EVALUATION AND OPTIMIZATION OF MOS DEVICE DRAIN CONDUCTANCE MODELING IN THE SPICE LEVEL 2 MODEL

by

Gregory D. Anderson

Memorandum No. UCB/ERL M84/3

6 January 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EVALUATION AND OPTIMIZATION OF MOS DEVICE DRAIN CONDUCTANCE MODELING IN THE SPICE LEVEL 2 MODEL

by

G. D. Anderson

Memorandum No. UCB/ERL M84/3

6 January 1984

EVALUATION AND OPTIMIZATION OF MOS DEVICE DRAIN CONDUCTANCE MODELING IN THE SPICE LEVEL 2 MODEL

by

Gregory D. Anderson

Memorandum No. UCB/ERL M84/3

6 January 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Evaluation and Optimization of MOS Device Drain Conductance Modeling in the SPICE Level 2 Model

by Gregory D. Anderson

Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley

ABSTRACT

This report addresses output conductance modeling in SPICE in the near-saturation region. I_d vs V_{ds} characteristics for MOS transistors fabricated in the Microelectronics Laboratory at the University of California, Berkeley were measured, and output conductance information taken from these measurements. Then, a set of optimal SPICE MOS Level 2 model parameters was extracted from the data in such a way as to minimize the error between the measured and simulated I_d-V_{ds} data, and differences between the two were quantified. In addition, the use of the TECAP2* transistor measurement system as a tool for measurement, parameter extraction and model development is discussed.

October 7, 1983

*The TECAP2 Measurement System is a product of the Engineering Productivity Division of Hewlett- Packard Co., Cupertino, CA

ACKNOWLEDGEMENT

Many people have contributed to the work which has culminated in this report. The support and encouragement provided by my research advisor, Professor P. R. Gray is greatly appreciated.

In addition, the assistance provided by Dr. E. Khalily and Mr. P. H. Decher of Hewlett-Packard with the TECAP2 system was invaluable to the completion of this project.

I would also like to thank Mr. P. W. Li for supplying the devices measured for this project, and last but not least, Bell Laboratories for continuing patience and financial support.

Table of Contents

I. Introduction.

As MOS circuits continue to play an increasing role in analog circuit design, proper modeling of MOS transistor characteristics becomes more important in circuit simulation. Much research has been devoted to modeling short channel effects [refs] such as threshold voltage shift, but less attention has been paid to modeling output conductance in the saturation and near saturation regions. Output conductance is an important parameter in analog and digital circuit design because it is one of the factors which determines the gain of an actively loaded inverter stage.

Consider the gain stage with cascode active load shown in Figure 1. Neglecting the output resistance of M_3, the voltage gain A_v of this stage is

$$A_v = -\frac{g_{m_3}}{g_{ds_1}}\left(1+\frac{g_{m_2}}{g_{ds_2}}\right)$$

where g_{m_2} is the transconductance of M_2, g_{m_3} is the transconductance of M_3, and g_{ds_1} and g_{ds_2} are the small signal output conductances of M_1 and M_2, respectively. When the cascode load is correctly designed, M_1 is biased so that it is operating with $V_{ds} \approx V_{dsat}$ in order to achieve maximum output voltage swing for the circuit while keeping all transistors biased in the saturation region to maintain linearity.

A typical I_d vs V_{ds} characteristic is shown in Figure 2. Recalling the definition of $g_{ds} = \frac{dI_d}{dV_{ds}}$, it is seen that g_{ds} is relatively constant in the region where $V_{ds} >> V_{dsat}$, where V_{dsat} is the saturation voltage and is indicated approximately on the graph. In the region where $V_{ds} \approx V_{dsat}$, the output conductance becomes a strong function of V_{ds}. Since M_1 is

biased so that $V_{ds_1} \approx V_{dsat_1}$, the gain of this example circuit is a function of V_{ds_1}. Proper modeling of the output conductance of MOS transistors operating in this region is essential to accurate simulation of such circuits.

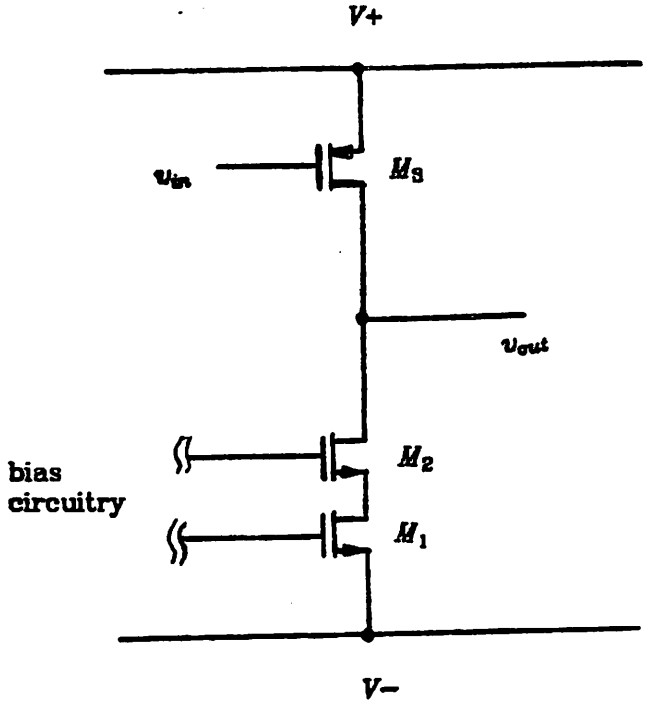

Figure 1: Cascode-loaded CMOS Gain Stage

II. Project Objective

In this paper, I_d vs V_{ds} characteristics for MOS transistors produced in the Microelectronics Laboratory at the University of California/Berkeley are measured, and output conductance data is taken from these measurements. Then, optimal SPICE MOS Level 2 [2] model parameters are extracted from the data in such a way as to minimize the error between the measured and simulated $I_d - V_{ds}$ curves. Finally, derivative g_{ds} data is taken from the measured and the simulated $I_d - V_{ds}$ characteristics, and differences between the two are shown. Particular emphasis is placed on the transition region where $V_{ds} \approx V_{dsat}$. In addition, the use of the TECAP2 transistor measurement system as a tool for measurement, parameter extraction and model development is discussed.

III. Data Collection and Reduction Procedure

A set of test devices produced in the Microelectronics Laboratory at Berkeley were obtained. These devices consisted of secondary test patterns on an experimental analog circuit wafer. NMOS and PMOS devices of varying geometries were available. N channel devices with $W=100\mu$, $L=10\mu$ were chosen primarily because they produced the most consistent results from device to device.

A. The TECAP2 Measurement System

The TECAP2 measurement system consists of the configuration shown schematically in Figure 3. The HP4145a semiconductor analyzer contains programmable voltage sources and voltage and current meters. Using the 9836 desktop computer, the 4145a can be programmed to take a series of current and voltage measurements, which the computer can then display graphically or store on disk. Initial measurements using this system were done at Hewlett Packard in Cupertino, Ca. Unfortunately, hardware problems with the probe station rendered useless the data taken at this time.

B. Measurements at Berkeley.

The transistor measurement system in the Solid State Laboratory at Berkeley is much more primitive than the TECAP2 system, and won't be described here. Measurements taken using this system were very noisy, and data smoothing was required to obtain useful results. See the discussion of data smoothing in section D.

C. Measurements Taken Using the TECAP2 System.

Because the measurement system at Berkeley was slow, noisy, and relatively hard to use, a return to HP to repair the prober was determined to be worthwhile. This was done, and measurements were taken. Comparison of data taken at Berkeley with measurements of the same device done at HP proved that the system at HP was indeed in working order. I_d vs V_{ds} was measured over a variety of ranges for each of the devices tested:

(1) V_{ds} 0 to 8 volts in 0.2 volt increments.

V_{gs} 0 to 5 volts in 1 volt steps.

This measurement yields a high voltage I-V characteristic which is typical of those used for model parameter extraction for circuit simulation. An example of the result of this measurement is shown in Figure 4.

(2) V_{ds} 0 to 500mV in 10mv increments.

V_{gs} 750mV to 1.25V in 25mV steps.

This measurement focuses on the area of interest in this paper: I_{ds} for V_{ds} in the region around saturation. The small (10mV) V_{ds} increments were necessary in order obtain a good approximation to the derivative g_{ds}. This measurement was done in two parts because TECAP2 can only store 500 measured data points at a time. An example of the result of this measurement is shown in Figures 5a and 5b.

All data was recorded on disk and later transferred to the ucbcad VAX computer. A listing of all data files appears in Appendix A.

D. Numerical Differentiation and Data Smoothing

A function $f(x)$ can be expanded about the point $x=x_0$ using the Taylor series:

(1) $f(x_0+\Delta x)=f(x_0)+f'(x_0)\Delta x+\frac{1}{2}f''(x_0)(\Delta x)^2+\cdots$

Similarly,

(2) $f(x_0-\Delta x)=f(x_0)-f'(x_0)\Delta x+\frac{1}{2}f''(x_0)(\Delta x)^2-\cdots$

subtracting (2) from (1) and dividing by $2\Delta x$ yields

(3) $f'(x_0)=\frac{f(x_0+\Delta x)-f(x_0-\Delta x)}{2\Delta x}+\cdots$

which is the so-called centered difference scheme for approximating the derivative $f'(x_0)$. This is the method which was used to numerically differentiate the measured I_d vs V_{ds} curves here.

Because the derivative data was of primary importance here, it was imperative that the measured I_d vs V_{ds} data contain as little noise as possible. The following steps were taken to reduce random error in those measurements:

(1) The probe station was well isolated; shielded probes and cables were used to minimize electrical interference.

(2) Measurements taken using the TECAP2 measurement system are actually made by the 4145a Semiconductor Analyzer. This device contains precision voltage sources, digital current meters and digital voltmeters for maximum accuracy. In addition, the 4145a has a user selectable "integration time"--short, medium, and long--- which means that the user may choose to have the instrument display for each data point the average of 8, 16, or 256 individual measurements, respectively. Statistically, the amount of random error in a measurement of this type is reduced by a

factor of $\frac{1}{\sqrt{n}}$, where n is the number of individual measurements. This means that a data point obtained using the HP 4145a will have inherently less random error than a single measurement taken using voltage sources and meters of comparable accuracy. All data used here was taken using the "medium integration time" (average of 16 measurements.)

(3) Polynomial interpolation was employed to smooth the data along each I_d vs V_{ds} curve using the following algorithm:

i) For each point I_i on a given I_d vs V_{ds} curve a least squares cubic polynomial was fit through five data points - the point of interest, plus the two points before it and after it.

ii) Using the polynomial thus obtained, a corrected value I_{a_i} for this data point is computed and recorded.

iii) Move to the next point I_{i+1} and begin again. The FORTRAN program *POLYSMOOTH* listed in Appendix B was written to perform this process. Using this technique the first two points and the last two points of each I_d vs V_{ds} curve remain unchanged.

This technique made a remarkable difference when applied to the data taken at Berkeley, but had almost no impact on the data taken using TECAP2. The TECAP2 data was used in all succeeding analysis.

IV. SPICE Model Parameter Extraction

A. Introduction

Comparison of the measured data with SPICE simulation proved difficult. First, all of the measured data was analyzed and a data set for one device was selected whose characteristics were typical of the group. The measured I_d vs V_{ds} plots for this device are shown in Figures 4 and 5.

From this data, sets of parameters for the MOS level 2 model were extracted. It was very important to obtain a set of optimal model parameters at this point, because errors in the model itself were sought, not errors in the choice of parameters. To accomplish this task, the TECAP2 system was again used, this time to perform parameter extraction using its built-in optimization routines. In order to use TECAP2 for this purpose, it was necessary to write a PASCAL subroutine implementing the SPICE level 2 MOS model in a form compatible with TECAP2. Optimal parameter extraction and model development using TECAP2 are discussed in the next chapter.

The procedure used was the following:

1. Optimal parameters were obtained.
2. I_d vs V_{ds} curves for $V_{ds} \approx V_{dsat}$ were generated.
3. Both measured and simulated data were differentiated to obtain conductance information.
4. The two were compared.

B. Experimental Procedure

The extraction of model parameters for the test device was performed by using the TECAP2 optimizer. Ideally, one would like to be able to find a single set of parameters which are suitable for use over all ranges of terminal voltages, but this was not feasible. Because this investigation was concerned primarily with dc current-voltage relationships and low frequency, small signal output conductance, the following SPICE parameters were optimized for: VTO, KP, NSUB, and LAMBDA. By adjusting these four parameters, the entire behavior of the dc transfer characteristics for the MOSFET as modeled by SPICE can be adjusted. All other parameters used in the SPICE model for this experiment are listed in Appendix C. The parameters for this device were obtained in the following manner:

1) Using the high voltage-high current curves as a basis for optimization, a set of parameters was chosen such that the error between the measured data points and the simulated values of I_d for I_d>1 mA was minimized. The result of simulation using these parameters is shown in Figure 6; the parameter values themselves are given in Table 1. These parameters were abandoned, however, because they proved to be inaccurate when used to model the behavior of the MOSFET for lower ranges of voltages and currents. Figure 7 shows how far in error the simulated results were compared with the measured data using these parameters over lower voltage ranges. The problem here is that the best choice of threshold voltage VTO for matching the level 2 model to higher voltage and current ranges is too low to give proper results at lower

voltages; the simulated drain current in this case is too high.

2) The high voltage-high current characteristics were again used, but this time the choice of model parameters was optimized for I_d<1 mA. The results are shown in Figure 8 and the parameters given in Table 2. When these model parameters were used for low voltage-low current simulation, the results were as shown in Figure 9.

3) Model parameters were optimized for the low voltage-low current region in question. The results are shown in Figure 10 and parameters are given in Table 3. These parameters were not used for the following reasons:

   a) The improvement in the accuracy of the model compared with results using the parameters obtained in part 2 was not significant.

   b) When these parameters were used in the model to simulate behavior at higher voltages, very wrong answers resulted due to the unnaturally high value for LAMBDA which was optimal for the lower regions. The result of applying these parameters to the high voltage region is shown in Figure 11.

The best set of general-purpose model parameters was the second set; these parameters were used in all succeeding SPICE simulations.

C. Comparison of Measured Data and SPICE Model Using Extracted Parameters

The I_d vs V_{ds} data shown in Figures 8 and 9 was differentiated to yield g_{ds} information. The result of differentiating the high voltage data is shown in Figure 12. As expected, SPICE does a reasonably good job of modeling the output conductance for this voltage and current range. Neglecting narrow width effects, the SPICE level 2 model equations for I_d are:

1) For $(V_{gs} > V_{th})$ and $V_{ds} < V_{dsat}$

$$I_d = \beta\left\{\left[V_{gs} - V_{bin} - \frac{V_{ds}}{2}\right]V_{ds} - \frac{2}{3}\gamma_s\left[(2\varphi_F + V_{ds} - V_{bs})^{\frac{3}{2}} - (2\varphi_F - V_{bs})^{\frac{3}{2}}\right]\right\}.$$

2) For $(V_{gs} > V_{th})$ and $V_{ds} > V_{dsat}$

$$I_d = \beta\left\{\left[V_{gs} - V_{bin} - \frac{V_{dsat}}{2}\right]V_{dsat} - \frac{2}{3}\gamma_s\left[(2\varphi_F + V_{dsat} - V_{bs})^{\frac{3}{2}} - (2\varphi_F - V_{bs})^{\frac{3}{2}}\right]\right\}.$$

where

$$\beta = \frac{W}{L(1-\lambda Vds)}\mu_s C_{ox}$$

By differentiating the above equations, the following expressions for g_{ds} are obtained:

1) For $(V_{gs} > V_{th})$ and $V_{ds} < V_{dsat}$

$$g_{ds} = \frac{\lambda Ids}{1-\lambda V_{ds}} + \beta\left[(V_{gs} - V_{bin} - V_{ds}) - \gamma_s\sqrt{2\varphi_F + V_{ds} - V_{bs}}\right]$$

2) For $(V_{gs} > V_{th})$ and $V_{ds} > V_{dsat}$

$$g_{ds} = \frac{\lambda I_{ds}}{1 - \lambda V_{ds}}$$

As shown in Figure 12, the measured g_{ds} data shows reasonably good agreement with that predicted by the model equations for the high-voltage, high current region; g_{ds} shows a negative linear dependence on V_{ds} for $V_{ds} < V_{dsat}$, and is relatively constant when the device is operating in the saturation region $(V_{ds} \geq V_{dsat})$.

In the region where $V_{ds} \approx V_{dsat}$ the situation is not well modeled, particularly for small $V_{gs} - V_{th}$. The result of differentiating the low voltage I_d vs V_{ds} data is shown in Figure 13. Inspection of these graphs shows that the SPICE model is clearly in error in several respects:

1) A negative linear dependence of g_{ds} on V_{ds} is predicted for $V_{ds} < V_{dsat}$, whereas the measured data shows more curvature. In fact, the measured data shows that g_{ds} is approximately inversely proportional to V_{ds}.

2) Predicted g_{ds} values for $V_{ds} = V_{dsat}$, V_{dsat} computed by SPICE, are factors of 3 to 12 lower than the measured values. This would result in very optimistic gain predictions from SPICE simulations for analog circuits biased in this region.

3) SPICE predicts values for V_{dsat} which are clearly too low for each curve shown; this results in SPICE predicting linear behavior where such behavior may not be present in the actual circuit.

To quantify this difference, the following procedure was employed:

a) The value of $g_{dsat} = g_{ds} \mid_{V_{ds} = V_{dsat}}$ was computed for each simulated g_{ds} vs V_{ds} curve.

b) The value of $V_{ds_{meas}}$ ($> V_{dsat}$) for which the measured value of g_{ds} equals g_{dsat} was found.

c) The difference between these two voltages was computed. This process is illustrated graphically in Figure 14.

A plot of ΔV_{dsat} vs V_{dsat} is given in Figure 15; the least squares line has the equation

$$\Delta V_{dsat} = a V_{ds} + b,$$

where $a = 0.065580859$ and $b = 0.110641748$.

V. Optimal Parameter Extraction Using TECAP2

The software organization of the TECAP2 system when used as a tool for parameter extraction can be thought of as shown in Figure 16. This system will iteratively solve for a set of model parameters for which the simulated results are optimally close to the measured result in the following way:

1) Using the most recent model parameters, the optimizer invokes the simulator using the user-selected model to obtain a set of simulated transistor characteristics.

2) The optimizer then compares the simulated results with the measured data and computes the relative error between the two.

3) If the error between the simulated and measured data is not significantly better than the error in the previous iteration (greater than a user-defined tolerance) the algorithm quits; otherwise the model parameters the user has specified are changed using the Levenberg-Marquardt algorithm and the process is repeated.

In this way, the optimizer changes the parameters until no further improvement is possible. For a further discussion of the optimizer, see section B.

A. Practical Considerations

At present TECAP2 is only available at Berkeley on an HP 9836 desktop computer belonging to the BIAS research group. Since no HP 4145a was available at the time of this writing, TECAP2 at Berkeley can only be used to process data taken using other TECAP2 systems and for

device simulation and model development. In this discussion a basic familiarity with the 9836 computer and the PASCAL 2.0 operating system is assumed.

In order to run TECAP2, the 9836 computer must be equipped with the PASCAL 2.0 operating system and must have available a minimum of 1 Megabyte of random access memory. In addition, the computer must be enhanced by the addition of some sort of external mass storage device, since all of TECAP2's executable code cannot be stored on a standard $5\frac{1}{4}$" floppy disk. At Berkeley this is accomplished with an HP 7912 hard disk belonging to the BIAS group; all of the relevant software is stored on system volume 33.

To run TECAP2 on the 9836, the user must first execute the file TECAP2.CODE. TECAP2 will then start up and prompt the user for a command. TECAP2 uses a menu driven user interface. The user inputs a series of instructions selected from the currently displayed menu as prompted by the program. The main menu is shown in Figure 17.

A general discussion of TECAP2's abilities is best left to the *TECAP2 Reference Manual*[6]; the intent of this section is to guide the user through the single process of obtaining a set of optimal model parameters for an I_{ds} vs V_d data set which is available on disk. To accomplish this task, the following steps should be taken:

1) **Read in the measured data set.** To do this, first press <M> to get the measurements menu. The screen will now show another menu with a selection of measurement related options. Press <4> and then <enter>; you will then be prompted for the name of the data file you wish to read in. A complete list of measured data files for devices characterized for this project is given in Appendix A. Once

the data is read in, it may be graphed or printed out by pressing <M7> or <M9> respectively, and then <enter>. As the measured data file is read in, measurement information such as voltage ranges and step sizes is also read, as is device data such as device type and geometry; this information was specified when the measurements were taken. For the purpose of parameter extraction in this section this is sufficient. Changing the measurement/simulation voltage step sizes and output variables is discussed in the next chapter.

