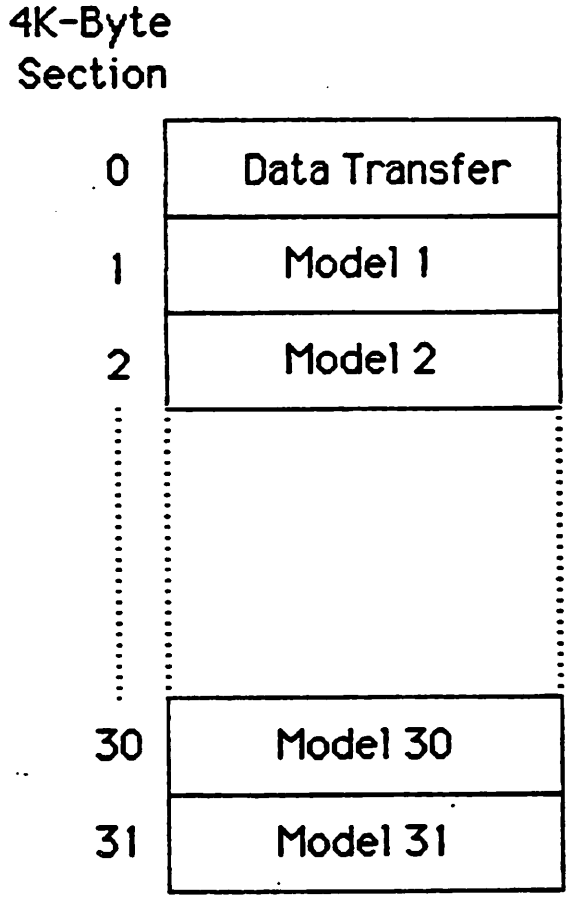
The CM's address is 17 bits wide, allowing for the complete addressing of the 128K bytes of CM. As described in Chapter 5, the address is composed of the Model, Subsection and Offset fields. The Model field comprises the highest 5 bits, the Subsection field is the next 2 bits, and the Offset field is the lowest 10 bits.



Coefficient Memory Address

The 5-bit Model field provides the address for 32 4K-byte sections. Data for a single transistor model is stored in a 4K-byte section.

The CM functions as the interface between the MMAP and IBM PC-XT. The first 4K-byte section (Model field set to 00000), as shown in Figure 6.18, is used by the PC to transfer data and commands to the MMAP. As described in Chapter 2, the IBM PC-XT (host machine) sends the MOS transistor's terminal voltages, scale factor and model pointer to the MMAP, the IBM PC-XT then signals the MMAP to begin evaluation, the MOS transistor's current and conductances are then calculated, and the results are returned to the IBM PC-XT. The input data and output results are transferred through the lowest 4K-byte section. 1K-byte of the the lowest 4K-byte section is used for each transistor being evaluated. Given in Table 6.5 is the configuration of the lowest 4K-byte section.



Coefficient Memory as the Interface to the MMAP
Figure 6.18

CM Memory Map for Data Transfer and Commands			
PC Address(Hex)			Data
0x80000			Model Reference Transistor 1
0x80003	-	0x80007	V_{ds} Transistor 1
0x80008	-	0x8000B	V_{gs} Transistor 1
0x8000C	-	0x8000F	V_{sb} Transistor 1
0x80010	-	0x80013	Channel Scale Transistor 1
0x80014	-	0x80017	I_{ds} Transistor 1
0x80018	-	0x8001B	G_{ds} Transistor 1
0x8001C	-	0x8001F	G_{gs} Transistor 1
0x80020	-	0x80023	G_{sb} Transistor 1
0x80024	-	0x803FF	Unused
0x80400			Model Reference Transistor 2
0x80403	-	0x80407	V_{ds} Transistor 2
0x80408	-	0x8040B	V_{gs} Transistor 2
0x8040C	-	0x8040F	V_{sb} Transistor 2
0x80410	-	0x80413	Channel Scale Transistor 2
0x80414	-	0x80417	I_{ds} Transistor 2
0x80418	-	0x8041B	G_{ds} Transistor 2
0x8041C	-	0x8041F	G_{gs} Transistor 2
0x80420	-	0x80423	G_{sb} Transistor 2
0x80424	-	0x807FF	Unused
0x80800			Model Reference Transistor 3
0x80803	-	0x80807	V_{ds} Transistor 3
0x80808	-	0x8080B	V_{gs} Transistor 3
0x8080C	-	0x8080F	V_{sb} Transistor 3
0x80810	-	0x80813	Channel Scale Transistor 3
0x80814	-	0x80817	I_{ds} Transistor 3
0x80818	-	0x8081B	G_{ds} Transistor 3
0x8081C	-	0x8081F	G_{gs} Transistor 3
0x80820	-	0x80823	G_{sb} Transistor 3
0x80824	-	0x80BFF	Unused
0x80C00			Model Reference Transistor 4
0x80C03	-	0x80C07	V_{ds} Transistor 4
0x80C08	-	0x80C0B	V_{ds} Transistor 4
0x80C0C	-	0x80C0F	V_{gs} Transistor 4
0x80C10	-	0x80C13	Channel Scale Transistor 4
0x80C14	-	0x80C17	I_{ds} Transistor 4
0x80C18	-	0x80C1B	G_{ds} Transistor 4
0x80C1C	-	0x80C1F	G_{gs} Transistor 4
0x80C20	-	0x80C23	G_{sb} Transistor 4
0x80C24	-	0x80FFD	Unused
0x80FFE			Check Completion
0x80FFF			Begin Evaluation

Table 6.5

The MMAP is signaled to begin operation when the IBM PC-XT writes to location 0x80FFF, the last byte in the lowest 4K-byte section. The MMAP notifies that the current transistor evaluation is finished by setting byte 0x80FFE. The IBM PC-XT polls this memory location to check if the MMAP has completed the evaluation.

The remaining 31 sections are used to store model data. As described in Chapter 5, each 4K-byte section storing model data is logically partitioned into 4 equal-sized subsections. The 256 discrete values of I_{ds}, G_{ds} and G_{gs} are stored in their individual 1K-byte subsection. The remainder of the data(refer to Table 6.2) is stored in the remaining 1K-byte subsection. The 2-bit wide Subsection field differentiates between the 4 1K-byte subsections. The memory map of a 4K-byte section of the CM used to store model data is given in Table 6.6.

CM Memory Map for Model Data			
Subsection + Offset Address(Hex)			Data Stored
0x000	-	0x3FF	I_{ds}
0x400	-	0x7FF	G_{ds}
0x800	-	0xBFF	G_{gs}
0xC00	-	0xC3F	V_{ds}
0xC40	-	0xC7F	V_{gs}
0xC80	-	0xCBE	$(\Delta V_{ds})^{-1}$
0xCC0	0xCBF - 0xCFF	0xCFE	unused $(\Delta V_{gs})^{-1}$ unused
0xD00	-	0xDFF	V_{sb}-Dependent Parameters
0xE00	-	0xE7F	V_{ds} Pointers
0xE80	-	0xEFF	V_{gs} Pointers
0xF00	-	0xF03	± 1.0 Constant
0xF04	-	0xF07	2.0 Constant
0xF08	-	0xF0B	3.0 Constant

Table 6.6

The floating-point values of 2.0 and 3.0 are stored and are required in the evaluation of the derivative of the polynomial equations. Either +1.0 or -1.0 is also stored. A value of +1.0 is stored if the model data is for a n-channel device, and a value of -1.0 is stored if

the model data is for a p-channel device. The values of the terminal voltages and scale factor[3] sent to the MMAP are multiplied by this term. Therefore, if the device is a p-channel transistor the appropriate change of sign is performed.

The values of I_{ds}, G_{ds} and G_{gs} are stored as 4-byte, single-precision, floating-point numbers. The configurations of the CM's address for accessing them are given in Figure 6.19a.

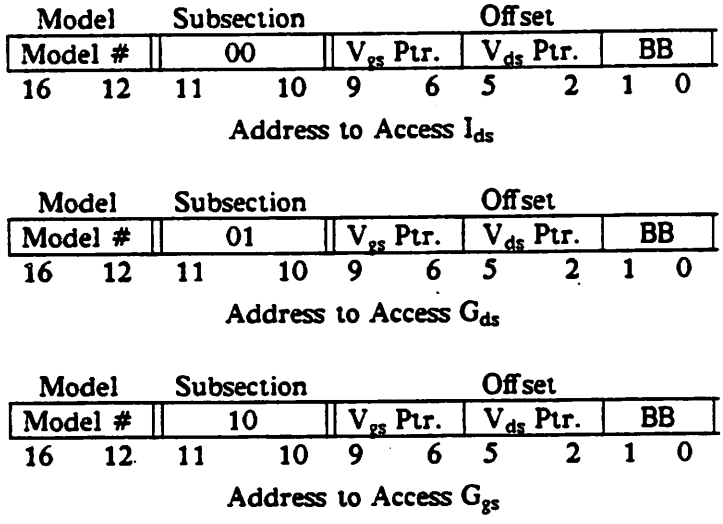


Figure 6.19a

The most-significant 8 bits of the Offset field consists of two 4-bit voltage-dependent terms which can address 256 floating-point numbers. The pointers are dependent on V_{ds} and V_{gs} respectively. The 2 lowest bits in the Offset field(BB) specify the byte of the floating-point number.

The discrete values of V_{ds}, V_{gs}, $(\Delta V_{ds})^{-1}$, and $(\Delta V_{gs})^{-1}$ are stored as single-precision floating-point numbers. The configurations of the CM's address for accessing them are given in Figure 6.19b

[3] Multiplying the scale factor by ± 1.0 results in the correct sign for the current and derivatives.

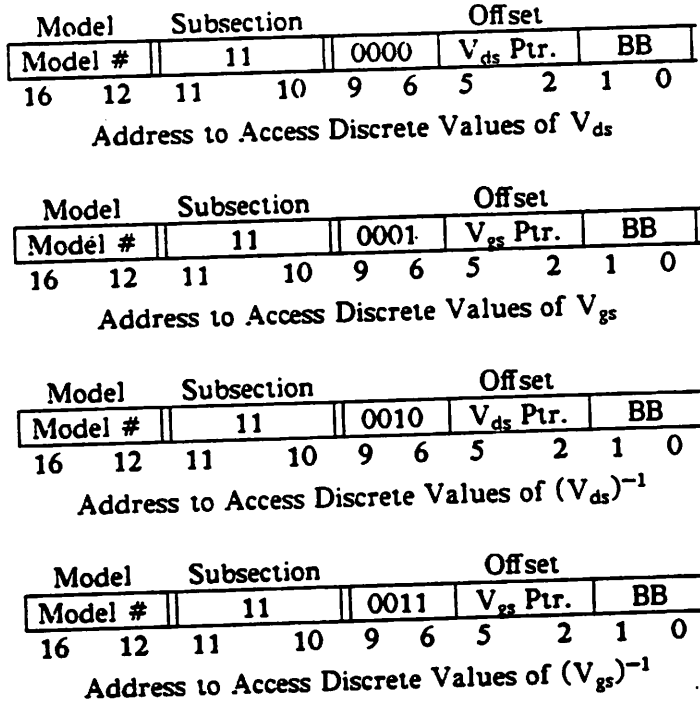
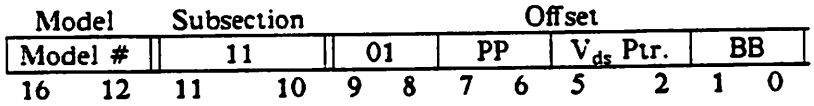


Figure 6.19b

When accessing V_{ds} and $(\Delta V_{ds})^{-1}$, the most-significant 4 bits of the Offset field define which of the floating-point entries is to be accessed. The next 4 bits contain the V_{ds} voltage-dependent pointer, and the 2 lowest bits(BB) specify the byte of the floating-point word. When accessing V_{gs} and $(\Delta V_{gs})^{-1}$, the most-significant 4 bits of the Offset field define which of the floating-point entries is to be accessed. The next 4 bits contain the V_{gs} voltage-dependent pointer, and the two lowest bits(BB) specify the byte of the floating-point word.

For each measured value of V_{ds} there are 4 V_{sb}-dependence parameters stored as single-precision floating-point numbers. The configuration of the CM's address for accessing them is given in Figure 6.19c.

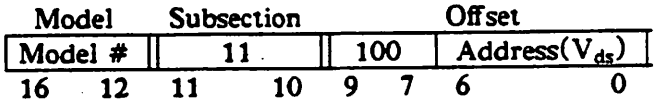


Address to Access V_{sb} Parameters

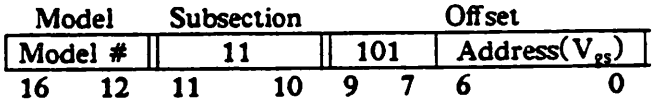
Figure 6.19c

In this case, the 2 highest bits of the Offset field specify the V_{sb}-dependent terms, and the next 2 bits of the Offset field specify one of the 4 parameters(PP). The next 4 bits contain the V_{ds} voltage-dependent pointer, and the two lowest bits(BB) specify the byte of the floating-point word.

Each pointer requires 1 byte of storage. The configurations of the CM's address for accessing the V_{ds} and V_{gs} pointers are given in Figure 6.19d.



Address to Access V_{ds} Pointers



Address to Access V_{gs} Pointers

Figure 6.19d

In this case, the 2 highest bits of the Offset field specify the pointers, the next bit differentiates between the V_{ds} or V_{gs} pointers, and the remaining 7 bits of the Offset field are generated from either the V_{ds} or V_{gs} voltages.

The addresses used to access the floating-point constants are given in Figure 6.19e.

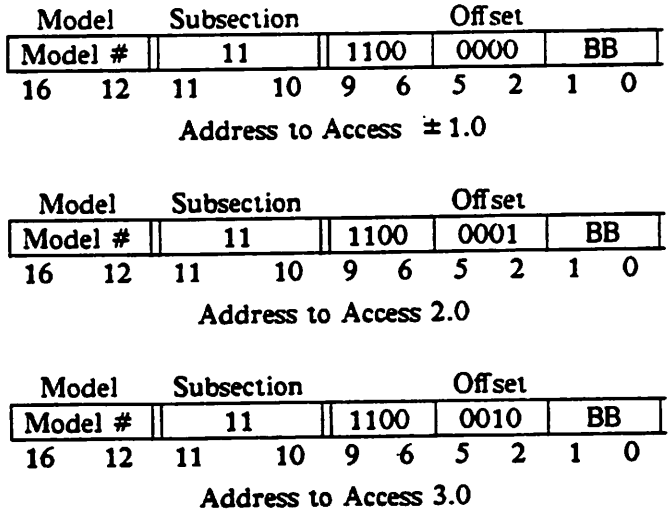


Figure 6.19e

The two lowest bits(BB) specify the byte of the floating-point word.

As described earlier, the CM's address register is composed of 6 74ls173 4-bit registers. The Model field composed of 2 4-bit registers, the Subsection field is composed of a single 4-bit register, and the Offset field is composed of 3 4-bit registers. The 3 registers comprising the Offset field can be loaded independent of each other.

6.4. MMAP Operation

The movement of data within the MMAP's Processor is described by the eight data-transfer operations listed below and illustrated in Figure 6.20.

(1) CC's dual-port register→ FPU

(2) CC's dual-port register→ Data-Input register

(3) CC's dual-port register→ CM's address register

(4) CC's static RAM → CC's dual-port register

(5) FPU → CC's static RAM

(6) Data-Output register → CC's static RAM

(7) CM → Data-Output register

(8) Data-Input register → CM

The first entry in each of the data-transfer operations is the datum's source and the second is the datum's destination. For example, data-transfer operation (5) designates the transfer of data from the output of the FPU to a specified memory location in the CC's static memory. The floating-point operations are associated with the data-transfer operation (1), the transfer of data to the FPU from the CC's dual-port register.

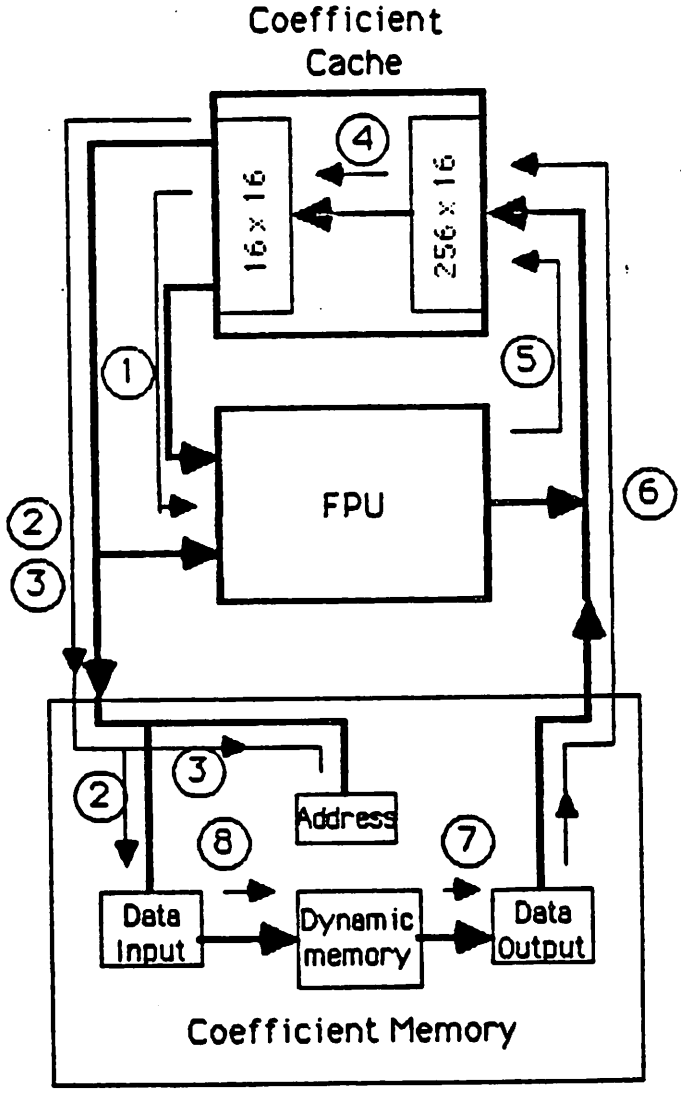
Several of the data-transfer operations can be performed concurrently. For example, data can be transferred from the CC's dual-port register to the input of the FPU at the same time as data is being written into the CC's static memory from the output of the FPU. Table 6.7 lists the operations that can be performed concurrently.

Operation	1	2	3	4	5	6	7	8
1		*	*		*	*	*	*
2	*		*		*	*	*	
3	*	*			*	*	*	*
4							*	*
5	*	*	*				*	*
6	*	*	*				*	*
7	*	*	*	*	*	*		
8	*		*	*	*	*		

Table 6.7

The numbers in the eight rows and eight columns represent the eight data-transfer operations. An asterisk(*) in a row of the table indicates that the operation associated with that column can be performed at the same time as the row operation. Concurrent data-transfer operations using the same data path require the transfer of the same data.

Operations 1-6 are executed in a single cycle of the MMAP clock. Operations 7 and 8, the CM read and write operations, are performed asynchronously. Provided there is no contention for the DRAM memory from either the PC-XT or memory refresh, the CM read and write operations require 4 PC-XT clock cycles.



Data-Transfer Operations
Figure 6.20

6.5. Microprogramming the Prototype MMAP

The two variations of the empirical MOS transistor model are described in Chapter 4. Both require the same data for their calculations, but differ only in their interpolation method. As described in Chapter 5, the procedure defining the transistor operation is stored directly as microinstructions in the Controller's microstore. The programming of the MMAP's microinstructions are presented in this section.

6.5.1. Overview

As described earlier, the lowest 4K bytes of the CM are used as the interface between the IBM PC-XT and the MMAP. The input data for a transistor evaluation are sent by the IBM PC-XT to the lowest section of the CM. Once the transfer of data by the PC to the MMAP is complete, the IBM PC-XT signals the MMAP to begin the transistor evaluation. The transistor data and necessary model data are read in from the CM into the CC. The floating-point computations are performed by the data path comprising the FPU and CC. The CC is used as a set of registers for the storage of the intermediate results of the floating-point computations, in addition to providing storage for the transistor and model data. After the MMAP performs the evaluation, the results are returned to the specified memory locations in the CM. The results are then available for use by the IBM PC.

6.5.2. Description

A description of the microprogram is given in this section. First, the organization of the CC's static memory is presented. The partitioning of the microprogram into seven steps is then presented, and the operation performed in during each step is described. The data stored in the CC after completion of each step is is used to illustrate the operations performed during each step.

As described in Section 6.2, the CC contains a 256 × 16 static memory. The static memory is logically partitioned as four 64 × 16 sections, one for each transistor. Single-precision, floating-point numbers are stored in two consecutive memory locations, with the most-significant 16 bits stored first and the least-significant 16 bits stored in the following location.

The procedure followed for the transistor evaluation can be partitioned into seven steps. The seven steps are listed below and are performed in sequence. A more detailed description of each step follows.

(1) Access the transistor data(differential terminal voltages, transistor dimensions and model reference pointer) from the CM. This data is originally sent by the PC-XT.

(2) Use the value of V_{ds} to access the the model data dependent only on V_{ds}. This data includes the voltage-dependent pointer, the nearest measured voltage, inverse differential voltage and V_{sb}-dependence parameters. Also read in the three floating-point constants.

(3) Multiply V_{ds}, V_{gs}, V_{sb} and S_c by ± 1, and then compute the effective value of V_{gs}, V_{gse}. In addition, compute the values of $\frac{\partial V_{gse}}{\partial V_{sb}}$ and $\frac{\partial V_{gse}}{\partial V_{ds}}$

(4) Use the value of V_{gse} to access the model data dependent only on V_{gs}. These are the voltage-dependent pointer, the nearest measured voltage and inverse differential voltage.

(5) Access the discrete values of I_{ds}, G_{ds} and G_{gs}. Four values of each are read.

(6) Calculate I_{ds}, G_{ds}, G_{gs} and G_{sb} using either the linear or cubic interpolation methods. Multiply the results by S_c.

(7) Store the results in the Coefficient Memory for use by the IBM PC-XT.

A more detailed description of the steps are given below. The data stored for one transistor in the CC is given in Table 6.8. Table 6.8 gives the data at every memory location at

the completion of each step.

Step 1

The first step in the procedure is to access the data sent by the PC to the MMAP. As given earlier, this data consists of a model pointer (Model #), the V_{ds}, V_{gs}, and V_{sb} terminal voltages, and S_c. The IBM PC-XT first sends the input data to the MMAP and then signals the MMAP to begin a transistor evaluation. The MMAP then reads the input data from the CM, which is acting as the interface for this operation, and stores the data in the CC. The exact memory locations within the CM are given in Table 6.5.

Step 2

Once the transistor data is read into the CC, the data is then used to access the three constants and the model data that is dependent only on V_{ds}. First, the three constants are read in from the CM. Then, the V_{sb} voltage-dependent pointer, j, for the model is accessed from the CM. The pointer is then used to access the appropriate measured voltage, $V_{ds\,j}$, and inverse differential voltage, $(\Delta V_{ds})^{-1}$. Also, the eight V_{sb} terms are read in corresponding to $V_{ds\,j}$ and $V_{ds\,j+1}$.

Step 3

The values of the terminal voltages and scale factor are multiplied by ± 1. The value of δV_{ds} is then calculated, where δV_{ds} is equal to $V_{ds} - V_{ds\,j}$. Finally, the value of V_{gse} and the partial derivatives of V_{gse}, $\frac{\partial V_{gse}}{\partial V_{sb}}$ and $\frac{\partial V_{gse}}{\partial V_{ds}}$, are calculated.

Step 4

The value of V_{gse} is first used to access the voltage-dependent pointer, k, from the CM. The value of k is stored in the CM. k is then used in the formation of the address used to access the appropriate measured voltage, $V_{gs\,k}$, and inverse differential voltage,

$(\Delta V_{gs})^{-1}$ from the CM and store them in the CC.

Step 5

The combination of the j and k pointers is used to access the discrete values of I_{ds}, G_{ds} and G_{gs} required for the evaluation. Four values of each the I_{ds}, G_{ds} and G_{gs} are accessed from the CM and stored in the CC.

Step 6

The value of δV_{gs} is first calculated, where $\delta V_{gs} = V_{gs} - V_{gs\,k}$. The values of the current and conductance are then calculated. Either linear or cubic interpolation can be used. The results then are multiplied by S_c. Since S_c is positive for n-channel transistors and negative for p-channel transistors, the sign of the current and derivatives are changed for the p-channel transistor.

Step 7

The resulting values of current and conductances are then sent to the CM from the CC. In this case the CM is acting as the interface between the MMAP and IBM PC-XT. The results are stored in the lowest 4K section of the CM at the memory locations specified in Table 6.5.

Memory	Data Stored in CM After Completion of Step						
Location	1	2	3	4	5	6	7
0	Model #	-	-	-	-	-	-
1	-	-	-	-	-	-	-
2	V_{ds}	-	δV_{ds}	-	-	-	-
3	V_{ds}	-	δV_{ds}	-	-	-	-
4	V_{gs}	-	V_{gse}	-	-	δV_{gs}	-
5	V_{gs}	-	V_{gse}	-	-	δV_{gs}	-
6	V_{sb}	-	-	-	-	-	-
7	V_{sb}	-	-	-	-	-	-
8	S_c	-	-	-	-	-	-
9	S_c	-	-	-	-	-	-
A	-	j	-	-	-	-	-
B	-	-	-	k	-	-	-
C	$(\Delta V_{ds\,j})^{-1}$	-	-	-	-	-	-
D	$(\Delta V_{ds\,j})^{-1}$	-	-	-	-	-	-
E	-	-	-	$(\Delta V_{gs\,k})^{-1}$	-	-	-
F	-	-	-	$(\Delta V_{gs\,k})^{-1}$	-	-	-
10	-	A_j	-	-	$I_{ds\,j,k}$	-	-
11	-	A_j	-	-	$I_{ds\,j,k}$	-	-
12	-	B_j	-	-	$I_{ds\,j+1,k}$	-	-
13	-	B_j	-	-	$I_{ds\,j+1,k}$	-	-
14	-	C_j	-	-	$I_{ds\,j,k+1}$	-	-
15	-	C_j	-	-	$I_{ds\,j,k+1}$	-	-
16	-	D_j	-	-	$I_{ds\,j+1,k+1}$	-	-
17	-	D_j	-	-	$I_{ds\,j+1,k+1}$	-	-
18	-	A_{j+1}	-	-	$G_{ds\,j,k}$	-	-
19	-	A_{j+1}	-	-	$G_{ds\,j,k}$	-	-
1A	-	B_{j+1}	-	-	$G_{ds\,j+1,k}$	-	-
1B	-	B_{j+1}	-	-	$G_{ds\,j+1,k}$	-	-
1C	-	C_{j+1}	-	-	$G_{ds\,j,k+1}$	-	-
1D	-	C_{j+1}	-	-	$G_{ds\,j,k+1}$	-	-
1E	-	D_{j+1}	-	-	$G_{ds\,j+1,k+1}$	-	-
1F	-	D_{j+1}	-	-	$G_{ds\,j+1,k+1}$	-	-

Memory	Data Stored in CM After Completion of Step						
Location	1	2	3	4	5	6	7
20	-	-	-	-	$G_{gs\,j,k}$	-	-
21	-	-	-	-	$G_{gs\,j,k}$	-	-
22	-	-	-	-	$G_{gs\,j-1,k}$	-	-
23	-	-	-	-	$G_{gs\,j-1,k}$	-	-
24	-	$V_{ds\,j}$			$G_{gs\,j,k+1}$	-	-
25	-	$V_{ds\,j}$			$G_{gs\,j,k+1}$	-	-
26	-	-	-	$V_{gs\,k}$	$G_{gs\,j-1,k-1}$	-	-
27	-	-	-	$V_{gs\,k}$	$G_{gs\,j-1,k-1}$	-	-
28	-	-	$\frac{\partial V_{gse}}{\partial V_{sb}}$	-	-	-	-
29	-	-	$\frac{\partial V_{gse}}{\partial V_{sb}}$	-	-	-	-
2A	-	-	$\frac{\partial V_{gse}}{\partial V_{ds}}$	-	-	-	-
2B	-	-	$\frac{\partial V_{gse}}{\partial V_{ds}}$	-	-	-	-
2C	-	-	-	-	-	I_{ds}	-
2D	-	-	-	-	-	I_{ds}	-
2E	-	-	-	-	-	G_{ds}	-
2F	-	-	-	-	-	G_{ds}	-
30	2.0	-	-	-	-	G_{gs}	-
31	2.0	-	-	-	-	G_{gs}	-
32	3.0	-	-	-	-	G_{sb}	-
33	3.0	-	-	-	-	G_{sb}	-
34	± 1.0	-	-	-	-	-	-
35	± 1.0	-	-	-	-	-	-
36	Temporary Register						
37	Temporary Register						
38	Temporary Register						
39	Temporary Register						
3A	Temporary Register						
3B	Temporary Register						
3C	Temporary Register						
3D	Temporary Register						
3E	Temporary Register						
3F	Temporary Register						

Table 6.8

The microinstructions are written in a symbolic format specific to the MMAP. Each instruction defines a set of data transfer operations and/or floating-point operation. The symbolic format is translated into the binary-encoded microinstructions.

6.6. Performance of the MMAP

The performance of the MMAP is presented in this section.

The prototype MMAP is separately programmed for both linear and cubic forms of the empirical transistor models, both of which are described in Chapter 4. The prototype is connected to the IBM PC-XT as shown in Figure 6.4 and is operated at a 1.12Mhz clock frequency. For both empirical models, four transistors are evaluated simultaneously by the prototype.

6.6.1. Microprogram Execution

The performance of the prototype is derived by measuring the time required by the MMAP to perform 10,000 executions of the microprogram, where four transistors are evaluated during each microprogram execution. The resulting time is divided by 10,000 to provide the time for a single execution of the MMAP microprogram.

The time-per-execution of the microprogram is given in Table 6.9.

MMAP Microprogram Execution Time		
Model	Clock Frequency	Time (ms)
Linear	1.12Mhz	4.6
Cubic	1.12Mhz	4.5
Linear (Simulated)	10.0Mhz	0.65
Cubic (Simulated)	10.0Mhz	0.64

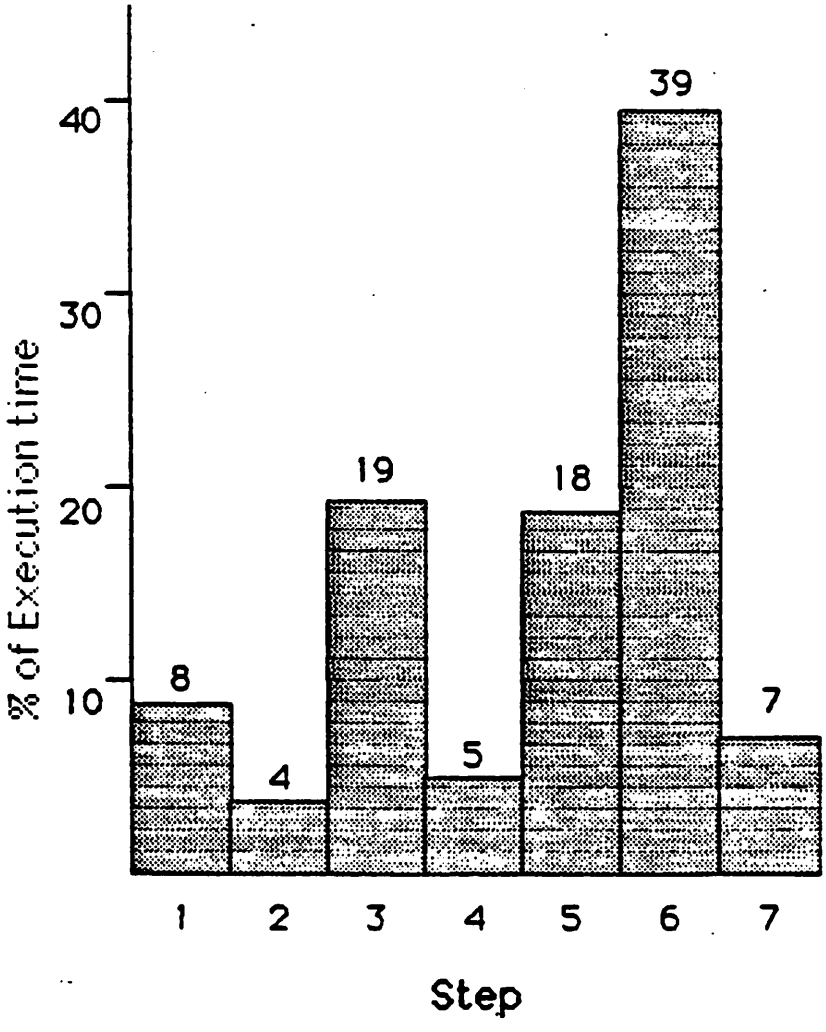
Table 6.9

For the MMAP operating at a clock speed of 1.12 Mhz, the execution of the microprogram using the linear-interpolation method requires 4.6ms, and the execution of the microprogram using the cubic-interpolation method requires 4.5ms. The microprogram using the

linear-interpolation method requires 0.1ms (2.2%) more time to execute than the microprogram using the cubic-interpolation method.

As described earlier, the MMAP can operate at a maximum clock rate of 10Mhz if the ROM memory is based on fast access (<50ns) fuse-linked PROMs. The performance of the MMAP is simulated operating at a clock frequency of 10Mhz. The simulation is performed at the register-transfer level using 1 cycle of the MMAP clock for each operation except for CM read/write operations which requires 2 cycles of the MMAP clock. (As described earlier, four IBM PC-XT clock cycles are required for a read/write operation to the CM's dynamic memory. With the IBM PC-XT operating at 4.47 Mhz and the MMAP at 10Mhz, the read/write operations require approximately two MMAP clock cycles.) The simulated microprogram-evaluation time of the MMAP operating at 10Mhz is included in Table 6.9. With the MMAP operating at a clock frequency of 10Mhz, the simulation of the MMAP gives 0.65ms for the evaluation of the microprogram using the linear-interpolation method and 0.64 for the evaluation of the microprogram using the cubic-interpolation method.

The graph in Figure 6.21 illustrates the profile of the execution of the microprogram based on linear interpolation. Each bar in the graph corresponds to one of the seven microprogram steps described in the previous section. Floating-point computations are performed in steps 3 and 6, and the execution of these two steps requires 60% of the total execution time. The remaining steps are used to access transistor and model data from the CM and return results to the CM, and they comprise the remaining 40% of the total execution time.



Profile of the MMAP's Microprogram
Figure 6.21

6.6.2. Transistor-Evaluation Time

The time-per-transistor evaluation is given in Table 6.10.

Time-per-transistor evaluation		
Model	Clock Frequency	Time (ms)
Linear	1.12Mhz	1.15
Cubic	1.12Mhz	1.13
Linear (Simulated)	10.0Mhz	0.162
Cubic (Simulated)	10.0Mhz	0.160

Table 6.10

The evaluation time is based on the MMAP evaluating four transistors simultaneously. From the measured performance of the MMAP given in Table 6.9, the time per transistor evaluation by the MMAP is 1.15ms using linear interpolation and 1.13ms using cubic interpolation. The time-per-transistor-model evaluation for the MMAP simulated operating at 10Mhz is 0.162ms using linear interpolation and 0.160ms using cubic interpolation.

A comparison between the transistor-evaluation time of the MMAP and the transistor-evaluation time of the IBM PC-XT is given in Table 6.11.

Time per Transistor Model Evaluation		
Model	Computer	Time(ms)
Linear	MMAP@1.12MHz	1.15
Cubic	MMAP@1.12MHz	1.13
Linear(Simulated)	MMAP@10.0MHz	0.162
Cubic(Simulated)	MMAP@10.0MHz	0.160
Shichman-Hodges	IBM PC-XT@4.47MHz	12
Level-2	IBM PC-XT@4.47MHz	24

Table 6.11

The models evaluated by the IBM PC_XT are the Shichman-Hodges(SH) and SPICE Level-2 MOS models[4]. The IBM PC-XT operates at a clock speed of 4.47MHz and uses the In8087 Numeric Data Processor. The model routines are written in the C programming language and compiled using the large program/data version of the Lattice C compiler[Lat84]. The data given for the analytic models evaluated on the IBM PC-XT are the average of the times required for the evaluation of the transistor in the

[4]The equations for both analytic models are given in Appendix B.

saturation. ohmic and cutoff regions. The evaluation includes the calculation of both current and derivatives. The evaluation of the SH MOS-transistor model by the IBM PC-XT requires 12ms. and the evaluation of the SPICE Level-2 MOS-transistor model by the IBM PC-XT requires 24ms. The measured transistor-evaluation time from the MMAP operating at 1.12 Mhz is 1.15ms and 1.13ms respectively for linear and cubic interpolation. For both interpolation methods. the MMAP. operating at a clock frequency of 1.12Mhz. performs a transistor evaluation a factor of 10 times faster than the IBM PC-XT performs an evaluation of the SH model and a factor of 20 times faster than the IBM PC-XT performs an evaluation of the SPICE Level-2 model. In comparison to the simulated performance of the MMAP operating at 10Mhz. the MMAP performs a transistor evaluation using either interpolation method 75 times faster than the IBM PC-XT performs an evaluation of the SH model and a factor of 150 times faster than the IBM PC-XT performs an evaluation of the SPICE Level-2 model.

6.6.3. Efficiency of the MMAP Architecture

The time required by the IBM PC-XT to evaluate the empirical model used with the MMAP is given in Table 6.12 The model routines are written in the C programming language and compiled using the large program/data version of the Lattice C compiler[Lat84].

Time-per-Transistor-Model Evaluation		
Model	Computer	Time(ms)
Linear	MMAP@1.12MHz	1.15
Cubic	MMAP@1.12MHz	1.13
Linear(Simulated)	MMAP@10.0MHz	0.162
Cubic(Simulated)	MMAP@10.0MHz	0.160
Linear	IBM PC-XT@4.47MHz	39
Cubic	IBM PC-XT@4.47MHz	38

Table 6.12

The IBM PC-XT require 39ms to evaluate the model using linear interpolation and 38ms to evaluate the model using cubic interpolation. The MMAP operating at a clock

frequency of 1.12 Mhz evaluates the two empirical models a factor of 30 times faster than the IBM PC-XT performing the evaluation.

As presented in Chapter 2, the MMAP must efficiently evaluate the MOS transistor model. To verify this, the evaluation times of the linear empirical model in ms are normalized with respect to the floating-point operation time in μs.

$$\bar{T} = \frac{\text{Model Evaluation Time (ms)}}{\text{Floating-Point Operation Time } (\mu s)} \tag{6.11}$$

$\bar{T}$ is a gauge of the performance of the transistor-model evaluation independent of the floating-point operation time. The smaller $\bar{T}$ is, the more efficient the processor is in performing the model evaluation.

The IBM PC-XT's 8087 NDP performs floating-point addition and subtraction on operands stored in its stack in 18μs, and it performs floating-point multiplication on operands stored in its stack in 28μs[Sta83]. The prototype MMAP operating at 1.12MHz performs floating-point addition, subtraction and multiplication in 8.9μs. From Table 6.12, the time required by the IBM PC-XT to evaluate the empirical model bassed on linear interpolation is 39ms. This time is normalized with respect to the floating-point multiplication time of the In8087, yielding

$$\bar{T}_{IBM} = \frac{39}{28} = 1.4. \tag{6.12}$$

The normalized time for the MMAP operating at 1.12MHz is

$$\bar{T}_{MMAP} = \frac{1.15}{8.9} = 0.13. \tag{6.13}$$

The value of $\bar{T}_{MMAP}$ is 10-times smaller than $\bar{T}_{IBM}$. Since $\bar{T}_{MMAP}$ and $\bar{T}_{IBM}$ are normalized with respect to the floating-point operation time, the order-of-magnitude difference is due to the specifically designed architecture of the MMAP and thus demonstrates that the MMAP architecture efficiently represents the transistor model.

6.7. BIASC Circuit-Simulation Program W/MMAP

The operation and performance of the BIASC circuit-simulation program running on the IBM PC-XT using the prototype MMAP is presented in this section.

