BERT – BERKELEY RELIABILITY TOOLS

by

Robert H. Tu, Elyse Rosenbaum, Chester C. Li,
Wilson Y. Chan, Peter M. Lee, Boon-Khim Liew,
J. David Burnett, Ping K. Ko, and Chenming Hu

Memorandum No. UCB/ERL M91/107

4 December 1991
(Revised 27 July 1992)

BERT – BERKELEY RELIABILITY TOOLS

by

Robert H. Tu, Elyse Rosenbaum, Chester C. Li,
Wilson Y. Chan, Peter M. Lee, Boon-Khim Liew,
J. David Burnett, Ping K. Ko, and Chenming Hu

Memorandum No. UCB/ERL M91/107

4 December 1991
(Revised 27 July 1992)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BERT – BERKELEY RELIABILITY TOOLS

by

Robert H. Tu, Elyse Rosenbaum, Chester C. Li,
Wilson Y. Chan, Peter M. Lee, Boon-Khim Liew,
J. David Burnett, Ping K. Ko, and Chenming Hu

Memorandum No. UCB/ERL M91/107

4 December 1991
(Revised 27 July 1992)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BERT - BERKELEY RELIABILITY TOOLS

Robert H. Tu, Elyse Rosenbaum, Chester C. Li, Wilson Y. Chan,
Peter M. Lee, Boon-Khim Liew, J. David Burnett, Ping K. Ko, and Chenming Hu

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

Abstract

Since its first release in May of 1989, the BErkeley Reliability Tools (BERT) has been steadily debugged, enhanced, and augmented. This is the second release of BERT and incorporates many new features not available in the first. BERT 2.0 is a circuit simulation program for reliability analysis, including the original modules: the Circuit Aging Simulator (CAS), Circuit Oxide Reliability Simulator (CORS), Electromigration (EM) - as well as a new module, the Bipolar Circuit Aging Simulator (BiCAS). Just like the first version of BERT, BERT 2.0 works with SPICE2 and SPICE3 circuit simulators (up to SPICE3ex). Furthermore, changes have been made so that the CORS module works with the IRSIM timing simulator. Other major enhancements include integration of BERT with the BSIM2 model, subcircuit expansion, and the addition of a second model for time dependent dielectric breakdown.

This work was supported by the Semiconductor Research Corporation

Preface

This manual describes Version 2.0 of the Berkeley Reliability Tools (BERT). It replaces UCB/ERL Memoranda M90/2, M90/3, and M90/4 which were provided to users of BERT Version 1.0. BERT 2.0 consists of four modules, each for a different reliability failure mechanism, which may be executed concurrently. The four modules and their primary authors are listed below.

Circuit Aging Simulator (CAS) - Peter M. Lee and Mary Kuo

Circuit Oxide Reliability Simulator (CORS) - Elyse Rosenbaum

Electromigration (EM) - Boon-Khim Liew

Bipolar Circuit Aging Simulator (BiCAS) - J. David Burnett and Chester Li

Although each module is identified with one or two primary authors, BERT is a group project. After BERT 1.0 was released, new students joined the project, Robert Tu, Chester Li and, later, Wilson Chan. These three students are responsible for most of the new features in BERT. Their hard work and dedication have resulted in a program which is much easier to use. Mention must also be made of our research advisors, Professors Chenming Hu and Ping Ko, who have guided our efforts the past three years. Their expertise and advice has been invaluable.

Elyse Rosenbaum
November 23, 1991

Table of Contents

1. Introduction[78]

In designing a complex circuit, designers make a large number of circuit simulations, design changes, and optimizations so that they can predict the circuit's performance reasonably accurately before committing it to silicon. It would be unthinkable to bypass the circuit simulation and analysis and rely entirely on the testing of finished IC's to discover errors or to find out if the performance of the circuit meets specifications. Yet, this is basically the way IC reliability is treated today.

A logical alternative is to predict circuit reliability at the circuit design stage. To achieve this goal, we must, for each failure mechanism, identify a set of parameters relevant to circuit reliability (these would be the device model parameters in the analogy of circuit performance simulations) and develop simple methods of extracting these parameters for a given process or technology usually involving accelerated DC stress tests on test structures.

At the present time, reliability assurance relies mainly on failure detection, which occurs only at the end of a lengthy product development and qualification process. It would be highly desirable to predict the circuit reliability at the circuit design stage as we predict circuit functionality and performance today. This is the function of BERT.

The BERT simulator contains modules for four reliability phenomena. They are CAS --Circuit Aging Simulator (for hot-carrier degradation in MOSFETs); CORS -- Circuit Oxide Reliability Simulator; EM -- Electromigration; and BiCAS -- Bipolar Circuit Aging Simulator (for hot-carrier degradation in BJT's). In general, the four modules can be categorized by two functional forms: modules that simulates aged circuit behavior and modules that predicts failure times. CAS and BiCAS allow the user to simulate circuit behavior after many years of continual use. On the other hand, CORS and EM project the fraction of circuits at a given time which will have undergone catastrophic failure due to time dependent breakdown or electromigration. The four modules can be used together or separately to simulate IC reliability. It is anticipated that additional models will be introduced in the future.

BERT is linked to SPICE externally in a pre- and post- processor fashion to form an independent simulator. BERT can be used with either SPICE 2 or 3 and with any of the MOSFET models (Level 1,2,3 or BSIM 1 and 2) and bipolar transistor models.

Users should expect a reliability simulator to require more user input than a circuit simulator, mainly in statistical data collection and parameter extraction on degraded devices. Considerable tweaking of the parameters may be necessary initially before quantitative agreement with failure data is obtained. This is somewhat similar to the early experience with IC process simulators and should not distract from the long-term potential of reliability simulations.

2. Bert Configuration and Operation

This chapter describes the organization and operation of BERT. A description of the program, the steps needed to install and run the program, and a programmer's guide are included. The programmer's guide contains a detailed description of the BERT program, and tips on how to modify the code to suit the user's own purpose. This chapter also describes a shell-script which allows for interaction with BERT in a user-friendly environment.