2) **Select the model.** Since TECAP2 is, after all, an Hewlett Packard product, the default transistor model is the HPSPICE MOSFET model. To change from the default model to the SPICE Level 2 model which has been implemented in this project, press <E> (for extract), then <1> and <enter>. You will be asked to select one of 6 models by number; the correct answer is 4. The sub-menu which is displayed whe <E> is pressed is shown in Figure 18.

3) **Set initial parameter values and decide which ones are to be optimized for.** By pressing <E2> <enter>, the user is presented with a table listing all model parameters, their present values, minimum and maximum values(for optimization), and a flag "P" for the optimizer program. An example of this table is shown in Figure 19. To change the value of any number in the table, simply use the cursor control keys to position the cursor to the beginning ot the value to be changed, type the new value, and press <enter>.

The numbers under Value are the current values of the parameters; in this case they are at their default values. The numbers under Minimum and Maximum are hard constraints for

the optimizer; during the optimization process the optimizer will vary specified parameters as it tries to minimize the error between measured and simulated results. The minimum and maximum values are limits beyond which the optimizer is not allowed to change a parameter. The value of "P" for each parameter is a flag which tells the optimizer whether or not to optimize that particular parameter; 1 means yes, 0 means no. For this project, KP, VTO, NSUB and LAMBDA were optimized.

The careful user will note that not all parameters for the Level 2 SPICE model are present in this parameter table. This is because not all effects modeled in the Level 2 SPICE model are included in the TECAP2 version. Specifically, the following effects are not included:

a) Subthreshold current.

b) Velocity saturation.

c) Narrow channel effects.

d) Thin oxide capacitance model.

The Level 2 MOS model in TECAP2 is discussed more fully in the next chapter.

4. **Set optimizer options.** By pressing <E3> <enter>, the user may set various options for the optimizer. The most important options to set are the upper and lower limits for optimization. These options govern the range on the y (I_d) axis of the graph over which the optimizer is to try and match the measured and simulated curves. For this project, the parameters which were finally used were obtained by optimizing over the 1nA to 1mA range on the I_d

vs V_{ds} hi voltage curves. It is a good idea to limit the range of optimization for two main reasons:

a) Limiting the range over which the optimizer must work reduces the number of data points which must be computed by the simulator and compared with measured values by the optimizer. Using too many points slows down considerably this already time consuming process, and may cause memory overflow problems.

b) If too wide a range of data is considered, or too many points are considered, mysterious system execution errors may result. TECAP2 recovers gracefully from such errors (generally divide by zero), but no parameters are obtained.

Other options may be set, but are best left at default values for now.

5) **Optimize.** Pressing <E> <enter> initiates the optimization process. The optimizer iterates to a solution, providing the user with intermediate results as it goes. Once it is finished, the parameters are stored internally unless the user specifies that he does not want to keep them. To see what the simulated device characteristics look like, press <S1S7> (S is for simulate), and <enter>. To plot the simulated results using a dashed line on the same graph as the measured data, press <M7S1P6S8> and <enter>.

The new parameters can be inspected by pressing <E2> <enter> as before. However, if a comparison with SPICE is desired, the user should instead get a parameter listing by pressing <E9> <enter>. This is because the parameters shown when <E2> is pressed are truncated to 4 significant digits in order to fit in the table. The listing given by

pressing <E9> gives all values to their full precision as stored in the 9836 computer; these parameters should be specified to their full precision on the .MODEL card in the SPICE input deck if it is desired that SPICE agree exactly with TECAP2.

B. The TECAP2 Optimizer[5]

The optimization package used in TECAP2 implements the Levenberg-Marquardt algorithm. This algorithm combines the method of steepest descent and the Gauss-Newton method as described below:

Given a real valued function $f(x)$: $R^n \rightarrow R$, where $x \varepsilon R^n$ is the vector of model parameters, n is the number of parameters to be optimized, and $f(x)$, the cost function to be minimized, is the sum of the squares of the difference between measured and simulated values of the specified output variables (I_d in our case) normalized to the measured variable, i.e.:

$$f(x)=\sum\left[\frac{I_{sim}(x,V_d,V_g)-I_{meas}(x,V_d,V_g)}{I_{meas}(x,V_d,V_g)}\right]^2$$

The problem is to find the vector x_{min}, i.e. a set of model parameters, which minimizes $f(x)$.

Method of Steepest Descent:

The method of steepest descent is an iterative method for finding x_{min}, the minimizer of the function $f(x)$. The method works by taking the gradient of the function, ∇f. The gradient points in the direction of greatest increase of the function f; the steepest descent method moves in opposite direction so that a new value of x is defined by

$$x_{k+1} = x_k - \alpha_k \nabla f_k$$

where ∇f_k is the value of the gradient at the point x_k, and $\alpha_k \geq 0$ is a scalar which determines the size of the step taken in the direction of steepest descent at the k_{th} iteration.

The three main problems with this method are:

1) Selection of a good starting point x_0. Hopefully the user will monitor the optimization process and make sure that the initial selection of parameters does not lead to an erroneous local minimum.

2) The gradient ∇f_k must be evaluated at each step. TECAP2 performs this operation using the forward difference approximation

$$\nabla f_k = \frac{f(x_k + \delta x) - f(x_k)}{\delta x}$$

This requires n+1 evaluations of the function f at each iteration point x_k. To reduce computation time, TECAP2 does not calculate the gradient at each iteration point. Instead, the gradient computed at iteration k is modified using a technique called Broyden's rank one correction for use as an approximation to the gradient in succeeding iterations. The actual gradient is recomputed after every n iterations or when λ, the marquardt parameter (described below), is increased.

Newton's Method:

Newton's method works on the assumption that in a neighborhood around x_{min}, the minimizer of the function f, $f(x)$ is well approximated by a truncated Taylor series:

$$f(x_k+\delta x_k) = f(x_k) + (\delta x_k)^T \nabla f_k + (\delta x_k)^T H_k \delta x_k$$

where H_k is Hessian or second gradient of f at the point x_k. It can be shown that the value of δx_k which minimizes the above expression is

$$\delta x_k = -\nabla f_k^T H_k^{-1}.$$

Thus,

$$x_{k+1} = x_k - \nabla f_k^T H_k^{-1}.$$

The Gauss-Newton method is a modified form of Newton's method in which the Hessian H_k is approximated by

$$H_k = 2\nabla f^T \nabla f,$$

where ∇f^T is the transpose of the gradient of f.

Levenberg - Marquardt Method:

The Levenberg - Marquardt method combines the two above approaches by generating a sequence of points $\{x_k\}$ converging to x_{min} in the following way:

$$x_{k+1} = x_k + P_k,$$

where

$$P_k = -(\nabla f^T \nabla f + \lambda I)^{-1} \nabla f^T f(x_k),$$

I is the identity matrix, and λ is the marquardt parameter, chosen from a sequence of non-negative real constants. For λ large relative to $|\nabla f^T \nabla f|$,

$$P_x \approx -\left[\frac{f(x_k)}{\lambda}\right] \nabla f_k.$$

and the algorithm behaves like the steepest descent algorithm with step size $\alpha_k = \frac{f(x_k)}{\lambda}$. During initial iterations when the error is large, λ is chosen large so that the method steepest descent, giving large improvement. In successive iterations, λ is reduced so that the method looks like the Gauss-Newton method. This gives nearly second order convergence near the minimum.

VI. Model Development Using TECAP2

Because TECAP2 is a Hewlett-Packard product, its standard model selection corresponds to the models available to users of HPSPICE. However, it is possible for users to install their own models, as was done with the SPICE level 2 MOS model for this project. This capability was intended to allow users to extract parameters for their own models. However, the fact that the TECAP2 has its own built-in circuit simulator capable of calling a user defined model routine makes it an ideal tool for developing new models and trying out changes in old ones; it is considerably easier to make a change in a TECAP2 model subroutine and see how it affects model performance than it is to make a similar change in SPICE 2G.6.

The process of installing a new MOS model in TECAP2 is described below. For additional information the interested reader is directed to the *TECAP2 System Designer's Manual* [5].

A complete listing of all program files relevant to TECAP2 is given in Appendix D. The main files of interest when installing a new model in TECAP2 are T_LIB2.CODE, T_TCP2S.CODE, STARTUP.M, T_USER.TEXT, which is also called the User Module, and LINK.TEXT. The only PASCAL source code available to the user is T_USER.TEXT; it is in this file that the user may install his own transistor model as described below.

Once a model subroutine has been written, the user then makes his own version of TECAP2, including the new model, in the following manner:

1) Compile the (modified) PASCAL source file T_USER.TEXT. The resulting code file must be named T_USER.CODE, which is the default name.

2) Stream the text file LINK.TEXT. This file invokes a set of system commands that link all of the compiled subroutines in the two object code libraries and the newly compiled User Module into a new version of TECAP2.CODE. Once this is done, TECAP2 is ready to be executed.

Writing the Model Subroutine

The TECAP2 User Module T_USER.TEXT is listed in Appendix E and will be referred to throughout the remainder of this section.

TECAP2 is set up to implement a total of 6 different models; models 1, 2, and 6 are HPSPICE models. Models 3, 4, and 5 are available for use by the user. Model 3 is a simple, classical MOS model intended to serve as an example for users wishing to install their own model. The MOS level 2 model written for this project resides in model 4 of the listing in Appendix E.

To install a new model, the user may either (1) modify one of the existing models, or (2) start from scratch. Modifying an existing model is generally much easier than starting completely from scratch. When modifying the User Module, none of the procedure declarations preceding the IMPLEMENT statement should be changed, as these declarations allow the model subroutines to interact with other routines in the program and access common data. Procedures D10-D14 and C10-C14 are included to allow the user to add commands to TECAP2 and will not be discussed here. The *set_constants* procedure can be left as is or changed as needed.

Referring again to the User Module, the procedure *model_three*, which implements the classical MOSFET model, follows *set_constants*.

It will be used here to illustrate how one goes about writing a model subroutine. First of all, the parameter list for the model procedure call is given; these are already in place in the User Module for models 3, 4, and 5 and should not be changed by the user. The purpose of each of the parameters passed to and from the procedure is fairly well explained by the comments. p is a data structure which includes all model parameter data. The initial VAR declaration should include all model parameters, both external and internal, as well as variables to be used for terminal voltages and currents.

The procedure *model_info* is used to set the model parameter names and their default values. The subprocedure *set_data* should be left exactly as given in the User Module. This procedure is called to set the model parameter names, default values, default minimum and maximum values (for the optimizer), and the units for the parameters; all of the information in the parameter table TECAP2 displays when <E2> <enter> is pressed appears exactly as given to the procedure *set_data*. When each of the external parameters has been set, the total number of external parameters should be stored in the variable *p.number*. External ac parameters such as CJO or MJSW in the level 2 model should be set next in the same manner, with the number of external ac parameters stored in the variable *p.acnumber*. Next, set the names of all internal parameters-- variable names which which will actually be used in model equations-- and record the number of internal parameters in the variables *p.internal* and *p.acinternal* as before. Be sure to increment the value of the pointer n as shown in the example so that no parameter information is overwritten. Notice that no ac parameters are specified in *model_three*.

Procedure *get_ext_par* is used to assign values from the external parameter table to the internal parameter variables; when writing or modifying this procedure be sure to match the internal parameters with the correct value. Device parameters should also be assigned to internal parameters at this time; specifically the assignments *xl:=dev.l* and *xw:=dev.w* must be made to get gate length and width information. Source and drain areas and perimeters for ac calculations are also assigned in this way, as is done in the Level 2 MOS model. Node assignments for the simulator are also made in this procedure. The device nodes are specified in the model by assigning appropriate locations to each node as shown in the diagram below:

```
        NN
        MN
WW MW   ME EE
        MS
        SS
```

TECAP2 allows the terminal resistances (Rs, Rd) to be included by using two sets of nodes--internal and external. External node locations are set to NN, WW, SS, and EE, while the internal node locations corresponding to terminals of an ideal device are assigned to MN, MW, MS, and ME. Unused nodes are set to XX; the reader is referred to model 4 (level 2 model) for an example.

The procedure *get_int_par* gets the intermediate parameters from the data structure as shown in the example.

Procedure *calculate_init* calculates the intermediate model parameters and stores them in the data structure. Again, care must be taken to match the correct parameter with its corresponding location in the data structure p. Procedure *calculate_id* is the main procedure which distinguishes one model from another, because this

where the equations governing drain current as a function of terminal voltages and model parameters are implemented. In this procedure equations for device currents and and derivatives are implemented. As is shown in the example, the derivatives need only be calculated when the flag *dflag* is true; this flag is set by the simulator when Rs and Rd are non-zero and iteration must be performed to simulate the device.

Procedure *device* calculates device related data such as mode of operation and determining whether or not a junction is forward biased. In addition, all capacitances and node charges should be calculated in this procedure. This procedure should be the calling routine for *calculate_id*. The current and charge vectors are also loaded in this procedure; the subscripts correspond to the device nodes specified in procedure *get_ext_par*. In addition, the conductance and susceptance matrices are loaded here as shown in the example.

Now that all necessary subprocedures have been defined, the main procedure of the model can be executed. The main procedure should have the following format:

```
BEGIN   {main part of the model}
IF infoflag THEN model_info
        ELSE BEGIN
           IF initflag THEN BEGIN
                        get_ext_par;
                        calculate_init;
                        END
                   ELSE BEGIN
                        get_int_par;
                        {Calculate values of voltage }
                        {variables needed for model  }
                        {equations from the voltages }
                        {in the vpin vector.         }
                        device;
                        END;
           END;
END; {end of model}
```

In the main routine it should be remembered that all of the voltages in the vector *vpin* are measured with respect to an external ground--even the body of the MOSFET. This means that such quantities as V_{ds} and V_{sb} must be calculated from the values in the voltage vector if they are desired. Do not forget to define the voltage across Rd and Rs as is done with the variables *vdxd* and *vsxs* in the example.

Other Remarks

As was mentioned earlier, the most efficient way to implement a new model is to modify the equations of an existing one, using as many of the old variable names as possible. The level 2 MOS model used for this project was written by changing some of the model equations in the HPSPICE model. This will simplify such things as loading the conductance and susceptance matrices. In particular, it is advisable to assign the device nodes as is done in the MOS level 2 model, because this configuration corresponds to the default configuration the system when measurements of MOSFETs are made. The sense of this orientation may be changed by the user, but this will not be discussed here for the sake of simplicity.

B. Testing the New Model

The new model in the User Module should be compiled and linked as discussed earlier. Once this is done, the new version TECAP2 can be executed. The model can be tested as follows:

1) **Select the model.** The model is selected by pressing <E1> <enter>, then selecting the number of the model to be tested.

2) **Select model parameters.** By pressing <E2> <enter>, the model parameter table for the new model is displayed. All parameter names and default values are set according to the procedure *model_info* in the User Module. The values of the parameters may be changed from their default values by moving the cursor to the appropriate position in the table with the cursor control keys and entering new values.

3) **Set up the simulation conditions.** The conditions governing measurements and device simulations are specified in the current Setup table. Such things as body bias, ranges for drain and gate voltage sweeps and the choice of output variables are specified here. There are two default setup tables provided by TECAP2; these may be accessed by pressing <U#> <enter>, where U stands for Use setup and # is the number of the desired setup table. The default table may be changed or a new one created by pressing <U#B> <enter>, where U and # have the same meanings as before and B stands for build setup.

4) **Run the Simulation.** After selecting a setup table the device may be simulated by pressing <S1><enter>. The results may be viewed graphically by pressing <S7> <enter>, or the numerical results viewed by pressing <S9> <enter>. All parameter values, including internal parameters, may be viewed by pressing <E9> <enter>– this is useful in debugging. Numerical output may be directed to the printer by pressing <O6> prior to one of the other commands. Other output control options are explained by the menu displayed when <O> is pressed.

Once the model is fully tested and debugged, it may be used with the optimizer and a measured data set as described in the section on Optimal Parameter Extraction. At this time it is not possible to test the ac sections of any model with TECAP2 because ac simulation and measurement has not yet been implemented in TECAP2. Until this feature becomes available, there is probably no reason to include ac parameters in the model procedure or to calculate susceptance parameters.

VII. Conclusion

The output conductance of MOS transistors has been measured and compared with the output conductance predicted by the SPICE Level 2 model. The model has been found to be in error in the region near the saturation point where $V_d \approx V_{ds}$ for small values of $V_{gs} - V_{th}$. In general, SPICE predicts a value for g_{ds} in this region which is too low, resulting in optimistic predictions when MOS circuits are simulated. Specifically:

1) SPICE predicts a value of V_{dsat} which is too low when compared with the measured data.

2) SPICE does not show the correct functional relationship between g_{ds} and V_{ds} for $V_{ds} < V_{dsat}$.

To obtain optimal model parameters for the simulations performed, a subroutine implementing the SPICE Level 2 MOS model has been written for the TECAP2 parameter optimizer. The fact that velocity saturation effects were not included in the TECAP2 version of the model proved to be of no consequence because inclusion of this effect would have caused an even lower value of V_{dsat} to be predicted by the model.

In addition, the use of the TECAP2 system for optimal parameter extraction and for model development has been discussed.

In the future, the TECAP2 system could be used to test new models for SPICE and to modify existing ones. For example, the Frohman-Grove [7] model for channel length modulation could easily be implemented in the SPICE Level 2 model now running on the TECAP2 system in the following manner:

In the model subroutine, if LAMBDA = 0, the effective channel length is calculated using the equation

$$\Delta L = X_D \left[\frac{V_{ds} - V_{dsat}}{4} + \sqrt{1 + \left[\frac{V_{ds} - V_{dsat}}{4} \right]^2} \right]^{\frac{1}{2}}$$

Instead of this equation, the Frohman-Grove formulation could be used:

$$\frac{1}{\Delta L} = \frac{1}{X_D \sqrt{V_{ds} - V_{dsat}}} + \frac{C_{ox}}{\varepsilon_{si}} \frac{ALPHA(V_{ds} - V_{gs}) + BETA(V_{gs} - V_{dsat})}{(V_{ds} - V_{dsat})}$$

This implementation would require the introduction of 2 new external parameters, ALPHA and BETA, but then these parameters would be available for optimization by the TECAP2 system.

Until a new model for output conductance is implemented in SPICE, the problems discussed in this paper will continue to be present. At this time, a stop-gap solution to the problem is to model MOS transistors which are biased such that $V_{ds} \approx V_{dsat}$ by specifying a value for the parameter LAMBDA which is artificially high. This will have the effect of increasing the output conductance predicted by SPICE to a level which is closer to reality for transistors biased on the edge of saturation, but which is grossly in error for conditions where $V_{ds} \gg V_{dsat}$.

SPICE PARAMETERS		
Parameter	Value	Unit
KP	38.61	$\mu A/V^2$
VTO	0.4374	Volts
NSUB	1.021E16	Cm^{-3}
LAMBDA	.01477	V^{-1}

Table 1: Level 2 MOS Model Parameters Obtained by Optimizing for $I_d > 1mA$

SPICE PARAMETERS		
Parameter	Value	Unit
KP	43.64	$\mu A/V^2$
VTO	0.7613	Volts
NSUB	2.209E15	Cm^{-3}
LAMBDA	.01646	V^{-1}

Table 2: Level 2 MOS Model Parameters Obtained by Optimizing for $I_d < 1mA$

SPICE PARAMETERS		
Parameter	Value	Unit
KP	31.01	$\mu A/V^2$
VTO	0.6881	Volts
NSUB	1.841E16	Cm^{-3}
LAMBDA	.09524	V^{-1}

Table 3: Level 2 MOS model parameters obtained by optimizing for Low Voltage/Current Ranges.