6.7.1. Model-Evaluation Routine

The prototype MMAP is used in conjunction with the BIASC circuit-simulation program running on the IBM PC-XT. BIASC's MOS-transistor evaluation routine is modified for use with the MMAP. As described in Chapter 2, the model routine is altered such that the IBM PC-XT does not remain idle while the MMAP is evaluating transistors, but the IBM PC-XT works in parallel with the MMAP. The MMAP evaluates the transistor equations; but the the IBM PC-XT limits the terminal voltages, transfers data to and from the MMAP, checks convergence, and loads the results from the MMAP into the circuit matrix. Since the MMAP evaluates four transistors at the same time, the IBM PC-XT performs its operations on four transistors at a time.

The organization of the transistor-model routine is described in Chapter 2. While the MMAP is evaluating the current set of four transistors, the IBM PC-XT loads the results from the evaluation of the previous four transistors and limits the terminal voltages of the next four transistors for the following evaluation. The model-evaluation routine is listed in Appendix H.

The time for the evaluation of the different parts of the model evaluation routine performed by the IBM PC-XT are listed in Table 6.13. The data is for an IBM PC-XT operating at a clock frequency of 4.47Mhz.

BIASC Model Evaluation - 4 Transistors	
IBM PC-XT Operations	Time(ms)
Access and Limit	2.5
Check Convergence and Load Matrix	11.1
Load MMAP	0.65
Unload MMAP	0.50
Total	14.75

Table 6.13

The sum of the times required to access, limit, check convergence and load matrix is 13.6ms for 4 transistors, and the total time for the IBM PC-XT operations is 14.75ms for 4 transistors. The sum of the times required to access, limit, check convergence and load matrix for the 4 transistors, 13.6ms, is greater than the time required by the MMAP to evaluate the 4 transistors, 4.6ms for linear interpolation and 4.5ms for cubic interpolation. Therefore, the MMAP remains idle after completing an evaluation while waiting for the IBM PC_XT to complete its operations. For both linear and cubic interpolation the MMAP remains idle for approximately 9ms, greater than 60% of the time. Once the model-evaluation routine is in the evaluation loop the total time to evaluate four transistors is equal to the total time used by the IBM PC-XT to perform its operations, 14.75ms, and the resulting time per transistor is 3.7ms. The speed of the model evaluation is limited by the IBM PC-XT.

6.7.2. Attached-Processor Efficiency of the Prototype MMAP

The Attached-Processor Efficiency of the MMAP, as defined in Chapter 2, is the ratio of the MMAP evaluation time to the sum of the MMAP evaluation time plus the communication time. From the data given in Table 6.13, 0.65ms are required to input data for four transistors to the MMAP from the IBM PC-XT, and 0.5ms are required to unload the results of the four transistor evaluations from the MMAP by the IBM PC-XT. The prototype MMAP operating at a clock frequency of 1.12Mhz and used with the IBM PC-XT has an attached-processor efficiency of

$$\text{AP Efficiency} = \frac{4.6}{4.6 + 0.5 + 0.65} \times 100 = 80\% . \qquad (6.14)$$

and thus only a 20% communication overhead. However, for the MMAP operating at 10Mhz, the attached-processor efficiency is

$$\text{AP Efficiency} = \frac{0.65}{0.65 + 0.5 + 0.65} \times 100 = 36\% . \qquad (6.15)$$

resulting in a 64% communication overhead. As described in Chapter 2, the amount of data transferred between the MMAP and the IBM PC-XT (host) is small relative to the amount of floating-point computation performed by the MMAP. However, the transfer of data by the IBM PC-XT is limited by the PC-XT's 8-bit data bus, requiring four memory operations to transfer a single-precision floating-point number.

6.7.3. Circuit-Simulation Examples

The performance of two example circuits running on the IBM PC-XT using the MMAP are presented in this section. The two example circuits are first presented in Chapter 2. The first circuit is a cascade of 25 NMOS depletion-load inverters, and the second circuit is a low-power CMOS operational amplifier. The third example circuit presented in Chapter 2 can not be evaluated by the IBM PC-XT using the MMAP due to the reduction in maximum memory available for the IBM PC-XT from 640K-bytes to 512K-bytes as a result of the MMAP.

Example 1: 25 Inverter Circuit

A circuit composed of a cascade of 25 depletion-load inverters is simulated on the IBM PC-XT using the BIASC circuit-simulation program. The circuit schematic is given in Figure 6.22, and the BIASC input listing is given in Appendix B. The circuit contained a total of 50 MOS transistors, 25 n-channel enhancement transistors and 25 n-channel depletion transistors. The enhancement and depletion transistors are represented by separate models.

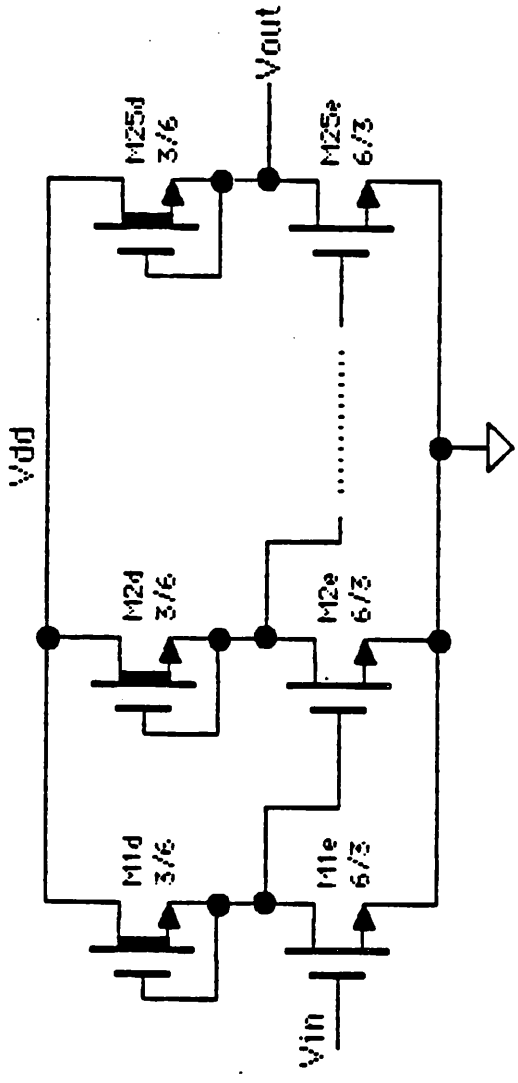
The chain of inverters is simulated for a DC sweep of the input voltage from 0 to 5 volts in 0.1 volt increments. The simulation is performed using the BIASC program with and without the MMAP. Without the MMAP, the evaluation is performed using both the SH and SPICE Level-2 models. With the MMAP, the evaluation is performed using both linear and cubic interpolation. The time, number of iterations, and time per iteration for the simulation are listed in Table 6.14.

50 Point DC Transfer of the 25 Cascaded NMOS Depletion-Load Inverter Circuit			
Configuration	Time[5] (sec)	# Iterations	Time (sec)/Iteration
BIASC Shichman-Hodges	390	197	1.95
BIASC SPICE Level-2	458	198	2.2
BIASC w/MMAP Linear	335	202	1.65
BIASC w/MMAP Cubic	330	199	1.65

Table 6.14

The simulation of the circuit without the MMAP and using the SH model is performed in 390 seconds requiring 197 iterations, and the simulation of the circuit without the MMAP and using the Level-2 model is performed in 458 seconds requiring 198 iterations. The simulation of the circuit with the MMAP and using the linear-interpolation model is performed in 335 seconds requiring 202 iterations, and the simulation of the circuit with the MMAP and using the cubic-interpolation model is performed in 330 seconds requiring 199 iterations. The time per iteration is 1.65 seconds for both linear and cubic interpolation, and the time per iteration for the SH model is 1.95 seconds, and for the Level-2 model is 2.2 seconds. The time per iteration is reduced by 16% with the use of the MMAP in comparison to the circuit simulated without the MMAP using the SH model, and the time per iteration is reduced by 25% with the use of the MMAP in comparison to the circuit simulated without the MMAP using the Level-2 model.

[5]Contains only the analysis time for the DC-transfer operation and is rounded to the nearest second.



25 Cascaded NMOS Depletion-Load Inverters
Figure 6.22

Example 2: Low-Power CMOS Operational Amplifier

A low-power CMOS operational amplifier is simulated on the IBM PC-XT using the BIASC circuit-simulation program. The circuit schematic is given in Figure 6.23, and the BIASC input listing is given in Appendix B. The circuit contained a total of 30 MOS transistors, 15 n-channel and 15 p-channel enhancement transistors. The n-channel and p-channel transistors are represented by separate models.

The DC operating-point of the CMOS amplifier is simulated. The simulation is performed using the BIASC program with and without the MMAP. Without the MMAP, the evaluation is performed using both the SH and SPICE Level-2 models. With the MMAP, the evaluation is performed using both linear and cubic interpolation. The time, number of iterations, and time per iteration for the simulation are listed in Table 6.15.

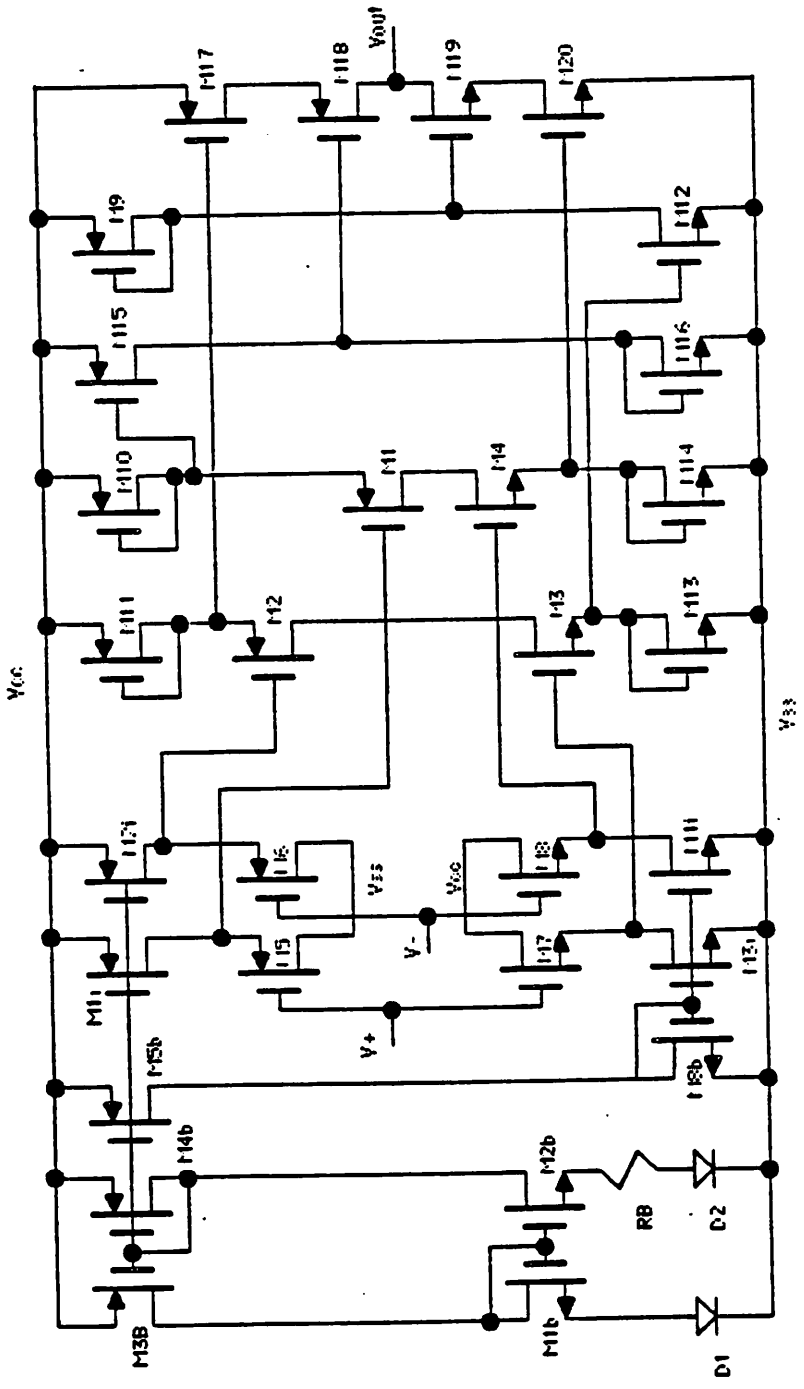
Low-Power CMOS Operational Amplifier			
Configuration	Time[6] (sec)	# Iterations	Time (sec)/Iteration
BIASC Shichman-Hodges	19.5	12	1.63
BIASC SPICE Level-2	23.0	11	2.1
BIASC w/MMAP Linear	17.5	12	1.46
BIASC w/MMAP Cubic	19.0	13	1.46

Table 6.15

The simulation of the circuit without the MMAP and using the SH model is performed in 19.5 seconds requiring 12 iterations, and the simulation of the circuit without the MMAP and using the Level-2 model is performed in 23.0 seconds requiring 11 iterations. The simulation of the circuit with the MMAP and using the linear-interpolation model is performed in 17.5 seconds requiring 12 iterations, and the simulation of the circuit with the MMAP and using the cubic-interpolation model is performed in 19.0 seconds requiring 13 iterations. The time per iteration is 1.46 seconds for both linear and cubic interpolation, and the time per iteration for the SH model is 1.63 seconds, and for

[6]Contains only the analysis time for the DC operating-point operation. The timing data for this example are generated by repeating the simulation of the circuit 10 times in a program loop, and dividing the total time by 10 and rounding to the nearest 0.5 second.

the Level-2 model is 2.1 seconds. The time per iteration is reduced by 10% with the use of the MMAP in comparison to the circuit simulated without the MMAP using the SH model. and the time per iteration is reduced by 30% with the use of the MMAP in comparison to the circuit simulated without the MMAP using the Level-2 model.



Low-Power CMOS Operational Amplifier
Figure 6.23

6.8. Chapter Summary

This chapter described a board-level prototype of the MOS-Model Attached Processor(MMAP) which is developed for use with the IBM PC-XT. The prototype is composed of 101 "off-the-shelf" SSI, MSI and LSI components, and implements the MMAP architecture described in Chapter 5. The prototype is wire-wrapped and built on two circuit boards.

The Coefficient Memory is dual-ported, accessed by both the MMAP and the IBM PC-XT, and also acts as the interface between the MMAP and the IBM PC-XT. The Coefficient Memory is 128K-bytes in size, and the 128K-bytes are divided into 32 4K-byte sections. Data for a single MOS transistor model is stored in a 4K-byte section. The prototype transfers data to and from the IBM PC-XT through the lowest 4K-byte section of Coefficient Memory. The remaining 31 4K-byte sections store data for 31 MOS transistor models. The design supports the concurrent evaluation of four MOS transistors.

The prototype MMAP operates at a clock frequency of 1.12Mhz, limited by the access time of the EPROMs comprising the Controller's control memory. The use of high-speed fuse-linked PROMs would allow a maximum clock frequency of 10Mhz, however, fuse-linked PROMs are not used because their cost is prohibitive and they cannot be reprogrammed.

Operating at a clock frequency of 1.12MHz, the MMAP performs 4 transistor evaluations in 4.6ms using the model based on linear interpolation and 4.5ms using the model based on cubic interpolation. The time per transistor evaluation performed by the MMAP operating at 1.12Mhz is a factor of 10 times faster than the evaluation of the Shichman-Hodges transistor equations in software by the IBM PC-XT and a factor of 30 times faster than the evaluation of the Level-2 equations. A profile of the execution of the MMAP's microprogram shows that the floating-point computations comprise approximately 60% of the total execution time, and the communication of data between the

Coefficient Memory and Coefficient Cache comprises the remaining 40%. A comparison of the model-evaluation times normalized to the the floating-point multiplication time demonstrates that the prototype MMAP operating at 1.12Mhz is 10 times more efficient than the IBM PC-XT w/In8087 operating at 4.47Mhz performing the evaluation of the empirical model equations.

The prototype MMAP is operated with the BIASC circuit-simulation program running on the IBM PC-XT. BIASC's model-evaluation routine is altered to use the MMAP for transistor model-evaluations. The model routine is similar to that described in Chapter 2, except that four transistors are evaluated at the same time. Operating at a clock frequency of 1.12 Mhz, the MMAP is shown to have an Attached-Processor Efficiency of 80%, however, simulation of the MMAP operating at 10Mhz shows that the Attached-Processor Efficiency would decrease to less than 40%. The low Attached-Processor Efficiency is a result of the IBM PC-XT's slow clock speed and 8-bit data bus.

The time per iteration of the DC-transfer simulation of a cascade of 25 NMOS inverters is reduced by 16% when the MMAP is used in comparison to the simulation without the MMAP using the Shichman-Hodges transistor model and is reduced by 25% when the MMAP is used in comparison to the simulation without the MMAP using the Level-2 model. The time per iteration of the DC operating-point simulation of a low-power CMOS amplifier is reduced by 10% when the MMAP is used in comparison to the simulation time without the MMAP using the Shichman-Hodges transistor model and is reduced by 30% when the MMAP is used in comparison to the simulation time without the MMAP using the Level-2 model.

CHAPTER 7

Conclusions and Further Work

The electrical simulation of MOS integrated circuits is computationally expensive, and a significant percentage of the computational time is comprised of MOS-transistor model evaluation. The purpose of the research presented in this dissertation is the development of a special-purpose processor, referred to as a MOS-Model Attached Processor (MMAP), to accelerate the evaluation of the MOS-transistor equations. The MMAP is designed specifically to evaluate the DC MOS-transistor equations and is used in conjunction with a host computer running a circuit-simulation program. The MMAP evaluates the transistor equations which would otherwise have been evaluated by the host computer.

The transistor model information is stored in the MMAP. Transistor input data, consisting of a model reference, terminal voltages and channel scale factor, are sent to the MMAP. The model pointer is used by the MMAP to access the correct model data. The MMAP evaluates the operating point of the device and returns the result to the host processor.

The MMAP, used in conjunction with a circuit-simulation program, has been shown to be a logical partition from both the circuit-simulation and the system-architecture perspective. From the circuit-simulation perspective, the MOS-transistor evaluation comprises a significant percentage of the total circuit-simulation time. From the system-architecture perspective, the MMAP performs a large amount of work while requiring only a minimal transfer of data. In addition, the host can perform operations in parallel to the MMAP.

A novel empirical model has been developed for use with the MMAP and is based on one-dimensional piecewise-cubic equations. The interpolation method used is based on the first-order behavior of the MOS transistors drain-to-source current and partial derivatives of current with respect to drain-to-source voltage and gate-to-source voltage. The empirical model reproduced the behavior of the Shichman-Hodges and SPICE Level-2 models, in addition to modeling the behavior of a device from measured data. The model is suited for use with the MMAP. The model can be evaluated without conditional branching and uses only floating-point addition, subtraction and multiplication. In addition, only single-precision floating-point computation is required.

The architecture of the MMAP exploits the characteristics of the empirical model. The floating-point hardware that is necessary for the model evaluation is readily segmented into a multiple stage pipeline, where each stage operates independent of all the others. Several transistors can be evaluated simultaneously, fully utilizing the pipelined floating-point hardware. In addition, since the transistor evaluation is performed without conditional branching the pipelined floating-point can be operated at maximum throughput.

The MMAP utilizes an internal memory hierarchy. The data for all of the transistor models are stored in the MMAP, however, only a subset of the models are used during one evaluation. A single-cycle memory is provided for the storage of model and device data required for the current transistor evaluation. This memory can be read from and written to at the same clock speed as the floating-point hardware, supplying the input operands and reading the results at the maximum possible rate.

A board-level prototype of the MMAP has been developed for use with the IBM PC-XT. The prototype is designed to evaluate 4 transistors concurrently. The MMAP has been shown to efficiently perform the transistor evaluation. The prototype MMAP is used in conjunction with a circuit-simulation program on the IBM PC-XT. A reduction in overall simulation times of up to 30% have been obtained. With the MMAP working in parallel to the IBM PC-XT, the MMAP remained idle half of the time waiting for the

IBM PC-XT to complete its operations. From the performance measurements of the MMAP the prototype is shown to have a communication overhead of 20%, however, simulation of the prototype MMAP operating at its maximum clock frequency is shown to have a communication overhead in excess of 60%. The high overhead is a result of the relatively slow transfer of floating-point data over the IBM PC-XT's 8-bit data bus. Future work should include the testing of the prototype MMAP with other computers that support both a faster processor and a faster memory-cycle time.

The work presented in this dissertation addressed the DC-MOS representation. Future work in the area of extending the capabilities of the MMAP to include the modeling of the MOS transistor capacitors is necessary. The most promising approach is to use charge as the state variable. The use of charge as the state variable is necessary to guarantee the conservation of charge in the MOS transistor [YEC83]. In addition to calculating the current and derivatives of current with respect to voltage, the MMAP would also calculate the charge associated with each of the four terminals and the partial derivatives of the charge with respect to voltage. The MMAP would store the charge representations of three of the four terminals (the fourth is dependent on the other 3 due to charge conservation), and perform the evaluation of the charge and the derivatives of charge using the same method that is used to represent the DC current.

The prototype MMAP described in this dissertation was used in conjunction with a direct-method circuit-simulation program. Future work should include the investigation of the use of the MMAP with relaxation-based simulation programs and design-optimization programs. For use with a design-optimization program, the sensitivity of a transistor's current with respect to its geometry (eg. channel width and length) would be calculated by the MMAP. In addition, the use of the MMAP with both direct and relaxation-based circuit-simulation programs on multiprocessor architectures should be investigated.

The implementation of the Model-Processing Unit (MPU) of the MMAP in a single VLSI component should be further considered. The MPU of the MMAP contains the

same functional components, although organized differently, as a special-purpose digital signal processor. A digital signal processor with 32b floating-point support has been realized in a single, 155,000 transistor, integrated circuit [KBF85]. With a single-chip version of the MPU, several MPU could be connected in parallel, as described in Chapter 5. The single-chip realization should emphasize the access of model data as well as the processing speed of the floating-point computations. The profile of the prototype MMAP's microprogram illustrated that 40% of the total time is used in the reading/writing of data between the MPU and the Coefficient Memory. To achieve the maximum increase in performance with multiple MPUs connected in parallel, the percentage of time required in reading/writing of data between the MPU and the MMAP's Coefficient Memory must be reduced.

APPENDIX A

The User's Manual and the Source Listing of the Program BIASC: a Circuit-Simulation Program for the IBM PC is available from the Software Distribution Office, Industrial Liaison Program, Electrical Engineering and Computer Science Department, University of California, Berkeley, CA 94720.

APPENDIX B

Example Circuit Listings

25 Cascaded NMOS Inverters

```
vdd vdd 0 5
vin 1 0 0 pulse 0 5 1n 1n 1n 40n 100n
m1e 2 1 0 0 mmode l 3u w 6u
m2e 3 2 0 0 mmode l 3u w 6u
m3e 4 3 0 0 mmode l 3u w 6u
m4e 5 4 0 0 mmode l 3u w 6u
m5e 6 5 0 0 mmode l 3u w 6u
m6e 7 6 0 0 mmode l 3u w 6u
m7e 8 7 0 0 mmode l 3u w 6u
m8e 9 8 0 0 mmode l 3u w 6u
m9e 10 9 0 0 mmode l 3u w 6u
m10e 11 10 0 0 mmode l 3u w 6u
m11e 12 11 0 0 mmode l 3u w 6u
m12e 13 12 0 0 mmode l 3u w 6u
m13e 14 13 0 0 mmode l 3u w 6u
m14e 15 14 0 0 mmode l 3u w 6u
m15e 16 15 0 0 mmode l 3u w 6u
m16e 17 16 0 0 mmode l 3u w 6u
m17e 18 17 0 0 mmode l 3u w 6u
m18e 19 18 0 0 mmode l 3u w 6u
m19e 20 19 0 0 mmode l 3u w 6u
m20e 21 20 0 0 mmode l 3u w 6u
m21e 22 21 0 0 mmode l 3u w 6u
m22e 23 22 0 0 mmode l 3u w 6u
m23e 24 23 0 0 mmode l 3u w 6u
m24e 25 24 0 0 mmode l 3u w 6u
m25e 26 25 0 0 mmode l 3u w 6u
m1d vdd 2 2 2 mmodd l 6u w 3u
m2d vdd 3 3 3 mmodd l 6u w 3u
m3d vdd 4 4 4 mmodd l 6u w 3u
m4d vdd 5 5 5 mmodd l 6u w 3u
m5d vdd 6 6 6 mmodd l 6u w 3u
m6d vdd 7 7 7 mmodd l 6u w 3u
m7d vdd 8 8 8 mmodd l 6u w 3u
m8d vdd 9 9 9 mmodd l 6u w 3u
m9d vdd 10 10 10 mmodd l 6u w 3u
m10d vdd 11 11 11 mmodd l 6u w 3u
m11d vdd 12 12 12 mmodd l 6u w 3u
m12d vdd 13 13 13 mmodd l 6u w 3u
m13d vdd 14 14 14 mmodd l 6u w 3u
m14d vdd 15 15 15 mmodd l 6u w 3u
m15d vdd 16 16 16 mmodd l 6u w 3u
m16d vdd 17 17 17 mmodd l 6u w 3u
m17d vdd 18 18 18 mmodd l 6u w 3u
m18d vdd 19 19 19 mmodd l 6u w 3u
m19d vdd 20 20 20 mmodd l 6u w 3u
m20d vdd 21 21 21 mmodd l 6u w 3u
m21d vdd 22 22 22 mmodd l 6u w 3u
m22d vdd 23 23 23 mmodd l 6u w 3u
m23d vdd 24 24 24 mmodd l 6u w 3u
m24d vdd 25 25 25 mmodd l 6u w 3u
```

```
m25d vdd 26 26 mmodd l 6u w 3u
model mmode level=2 vto=1 kp=20u phi=.8 lambda=.015 cox=0
model mmodd level=2 vto=-2 kp=20u phi=.8 lambda=.02 cox=0
print dc v 2 3 4 5
set maxit=100
dctr vin 0 5 0.02
quit
```

Low-Power CMOS Amplifier

```
vcc vcc 0 5
vss vss 0 -5
vp vp 0 0
d1 1 vss dmod1
d2 2 vss dmod1 2
rb 3 2 1350
m1b 4 4 1 1 mmodn w 128u l 12u
m2b 5 4 3 3 mmodn w 128u l 12u
m3b 4 5 vcc vcc mmodp w 128u l 6u
m4b 5 5 vcc vcc mmodp w 128u l 6u
m5b 6 5 vcc vcc mmodp w 128u l 6u
m8b 6 6 vss vss mmodn w 128u l 12u
mi1 8 5 vcc vcc mmodp w 128u l 6u
mi2 12 5 vcc vcc mmodp w 128u l 6u
mi3 7 6 vss vss mmodn w 128u l 12u
mi4 15 6 vss vss mmodn w 128u l 12u
m1 10 8 14 14 mmodn w 200u l 3u
m2 11 12 13 13 mmodn w 200u l 3u
m3 22 7 13 vcc mmodp w 200u l 3u
m4 16 15 14 vcc mmodp w 200u l 3u
m5 vss vp 8 vcc mmodp w 200u l 3u
m6 vss vn 12 vcc mmodp w 200u l 3u
m7 vcc vp 7 7 mmodn w 200u l 3u
m8 vcc vn 15 15 mmodn w 200u l 3u
m9 9 9 vcc vcc mmodp w 18u l 3u
m10 10 10 vcc vcc mmodp w 180u l 7.5u
m11 11 11 vcc vcc mmodp w 180u l 7.5u
m12 9 22 vss vss mmodn w 90u l 7.5u
m13 22 22 vss vss mmodn w 90u l 7.5u
m14 16 16 vss vss mmodn w 90u l 7.5u
m15 17 10 vcc vcc mmodp w 180u l 7.5u
m16 17 17 vss vss mmodn w 9u l 3u
m17 19 11 vcc vcc mmodp w 180u l 7.5u
m18 vn 9 19 vcc mmodp w 72u l 3u
m19 vn 17 18 18 mmodn w 36u l 3u
m20 18 16 vss vss mmodn w 90u l 7.5u
model dmod1 is=1e-16
model mmodn kp=70u level=2 vto=.7 lambda=.01 gamma=.3 phi=.56
model mmodp kp=35u level=2 type=pmos vto=-.7 lambda=.01 gamma=.3 phi=.56
set maxit=100
print dc v vn
dctr vp -0.5 0.5 0.01
bias
quit
```

Worst-Case Path Through Op-Code PLA

```
vbb 1  0 -3.59
vdd 2  0 4.50
m1 3  4  2  1 md   l 6u w 8u
m2 2  5  5  1 md   l 6u w 8u
m3 5  6  0  1 mn   l 3u w 14u
m4 7  8  0  1 mn   l 3u w 100u
m5 4  8  0  1 mn   l 3u w 10u
m6 2  8  8  1 md   l 6u w 8u
m7 8  9  0  1 mn   l 3u w 18u
m8 7  4  2  1 md   l 5u w 30u
m9 7  4  2  1 mn   l 3u w 120u
m10 3  4  3  1 md   l 20u w 20u
m11 0  10  0  1 md   l 7u w 7u
m12 2  4  3  1 mn   l 3u w 30u
m13 3  10  0  1 mn   l 3u w 20u
m14 2  2  10  1 me   l 10u w 5u
m15 10  4  0  1 mn   l 3u w 7u
m16 9  2  4  1 mn   l 3.5u w 25u
m17 11  11  2  1 md   l 6u w 9u
m18 11  12  0  1 mn   l 3u w 120u
m19 11  13  0  1 mn   l 3u w 20u
m20 2  14  0  1 mn   l 3u w 150u
m21 2  15  0  1 mn   l 3u w 150u
m22 2  16  0  1 mn   l 3u w 250u
m23 0  17  0  1 mn   l 3u w 400u
m24 17  2  12  1 mn   l 3.5u w 160u
m25 0  18  0  1 mn   l 3u w 20u
m26 18  2  13  1 mn   l 3.5u w 8u
r1 19  20 1k
m27 21  22  0  1 mn   l 3u w 10u
m28 2  22  22  1 md   l 8u w 6u
m29 22  23  0  1 mn   l 3u w 8u
m30 2  24  24  1 md   l 8u w 6u
m31 24  25  0  1 mn   l 3u w 8u
m32 26  24  0  1 mn   l 3u w 10u
m33 27  0  0  1 me   l 4u w 30u
m34 15  15  2  1 md   l 10u w 4u
m35 14  14  2  1 md   l 10u w 4u
m36 28  0  0  1 mn   l 3u w 750u
m37 29  0  0  1 mn   l 3u w 600u
m38 7  30  26  1 mn   l 3u w 60u
m39 7  31  21  1 mn   l 3u w 60u
m40 2  32  32  1 md   l 6u w 9u
m41 32  13  0  1 mn   l 3u w 15u
m42 29  33  2  1 mn   l 3u w 15u
m43 29  16  0  1 mn   l 3u w 15u
m44 34  34  2  1 md   l 8u w 5u
m45 6  35  34  1 mn   l 3u w 20u
m46 36  33  2  1 mn   l 3u w 30u
m47 37  33  2  1 me   l 4u w 8u
m48 39  33  2  1 mn   l 3u w 20u
```

```
m49 12 33 2 1 me 1 4u w 160u
m50 12 19 0 1 mn 1 3u w 180u
m51 36 40 0 1 mn 1 3u w 100u
m52 36 41 0 1 mn 1 3u w 100u
m53 34 12 0 1 mn 1 3u w 120u
m54 42 43 21 1 mn 1 3u w 30u
m55 28 33 2 1 mn 1 3u w 15u
m56 28 16 0 1 mn 1 3u w 15u
r2 42 16 .7k
m57 43 44 37 1 mn 1 3.5u w 8u
m58 31 44 23 1 mn 1 3.5u w 12u
r3 48 0 .1k
m59 48 20 47 1 mn 1 3u w 9u
m60 47 33 2 1 me 1 4u w 8u
r4 49 48 .1k
m61 49 20 46 1 mn 1 3u w 9u
m62 46 33 2 1 me 1 4u w 8u
r5 50 49 .1k
m63 50 20 45 1 mn 1 3u w 9u
m64 45 33 2 1 me 1 4u w 8u
r6 51 50 .10k
m65 30 44 25 1 mn 1 3.5u w 12u
m66 9 5 0 1 mn 1 3u w 15u
m67 9 6 2 1 md 1 6u w 9u
m68 9 6 2 1 me 1 4u w 45u
m69 34 13 0 1 mn 1 3u w 20u
m70 6 6 2 1 md 1 6u w 9u
m71 51 20 13 1 mn 1 3u w 9u
m72 13 33 2 1 me 1 4u w 8u
m73 19 52 53 1 mn 1 3u w 50u
m74 54 44 52 1 mn 1 3.5u w 10u
m75 44 44 55 1 mn 1 3.5u w 10u
m76 38 55 53 1 mn 1 3u w 50u
m77 56 57 57 1 ml 1 12u w 4u
m78 2 57 57 1 md 1 8u w 8u
m79 57 54 0 1 mn 1 3u w 8u
m80 2 54 54 1 md 1 6u w 9u
m81 54 56 0 1 mn 1 3u w 15u
m82 39 58 56 1 mn 1 4u w 8u
m83 29 2 27 1 me 1 4u w 12u
m84 2 59 59 1 md 1 6u w 9u
m85 59 60 0 1 mn 1 3u w 20u
m86 28 2 60 1 me 1 4u w 12u
m87 2 33 15 1 mn 1 3u w 10u
m88 15 61 0 1 mn 1 3u w 15u
m89 2 33 14 1 mn 1 3u w 10u
m90 14 59 62 1 mn 1 3u w 45u
m91 62 27 63 1 mn 1 3u w 45u
m92 63 64 0 1 mn 1 3u w 45u
m93 62 0 0 1 mn 1 3u w 30u
m94 2 61 61 1 md 1 6u w 9u
m95 61 60 0 1 mn. 1 3u w 15u
m96 61 27 65 1 mn 1 3u w 30u
m97 65 64 0 1 mn 1 3u w 30u
m98 61 0 0 1 mn 1 3u w 15u
m99 2 40 40 1 md 1 6u w 12u
```

```
m100 40  14  0  1 mn   l 3u w 25u
m101 40  33  0  1 mn   l 3u w 25u
m102 40  0  0  1 mn   l 3u w 25u
m103 40  0  0  1 mn   l 3u w 25u
m104 2  41  41  1 md   l 6u w 12u
m105 41  15  0  1 mn   l 3u w 25u
m106 41  33  0  1 mn   l 3u w 25u
m107 41  0  0  1 mn   l 3u w 25u
m108 41  0  0  1 mn   l 3u w 25u
m109 2  25  25  1 md   l 6u w 9u
m110 25  32  0  1 mn   l 3u w 15u
m111 25  0  0  1 mn   l 3u w 15u
m112 2  23  23  1 md   l 6u w 12u
m113 23  0  0  1 mn   l 3u w 25u
m114 23  13  0  1 mn   l 3u w 25u
m115 23  0  0  1 mn   l 3u w 25u
m116 23  0  0  1 mn   l 3u w 25u
dax1  1  65      da 330.00p
dpx1  1  65      dp 156.00u
dax2  1  61      da 456.00p
dpx2  1  61      dp 210.00u
dax3  1  63      da 450.00p
dpx3  1  63      dp 216.00u
dax4  1  62      da 615.00p
dpx4  1  62      dp 294.00u
dax5  1  60      da 93.00p
dpx5  1  60      dp 42.00u
dax6  1  59      da 206.00p
dpx6  1  59      dp 94.00u
dax7  1  57      da 243.00p
dpx7  1  57      dp 94.00u
dax8  1  56      da 162.00p
dpx8  1  56      dp 60.00u
dax9  1  38      da 245.00p
dpx9  1  38      dp 118.00u
dax10  1  55      da 85.00p
dpx10  1  55      dp 38.00u
dax11  1  44      da 85.00p
dpx11  1  44      dp 38.00u
dax12  1  52      da 85.00p
dpx12  1  52      dp 38.00u
dax13  1  54      da 271.00p
dpx13  1  54      dp 122.00u
dax14  1  53      da 490.00p
dpx14  1  53      dp 236.00u
dax15  1  19      da 245.00p
dpx15  1  19      dp 118.00u
dax16  1  51      da 81.00p
dpx16  1  51      dp 36.00u
dax17  1  20      da 1p
dpx17  1  20      dp 1u
dax18  1  40      da 414p
dpx18  1  40      dp 128u
dax19  1  25      da 384.00p
dpx19  1  25      dp 174.00u
dax20  1  30      da 93.00p
```

```
dpx20 1 30      dp 42.00u
dax21 1 45      da 645p
dpx21 1 45      dp 288u
dax22 1 50      da 81.00p
dpx22 1 50      dp 36.00u
dax23 1 49      da 81.00p
dpx23 1 49      dp 36.00u
dax24 1 48      da 81.00p
dpx24 1 48      dp 36.00u
dax25 1 46      da 645p
dpx25 1 46      dp 288u
dax26 1 47      da 645p
dpx26 1 47      dp 288u
dax27 1 23      da 766.00p
dpx27 1 23      dp 356.00u
dax28 1 31      da 93.00p
dpx28 1 31      dp 42.00u
dax29 1 43      da 81.00p
dpx29 1 43      dp 34.00u
dax30 1 42      da 165.00p
dpx30 1 42      dp 78.00u
dax31 1 39      da 206.00p
dpx31 1 39      dp 92.00u
dax32 1 37      da 162.00p
dpx32 1 37      dp 68.00u
dax33 1 36      da 900p
dpx33 1 36      dp 220u
dax34 1 6      da 206.00p
dpx34 1 6      dp 94.00u
dax35 1 34      da 735p
dpx35 1 34      dp 300u
dax36 1 41      da 414p
dpx36 1 41      dp 128u
dax37 1 32      da 186.00p
dpx37 1 32      dp 84.00u
dax38 1 29      da 2425p
dpx38 1 29      dp 1100u
dax39 1 28      da 2425p
dpx39 1 28      dp 1100u
dax40 1 14      da 1p
dpx40 1 14      dp 1u
dax41 1 15      da 1p
dpx41 1 15      dp 1u
dax42 1 27      da 258.00p
dpx42 1 27      dp 120.00u
dax43 1 26      da 1800p
dpx43 1 26      dp 800u
dax44 1 24      da 162.00p
dpx44 1 24      dp 64.00u
dax45 1 22      da 162.00p
dpx45 1 22      dp 64.00u
dax46 1 21      da 1800p
dpx46 1 21      dp 800u
dax47 1 13      da 645p
dpx47 1 13      dp 288u
dax48 1 18      da 81.00p
```

```
dpx48  1  18      dp  34.00u
dax49  1  12      da  12900p
dpx49  1  12      dp  5760u
dax50  1  17      da  685.00p
dpx50  1  17      dp  338.00u
dax51  1  11      da  735p
dpx51  1  11      dp  300u
dax52  1  9       da  735p
dpx52  1  9       dp  300u
dax53  1  10      da  162.00p
dpx53  1  10      dp  60.00u
dax54  1  8       da  198.00p
dpx54  1  8       dp  88.00u
dax55  1  4       da  230.00p
dpx55  1  4       dp  106.00u
dax56  1  7       da  3500p
dpx56  1  7       dp  1000u
dax57  1  5       da  182.00p
dpx57  1  5       dp  80.00u
dax58  1  3       da  621.00p
dpx58  1  3       dp  286.00u
model mn level=2 vto=.880 kp=5.07e-5 gamma=.1 phi=.561 lambda=.02
model me level=2 vto=.49 kp=5.33e-5 gamma=.1 phi=.561 lambda=.02
model ml level=2 vto=-1.95 kp=4.79e-5 gamma=.1 phi=.561 lambda=.02
model md level=2 vto=-2.1 kp=4.65e-5 gamma=.1 phi=.561 lambda=.02
model da is=7.242e-6
model dp is=6.234e-11
print tran v 33 58 53 13 21
print dc v 33 58 53 13 21
vprech 33 0 0.5 pulse 0.5 4.5 11n 7n 6n 35n
vpw 58 0 0.5 pulse 0.5 4.5 60n 7n 6n 35n
vqmux07 39 0 0
viso 44 0 4.5
vpd 53 0 pulse 0.0 4.5 90n 10n 6n 200n
vqf 64 0 4.5
vref 35  0  2.2
set abstol=1n maxit=20 dcit=200 dctrit=200 dctotal=10000
bias
dctr vprech 0.5 4.5 0.15
quit
```

APPENDIX C

Analytic Transistor Models

Shichman-Hodges Equations:

For $V_{gs} \geq V_t$ and $V_{ds} \leq V_{gs} - V_t$

$$I_{ds} = k_n \frac{W}{L} V_{ds} \left(V_{gs} - V_t - \frac{V_{ds}}{2} \right) \left(1 + \lambda V_{ds} \right)$$

For $V_{gs} \geq V_t$ and $V_{ds} > V_{gs} - V_t$

$$I_{ds} = \frac{k_n}{2} \frac{W}{L} \left(V_{gs} - V_t \right)^2 \left(1 + \lambda V_{ds} \right)$$

For $V_{gs} < V_t$

$$I_{ds} = 0$$

$$V_t = V_{t0} + \gamma \left(\sqrt{V_{sb} + 2|\phi_p|} - \sqrt{2|\phi_p|} \right)$$

SPICE Level-2 Equations:

For $V_{gs} \geq V_t$ and $V_{ds} \leq V_{ds\,sat}$

$$I_{ds} = \mu_n C_{ox} \frac{W}{L} \left\{ \left(V_{gs} - V_{FB} - 2|\phi_p| - \frac{V_{ds}}{2} \right) V_{ds} - \frac{2}{3} \frac{\sqrt{2 \epsilon_s q N_A}}{C_{ox}} \left[(2|\phi_p| + V_{ds} + V_{sb})^{1.5} - (2|\phi_p| + V_{sb})^{1.5} \right] \right\}$$

For $V_{gs} \geq V_t$ and $V_{ds} > V_{ds\,sat}$

$$I_{ds} = \mu_n C_{ox} \frac{W}{L} \left\{ \left(V_{gs} - V_{FB} - 2|\phi_p| - \frac{V_{dssat}}{2} \right) V_{dssat} - \frac{2}{3} \frac{\sqrt{2 \epsilon_s q N_A}}{C_{ox}} \left[(2|\phi_p| + V_{dssat} + V_{sb})^{1.5} - (2|\phi_p| + V_{sb})^{1.5} \right] \right\}$$

For $V_{gs} < V_t$

$$I_{ds} = 0$$

$$V_{dssat} = V_{gs} - V_{FB} - 2|\phi_p| - \frac{\epsilon_s q N_A}{C^2_{ox}} \left[\sqrt{1 + \frac{2 C^2_{ox}}{\epsilon_s q N_A} (V_{gs} + V_{sb} - V_{FB})} - 1 \right]$$

$$V_t = V_{FB} + 2|\phi_p| + \frac{\sqrt{2 \epsilon_s N_A (2|\phi_p| + V_{sb})}}{C_{ox}}$$

APPENDIX D

Enhanced Monotonic Piecewise Cubic Interpolation(EMPCI) of 1 Independent Variable

Enhanced Monotonic, Piecewise-Cubic Interpolation(EMPCI) is an interpolation method that fits piecewise, cubic-polynomial equations to monotone data; producing a composite function that is monotonic, whose composite derivative conforms to the shape of the data points. In addition, the difference in the second derivatives of adjacent polynomials at their boundaries is minimized. The values of derivatives at the data points are calculated to meet the specified criteria. Fritsch and Carlson's monotone piecewise cubic interpolation method[FrC80] provides the basis for the work presented in this appendix.