2.1 SYSTEM CONFIGURATION

Figs. 2.1 and 2.2 show the general operation of BERT. As shown in the figures, BERT consists of a pre- and post-processor to SPICE, with several intermediate files created for communication between the pre- and post-processor (Fig. 2.1). The pre-processor interprets the BERT commands, prepares the input deck so that it is SPICE-compatible, and writes information to an intermediate file for communication with the post-processor.

If CAS, CORS, or BiCAS are called, the pre-processor requests SPICE to print out all node voltages necessary for the calculation of substrate current, oxide voltage or junction voltage, respectively, by the post-processor. Irsim can provide node voltages to CORS in lieu of SPICE; this is detailed in Chapter 7. The post-processor calculations provide the information wanted by the user of CORS or EM but further simulation is necessary to derive the aged circuit waveforms usually requested by the user of CAS or BiCAS. EM requests SPICE to print out all the branch currents so that current density may be calculated by the post-processor.

If aging is requested in CAS, the post-processor creates the file **agetable** listing the ages of all MOSFETs in the circuit (Fig. 2.1). The pre-processor is run a second time with the original input file as its argument (Fig. 2.2) to create the aged model parameters. The pre-processor recognizes that it should perform this function by the presence of file "agetable." The aged model parameters are based on the data in the experimentally-derived stressed device parameter files (barrels in Fig. 2.2). The pre-processor also creates a new input deck to run with the new aged model parameter files. The pre- and post-processor combination is run with this new input deck to obtain the aged behavior of the circuit.

The BiCAS module operates similarly. However, different file names are used, such as **agetableBJT**. The stressed model parameter sets will have "BJT" added to their filename.

The simpler CORS and EM modules run in "single-pass" mode - unless the user wishes to have the effects of burn-in simulated by CORS. The user may not request CAS, BiCAS, or EM analysis along with burn-in. Burn-in is simulated by two passes through CORS; this process is explained in Appendix A in the .BURNIN command summary.

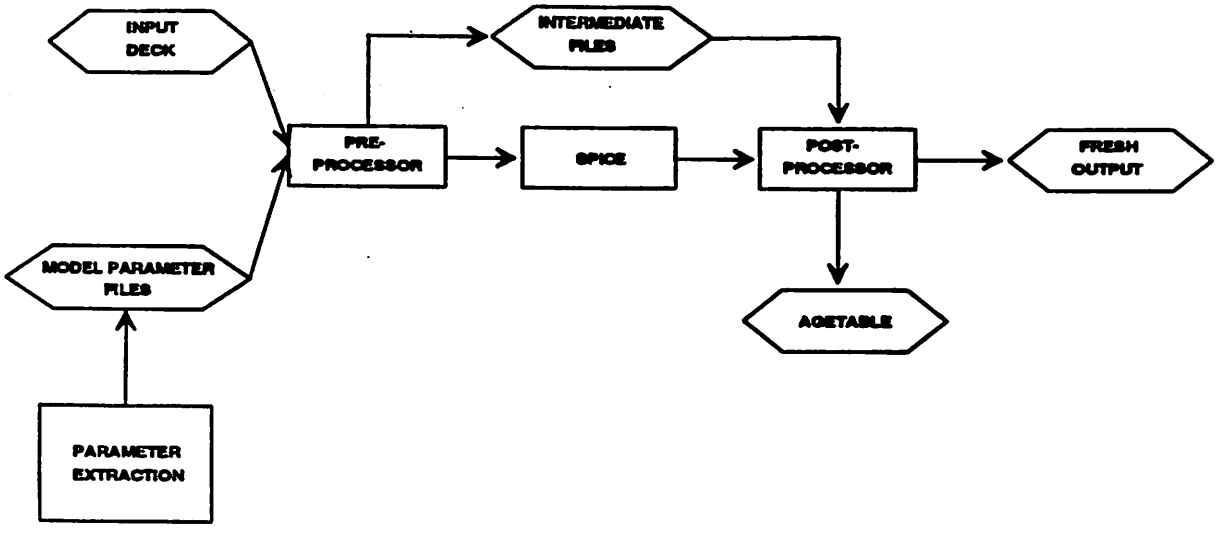


Fig. 2.1 BERT system configuration: First pass is to calculate degradation information (such as device lifetime) and the agetable.