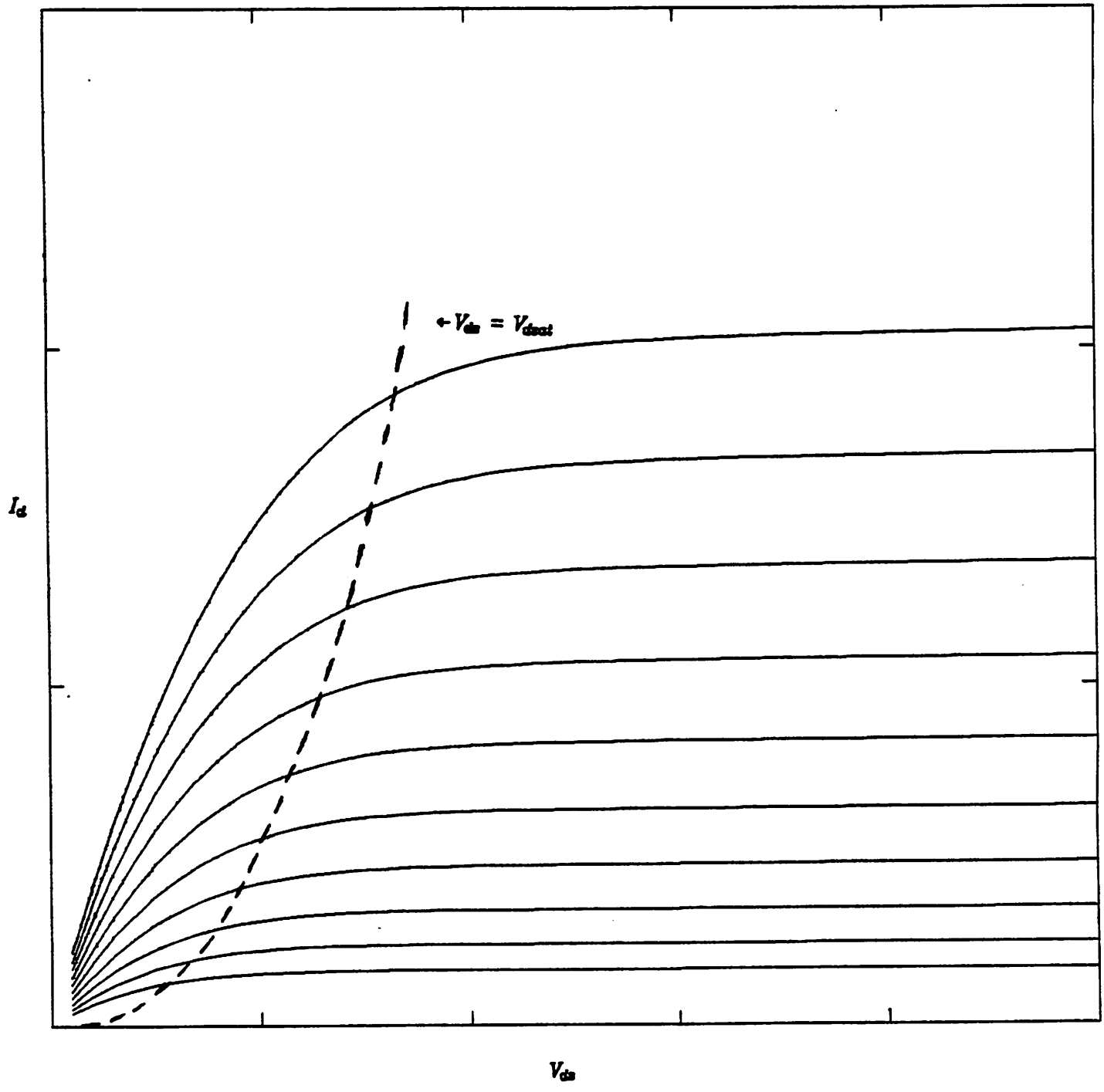

Figure 2: Typical I_d *vs* V_{ds} Characteristic

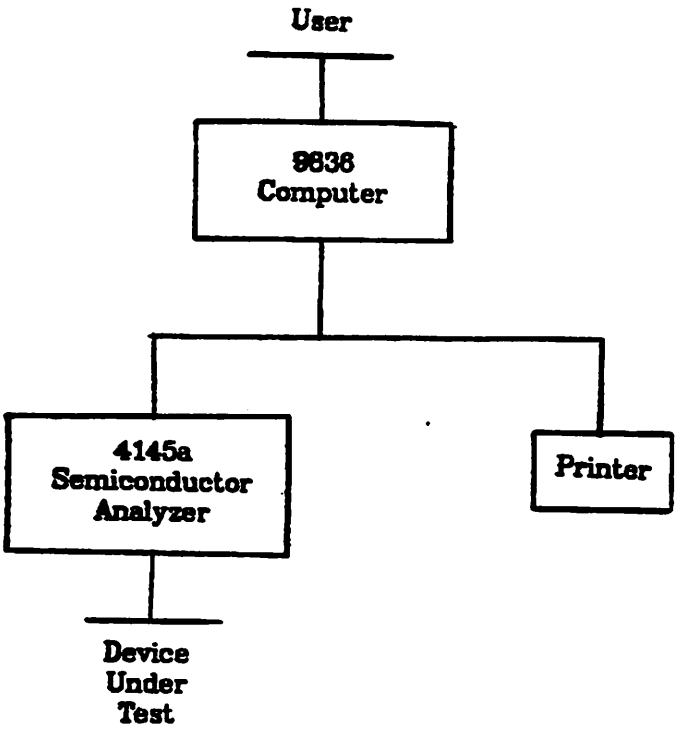

Figure 3: TECAP2 Hardware Setup

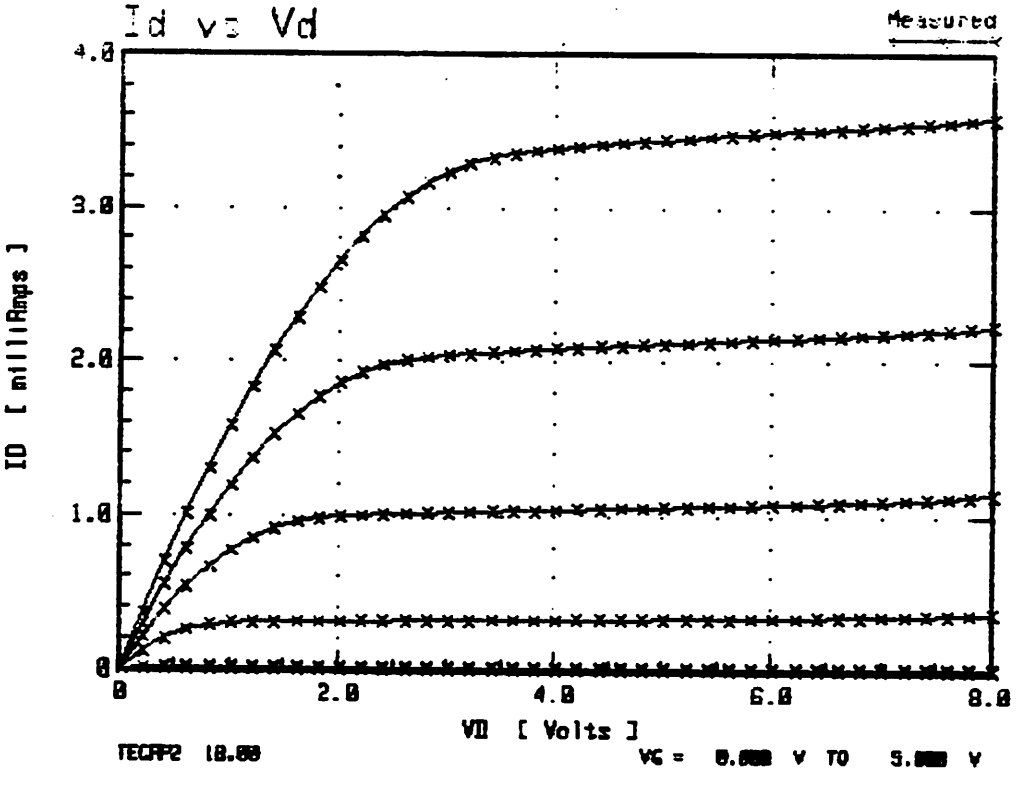

Figure 4: High Voltage/Current I_d-V_{ds} Characteristic

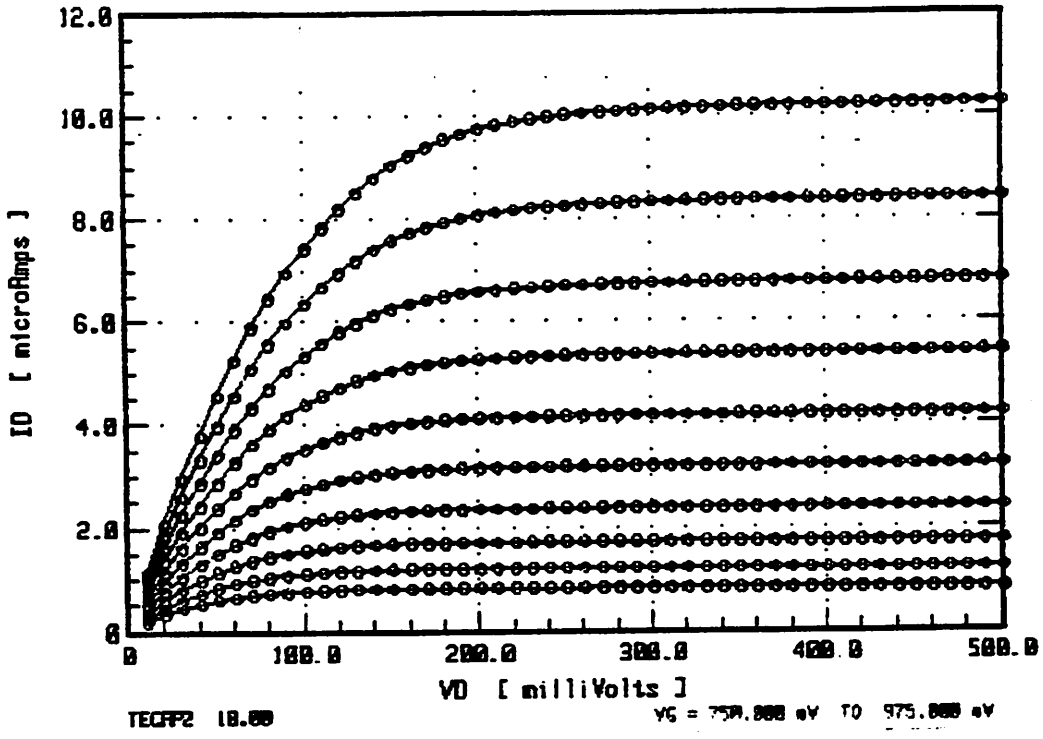

Figure 5a: First Half of Low Voltage/Current I_d-V_{ds} Characteristic

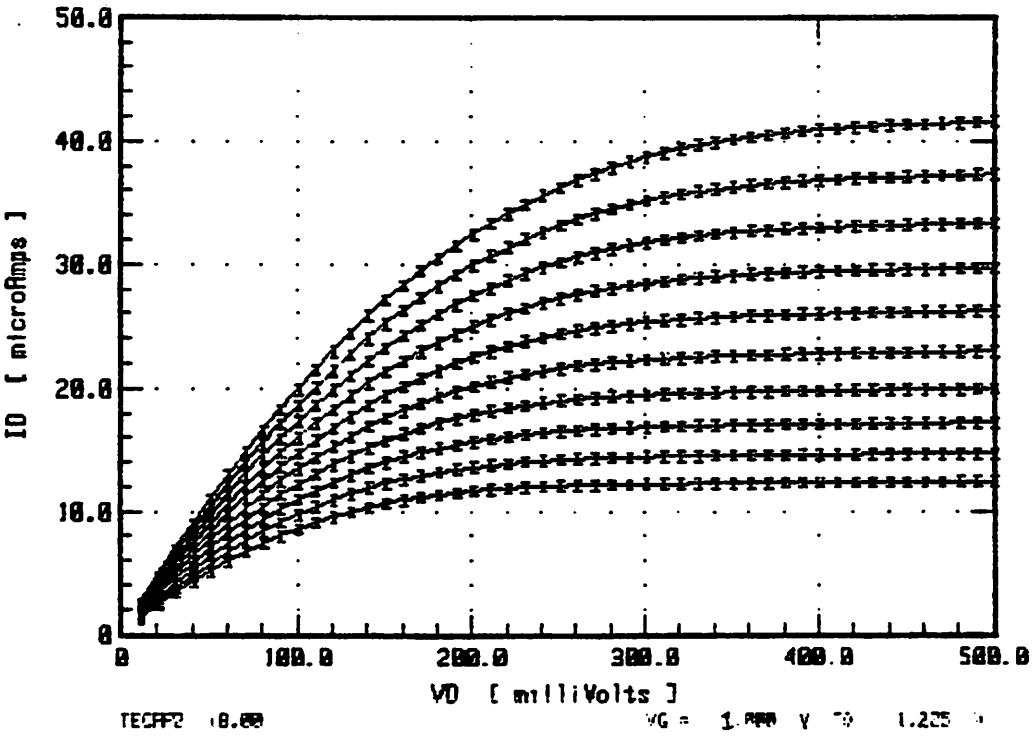

Figure 5b: Second Half of Low Voltage/Current I_d-V_{ds} Characteristic

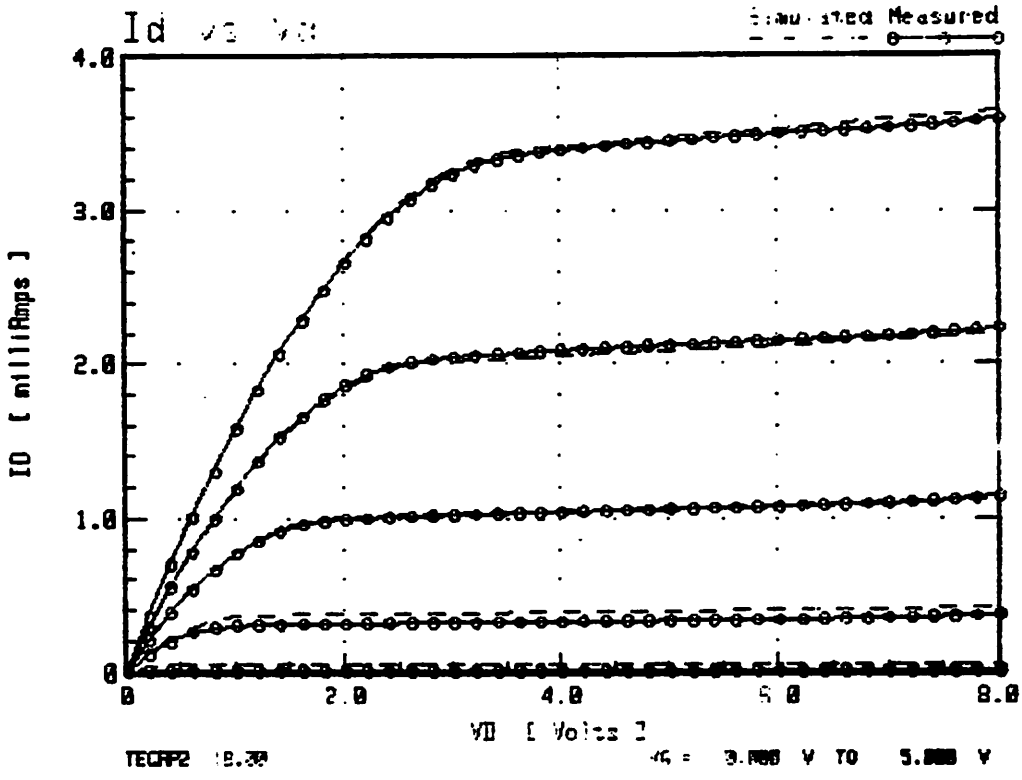

Figure 6: Measured and Simulated High Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for I_d > 1mA.

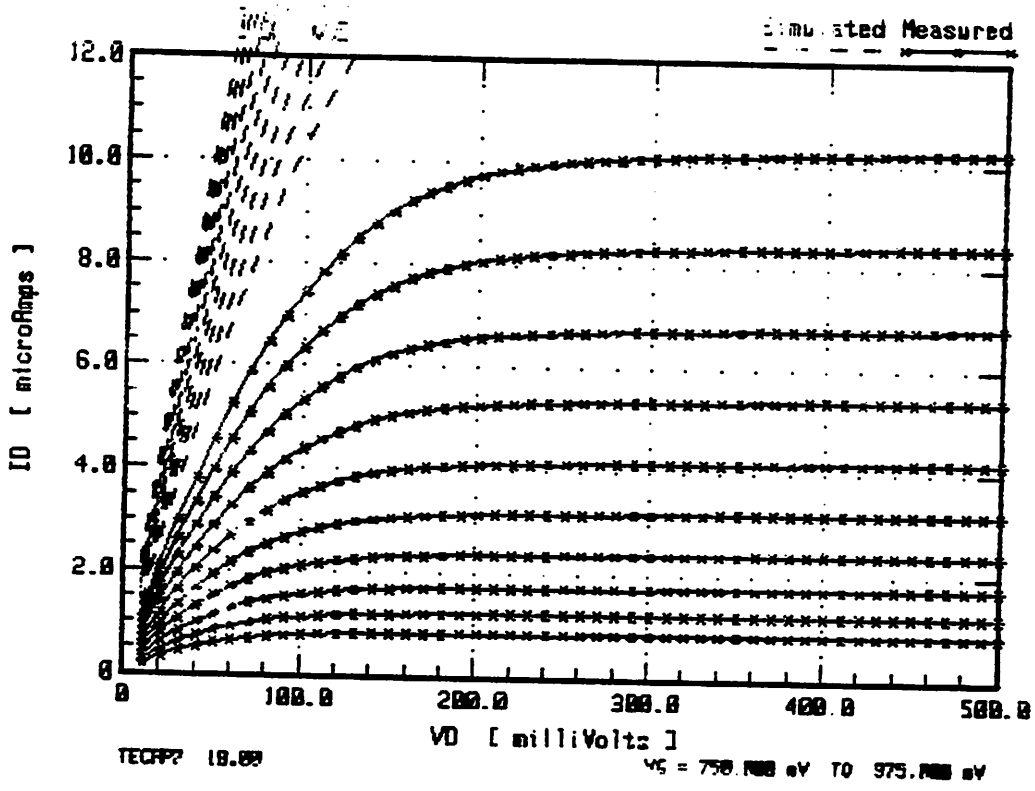

Figure 7: Measured and Simulated Low Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for I_d > 1mA.

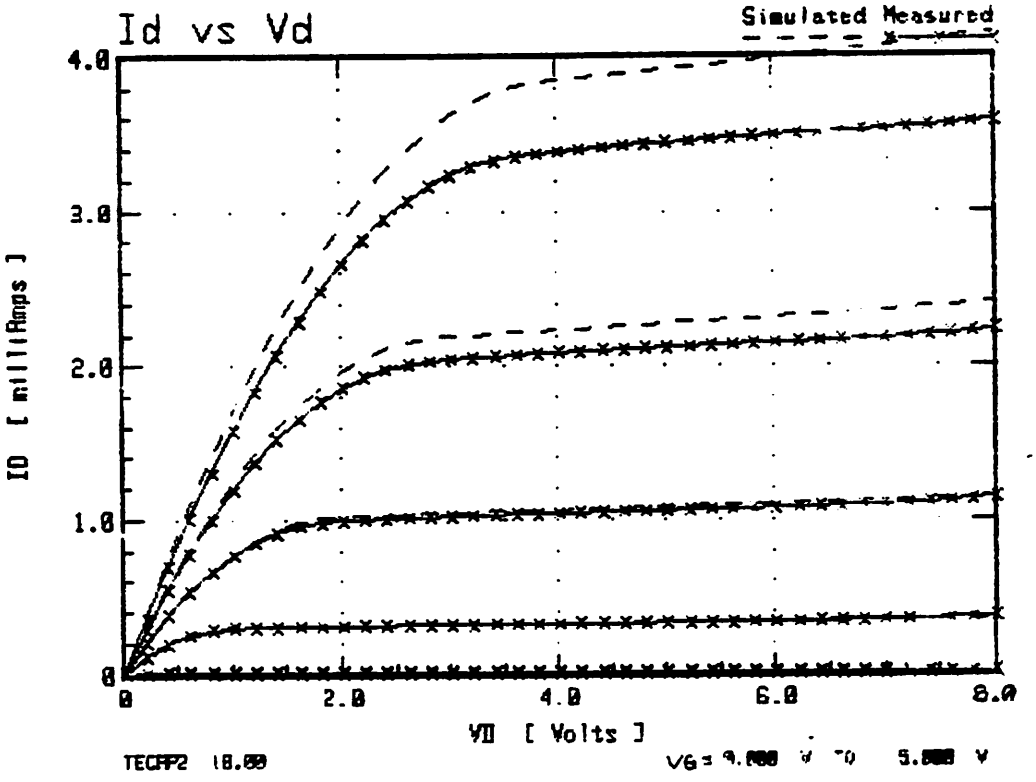

Figure 8: Measured and Simulated High Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for I_g < 1mA.