The EMPCI method is derived to represent one-dimensional MOS transistor I-V characteristics. This appendix provides the basis for the EMPCI method as applied to one independent variable. A description of the EMPCI method is first presented, followed by a description of the method. Then, examples of the EMPCI method are given. Finally, the problems related to function discontinuities at the boundaries of adjacent polynomials due to finite numerical precision are described.

D.1. Overview

Interpolation of monotone data with the further restrictions of the MOS transistor current-voltage characteristics is accomplished by use of the EMPCI method. EMPCI is an interpolation method that constructs a monotonic function composed of piecewise cubic equations interpolating between monotone data. The function f(x), illustrated by Figure D.1, is the composite function, composed of piecewise cubic polynomial segments.

$$f(x) = p_i(x) , \quad i = 1, N \tag{D.1}$$

There is a unique cubic polynomial segment for each value of x.

$$p_i(x) = a_i + b_i (x - x_i) + c_i (x - x_i)^2 + d_i (x - x_i)^3 \quad \text{for } x_i \leqslant x < x_{i+1} \tag{D.2}$$

The coefficients a_i, b_i, c_i and d_i are calculated from the values of $f(x_i)$, $f'(x_i)$, $f(x_{i+1})$ and $f'(x_{i+1})$.

$$a_i = f(x_i) \tag{D.3}$$

$$b_i = f'(x_i) \tag{D.4}$$

$$c_i = \frac{3F_i - 2f'(x_i) - f'(x_{i+1})}{\Delta x_i} \tag{D.5}$$

$$d_i = \frac{f'(x_i) + f'(x_{i+1}) - 2F_i}{\Delta x_i^2} \tag{D.6}$$

$$F_i = \frac{f(x_{i+1}) - f(x_i)}{\Delta x_i} \tag{D.7}$$

$$\Delta x_i = x_{i+1} - x_i \tag{D.8}$$

The composite function is continuous.

$$p_i(x_{i+1}) = p_{i+1}(x_{i+1}) , \quad i = 1, N \tag{D.9}$$

and continuous in first derivative.

$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}) , \quad i = 1, N \tag{D.10}$$

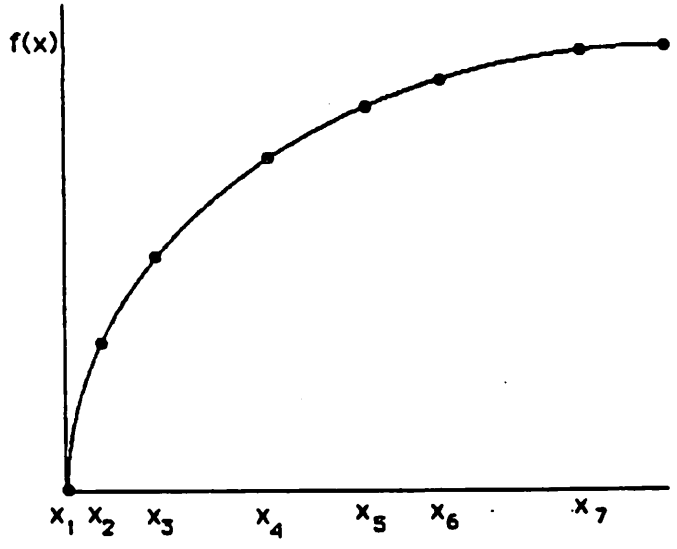
The discrete values of the function f(x) are known. The EMPCI method calculates the derivative values at the data points such that the composite curve is monotonic.

$$p_i'(x) \geqslant 0 \quad \text{for } x_i \leqslant x < x_{i+1} \tag{D.11}$$

The derivative of the polynomial has either no local minima, no local maxima, a single minima, or a single maxima over the range of a cubic polynomial. The shape of the derivative characteristic of a piecewise cubic polynomial is dependent on the measured data points at the polynomial's end points, $f(x_i)$ and $f(x_{i+1})$, and the the data points before and after the polynomial's end points, $f(x_{i-1})$ and $f(x_{i+2})$.

The piecewise-cubic equations representing the transistor's I-V characteristics are solved using iterative techniques. The Newton-Raphson iterative method is generally used because of it's quadratic convergence. The Newton-Raphson iterative method requires that the nonlinear function have continuous first and second derivatives for maximum rate of convergence. Since only the first derivative is continuous at boundaries of polynomials, the difference between second derivatives at boundaries of piecewise polynomials is minimized.

$$\text{Minimize} \left| \frac{d^2p_i}{dx^2} - \frac{d^2p_{i+1}}{dx^2} \right|_{x = x_{i+1}} \tag{D.12}$$



Piecewise Cubic Function
Figure D.1

D.2. The Set of Equations Defining EMPCI Polynomials

The values of derivatives at measured data points are derived by the requirements of EMPCI polynomials. A set of inequalities as functions of data point derivatives are defined. The inequalities are based on the monotonic behavior of the cubic polynomials, the sign of the second derivative at the boundaries of the cubic polynomials, and the

limits on the magnitude of derivative values. Also a set of equations representing the difference in second derivative at polynomial boundaries as a function of derivative is defined.

D.2.1. Inequalities Constraining Value of Derivative

The slope of line segments joining adjacent data points, F_i ($F_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$), is calculated for each interval. Since the data is monotonic, F_i is always positive. Dependent on the values of F_i, each segment is classified dependent on its relationship to neighboring F's.

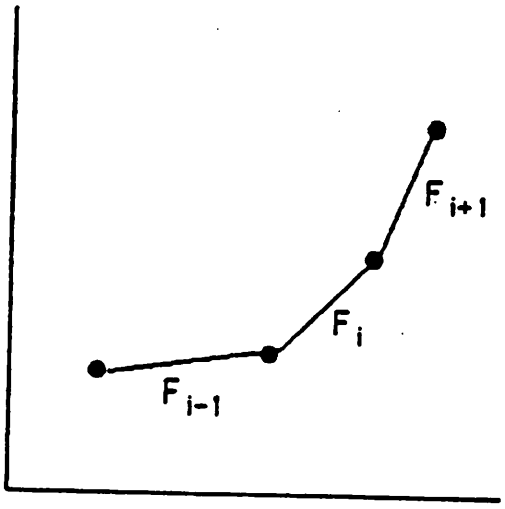
Case 1: $F_{i-1} \leqslant F_i \leqslant F_{i+1}$ (D.13)

Case 2: $F_{i-1} > F_i > F_{i+1}$ (D.14)

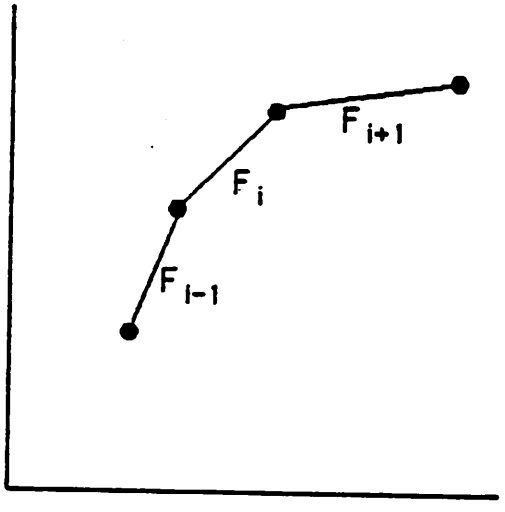
Case 3: $F_{i-1} \leqslant F_i > F_{i+1}$ (D.15)

Case 4: $F_{i-1} > F_i \leqslant F_{i+1}$ (D.16)

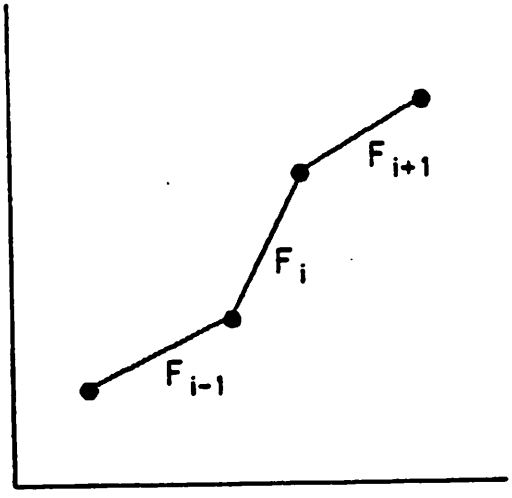
The four different cases are illustrated in Figures D.2a, D.2b, D.2c and D.2d.



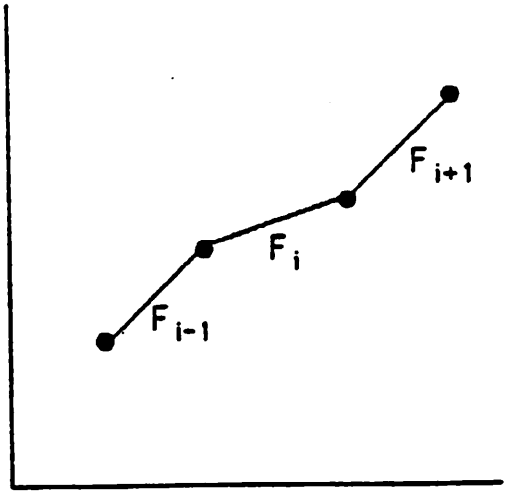
Case 1
Figure D.2a



Case 2
Figure D.2b



Case 3
Figure D.2c



Case 4
Figure D.2d

Each case specifies the sign of second derivative of the cubic polynomial over the interval with respect to x. The second derivative of the cubic polynomial is a linear function of x.

$$\frac{d^2 p_i}{d x^2} = 2 c_i + 6 d_i (x - x_i) \tag{D.17}$$

(coefficients c_i and d_i are given in Equations D.5 and D.6) Only the sign of the second derivative at the end points of the polynomials are considered since the second derivative is a linear function of x. The second derivative at the polynomial end points are linear functions of derivative.

$$\left.\frac{d^2 p_i}{d x^2}\right|_{x = x_i} = 2 \frac{3F_i - 2f'(x_i) - f'(x_{i+1})}{\Delta x_i} \tag{D.18}$$

$$\left.\frac{d^2 p_i}{d x^2}\right|_{x = x_{i+1}} = 2 \frac{-3F_i + f'(x_i) + 2f'(x_{i+1})}{\Delta x_i} \tag{D.19}$$

The four cases represent the four different forms of the first derivative. The slope of the line segments joining data points are increasing in Case 1. and therefore the polynomial's first derivative must increase over the segment(the second derivative of the function is positive). In Case 2. the slope of line segments joining data points are decreasing. and therefore the polynomial's first derivative must decrease(the second derivative of the function is negative). Cases 3 and 4 represent a transition in derivative behavior. In Case 3. the function's second derivative is initially positive. and then becomes negative over the segment; representing a maximum in the polynomial's first derivative. The function's second derivative is initially negative in Case 4. and then becomes positive over the segment. representing a minimum in the polynomial's first derivative.

The monotone behavior of a polynomial segment of type Case 4 is also limited by the following inequality.

$$f'(x_i) + f'(x_{i+1}) - 3F_i \leq 0 \tag{D.20}$$

(D.20)

The above inequality is not the exact limit on the monotonicity of the polynomial segment but is the conservative lower bound that can be represented as a linear function of derivative.

A final restriction is placed on the range of values of derivative. The derivative is constrained to be no greater than the maximum of the slopes of the two adjacent line segments and no less than the minimum of the slopes of the two adjacent line segments.

$$\text{Minimum}\,(F_{i-1}, F_i) \leqslant f'_i(x_i) \tag{D.21}$$

$$\text{Maximum}\,(F_{i-1}, F_i) \geqslant f'_i(x_i) \tag{D.22}$$

These inequalities are used to prohibit large changes in derivative over the range of a the segment.

The inequalities for the four cases are summarized in the following tables.

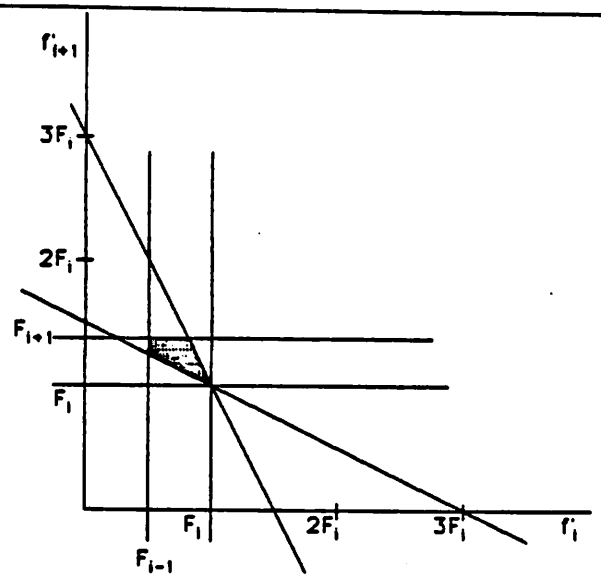
Case 1
$3F_i - 2f'(x_i) - f'(x_{i+1}) \geqslant 0$
$-3F_i + f'(x_i) + 2f'(x_{i+1}) \geqslant 0$

Case 2
$3F_i - 2f'(x_i) - f'(x_{i+1}) < 0$
$-3F_i + f'(x_i) + 2f'(x_{i+1}) < 0$

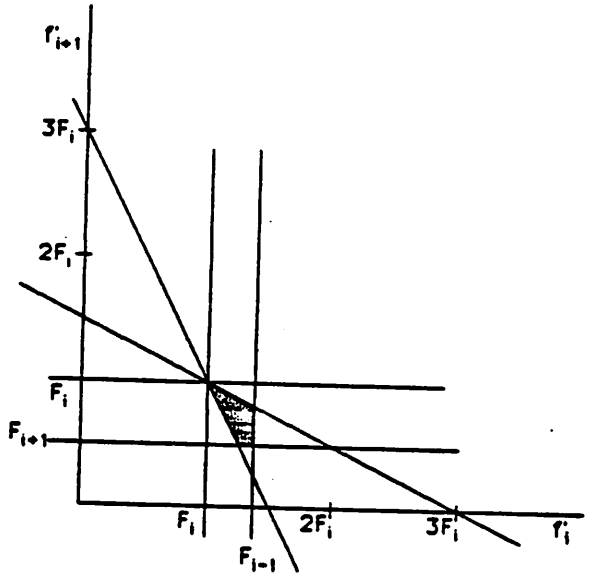
Case 3
$3F_i - 2f'(x_i) - f'(x_{i+1}) \geqslant 0$
$-3F_i + f'(x_i) + 2f'(x_{i+1}) < 0$

Case 4
$3F_i - 2f'(x_i) - f'(x_{i+1}) < 0$
$-3F_i + f'(x_i) + 2f'(x_{i+1}) \geqslant 0$
$f'(x_i) + f'(x_{i+1}) - 3F_i \leqslant 0$

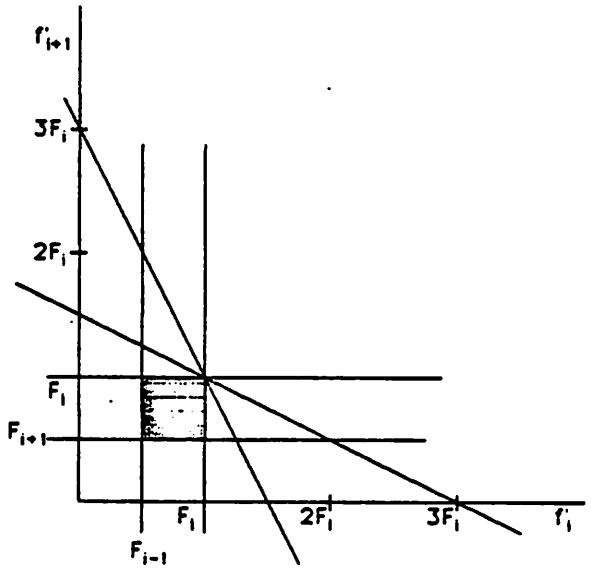
A graphic representation of these inequalities are given in Figures D.3a, D.3b, D.3c and D.3d. In each figure, the hatched area represents the valid range of derivative values.



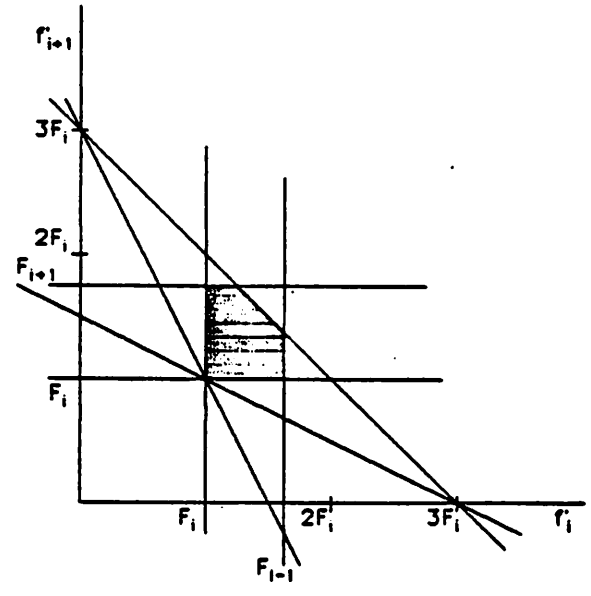
Case 1 Inequalities
Figure D.3a



Case 2 Inequalities
Figure D.3b



Case 3 Inequalities
Figure D.3c



Case 4 Inequalities
Figure D.3d

The complete set of inequalities is represented in matrix form.

$$G f' - h > 0 \tag{D.23}$$

Matrix G consists of all the linear coefficients of the derivatives, and the vector h contains the constant terms. f' is the vector of unknown derivatives.

D.2.2. Minimizing The Error in Second Derivative

The second derivative is not guaranteed to be continuous at the boundaries of adjacent polynomials. The EMPCI method imposes additional constraints such that the sum of the errors in second derivatives of adjacent polynomials is minimized. The difference in second derivative at point i is a linear function of f'.

$$\Delta \text{ Second} = 2 c_i + 6 d_i \Delta x_i - 2 c_{i+1} \tag{D.24}$$

$$= f'_i + 4f'_{i+1} + f'_{i+2} - 3(F_i + F_{i+1}) \tag{D.25}$$

The linear system of error equations are presented in matrix form as

$$S_e = E f' - d , \tag{D.26}$$

where the error in second derivative for each polynomial boundary is contained in the vector S_e. The total error due to mismatch in second derivatives is set equal to the positive square root of the sum of the squares of the errors at the individual polynomial boundaries.

$$\text{Total Error in Second Derivative} = \| S_e \| \tag{D.27}$$

The limiting of error in second derivative is represented by minimizing the total error in second derivative.

D.3. Solution of the System of Equations Defining the EMPCI Polynomials

The system of equations defining the f' s is

$$\text{Minimize } \| Ef' - d \| \text{ subject } Gf' \geq h . \tag{D.28}$$

which is a linear least squares problem with linear equality constraints [LaH74]. The problem defined is of the form that can be solved using the Modified Simplex

Method[Str76], a linear programming method. The computation time of the solution of the system of equations grows in nonpolynomial time[Adl84] and is not feasible for use with problems an arbitrary number of segments.

D.3.1. Solution for f' Using an Iterative Heuristic - Preliminaries

The form of the E and d matrices are known, and thus the error due to difference in second derivative (S_e) can be derived.

$$||S_e||^2 = (Ef' - d)^T (Ef' - d) \tag{D.29}$$

The matrices E and d are expanded in terms of F and f'.

$$\begin{aligned} ||S_e||^2 = {} & (f'_0 + 4f'_1 + f'_2 - 3F_0 - 3F_1)^2 \\ & + (f'_1 + 4f'_2 + f'_3 - 3F_1 - 3F_2)^2 \\ & \vdots \\ & + (f'_{N-3} + 4f'_{N-2} + f'_{N-1} - 3F_{N-3} - 3F_{N-2})^2 \end{aligned} \tag{D.30}$$

The change in $||S_e||^2$ due to small changes in f' can be calculated by expanding the function about f'.

$$||S_e(f' + \Delta f')||^2 = ||S_e(f')||^2 + \sum_{i=0}^{i=M} \frac{\partial ||S_e||^2}{\partial f'_i} \Delta f'_i \tag{D.31}$$

The values of the partial derivatives are easily calculated, and the effect of small changes in f'_i on the error is provided.

D.3.2. Algorithm to Calculate f'

The matrices representing inequality constraints and error in the second derivative are set up. These matrices are constant, dependent only on the values of the data points. Next, the initial values of f' are calculated. The initial values are derived from a cubic spline[DaB74] fit of the data points. The cubic spline guarantees the continuity of the second derivative, and therefore the initial error in second derivative is zero. The following is then repeated until the error in second derivative has reached a suitable minimum

and the inequality constraint is satisfied: First, the set of inequalities is checked. Then, if an inequality is not satisfied, the f's which effect the inequality are noted. Those f's are then changed in the direction which allows the inequality to be satisfied. The change in f's are small. As a result, the inequality may remain unsatisfied, but with the difference reduced. Finally, the remaining f's that were not changed to meet the inequality constraints are changed to reduce the error in second derivative. The values of f' calculated meet the inequality requirements and limit the total error in second derivative. (The total error in second derivative is not guaranteed to be the minimal solution.)

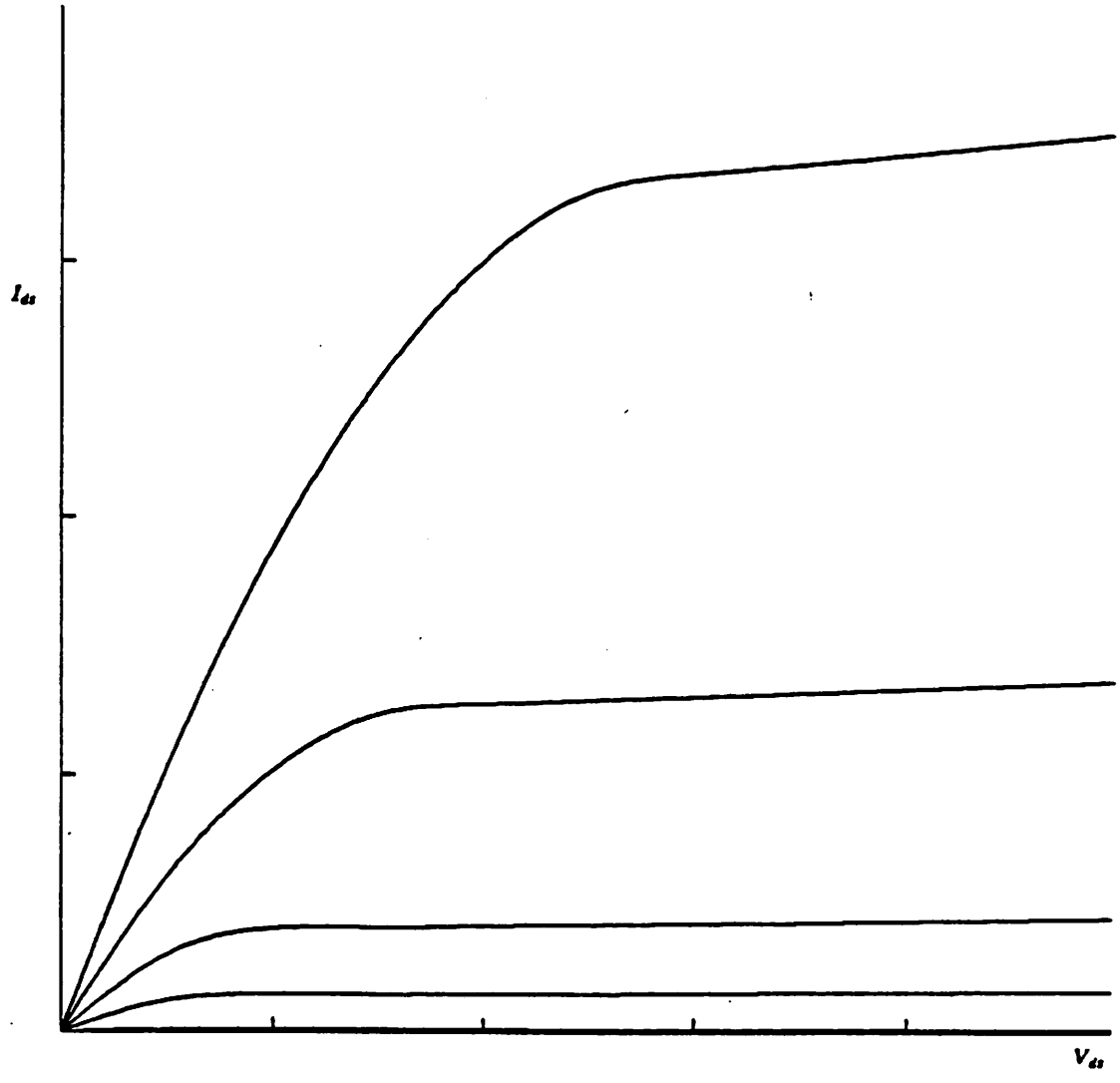
D.4. Example

Data for a MOS transistor's drain-to-source current as a function of drain-to-source voltage are given in the following table.

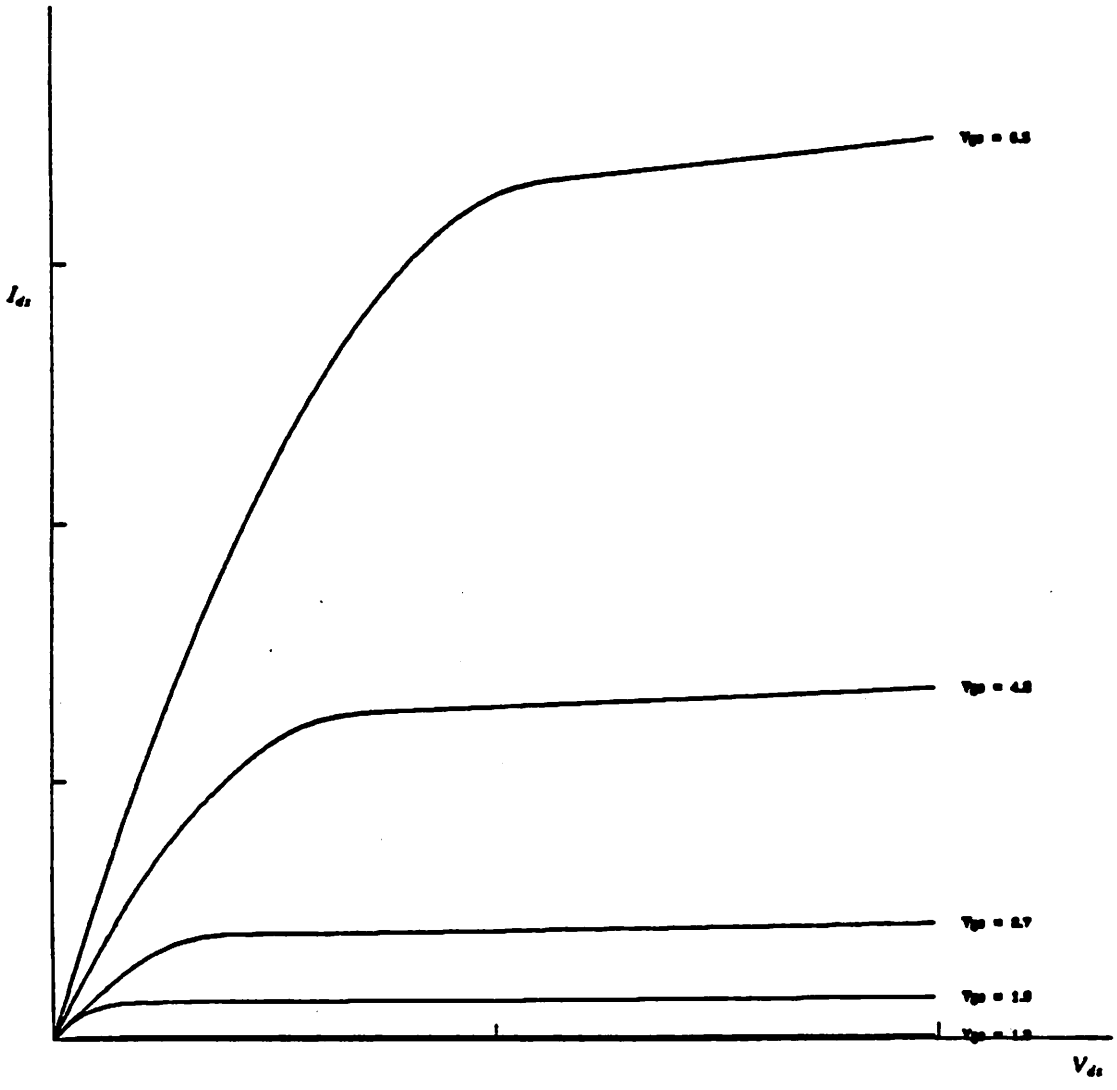
V_{ds}	I_{ds}				
	V_{gs} = 1.0 v	V_{gs} = 1.9 v	V_{gs} = 2.7 v	V_{gs} = 4.2 v	V_{gs} = 6.3 v
0.0	0.0	0.0	0.0	0.0	0.0
1.0	9.091e-7	1.414e-5	3.030e-5	6.061e-5	1.030e-4
2.0	9.184e-7	1.469e-5	4.082e-5	1.020e-4	1.878e-4
3.0	9.278e-7	1.485e-5	4.124e-5	1.237e-4	2.536e-4
4.0	9.375e-7	1.500e-5	4.167e-5	1.276e-4	3.000e-4
5.0	9.474e-7	1.516e-5	4.211e-5	1.290e-4	3.263e-4
6.0	9.575e-7	1.532e-5	4.255e-5	1.303e-4	3.336e-4
7.0	9.677e-7	1.548e-5	4.301e-5	1.317e-4	3.372e-4
8.0	9.783e-7	1.565e-5	4.348e-5	1.332e-4	3.409e-4
9.0	9.890e-7	1.582e-5	4.396e-5	1.346e-4	3.446e-4
10.0	1.000e-6	1.600e-5	4.444e-5	1.361e-4	3.484e-4

Table D.1

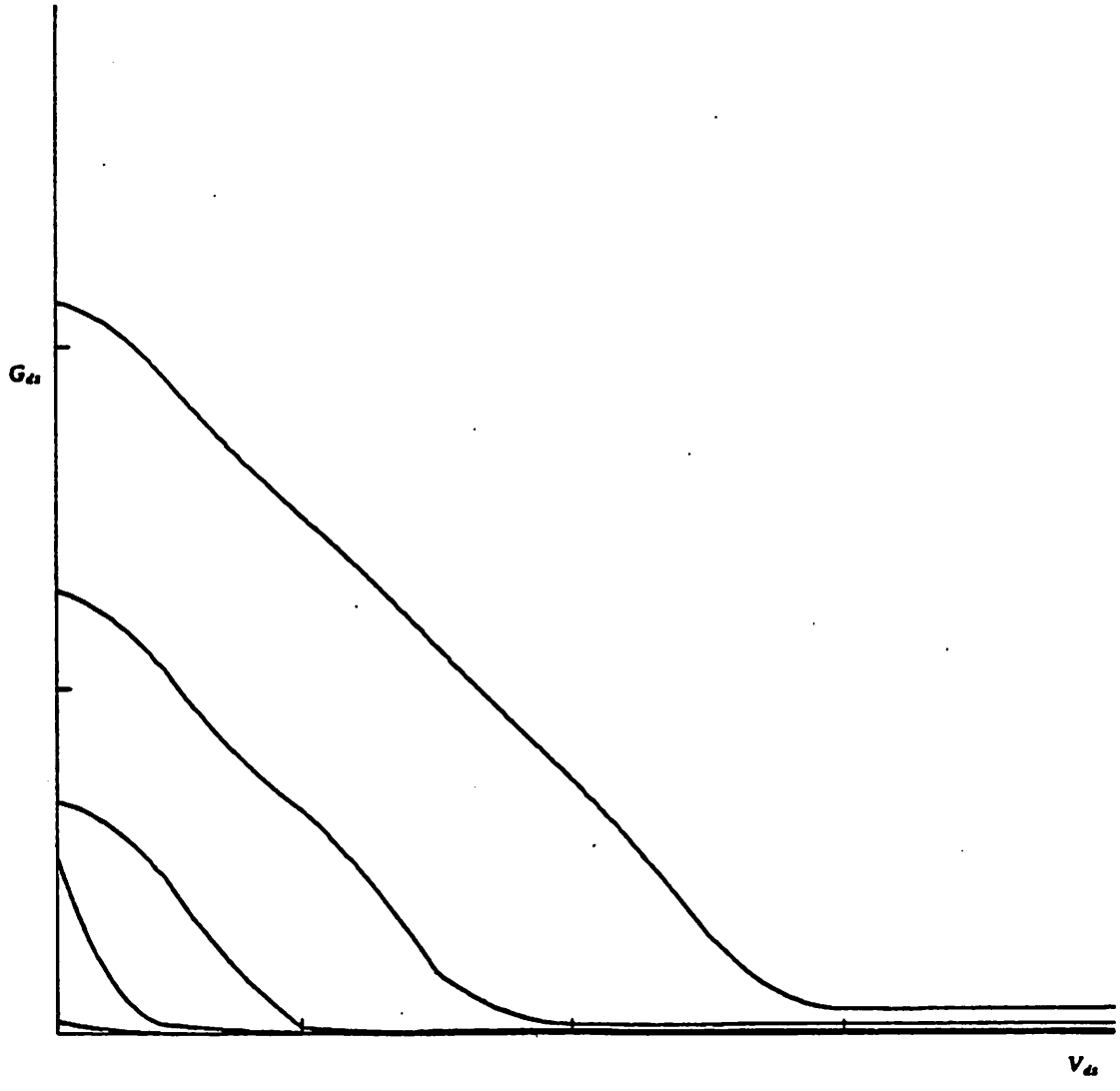
The constraint matrix and second derivative error matrix are set up. The initial derivatives are calculated using cubic splines, as shown in Figure D.4. The system of equations are iteratively solved for each value of V_{gs}. The monotonic requirements are met while minimizing the error in second derivatives. The current and derivative of the piecewise polynomial functions are plotted in Figures D.5 and D.6.



Cubic Spline - Initial Solution
Figure D.4



Final Solution I_{ds} vs. V_{ds}
Figure D.5



Final Solution G_{ds} vs. V_{ds}
Figure D.6

The square root of S_e for the five piecewise polynomial functions is given in Table D.2. In addition, an approximate maximum error(< true maximum) for the five piecewise polynomial functions, where the functions meet all the constraint requirements is also given in Table D.2.

V_{gs}	$\sqrt{S_e}$	Max $\sqrt{S_e}$
1.0	7.39e-9	1.33e-6
1.9	5.81e-8	1.78e-5
2.7	7.12e-6	1.55e-5
4.2	7.45e-6	2.69e-5
6.3	1.71e-6	3.78e-5

Table D.2

D.5. Numerical Accuracy

The accuracy of the polynomial calculation is limited by the finite precision of the numerical calculations. The finite precision of numeric operations result in errors due to both roundoff and truncation. The errors are only of concern at the boundaries of adjacent polynomials where they can cause discontinuities in the overall function.

$$p_i(x_{i+1}) \neq p_{i+1}(x_{i+1}) \tag{D.32}$$

The cubic polynomial can be rewritten in the following form for computation. Here the value of $\delta x_i = x - x_i$.

$$\begin{aligned} p_i(x) &= f_i(x_i)[1 - 3(\delta x_i \Delta x_i^{-1})^2 + 2(\delta x_i \Delta x_i^{-1})^3] \\ &+ f_i(x_{i+1})[3 - 2(\delta x_i \Delta x_i^{-1})](\delta x_i \Delta x_i^{-1})^2 \\ &+ f'_i(x_i)[1 - 2(\delta x_i \Delta x_i^{-1}) + (\delta x_i \Delta x_i^{-1})^2]\delta x_i \\ &+ f'_i(x_{i+1})[-1 + (\delta x_i \Delta x_i^{-1})]\delta x_i^2 \Delta x_i^{-1} \end{aligned} \tag{D.33}$$

For the polynomial to be continuous at its boundaries, the following limit must be satisfied.

$$\lim_{x \to x_{i-1}} \frac{x - x_i}{x_{i+1} - x_i} = 1 \qquad (D.34)$$

The polynomial $p_i(x)$ evaluated at $x = x_{i+1}$ is then equal to $f(x_{i+1})$, and the composite function is continuous.

D.6. Summary

Piecewise cubic polynomials are used to interpolate between discrete values of MOS transistor drain-to-source current. The piecewise polynomials provide a function that is continuous with continuous first derivatives. Values for the derivatives at the data points are chosen such that the composite function is monotonic, the "shape" of the derivative is correct, and the sum of the errors in second derivatives at the boundaries of adjacent polynomials are minimized.