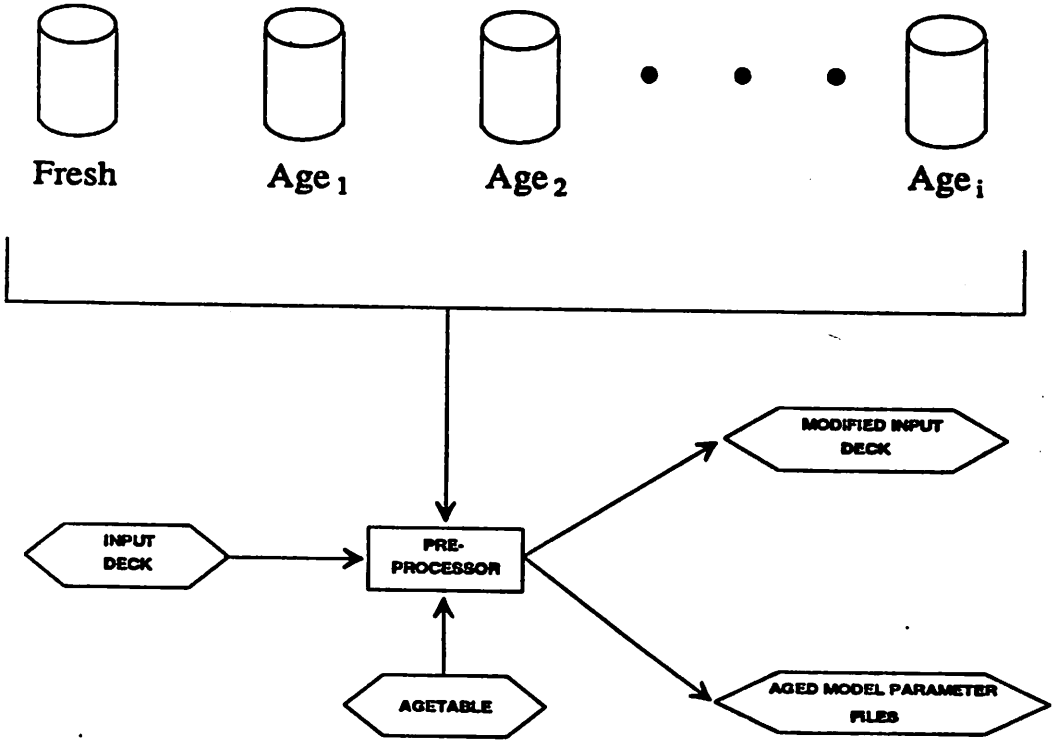


Fig. 2.2 BERT system configuration: Second pass is to generate the aged model parameters at the future time point specified by the '.age' command.

2.2 INSTALLING AND RUNNING BERT

A standard makefile exists in the bert directory for compiling of both the pre- and post-processor. Simply typing **'make'** on UNIX systems in bert/ will compile all modules and place all executable codes in the bert/exe/ directory. All CAS-related files (source code, sample input decks and sample BSIM1 and BSIM2 model parameter files) are located in bert/CAS; all CORS related files are located in bert/CORS; all EM related files are located in bert/EM; and all BiCAS files are located in bert/BiCAS. To remove all object files (*.o) from these directories, type **'make clean'**. The user is encourgaed to examine the README files and sample input decks inside the Sample directories (bert/CORS/Sample, bert/CAS/Sample, etc).

Typing **'make,'** will also create the executable codes for the layout extractor **mextra**, **sim2spice** and a cif plotting program **cif2ps**, all for use with EM. A "stand alone" version of the electromigration simulator can be created by typing **make emonly** in the **BERT/EM** directory.

To run BERT execute the shell-script (Section 2.4) or type

prebert *-x deck* **| spice | postbert >** *outfile*

where the deck is the file containing the circuit description and bert commands; *x* is "2" for SPICE2G6; "3" for SPICE3C1,SPICE3D2, SPICE3E1, or SPICE3E2; or "4" for the special version of SPICE3C1 where the level 4 MOSFET model is the BSIM2 model. The default (if no option is specified) is "3."

To use BERT to run EM and CORS analyses or to only find device lifetime from CAS and BiCAS, the above execution is the only step required. To simulate circuit aging, the following three lines must be executed in the order shown:

prebert *-x deck* **| spice | postbert >** *outfile1* (to generate the agetable/agetableBJT),

prebert *-x deck* (to generate the aged process files),

prebert *-x* **inpdeck | spice | postbert >** *outfile2* (to simulate the aged circuit).

The second step generates *inpdeck*. ".age/.ageBJT" and ".ageproc/.ageprocBJT" aging commands are omitted from *inpdeck*.

A different procedure is required to use Irsim rather than SPICE with BERT. Details may be found in Chapter 7. In this release of BERT, CORS is the only module capable of processing data from Irsim.

People who would just like to use BERT,as is, and have no need for programming details should skip the next section (2.3) which describes the detailed implementation of BERT.

2.3 BERT PROGRAMMER'S GUIDE

This section provides information required only by users planning to modify the source code. Four modules (CAS,CORS,EM, and BiCAS) are currently implemented in BERT in modular fashion. Section 2.3.1 explains how to place new modules in BERT. Section 2.3.2 further details the CAS module and suggests methods of adding new device models, and the remaining two sections examine debugging options and explain how to modify BERT for use with a circuit simulator other than SPICE.

2.3.1 Overall BERT Structure

Fig. 2.1 outlines the configuration of the pre- and post-processor in relation to SPICE. Basically, the pre-processor prebert reads in the SPICE input deck and device model parameter files, filters from the input deck all non-SPICE commands, and adds SPICE commands that are needed for BERT calculations (e.g. voltage node printout commands for CAS or CORS). If the CAS or BiCAS modules have been invoked, the following steps are taken. If the model parameter files declared in the **.PROCESS** command are BSIM1 process files (see Ref[8] or Section 2.1), they are converted to the SPICE .model format and copied to the temporary files *rwmdx*, where *x* is an integer which labels the different model parameter sets. The *rwmdx* files are created for use by the post-processor postbert. When using SPICE2 with the BSIM1 model in table format, the temporary files *RWPRCX* are also created; these are in the same format as the BSIM1 process files except all lines dealing with I_{Sub} and the degradation parameters are commented out. These files are used in the **.PROCESS** declaration in the modified SPICE2 input deck. For models in .model format, the temporary files *rwmdx* are appended to the input deck, with I_{Sub} and degradation parameters commented out during the copy.

Prebert also creates a file called *rawsub* to communicate with postbert. The file *rawsub* contains information such as which BERT-specific commands have been specified the number of transistors to be analyzed, etc.

When prebert is run with the file *agetable* present, prebert will read in the transistor ages from *agetable*, generate the aged model parameters, and place them into files named *AGEx*, where *x* differentiates between model parameters sets. Different aged model parameters sets exist either because the fresh transistors had different parameter sets or the aged transistors have different ages. Prebert then creates the file inpdeck which contains a modified input file so that the new model parameter files *AGEx* can be used. The aging commands **.AGE** and **.AGEPROC** are also deleted in *inpdeck*. In a similar way, BiCAS creates an aged input deck.

Postbert reads the SPICE output file from the standard input, and *rawsub*. If CAS or BiCAS has been invoked, all model parameters are loaded in by opening the *rwmdx* files. After all calculations are completed, postbert writes to standard output any output requested from SPICE by the user with the results of BERT appended. All temporary files, such as *rawsub* and *rwmdx* files are erased.

Both prebert and postbert are written in standard UNIX C and use dynamic memory allocation (using malloc() and calloc()) so that there is no inherent limitations to the number of transistors or number of SPICE timesteps that can be accommodated (available memory and hard disk space become the determining factor). Program testing has been done on DEC VAX machines, the DEC 3100 and 5000 workstations, the Sun 3/60, and the Sun SPARCstation .