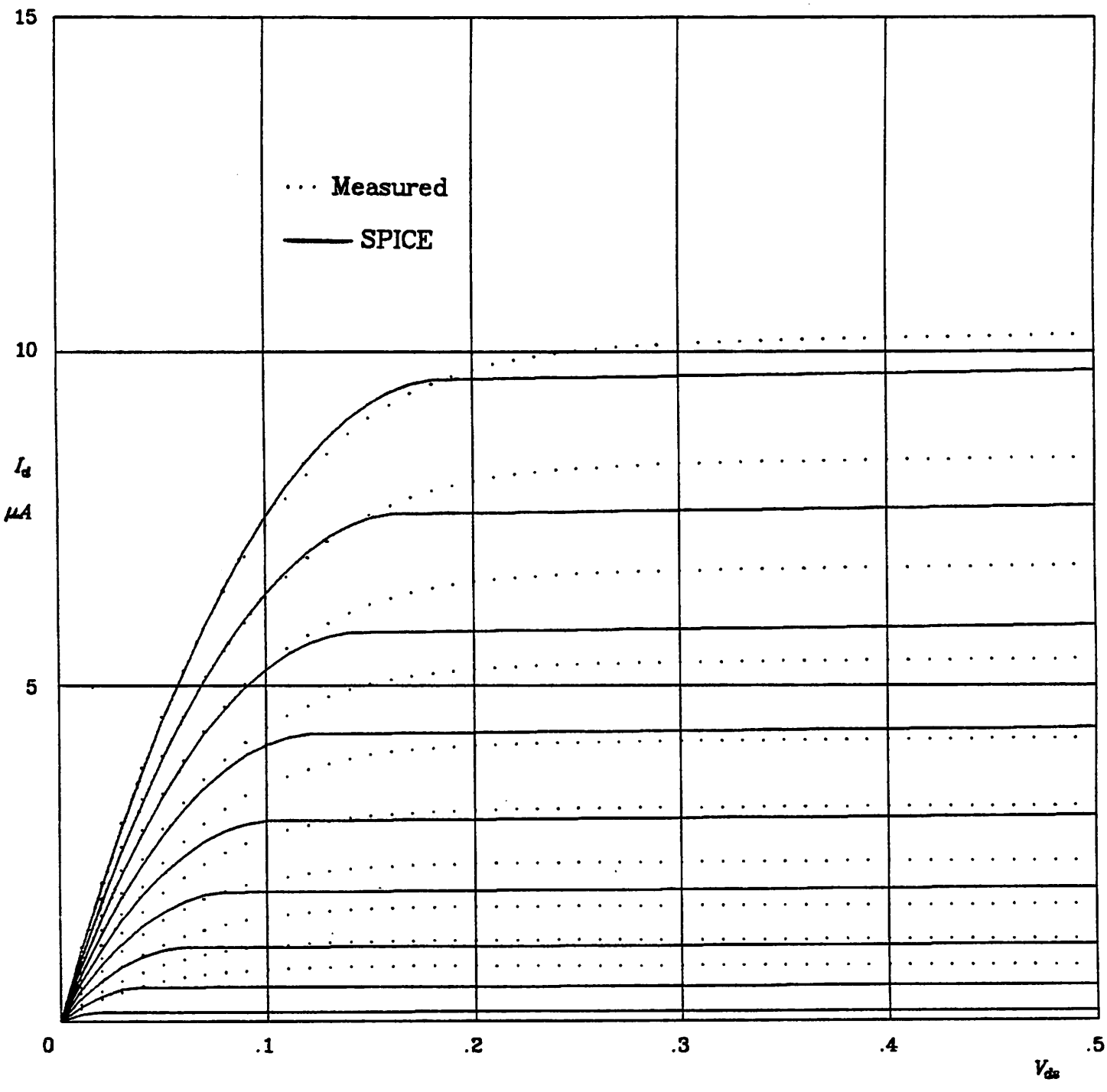

Figure 9a: Measured and Simulated Low Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for I_d < 1mA. V_{gs} from 0.75V to 0.975V.

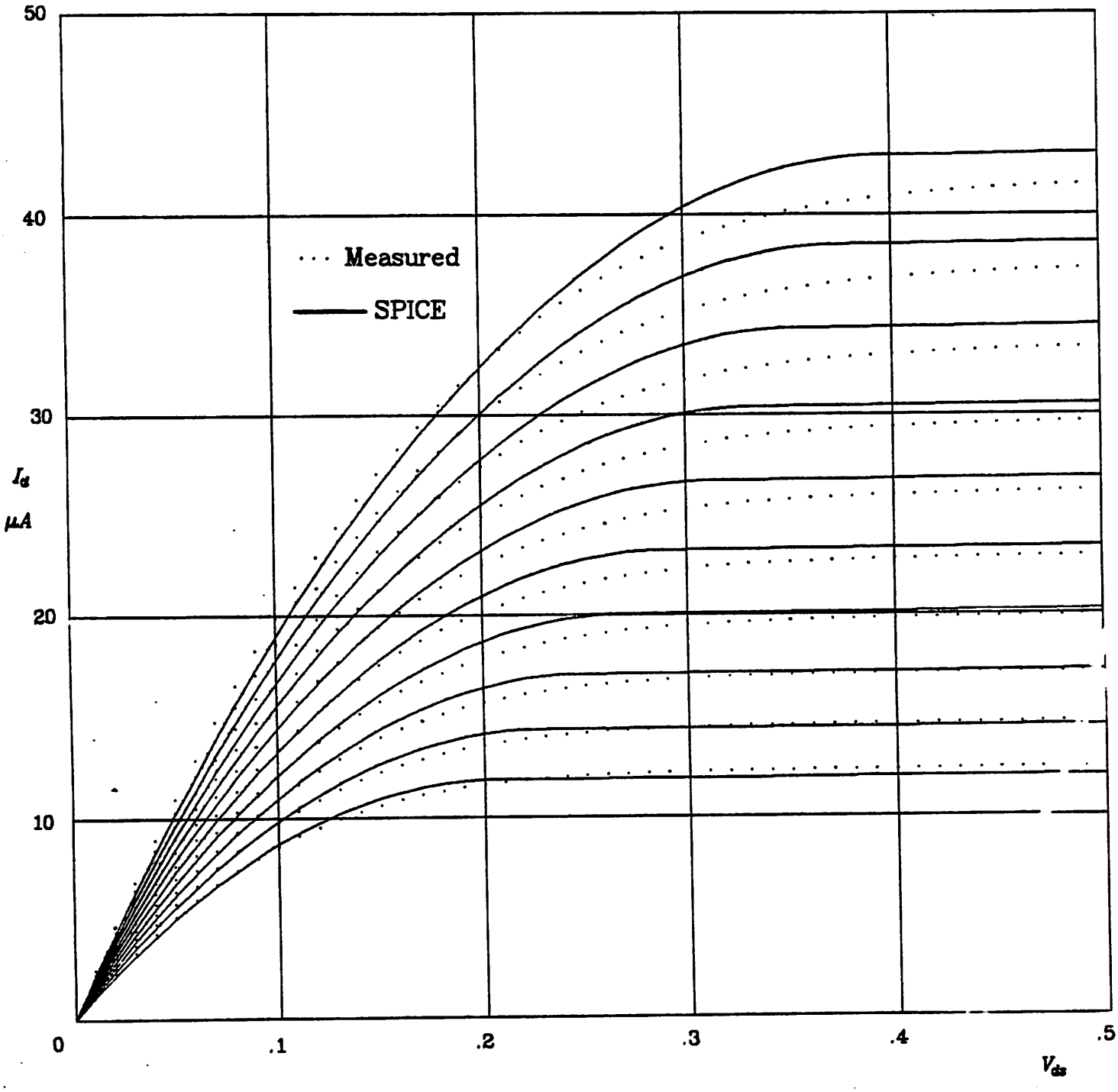

Figure 9b: Measured and Simulated Low Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for I_d < 1mA. V_{gs} from 1.00V to 1.225V.

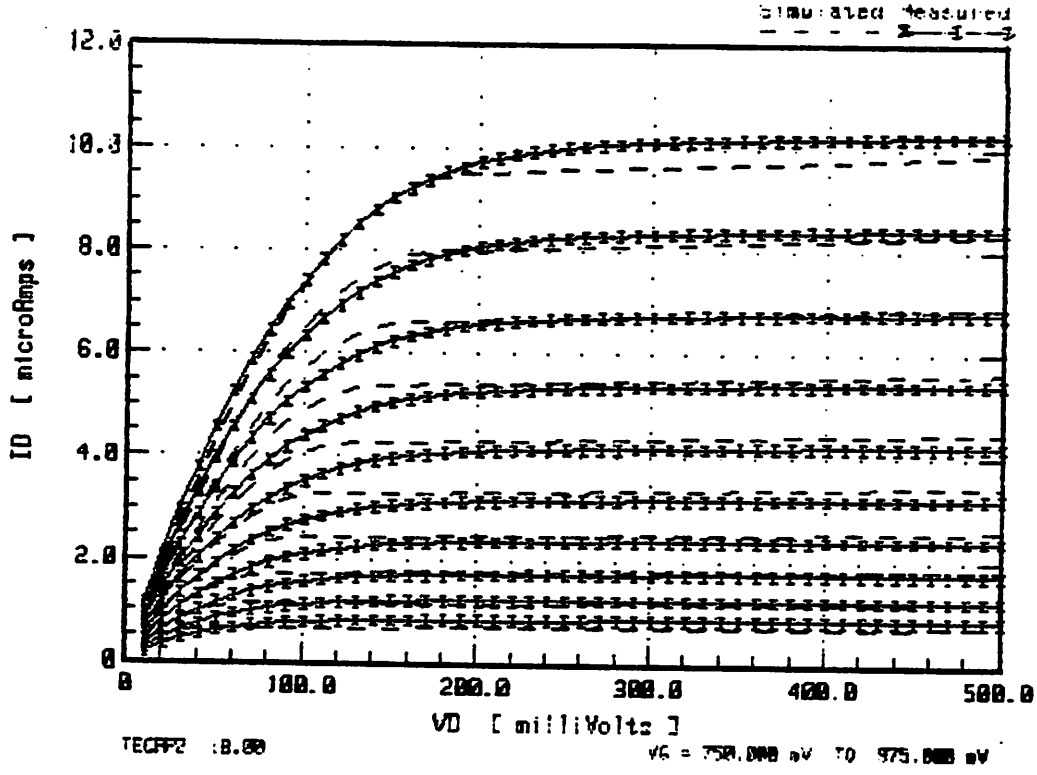

Figure 10: Measured and Simulated Low Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for the Data Shown.

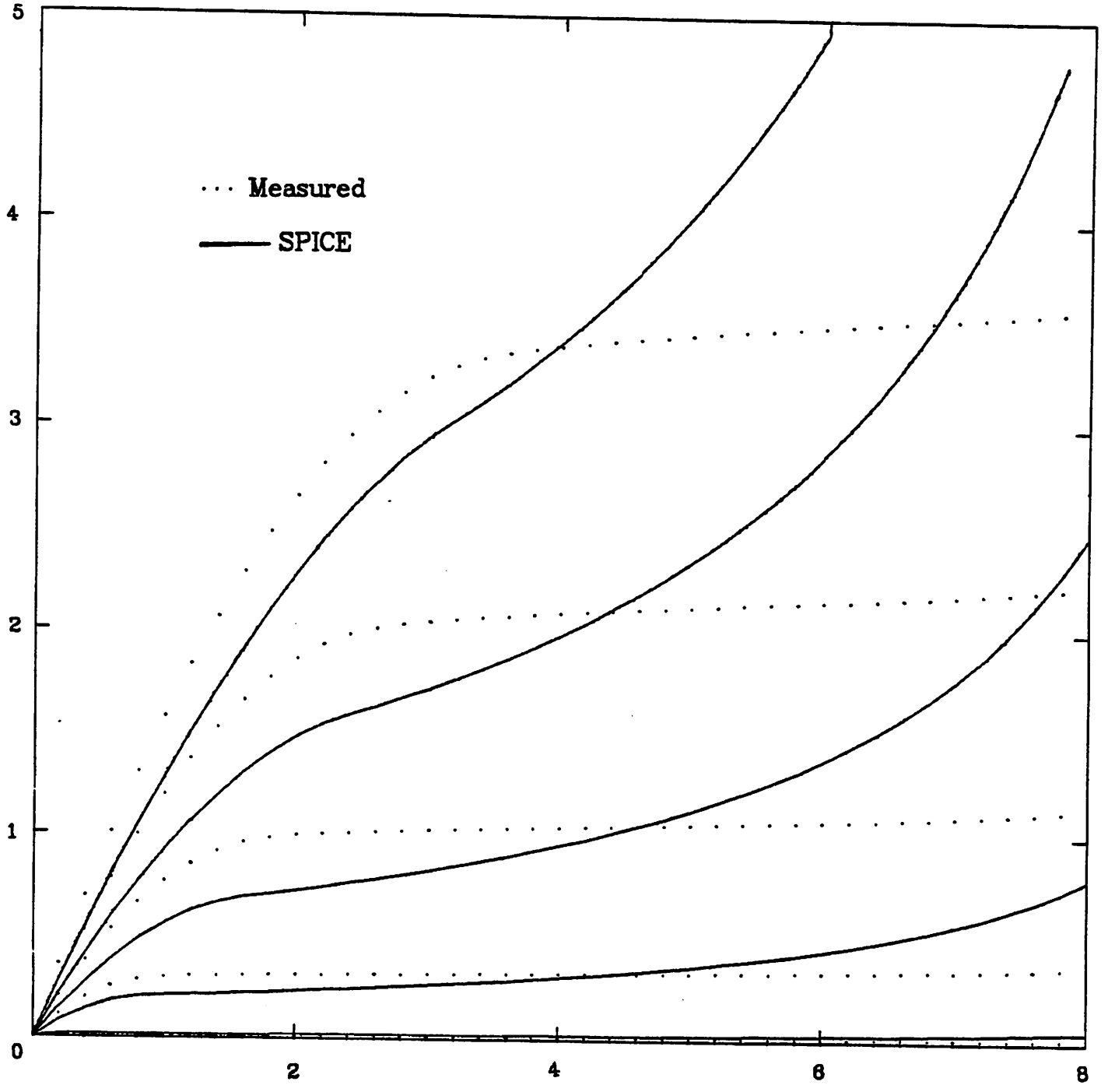

Figure 11: Measured and Simulated High Voltage I_d-V_{ds} Characteristics Using Parameters Optimized for Low Voltage/Current Ranges.

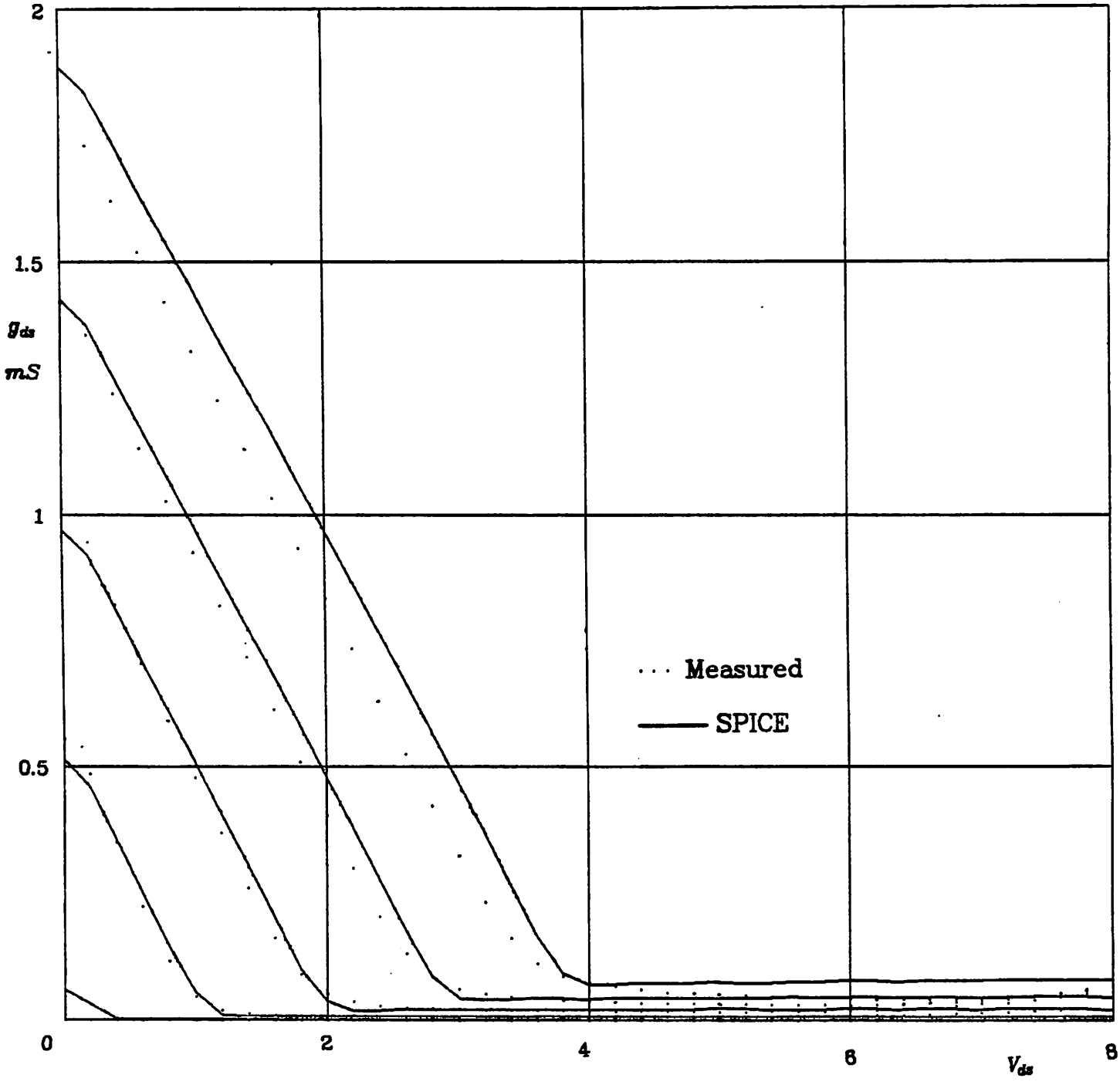

Figure 12: g_{ds} **vs** V_{ds} **For the High Voltage/Current Range.**

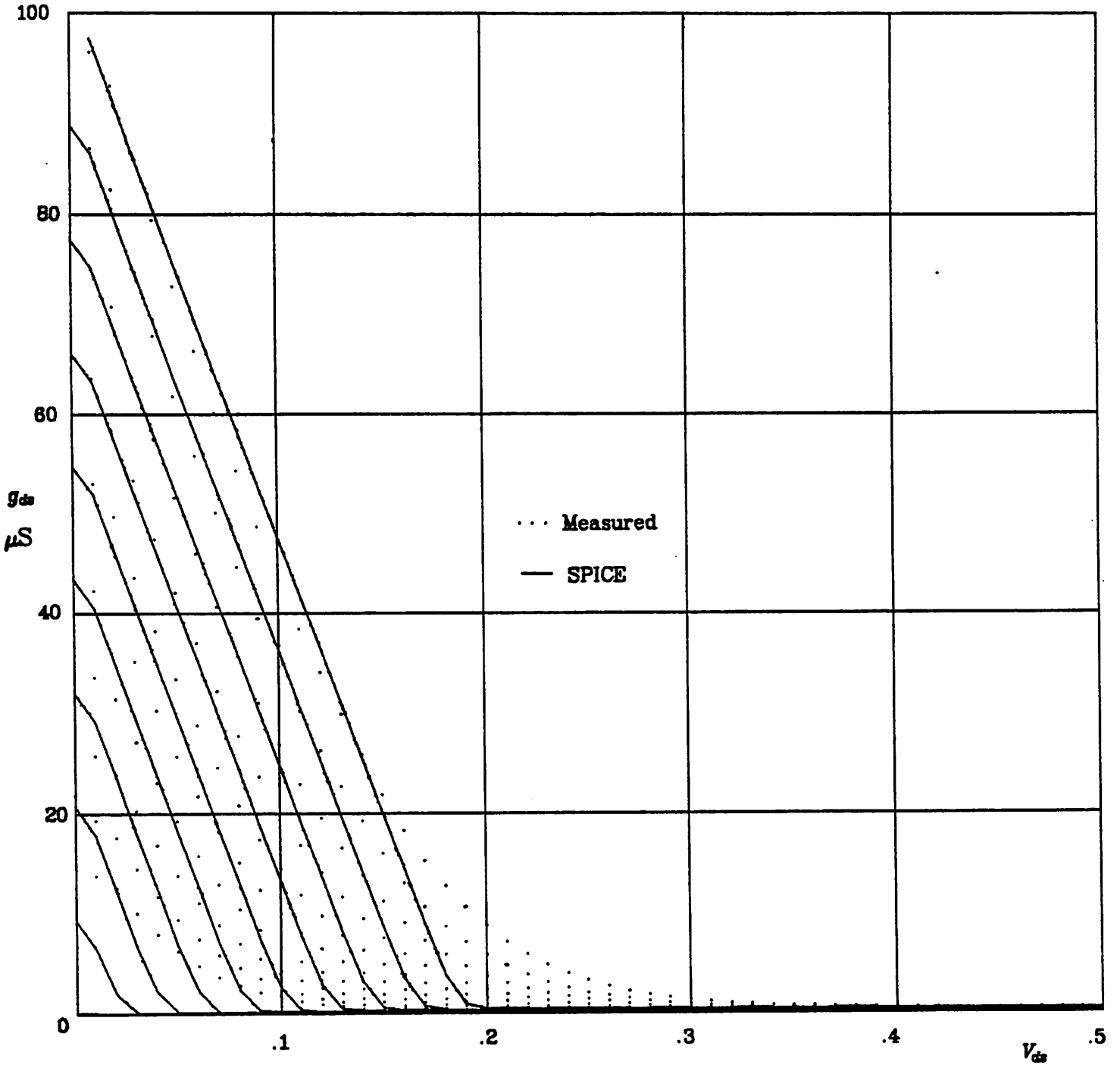

Figure 13a: Measured and Simulated Values of g_{ds} vs V_{ds} for V_{gs} from 0.75V to 0.975V