APPENDIX E

POLY_MOS Program Listing

This appendix contains a listing of the POLY_MOS program written in the C language.

```
int    *area_ds, *area_gs, num_ds, num_gs, num_sb;
int    *ca, *da;
int    *cb, *db;
double *Ids, *Gds, *Ggs;
double *Vds, *Vgs, *Vsb;
double *DVds, *DVgs;
double **G_d, *F_d, *B_d;
double **G_g, *F_g, *B_g;
double **SB;
#undef fabs()
#undef max()
#undef min()
#define fabs(x) (x >= 0.0 ? (x) : -(x))
#define max(x,y) (x >= y ? (x) : (y))
#define min(x,y) (x < y ? (x) : (y))
main()
{

   int poly, i;
   extern int num_ds;
   extern double *B_d, *B_g, *Ids, *Gds;

   /* read-in data     */
   read_in();

   /* solve for polynomials Ids vs Vds */
   for( poly=0 ; poly < num_gs ; poly++ ) {
      ids_vs_vds(poly);
      for(i = 0 ; i < num_ds ; i++ ) {
         Gds[poly*num_ds + i] = B_d[i];
      }
   }

   for( poly=0 ; poly < num_ds ; poly++ ) {
      ids_vs_vgs(poly);
      for(i = 0 ; i < num_gs ; i++ ) {
         Ggs[i*num_ds + poly] = B_g[i];
      }
   }
   ids_vs_vsb();
   dump_result();
}

dump_result()
{

   FILE *fp, *fopen();
   int i, j, k, l;
   extern int num_ds, num_gs, num_sb;
   extern double  *Vds, *Vgs, *Vsb;
   extern double *Ids, *Gds, *Ggs, **SB;

   fp = fopen("results","w");
   fprintf(fp,"%d %d %d0,num_ds,num_gs,num_sb);

   for(k=0;k<num_sb;k++) {
```

```
      for(j=0;j<num_gs;j++) {
         for(i=0;i<num_ds;i++) {
            l=i+num_ds*(j+num_gs*k);
            fprintf(fp,"%e %e %e %e %e %e0,Vds[i],Vgs[j],Vsb[k],Ids[l],Gds[l],Ggs[l]);
         }
      }
   }
   for(i=0;i<num_ds;i++) {
      for(j=0;j<4;j++) {
         fprintf(fp,"%e ",SB[i][j]);
      }
      fprintf(fp,"0);
   }
   fclose(fp);
}
read_in()
{

   int i, j, k, l;
   double t1, t2, t3, t4;
   extern int num_ds, num_gs, num_sb;
   extern int *area_ds, *area_gs;
   extern int *ca, *da;
   extern int *cb, *db;
   extern double *Ids, *Gds, *Ggs;
   extern double *Vds, *Vgs, *Vsb;
   extern double *DVds, *DVgs;
   extern double **G_d, *F_d, *B_d;
   extern double **G_g, *F_g, *B_g;
   extern double **SB;

   scanf("%d %d %d0,&num_ds,&num_gs,&num_sb);

   /* ********************************************************** */
   /* Allocate Memory                                            */

   area_ds = (int *) malloc(num_ds*sizeof(int));
   area_gs = (int *) malloc(num_gs*sizeof(int));
   ca = (int *) malloc(num_ds*sizeof(int));
   da = (int *) malloc(num_ds*sizeof(int));
   cb = (int *) malloc(num_gs*sizeof(int));
   db = (int *) malloc(num_gs*sizeof(int));

   Vds = (double *) malloc(num_ds*sizeof(double));
   Vgs = (double *) malloc(num_gs*sizeof(double));
   DVds = (double *) malloc(num_ds*sizeof(double));
   DVgs = (double *) malloc(num_gs*sizeof(double));
   Vsb = (double *) malloc(num_sb*sizeof(double));
   Ids = (double *) malloc(num_ds*num_gs*num_sb*sizeof(double));
   Gds = (double *) malloc(num_ds*num_gs*num_sb*sizeof(double));
   Ggs = (double *) malloc(num_ds*num_gs*num_sb*sizeof(double));
   F_d = (double *) malloc(4*num_ds*sizeof(double));
   B_d = (double *) malloc(num_ds*sizeof(double));
   G_d = (double **) malloc(4*num_ds*sizeof(double));
   F_g = (double *) malloc(4*num_gs*sizeof(double));
   B_g = (double *) malloc(num_gs*sizeof(double));
```

```
  G_g = (double **) malloc(4*num_gs*sizeof(double));
  SB = (double **) malloc(num_ds*sizeof(double));

  for(i=0;i<4*num_ds;i++) {
     G_d[i] = (double *) malloc(num_ds*sizeof(double));
  }
  for(i=0;i<4*num_gs;i++) {
     G_g[i] = (double *) malloc(num_gs*sizeof(double));
  }
  for(i=0;i<num_ds;i++) {
     SB[i] = (double *) malloc(4*sizeof(double));
  }

  /* ********************************************************** */


  /* ********************************************************** */
  /* Store Data                                                 */
  for(i=0;i<num_sb;i++) {
     for(j=0;j<num_gs;j++) {
        for(k=0;k<num_ds;k++) {
           scanf("%e %e %e %e0.&t1.&t2.&t3.&t4);
           l = k + num_ds*(j + i*num_gs);
           Vds[k]=t1;
           Vgs[j]=t2;
           Vsb[i]=t3;
           Ids[l]=t4;
        }
     }
  }

  return;

}
init_deriv_vds()
{

  int i, j, k;
  extern double *Vds, *Vgs, *DVds;
  extern int *ca, *da, *area_ds, num_ds;
  extern double *B_d;

  /* ********************************************** */
  /* zero out B matrix */
  for(i=0;i<num_ds;i++) {
     B_d[i] = 0.0;
  }
  /* ********************************************** */


  /* ********************************************** */
  for(i=1;i<num_ds-1;i++) {
     B_d[i] = (DVds[i-1] + DVds[i])/2.0;
  }

  if(area_ds[0] == 1) {
```

```
        B_d[0] = 1.1*DVds[0];
    }
    else{
        B_d[0] = 0.9*DVds[0];
    }

    if(area_ds[num_ds-2] == 2) {
        B_d[num_ds-1] = 1.1*DVds[num_ds-2];
    }
    else{
        B_d[num_ds-1] = 0.9*DVds[num_ds-2];
    }

    /* ********************************************** */

    return;

}

init_deriv_vgs()
{

    int i, j, k;
    extern double *Vds, *Vgs, *DVds;
    extern int *ca, *da, *area_gs, num_gs;
    extern double *B_g;

    /* ********************************************** */
    /* zero out B matrix */
    for(i=0;i<num_gs;i++) {
        B_g[i] = 0.0;
    }
    /* ********************************************** */


    /* ********************************************** */
    for(i=1;i<num_gs-1;i++) {
        B_g[i] = (DVgs[i-1] + DVgs[i])/2.0;
    }

    if(area_gs[0] == 1) {
        B_g[0] = 1.1*DVgs[0];
    }
    else{
        B_g[0] = 0.9*DVgs[0];
    }

    if(area_gs[num_gs-2] == 2) {
        B_g[num_gs-1] = 1.1*DVgs[num_gs-2];
    }
    else{
        B_g[num_gs-1] = 0.9*DVgs[num_gs-2];
    }

    /* ********************************************** */
```

```
    return;

}

save_deriv(version,poly)
int version,poly;
{

    FILE *fp, *fopen();
    char *s;
    int i, j, k, l;

    extern int    num_ds, num_gs, num_sb;
    extern double *Vds, *Vgs, *Vsb, *B_d;
    extern double *Gds, *Ggs, *Ids;

    s = (char *) malloc(15*sizeof(char));

    sprintf(s,"vrs_%d_%d",poly,version);

    fp = fopen(s,"w");

    fprintf(fp,"%d %d %d\0",num_ds,num_gs,num_sb);
    for( i = 0 ; i<num_ds ; i++ ) {
       fprintf(fp,"%e %e %e %e %e\0",
                Vds[i],Vgs[poly],Vsb[0],Ids[poly*num_ds +i],B_d[i]);
    }

    fclose(fp);
}
ids_vs_vds(poly)
int poly;
{

    int i, j, k, bad;
    double tmp, err, merr, delta, mdelta, ddelta;
    double dc1, dc2;

    extern int *ca, *da, num_ds;
    extern double **G_d, *F_d, *B_d, *DVds;

    mdelta = 10.0;
    ddelta = 1.0;

    load_matrix_ids_vs_vds(poly);

    init_deriv_vds();

    for(k=0,delta=mdelta;k<1000;k++,delta+=ddelta) {

       for(i=0;i<num_ds;i++) {
          ca[i]=0;
          da[i]=0;
       }

       for(i=0,err=0.0;i<4*num_ds-1;i++) {
```

```
        for(j=0,tmp= 0.0;j<num_ds;j++) {
            tmp += G_d[i][j]*B_d[j];
        }
        tmp -= F_d[i];
        if(tmp<0.0) {
            err -= tmp;
            for(j=0;j<num_ds;j++) {
                if(G_d[i][j] != 0) {
                    ++ca[j];
                    if(G_d[i][j] > 0.0) ++da[j];
                    if(G_d[i][j] < 0.0) --da[j];
                }
            }
        }
    }

    for(j=0,bad=0;j<num_ds;j++) {
        if(j == num_ds - 1) {
            dc1 = 0.1*DVds[j-1];
            dc2 = 0.0;
        }
        else if(!j) {
            dc1 = 0.1*DVds[j];
            dc2 = 0.0;
        }
        else {
            dc1 = DVds[j];
            dc2 = DVds[j-1];
        }

        if(ca[j]) {
            ++bad;
            if(da[j] > 0) {
                B_d[j] += fabs(dc1-dc2)/delta;
            }
            if(da[j] < 0) {
                B_d[j] -= fabs(dc1-dc2)/delta;
            }
        }
    }

    if(!bad) break;

  }

  for(i=0;i<num_ds;i++) {
    if(ca[i]) {
      printf("ERROR:deriv[%d] is bad0,i);
      exit(0);
    }
  }

}
ids_vs_vgs(poly)
int poly;
{
```

```
int i, j, k, bad;
double tmp, err, merr, delta, mdelta, ddelta;
double dc1, dc2;

extern int *cb, *db, num_gs;
extern double **G_g, *F_g, *B_g, *DVgs;

mdelta = 10.0;
ddelta = 1.0;

load_matrix_ids_vs_vgs(poly);

init_deriv_vgs();

for(k=0,delta=mdelta;k<1000;k++,delta+=ddelta) {

   for(i=0;i<num_gs;i++) {
      cb[i]=0;
      db[i]=0;
   }

   for(i=0,err=0.0;i<4*num_gs-1;i++) {

      for(j=0,tmp= 0.0;j<num_gs;j++) {
         tmp += G_g[i][j]*B_g[j];
      }
      tmp -= F_g[i];
      if(tmp<0.0) {
         err -= tmp;
         for(j=0;j<num_gs;j++) {
            if(G_g[i][j] != 0) {
               ++cb[j];
               if(G_g[i][j] > 0.0) ++db[j];
               if(G_g[i][j] < 0.0) --db[j];
            }
         }
      }
   }

   for(j=0,bad=0;j<num_gs;j++) {
      if(j == num_gs - 1) {
         dc1 = 0.1*DVgs[j-1];
         dc2 = 0.0;
      }
      else if(!j) {
         dc1 = 0.1*DVgs[j];
         dc2 = 0.0;
      }
      else {
         dc1 = DVgs[j];
         dc2 = DVgs[j-1];
      }

      if(cb[j]) {
         ++bad;
         if(db[j] > 0) {
```

```
            B_g[j] += fabs(dc1-dc2)/delta;
          }
          if(db[j] < 0) {
            B_g[j] -= fabs(dc1-dc2)/delta;
          }
        }
      }

      if(!bad) break;

    }

    for(i=0;i<num_gs;i++) {
      if(cb[i]) {
        printf("ERROR:deriv[%d] is bad0,i);
        exit(0);
      }
    }

}
ids_vs_vsb()

{

    int i,j,k,l,n;
    double **amat, **atmat, **atamat, *bmat, *atbmat;
    double Vgse(), temp;
    extern int num_ds, num_gs, num_sb;
    extern double *Vds, *Vgs, *Vsb, **SB;

    /* ********************************************** */
    /* Allocate memory                                */

    amat = (double **) malloc (num_gs*num_sb*sizeof(double));
    bmat = (double *) malloc (num_gs*num_sb*sizeof(double));

    atbmat = (double *) malloc (4*sizeof(double));
    atmat = (double **) malloc (4*sizeof(double));
    atamat = (double **) malloc (4*sizeof(double));

    for(i=0;i<num_gs*num_sb;i++) {
      amat[i] = (double *) malloc (4*sizeof(double));
    }

    for(i=0;i<4;i++) {
      atmat[i] = (double *) malloc (num_gs*num_sb*sizeof(double));
    }

    for(i=0;i<4;i++) {
      atamat[i] = (double *) malloc (4*sizeof(double));
    }
    /* ********************************************** */

    /* ********************************************** */
    /* step through for each measured voltage         */
    for(i=0;i<num_ds;i++) {
```

```
/* ********************************************** */
/* Load a matrix (amat) */
for(j=0;j<num_gs;j++) {
   for(k=0;k<num_sb;k++) {
      amat[k*num_gs+j][0] = 1.0;
      amat[k*num_gs+j][1] = Vsb[k];
      amat[k*num_gs+j][2] = Vsb[k]*Vsb[k];
      amat[k*num_gs+j][3] = Vsb[k]*Vsb[k]*Vsb[k];
      bmat[k*num_gs+j]    = Vgse(i,j,k) - Vgs[j];
   }
}

/* ********************************************** */

/* ********************************************** */
/* calculate the transpose of the a matrix      */
for(j = 0; j < 4; ++j) {
   for(k = 0; k < num_gs*num_sb; ++k) {
      atmat[j][k] = amat[k][j];
   }
}

/* ********************************************** */

/* ********************************************** */
/* calculate the a transpose times a matrix     */
for(j = 0; j < 4; ++j) {
   for(k = 0; k < 4; ++k) {
      atamat[j][k] = 0.0;
      for(l = 0; l < num_gs*num_sb; ++l) {
         atamat[j][k] = atamat[j][k] + atmat[j][l]*amat[l][k];
      }
   }
}
/* ********************************************** */

/* ********************************************** */
/* calculate the a transpose times the b matrix */
for(j = 0; j < 4; ++j) {
   atbmat[j] = 0.0;
   for(l = 0; l < num_gs*num_sb; ++l) {
      atbmat[j] = atbmat[j] + atmat[j][l]*bmat[l];
   }
}
/* ********************************************** */

/* ********************************************** */
/* perform the LU factorization on the ata matrix      */
for(n = 0; n < 4; ++n) {
   for(j = n; j < 4; ++j) {
      temp=0.0;
      for(k = 0, temp=0.0;k < n; ++k) {
         temp += atamat[n][k]*atamat[k][j];
      }
      atamat[n][j] = atamat[n][j] - temp;
   }
```

```
            for(j = n+1; j < 4; ++j) {
               temp=0.0;
               for(k = 0;temp=0.0; k < n; ++k) {
                  temp += atamat[j][k]*atamat[k][n];
               }
               atamat[j][n] = (atamat[j][n] - temp)/atamat[n][n];
            }
         }
         /* ********************************************** */

         /* ********************************************** */
         /* perform foreward and backward substitution for answers      */
         for(n = 0; n < 4; ++n) {
            temp=0.0;
            for(k = 0; k < n; ++k) {
               temp += atamat[n][k]*atbmat[k];
            }
            atbmat[n] = atbmat[n] - temp;
         }
         for(n = 3; n >= 0; --n) {
            temp=0.0;
            for(k = 3; k > n; --k) {
               temp += atamat[n][k]*atbmat[k];
            }
            atbmat[n] = (atbmat[n] - temp)/atamat[n][n];
         }
         /* ********************************************** */

         /* ********************************************** */
         for(j = 0; j< 4; ++j) {
            SB[i][j]=atbmat[j];
         }

      }
   }

double
Vgse(p_ds,p_gs,p_sb)
int p_ds,p_gs,p_sb;
{

   int cur_pos, iter, i, j, k;
   double real_ids, c_ids;
   double g1, g2, i1, i2, dv, vg, ovg;
   double a, b, c, d;
   extern int num_ds, num_gs, num_sb;
   extern double *Vgs, *Ids, *Ggs;

   /* ********************************************** */
   /* get discrete current     */
   cur_pos = p_sb*num_ds*num_gs + p_gs*num_ds + p_ds;
   real_ids = Ids[cur_pos];

   /* ********************************************** */
   /* find which section are we in                */
   for(i=0;i<num_gs;i++) {
```

```
      if(Ids[i*num_ds+p_ds] > real_ids) break;
   }
   if(i == num_gs) {
      return(Vgs[--i]);
   }
   if(i == 0) {
      return(Vgs[0]);
   }

   /* ************************************************ */
   /* access data necessary and calculate coefficients */
   --i;
   i1 = Ids[i*num_ds+p_ds];
   g1 = Ggs[i*num_ds+p_ds];
   dv = Vgs[i+1] - Vgs[i];
   ++i;
   i2 = Ids[i*num_ds+p_ds];
   g2 = Ggs[i*num_ds+p_ds];
   --i;
   a = i1;
   b = g1;
   c = (3*(i2-i1)/dv - 2*g1 - g2)/dv;
   d = (g1 + g2 - 2*(i2-i1)/dv)/(dv*dv);

   /* ************************************************ */
   /* use a fixed-point iteration                      */
   vg=dv/2.0;
   ovg = dv/4;
   iter = 0;
   while(fabs(vg-ovg) > (1.0e-4)*max(fabs(vg),fabs(ovg)) + 1.0e-7) {
      ovg = vg;
      if(vg>=0.0 && vg<=dv)
         vg = (real_ids - a - c*vg*vg - d*vg*vg*vg)/b;
      else if (vg > dv)
         vg = (real_ids - i2)/g2;
      else ;
      c_ids = a + vg*(b + vg*(c + d*vg));
   }

   return(Vgs[i]+vg);

}
load_matrix_ids_vs_vds(poly)
int poly;
{

   int i, j;
   extern double *Ids, *Vds, *DVds;
   extern int *area_ds, num_ds;
   extern double **G_d, *F_d;

   /* ************************************************** */
   /* calculate delta                                    */
   for( i = 0, j = num_ds*poly ; i < num_ds - 1; i++, j++ ) {
      DVds[i] = (Ids[j+1] - Ids[j])/(Vds[i+1] - Vds[i]);
   }
```

```
/* ********************************************************** */

/* ********************************************************** */
/* Classify Area                                              */
for( i = 1 ; i < num_ds - 2 ; i++ ) {
   if(DVds[i] <= DVds[i-1] && DVds[i+1] <= DVds[i]) {
      area_ds[i] = 1;
   }
   else if(DVds[i] > DVds[i-1] && DVds[i+1] > DVds[i]) {
      area_ds[i] = 2;
   }
   else if(DVds[i] <= DVds[i-1] && DVds[i+1] > DVds[i]) {
      area_ds[i] = 3;
   }
   else {
      area_ds[i] = 4;
   }
}

if(area_ds[1] == 1 || area_ds[1] == 3) {
   area_ds[0] = 1;
}
else {
   area_ds[0] = 2;
}

if(area_ds[num_ds-3] == 2 || area_ds[num_ds-3] == 3) {
   area_ds[num_ds-2] = 2;
}
else {
   area_ds[num_ds-2] = 1;
}
/* ********************************************************** */


/* ********************************************************** */
/* Zero out matrix                                            */
for(i=0;i<4*num_ds;i++) {
   F_d[i] = 0.0;
   for(j=0;j<num_ds;j++) {
      G_d[i][j] = 0.0;
   }
}
/* ********************************************************** */


/* ********************************************************** */
/* Load Matrix With Inequality Terms                          */
for(i=1;i<num_ds - 1;i++) {

   /* condition 1  */
   if(area_ds[i]==1 || area_ds[i]==3) {
      G_d[2*i][i] = 2.0;
      G_d[2*i][i+1] = 1.0;
      F_d[2*i] = 3.0*DVds[i];
   }
```

```
    else {
        G_d[2*i][i] = -2.0;
        G_d[2*i][i+1] = -1.0;
        F_d[2*i] = -3.0*DVds[i];
    }

    /* condition 2  */
    if(area_ds[i]==1 || area_ds[i]==4) {
        G_d[2*i+1][i] = -1.0;
        G_d[2*i+1][i+1] = -2.0;
        F_d[2*i+1] = -3.0*DVds[i];
    }
    else {
        G_d[2*i+1][i] = 1.0;
        G_d[2*i+1][i+1] = 2.0;
        F_d[2*i+1] = 3.0*DVds[i];
    }
}

for( i = 2*num_ds - 2, j = 0 ; j < num_ds ; i++, j++ ) {
    G_d[i][j] = 1.0;
    if(!j) {
        if(area_ds[0] == 1) {
            F_d[i] = DVds[0];
        }
        else {
            F_d[i] = 0.0;
        }
    }
    if(j == num_ds - 1) {
        if(area_ds[num_ds-2] == 1) {
            F_d[i] = 0.0;
        }
        else {
            F_d[i] = DVds[j-1];
        }
    }
    else {
        F_d[i] = min(DVds[j-1],DVds[j]);
    }
}

for( i = 3*num_ds - 2, j = 0 ; j < num_ds - 1 ; i++, j++ ) {
    G_d[i][j] = -1.0;
    if(!j ) {
        if(area_ds[0] == 1) {
            F_d[i] = -1.0e10;
        }
        else {
            F_d[i] = -DVds[0];
        }
    }

    if(j == num_ds - 1) {
        if(area_ds[num_ds-2] == 1) {
            F_d[i] = -DVds[j-1];
```

```
        }
        else {
            F_d[i] = -1.0e10;
        }
      }

      else {
        F_d[i] = - max(DVds[j-1],DVds[j]);
      }
    }

    for( i = 4*num_ds - 2, j = 0 ; j < num_ds - 1 ; j++ ) {
      if(area_ds[j] == 3) {
        G_d[i][j] = -1.0;
        G_d[i][j+1] = -1.0;
        F_d[i] = -3.0*DVds[j];
        i++;
      }
    }
    /* ********************************************************** */

    return;
}



load_matrix_ids_vs_vgs(poly)
int poly;
{

    int i, j;
    extern double *Ids, *Vgs, *DVgs;
    extern int *area_gs, num_gs, num_ds;
    extern double **G_g, *F_g;

    /* ********************************************************** */
    /* calculate delta                                            */
    for( i = 0, j = poly ; i < num_gs - 1; i++, j+=num_ds ) {
      DVgs[i] = (Ids[j+num_ds] - Ids[j])/(Vgs[i+1] - Vgs[i]);
    }
    /* ********************************************************** */

    /* ********************************************************** */
    /* Classify Area                                              */
    for( i = 1 ; i < num_gs - 2 ; i++ ) {
      if(DVgs[i] <= DVgs[i-1] && DVgs[i+1] <= DVgs[i]) {
        area_gs[i] = 1;
      }
      else if(DVgs[i] > DVgs[i-1] && DVgs[i+1] > DVgs[i]) {
        area_gs[i] = 2;
      }
      else if(DVgs[i] <= DVgs[i-1] && DVgs[i+1] > DVgs[i]) {
        area_gs[i] = 3;
      }
      else {
```

```
      area_gs[i] = 4;
   }
}

if(area_gs[1] == 1 || area_gs[1] == 3) {
   area_gs[0] = 1;
}
else {
   area_gs[0] = 2;
}

if(area_gs[num_gs-3] == 2 || area_gs[num_gs-3] == 3) {
   area_gs[num_gs-2] = 2;
}
else {
   area_gs[num_gs-2] = 1;
}
/* ******************************************************** */


/* ******************************************************** */
/* Zero out matrix                                          */
for(i=0;i<4*num_gs;i++) {
   F_g[i] = 0.0;
   for(j=0;j<num_gs;j++) {
      G_g[i][j] = 0.0;
   }
}
/* ******************************************************** */


/* ******************************************************** */
/* Load Matrix With Inequality Terms                        */
for(i=1;i<num_gs - 1;i++) {

   /* condition 1  */
   if(area_gs[i]==1 || area_gs[i]==3) {
      G_g[2*i][i] = 2.0;
      G_g[2*i][i+1] = 1.0;
      F_g[2*i] = 3.0*DVgs[i];
   }
   else {
      G_g[2*i][i] = -2.0;
      G_g[2*i][i+1] = -1.0;
      F_g[2*i] = -3.0*DVgs[i];
   }

   /* condition 2  */
   if(area_gs[i]==1 || area_gs[i]==4) {
      G_g[2*i+1][i] = -1.0;
      G_g[2*i+1][i+1] = -2.0;
      F_g[2*i+1] = -3.0*DVgs[i];
   }
   else {
      G_g[2*i+1][i] = 1.0;
      G_g[2*i+1][i+1] = 2.0;
```

```
      F_g[2*i+1] = 3.0*DVgs[i];
   }
}

for( i = 2*num_gs - 2, j = 0 ; j < num_gs ; i++, j++ ) {
   G_g[i][j] = 1.0;
   if(!j) {
      if(area_gs[0] == 1) {
         F_g[i] = DVgs[0];
      }
      else {
         F_g[i] = 0.0;
      }
   }
   if(j == num_gs - 1) {
      if(area_gs[num_gs-2] == 1) {
         F_g[i] = 0.0;
      }
      else {
         F_g[i] = DVgs[j-1];
      }
   }
   else {
      F_g[i] = min(DVgs[j-1],DVgs[j]);
   }
}

for( i = 3*num_gs - 2, j = 0 ; j < num_gs - 1 ; i++, j++ ) {
   G_g[i][j] = -1.0;
   if(!j ) {
      if(area_gs[0] == 1) {
         F_g[i] = -1.0e10;
      }
      else {
         F_g[i] = -DVgs[0];
      }
   }

   if(j == num_gs - 1) {
      if(area_gs[num_gs-2] == 1) {
         F_g[i] = -DVgs[j-1];
      }
      else {
         F_g[i] = -1.0e10;
      }
   }

   else {
      F_g[i] = - max(DVgs[j-1],DVgs[j]);
   }
}

for( i = 4*num_gs - 2, j = 0 ; j < num_gs - 1 ; j++ ) {
   if(area_gs[j] == 3) {
      G_g[i][j] = -1.0;
      G_g[i][j+1] = -1.0;
```

```
            F_g[i] = -3.0*DVgs[j];
            i++;
        }
    }
    /* ************************************************** */

    return;
}
```

APPENDIX F

Example Data

This appendix contains a listing of the data used to generate the examples presented in Chapter 4.

Data Generated from the Shichman-Hodges Model

VDS	VGS	VSB	IDS
0.000000e+00	1.000000e+00	0.000000e+00	2.000000e-08
1.250000e-01	1.000000e+00	0.000000e+00	2.014062e-08
2.500000e-01	1.000000e+00	0.000000e+00	2.031250e-08
3.750000e-01	1.000000e+00	0.000000e+00	2.051562e-08
5.000000e-01	1.000000e+00	0.000000e+00	2.075000e-08
7.500000e-01	1.000000e+00	0.000000e+00	2.131250e-08
1.000000e+00	1.000000e+00	0.000000e+00	2.200000e-08
1.250000e+00	1.000000e+00	0.000000e+00	2.281250e-08
1.500000e+00	1.000000e+00	0.000000e+00	2.375000e-08
2.000000e+00	1.000000e+00	0.000000e+00	2.600000e-08
2.500000e+00	1.000000e+00	0.000000e+00	2.875000e-08
3.000000e+00	1.000000e+00	0.000000e+00	3.200000e-08
4.000000e+00	1.000000e+00	0.000000e+00	4.000000e-08
5.000000e+00	1.000000e+00	0.000000e+00	5.000000e-08
1.000000e+01	1.000000e+00	0.000000e+00	1.300000e-07
2.000000e+01	1.000000e+00	0.000000e+00	4.400000e-07
0.000000e+00	1.500000e+00	0.000000e+00	3.750000e-08
1.250000e-01	1.500000e+00	0.000000e+00	1.134132e-06
2.500000e-01	1.500000e+00	0.000000e+00	1.922235e-06
3.750000e-01	1.500000e+00	0.000000e+00	2.399477e-06
5.000000e-01	1.500000e+00	0.000000e+00	2.563503e-06
7.500000e-01	1.500000e+00	0.000000e+00	2.576884e-06
1.000000e+00	1.500000e+00	0.000000e+00	2.590520e-06
1.250000e+00	1.500000e+00	0.000000e+00	2.604415e-06
1.500000e+00	1.500000e+00	0.000000e+00	2.618570e-06
2.000000e+00	1.500000e+00	0.000000e+00	2.647667e-06
2.500000e+00	1.500000e+00	0.000000e+00	2.677829e-06
3.000000e+00	1.500000e+00	0.000000e+00	2.709074e-06
4.000000e+00	1.500000e+00	0.000000e+00	2.774891e-06
5.000000e+00	1.500000e+00	0.000000e+00	2.845278e-06
1.000000e+01	1.500000e+00	0.000000e+00	3.272500e-06
2.000000e+01	1.500000e+00	0.000000e+00	4.624167e-06
0.000000e+00	2.000000e+00	0.000000e+00	6.000000e-08
1.250000e-01	2.000000e+00	0.000000e+00	2.409765e-06
2.500000e-01	2.000000e+00	0.000000e+00	4.457297e-06
3.750000e-01	2.000000e+00	0.000000e+00	6.200314e-06
5.000000e-01	2.000000e+00	0.000000e+00	7.636508e-06
7.500000e-01	2.000000e+00	0.000000e+00	9.579079e-06
1.000000e+00	2.000000e+00	0.000000e+00	1.026608e-05
1.250000e+00	2.000000e+00	0.000000e+00	1.031922e-05
1.500000e+00	2.000000e+00	0.000000e+00	1.037303e-05
2.000000e+00	2.000000e+00	0.000000e+00	1.048267e-05
2.500000e+00	2.000000e+00	0.000000e+00	1.059507e-05
3.000000e+00	2.000000e+00	0.000000e+00	1.071030e-05
4.000000e+00	2.000000e+00	0.000000e+00	1.094957e-05
5.000000e+00	2.000000e+00	0.000000e+00	1.120111e-05
1.000000e+01	2.000000e+00	0.000000e+00	1.267000e-05
2.000000e+01	2.000000e+00	0.000000e+00	1.714667e-05
0.000000e+00	2.500000e+00	0.000000e+00	8.750000e-08
1.250000e-01	2.500000e+00	0.000000e+00	3.690398e-06

2.500000e-01 2.500000e+00 0.000000e+00 6.997360e-06
3.750000e-01 2.500000e+00 0.000000e+00 1.000615e-05
5.000000e-01 2.500000e+00 0.000000e+00 1.271451e-05
7.500000e-01 2.500000e+00 0.000000e+00 1.722079e-05
1.000000e+00 2.500000e+00 0.000000e+00 2.049766e-05
1.250000e+00 2.500000e+00 0.000000e+00 2.252621e-05
1.500000e+00 2.500000e+00 0.000000e+00 2.328713e-05
2.000000e+00 2.500000e+00 0.000000e+00 2.353100e-05
2.500000e+00 2.500000e+00 0.000000e+00 2.378046e-05
3.000000e+00 2.500000e+00 0.000000e+00 2.403567e-05
4.000000e+00 2.500000e+00 0.000000e+00 2.456402e-05
5.000000e+00 2.500000e+00 0.000000e+00 2.511750e-05
1.000000e+01 2.500000e+00 0.000000e+00 2.832250e-05
2.000000e+01 2.500000e+00 0.000000e+00 3.800750e-05
0.000000e+00 3.000000e+00 0.000000e+00 1.200000e-07
1.250000e-01 3.000000e+00 0.000000e+00 4.976030e-06
2.500000e-01 3.000000e+00 0.000000e+00 9.542423e-06
3.750000e-01 3.000000e+00 0.000000e+00 1.381699e-05
5.000000e-01 3.000000e+00 0.000000e+00 1.779752e-05
7.500000e-01 3.000000e+00 0.000000e+00 2.486751e-05
1.000000e+00 3.000000e+00 0.000000e+00 3.073424e-05
1.250000e+00 3.000000e+00 0.000000e+00 3.537922e-05
1.500000e+00 3.000000e+00 0.000000e+00 3.878354e-05
2.000000e+00 3.000000e+00 0.000000e+00 4.179267e-05
2.500000e+00 3.000000e+00 0.000000e+00 4.223401e-05
3.000000e+00 3.000000e+00 0.000000e+00 4.268519e-05
4.000000e+00 3.000000e+00 0.000000e+00 4.361826e-05
5.000000e+00 3.000000e+00 0.000000e+00 4.459444e-05
1.000000e+01 3.000000e+00 0.000000e+00 5.023000e-05
2.000000e+01 3.000000e+00 0.000000e+00 6.720667e-05
0.000000e+00 3.500000e+00 0.000000e+00 1.575000e-07
1.250000e-01 3.500000e+00 0.000000e+00 6.266663e-06
2.500000e-01 3.500000e+00 0.000000e+00 1.209249e-05
3.750000e-01 3.500000e+00 0.000000e+00 1.763283e-05
5.000000e-01 3.500000e+00 0.000000e+00 2.288552e-05
7.500000e-01 3.500000e+00 0.000000e+00 3.251922e-05
1.000000e+00 3.500000e+00 0.000000e+00 4.097583e-05
1.250000e+00 3.500000e+00 0.000000e+00 4.823724e-05
1.500000e+00 3.500000e+00 0.000000e+00 5.428496e-05
2.000000e+00 3.500000e+00 0.000000e+00 6.266350e-05
2.500000e+00 3.500000e+00 0.000000e+00 6.595572e-05
3.000000e+00 3.500000e+00 0.000000e+00 6.665886e-05
4.000000e+00 3.500000e+00 0.000000e+00 6.811228e-05
5.000000e+00 3.500000e+00 0.000000e+00 6.963194e-05
1.000000e+01 3.500000e+00 0.000000e+00 7.839250e-05
2.000000e+01 3.500000e+00 0.000000e+00 1.047442e-04
0.000000e+00 4.000000e+00 0.000000e+00 2.000000e-07
1.250000e-01 4.000000e+00 0.000000e+00 7.562296e-06
2.500000e-01 4.000000e+00 0.000000e+00 1.464755e-05
3.750000e-01 4.000000e+00 0.000000e+00 2.145366e-05
5.000000e-01 4.000000e+00 0.000000e+00 2.797853e-05
7.500000e-01 4.000000e+00 0.000000e+00 4.017593e-05
1.000000e+00 4.000000e+00 0.000000e+00 5.122241e-05
1.250000e+00 4.000000e+00 0.000000e+00 6.110025e-05
1.500000e+00 4.000000e+00 0.000000e+00 6.979138e-05
2.000000e+00 4.000000e+00 0.000000e+00 8.353933e-05

2.500000e+00 4.000000e+00 0.000000e+00 9.231401e-05
3.000000e+00 4.000000e+00 0.000000e+00 9.595668e-05
4.000000e+00 4.000000e+00 0.000000e+00 9.804609e-05
5.000000e+00 4.000000e+00 0.000000e+00 1.002300e-04
1.000000e+01 4.000000e+00 0.000000e+00 1.128100e-04
2.000000e+01 4.000000e+00 0.000000e+00 1.506200e-04
0.000000e+00 4.500000e+00 0.000000e+00 2.475000e-07
1.250000e-01 4.500000e+00 0.000000e+00 8.862929e-06
2.500000e-01 4.500000e+00 0.000000e+00 1.720761e-05
3.750000e-01 4.500000e+00 0.000000e+00 2.527950e-05
5.000000e-01 4.500000e+00 0.000000e+00 3.307653e-05
7.500000e-01 4.500000e+00 0.000000e+00 4.783764e-05
1.000000e+00 4.500000e+00 0.000000e+00 6.147399e-05
1.250000e+00 4.500000e+00 0.000000e+00 7.396826e-05
1.500000e+00 4.500000e+00 0.000000e+00 8.530280e-05
2.000000e+00 4.500000e+00 0.000000e+00 1.044202e-04
2.500000e+00 4.500000e+00 0.000000e+00 1.186773e-04
3.000000e+00 4.500000e+00 0.000000e+00 1.279191e-04
4.000000e+00 4.500000e+00 0.000000e+00 1.334197e-04
5.000000e+00 4.500000e+00 0.000000e+00 1.363886e-04
1.000000e+01 4.500000e+00 0.000000e+00 1.534825e-04
2.000000e+01 4.500000e+00 0.000000e+00 2.048342e-04
0.000000e+00 5.000000e+00 0.000000e+00 3.000000e-07
1.250000e-01 5.000000e+00 0.000000e+00 1.016856e-05
2.500000e-01 5.000000e+00 0.000000e+00 1.977267e-05
3.750000e-01 5.000000e+00 0.000000e+00 2.911034e-05
5.000000e-01 5.000000e+00 0.000000e+00 3.817954e-05
7.500000e-01 5.000000e+00 0.000000e+00 5.550436e-05
1.000000e+00 5.000000e+00 0.000000e+00 7.173057e-05
1.250000e+00 5.000000e+00 0.000000e+00 8.684127e-05
1.500000e+00 5.000000e+00 0.000000e+00 1.008192e-04
2.000000e+00 5.000000e+00 0.000000e+00 1.253060e-04
2.500000e+00 5.000000e+00 0.000000e+00 1.450456e-04
3.000000e+00 5.000000e+00 0.000000e+00 1.598865e-04
4.000000e+00 5.000000e+00 0.000000e+00 1.742330e-04
5.000000e+00 5.000000e+00 0.000000e+00 1.781078e-04
1.000000e+01 5.000000e+00 0.000000e+00 2.004100e-04
2.000000e+01 5.000000e+00 0.000000e+00 2.673867e-04
0.000000e+00 6.000000e+00 0.000000e+00 4.200000e-07
1.250000e-01 6.000000e+00 0.000000e+00 1.279483e-05
2.500000e-01 6.000000e+00 0.000000e+00 2.491780e-05
3.750000e-01 6.000000e+00 0.000000e+00 3.678701e-05
5.000000e-01 6.000000e+00 0.000000e+00 4.840055e-05
7.500000e-01 6.000000e+00 0.000000e+00 7.085278e-05
1.000000e+00 6.000000e+00 0.000000e+00 9.225873e-05
1.250000e+00 6.000000e+00 0.000000e+00 1.126023e-04
1.500000e+00 6.000000e+00 0.000000e+00 1.318670e-04
2.000000e+00 6.000000e+00 0.000000e+00 1.670927e-04
2.500000e+00 6.000000e+00 0.000000e+00 1.977972e-04
3.000000e+00 6.000000e+00 0.000000e+00 2.238363e-04
4.000000e+00 6.000000e+00 0.000000e+00 2.613096e-04
5.000000e+00 6.000000e+00 0.000000e+00 2.782278e-04
1.000000e+01 6.000000e+00 0.000000e+00 3.130300e-04
2.000000e+01 6.000000e+00 0.000000e+00 4.175067e-04