The following subsections describe prebert and postbert in more detail.

Prebert:

Fig. 2.3.1 shows how the main routine in the source code file *prebert.c* is organized. The routine OpenRaw() opens *rawsub* for writing, and the routine ArgU() reads in all arguments and makes sure relavant files can be opened. At this stage, the input deck is temporarily copied to a file called *rawinp1*.

Control is then transferred to the routine PreFilter() which scans *rawinp1* for BERT-specific commands and sets up the necessary flags. The flags HotModel, DBModel, EMModel and/or

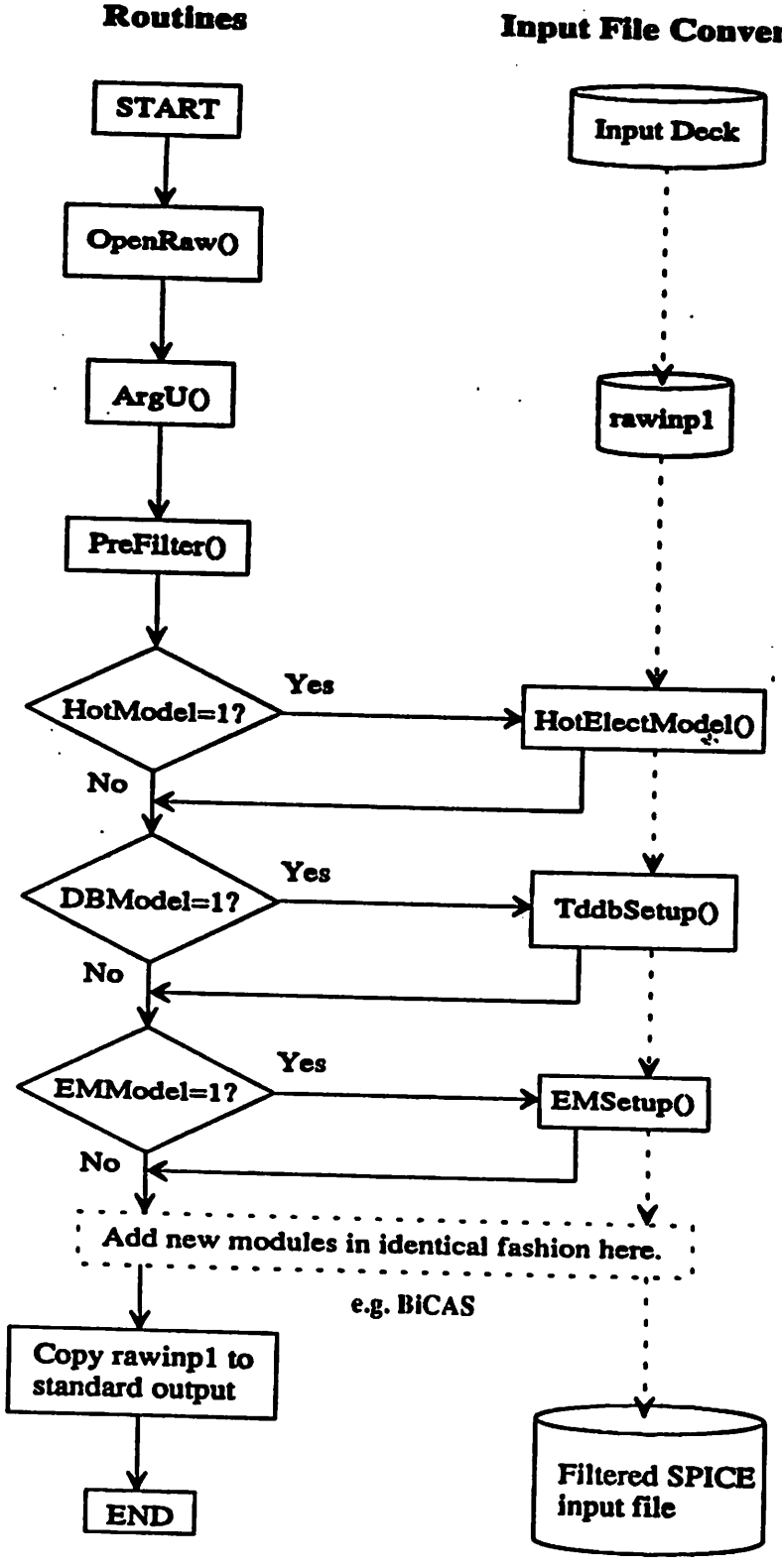


Fig. 2.3.1 The structure of the main program in *prebert.c*. The solid arrows show the path of program execution, while the dotted arrows show the path the input deck takes through the different modules. Square symbols represent routines or a particular program function, diamonds are decision points, and barrels represent files.

BJTPass are set to 1 if CAS-specific, CORS-specific, electromigration-specific, or BiCAS-specific commands are present, respectively. Once this is done, control is passed to each of the four modules depending upon the status of the appropriate flag.

Any module invoked reads in *rawinp1*, adds lines to *rawsub*, and modifies *rawinp1* as necessary (for example, in a CAS simulation, .PRINT commands for voltage printouts are added). In practice, since reading and writing simultaneously to the same file is not permitted, the original *rawinp1* is copied to a file *rawinp2*. All additions or modifications necessary are also done in the file *rawinp2*. Finally, before the particular module is exited, *rawinp2* is moved to *rawinp1*. Once all necessary modules are executed, the main routine checks for any errors that may have occurred, and if none have occurred, *rawinp1* is dumped to standard output after appending the model parameter files. Prebert then erases *rawinp1*.

Error flagging is handled by the global integer variable 'Error' and the character string 'ErrMsg' which contains a description of the error (including the one-letter module identifier and two-digit error code for prebert errors). If an error occurs, ErrMsg is printed to standard error so that the user can immediately see that something is wrong by looking at his or her screen. ErrMsg is also placed in the *rawsub* file so that postbert can print it into the standard output as well.