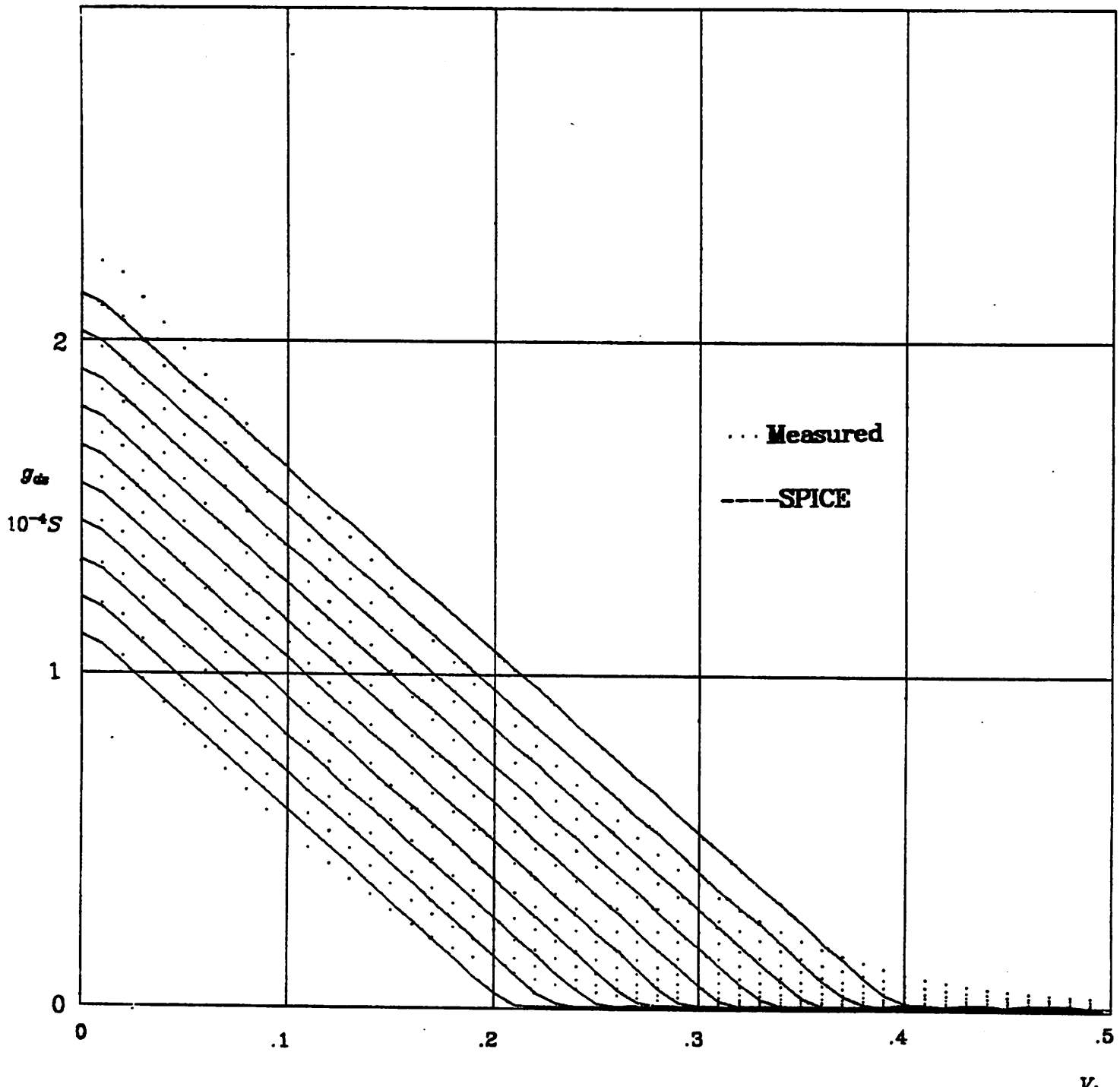

Figure 13b: Measured and Simulated Values of g_{ds} vs V_{ds} for V_{gs} from 1.00V to 1.225V

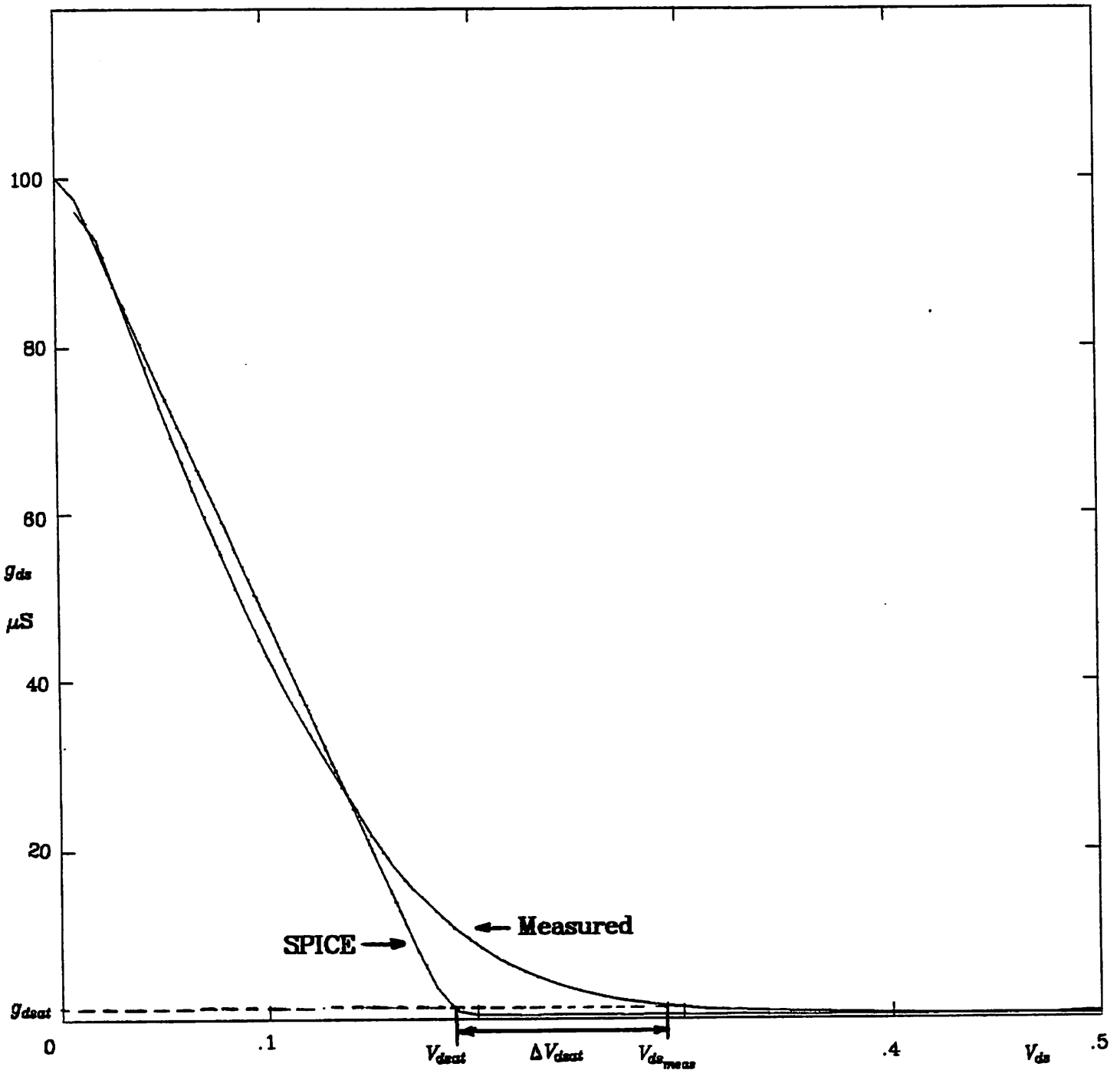

Figure 14: Graphical Interpretation of ΔV_{dsat}.

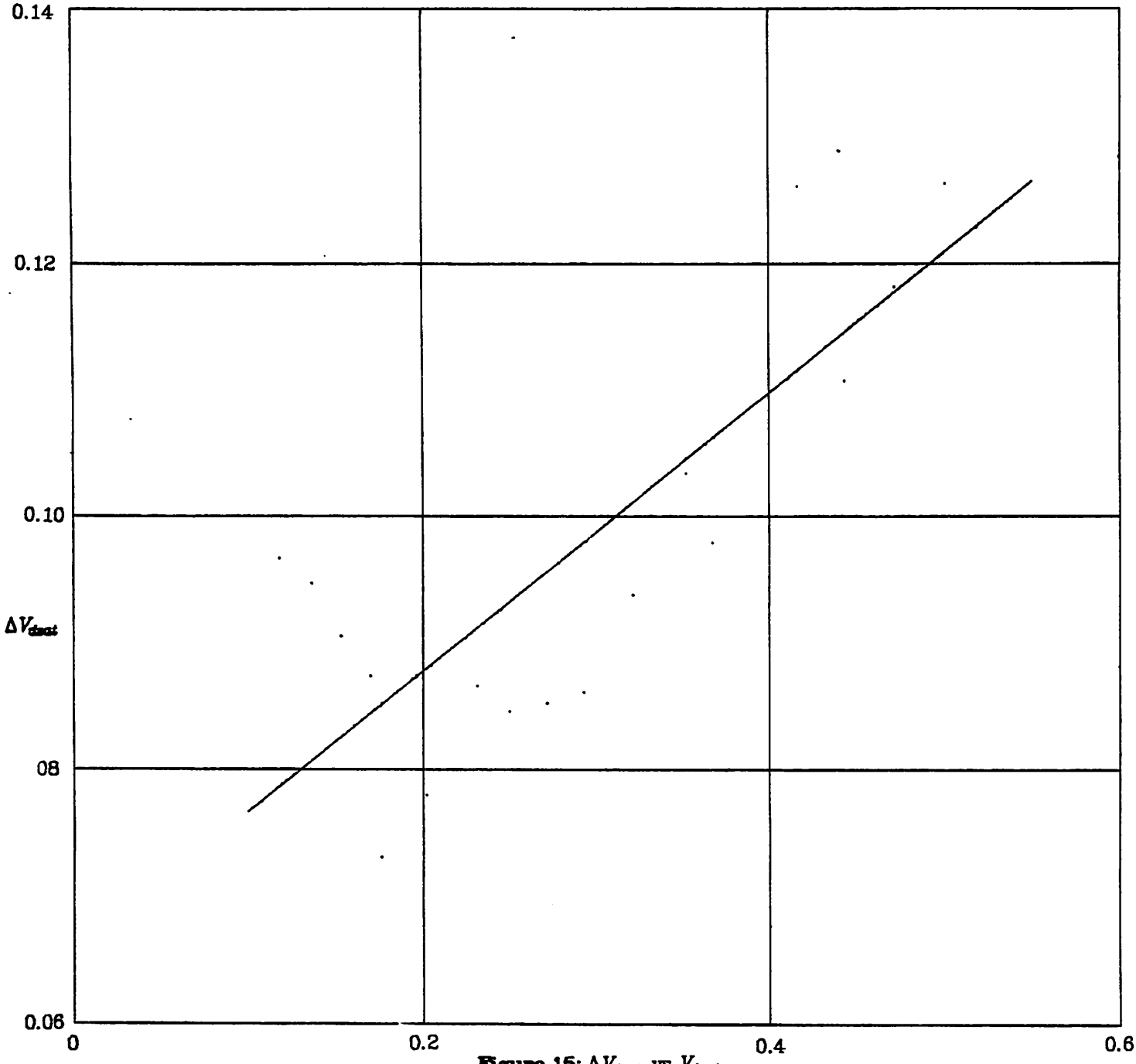

Figure 15: ΔV_{dsat} vs V_{dsat}

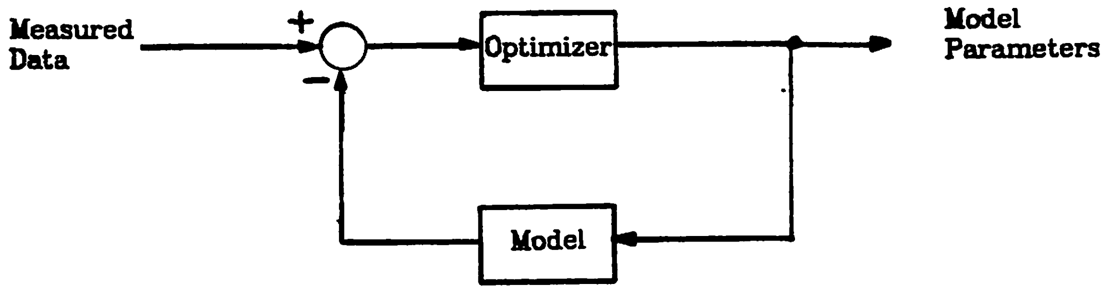

Figure 16: Parameter Optimization With TECAP2.

```
|  TECAP2     Main Menu          Subset                    Status:
|______________________|  |________________________|   |__________________________
|                      |  |                        |   | 0: 2 AM    Sep 22.1983
|D)  Device data       |  |A)  Al                  |   |
|C)  Connections       |  |A1) Interactive         |   |Device:        Test  [NMOS]
|U)  Use setup         |  |A2) Batch mode          |   |
|M)  Measure           |  |A3) Repeat              |   |Setup:   1 = [Id vs Vd]
|E)  Extract           |  |                        |   |Model:   1 = [HPSPICE-MOS]
|S)  Simulate          |  |A5) Exit TECAP2         |   |
|                      |  |                        |   |Plot output:   GRAPHIC_CRT
|O)  Output control    |  |                        |   |Text output:   ALPHA_CRT
|P)  Plot control      |  |                        |   |
|F)  Filer             |  |A9) News                |   |Plot type:  LIN vs LIN
|                      |  |                        |   |Line type:  SOLID
|B)  Build setup       |  |                        |   |Pen color:  #1
|I)  Input sequence    |  |                        |   |Grids    :  TRUE
|Q)  use seQuence      |  |                        |   |
|A)  Action control <==|  |                        |   |Prefix volume = [TECAP:]
|______________________|  |________________________|   |Last sequence:

Type a sequence of commands
```

Figure 17: TECAP2 Main Menu

```
TECAP2    Main Menu
D)  Device data
C)  Connections
U)  Use setup
M)  Measure
E)  Extract              <==
S)  Simulate

O)  Output control
P)  Plot control
F)  Filer

B)  Build setup
I)  Input sequence
Q)  use seQuence
A)  Action control

Subset
E)  Optimize par.
E1) Select model
E2) Enter/Select par
E3) Optimizer Option
E4) Read par.
E5) Save par.
E6) Read par flags

E9) Print par.
E10) Ext level0 par
E11) Ext level1 par
E12) Ext level2 par
E13) Ext level3 par

Status:  <<version 1B.00>>
 0: 2 AM   Sep 22,1983

Device:        Test  [NMOS]

Setup:  1 = [Id vs Vd]
Model:  1 = [HPSPICE-MOS]

Plot output:   GRAPHIC_CRT
Text output:   ALPHA_CRT

Plot type:  LIN vs LIN
Line type:  SOLID
Pen color:  #1
Grids    :  TRUE

Prefix volume = [TECAP:]
Last sequence:
  [I8.]

Type a sequence of commands:
```

Figure 18: TECAP2 Extract Sub-menu

MOS LEVEL 2 model parameters:

Name	Value	Minimum	Maximum	P	Unit
UO	*7.000E+002	1.000E+002	2.000E+003	0	Cm2/V.S
KP	4.364E-005	1.000E-006	1.000E-003	1	A/V2
VTO	7.613E-001	-1.00E+001	1.000E+001	1	Volt
NSS	0.000E+000	0.000E+000	1.000E+012	0	1/Cm2
TPG	1.000E+000	-1.00E+000	1.000E+000	0	-
	0.000E+000	0.000E+000	0.000E+000	0	
TOX	5.000E-008	1.000E-009	1.000E-005	0	Meter
NSUB	2.209E+015	1.000E+013	1.000E+019	1	1/Cm3
GAMMA	0.000E+000	0.000E+000	5.000E+000	0	V^.5
PHI	0.000E+000	4.000E-001	1.000E+000	0	Volt
	0.000E+000	0.000E+000	0.000E+000	0	
	0.000E+000	0.000E+000	0.000E+000	0	
UCRIT	1.000E+004	1.000E+002	1.000E+006	0	V/Cm
UEXP	0.000E+000	0.000E+000	2.000E+000	0	-
UTRA	0.000E+000	0.000E+000	1.000E+000	0	-
LAMBDA	1.646E-002	0.000E+000	1.000E-001	1	1/V

Use KNOB/ARROWS, Type new value+ENTER,
/ to page,   EXECUTE to exit

Figure 19: SPICE Level 2 Model Parameter Table

Appendix A: Listing of TECAP2 Data Files in Directory

/pa/users/gregory/project/hpdata

README gdgrph.f gdgrph measgr measgr.f
n10a2a n10a2b n10a2c n10b2a n10b2b
n10b2c n10c1a n10c1b n10c1c n10d4a
n10d4b n10d4c n10e1a n10e1b n10e1c
n10f1a n10f1b n10f1c n4a2a n4a2b
n4a2c n4b2a n4b2b n4b2b2 n4b2c
n4b2c2 n4e1a n4e1b n4e1b2 n4e1c
n4e1c2 p10a4a p10a4b p10a4c polysmooth
polysmooth.f

This directory contains files of measured data taken using the TECAP2 measurement system. The key to the filenames is as follows:

example: n10f1a

The n indicates an NMOS device: p means PMOS.
The 10 indicates that the drawn channel length was 10 microns (channel width is 100 microns in all cases.)
f1 refers to the location of the device on the test wafer.
The last letter in the filename (a, b, or c) refers to the range of terminal voltages over which the device was tested.

a: Vds 0 to 8v in 0.2v increments
Vgs 0 to 5v in 1v increments

b: Vds 0.01 to 0.50v in 0.01v increments
Vgs 0.75 to 0.975 in 0.025v increments

c: Vds 0.01 to 0.50v in 0.01v increments
Vgs 1.00 to 1.225 in 0.025v increments

The program measgr accepts as input a TECAP2 data file such as the one in this directory and outputs a file which contains the data in a simple x-y format suitable for use in conjunction with the UNIX graph and plot commands.

The program polysmooth accepts as input a TECAP2 data file and performs numerical smoothing on the data, outputting a TECAP2-like file which can be input to measgr if desired.

The program gdgrph accepts as input a TECAP2 data file and differentiates it numerically to obtain gds information and outputs it in a form suitable for graphing via the UNIX graph and plot commands.