Data Generated from the SPICE Level-2 Model

VDS	VGS	VSB	IDS
0.000000e+00	1.000000e+00	0.000000e+00	2.000000e-08
1.250000e-01	1.000000e+00	0.000000e+00	2.014062e-08
2.500000e-01	1.000000e+00	0.000000e+00	2.031250e-08
3.750000e-01	1.000000e+00	0.000000e+00	2.051562e-08
5.000000e-01	1.000000e+00	0.000000e+00	2.075000e-08
7.500000e-01	1.000000e+00	0.000000e+00	2.131250e-08
1.000000e+00	1.000000e+00	0.000000e+00	2.200000e-08
1.250000e+00	1.000000e+00	0.000000e+00	2.281250e-08
1.500000e+00	1.000000e+00	0.000000e+00	2.375000e-08
2.000000e+00	1.000000e+00	0.000000e+00	2.600000e-08
2.500000e+00	1.000000e+00	0.000000e+00	2.875000e-08
3.000000e+00	1.000000e+00	0.000000e+00	3.200000e-08
4.000000e+00	1.000000e+00	0.000000e+00	4.000000e-08
5.000000e+00	1.000000e+00	0.000000e+00	5.000000e-08
1.000000e+01	1.000000e+00	0.000000e+00	1.300000e-07
2.000000e+01	1.000000e+00	0.000000e+00	4.400000e-07
0.000000e+00	1.500000e+00	0.000000e+00	3.750000e-08
1.250000e-01	1.500000e+00	0.000000e+00	1.104777e-06
2.500000e-01	1.500000e+00	0.000000e+00	1.807945e-06
3.750000e-01	1.500000e+00	0.000000e+00	2.148528e-06
5.000000e-01	1.500000e+00	0.000000e+00	2.186923e-06
7.500000e-01	1.500000e+00	0.000000e+00	2.198392e-06
1.000000e+00	1.500000e+00	0.000000e+00	2.210098e-06
1.250000e+00	1.500000e+00	0.000000e+00	2.222042e-06
1.500000e+00	1.500000e+00	0.000000e+00	2.234225e-06
2.000000e+00	1.500000e+00	0.000000e+00	2.259319e-06
2.500000e+00	1.500000e+00	0.000000e+00	2.285393e-06
3.000000e+00	1.500000e+00	0.000000e+00	2.312464e-06
4.000000e+00	1.500000e+00	0.000000e+00	2.369659e-06
5.000000e+00	1.500000e+00	0.000000e+00	2.431040e-06
1.000000e+01	1.500000e+00	0.000000e+00	2.806483e-06
2.000000e+01	1.500000e+00	0.000000e+00	4.002810e-06
0.000000e+00	2.000000e+00	0.000000e+00	6.000000e-08
1.250000e-01	2.000000e+00	0.000000e+00	2.380410e-06
2.500000e-01	2.000000e+00	0.000000e+00	4.343008e-06
3.750000e-01	2.000000e+00	0.000000e+00	5.949365e-06
5.000000e-01	2.000000e+00	0.000000e+00	7.200208e-06
7.500000e-01	2.000000e+00	0.000000e+00	8.635408e-06
1.000000e+00	2.000000e+00	0.000000e+00	8.841920e-06
1.250000e+00	2.000000e+00	0.000000e+00	8.887758e-06
1.500000e+00	2.000000e+00	0.000000e+00	8.934185e-06
2.000000e+00	2.000000e+00	0.000000e+00	9.028835e-06
2.500000e+00	2.000000e+00	0.000000e+00	9.125931e-06
3.000000e+00	2.000000e+00	0.000000e+00	9.225534e-06
4.000000e+00	2.000000e+00	0.000000e+00	9.432524e-06
5.000000e+00	2.000000e+00	0.000000e+00	9.650357e-06
1.000000e+01	2.000000e+00	0.000000e+00	1.092540e-05
2.000000e+01	2.000000e+00	0.000000e+00	1.482054e-05
0.000000e+00	2.500000e+00	0.000000e+00	8.750000e-08
1.250000e-01	2.500000e+00	0.000000e+00	3.661043e-06

2.500000e-01 2.500000e+00 0.000000e+00 6.883071e-06
3.750000e-01 2.500000e+00 0.000000e+00 9.755203e-06
5.000000e-01 2.500000e+00 0.000000e+00 1.227821e-05
7.500000e-01 2.500000e+00 0.000000e+00 1.627712e-05
1.000000e+00 2.500000e+00 0.000000e+00 1.887642e-05
1.250000e+00 2.500000e+00 0.000000e+00 2.006818e-05
1.500000e+00 2.500000e+00 0.000000e+00 2.022343e-05
2.000000e+00 2.500000e+00 0.000000e+00 2.043539e-05
2.500000e+00 2.500000e+00 0.000000e+00 2.065227e-05
3.000000e+00 2.500000e+00 0.000000e+00 2.087420e-05
4.000000e+00 2.500000e+00 0.000000e+00 2.133382e-05
5.000000e+00 2.500000e+00 0.000000e+00 2.181552e-05
1.000000e+01 2.500000e+00 0.000000e+00 2.460777e-05
2.000000e+01 2.500000e+00 0.000000e+00 3.305453e-05
0.000000e+00 3.000000e+00 0.000000e+00 1.200000e-07
1.250000e-01 3.000000e+00 0.000000e+00 4.946675e-06
2.500000e-01 3.000000e+00 0.000000e+00 9.428133e-06
3.750000e-01 3.000000e+00 0.000000e+00 1.356604e-05
5.000000e-01 3.000000e+00 0.000000e+00 1.736122e-05
7.500000e-01 3.000000e+00 0.000000e+00 2.392383e-05
1.000000e+00 3.000000e+00 0.000000e+00 2.911300e-05
1.250000e+00 3.000000e+00 0.000000e+00 3.292119e-05
1.500000e+00 3.000000e+00 0.000000e+00 3.533782e-05
2.000000e+00 3.000000e+00 0.000000e+00 3.654473e-05
2.500000e+00 3.000000e+00 0.000000e+00 3.693083e-05
3.000000e+00 3.000000e+00 0.000000e+00 3.732559e-05
4.000000e+00 3.000000e+00 0.000000e+00 3.814215e-05
5.000000e+00 3.000000e+00 0.000000e+00 3.899664e-05
1.000000e+01 3.000000e+00 0.000000e+00 4.393247e-05
2.000000e+01 3.000000e+00 0.000000e+00 5.880996e-05
0.000000e+00 3.500000e+00 0.000000e+00 1.575000e-07
1.250000e-01 3.500000e+00 0.000000e+00 6.237308e-06
2.500000e-01 3.500000e+00 0.000000e+00 1.197820e-05
3.750000e-01 3.500000e+00 0.000000e+00 1.738188e-05
5.000000e-01 3.500000e+00 0.000000e+00 2.244922e-05
7.500000e-01 3.500000e+00 0.000000e+00 3.157555e-05
1.000000e+00 3.500000e+00 0.000000e+00 3.935458e-05
1.250000e+00 3.500000e+00 0.000000e+00 4.577920e-05
1.500000e+00 3.500000e+00 0.000000e+00 5.083924e-05
2.000000e+00 3.500000e+00 0.000000e+00 5.681422e-05
2.500000e+00 3.500000e+00 0.000000e+00 5.800940e-05
3.000000e+00 3.500000e+00 0.000000e+00 5.862801e-05
4.000000e+00 3.500000e+00 0.000000e+00 5.990684e-05
5.000000e+00 3.500000e+00 0.000000e+00 6.124416e-05
1.000000e+01 3.500000e+00 0.000000e+00 6.895625e-05
2.000000e+01 3.500000e+00 0.000000e+00 9.216249e-05
0.000000e+00 4.000000e+00 0.000000e+00 2.000000e-07
1.250000e-01 4.000000e+00 0.000000e+00 7.532941e-06
2.500000e-01 4.000000e+00 0.000000e+00 1.453326e-05
3.750000e-01 4.000000e+00 0.000000e+00 2.120272e-05
5.000000e-01 4.000000e+00 0.000000e+00 2.754223e-05
7.500000e-01 4.000000e+00 0.000000e+00 3.923226e-05
1.000000e+00 4.000000e+00 0.000000e+00 4.960116e-05
1.250000e+00 4.000000e+00 0.000000e+00 5.864222e-05
1.500000e+00 4.000000e+00 0.000000e+00 6.634565e-05
2.000000e+00 4.000000e+00 0.000000e+00 7.769006e-05

2.500000e+00 4.000000e+00 0.000000e+00 8.351983e-05
3.000000e+00 4.000000e+00 0.000000e+00 8.481835e-05
4.000000e+00 4.000000e+00 0.000000e+00 8.666562e-05
5.000000e+00 4.000000e+00 0.000000e+00 8.859664e-05
1.000000e+01 4.000000e+00 0.000000e+00 9.972247e-05
2.000000e+01 4.000000e+00 0.000000e+00 1.331700e-04
0.000000e+00 4.500000e+00 0.000000e+00 2.475000e-07
1.250000e-01 4.500000e+00 0.000000e+00 8.833574e-06
2.500000e-01 4.500000e+00 0.000000e+00 1.709332e-05
3.750000e-01 4.500000e+00 0.000000e+00 2.502855e-05
5.000000e-01 4.500000e+00 0.000000e+00 3.264023e-05
7.500000e-01 4.500000e+00 0.000000e+00 4.689397e-05
1.000000e+00 4.500000e+00 0.000000e+00 5.985275e-05
1.250000e+00 4.500000e+00 0.000000e+00 7.151023e-05
1.500000e+00 4.500000e+00 0.000000e+00 8.185707e-05
2.000000e+00 4.500000e+00 0.000000e+00 9.857089e-05
2.500000e+00 4.500000e+00 0.000000e+00 1.098831e-04
3.000000e+00 4.500000e+00 0.000000e+00 1.156634e-04
4.000000e+00 4.500000e+00 0.000000e+00 1.184485e-04
5.000000e+00 4.500000e+00 0.000000e+00 1.210848e-04
1.000000e+01 4.500000e+00 0.000000e+00 1.362657e-04
2.000000e+01 4.500000e+00 0.000000e+00 1.818784e-04
0.000000e+00 5.000000e+00 0.000000e+00 3.000000e-07
1.250000e-01 5.000000e+00 0.000000e+00 1.013921e-05
2.500000e-01 5.000000e+00 0.000000e+00 1.965838e-05
3.750000e-01 5.000000e+00 0.000000e+00 2.885939e-05
5.000000e-01 5.000000e+00 0.000000e+00 3.774324e-05
7.500000e-01 5.000000e+00 0.000000e+00 5.456069e-05
1.000000e+00 5.000000e+00 0.000000e+00 7.010933e-05
1.250000e+00 5.000000e+00 0.000000e+00 8.438324e-05
1.500000e+00 5.000000e+00 0.000000e+00 9.737349e-05
2.000000e+00 5.000000e+00 0.000000e+00 1.194567e-04
2.500000e+00 5.000000e+00 0.000000e+00 1.362514e-04
3.000000e+00 5.000000e+00 0.000000e+00 1.476308e-04
4.000000e+00 5.000000e+00 0.000000e+00 1.552803e-04
5.000000e+00 5.000000e+00 0.000000e+00 1.587338e-04
1.000000e+01 5.000000e+00 0.000000e+00 1.786143e-04
2.000000e+01 5.000000e+00 0.000000e+00 2.383257e-04
0.000000e+00 6.000000e+00 0.000000e+00 4.200000e-07
1.250000e-01 6.000000e+00 0.000000e+00 1.276547e-05
2.500000e-01 6.000000e+00 0.000000e+00 2.480351e-05
3.750000e-01 6.000000e+00 0.000000e+00 3.653607e-05
5.000000e-01 6.000000e+00 0.000000e+00 4.796425e-05
7.500000e-01 6.000000e+00 0.000000e+00 6.990911e-05
1.000000e+00 6.000000e+00 0.000000e+00 9.063749e-05
1.250000e+00 6.000000e+00 0.000000e+00 1.101443e-04
1.500000e+00 6.000000e+00 0.000000e+00 1.284213e-04
2.000000e+00 6.000000e+00 0.000000e+00 1.612434e-04
2.500000e+00 6.000000e+00 0.000000e+00 1.890030e-04
3.000000e+00 6.000000e+00 0.000000e+00 2.115806e-04
4.000000e+00 6.000000e+00 0.000000e+00 2.406419e-04
5.000000e+00 6.000000e+00 0.000000e+00 2.495963e-04
1.000000e+01 6.000000e+00 0.000000e+00 2.808196e-04
2.000000e+01 6.000000e+00 0.000000e+00 3.745595e-04

Measured Data From a 1.4mu channel device

VDS	VGS	VSB	IDS
0.0	1.0	0.0	1.0e-9
0.25	1.0	0.0	1.01e-9
0.5	1.0	0.0	1.03e-9
0.75	1.0	0.0	1.06e-9
1.0	1.0	0.0	1.10e-9
1.25	1.0	0.0	1.15e-9
1.5	1.0	0.0	1.21e-9
1.75	1.0	0.0	1.28e-9
2.0	1.0	0.0	1.36e-9
2.25	1.0	0.0	1.45e-9
2.5	1.0	0.0	1.55e-9
3.0	1.0	0.0	1.78e-9
3.5	1.0	0.0	2.08e-9
4.0	1.0	0.0	2.48e-9
4.5	1.0	0.0	3.00e-9
5.0	1.0	0.0	3.60e-9
0.0	2.0	0.0	1.0e-9
0.25	2.0	0.0	9.717e-6
0.5	2.0	0.0	15.815e-6
0.75	2.0	0.0	18.734e-6
1.0	2.0	0.0	19.850e-6
1.25	2.0	0.0	20.349e-6
1.5	2.0	0.0	20.655e-6
1.75	2.0	0.0	20.904e-6
2.0	2.0	0.0	21.109e-6
2.25	2.0	0.0	21.305e-6
2.5	2.0	0.0	21.489e-6
3.0	2.0	0.0	21.814e-6
3.5	2.0	0.0	22.135e-6
4.0	2.0	0.0	22.455e-6
4.5	2.0	0.0	22.835e-6
5.0	2.0	0.0	23.355e-6
0.0	3.0	0.0	2.1e-9
0.25	3.0	0.0	17.035e-6
0.5	3.0	0.0	30.040e-6
0.75	3.0	0.0	38.840e-6
1.0	3.0	0.0	44.035e-6
1.25	3.0	0.0	46.760e-6
1.5	3.0	0.0	48.149e-6
1.75	3.0	0.0	48.969e-6
2.0	3.0	0.0	49.524e-6
2.25	3.0	0.0	49.969e-6
2.5	3.0	0.0	50.355e-6
3.0	3.0	0.0	51.008e-6
3.5	3.0	0.0	51.595e-6
4.0	3.0	0.0	52.145e-6
4.5	3.0	0.0	52.735e-6
5.0	3.0	0.0	53.499e-6
0.0	5.0	0.0	4.6e-9
0.25	5.0	0.0	29.225e-6

0.5	5.0	0.0	54.294e-6
0.75	5.0	0.0	74.469e-6
1.0	5.0	0.0	89.735e-6
1.25	5.0	0.0	100.630e-6
1.5	5.0	0.0	108.150e-6
1.75	5.0	0.0	113.000e-6
2.0	5.0	0.0	116.100e-6
2.25	5.0	0.0	118.200e-6
2.5	5.0	0.0	119.600e-6
3.0	5.0	0.0	121.550e-6
3.5	5.0	0.0	122.900e-6
4.0	5.0	0.0	124.050e-6
4.5	5.0	0.0	125.050e-6
5.0	5.0	0.0	126.100e-6

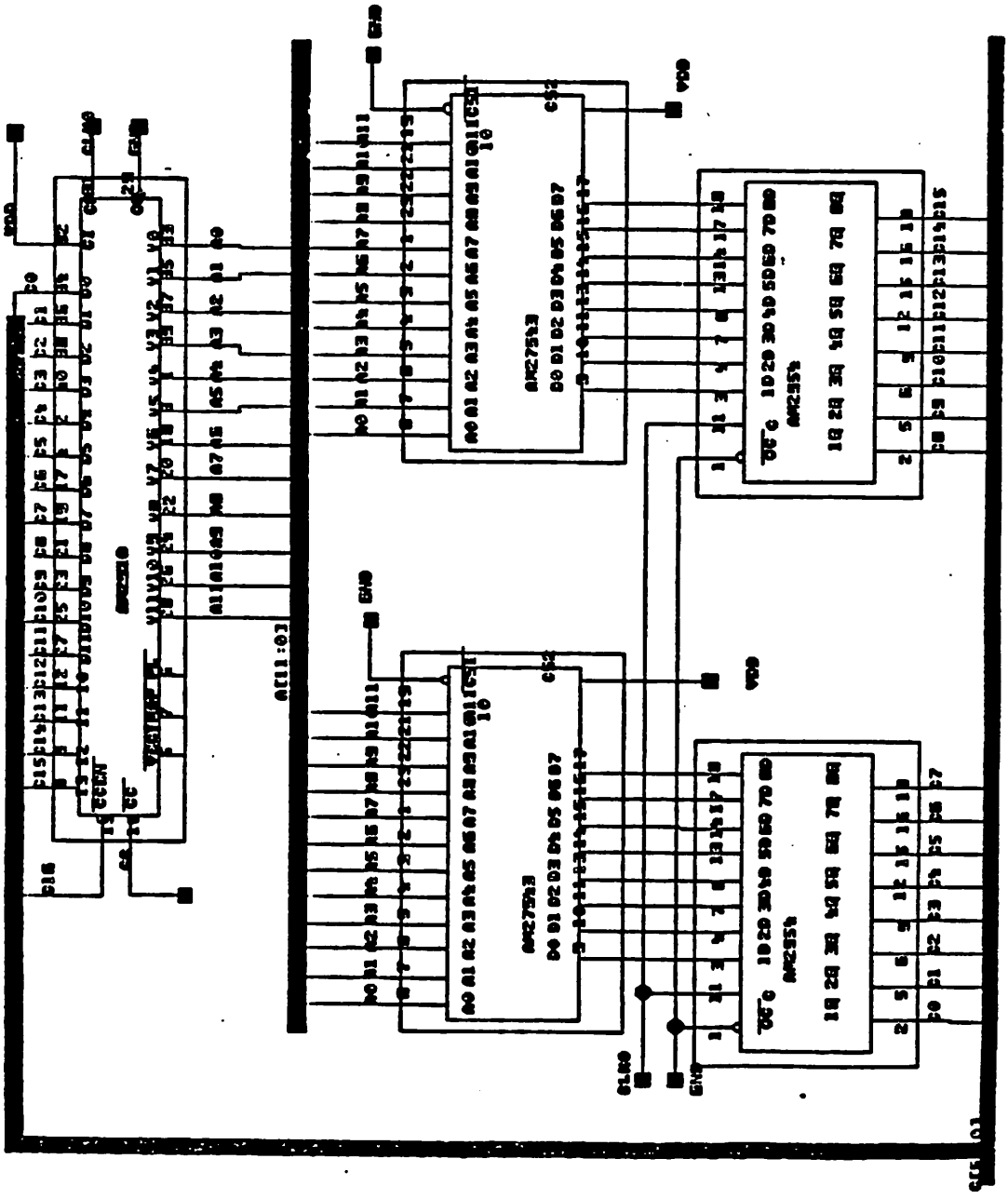
APPENDIX G

Prototype MMAP Schematics and Parts List

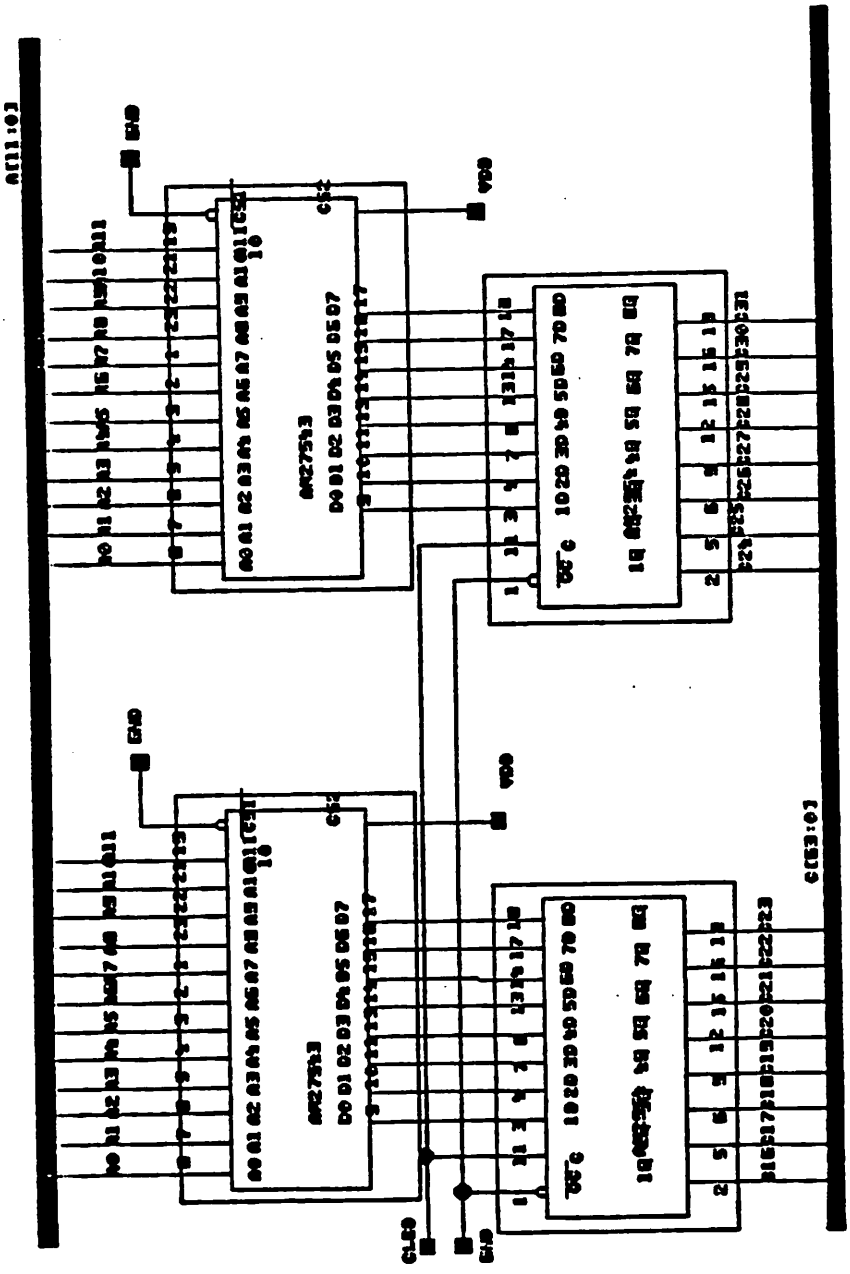
Package #	Part #	Description
U1	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U2	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U3	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U4	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U5	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U6	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U7	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U8	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U9	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U10	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U11	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U12	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U13	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U14	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U15	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U16	Intel 2164A	65,636 × 1 Bit Dynamic RAM
U17	Intel 8207	Dual-Port Memory Controller
U18	Weitek 1033	32-Bit ALU
U19	Weitek 1033	32-Bit Multiplier
U20	AMD 2910	Micro-Controller
U21	74LS173	Quad Register w/clear, clk enable & tri-state output
U22	74LS173	Quad Register w/clear, clk enable & tri-state output
U23	74LS173	Quad Register w/clear, clk enable & tri-state output
U24	74LS173	Quad Register w/clear, clk enable & tri-state output
U25	74LS173	Quad Register w/clear, clk enable & tri-state output
U26	74LS173	Quad Register w/clear, clk enable & tri-state output
U27	74LS173	Quad Register w/clear, clk enable & tri-state output
U28	74LS173	Quad Register w/clear, clk enable & tri-state output
U29	74LS173	Quad Register w/clear, clk enable & tri-state output
U30	AMD 29705A	16 × 4 Dual-Port Register
U31	AMD 29705A	16 × 4 Dual-Port Register
U32	AMD 29705A	16 × 4 Dual-Port Register
U33	AMD 29705A	16 × 4 Dual-Port Register
U34	AMD 91122	256 × 4 Static Memory
U35	AMD 91122	256 × 4 Static Memory
U36	AMD 91122	256 × 4 Static Memory
U37	AMD 91122	256 × 4 Static Memory
U38	EPROM	4K × 8 EPROM
U39	EPROM	4K × 8 EPROM
U40	EPROM	4K × 8 EPROM
U41	EPROM	4K × 8 EPROM
U42	EPROM	4K × 8 EPROM
U43	EPROM	4K × 8 EPROM
U44	EPROM	4K × 8 EPROM
U45	EPROM	4K × 8 EPROM
U46	AMD 2954	Octal Register
U47	AMD 2954	Octal Register
U48	AMD 2954	Octal Register
U49	AMD 2954	Octal Register
U50	AMD 2954	Octal Register

Package #	Part #	Description
U51	AMD 2954	Octal Register
U52	AMD 2954	Octal Register
U53	AMD 2954	Octal Register
U54	74ls00	Quad Two-Input Nand
U55	74ls00	Quad Two-Input Nand
U56	74ls00	Quad Two-Input Nand
U57	74ls00	Quad Two-Input Nand
U58	74ls00	Quad Two-Input Nand
U59	74ls00	Quad Two-Input Nand
U60	74ls00	Quad Two-Input Nand
U61	74ls00	Quad Two-Input Nand
U62	74ls00	Quad Two-Input Nand
U63	74ls00	Quad Two-Input Nand
U64	74ls00	Quad Two-Input Nand
U65	74ls01	Open-Collector Quad Two-Input Nand
U66	74ls04	Hex Inverters
U67	74ls04	Hex Inverters
U68	74ls04	Hex Inverters
U69	74ls04	Hex Inverters
U70	74ls04	Hex Inverters
U71	74ls04	Hex Inverters
U72	74ls10	Triple Three-Input Nand
U73	74ls10	Triple Three-Input Nand
U74	74ls10	Triple Three-Input Nand
U75	74ls10	Triple Three-Input Nand
U76	74ls10	Triple Three-Input Nand
U77	74ls20	Dual Four-Input Nand
U78	74ls30	Single Eight-Input Nand
U79	74ls32	Quad Two-Input Nor
U80	74ls74	Dual Clocked D Flip-Flops
U81	74ls133	Single Thirteen-Input Nand
U82	74ls151	8-to-1 Demultiplexer
U83	74ls157	Quad 2-to-1 Data Select
U84	74ls165	Parallel-Load 8-bit Shift Register
U85	74ls244	Octal Non-Inverting Buffer
U86	74ls244	Octal Non-Inverting Buffer
U87	74ls244	Octal Non-Inverting Buffer
U88	74ls30	Single Eight-Input Nand
U89	74ls165	Parallel-Load 8-Bit Shift Register
U90	74ls165	Parallel-Load 8-Bit Shift Register
U91	74ls31	Delay Block
U92	74ls173	Quad Register
U93	74ls173	Quad Register
U94	74ls173	Quad Register
U95	74ls173	Quad Register
U96	74ls173	Quad Register
U97	74s287	256 × 4 ROM
U98	74s287	256 × 4 ROM
U99	74ls245	Octal Bus Tranceiver
U100	74ls373	Octal D-Type Latches
U101	74ls373	Octal D-Type Latches