Following is a list of the source code files and a brief description of each:

1) prebert.c: This file contains the main routines of BERT and CAS.
2) procsub.c:This file contains routines that create the *rwmdx* and *RWPRCX* files. The *rwmdx* files are in SPICE '.model' format, while the *RWPRCX* files are in BSIM1 process file format with the Isub and degradation parameters commented out.
3) age.c: This file contains the main routines that calculate the aged model parameter sets for CAS.
4) bsimext.c:This file contains routines that extract the aged model parameter sets for the BSIM1 model (SPICE3 Level 4).
5) spext.c:This file contains routines that extract the aged model parameter sets for SPICE Level 1, 2, and 3 models.
6) tddb.c: This file contains all routines related to CORS.
7) preem.c: This file contains all routines related to electromigration simulation.
8) premisc.c: This file contains miscellaneous routines used by the rest of the program.
9) bertpr.h, bertpr2.h:These files contain the global variable declarations of prebert. *bertpre.h* contains all variable declarations and is included only in *prebert.c* and *procsub.c*. All variables in *bertpr2.h* are declared as extern variables and are used for the other source files.
10) tdbprdef.h: This file contains the variable declarations for tddb.c.
11) empredef.h: This file contains the variable declarations for the electromigration module.
12) preirsim.c: This file contains the routines for parsing Irsim input files.
13) preirsim.h: This file contains the variable declarations for preirsim.c.
14) prebjt.c: This file contains all routines related to BiCAS.
15) bjtprdef.h, bjtprdef2.h: These files contain global variable declarations for BiCAS.
16) agebjt.c: THis file contains the main routines that calculate the aged model parameter sets for BiCAS
17) expckt.c: This file contains routines which allow subcircuit expansions.

18) bsim2ext.c: This file contains routines that extract the aged model parameter sets for the BSIM2 model.

The following items must be done to add a new module to prebert:

1) Code to search for new keywords and commands should be added to the routine PreFilter().
2) Appropriate global flags must be set according to the keywords found during execution of PreFilter() so that control is transferred to the new module when required. The global flags should be declared in the variable declarations files *bertpr.h* and as an extern variable in the new module's "definition" (*.h) file.
3) Each module must adhere to the rule of reading *rawinp1*, writing the modified input file to *rawinp2*, then moving *rawinp2* to *rawinp1* (*rawinp2* should not remain after the module is exited). All non-SPICE commands related to the module should be commented out. The module should be in its own source file, and it may have its own variable declaration file. Any other intermediate files that need to be created must begin with the word 'raw', and any files not needed by the remaining portion of prebert or postbert must be deleted before exiting the module.
4) If any errors are detected, the variable 'Error' must be set to 1 and a description of the error including its error code should be copied to the string variable 'ErrMsg'. The error code consists of one or two letters identifying the module involved, followed by a two-digit number, in turn followed by a colon before the actual error message (see Appendix A for error codes for examples).

Postbert:

Fig. 2.3.2 shows the routines within the main program of *postbert.c*. The format for the program execution path, the path of the output file, and the symbols are the same as in Fig. 2.3.1. Following some error checking, standard input (i.e. the SPICE output file) is copied to the file *rawoutl*. Similar in function to *rawinp1* in prebert, *rawoutl* becomes the file that each module reads to do its calculations. Any deletions to the output file (for example, deleting all CAS-requested voltage node printouts) are made by copying *rawoutl* to file *rawout2*, making the necessary changes and subsequently moving *rawout2* to *rawoutl* before the module is exited. Any information to be added to the output by each module is saved in separate files (for instance, the file *rawhot* in CAS), and these files are appended to the final *rawoutl* file after all modules have been executed. Finally, *rawoutl* is placed into standard output.

Error flagging is handled in a similar fashion to prebert, except that the integer variable Error is equal to 1 if a postbert error has occurred, and Error is equal to 2 if a prebert error previously occurred (the prebert error assignment is automatically handled by the main program as long as each module in prebert correctly assigns the error variable and message). Again, all error messages are written into the character string ErrMsg, including the one- to two-letter module identifier and, this time, a three-digit error code for postbert errors, and the error messages are printed out both to standard error (which shows up immediately on the screen) and to the output file.