Appendix B: Program *POLYSMOOTH*

```
**********************************************************************
c This program reads an HP TECAP data file and smooths the data using polynomial
c regression.  The output is in a format suitable for differentiation using
c program gdgrph.
**********************************************************************

      program polysm
      integer i,j,vdlim,vglim,lim,left,right,point
      doubleprecision a(1000),y(1000)
      doubleprecision vdstrt,vdstop,vdinc,vgstrt,vgstop,vginc
      doubleprecision h(4,4),coef(4),sum(6),b(4),x
      character dummy*40
c
c    The dummy variable picks off the first several lines of text in the
c    TECAP data file.
c
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,x
      print*,x
      read*,x
      print*,x
      read*,dummy
      print*,dummy
      read*,dummy,dummy,dummy,dummy,vdstrt,vdstop,vdlim
      print*,dummy,dummy,dummy,dummy,vdstrt,vdstop,vdlim
c     print*,dummy,vdstrt,vdstop,vdlim
      read*,dummy,dummy,dummy,dummy,vgstrt,vgstop,vglim
      print*,dummy,dummy,dummy,dummy,vgstrt,vgstop,vglim
c     print*,dummy,vgstrt,vgstop,vglim
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,dummy
      print*,dummy
      read*,x
      print*,x
      read*,dummy
      print*,dummy
      vdinc = (vdstop-vdstrt)/(vdlim-1)
      vginc = (vgstop-vgstrt)/(vglim-1)
      lim=vglim*vdlim
c     print*,vglim,vdlim,lim
      read*,(a(j), j=1,lim)
      do 1 j=1,lim
        if (a(j).lt.0.0) a(j)=0.0
1     continue
c
c  Now smooth the data picking 5 points at a time and fitting a least-
c  squares polynomial through the data points.
c
      left=1
```

```
      right=5
      point=3
10      if (right.gt.lim) go to 60
        y(left)=a(left)
        y(left+1)=a(left+1)
15      continue
          do 30 i=1,4
            sum(i)=0.
            b(i)=0.
30        continue
          sum(5)=0.
          sum(6)=0.
          do 40 i=left,right
            x=vdstrt+mod(i-1,vdlim)*vdinc
            sum(1)=sum(1)+x
            sum(2)=sum(2)+x**2
            sum(3)=sum(3)+x**3
            sum(4)=sum(4)+x**4
            sum(5)=sum(5)+x**5
            sum(6)=sum(6)+x**6
            b(1)=b(1)+a(i)
            b(2)=b(2)+a(i)*x
            b(3)=b(3)+a(i)*x*x
            b(4)=b(4)+a(i)*x**3
 40       continue
          h(1,1)=5.
          h(1,2)=sum(1)
          h(1,3)=sum(2)
          h(1,4)=sum(3)
          h(2,1)=sum(1)
          h(2,2)=sum(2)
          h(2,3)=sum(3)
          h(2,4)=sum(4)
          h(3,1)=sum(2)
          h(3,2)=sum(3)
          h(3,3)=sum(4)
          h(3,4)=sum(5)
          h(4,1)=sum(3)
          h(4,2)=sum(4)
          h(4,3)=sum(5)
          h(4,4)=sum(6)
          call solve(h,coef,b,4)
          x=vdstrt + mod(point-1,vdlim)*vdinc
          y(point)=coef(1)+coef(2)*x+coef(3)*x**2 +coef(4)*x**3
          if (int(right/vdlim)*vdlim.eq.right) go to 50
          left=left+1
          right=right+1
          point=point+1
          go to 15
50      continue
          y(right-1)=a(right-1)
          y(right)=a(right)
          right=right+5
          point=point+5
          left=left+5
        go to 10
60      continue
        do 100 j=1,lim
          print*,y(j)
100     continue
        stop
        end

**********************************************************************
c  Subroutine to solve Ax=b, where A is an n by n non-singular matrix.
```

```
c  subroutine does not check for singularities, so it will blow up!!!
*****************************************************************
      subroutine solve(a,x,b,n)
                        integer n
                        doubleprecision a(1:n, 1:n),x(1:n),b(1:n)
c
       doubleprecision q,temp
      do 8 i=1,n-1
c
c  perform partial pivoting (row swap only)
c
         do 3 j=i,n
           if (abs(a(j,i)).le.abs(a(i,i))) go to 3
             do 2 k=i,n
               temp=a(i,k)
               a(i,k)=a(j,k)
               a(j,k)=temp
2            continue
             temp=b(i)
             b(i)=b(j)
             b(j)=temp
3        continue
c
c     perform elimination
c
         do 7 k=i+1,n
           q=-a(k,i)/a(i,i)
           a(k,i)=0.
           b(k)=q*b(i)+b(k)
           do 6 j=i+1,n
             a(k,j)=q*a(i,j)+a(k,j)
6          continue
7        continue
8     continue
c
c  now do back substitution
c
      do 10 i=n,1,-1
        do 9 j=i+1,n
          b(i)=b(i)-a(i,j)*x(j)
 9      continue
        x(i)=b(i)/a(i,i)
 10   continue
      return
      end
```

Appendix C.

The Level 2 MOS model parameters in the table below were used in all SPICE simulations, but were not adjusted during the optimization process.

SPICE PARAMETERS		
Parameter	Value	Unit
UO	700	$Cm^2/(V-s)$
TOX	0.05	μM
XJ	0.4	μM
LD	0.2	μM
W	100	μM
L	10	μM

All other parameters were left at their default values; no ac parameters were specified.

Appendix D: TECAP2 Program Files.

The following files are necessary to the use of the TECAP2 system:

T_LIB2S.CODE

T_TCPCS.CODE

STARTUP.M

LINK.TEXT

T_USER.TEXT

T_USER.CODE

T_LIBS2.CODE and *T_TCPCS.CODE* are library modules containing compiled TECAP2 routines.

STARTUP.M is a file containing initial conditions for which the program searches when it begins execution.

LINK.TEXT is a file of commands which are to be executed on the HP 9836 desktop computer via the *stream* command. When this file is "streamed", the library modules and the User Module (see below) are linked together to form an executable file, *TECAP2* which is then the new version of TECAP2.

T_USER.TEXT is the User Module which contains the user-defined transistor models as discussed in section VI.

T_USER.CODE is the compiled version of the *T_USER.TEXT* PASCAL code. This file is linked with the other libraries to form the TECAP2 program.