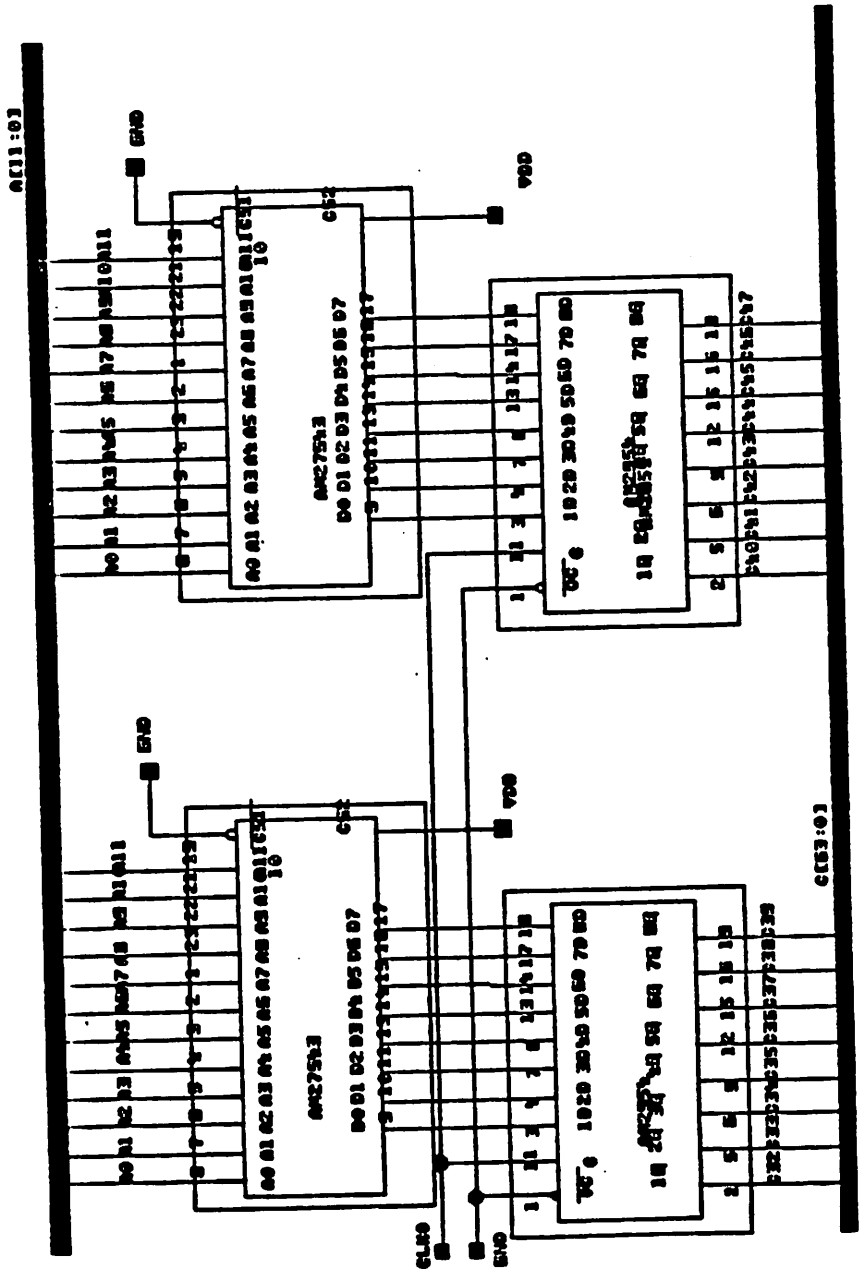
Schematic: 1/34 Controller: 1/5



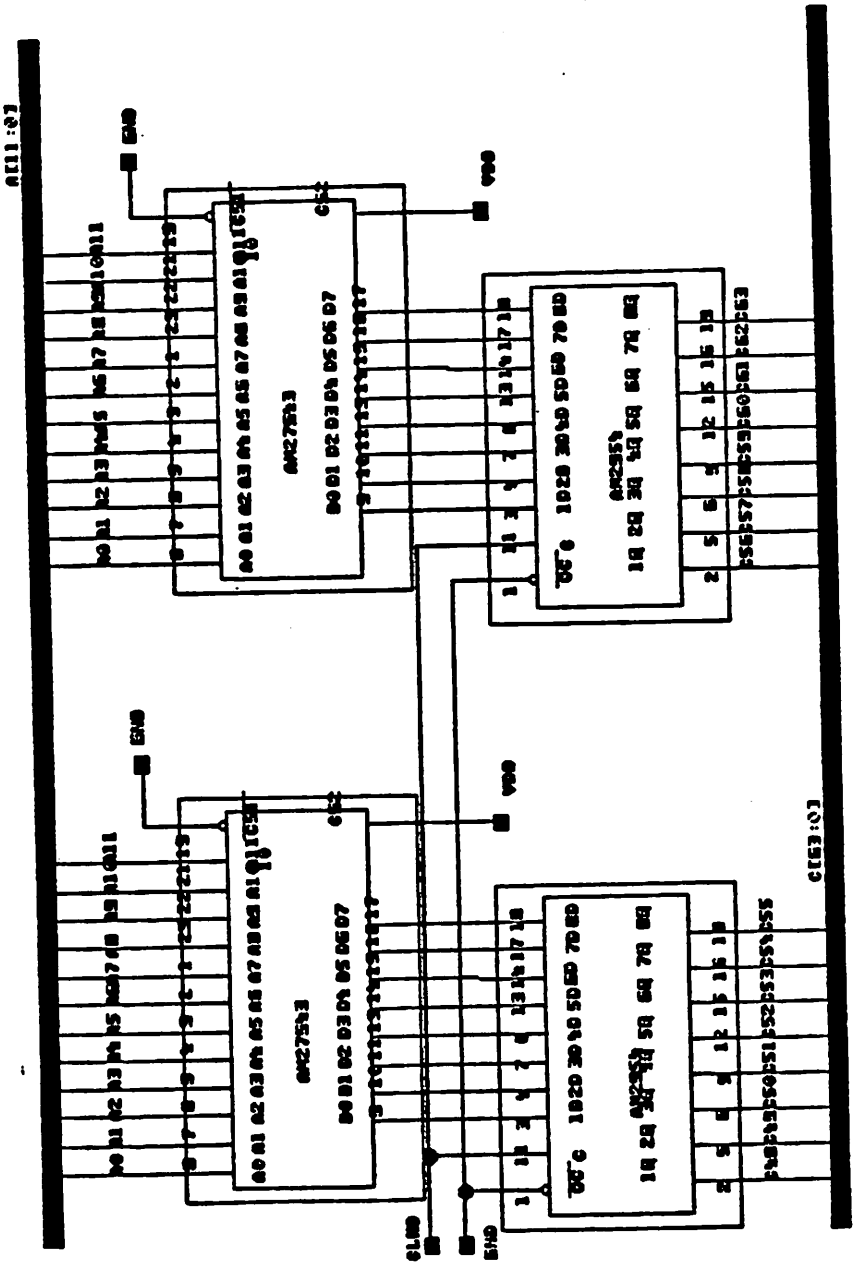
Schematic: 2/34 Controller: 2/5



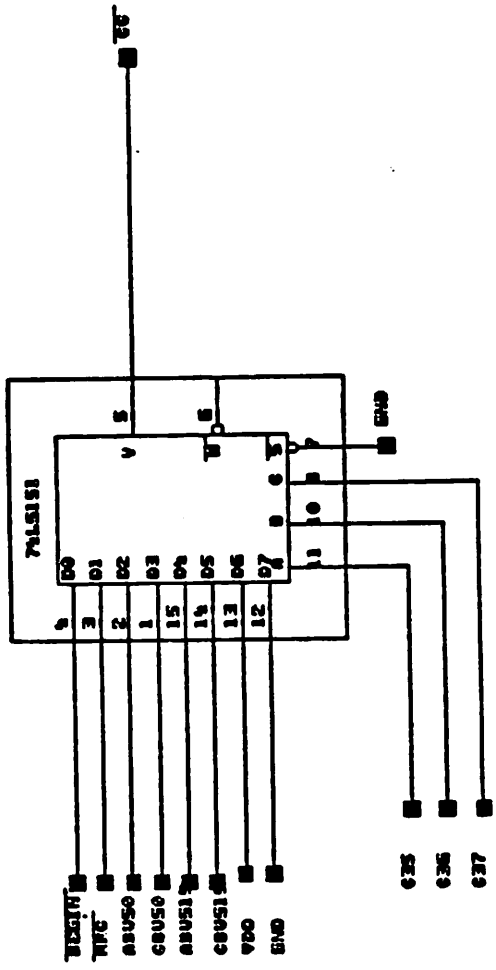
Schematic: 3/34 Controller: 3/5



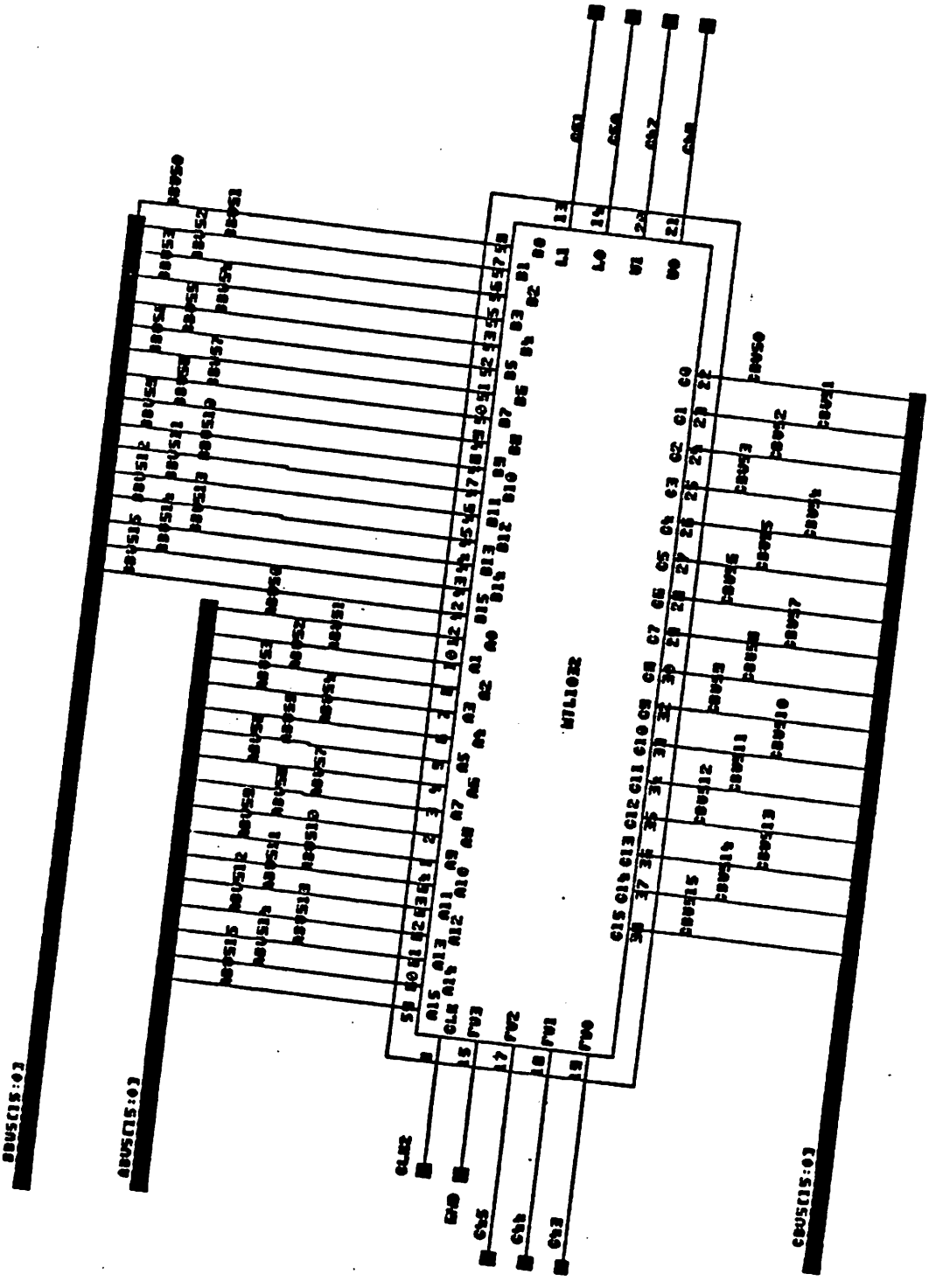
Schematic: 4/34 Controller: 4/5



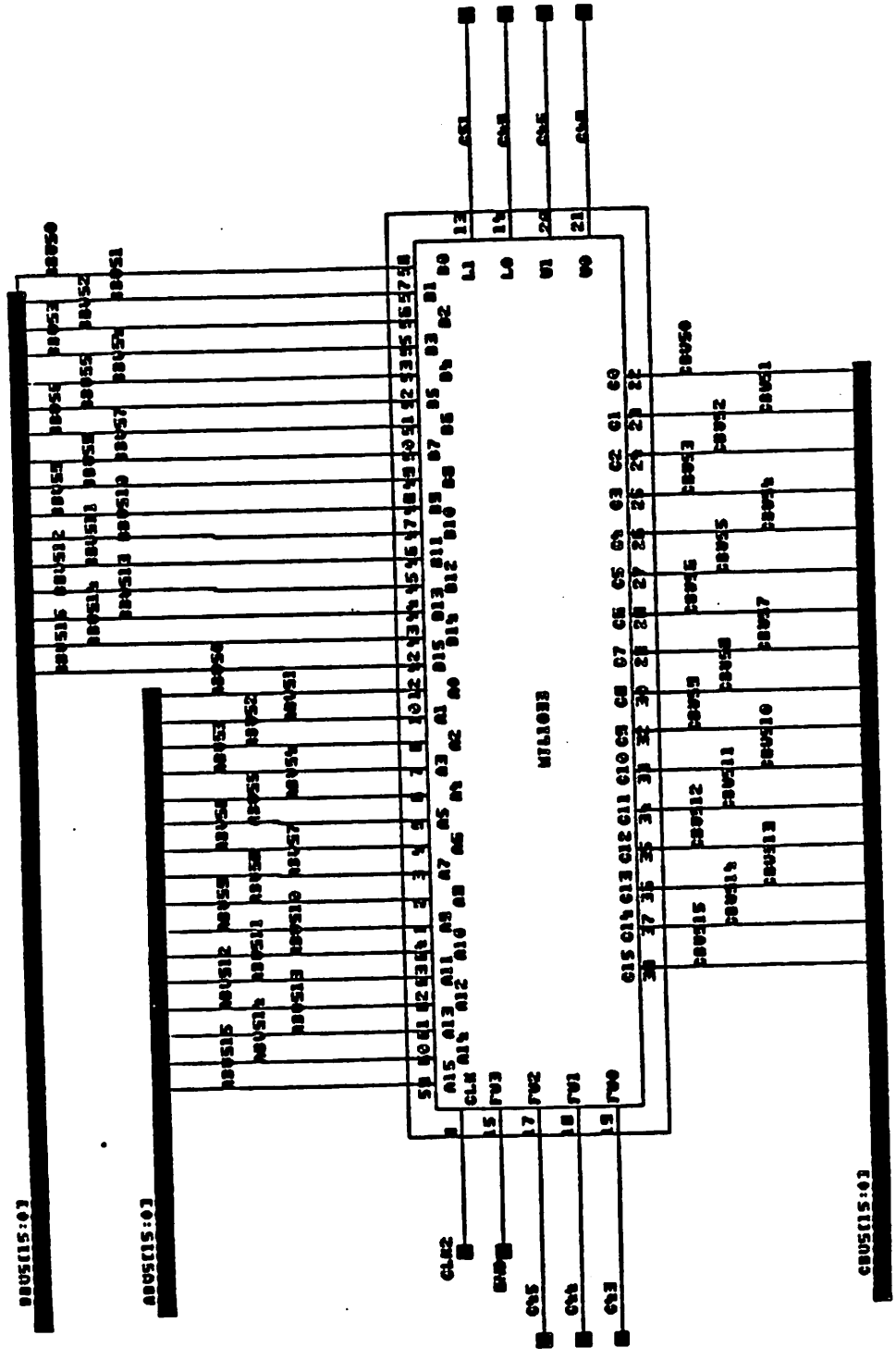
Schematic: 5/34 Controller: 5/5



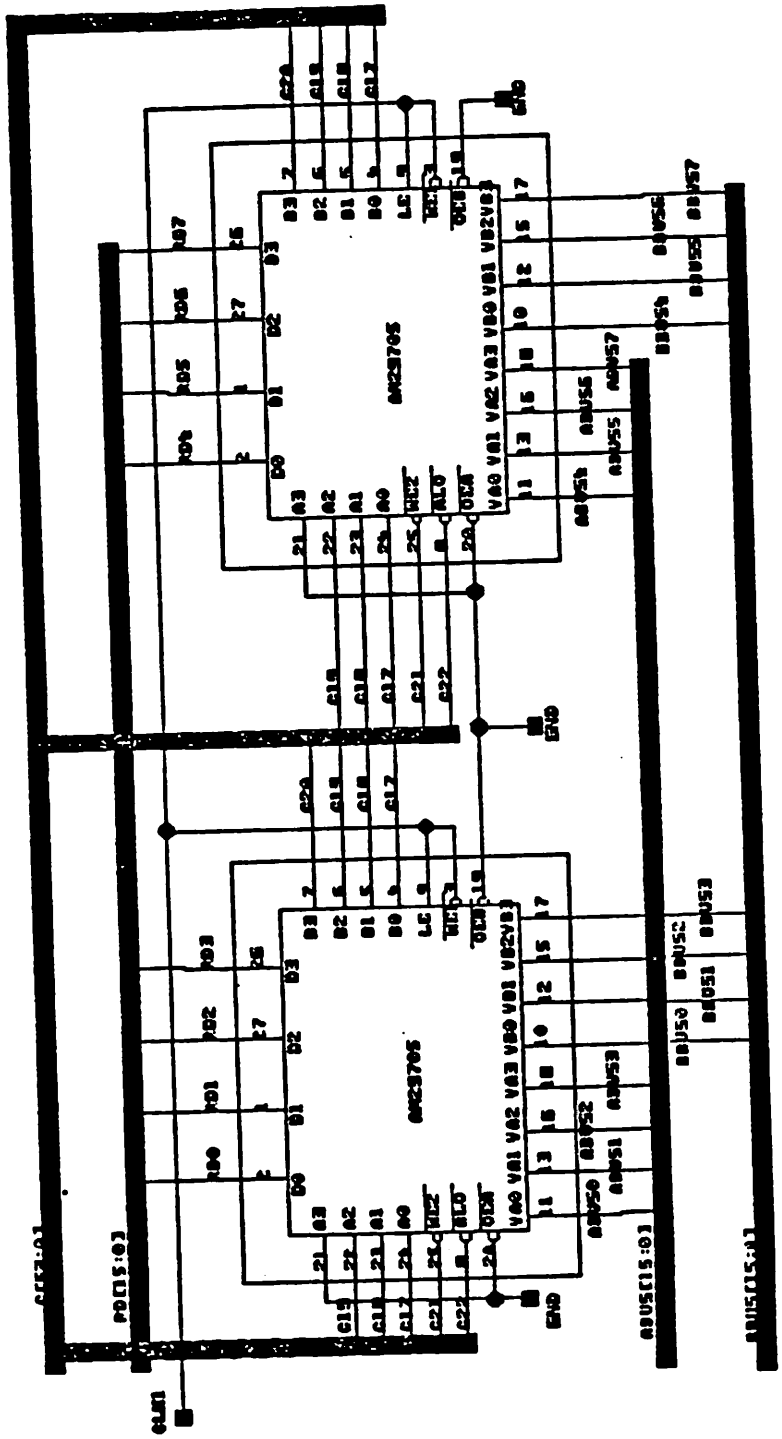
Schematic: 6/34 FPU: 1/2



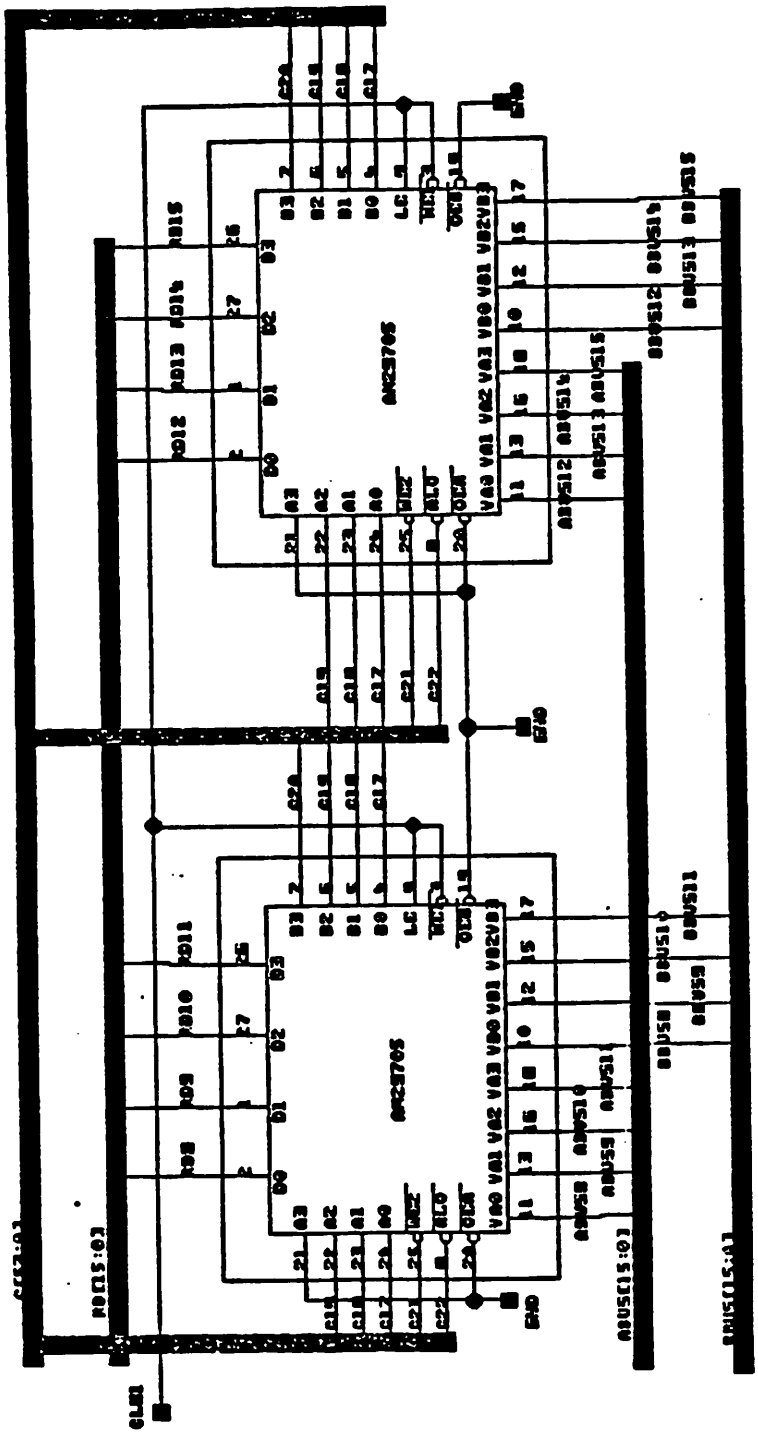
Schematic: 7/34 FPU: 2/2



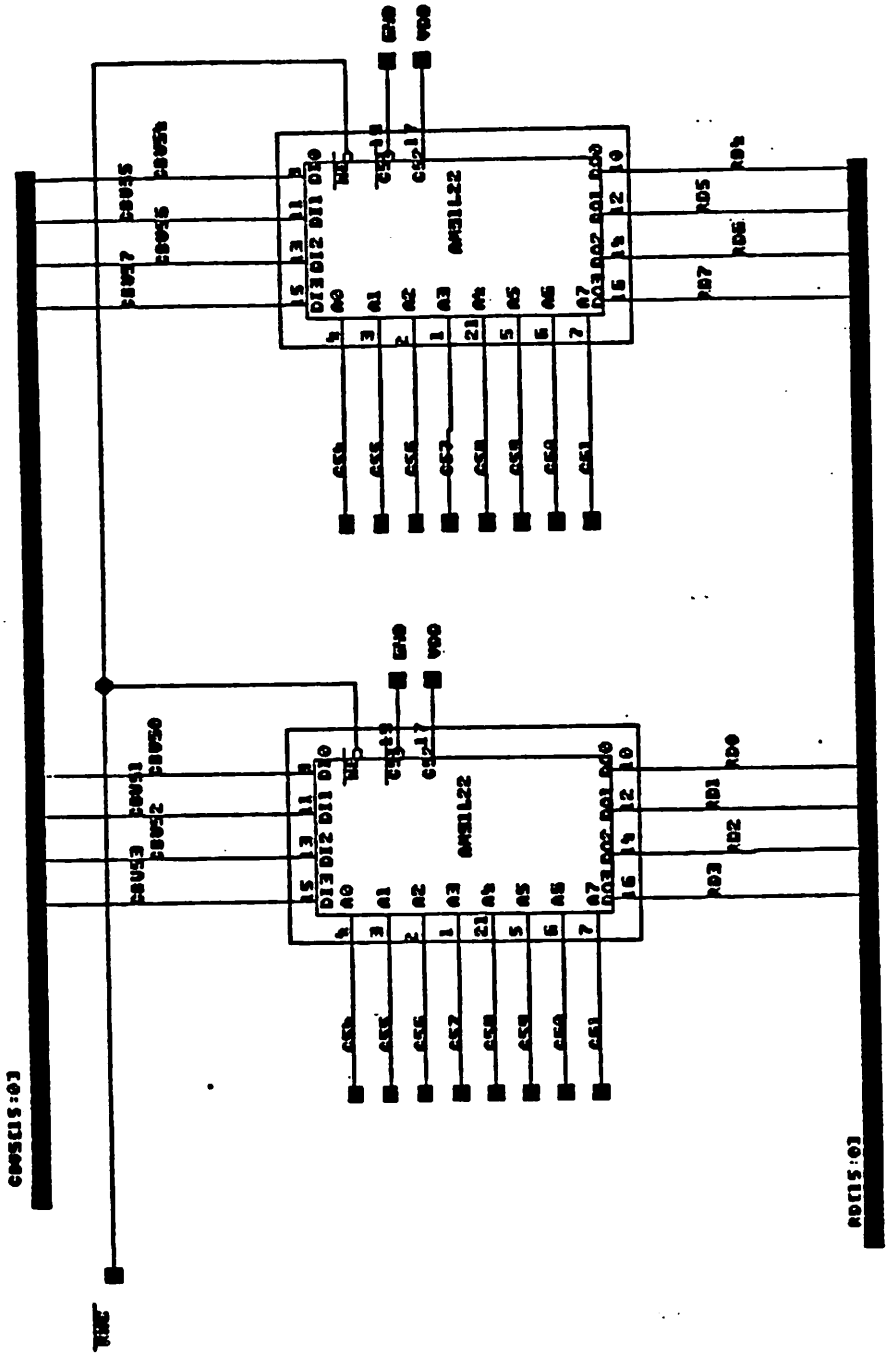
Schematic: 8/34 Coefficient Cache: 1/5



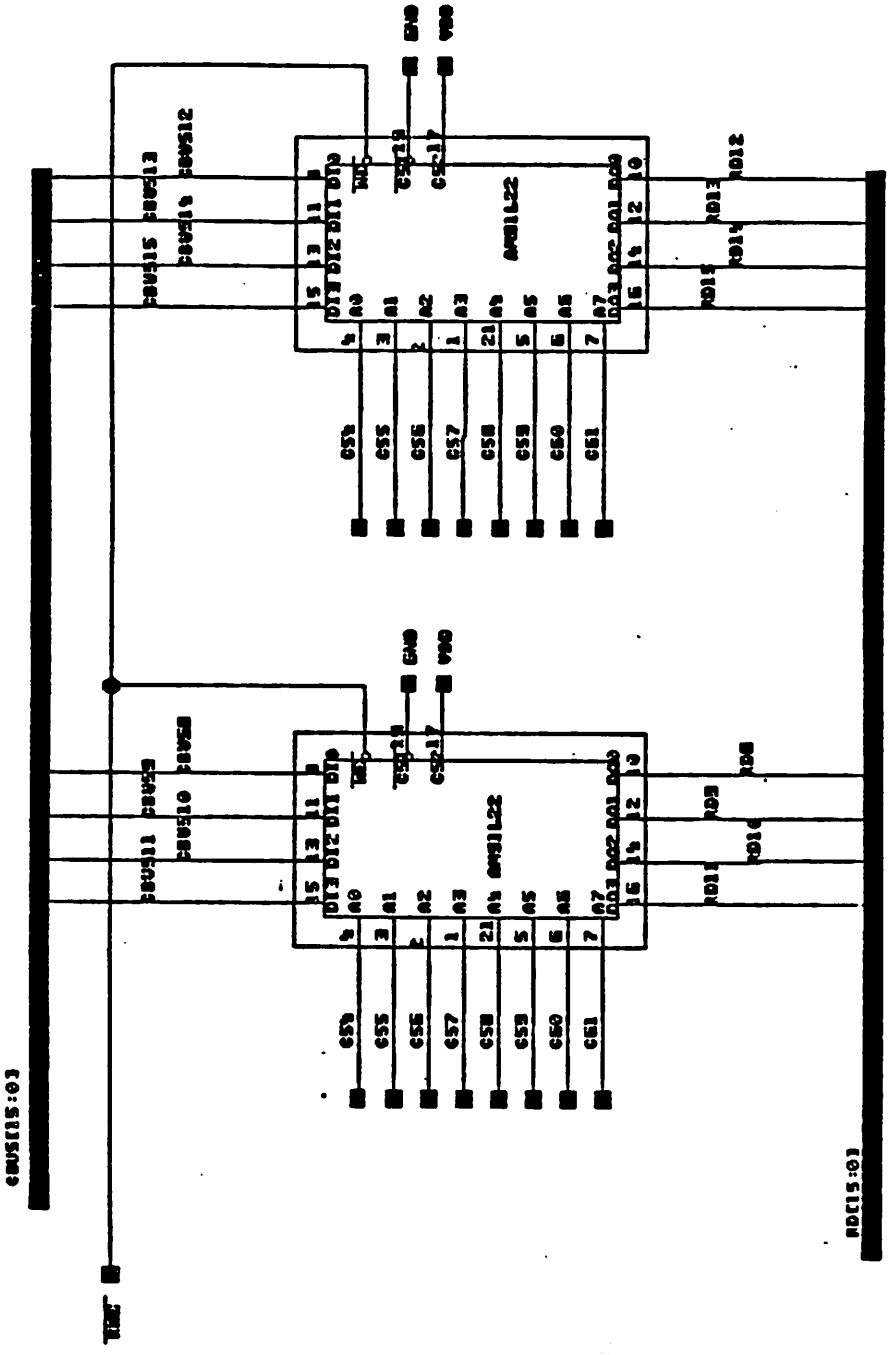
Schematic: 9/34 Coefficient Cache: 2/5



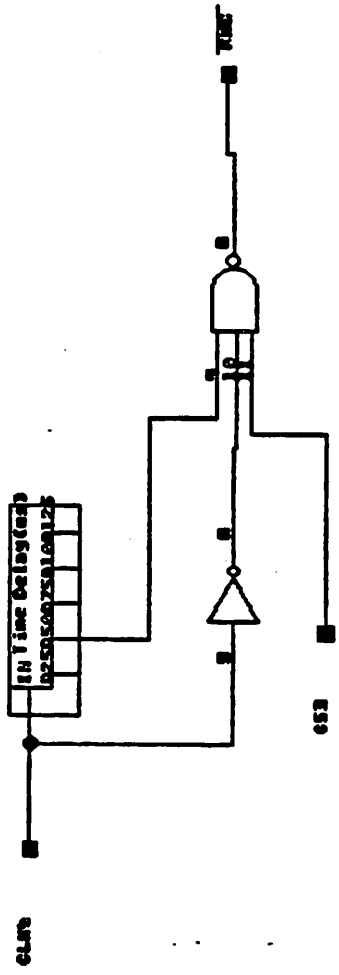
Schematic: 10/34 Coefficient Cache: 3/5



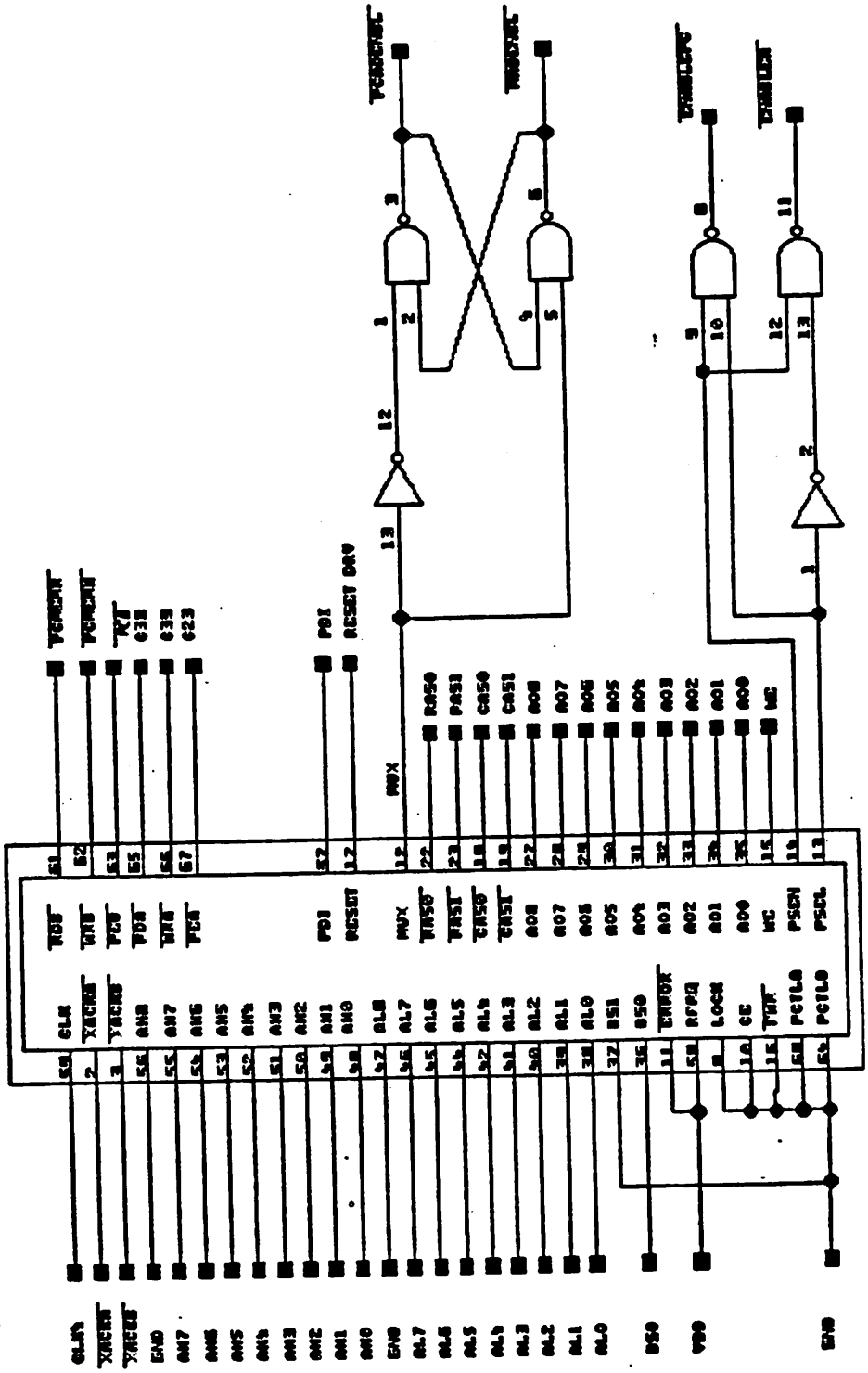
Schematic: 11/34 Coefficient Cache: 4/5



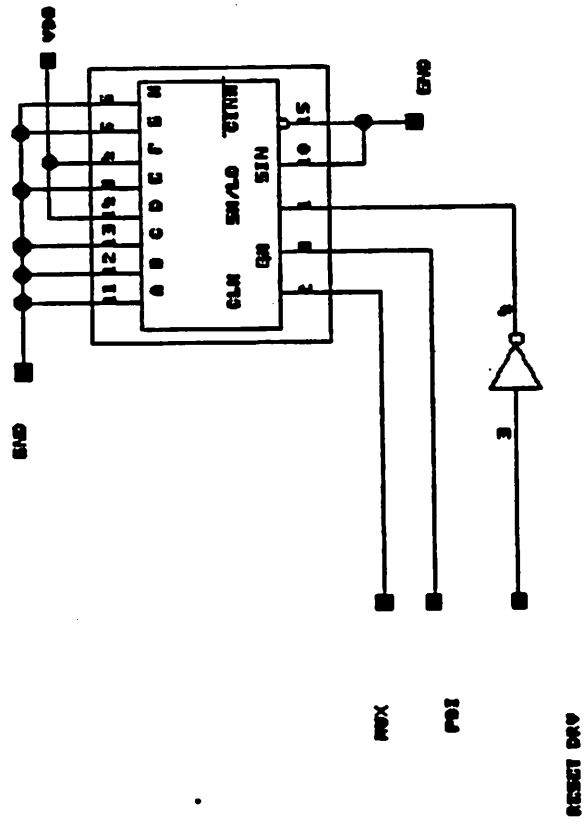
Schematic: 12/34 Coefficient Cache: 5/5



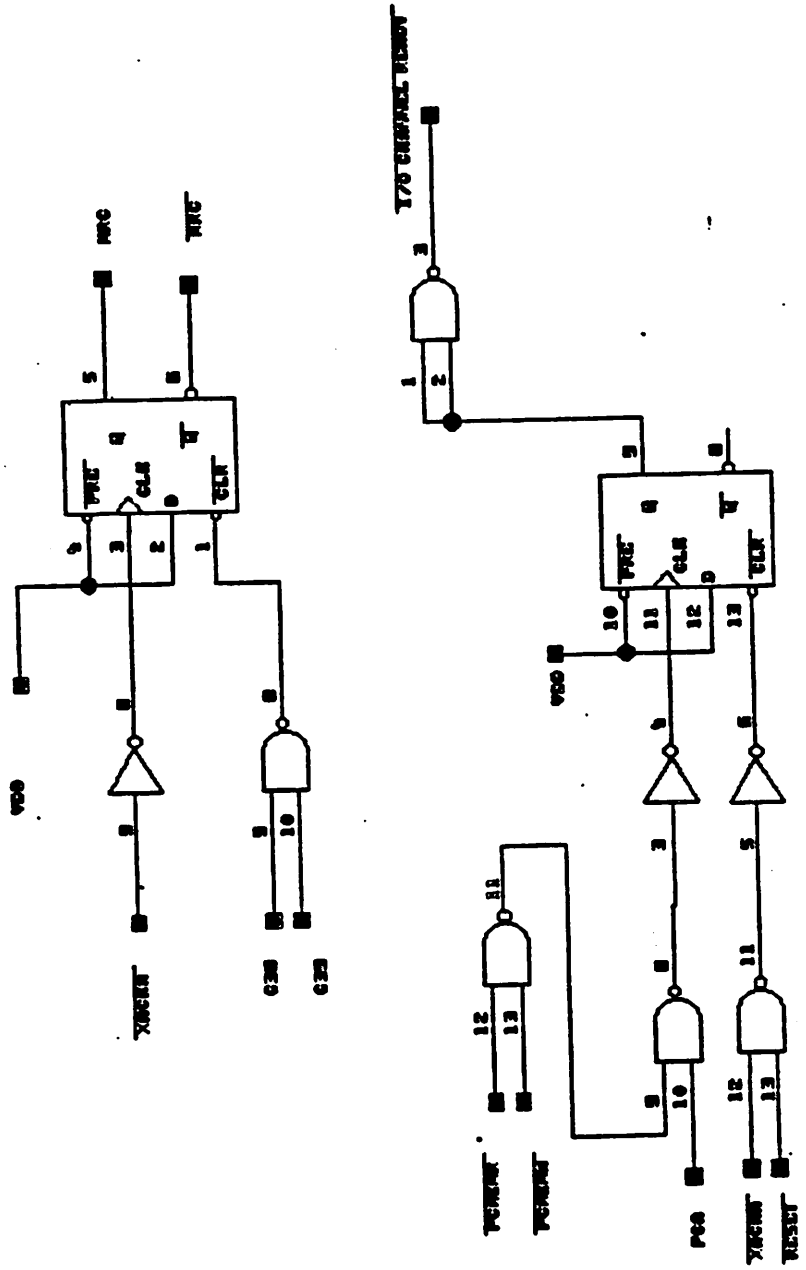
Schematic: 13/34 Coefficient Memory and Interface: 1/22



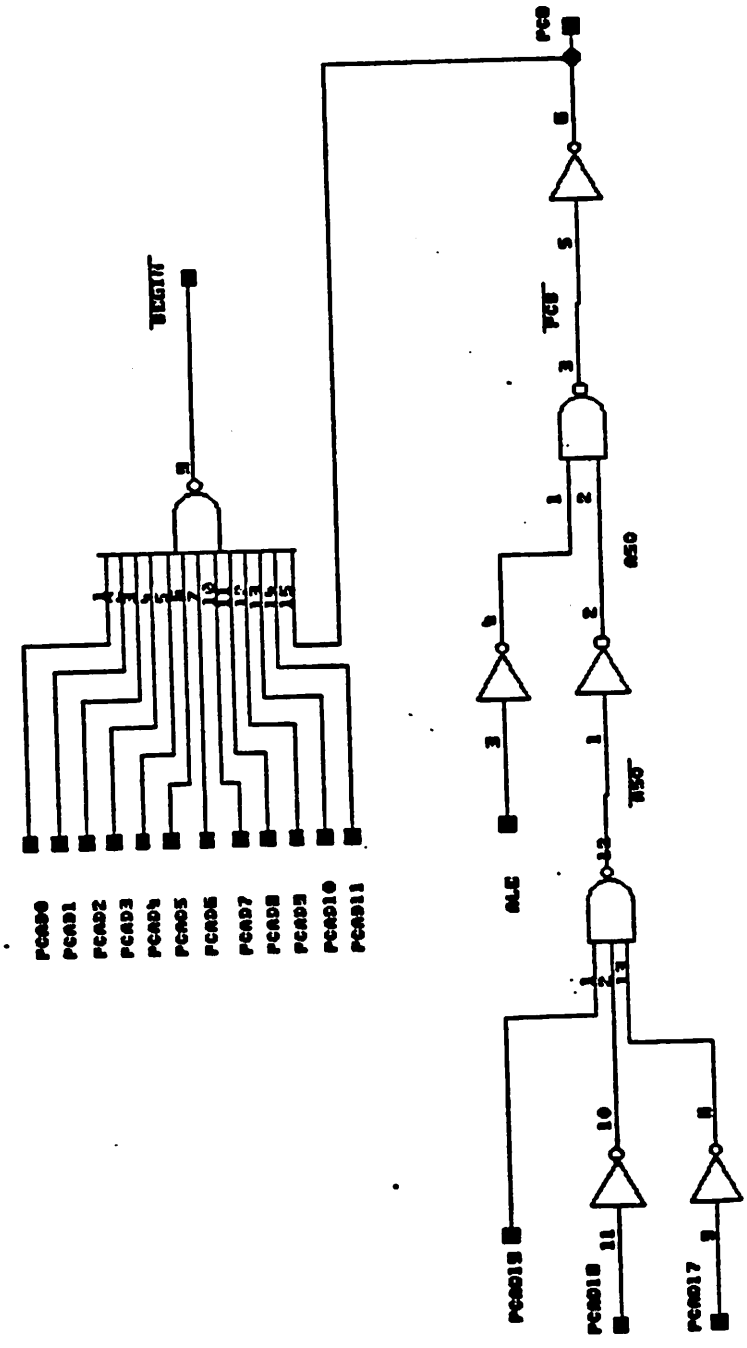
Schematic: 14/34 Coefficient Memory and Interface: 2/22



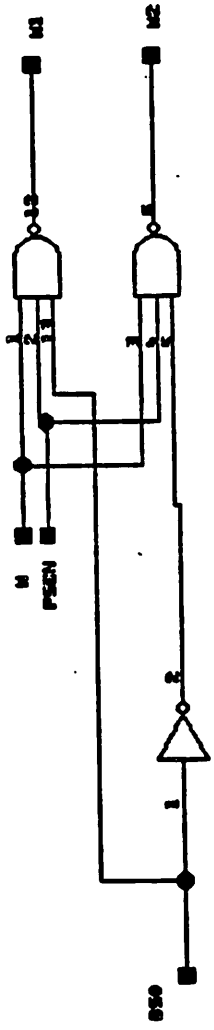
Schematic: 15/34 Coefficient Memory and Interface: 3/22



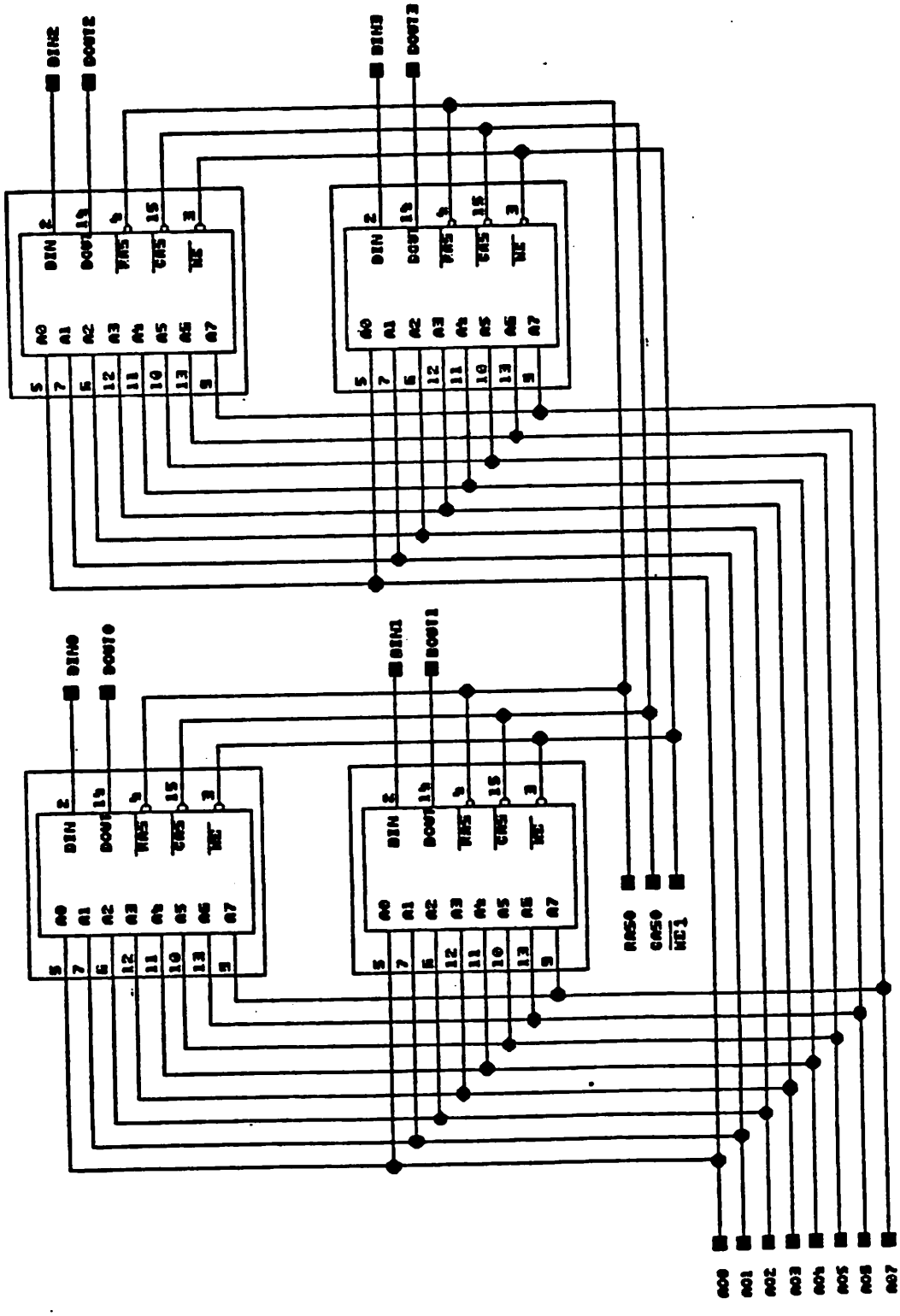
Schematic: 16/34 Coefficient Memory and Interface: 4/22



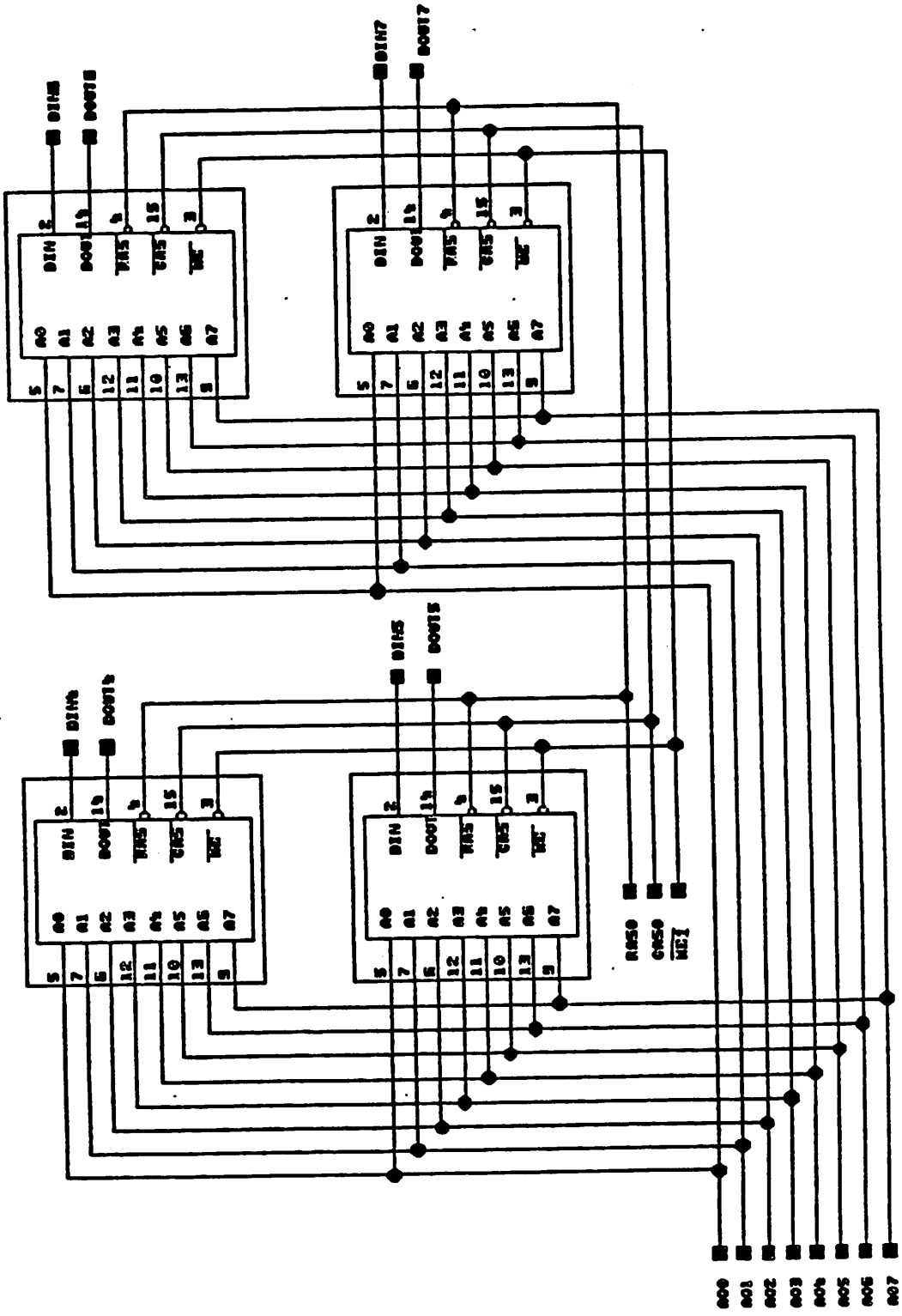
Schematic: 17/34 Coefficient Memory and Interface: 5/22