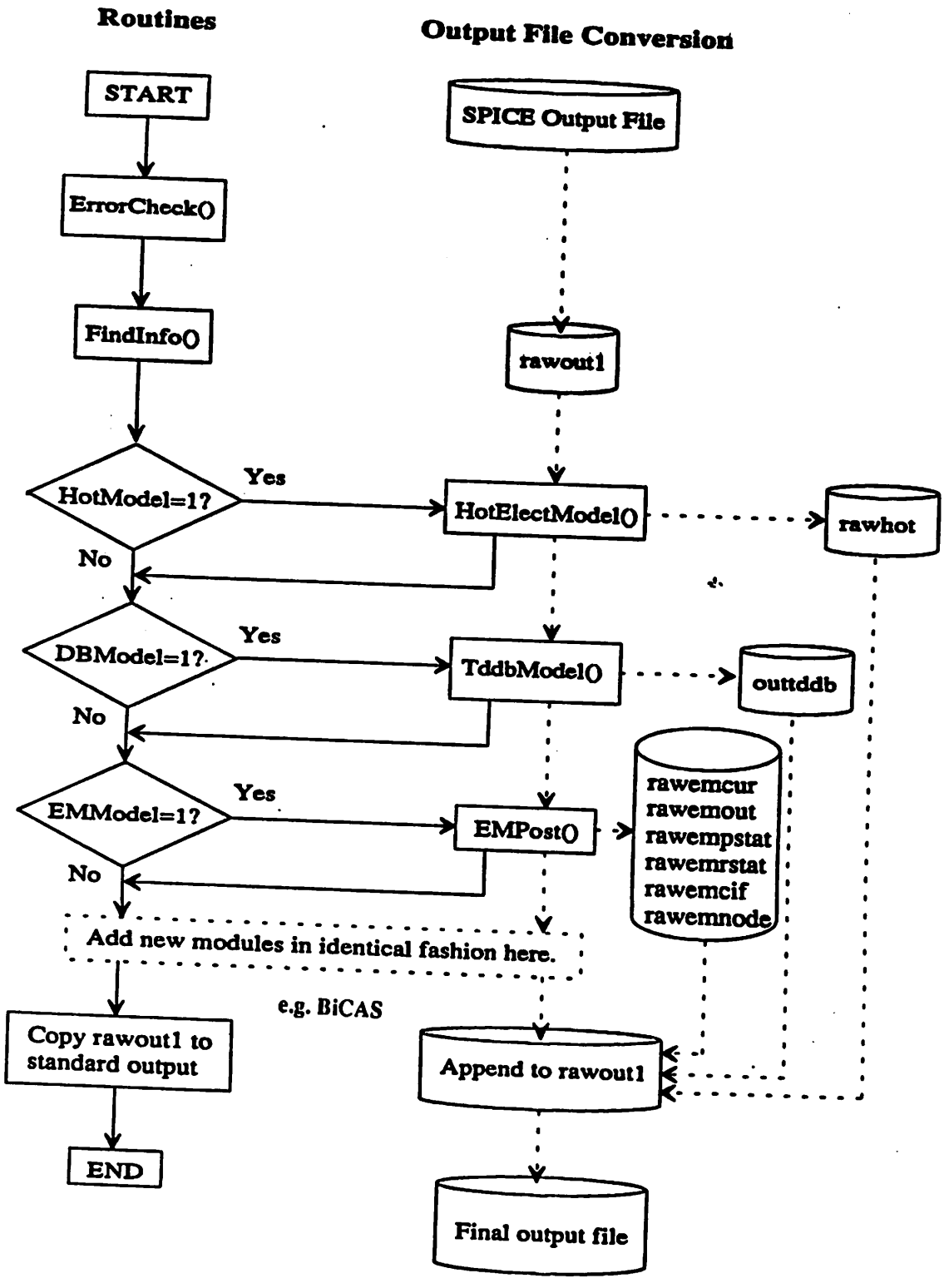


Fig. 2.3.2 The structure of the main program in *postbert.c*. The solid arrows show the path of program execution, while the dotted arrows show the path the SPICE output file takes through the different modules. Square symbols represent routines or a particular program function, diamonds are decision points, and barrels represent files.

Following is a list of the source code files and a brief description of each:

1) postbert.c: This file contains all the main routines of BERT.
2) readpar.c:This file contains routines that read various transistor information and obtain model parameters from the *rwmdx* files.
3) degcalc.c: This file contains the Isub, Igate, and degradation models for CAS.
4) mos.c: This file contains all the drain current models (SPICE Level 1, 2, 3, and 4 (BSIM1)).
5) output.c: This file contains all the routines used for CAS output file printout.
6) postddb.c, nrcalcs.c: These files contain all CORS-related routines.
7) postem1.c, postem2.c:These files contain all routines associated with the electromigration module.
8) postmisc.c: This file contains miscellaneous routines used by the rest of the program.
9) bertpo.h, bertpo2.h:These files contain the global variable declarations of postbert. *bertpo.h* contains all variable declarations and is included only in *postbert.c*. All variables in *bertpo2.h* are declared as extern variables and are used for the other source files .
10) tdbpodef.h: This file contains the variable declarations for CORS.
11) empodef.h: This file contains the variable declarations for the electromigration module.
12) postbjt.c: This file contains all routines associated with the BiCAS module.
13) bjtpodef.h: This file contains the variable declarations for the BiCAS module.
14) consort.c: This file contains routines for parsing the output of irsim and reformatting it.
15) bsim2.c: This file contains the routines to do BSIM2 model aging computation.

When adding new modules to postbert, these guidelines should be followed:

1) Declare global flagging variables in the variable declaration files *bertpo.h* and the new module's include file (as an extern in the new include file).
2) Call module from the main routine.
3) In the routine FindInfo() in the main routine, modify code to search for new keywords in the *rawsub* file
4) Each module should be written in its own source code file, with any variables specific to the module declared in its own variable declarations file.
5) Within each module, the file *rawout1* should be read to obtain waveform information necessary for reliability calculations. *rawout1* should be copied unaltered to *rawout2* except for items requested of SPICE by the module prebert during preprocessing that are no longer necessary. However, care must be taken not to delete information that will be used by any subsequent modules (For example, voltage node printouts are used by both CAS and CORS. CAS deletes the printout from *rawout2* unless CORS is requested. In this case, CORS deletes the node voltage printouts from *rawout2*. Any information that needs to be added to the output file should be stored in a separate file such as *outtddb*, *rawdevtab*, Ifoutbjt, etc. Finally, *rawout2* should be copied to rawout1 before the module is exited.

6) Make sure postbert appends the module's output file to rawout1 in the main routine after all modules have been executed.

2.3.2 Adding New Device Models to CAS

Adding new device models into the CAS portion of BERT involves a fair amount of effort because there will be new parameters, which means that new keywords must be searched for and new data structures must be created. However, the necessary routines can be created in parallel with the existing routines for the present models.

First of all, in prebert, no modification is necessary if the device model parameters are to be declared in the usual '.model' format. If the model parameter format is unlike that of the SPICE .model card, a translator routine such as Proc2ModSub() in procsub.c must be written to translate the format to a form recognizable by SPICE. Of course, the new model must already be implemented in SPICE.

In postbert, the formulation of the new model should be added as a routine to the file mos.c for drain current models, and degcalc.c for models involved with device degradation (including substrate and gate currents). Once these have been added, the routines involving calculation of currents should be used in the routine CalculateCurrents() in *postbert.c*, while routines involved with calculating device degradation should be included in SubAnalysis() in *postbert.c* in a similar fashion to the existing degradation routines.

The new model parameters should be appended to the existing TmArray data structure in the variable declaration files *bertpo.h* and *bertpo2.h*. A new routine in parallel with ReadParameters() should be created in *readpar.c* to recognize new parameter keywords and load the model parameters into the appropriate variables. A routine similar to BSIMsetup() and SPICEsetup() should be created to assign default values to parameters not assigned explicitly.