Appendix E: TECAP2 Users Module Listing (contains SPICE Level 2 Model).

```
{   ********************  TECAP 2   USER'S MODULE  ********************

    This file contains the procedures for TECAP2 which may be modefied
    by user in order to enhance the capabilities of the released program.

    This modifications should be only done by people familiar with the
    9836 PASCAL system as well as TECAP2 concepts. TECAP2 SYSTEM DESIGNER'S
    MANUAL contains information on how these modification may be done.
    It is the user's responsiblity to follow the instruction carefully.
    HPDA does not intend to provide any support for user's modifications.

    Procedures D10 through D14 and C10..C14 are the dummy procedures for
    the implemented commands. Procedure USER_INIT should supply the command
    names to be included in the menu. Procedure MODEL_THREE is the MOS
    classical model which is supplied here as an example. Other models
    may be implemented in procedures MODEL_FOUR and MODEL_FIVE following
    the same structure.

    ************************************************************************  }


$ref 40$
$ucsd$
$sysprog$
MODULE  user_module;


{ ******     IMR     Version: 1B.00    Date:  7-28-83 ****** }
{ ****** Prototype Version: 1A.00    Date:  5-03-83 ****** }

IMPORT                  { access global data and a few usefull functions }

      $search 'T_LIB2S'$     tecap_data_base ,
      $search 'T_LIB2S'$     tecap_utility ;


EXPORT                  { don't touch procedures declarations in here }

  TYPE  menu_array = ARRAY[1..10] OF STRING[20];

  PROCEDURE user_init(VAR m : menu_array);
  PROCEDURE D10;
  PROCEDURE D11;
  PROCEDURE D12;
  PROCEDURE D13;
  PROCEDURE D14;
  PROCEDURE C10;
  PROCEDURE C11;
  PROCEDURE C12;
  PROCEDURE C13;
  PROCEDURE C14;
  PROCEDURE model_three(VAR p     : par_type;
                        VAR vpin : array8;
                        VAR ipin : array8;
                        VAR qpin : array8;
```

```
                        VAR gmat : array88:
                        VAR xmat : array88;
                        VAR nod  : arraynn;
                            infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);
  PROCEDURE model_four (VAR p    : par_type;
                        VAR vpin : array8;
                        VAR ipin : array8;
                        VAR qpin : array8;
                        VAR gmat : array88;
                        VAR xmat : array88;
                        VAR nod  : arraynn;
                            infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);
  PROCEDURE model_five (VAR p    : par_type:
                        VAR vpin : array8;
                        VAR ipin : array8;
                        VAR qpin : array8;
                        VAR gmat : array88;
                        VAR xmat : array88;
                        VAR nod  : arraynn;
                            infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);


{_____________________________________________________}

IMPLEMENT



VAR     boltz     , charge , ctok , eps0 ,
        epssil , epsox , pi , ref_temp  ,
        nom_temp ,
        ref_vt , ref_eg , ref_ni          : REAL;
        debug                             : BOOLEAN;


PROCEDURE user_init(VAR m : menu_array);
BEGIN
                                  { remove braces and add command names in here }
m[1] := 'D10)                     ';
m[2] := 'D11)                     ';
m[3] := 'D12)                     ';
m[4] := 'D13)                     ';
m[5] := 'D14)                     ';
m[6] := 'C10)                     ';
m[7] := 'C11)                     ';
m[8] := 'C12)                     ';
m[9] := 'C13)                     ';
m[10]:= 'C14)                     ';
END;



PROCEDURE D10;      { command D10 }
BEGIN
IF questionflag THEN BEGIN
                     { add your code here for required data from user }
                     END:
IF executionflag THEN BEGIN
                      { add your code here for execution part }
                      END;
```

```
PROCEDURE model_three(VAR p    : par_type;     { array of parameters }
                      VAR vpin : array8;       { node voltages }
                      VAR ipin : array8;       { node currents }
                      VAR qpin : array8;       { node charges  }
                      VAR gmat : array88;      { real part of Y matrix }
                      VAR xmat : array88:      { imag part of Y matrix }
                      VAR nod  : arraynn;      { node names    }
                          infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);

VAR vdb,vsb,vgb,vdxd,vsxs,
    lambda,xl,xw,devtype,
    uo,vto,tox,nsub,cox,phi,gamma,vfb,beta,idrain,
    vt,eg,ni                                          : REAL;
    n                                                 : INTEGER;

   PROCEDURE model_info;    {setup model parameters}

   VAR n:INTEGER;

         PROCEDURE set_data(n:INTEGER;name,unit:string_80;value,min,max:REAL);
         BEGIN
         p.name[n]:=name;
         p.unit[n]:=unit;
         p.value[n]:=value;
         p.min[n] :=min;
         p.max[n] :=max;
         p.pri[n] :=0;
         END;

   BEGIN
   p.title:='CLASSIC-MOS';
   set_data(1,'UO',   'Cm2/V.S'.700.   100,2000);
   set_data(2,'VTO',   'Volt',  0,     -10,10);
   set_data(3,'NSUB',  '1/Cm3', 1E15,  1E13,1E19);
   set_data(4,'LAMBDA','Meter', 0,     0,1E-5);
   set_data(5,'TOX',   'Meter', 1E-7,  1E-9,1E-5);
   p.number:=5;
   n:=p.number:
   p.acnumber:=0;
   n:=n+p.acnumber;
   p.name[n+1] := ' xl':
   p.name[n+2] := ' xw':
   p.name[n+3] := ' cox';
   p.name[n+4] := ' vfb';
   p.name[n+5] := ' phi';
   p.name[n+6] := ' gamma';
   p.name[n+7] := ' beta';
   p.name[n+8] := ' lambda';
   p.internal:=8;
   p.acinternal:=0;                    { ac intermediate }
   END:

   PROCEDURE get_ext_par:   {get model parameters from array}
   BEGIN
   uo:=p.value[1]:
   vto:=p.value[2]:
   nsub:=p.value[3];
   lambda:=p.va ue[4];
   tox:=p.value 5];
   xl :=dev.l:
```

```
END;


PROCEDURE D11;
BEGIN
END;

PROCEDURE D12;
BEGIN
END;

PROCEDURE D13;
BEGIN
END;

PROCEDURE D14;
BEGIN
END;

PROCEDURE C10;
BEGIN
END;

PROCEDURE C11;
BEGIN
END;

PROCEDURE C12;
BEGIN
END;

PROCEDURE C13;
BEGIN
END;

PROCEDURE C14;
BEGIN
END;



PROCEDURE set_constants;

BEGIN
boltz     := 1.3806226e-23;                    { J/K }
charge    := 1.6021918e-19;                    { C }
ctok      := 273.15;                           { Centigrade to Kelvin }
eps0      := 8.854214871e-14;                  { F/Cm }
epssil    := 11.7 * eps0;                      { Silicon permittivity }
epsox     := 3.9  * eps0;                      { SiO2 permittivity }
pi        := 3.141592654;
nom_temp  := 25 + ctok;                        { nominal   temperature in K }
ref_temp  := 27 + ctok;                        { reference temperature in K }
ref_vt    := boltz * ref_temp / charge;        { kT/q }
ref_eg    := 1.1151;                           { enrgy gap for Si }
ref_ni    := 1.45e10;                          { intrinsic carrier in Cm-3}
END;


{ CLASSICAL  MOSFET  model, don't touch this model. this is just an example }
```

```
xw :=dev.w;

nod[1]:=XX;          { not used }
nod[2]:=WW;          { gate     }
nod[3]:=XX;          { not used }
nod[4]:=EE;          { bulk     }
nod[5]:=NN;          { drain    }
nod[6]:=SS;          { source   }

END;

PROCEDURE get_int_par;
BEGIN
n:=p.number+p.acnumber;
xl   := p.value[n+1]  ;
xw   := p.value[n+2]  ;
cox  := p.value[n+3]  ;
vfb  := p.value[n+4]  ;
phi  := p.value[n+5]  ;
gamma := p.value[n+6]  ;
beta  := p.value[n+7]  ;
lambda:= p.value[n+8]  ;
END;


PROCEDURE calculate_init;    {find intermediate parameters}
                             {this combines MMFIX and MOSMDL in hpspice}

VAR   n : INTEGER;

BEGIN
set_constants;
IF dev.typ=nmos THEN  devtype:=1
                ELSE  devtype:=-1;
vt   := ref_vt;
eg   := ref_eg;
ni   := ref_ni;

cox:=epsox/tox/100;
phi:=2*vt*ln(nsub/ni);
gamma:=sqrt(2*epssil*charge*nsub)/cox;
IF phi=0 THEN phi:=0.6;
                              {convert to CM}
lambda:=lambda*100;
tox:=tox*100:
xl:=xl*100;
xw:=xw*100;

vfb:=devtype*vto-gamma*sqrt(phi)-phi;
beta:=uo*cox*xw/xl;

n:=p.number+p.acnumber;
                {load intermediate parameters in the parameter array}

p.value[n+1] := xl;
p.value[n+2] := xw:
p.value[n+3] := cox:
p.value[n+4] := vfb;
p.value[n+5] := phi;
p.value[n+6] := gamma:
p.value[n+7] := beta;
```

```
p.value[n+8] := lambda;

END;



PROCEDURE calculate_id(vdb,vsb,vgb:REAL;       {same as CALCQ in hpspice}
                       VAR idrain,
                           didvg,didvd,didvs :REAL;
                           dflag,acflag : BOOLEAN);

       { dflag  : derivatives are calculated when true }
       { acflag : charges and capacitances are calculated when true }

VAR    vd,vs,vg,ve.
       vs2,vs3,vs5,vsp5,vs1p5,vs2p5,
       ve2,ve3,ve5,vep5,ve1p5,ve2p5,
       arg,arg1,arg2,arg3,arg1p5.arg2p5.sqarg,
       vth,gamma2,vsat,didve,
       dvedvd,dvedvg.x1fact,c1fact                    : REAL;


BEGIN
vd  := phi+vdb;
vs  := phi+vsb;
vg  := vgb-vfb;
vsp5:= sqrt(vs);
vth := gamma*vsp5+vs;

IF vg>=vth THEN
   BEGIN              {  'on' region (linear and saturated)  }

   gamma2:=gamma*0.5;
   sqarg:=sqrt(gamma2*gamma2+vg);
   vsat:=(sqarg-gamma2)*(sqarg-gamma2);
   vs2:=vs*vs;
   vs3:=vs2*vs;
   vs5:=vs3*vs2;
   vs1p5:=vs*vsp5;
   vs2p5:=vs1p5*vs;

   IF vd<=vsat THEN  BEGIN                {.. linear region  }
                     ve:=vd:
                     dvedvd:=1:
                     dvedvg:=0;
                     END
               ELSE  BEGIN                {.. saturated region  }
                     ve:=vsat:
                     dvedvd:=0;
                     dvedvg:=0;           { dvedvg:=1.0d0-gamma2/sqarg  }
                     END;

   ve2:=ve*ve:
   ve3:=ve2*ve;
   ve5:=ve3*ve2;
   vep5:=sqrt(ve);
   ve1p5:=ve*vep5:
```

```
   ve2p5:=ve1p5*ve;
   arg2:=0.5*(ve2-vs2);
   arg1p5:=gamma*(ve1p5-vs1p5)/1.5;
   idrain:=vg*(ve-vs)-arg1p5-arg2;
   IF dflag THEN BEGIN
                 didve:=vg-gamma*vep5-ve;
                 didvg:=ve-vs+didve*dvedvg;
                 didvs:=-vg+gamma*vsp5+vs;
                 END;

   IF dflag THEN   didvd:=didve*dvedvd;

                                  {.. channel length modulation  }
   IF lambda>0 THEN
      BEGIN                    {.. simple (1+vds*lambda/l) formulation  }
      xlfact:=lambda/xl;
      clfact:=1+xlfact*(vd-vs);
      IF dflag THEN
         BEGIN
         didvd:=clfact*didvd+idrain*xlfact;
         didvs:=clfact*didvs-idrain*xlfact;
         didvg:=clfact*didvg;
         END;
      idrain:=idrain*clfact;
      END;

   END

ELSE                          {.. cut-off region (vg<vth)  }
   BEGIN
   idrain:=0;
   didvg:=0;
   didvd:=0;
   didvs:=0;
   END;

END;     { END of calculate_id (calcq) }


PROCEDURE device;                    {same as MOSMDL in hpspice}

VAR  gccdd,gccbd,gccss,gccbs,igate,devmod,
     didvg,didvd,didvs,
     gccgg,gccgd,gccgs,gccbg,gccdg,gccds,gccsd,gccsg     : REAL ;

BEGIN

IF dev.typ=nmos THEN  devtype:=1
                ELSE  BEGIN
                      devtype:=-1;
                      vdb := -vdb;
                      vsb := -vsb;
                      vgb := -vgb;
                      END;
{  compute drain current and derivatives  }

IF vdb>=vsb THEN
   BEGIN                              {.. normal operation  }
   devmod:=1;
   calculate_id(vdb,vsb,vgb,
```

```
                idrain,
                didvg,didvd,didvs,
                dflag,acflag);
   END
ELSE
   BEGIN                                {.. inverted operation  }
   devmod:=-1;
   calculate_id(vsb,vdb,vgb,
                idrain,
                didvg,didvd,didvs,
                dflag,acflag);
   idrain:=-idrain;
   END;
idrain:=beta*idrain;
IF dflag THEN BEGIN
              didvg:=devmod*beta*didvg;
              didvd:=devmod*beta*didvd;
              didvs:=devmod*beta*didvs;
              END;

IF dflag THEN BEGIN
              gccbd:=0;
              gccbs:=0;
              gccss:=0;
              gccdd:=0;
              gccdg:=didvg;
              gccdd:=gccdd+didvd;
              gccds:=didvs;
              gccsg:=-didvg;
              gccsd:=-didvd;
              gccss:=gccss-didvs;
              END;


{        dx  g   sx  b   d   s   }
{  dx    1                  2       }
{   g        3       4   5   6   }
{  sx            7                  8   }
{   b        9       10  11  12  }
{   d    13  14      15  16  17  }
{   s        18  19  20  21  22  }

{  load conductance matrix  }

IF dflag THEN
   BEGIN
   gmat[1,1]:=0;
   gmat[1,2]:=0;
   gmat[1,3]:=0;
   gmat[1,4]:=0;
   gmat[1,5]:=0;
   gmat[1,6]:=0;
   gmat[2,1]:=0;
   gmat[2,2]:=0;
   gmat[2,3]:=0;
   gmat[2,4]:=0;
   gmat[2,5]:=0;
   gmat[2,6]:=0;
   gmat[3,1]:=0;
   gmat[3,2]:=0;
   gmat[3,3]:=0;
```

```
      gmat[3,4]:=0;
      gmat[3,5]:=0;
      gmat[3,6]:=0;
      gmat[4,1]:=0;
      gmat[4,2]:=0;
      gmat[4,3]:=0;
      gmat[4,4]:=-gccbd-gccbs;
      gmat[4,5]:=gccbd;
      gmat[4,6]:=gccbs;
      gmat[5,1]:=0;
      gmat[5,2]:=gccdg;
      gmat[5,3]:=0;
      gmat[5,4]:=-gccdd-gccdg-gccds;
      gmat[5,5]:=gccdd;
      gmat[5,6]:=gccds;
      gmat[6,1]:=0;
      gmat[6,2]:=gccsg;
      gmat[6,3]:=0;
      gmat[6,4]:=-gccsd-gccsg-gccss;
      gmat[6,5]:=gccsd;
      gmat[6,6]:=gccss;
      END;

   {  load pin currents  }

   ipin[1]:=0;
   ipin[2]:=0;
   ipin[3]:=0;
   ipin[4]:=0;
   ipin[5]:=idrain*devtype -ipin[1] ;
   ipin[6]:=-ipin[1]-ipin[2]-ipin[3]-ipin[4]-ipin[5];

   END;


BEGIN                        {main part of model3}
IF infoflag THEN model_info
            ELSE BEGIN
                 IF initflag THEN BEGIN
                                  get_ext_par;
                                  calculate_init;
                                  END
                             ELSE BEGIN
                                  get_int_par;

                                  vpin[3]:=vpin[6];
                                  vpin[1]:=vpin[5];

                                  vdb:=vpin[5]-vpin[4];
                                  vsb:=vpin[6]-vpin[4];
                                  vgb:=vpin[2]-vpin[4];
                                  vdxd:=vpin[1]-vpin[5];
                                  vsxs:=vpin[3]-vpin[6];
                                  device;
                                  END;
                 END;
END;   {end of model_three}

               {      THE BERKELEY SPICE LEVEL 2 MOSFET MODEL                  }
{                                                                              }
{   AT PRESENT THIS MODEL INCLUDES ONLY DC PARAMTERS;                          }
```

```
{   AC PARAMETERS ARE AS IN THE HP SPICE MOSFET MODEL                    }
{   THE FOLLOWING EFFECTS ARE NOT(!) MODELED AS IN                       }
{   THE SPICE2G.6 LEVEL2 MODEL:                                          }
{       1) Subthreshold current(no weak inversion)                       }
{       2) Velocity saturation                                           }
{       3) Narrow channel effects                                        }
{       4) No thin oxide capacitance model                               }
{       5) No Temperature updating--all simulations are                  }
{          performed at 27C                                              }
{                                                                        }

    {***  Version: 1X.01    Date: 9/01    By: G.Anderson ***}


                { SPICE LEVEL 2 MOS model }

PROCEDURE model_four (VAR p     : par_type;
                      VAR vpin  : array8;
                      VAR ipin  : array8;
                      VAR qpin  : array8;
                      VAR gmat  : array88;
                      VAR xmat  : array88;
                      VAR nod   : arraynn;
                          infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);

CONST gmin = 1e-12;                    { minimum conductance }

VAR vbs,vbd,vdb,vsb,vgb,vdxd,vsxs,
    kp,tpg,nss,lambda,js,
    xj,xd,ld,xl,xw,
    cbd,cbs,cj,cjsw,cgs,cgd,cgb,fc,
    mj,mjsw,
    f1,f2,f3,f4,f5,f6,fcpb,xfc,
    rd,rs,gd,gs,
    devtype,ucrit,
    uo,vto,tox,nsub,
    cox,phi,gamma,vfb,beta,idrain,
    von,vdsat,uexp,utra,vbp,
    dclfct,pb,
    ad,as,pd,ps,cdsat,cssat,
    vt,eg,ni                                          : REAL;
    n                                                 : INTEGER;

   PROCEDURE model_info;    {setup model parameters}

   VAR n:INTEGER;

       PROCEDURE set_data(n:INTEGER;name,unit:string_80;value,min,max:REAL);
       BEGIN
       p.name[n]:=name;
       p.unit[n]:=unit;
       p.value[n]:=value;
       p.min[n] :=min;
       p.max[n] :=max;
       p.pri[n] :=0;
       END;

   BEGIN
   FOR n:=1 TO 29 DO
   set_data(n,'','',0,0,0);
```

```
p.title:='MOS LEVEL 2';
set_data(1,'UO',    'Cm2/V.S',700.    100,2000);
set_data(2,'KP',    'A/V2',   0.      1E-6,1E-3);
set_data(3,'VTO',   'Volt',   0,      -10,10);
set_data(4,'NSS',   '1/Cm2',  0,      0,1E12);
set_data(5,'TPG',   '-',      1,      -1,1);
set_data(7,'TOX',   'Meter',  1E-7,   1E-9,1E-5);
set_data(8,'NSUB',  '1/Cm3',  1E15,   1E13,1E19);
set_data(9,'GAMMA', 'V^.5',   0,      0,5.0);
set_data(10,'PHI',  'Volt',   0,      0.4,1.0);
set_data(13,'UCRIT','V/Cm',   1E4,    100,1E6);
set_data(14,'UEXP', '-',      0,      0,2);
set_data(15,'UTRA', '-',      0,      0,1);
set_data(16,'LAMBDA','1/V'    ,0,     0,1E-1);
set_data(19,'RS',   'Ohm',    0,      0,1E6);
set_data(20,'RD',   'Ohm',    0,      0,1E6);
set_data(21,'XJ',   'Meter',0,      0,1E-5);
set_data(22,'LD',   'Meter',0,      0,1E-5);
set_data(28,'PB',   'Volt' ,0.8,    0,5);
set_data(29,'JS',   'A/m2' ,1E-4,   0,1E-9);
p.number:=29;
n:=p.number;
set_data(n+1,'CJ',      'F/m2',   0,     0,1);
set_data(n+2,'MJ',      '-',      0.5,   0,0.99);
set_data(n+3,'CJSW',    'F/m',    0,     0,1);
set_data(n+4,'MJSW',    '-',      0.33,  0,0.99);
set_data(n+5,'CGS',     'F/m',    0,     0,1);
set_data(n+6,'CGD',     'F/m',    0,     0,1);
set_data(n+7,'CGB',     'F/m',    0,     0,1);
set_data(n+8,'CBD',     'F/m2',   0,     0,1);
set_data(n+9,'CBS',     'F/m2',   0,     0,1);
set_data(n+10,'FC',     '-',      0.5,   0,0.95);
p.acnumber:=10;
n:=n+p.acnumber;
p.name[n+1] := ' xl';
p.name[n+2] := ' xw';
p.name[n+3] := ' cox';
p.name[n+4] := ' vfb';
p.name[n+5] := ' phi';
p.name[n+6] := ' gamma';
p.name[n+10] := ' beta';
p.name[n+11] := ' vbp';
p.name[n+12] := ' lambda';
p.name[n+13] := ' js';
p.name[n+14] := ' xd';
p.name[n+15] := ' xj';
p.name[n+16] := ' ld';
p.name[n+18] := ' gd';
p.name[n+19] := ' gs';
p.name[n+20] := ' vt';
p.name[n+21] := ' pb';
p.name[n+22] := ' ad';
p.name[n+23] := ' as';
p.name[n+24] := ' pd';
p.name[n+25] := ' ps';
p.internal:=25;
n:=n+p.internal;
p.name[n+1] := ' cgs';
p.name[n+2] := ' cgd';
p.name[n+3] := ' cgb';
```

```
p.name[n+4] := ' cbd';
p.name[n+5] := ' cbs';
p.name[n+6] := ' cjsw';
p.name[n+7] := ' mj';
p.name[n+8] := ' mjsw';
p.name[n+9] := ' fcpb';
p.name[n+10] := ' f1';
p.name[n+11] := ' f2';
p.name[n+12] := ' f3';
p.name[n+13] := ' f4';
p.name[n+14] := ' f5';
p.name[n+15] := ' f6';
p.acinternal:=15;                    { ac intermediate }

END;

PROCEDURE get_ext_par;    {get model parameters from array}
BEGIN
uo:=p.value[1];
kp:=p.value[2];
vto:=p.value[3];
nss:=p.value[4];
tpg:=p.value[5];
tox:=p.value[7];
nsub:=p.value[8];
gamma :=p.value[9];
phi:=p.value[10];
ucrit:=p.value[13];
lambda:=p.value[16];
rs:=p.value[19];
rd:=p.value[20];
xj:=p.value[21];
ld:=p.value[22];
pb :=p.value[28];
js :=p.value[29];
xl :=dev.l;
xw :=dev.w;
ad :=dev.ad;
as :=dev.as;
pd :=dev.pd;
ps :=dev.ps;
n:=p.number;

cj:=p.value[n+1];
mj:=p.value[n+2];
cjsw:=p.value[n+3];
mjsw:=p.value[n+4];
cgs:=p.value[n+5];
cgd:=p.value[n+6];
cgb:=p.value[n+7];
cbd:=p.value[n+8];
cbs:=p.value[n+9];
fc:=p.value[n+10];

nod[1]:=NN;          { ext drain}
nod[2]:=WW;          { gate     }
nod[3]:=SS;          { ext source }
nod[4]:=EE;          { bulk     }
nod[5]:=MN;          { int drain    }
nod[6]:=MS;          { int source   }
nod[7]:=XX;
```

```
nod[8]:=XX;

IF rs=0 THEN BEGIN nod[6]:=SS; nod[3]:=XX;  END;
IF rd=0 THEN BEGIN nod[5]:=NN; nod[1]:=XX;  END;

END;

PROCEDURE get_int_par;
BEGIN
uexp:=p.value[14];
utra:=p.value[15];

n:=p.number+p.acnumber;
xl  := p.value[n+1]  ;
xw  := p.value[n+2]  ;
cox := p.value[n+3]  ;
vfb := p.value[n+4]  ;
phi := p.value[n+5]  ;
gamma := p.value[n+6]  ;
beta  := p.value[n+10]  ;
vbp   := p.value[n+11]  ;
lambda:= p.value[n+12]  ;
js := p.value[n+13]  ;
xd := p.value[n+14]  ;
xj := p.value[n+15]  ;
gd := p.value[n+18]  ;
gs := p.value[n+19]  ;
vt := p.value[n+20]  ;
pb := p.value[n+21];
ad := p.value[n+22];
as := p.value[n+23];
pd := p.value[n+24];
ps := p.value[n+25];

n := n+25;
cgs:= p.value[n+1] ;
cgd:= p.value[n+2] ;
cgb:= p.value[n+3] ;
cbd:= p.value[n+4] ;
cbs:= p.value[n+5] ;
cjsw:= p.value[n+6] ;
mj  := p.value[n+7] ;
mjsw:= p.value[n+8] ;
fcpb:= p.value[n+9] ;
f1:= p.value[n+10] ;
f2:= p.value[n+11] ;
f3:= p.value[n+12] ;
f4:= p.value[n+13] ;
f5:= p.value[n+14] ;
f6:= p.value[n+15] ;
END;


PROCEDURE calculate_init;   {find intermediate parameters}

VAR   n : INTEGER;
      fermis,wkfng,fermig,wkfngs,factor,
      new_temp,kt,ratio,ratio1p5,arg,pbfact,vstrip,
      oldgat,gatnew,oldpb,pb_ratio,pb_ratio_p5    : REAL;

BEGIN
```

```
set_constants;
IF dev.typ=nmos THEN  devtype:=1
                ELSE  devtype:=-1;
vt  := ref_vt;
eg  := ref_eg;
ni  := ref_ni;

cox:=epsox/tox/100;
IF kp=0 THEN kp:=uo*cox;
IF nsub>0  THEN BEGIN
                IF nsub<=ni THEN writeln('ERROR  nsub < ni');
                IF phi<=0 THEN  phi:=2*vt*ln(nsub/ni);
                IF gamma<=0 THEN gamma:=sqrt(2*epssil*charge*nsub)/cox;
                fermis:=devtype*0.5*phi;
                wkfng:=3.2;
                              {poly gate work FUNCTION}
                IF tpg<>0 THEN  BEGIN
                                fermig:=devtype*tpg*eg*0.5;
                                wkfng:=3.25+0.5*eg-fermig;
                                END;
                wkfngs:=wkfng-(3.25+0.5*eg+fermis);
                IF vto=0 THEN
                   vto:=wkfngs-nss*charge/cox+devtype*(phi+gamma*sqrt(phi));
                END;
IF phi=0 THEN phi:=0.66;
IF phi<0.1 THEN phi:=0.1;
tox:=tox*100;
js:=js*0.0001;
xj:=xj*100;
ld:=ld*100;

IF acflag THEN BEGIN
                IF cbd=0 THEN cbd:=cj;
                IF cbs=0 THEN cbs:=cj;
                cgs:=cgs*0.01;
                cgd:=cgd*0.01;
                cgb:=cgb*0.01;
                cbd:=cbd*0.0001;
                cbs:=cbs*0.0001;
                cj:=cj*0.0001;
                cjsw:=cjsw*0.01;
                IF fc>0.95 THEN fc:=0.95;
                IF mj=1 THEN mj:=0.99;
                IF mjsw=1 THEN mjsw:=0.99;
                END;

IF rd=0 THEN gd:=0
        ELSE gd:=1/rd;
IF rs=0 THEN gs:=0
        ELSE gs:=1/rs;

vfb:=devtype*vto-gamma*sqrt(phi)-phi;
vbp:=ucrit*epssil/cox;
IF nsub>0 THEN xd:=sqrt(2*epssil/charge/nsub)
          ELSE xd:=0;
                                {device parameters}
IF areaflag THEN BEGIN
                 fcpb:=fc*pb;
                 xfc:=ln(1-fc);
                 f1:=pb*(1-exp((1-mj)*xfc))/(1-mj);
```

```
                    f2:=exp((1+mj)*xfc);
                    f3:=1-fc*(1+mj);
                    f4:=pb*(1-exp((1-mjsw)*xfc))/(1-mjsw);
                    f5:=exp((1+mjsw)*xfc);
                    f6:=1-fc*(1+mjsw);
                    END;

{This temperature updating code is taken verbatim from the HPSPICE MOS}
{model, and does not seem to work.  Since temperature variations are }
{not part of my investigation, this (erroneous) section is ignored.   }

  new_temp := temperature + ctok;                    {temperature update}
  {kt :=boltz*new_temp;}
  {vt :=kt/charge;}
  {writeln('vt=',vt);}
  {ratio := new_temp/nom_temp;}
  {ratio1p5 := ratio*sqrt(ratio);}

  {eg  := 1.16-(7.02e-4*new_temp*new_temp)/(new_temp+1108.0);}
  {arg := -eg/(kt+kt)+ref_eg/(boltz*(ref_temp+ref_temp));}
  {ni  := ref_ni*sqrt(new_temp/ref_temp)*(new_temp/ref_temp)*exp(charge*arg);}
  {pbfact :=(vt+vt)*ln(ref_ni/ni);}

  {kp := kp / ratio1p5;}
  {uo := uo / ratio1p5; }
  {vstrip  := vfb+0.5*devtype*phi;}
  {phi := ratio*phi+pbfact;}
  {vfb := vstrip-0.5*devtype*phi;}
  {vto := devtype*(vfb+gamma*sqrt(phi)+phi);}
  {js  := js*exp(-eg/vt+ref_eg/ref_vt);}
  IF acflag THEN BEGIN
                 oldpb := pb;
                 pb := ratio*oldpb+pbfact;
                 pb_ratio := oldpb/pb;
                 pb_ratio_p5 := sqrt(pb_ratio);
                 cbd := cbd * pb_ratio_p5;
                 cbs := cbs * pb_ratio_p5;
                 fcpb := fcpb / pb_ratio;
                 f1   := f1   / pb_ratio;
                 f4   := f4   / pb_ratio;
                 END;

  pd:=pd*10000;
  ps:=ps*10000;
  ad:=ad*100;
  as:=as*100;
  xl:=xl*100;
  xw:=xw*100;
  xl:=xl-2*ld;
  vto:=devtype*vto;
  beta:=kp*xw/xl;
  cssat := js*ad;
  cdsat := js*as;

  n:=p.number+p.acnumber;
                 {load intermediate parameters in the parameter array}

  p.value[n+1] := xl;
  p.value[n+2] := xw;
  p.value[n+3] := cox;
  p.value[n+4] := vfb;
```

```
p.value[n+5] := phi;
p.value[n+6] := gamma;
p.value[n+10] := beta;
p.value[n+11] := vbp;
p.value[n+12] := lambda;
p.value[n+13] := js;
p.value[n+14] := xd;
p.value[n+15] := xj;
p.value[n+16] := ld;
p.value[n+18] := gd;
p.value[n+19] := gs;
p.value[n+20] := vt;
p.value[n+21] := pb;
p.value[n+22] := ad;
p.value[n+23] := as;
p.value[n+24] := pd;
p.value[n+25] := ps;

n := n+25;
IF acflag THEN BEGIN
               p.value[n+1] := cgs;
               p.value[n+2] := cgd;
               p.value[n+3] := cgb;
               p.value[n+4] := cbd;
               p.value[n+5] := cbs;
               p.value[n+6] := cjsw;
               p.value[n+7] := mj;
               p.value[n+8] := mjsw;
               p.value[n+9] := fcpb;
               p.value[n+10] := f1;
               p.value[n+11] := f2;
               p.value[n+12] := f3;
               p.value[n+13] := f4;
               p.value[n+14] := f5;
               p.value[n+15] := f6;
               END;
END;



PROCEDURE calculate_id(vdb,vsb,vgb:REAL; VAR idrain,qg,qc,qb,
                           ccgg,ccgd,ccgs,ccbg,ccbd,ccbs,
                           didvg,didvd,didvs :REAL;
                           dflag,acflag : BOOLEAN);

      { dflag  : derivatives are calculated when true }
      { acflag : charges and capacitances are calculated when true }

VAR   vd,vs,vg,ve,vbi,v1,
      vs2,vs3,vs5,vsp5,vs1p5,vs2p5,
      vdp5,args,argd,gfacts,gfactd,clfact2,
      ve2,ve3,ve5,vep5,ve1p5,ve2p5,
      arg,arg1,arg2,arg3,arg1p5,arg2p5,varg,
      gammad,gamma2,gfact,vth,sqarg,vsat,
      didve,d2idve,delv,trafac,dtrdve,
      dvedvd,dvedvg,dqgdve,dqbdve,vdenom,
      ufact,dcoef,dufact,xlfact,clfact,dclfact,
      xleff,xwb,xk1,temp                          : REAL;
```

```
                didvg:=ve-vs+didve*dvedvg;
                didvs:=-vg+gammad*vsp5+vs;
                END;
IF acflag THEN  {  here trust that hpspice and spice2g.6 are the same }
   BEGIN                                { calculate charge and C's }
   IF  abs(idrain)<1e-5 THEN
      BEGIN                             { special CASE  ve ~ vs  }
      qg:=cox*(vg-vs);
      ccgg:=cox;
      ccgd:=-0.5*cox;
      ccgs:=ccgd;
      qb:=-cox*gammad*vsp5;
      ccbg:=0;
      ccbd:=-cox*0.25*gammad/max(vsp5,0.01);
      ccbs:=ccbd;
      END
   ELSE                                 { normal CASE }
      BEGIN
      arg2p5:=gammad*0.4*(ve2p5-vs2p5);
      varg:=(vg*arg2-arg2p5-(ve3-vs3)/3)/idrain;
      qg:=cox*(vg-varg);
      dqgdve:=cox/idrain*(varg-ve)*didve;
      ccgg:=cox*(1-(arg2-varg*(ve-vs))/idrain)+dqgdve*dvedvg;
      ccgd:=dqgdve*dvedvd;
      ccgs:=cox/idrain*(varg-vs)*didvs;
      qb:=-cox/idrain*(vg*arg1p5-gammad*gammad*arg2-arg2p5);
      dqbdve:=-cox/idrain*(gammad*vep5+qb/cox)*didve;
      ccbd:=dqbdve*dvedvd;
      ccbs:=-cox/idrain*(gammad*vsp5+qb/cox)*didvs;
      ccbg:=-cox/idrain*(arg1p5+qb/cox*(ve-vs))+dqbdve*dvedvg;
      END;
   END;

IF uexp<>0 THEN
   BEGIN                          {.. mobility factor (a-la bdm)  }
   vdenom:=vg-vth-utra*(ve-vs);
   IF vdenom>vbp THEN
      BEGIN
      arg:=vbp/vdenom;
      ufact:=exp(uexp*ln(arg));
      IF dflag THEN
         BEGIN
         dcoef:=-uexp*ufact*arg/vbp:
           {      didvg:=ufact*didvg+idrain*dcoef   }
         didvg:=ufact*didvg+idrain*dcoef*(1-utra*dvedvg);
         didvs:=ufact*didvs-idrain*dcoef*(0.5*gammad/vsp5+1-utra);
         didve:=ufact*didve-idrain*dcoef*utra;
         END;
      idrain:=idrain*ufact;
      END;
   END;

                               { .. done with 've', use it  }

IF dflag THEN   didvd:=didve*dvedvd;

                               {.. channel length modulation  }
   IF (lambda>0) or (nsub=0) THEN
      BEGIN                    {.. simple 1/(1-vds*lambda) formulation  }
      clfact:=1/(1-lambda*(vd-vs));
      IF dflag THEN
```

```
BEGIN
vd  := max(phi+vdb,1e-8);
vs  := max(phi+vsb,1e-8);
vg:=vgb-vfb;
vsp5:=sqrt(vs);
vdp5:=sqrt(vd);
IF (gamma=0) or (xj=0) THEN   gammad:=gamma
ELSE
   BEGIN
   args:=sqrt(1+xd*2*vsp5/xj);
   argd:=sqrt(1+xd*2*vdp5/xj);
   gfacts:=0.5*xj/xl*(args-1);
   gfactd:=0.5*xj/xl*(argd-1);
   gfact:=1-gfacts-gfactd;
   gammad:=gamma*gfact;
   END;

vth:=gammad*vsp5+vs;
                    {  narrow channel effect is ignored here  }
von:=vth+vfb-vsb;   {  von is threshold voltage referenced TO source  }
{  I'll keep hp's reference here, but I'm not thrilled about it  }
vdsat:=0;
IF vg>vth THEN    { vg>=vth  }

   BEGIN             {  'on' region (linear and saturated)  }

   gamma2:=gammad*0.5;
   sqarg:=sqrt(gamma2*gamma2+vg);
   vsat:=(sqarg-gamma2)*(sqarg-gamma2);
   vs2:=vs*vs;
   vs3:=vs2*vs;
   vs5:=vs3*vs2;
   vs1p5:=vs*vsp5;
   vs2p5:=vs1p5*vs;
   vdsat:=vsat-vs;    { .. vdsat is referenced TO vds for printing only  }


   IF vd<=vsat THEN  BEGIN                {.. linear region  }
                     ve:=vd;
                     dvedvd:=1;
                     dvedvg:=0;
                     END
               ELSE  BEGIN                {.. saturated region  }
                     ve:=vsat;
                     dvedvd:=0;
                     dvedvg:=0;           { dvedvg:=1.0d0-gamma2/sqarg  }
                     END;

   ve2:=ve*ve;
   ve3:=ve2*ve;
   ve5:=ve3*ve2:
   vep5:=sqrt(ve);
   ve1p5:=ve*vep5;
   ve2p5:=ve1p5*ve;
   arg2:=0.5*(ve2-vs2);
   arg1p5:=gammad*(ve1p5-vs1p5)/1.5;
   idrain:=vg*(ve-vs)-arg1p5-arg2;
   IF dflag THEN BEGIN
                 didve:=vg-gammad*vep5-ve;
```

```
            BEGIN
            clfact2:=clfact*clfact;
            didvd:=clfact*didvd-idrain*lambda*clfact2;
            didvs:=clfact*didvs+idrain*lambda*clfact2;
            didvg:=clfact*didvg;
            END;
         idrain:=idrain*clfact;
         END
      ELSE
         BEGIN            {..(lousy) frohman-grove modified a-la newton  }
         arg1:=(vd-vsat)/4;
         arg2:=sqrt(1+arg1*arg1);
         arg3:=sqrt(arg1+arg2);
         clfact:=1/(1-xd/xl*arg3);
         IF dflag THEN dclfct:=0.125*clfact*clfact*xd/xl*(1+arg1/arg2)/arg3;
         xleff:=xl/clfact;
         xwb:=xd*sqrt(pb);
         IF (xleff<xwb) THEN
            BEGIN     {.. limit channel shortening at punch-through  }
            clfact:=xl/xwb/xwb*(2*xwb-xl/clfact);
            IF dflag THEN dclfct:=0.125*xl/xwb/xwb*xd*(1+arg1/arg2)/arg3;
            END;
         IF dflag THEN
            BEGIN
            didvd:=clfact*didvd+idrain*dclfct;
            didvg:=clfact*didvg-idrain*dclfct*dvedvg;
            didvs:=clfact*didvs;
            END;
         idrain:=idrain*clfact;
         END;
      END

ELSE                              {.. cut-off region (vg<vth)  }
   BEGIN              {  at this point do not model subthreshold current  }
   idrain:=0;
   didvg:=0;
   didvd:=0;
   didvs:=0;
   IF acflag THEN
      BEGIN
      IF vg<0 THEN
         BEGIN
         qg:=cox*vg;
         ccgg:=cox;
         END
      ELSE
         BEGIN
         gamma2:=gammad*0.5;
         sqarg:=sqrt(gamma2*gamma2+vg);
         qg:=gammad*cox*(sqarg-gamma2);
         ccgg:=0.5*cox*gammad/sqarg;
         END;
      qb:=-qg;
      ccbg:=-ccgg;
      ccgd:=0;
      ccgs:=0;
      ccbd:=0;
      ccbs:=0;
      END;
   END;
```

```
IF acflag THEN qc:=-(qg+qb);
END;     { END of calculate_id (calcq) }


PROCEDURE device;                    {same as MOSMDL in hpspice}

VAR  fivevt,geqbs,ibulk,evbs,ibd,evbd,geqbd,
     gccdd,gccbd,gccss,gccbs,igate,devmod,
     didvg,didvd,didvs,
     capbs,capbd,qbs,qbd,czbd,czbs,
     czbdsw,czbssw,twopb,fcpb2,czbsf2,czbdf2,
     cbssw5,cbdsw5,sarg1,sarg2,
     xtemp1,xtemp2,xtemp3,
     covlgs,covlgd,covlgb,mj,mjsw,
     xccg2,xccd2,xccs2,xccdg,xccdd,xccds,xccss,
     xccgg,xccgd,xccgs,xccbg,xccbd,xccbs,xccsg,xccsd,
     cssat,cdsat,sarg,
     gccgg,gccgd,gccgs,gccbg,gccdg,gccds,gccsd,gccsg,
     ccgg,ccgd,ccgs,ccbg,ccbd,ccbs,
     qgd,qgs,qgb,qgate,qchan,qbulk                    : REAL ;

BEGIN

IF dev.typ=nmos THEN  devtype:=1
                ELSE  BEGIN
                      devtype:=-1;
                      vdb := -vdb;
                      vsb := -vsb;
                      vgb := -vgb;
                      END;
vbd := -vdb;
vbs := -vsb;

{  determine bulk-drain and bulk-source diode terms  }

IF areaflag THEN
   BEGIN
   fivevt:=-5*vt;
   IF vbs<=fivevt THEN BEGIN
                       geqbs:=-cssat/vbs+gmin;
                       ibulk:=geqbs*vbs;
                       END
                  ELSE BEGIN
                       evbs:=exp(vbs/vt);
                       geqbs:=cssat*evbs/vt+gmin;
                       ibulk:=cssat*(evbs-1)+gmin*vbs;
                       END;
   IF vbd<=fivevt THEN BEGIN
                       geqbd:=-cdsat/vbd+gmin;
                       ibd:=geqbd*vbd;
                       ibulk:=ibulk+ibd;
                       END
                  ELSE BEGIN
                       evbd:=exp(vbd/vt);
                       geqbd:=cdsat*evbd/vt+gmin;
                       ibd:=cdsat*(evbd-1)+gmin*vbd;
                       ibulk:=ibulk+ibd;
                       END;
```

```
                {.. ibd must also be subtracted from drain current  }
   IF  dflag THEN BEGIN
                gccdd:=geqbd;
                gccbd:=-geqbd;
                gccss:=geqbs;
                gccbs:=-geqbs;
                END;
   END
ELSE
   BEGIN        {.. zero out some conductances and igate  }
   ibd:=0;
   ibulk:=0;
   igate:=0;
   gccgg:=0;
   gccgd:=0;
   gccgs:=0;
   gccbg:=0;

   gccdd:=0;
   gccbd:=0;
   gccss:=0;
   gccbs:=0;
   END;

{  compute drain current and derivatives  }

cox:=cox*xl*xw;
IF vbd<=vbs THEN
   BEGIN                                {.. normal operation  }
   devmod:=1;
   calculate_id(vdb,vsb,vgb,
                idrain,qgate,qchan,qbulk,
                ccgg,ccgd,ccgs,ccbg,ccbd,ccbs,
                didvg,didvd,didvs,
                dflag,acflag);
   END
ELSE
   BEGIN                                {.. inverted operation  }
   devmod:=-1;
   calculate_id(vsb,vdb,vgb,
                idrain,qgate,qchan,qbulk,
                ccgg,ccgd,ccgs,ccbg,ccbd,ccbs,
                didvg,didvd,didvs,
                dflag,acflag);
   idrain:=-idrain;
   END;
idrain:=beta*idrain-ibd;
IF dflag THEN BEGIN
               didvg:=devmod*beta*didvg;
               didvd:=devmod*beta*didvd;
               didvs:=devmod*beta*didvs;
               END;

IF dflag THEN BEGIN
               gccdg:=didvg;
               gccdd:=gccdd+didvd;
               gccds:=didvs;
               gccsg:=-didvg;
               gccsd:=-didvd;
               gccss:=gccss-didvs;
               END;
```

```
IF acflag and areaflag THEN
   BEGIN

{  charge storage elements  }

{.. bulk-drain and bulk-source depletion capacitances  }
{   bottom and side wall junction  }

   capbs:=0;
   capbd:=0;
   qbs:=0;
   qbd:=0;
   czbd:=cbd*ad;
   czbs:=cbs*as;

   IF (czbd<>0) or (czbs<>0) or (cjsw<>0) THEN
      BEGIN
      czbdsw:=cjsw*pd;
      czbssw:=cjsw*ps;
      twopb:=pb+pb;
      IF (vbs>=fcpb) or (vbd>=fcpb) THEN
         BEGIN
         fcpb2:=fcpb*fcpb;
         czbsf2:=czbs/f2;
         czbdf2:=czbd/f2;
         cbssw5:=czbssw/f5:
         cbdsw5:=czbdsw/f5:
         END;

{  bulk TO source junction  }

      IF vbs<=fcpb THEN
         BEGIN                              {  vbs < fcpb  }
         sarg:=1-vbs/pb;
         IF czbs<>0 THEN BEGIN
                         sarg1:=exp(mj*ln(sarg));
                         capbs:=czbs/sarg1;
                         qbs:=czbs*(1-sarg/sarg1)*pb/(1-mj);
                         END;
         IF czbssw<>0 THEN BEGIN
                           sarg2:=exp(mjsw*ln(sarg));
                           capbs:=capbs+czbssw/sarg2;
                           qbs:=qbs+czbssw*(1-sarg/sarg2)*pb/(1-mjsw);
                           END;
         END
      ELSE
         BEGIN                                    {  vbs >:= fcpb  }
         xtemp1:=vbs/pb;
         xtemp2:=vbs-fcpb;
         xtemp3:=(vbs*vbs-fcpb2)/twopb;
         IF czbs<>0 THEN BEGIN
                         capbs:=czbsf2*(f3+xtemp1*mj):
                         qbs:=czbs*f1+czbsf2*(f3*xtemp2+xtemp3*mj);
                         END;
         IF czbssw<>0 THEN BEGIN
                           capbs:=capbs+cbssw5*(f6+xtemp1*mjsw);
                           qbs:=qbs+czbssw*f4+cbssw5*(f6*xtemp2+xtemp3*mjsw);
                           END;
         END;
```

```
xdcop[4]:=geqbd:
}
{.. idrain is used in printing as well as noise calculation  }
{
xdcop[5]:=idrain;
xdcop[6]:=ibulk;
xdcop[7]:=geqbs:
}
{.. this is the 'gm' term used in the noise calculation  }
{
xdcop[8]:=didvg;
xdcop[9]:=ccgg;
xdcop[10]:=ccgd;
xdcop[11]:=ccgs;
xdcop[12]:=ccbg;
xdcop[13]:=ccbd;
xdcop[14]:=ccbs;
xdcop[15]:=capbd;
xdcop[16]:=capbs;
}

{  transient analysis  }

{.. divide up the channel charge 50/50 TO source and drain  }
{.. note that symmetry also precludes need for 'devmod' decisions  }

IF acflag THEN
   BEGIN
   {
   xccg2:=-0.5*(ccgg+ccbg);
   xccd2:=-0.5*(ccgd+ccbd);
   xccs2:=-0.5*(ccgs+ccbs);
   xccdg:=xccg2-covlgd;
   xccdd:=xccd2+capbd+covlgd;
   xccds:=xccs2;
   xccsg:=xccg2-covlgs;
   xccsd:=xccd2;
   xccss:=xccs2+capbs+covlgs;
   xccgg:=ccgg+covlgd+covlgs+covlgb;
   xccgd:=ccgd-covlgd;
   xccgs:=ccgs-covlgs;
   xccbg:=ccbg-covlgb;
   xccbd:=ccbd-capbd;
   xccbs:=ccbs-capbs;
   }

{  load susceptance matrix  }

{        dx  g   sx  b   d   s   }
{  dx    1                  2       }
{   g        3       4   5   6   }
{  sx            7                  8   }
{   b        9       10  11  12  }
{   d    13  14      15  16  17  }
{   s        18  19  20  21  22  }

   {
   xmat[1,1]:=0;
   xmat[1,2]:=0;
   xmat[1,3]:=0;
   xmat[1,4]:=0;
```

```
{  bulk TO drain junction  }

      IF vbd<=fcpb THEN
         BEGIN                              {  vbd < fcpb  }
         sarg:=1-vbd/pb;
         IF czbd<>0 THEN BEGIN
                         sarg1:=exp(mj*ln(sarg));
                         capbd:=czbd/sarg1;
                         qbd:=czbd*(1-sarg/sarg1)*pb/(1-mj);
                         END;
         IF czbdsw<>0 THEN BEGIN
                           sarg2:=exp(mjsw*ln(sarg));
                           capbd:=capbd+czbdsw/sarg2;
                           qbd:=qbd+czbdsw*(1-sarg/sarg2)*pb/(1-mjsw);
                           END;
         END
      ELSE
         BEGIN                              {  vbd >:= fcpb  }
         xtemp1:=vbd/pb;
         xtemp2:=vbd-fcpb;
         xtemp3:=(vbd*vbd-fcpb2)/twopb;
         IF czbd<>0 THEN BEGIN
                         capbd:=czbdf2*(f3+xtemp1*mj);
                         qbd:=czbd*f1+czbdf2*(f3*xtemp2+xtemp3*mj);
                         END;
         IF czbdsw<>0 THEN BEGIN
                           capbd:=capbd+cbdsw5*(f6+xtemp1*mjsw);
                           qbd:=qbd+czbdsw*f4+cbdsw5*(f6*xtemp2+xtemp3*mjsw);
                           END;
         END;
      END;
   END;

{.. bulk and channel charge (plus overlaps)  }

IF acflag THEN
   BEGIN
   {
   covlgs:=cgs*xw;
   covlgd:=cgd*xw;
   covlgb:=cgb*xl;

   qgd:=covlgd*(vgb-vdb);
   qgs:=covlgs*(vgb-vsb);
   qgb:=covlgb*vgb;
   qpin[1]:=0;
   qpin[2]:=qgate+qgb+qgd+qgs;
   qpin[3]:=0;
   qpin[4]:=qbulk+qbd+qbs-qgb;
   qpin[5]:=qchan*0.5-qgd-qbd;
   qpin[6]:=-qpin[2]-qpin[4]-qpin[5];
   }
   END;

{  store small-signal parameters  }

{
xdcop[1]:=didvg;
xdcop[2]:=didvd;
xdcop[3]:=didvs;
```

```
    gmat[4,4]:=-gccbd-gccbs;
    gmat[4,5]:=gccbd;
    gmat[4,6]:=gccbs;
    gmat[5,1]:=-gd;
    gmat[5,2]:=gccdg;
    gmat[5,3]:=0;
    gmat[5,4]:=-gccdd-gccdg-gccds;
    gmat[5,5]:=gccdd+gd;
    gmat[5,6]:=gccds;
    gmat[6,1]:=0;
    gmat[6,2]:=gccsg;
    gmat[6,3]:=-gs;
    gmat[6,4]:=-gccsd-gccsg-gccss;
    gmat[6,5]:=gccsd;
    gmat[6,6]:=gccss+gs;
    END;

  {  load pin currents  }

  ipin[1]:=gd*vdxd;
  ipin[2]:=igate*devtype;
  ipin[3]:=gs*vsxs;
  ipin[4]:=ibulk*devtype;
  ipin[5]:=idrain*devtype-ipin[1];
  ipin[6]:=-ipin[1]-ipin[2]-ipin[3]-ipin[4]-ipin[5];

  END;



BEGIN                       {main part of model4}
IF infoflag THEN model_info
            ELSE BEGIN
                 IF initflag THEN BEGIN
                                  get_ext_par;
                                  calculate_init;
                                  END
                             ELSE BEGIN
                                  get_int_par;

                                  IF gd=0 THEN vpin[1]:=vpin[5];
                                  IF gs=0 THEN vpin[3]:=vpin[6];

                                  vdb:=vpin[5]-vpin[4];
                                  vsb:=vpin[6]-vpin[4];
                                  vgb:=vpin[2]-vpin[4];
                                  vdxd:=vpin[1]-vpin[5];
                                  vsxs:=vpin[3]-vpin[6];
                                  device;
                                  END;
                 END;
END;   { model_four }
```

```
   xmat[1,5]:=0;
   xmat[1,6]:=0;
   xmat[2,1]:=0;
   xmat[2,2]:=xccgg;
   xmat[2,3]:=0;
   xmat[2,4]:=-xccgd-xccgg-xccgs;
   xmat[2,5]:=xccgd;
   xmat[2,6]:=xccgs;
   xmat[3,1]:=0;
   xmat[3,2]:=0;
   xmat[3,3]:=0;
   xmat[3,4]:=0;
   xmat[3,5]:=0;
   xmat[3,6]:=0;
   xmat[4,1]:=0;
   xmat[4,2]:=xccbg;
   xmat[4,3]:=0;
   xmat[4,4]:=-xccbd-xccbg-xccbs;
   xmat[4,5]:=xccbd;
   xmat[4,6]:=xccbs;
   xmat[5,1]:=0;
   xmat[5,2]:=xccdg;
   xmat[5,3]:=0;
   xmat[5,4]:=-xccdd-xccdg-xccds;
   xmat[5,5]:=xccdd;
   xmat[5,6]:=xccds;
   xmat[6,1]:=0;
   xmat[6,2]:=xccsg;
   xmat[6,3]:=0;
   xmat[6,4]:=-xccsd-xccsg-xccss;
   xmat[6,5]:=xccsd;
   xmat[6,6]:=xccss;
   }
   END;

{  load conductance matrix  }

IF dflag THEN
   BEGIN
   gmat[1,1]:=gd;
   gmat[1,2]:=0;
   gmat[1,3]:=0;
   gmat[1,4]:=0;
   gmat[1,5]:=-gd;
   gmat[1,6]:=0;
   gmat[2,1]:=0;
   gmat[2,2]:=0;
   gmat[2,3]:=0;
   gmat[2,4]:=0;
   gmat[2,5]:=0;
   gmat[2,6]:=0;
   gmat[3,1]:=0;
   gmat[3,2]:=0;
   gmat[3,3]:=gs;
   gmat[3,4]:=0;
   gmat[3,5]:=0;
   gmat[3,6]:=-gs;
   gmat[4,1]:=0;
   gmat[4,2]:=0;
   gmat[4,3]:=0;
```