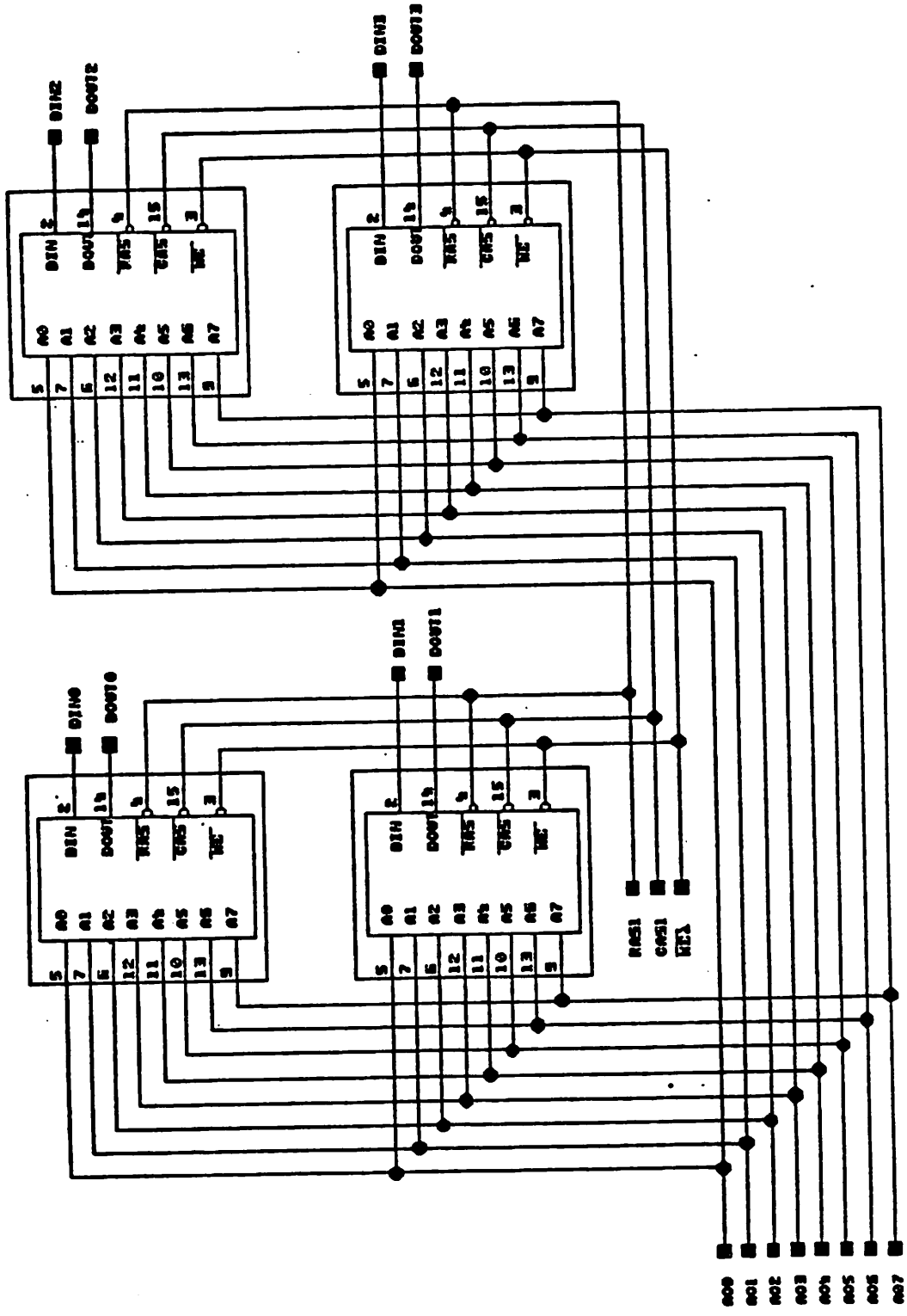
Schematic: 18/34 Coefficient Memory and Interface: 6/22



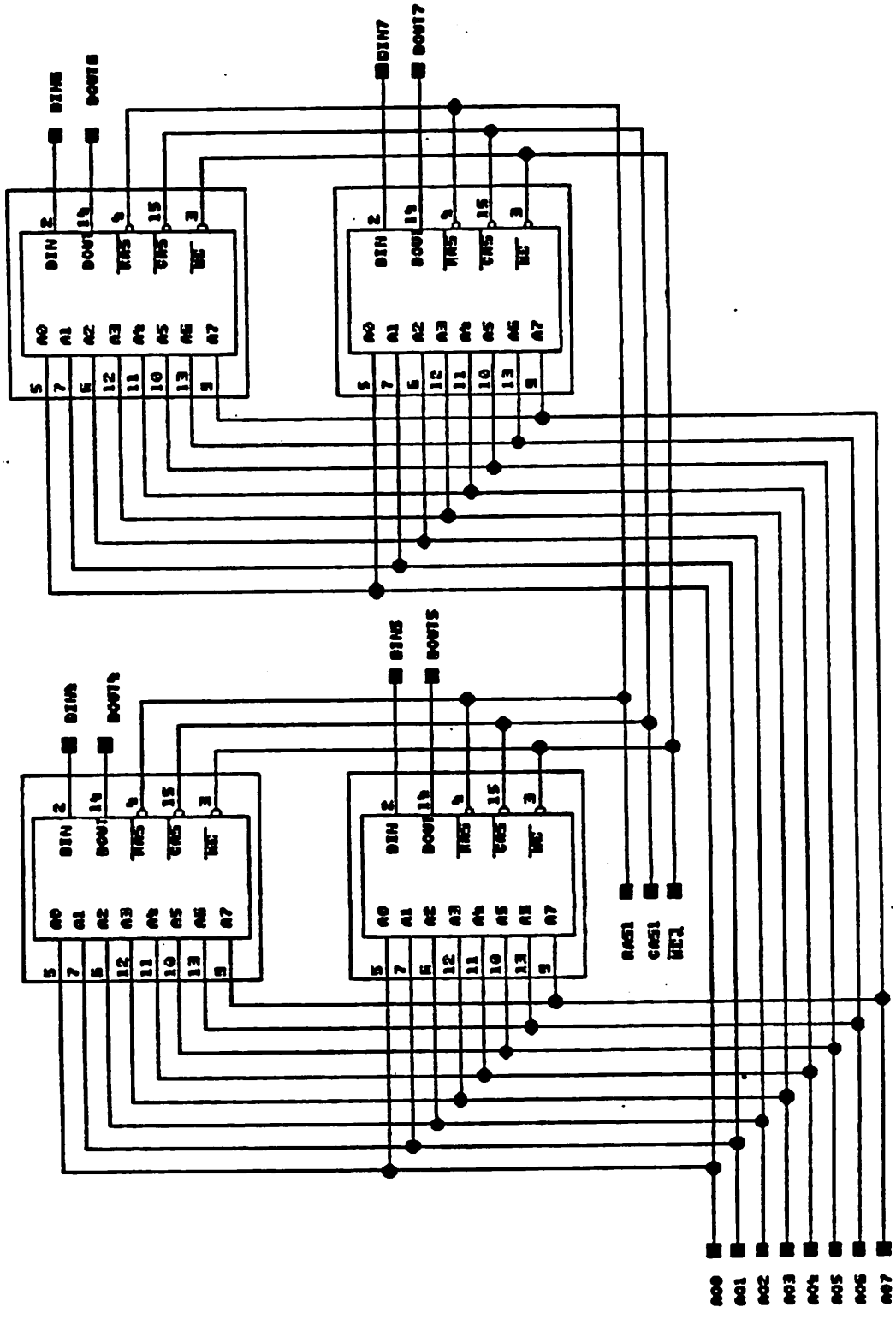
Schematic: 19/34 Coefficient Memory and Interface: 7/22



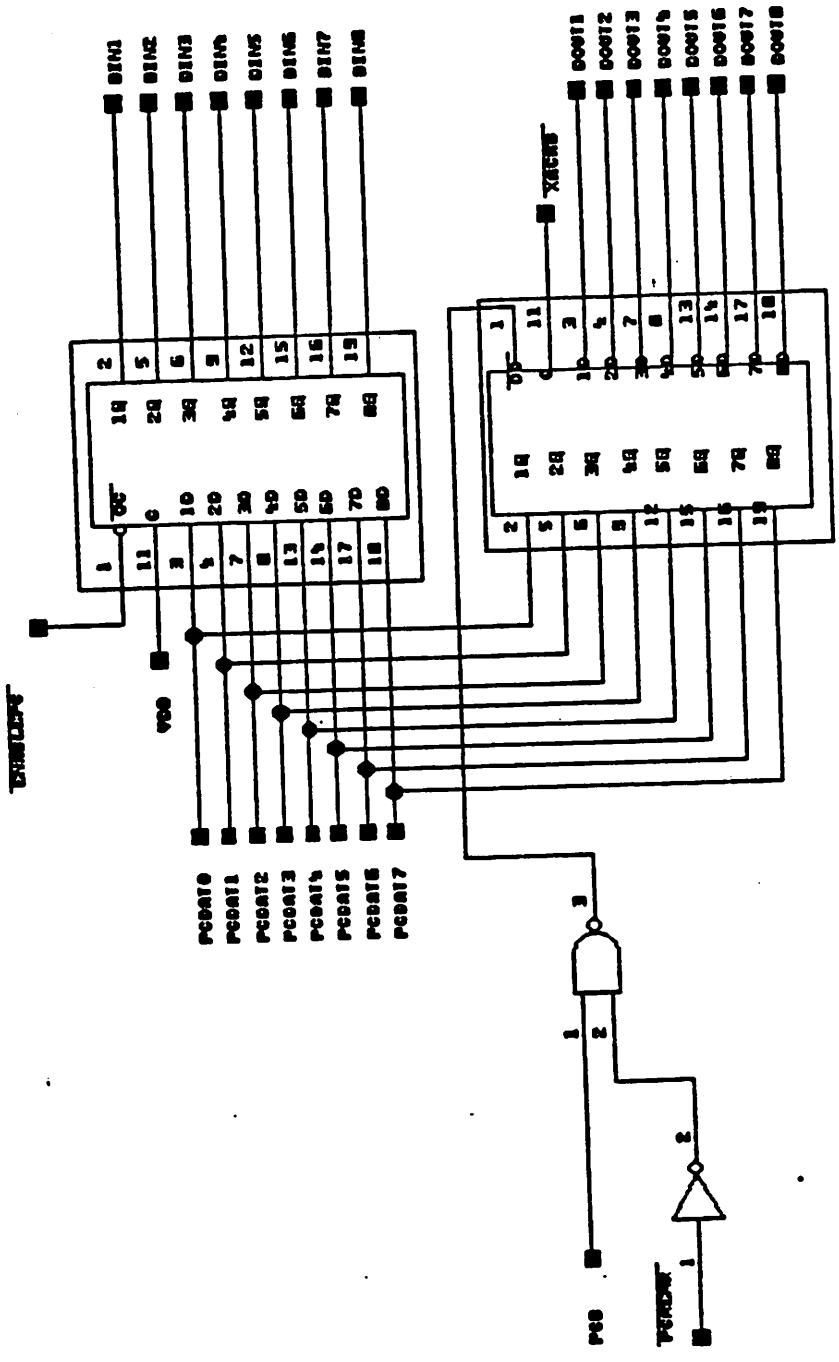
Schematic: 20/34 Coefficient Memory and Interface: 8/22



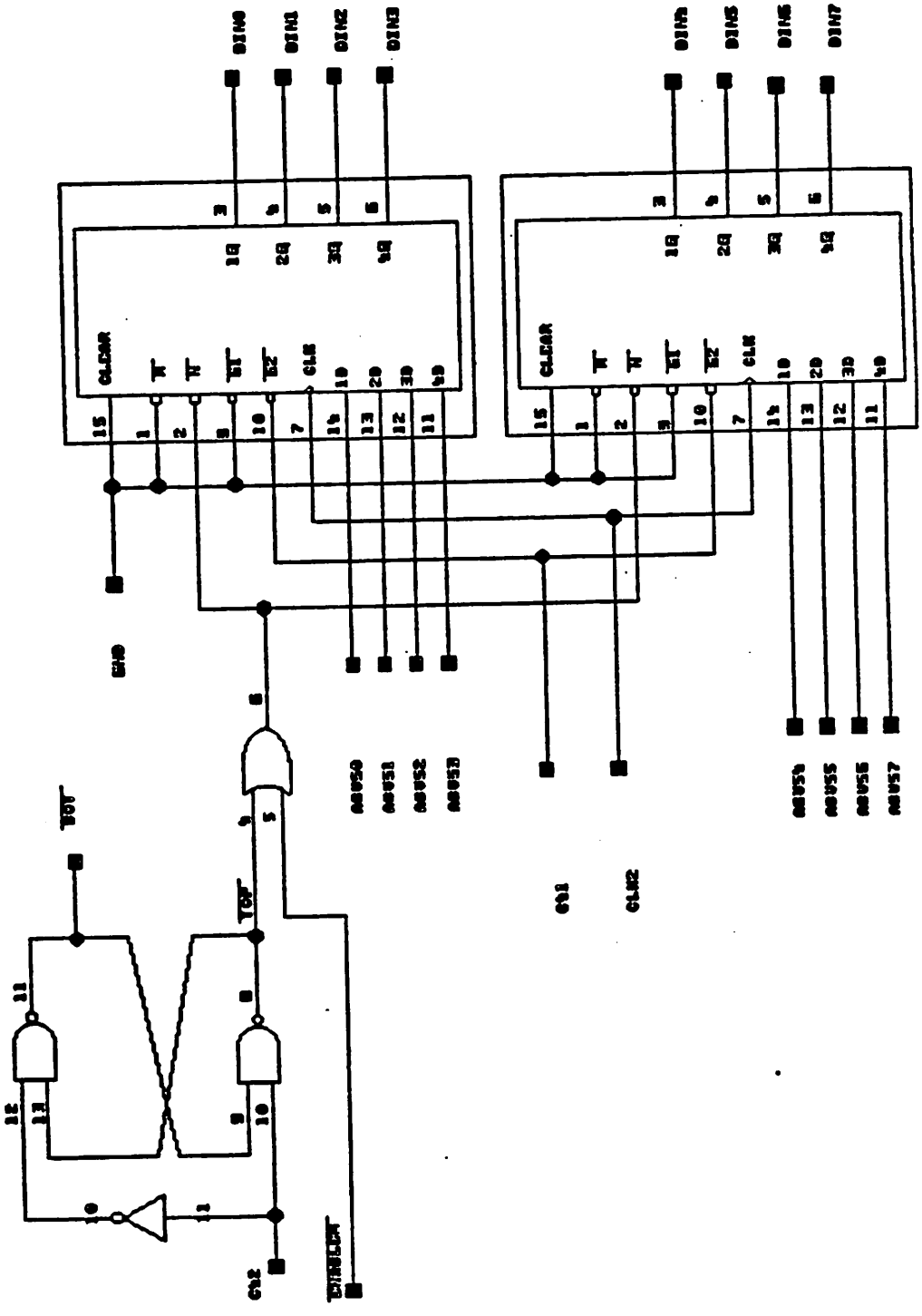
Schematic: 21/34 Coefficient Memory and Interface: 9/22



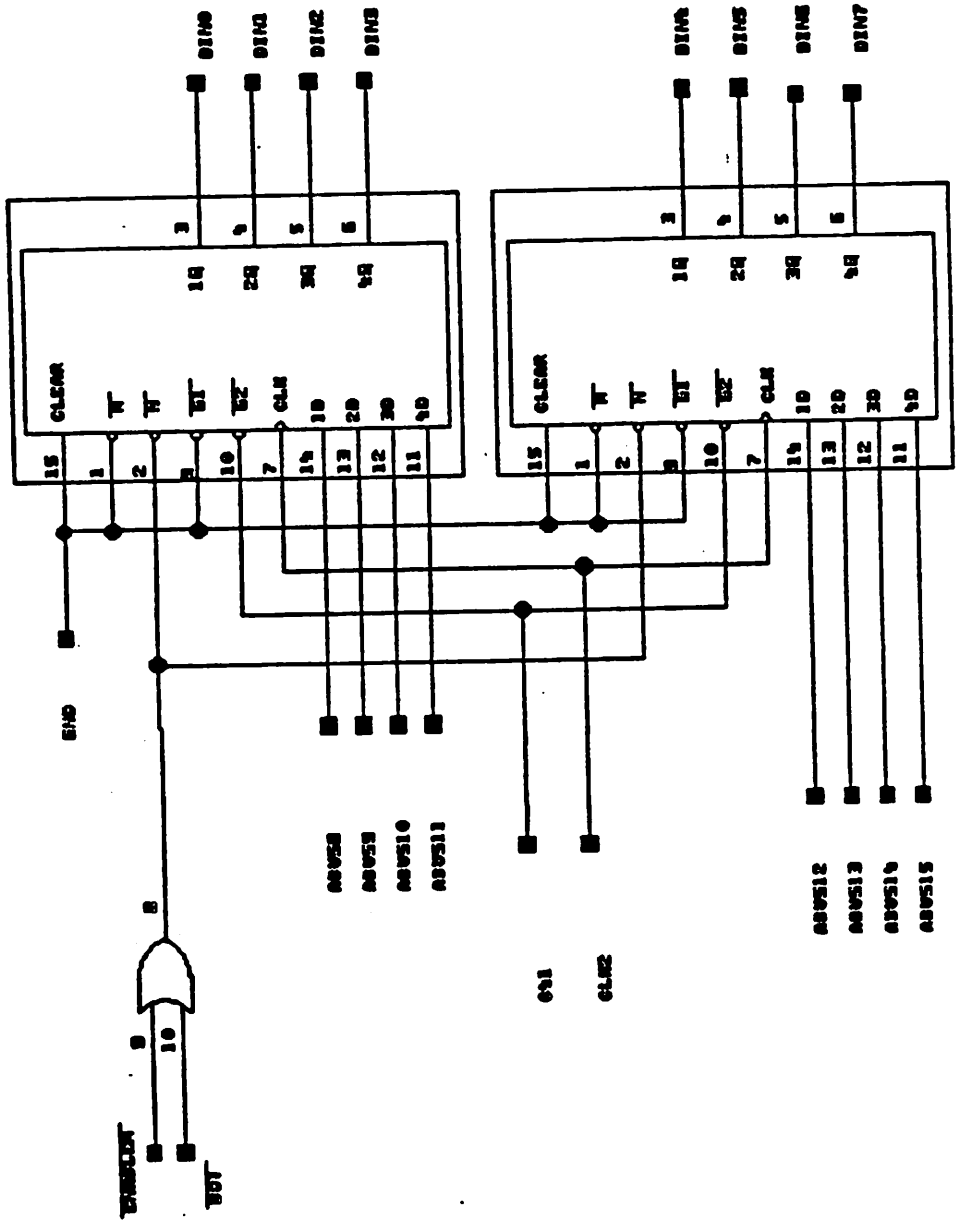
Schematic: 22/34 Coefficient Memory and Interface: 10/22



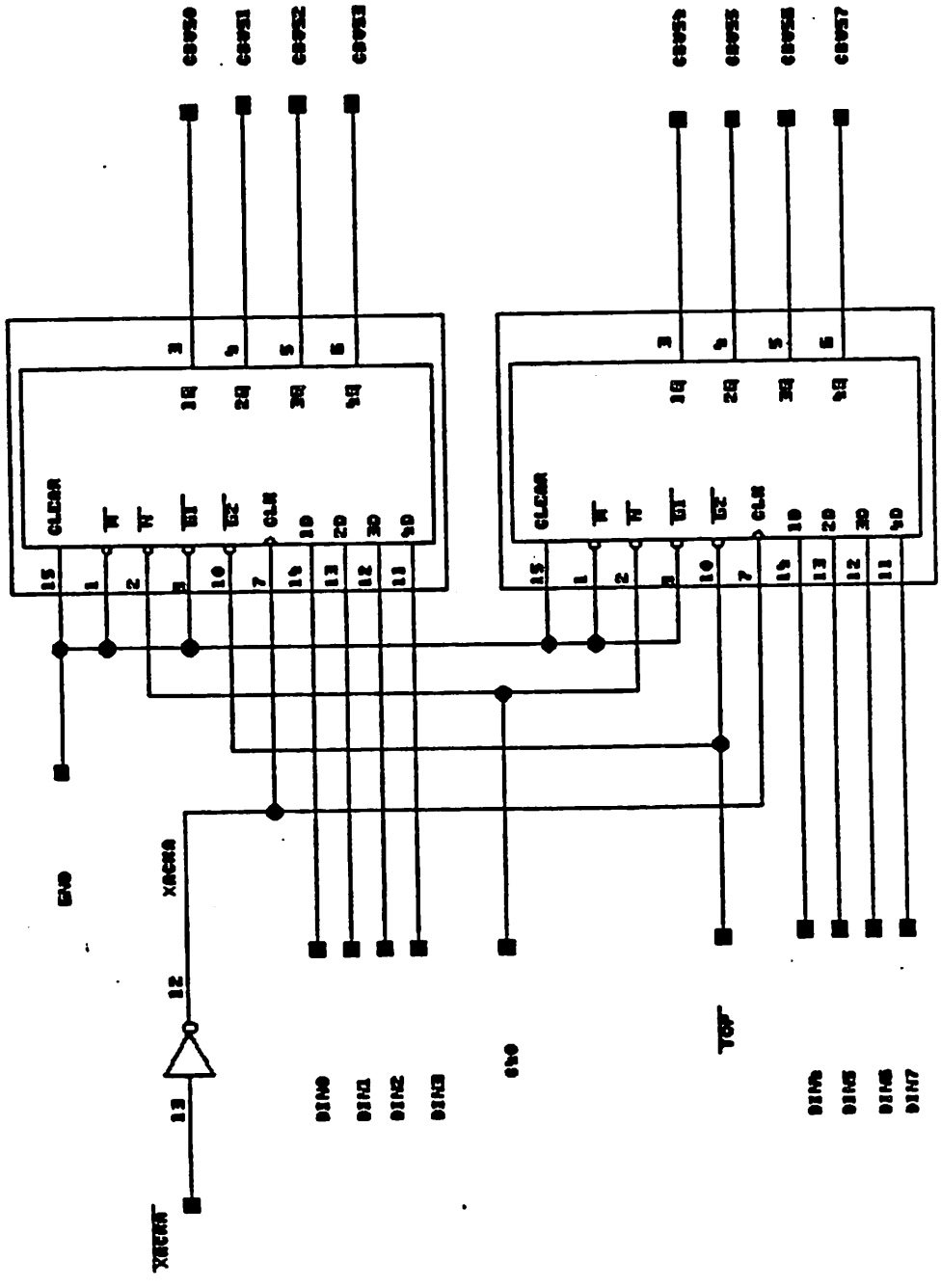
Schematic: 23/34 Coefficient Memory and Interface: 11/22



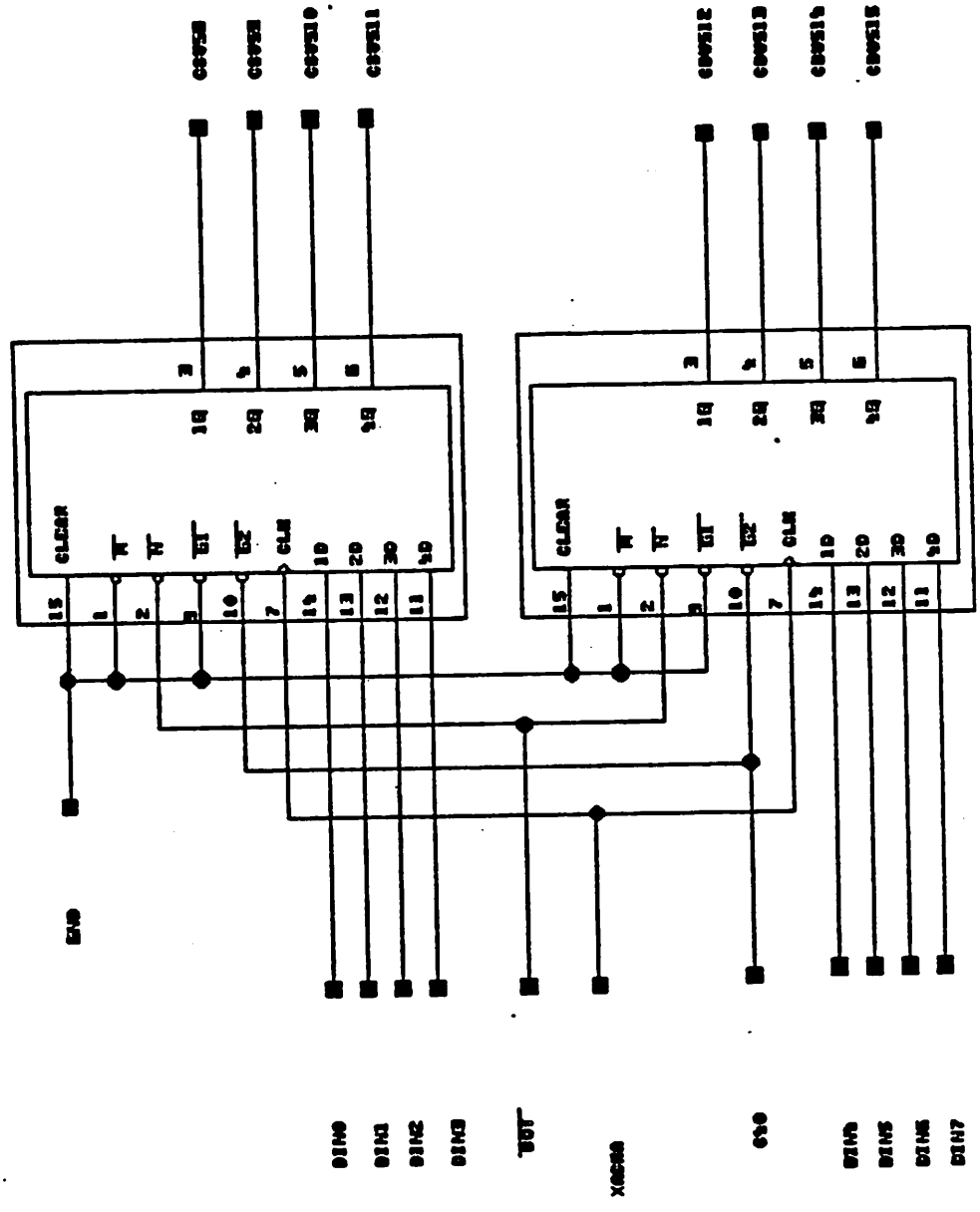
Schematic: 24/34 Coefficient Memory and Interface: 12/22



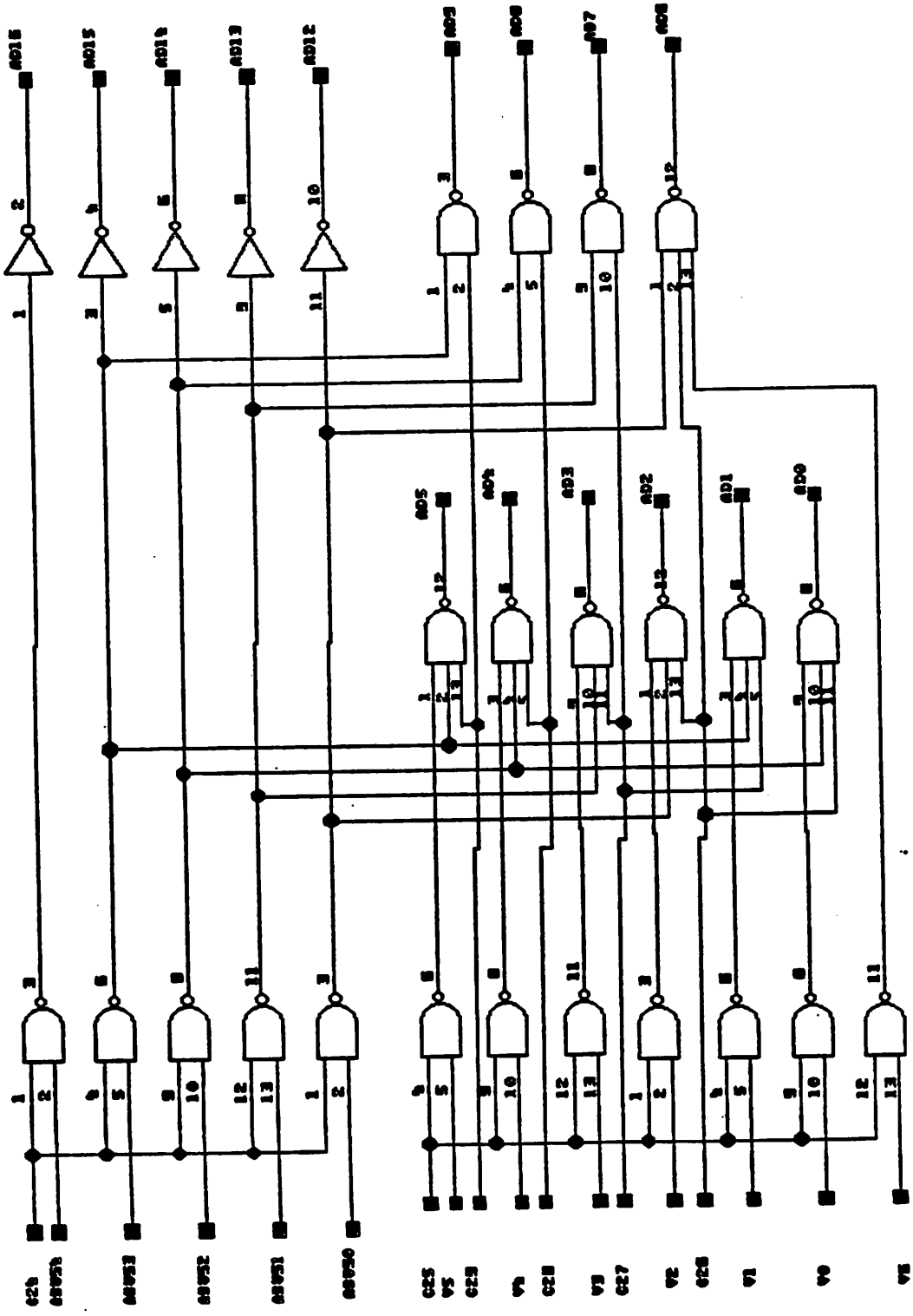
Schematic: 25/34 Coefficient Memory and Interface: 13/22



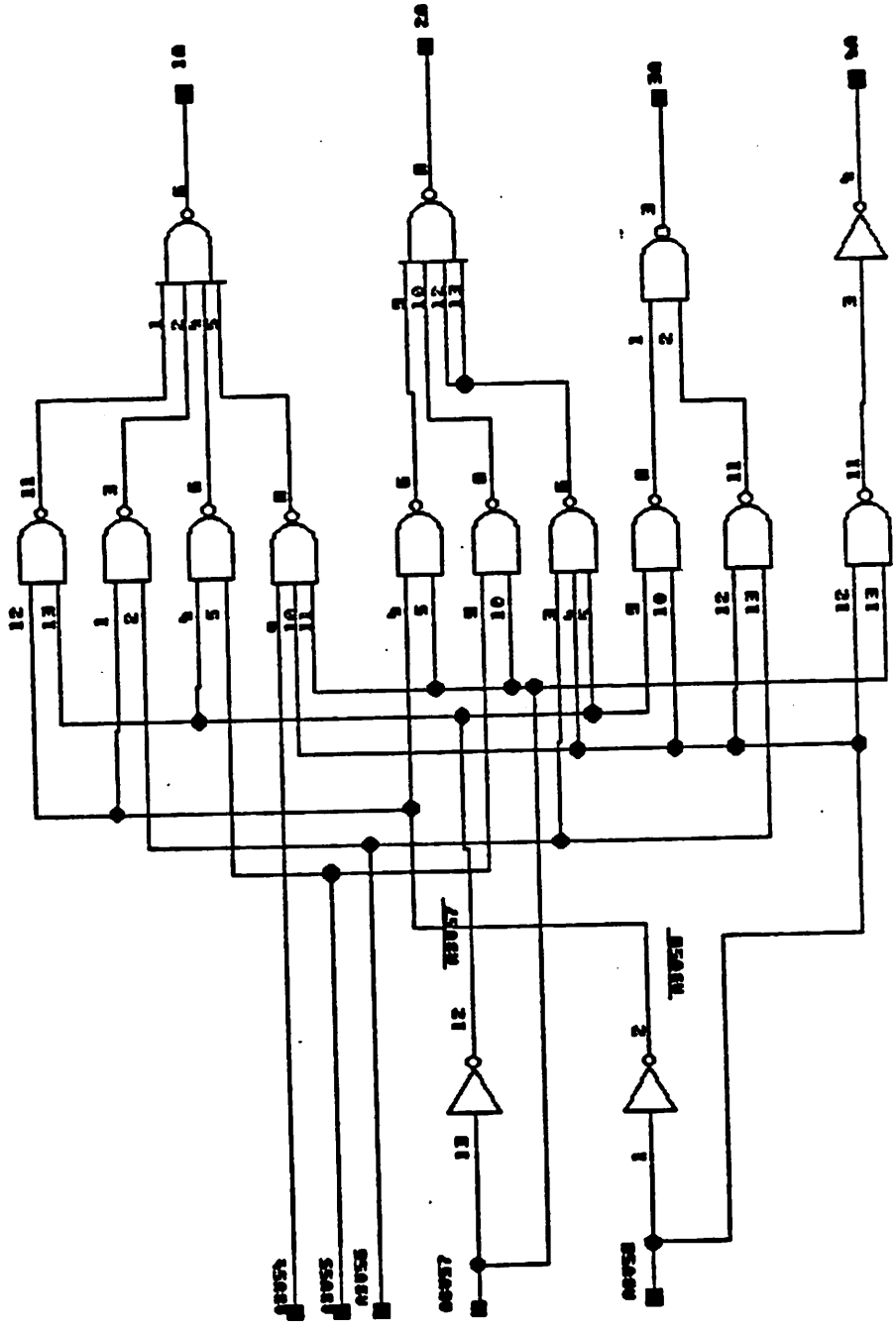
Schematic: 26/34 Coefficient Memory and Interface: 14/22



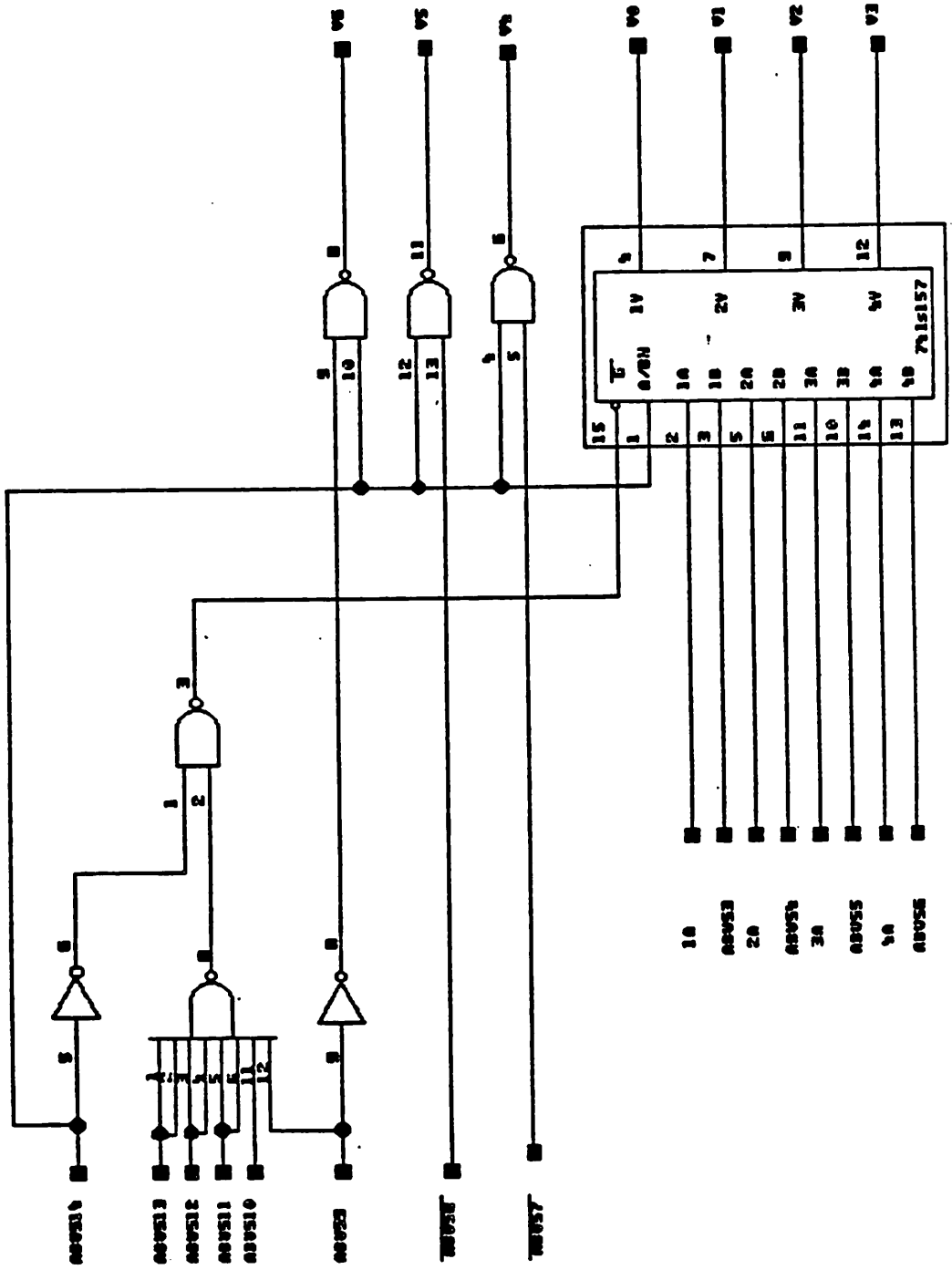
Schematic: 27/34 Coefficient Memory and Interface: 15/22



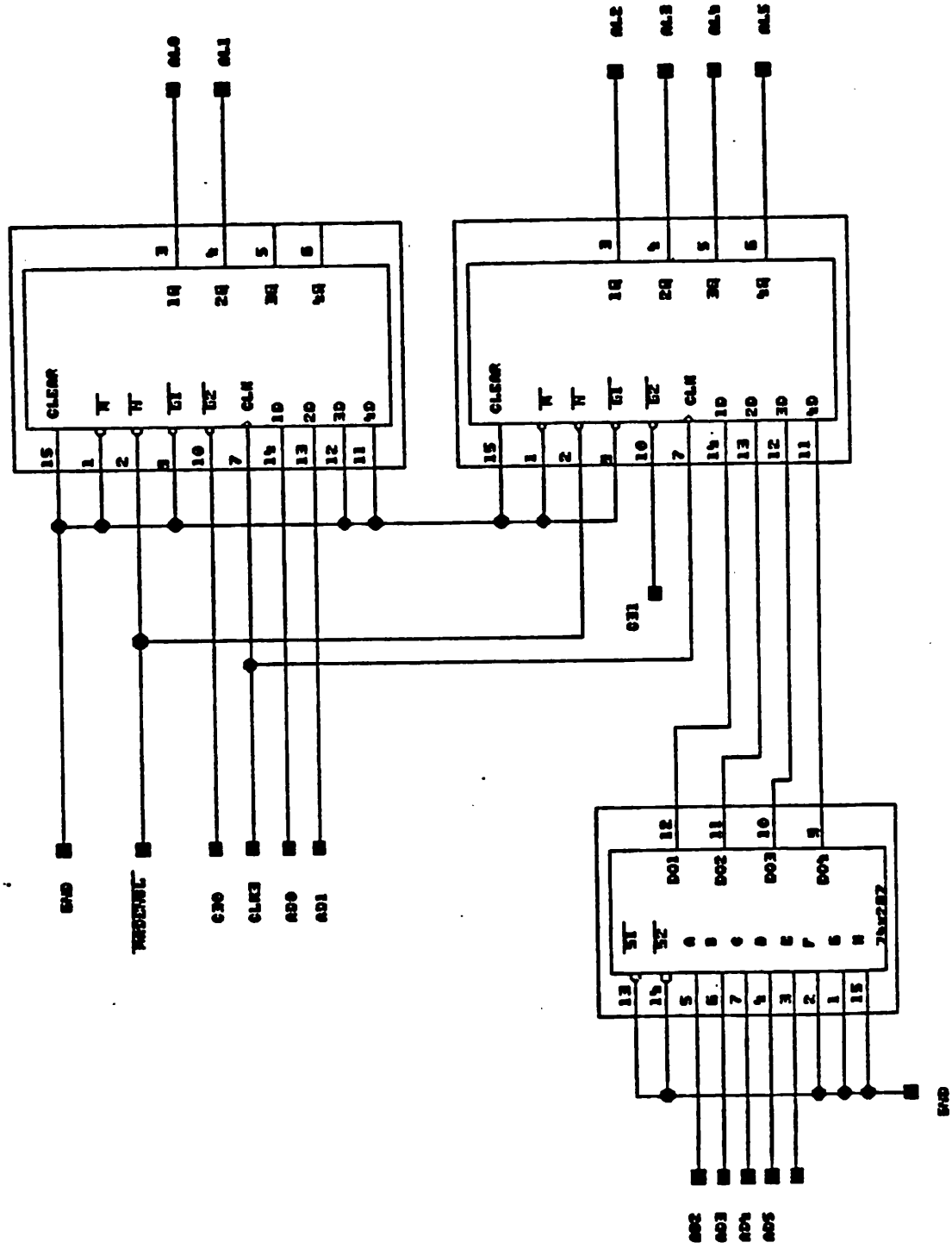
Schematic: 28/34 Coefficient Memory and Interface: 16/22