Fig. 2.3.3 shows the algorithm in *age.c* used in prebert to calculate aged model parameters for each transistor listed in the agetable. After general memory allocation (MemAlloc()), the SPICE input file is read to find the files listed in the .AGEPROC command (GetAgeCards()). The agetable is read by GetDevAge(), and then the loop to calculate the aged model parameters for each device listed in the agetable begins. The appropriate pre-stressed model parameter files are loaded in ReadAgePar(), and a check is made to see what model is used. The routines BSGenAgeParm() and SPGenAgeParm() do the actual calculation to find the aged model parameter set from the pre-stressed model parameters and device age, and the newly-created parameters are placed in the *AGEX* files. Memory space is freed for use by the next loop if more devices remain in the *agetable*. Once all aged model parameters have been found for all devices, then GenInpDeck() takes the original SPICE input file and creates a modified input deck *inpdeck* for use with the new aged parameter sets.

The aging portion of CAS uses separate data structures for the BSIM1 model and the other SPICE models. In addition, separate data structures are used for the pre-stressed model parameters and calculated aged model parameters. For new models, new structures should be created, and the routine ReadAgePar() must be modified to load the new parameters. New aged-parameter extraction routines specific to the model should be written (as indicated in the dotted box in Fig. 2.3.3). The extraction routines written for the SPICE Level 1, 2, and 3 models are the best routines to follow when creating the new routines.

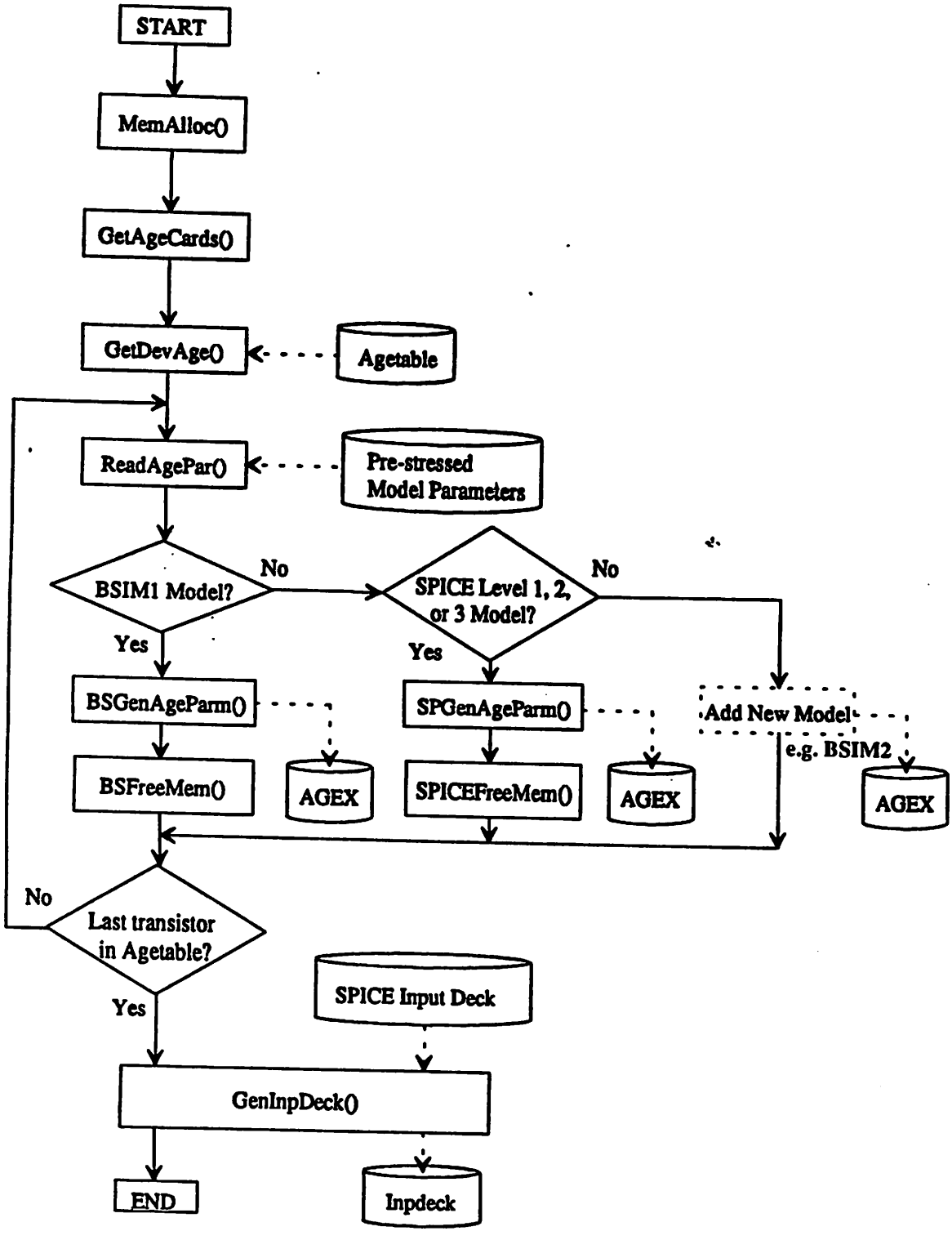


Fig. 2.3.3 Algorithm implemented in *age.c* of prebert to calculate aged model parameters from pre-stressed model parameters and device age. Symbols have the same meaning as in Figs 2.3.1 and 2.3.2.

2.3.3 Modifying BERT To Work With Other Circuit Simulators

BERT, in its present form, cannot be used with circuit simulators other than SPICE2G.6, SPICE3Cx... SPICE3Ex, and Irsim because of its use of specific keywords and patterns to recognize input deck commands and output file information. If a different circuit simulator is to be used, then all parsing routines in prebert and postbert must be modified so that the correct keywords are searched. Prebert must be further modified to output the appropriate commands for the new simulator. These functions are concentrated in prebert routines (and subroutines of) ReadVoltage(), TddbModel(), EMPost(), ReadVoltageBJT(). In postbert, the routines to be modified are main(), FindInfo() in postbert.c and the subroutines they call.