IX. References

General Short Channel Device Modeling

[1] L. D. Yau, "A Simple Theory to Predict the Threshold Voltage of Short-Channel IGFET's", *Solid State Electronics*, Volume 17, pp. 1059-1063, (1974).

[2] F. Van de Weile, W. L. Engle, and P. G. Jespers, Eds. *Process and Device Modeling for IC Design*, Noordhoff, Leyden, 1977.

[3] F. M. Klaassen and W. D. J de Groot, "Modelling of Scaled Down MOS Transistors", *Solid State Electronics*, Volume 23, pp. 237-242 (1980).

SPICE Models

[4] A. Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2", Memo No. UCB/ERL M80/7, Electronics Research Laboratory, University of California, Berkeley, Feb. 1980.

TECAP2

[5] E. Khalily, P. Decher, A. Tamer, I. Klein, *TECAP2 System Designer's Manual*, Version 1A.33, Engineering Productivity Division, Hewlett-Packard Co., Cupertino CA, 1982.

[6] E. Khalily, P. Decher, A. Tamer, I. Klein, *TECAP2 Reference Manual*, Part # DA 355, Engineering Productivity Division, Hewlett-Packard Co., Cupertino, CA.

Output Conductance Modeling

[7] D. Frohman-Bentchkowsky, A. S. Grove, "Conductance of MOS Transistors in Saturation", *IEEE Transactions on Electron Device*, Volume ED-16, No. 1, pp. 108-113, January, 1969.

```
PROCEDURE model_five (VAR p     : par_type;
                      VAR vpin  : array8;
                      VAR ipin  : array8;
                      VAR qpin  : array8;
                      VAR gmat  : array88;
                      VAR xmat  : array88;
                      VAR nod   : arraynn;
                          infoflag,initflag,dflag,acflag,areaflag : BOOLEAN);


BEGIN
p.title:='          ';
p.number:=0;
p.acnumber:=0;
{ add model equations as in model_three }
END;   { end of model five }


END.   { end of user_module }
```