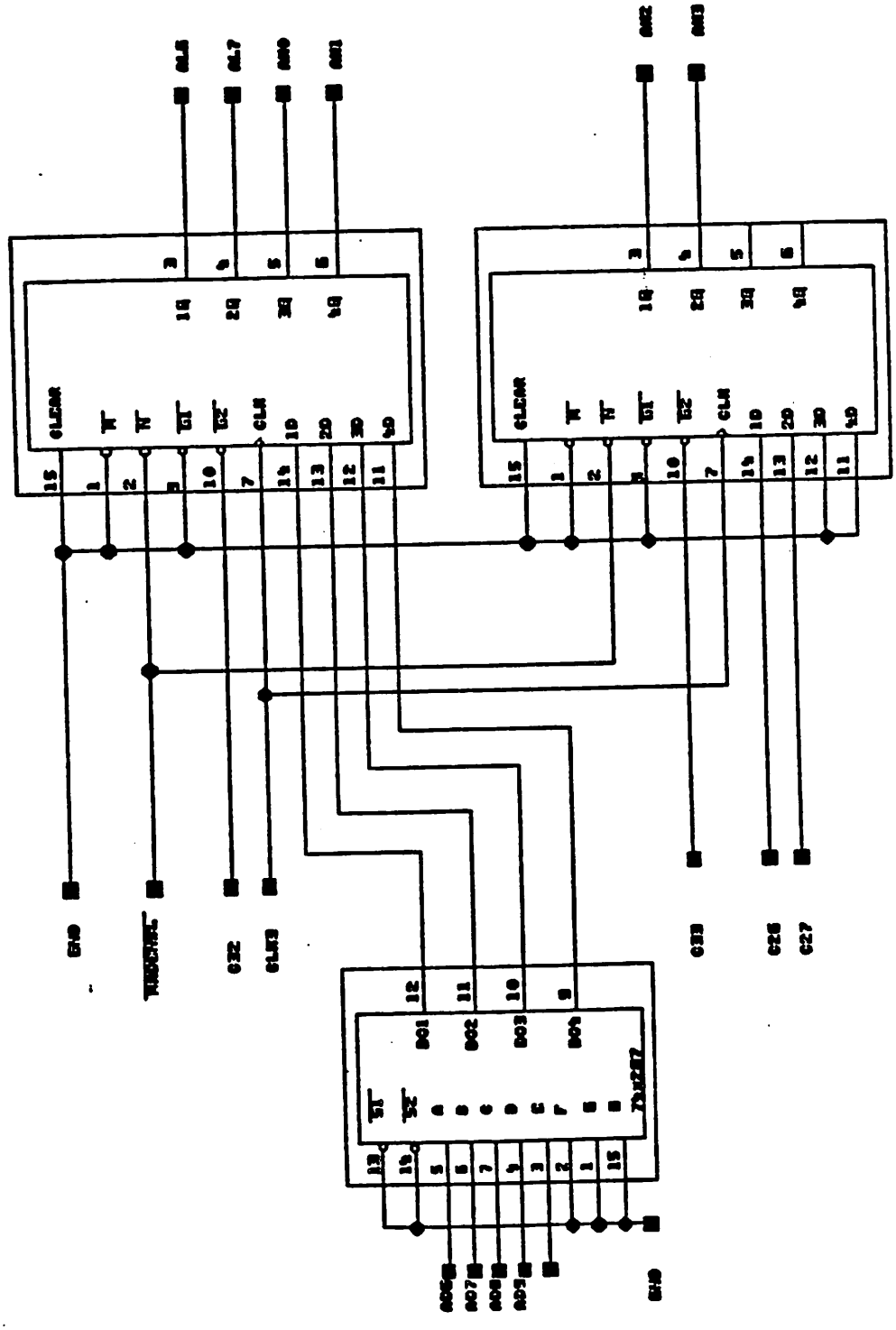
Schematic: 29/34 Coefficient Memory and Interface: 17/22



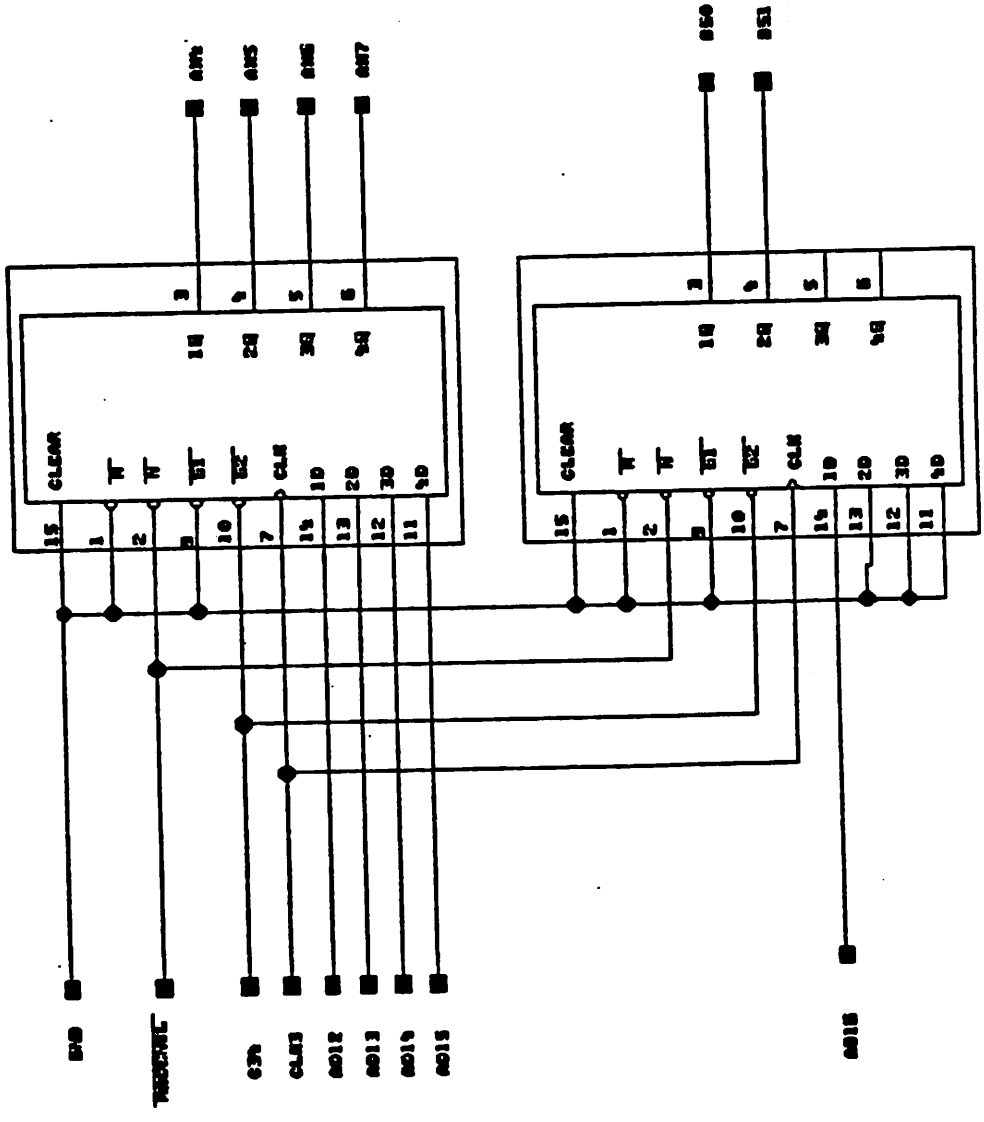
Schematic: 30/34 Coefficient Memory and Interface: 18/22



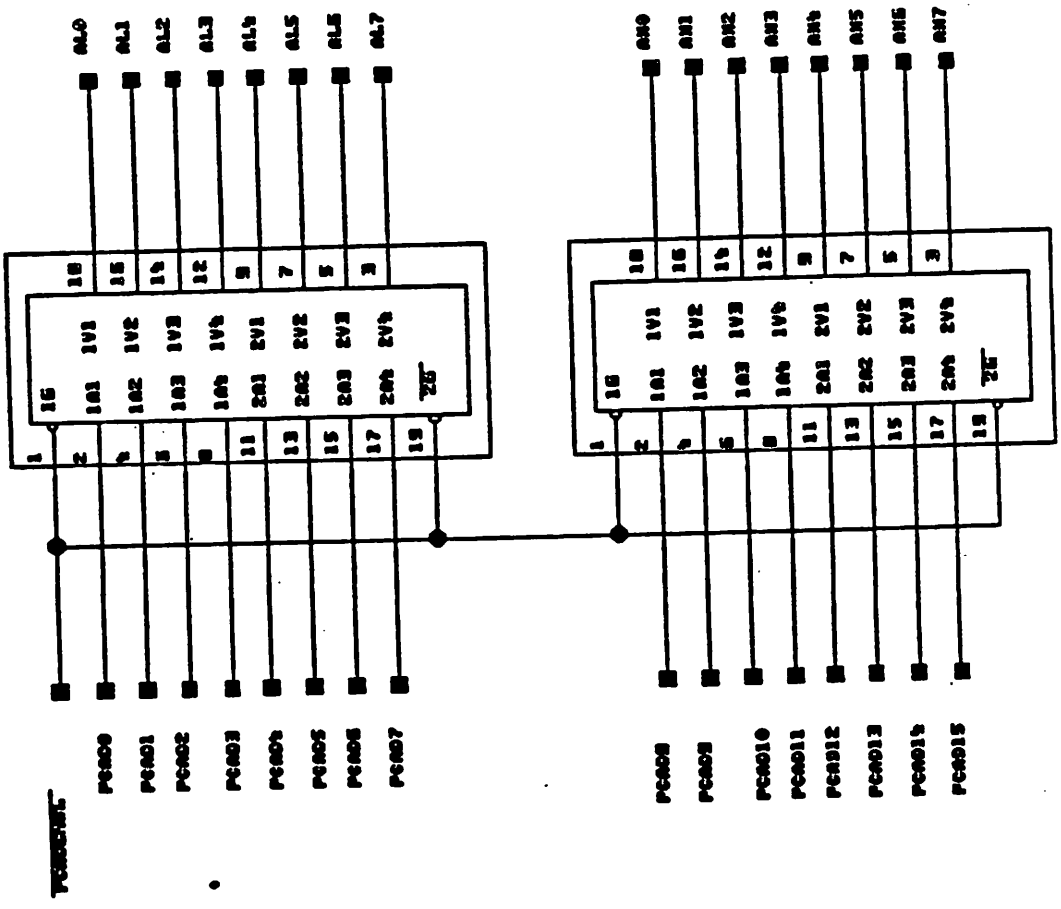
Schematic: 31/34 Coefficient Memory and Interface: 19/22



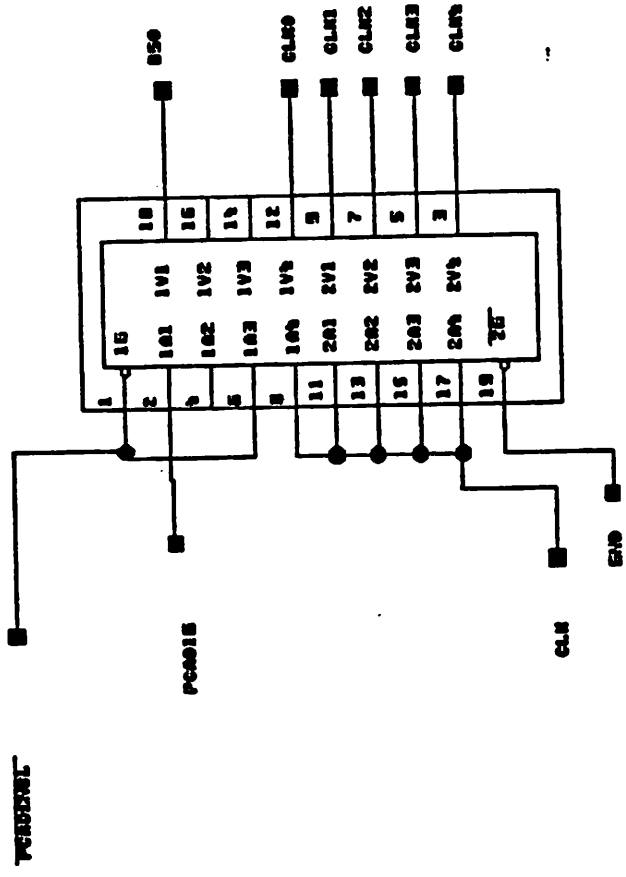
Schematic: 32/34 Coefficient Memory and Interface: 20/22



Schematic: 33/34 Coefficient Memory and Interface: 21/22



Schematic: 34/34 Coefficient Memory and Interface: 22/22



APPENDIX H

BIASC Model-Evaluation Routine Adjusted for the MMAP

C-Program Routine

```
/*
 * Update MOS elements with MMAP.
 */

#include <stdio.h>
#include <math.h>

#include "defs.h"
#include "types.h"
#include "nextern.h"

#define VMAX 30.0

struct send_data {
    char model[4];
    float vds[4];
    float vgs[4];
    float vsb[4];
    float scale[4];
};

struct receive_data {
    float ids[4];
    float gds[4];
    float ggs[4];
    float gsb[4];
};

extern double  abstol, reltol, vntol, trtol;

/*
 * Update mos elements.
 */

hupmos()
{
    int rev[4], orev[4], i;
    double vd,vg,vs,vb,idn;

    struct send_data mosl;
    struct receive_data mosr;

    struct mosfet *m, *pm, *cm;
    struct mosimfo *mi;

    extern int nocon;
    extern struct mosfet *mos;

    /* limit terminal voltages of first four of less transistors */
    for(i=0, m=mos, pm=mos ; m && i<4 ; m=m->nextmos, i++) {
        mi=m->mim;
        vd=0.0;
```

```
    vg=0.0;
    vs=0.0;
    vb=0.0;
    if (mi->ndvalue) vd = mi->ndvalue->ovalue;
    if (mi->ngvalue) vg = mi->ngvalue->ovalue;
    if (mi->nsvalue) vs = mi->nsvalue->ovalue;
    if (mi->nbvalue) vb = mi->nbvalue->ovalue;
    if (vd < vs) {
       rev[i] = 1;
       mosl.vds[i] = min(vs-vd,VMAX);
       mosl.vgs[i] = max(min(vg-vd,VMAX),0.0);
       mosl.vsb[i] = max(vd-vb,-0.6);
    }
    else {
       rev[i] = 0;
       mosl.vds[i] = min(vd-vs,VMAX);
       mosl.vsb[i] = max(vs-vb,-0.6);
       mosl.vgs[i] = max(min(vg-vs,VMAX),0.0);
    }
    mosl.scale[i] = m->scale;
      mosl.model[i] = m->modnum;
}

/* send data to MMAP and signal MMAP to begin          */
loadd(&mosl);
start();
cm=m;

/* begin internal loop    */
while(cm) {
   for(i=0, m=cm ; m && i<4 ; m=m->nextmos, i++) {
   /* limit terminal voltages of first four transistors */
       mi=m->mim;
       vd=0.0;
       vg=0.0;
       vs=0.0;
       vb=0.0;
       if (mi->ndvalue) vd = mi->ndvalue->ovalue;
       if (mi->ngvalue) vg = mi->ngvalue->ovalue;
       if (mi->nsvalue) vs = mi->nsvalue->ovalue;
       if (mi->nbvalue) vb = mi->nbvalue->ovalue;
         orev[i] = rev[i];
       if (vd < vs) {
          rev[i] = 1;
          mosl.vds[i] = min(vs-vd,VMAX);
          mosl.vgs[i] = max(min(vg-vd,VMAX),0.0);
          mosl.vsb[i] = max(vd-vb,-0.6);
       }
       else {
          rev[i] = 0;
          mosl.vds[i] = min(vd-vs,VMAX);
          mosl.vsb[i] = max(vs-vb,-0.6);
          mosl.vgs[i] = max(min(vg-vs,VMAX),0.0);
       }
       mosl.scale[i] = m->scale;
         mosl.model[i] = m->modnum;
```

```
}

/* access results from mmap */
unloadd(&mosr);

/* send data to MMAP and signal MMAP to begin                    */
loadd(&mosl);
start();
  cm=m;

/* Load Circuit Matrix  and check convergence */
for(i=0, m=pm; pm && i<4 ; m=m->nextmos,i++) {
   mi=m->mim;
   idn = mosr.ids[i] - mosr.gds[i]*mosl.vds[i];
   idn -= mosr.ggs[i]*mosl.vgs[i] + mosr.gsb[i]*mosl.vsb[i];

   /* check for convergence                                     */
   if (abs(mi->oids-mosr.ids[i])
       >= abstol + reltol * min(abs(mosr.ids[i]),abs(mi->oids)))
      ++nocon;

   mi->oids = mosr.ids[i];

   if (!orev[i]) {
      if (mi->mpt11)
         mi->mpt11->value += mosr.gds[i];
      if (mi->mpt12)
         mi->mpt12->value += mosr.ggs[i];
      if (mi->mpt13)
         mi->mpt13->value += mosr.gsb[i] - mosr.gds[i] -mosr.ggs[i];
      if (mi->mpt14)
         mi->mpt14->value -= mosr.gsb[i];
      if (mi->mpt31)
         mi->mpt31->value -= mosr.gds[i];
      if (mi->mpt32)
         mi->mpt32->value -= mosr.ggs[i];
      if (mi->mpt33)
         mi->mpt33->value += mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
      if (mi->mpt34)
         mi->mpt34->value += mosr.gsb[i];
      if (mi->mrhs1)
         mi->mrhs1->rhvalue -= idn;
      if (mi->mrhs3)
         mi->mrhs3->rhvalue += idn;
   }

   else {
      if (mi->mpt11)
         mi->mpt11->value += mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
      if (mi->mpt12)
         mi->mpt12->value -= mosr.ggs[i];
      if (mi->mpt13)
         mi->mpt13->value -= mosr.gds[i];
      if (mi->mpt14)
         mi->mpt14->value += mosr.gsb[i];
      if (mi->mpt31)
```

```
            mi->mpt31->value -= mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
        if (mi->mpt32)
            mi->mpt32->value += mosr.ggs[i];
        if (mi->mpt33)
            mi->mpt33->value += mosr.gds[i];
        if (mi->mpt34)
            mi->mpt34->value -= mosr.gsb[i];
        if (mi->mrhs1)
            mi->mrhs1->rhvalue += idn;
        if (mi->mrhs3)
            mi->mrhs3->rhvalue -= idn;
    }

  }
    pm = m;
}

/* access results from mmap */
unloadd(&mosr);

for(i=0, m=pm; pm && i<4 ; m=m->nextmos,i++) {
  mi=m->mim;
  idn = mosr.ids[i] - mosr.gds[i]*mosl.vds[i];
  idn -= mosr.ggs[i]*mosl.vgs[i] + mosr.gsb[i]*mosl.vsb[i];

  /* check for convergence                                    */
  if (abs(mi->oids-mosr.ids[i])
      >= abstol + reltol * min(abs(mosr.ids[i]),abs(mi->oids)))
     ++nocon;

  mi->oids = mosr.ids[i];

  if (!rev[i]) {
     if (mi->mpt11)
        mi->mpt11->value += mosr.gds[i];
     if (mi->mpt12)
        mi->mpt12->value += mosr.ggs[i];
     if (mi->mpt13)
        mi->mpt13->value += mosr.gsb[i] - mosr.gds[i] - mosr.ggs[i];
     if (mi->mpt14)
        mi->mpt14->value -= mosr.gsb[i];
     if (mi->mpt31)
        mi->mpt31->value -= mosr.gds[i];
     if (mi->mpt32)
        mi->mpt32->value -= mosr.ggs[i];
     if (mi->mpt33)
        mi->mpt33->value += mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
     if (mi->mpt34)
        mi->mpt34->value += mosr.gsb[i];
     if (mi->mrhs1)
        mi->mrhs1->rhvalue -= idn;
     if (mi->mrhs3)
        mi->mrhs3->rhvalue += idn;
  }

  else {
```

```
        if (mi->mpt11)
           mi->mpt11->value += mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
        if (mi->mpt12)
           mi->mpt12->value -= mosr.ggs[i];
        if (mi->mpt13)
           mi->mpt13->value -= mosr.gds[i];
        if (mi->mpt14)
           mi->mpt14->value += mosr.gsb[i];
        if (mi->mpt31)
           mi->mpt31->value -= mosr.gds[i] + mosr.ggs[i] - mosr.gsb[i];
        if (mi->mpt32)
             mi->mpt32->value += mosr.ggs[i];
        if (mi->mpt33)
             mi->mpt33->value += mosr.gds[i];
        if (mi->mpt34)
             mi->mpt34->value -= mosr.gsb[i];
        if (mi->mrhs1)
             mi->mrhs1->rhvalue += idn;
        if (mi->mrhs3)
             mi->mrhs3->rhvalue -= idn;
     }

   }

}

start()
{

   char *ibuffer=0;

   /* Load check character     */
   poke(0x8000,0x0FFE,ibuffer,1);

   /* Load start character     */
   peek(0x8000,0x0FFF,ibuffer,4);

   return;

}
```

Assembly-Language Routines

```
                TITLE LOAD ROUTINE ASSEMBLER
                NAME LOADD
                INCLUDE     DOS.MAC

DSG             EQU         8000H

                PSEG

                PUBLIC      LOADD
LOADD           PROC  FAR

; Save state of registers to be used
                PUSH  AX
                PUSH  CX
                PUSH  SI
                PUSH  DI
                PUSH  DS
                PUSH  ES

; Clear D flag so the SI register will be incremented
                CLD

; Load Extra Segment Via AX register
                MOV         AX.DSG
                MOV         ES.AX

; Load pointer address that is passed to function
                PUSH  BP
                MOV   BP.SP
                MOV         SI.[BP+18]
                POP         BP

; Transfer data via the lower 8 bits of the AX register
; Model
                LODSB
                MOV   ES:[0H].AL
                LODSB
                MOV   ES:[100H].AL
                LODSB
                MOV   ES:[200H].AL
                LODSB
                MOV   ES:[300H].AL

; Vds
                LODSB
                MOV   ES:[7H].AL
                LODSB
                MOV   ES:[6H].AL
                LODSB
                MOV   ES:[5H].AL
                LODSB
                MOV   ES:[4H].AL
                LODSB
```

```
MOV  ES:[107H].AL
LODSB
MOV  ES:[106H].AL
LODSB
MOV  ES:[105H].AL
LODSB
MOV  ES:[104H].AL
LODSB
MOV  ES:[207H].AL
LODSB
MOV  ES:[206H].AL
LODSB
MOV  ES:[205H].AL
LODSB
MOV  ES:[204H].AL
LODSB
MOV  ES:[307H].AL
LODSB
MOV  ES:[306H].AL
LODSB
MOV  ES:[305H].AL
LODSB
MOV  ES:[304H].AL
```

: Vgs

```
LODSB
MOV  ES:[0BH].AL
LODSB
MOV  ES:[0AH].AL
LODSB
MOV  ES:[9H].AL
LODSB
MOV  ES:[8H].AL
LODSB
MOV  ES:[10BH].AL
LODSB
MOV  ES:[10AH].AL
LODSB
MOV  ES:[109H].AL
LODSB
MOV  ES:[108H].AL
LODSB
MOV  ES:[20BH].AL
LODSB
MOV  ES:[20AH].AL
LODSB
MOV  ES:[209H].AL
LODSB
MOV  ES:[208H].AL
LODSB
MOV  ES:[30BH].AL
LODSB
MOV  ES:[30AH].AL
LODSB
MOV  ES:[309H].AL
LODSB
```

```
                MOV   ES:[308H].AL

: Vsb
                LODSB
                MOV   ES:[0FH].AL
                LODSB
                MOV   ES:[0EH].AL
                LODSB
                MOV   ES:[0DH].AL
                LODSB
                MOV   ES:[0CH].AL
                LODSB
                MOV   ES:[10FH].AL
                LODSB
                MOV   ES:[10EH].AL
                LODSB
                MOV   ES:[10DH].AL
                LODSB
                MOV   ES:[10CH].AL
                LODSB
                MOV   ES:[20FH].AL
                LODSB
                MOV   ES:[20EH].AL
                LODSB
                MOV   ES:[20DH].AL
                LODSB
                MOV   ES:[20CH].AL
                LODSB
                MOV   ES:[30FH].AL
                LODSB
                MOV   ES:[30EH].AL
                LODSB
                MOV   ES:[30DH].AL
                LODSB
                MOV   ES:[30CH].AL

: Scale
                LODSB
                MOV   ES:[13H].AL
                LODSB
                MOV   ES:[12H].AL
                LODSB
                MOV   ES:[11H].AL
                LODSB
                MOV   ES:[10H].AL
                LODSB
                MOV   ES:[113H].AL
                LODSB
                MOV   ES:[112H].AL
                LODSB
                MOV   ES:[111H].AL
                LODSB
                MOV   ES:[110H].AL
                LODSB
                MOV   ES:[213H].AL
                LODSB
```

```
                MOV   ES:[212H].AL
                LODSB
                MOV   ES:[211H].AL
                LODSB
                MOV   ES:[210H].AL
                LODSB
                MOV   ES:[313H].AL
                LODSB
                MOV   ES:[312H].AL
                LODSB
                MOV   ES:[311H].AL
                LODSB
                MOV   ES:[310H].AL

; Return the state of the machine
                POP           ES
                POP           DS
                POP           DI
                POP           SI
                POP           CX
                POP           AX

                RET
LOADD           ENDP
                PAGE
                ENDPS
                END
```

```
                TITLE UNLOAD ROUTINE
                NAME UNLOADD
                INCLUDE     DOS.MAC

DSG             EQU         8000H
CNT             EQU         0FH
INDO  EQU           0H

                PSEG

                PUBLIC      UNLOADD
UNLOADD     PROC  FAR

: Save state of registers to be used
                PUSH  CX
                PUSH  SI
                PUSH  DI
                PUSH  DS
                PUSH  ES

: Clear D flag so the SI register will be incremented
                CLD

: Load Extra Segment Via AX register
                MOV         AX.DS
                MOV         ES.AX
                MOV         AX.DSG
                MOV         DS.AX

: Load pointer address that is passed to function
                PUSH  BP
                MOV   BP.SP
                MOV         DI.[BP+16]
                POP         BP

: lds
                MOV   AL.DS:[3H]
                STOSB
                MOV   AL.DS:[2H]
                STOSB
                MOV   AL.DS:[1H]
                STOSB
                MOV   AL.DS:[0H]
                STOSB
                MOV   AL.DS:[403H]
                STOSB
                MOV   AL.DS:[402H]
                STOSB
                MOV   AL.DS:[401H]
                STOSB
                MOV   AL.DS:[400H]
                STOSB
                MOV   AL.DS:[803H]
                STOSB
                MOV   AL.DS:[802H]
                STOSB
```

```
                MOV   AL.DS:[801H]
                STOSB
                MOV   AL.DS:[800H]
                STOSB
                MOV   AL.DS:[0C03H]
                STOSB
                MOV   AL.DS:[0C02H]
                STOSB
                MOV   AL.DS:[0C01H]
                STOSB
                MOV   AL.DS:[0C00H]
                STOSB

: Gds
                MOV   AL.DS:[7H]
                STOSB
                MOV   AL.DS:[6H]
                STOSB
                MOV   AL.DS:[5H]
                STOSB
                MOV   AL.DS:[4H]
                STOSB
                MOV   AL.DS:[407H]
                STOSB
                MOV   AL.DS:[406H]
                STOSB
                MOV   AL.DS:[405H]
                STOSB
                MOV   AL.DS:[404H]
                STOSB
                MOV   AL.DS:[807H]
                STOSB
                MOV   AL.DS:[806H]
                STOSB
                MOV   AL.DS:[805H]
                STOSB
                MOV   AL.DS:[804H]
                STOSB
                MOV   AL.DS:[0C07H]
                STOSB
                MOV   AL.DS:[0C06H]
                STOSB
                MOV   AL.DS:[0C05H]
                STOSB
                MOV   AL.DS:[0C04H]
                STOSB

: Ggs
                MOV   AL.DS:[0BH]
                STOSB
                MOV   AL.DS:[0AH]
                STOSB
                MOV   AL.DS:[9H]
                STOSB
                MOV   AL.DS:[8H]
                STOSB
```

```
        MOV   AL.DS:[40BH]
        STOSB
        MOV   AL.DS:[40AH]
        STOSB
        MOV   AL.DS:[409H]
        STOSB
        MOV   AL.DS:[408H]
        STOSB
        MOV   AL.DS:[80BH]
        STOSB
        MOV   AL.DS:[80AH]
        STOSB
        MOV   AL.DS:[809H]
        STOSB
        MOV   AL.DS:[808H]
        STOSB
        MOV   AL.DS:[0C0BH]
        STOSB
        MOV   AL.DS:[0C0AH]
        STOSB
        MOV   AL.DS:[0C09H]
        STOSB
        MOV   AL.DS:[0C08H]
        STOSB

; Gsb
        MOV   AL.DS:[0FH]
        STOSB
        MOV   AL.DS:[0EH]
        STOSB
        MOV   AL.DS:[0DH]
        STOSB
        MOV   AL.DS:[0CH]
        STOSB
        MOV   AL.DS:[40FH]
        STOSB
        MOV   AL.DS:[40EH]
        STOSB
        MOV   AL.DS:[40DH]
        STOSB
        MOV   AL.DS:[40CH]
        STOSB
        MOV   AL.DS:[80FH]
        STOSB
        MOV   AL.DS:[80EH]
        STOSB
        MOV   AL.DS:[80DH]
        STOSB
        MOV   AL.DS:[80CH]
        STOSB
        MOV   AL.DS:[0C0FH]
        STOSB
        MOV   AL.DS:[0C0EH]
        STOSB
        MOV   AL.DS:[0C0DH]
        STOSB
```

```
                MOV   AL.DS:[0C0CH]
                STOSB

; Return the state of the machine
                POP             ES
                POP             DS
                POP             DI
                POP             SI
                POP             CX

                RET
UNLOADD         ENDP
                PAGE
                ENDPS
                END
```

REFERENCES

[Adl84] P. Adler. *IEOR262A Class Notes*. University of California. Berkeley. California. 1984.

[AMD83] *Bipolar Microprocessor Logic and Interface*. Advanced Micro Devices Co.. 1983.

[BVS83] J. Barby. J. Vlach and K. Singhal. "Polynomial Splines for FET Models". *Proc. 1983 Internaltional Symposium on Circuits and Systems*. May 1983.

[BVS84] J. Barby. J. Vlach and K. Singhal. "Optimized Polynomial Splines for FET Models". *Proc. 1984 Internaltional Symposium on Circuits and Systems*. May 1984.

[Blu85] W. Blum. "The PSPICE Simulation Program". *Wescon/85 Professional Program Session Record 32/2* (November 1985).

[Bur84] J. L. Burns. "Empirical Mosfet Models for Circuit Simulation". Memo No. UCB/Electronics Research Lab. M84/43. Electronics Research Laboratory. University of California. Berkeley. California. May 1984.

[ChL75] L. Chua and P. Lin. *Computer Aided Analysis of Electronic Circuits; Algorithms and Computational Techniques*. Prentice Hall. New Jersey. 1975.

[Coh76] E. Cohen. "Program Reference For SPICE2". Memo No. Electronics Research Lab.-M592. Electronics Research Laboratory. University of California. Berkeley. June 1976.

[Coh81] E. Cohen. "Performance Limits of Integrated Circuit Simulation on a Dedicated Minicomputer System". Memo No. UCB/Electronics Research Lab. M81/29. Electronics Research Laboratory. University of California. Berkeley. California. May 1981.

[D'R85] J. L. D'Arcy and R. C. Rennick, "Mosfet Paramter Optimization for Accurate Output Conductance Modeling", *Proc. 1985 IEEE Custom Integrated Circuits Conference*, Portland, Oregon, May 1985.

[DaB74] G. Dahlquist and A. Bjorck, *Numerical Methods*, Prentice-Hall, 1974.

[Dec84] P. Decher, Private Communication, 1984.

[Deu85] J. T. Deutsch, "Algorithms and Architecture for Multiprocessor-Based Circuit Simulation", Memo No. UCB/Electronics Research Lab. M85/39, Electronics Research Laboratory, University of California, Berkeley, California, May 1985.

[FrC80] F. N. Fritsch and R. E. Carlson, "Monotone Piecewise Cubic Interpolation", *SIAM Journal of Numerical Analysis 17*, 2 (April 1980).

[Gyu81] R. S. Gyurcsik, *BIASB: Circuit Simulation Program For The Hewlett-Packard 9845*, University of California, Berkeley, December 1981.

[GMP84] R. S. Gyurcsik, K. Mayaram and D. O. Pederson, "Language Comparison For Circuit Simulation On Desktop Computers ", *Proc. IEEE International Symposium on Circuits and Systems*, Montreal, Canada, May 1984.

[GBP84] R. S. Gyurcsik, R. Brown and D. O. Pederson, "Circuit and Logic Simulation on the IBM Personal Computer", *Proc. IEEE International Symposium on Circuits and Systems*, Montreal, Canada, May 1984.

[Gyu85] R. S. Gyurcsik, "BIASC: A Circuit Smulation Program for the IBM PC", *Wescon/85 Professional Program Session Record 32/4* (November 1985).

[HoS84] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, 1984.

[IBM84] *Technical Reference Personal Computer XT*, IBM, 1984.

[Int83] *Memory Components Handbook*, Intel Co., 1983.

[Int84] *Microsystem Components Handbook*, Intel Co., 1984.

[JNP86] G. Jacob, A. R. Newton and D. O. Pederson, "Direct-Method Circuit Simulation Using Multiprocessors", *Proc. 1986 Internaltional Symposium on Circuits and Systems*, May 1986.

[Kah84] W. Kahan, *CS278 Class Notes*, University of California, Berkeley, California, 1984.

[KBF85] R. Kershaw, L. Bays, R. Freyman, J. Klinkowski, C. Miller, K. Mondal, H. Moscovitz, W. Stocker, L. Tran, W. Hays, J. Boddie, E. Fields, C. Garen and J. Tow, "A programmable Digital Signal Processor with 32b Floating-Point Arithmetic", *Proc. 1985 IEEE International SOlid-State Circuits Conference*, New York, New York, February 1985.

[LaH74] C. Lawson and R. Hanson, *Solving Least Sqares Problems*, Prentice-Hall, 1974.

[LRS81] E. Lelarasmee, A. E. Ruehli and A. L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large-Scale Integrated Circuits", Memo No. UCB/Electronics Research Lab. M81/75, Electronics Research Laboratory, University of California, Berkeley, California, June 1981.

[Liu81] S. Liu, "A Unified CAD Model for MOSFETs", Memo No. UCB/Electronics Research Lab. M81/31, Electronics Research Laboratory, University of California, Berkeley, California, May 1981.

[Man82] M. Mano, *Computer System Architecture*, Prentice-Hall, 1982.

[May86] K. Mayaram, Private Communication, 1986.

[McC] W. J. McCalla, Computer-Aided Circuit Simulation Techniques, pre-publication manuscript.

[Nag75] L. W. Nagel, "SPICE2 - A Computer Program to Simulate Semiconductor Circuits", Memo No. Electronics Research Lab.-M520, Electronics Research Laboratory, University of California, Berkeley, California, May 1975.

[New78] A. R. Newton. "The Simulation of Large-Scale Integrated Circuits". Memo No. UCB/Electronics Research Lab. M78/52. Electronics Research Laboratory. University of California. Berkeley. California. July 1978.

[NeS84] A. R. Newton and A. L. Sangiovanni-Vincentelli. "Relaxation-Based Electrical Simulation". *IEEE Transactions on Computer Aided Design CAD-3,4* (October 1984). 308-329.

[Qua85] T. L. Quarles. Private communications. 1985.

[ReA80] R. Rector and G. Alexy. *The 8086 Book*. McGraw-Hall. 1980.

[Sal84] R. Saleh. "Iterated Timing Analysis and SPLICE1". Memo No. UCB/Electronics Research Lab. M84/2. Electronics Research Laboratory. University of California. Berkeley. California. January 1984.

[San80] A. L. Sangiovanni-Vincentelli. "Circuit Simulation". *NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits*. Sogesta-Urbino. Italy. July 1980.

[She85] B. J. Sheu. "MOS Transistor Modeling and Characterization for Circuit Simulation". Memo No. UCB/Electronics Research Lab. M85/85. Electronics Research Laboratory. University of California. Berkeley. California. October 1985.

[ShH68] H. Shichman and D. A. Hodges. "Modeling and Simulation of Insulated-Gate Field-Effect Transistor Switching Circuits". *IEEE Journal of Solid-State Circuits SC-3* (September 1968). 285-289.

[SSM82] T. Shima. T. Sugawara. S. Moriyama and H. Yamada. "Three-Dimensional Table Look-Up MOSFET Model for Precise Circuit Simulation". *IEEE Journal of Solid-State Circuits CAS-22*. 12 (June 1982). 901-909.

[SYD83] T. Shima. H. Yamada and R. Dang. "Table Look-Up MOSFET Modeling System Using a 2-D Device Simulator and Monotonic Piecewise Cubic Interpolation". *IEEE Transactions on Computer Aided Design CAD-2*. 2 (April 1983). 121-126.

[SBN82] D. Siewiorek, C. Bell and A. Newell. *Computer Structures Principles and Examples*. McGraw Hill. 1982.

[Sta83] R. Starz. *8087 Application and Programming for the IBM PC and Other PC's*. Brady. 1983.

[Str76] G. Strang. *Linear Algebra and its Applications*. Academic Press. 1976.

[Sub85] P. Subramaniam. "Modeling MOS Circuits for Timing Simulation". *Proc. 1985 IEEE Custom Integrated Circuits Conference*. Portland. Oregon. May 1985.

[Tex76] *The TTL Data Book for Design Engineers*. Texas Instruments Co.. 1976.

[TsM84] Y. Tsividis and G. Masetti. "Problems in the Precision Modeling of the MOS Transistor for Analog Applications". *IEEE Transactions on Computer Aided Design CAD-1,4* (January 1984). 72-79.

[VZN80] A. Vladimirescu. K. Zhang. A. R. Newton. D. O. Pederson and A. Sangiovanni-Vincentelli. *SPICE Version 2G User's Guide*. University of California. Berkeley. October 1980.

[VlL80] A. Vladimirescu and S. Liu. "The Simulation of MOS Integrated Circuits Using SPICE2". Memo No. UCB/Electronics Research Lab. M80/7. Electronics Research Laboratory. University of California. Berkeley. California. October 1980.

[Vla82] A. Vladimirescu. "LSI Circuit Simulation on Vector Computers". Memo No. UCB/Electronics Research Lab. M82/75. Electronics Research Laboratory. University of California. Berkeley. California. October 1982.

[WJM73] W. T. Weeks. A. J. Jimenez. G. W. Mahoney. D. Mehta. H. Qassemzadeh and T. R. Scott. "Algorithms for ASTAP - A Network Analysis Program". *IEEE Transactions on Circuit Theory CT.-20* (November 1973). 628-634.

[Wei84] *WTL 1032/1033 High-Speed 32-Bit Floating-Point Multiplier/ALU*. Weitek Co.. 1984.

[WhS83] J. White and A. L. Sangiovanni-Vincentelli, "Relax2: A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits", *Proc. 1983 Internaltional Symposium on Circuits and Systems*, May 1983.

[WSS85] J. White, R. Saleh, A. L. Sangiovanni-Vincentelli and A. R. Newton, "Accelerating Relaxation Algorithms For Circuit Simulation Using Waveform-Newton,Iterative Step Size Refinement, and Parallel Techniques", *Proc. IEEE International Conference on Comuter Aided Design*, Santa Clara, California, September 1985.

[YEC83] P. Yang, B. Epler and P. Chatterjee, "An Investigation of the Charge Conservation Problem for MOSFET Circuit Simulation", *IEEE Journal of Solid-State Circuits SC-18* (February 1983), 128-138.