2.3.4 Debugging Options in BERT

To use the UNIX C debuggers (such as dbx), alter the top-level *Makefile* so that CFLAGS is defined as '-g' (to include the symbolic table used for debugging) rather than '-O' (optimized for execution).

2.3.5 Summary

This section has provided some hints and guidelines to follow when adding new modules to BERT, new device models to CAS, and when use of a circuit simulator other than SPICE is desired. Adding new modules to BERT is relatively simple because of the modular structure. Adding new device models to CAS involves more effort because of the new model parameters that need to be recognized. Modifying BERT to be used for non-SPICE circuit simulators involves modifying input parsing and output routines and requires a varying degree of effort depending upon how similar the circuit simulator is to SPICE.

2.4 BERT SHELL SCRIPT PROGRAM FOR UNIX ENVIRONMENTS

A shell script-program for BERT has been developed for use in a UNIX environment. A menu-driven system enables the user to choose the desired simulation without having to enter the lengthy piping commands. The shell-script has also been designed to alert the user whenever he or she may be executing BERT in an erroneous fashion. The shell-script was designed to be informative and user-friendly. All operations are automated for convenience and speed. In addition, an option is added to iteratively age the circuit so that ongoing degradation can be taken into account.

To call the shell, simply type

bert < *input file* > < *output file* >

Specifying the input and the output file in the command line is optional; the shell will prompt the user to enter them if they are not specified.

Fig. 2.4.1 shows the opening menu. Three different options are available: 1) Simulation using SPICE, 2) Simulation using IRSIM, 3) Exit the program. These options are self-explanatory. Choosing option 1 leads to the main menu. Fig. 2.4.2 shows the main menu. Three different options are available: 1) a one-pass simulation (such as calculating the degradation information in CAS, or doing a CORS or electromigration simulation), 2) a CAS/BiCAS aging type simulation (multiple-pass circuit aging), or 3) and program termination.

Fig. 2.4.3 shows the menu when the one-pass option is selected. The first option allows the user to alter the input file by entering the UNIX "vi" editor, while the second option permits the user to use entirely different input and output files. Option (3) makes it possible to call and use model parameter files from a different directory than the one in which the simulation is done. This allows the user to store all his model parameter files in one directory while switching from directory to directory for his simulations. Option (4) starts the actual simulation, option (5) returns the user to the main menu (Fig. 2.4.2), and option (6) exits the shell.

Fig. 2.4.4 shows the screen format when option (3) (selecting a new path for the parameter files) is chosen. Presently, the user can customize his shell by writing in four different often-used paths in the shell code. Path (1) is the default path that is active whenever the shell program is started. The user can also enter an entirely new path (option (5)). This path, however, will not be stored when the shell is exited. Option (6) allows the user to stay with the present path listed at the top of the screen.

After doing all the necessary adjustments, the user can select option (4) in the one-pass menu (Fig. 2.4.3) to start the simulation. While the programs are running, the present status of the execution is successively displayed until the END OF SIMULATION menu appears (Fig. 2.4.5). Here, the user has the choice of viewing the newly created output file, going back to the one-pass menu (Fig 2.4.3), going back to the main menu (Fig. 2.4.2), or exiting the shell altogether.

When the CAS/BiCAS option is selected from the main menu, a menu similar to the one-pass menu is displayed (Fig. 2.4.6). All options are identical, except for option (2). This option is only available to CAS. This option enables iterative simulation so that ongoing degradation can be taken into account. For instance, the user may want to simulate his circuit 10 years in the future. He may iterate only once so that the aged process files created by CAS are directly based on the degradation that occurred in the fresh circuit, or he may subdivide the 10 years into, for example, 10 intervals equally spaced in log time, so that each CAS simulation will generate model parameter files that have aged for an intermediate length of time. The aged model parameters of the first simulation is used by the next CAS simulation to produce the next set of aged model parameters files. This cycle is continued progressively until the 10 years is reached. In this way, the change in circuit degradation from continually changing device characteristics can be taken into account. Greater accuracy can undoubtedly be achieved with a larger number of iterations, but with a sacrifice in speed and CPU time.

Once the simulation is started by selection of option (5) from the CAS/BiCAS menu (Fig. 2.4.6), diagnostics that are similar to that of the one-pass case are displayed. The diagnostics show the present status of the simulation, with the END OF SIMULATION menu again appearing when program execution is completed. The same options as in the one-pass case are present, except that the user can now view the output files of both the fresh and the aged circuit.

When the Irsim Option is selected, the menu in Fig. 2.4.7 will be diplayed. This menu is basically the same as the one-pass menu.

Once the shell script is exited, all temporary files used by the shell and the pre- and post-processors are erased. The input file, the fresh and aged output files, the agetable of each iteration, and the aged model parameter files remain. The fresh output file can be identified by a ".fr" suffix added to the name of the output file specified by the user. A word of caution. The BERT system uses temporary files beginning with "raw" and "age", both in lower and upper cases. The user should avoid naming his personal files matching this pattern, as these files will be overwritten and erased when BERT is exited.

Following are instructions for installing the shell-script program. The shell-script is a normal text file and can be copied directly into the desired directory for use. Four additional items, however, must be taken care of by the user. These involve editing the shell-script. All four modifications are made at the beginning of the program, and directions are contained in the listing (Fig. 2.4.8). First, you must specify the path of your SPICE/Irsim executable code so that BERT knows where to find SPICE/Irsim. Specify the path after the "alias spice" statement (e.g. in Fig 2.4.8, the path for SPICE is specified as "usr/cad/spice3"). Second, you must determine the proper flag for the pre-processor *prebert* so that it will work correctly with the SPICE version specified in the "alias spice" statement. This flag which appears in double quotes after "set preflag = " in Fig. 2.4.8, corresponds to the '-x' flag in Section 2.2. Third, the location of the various programs required to run BERT must be specified in the seven "alias" statements that follow next. The relevant paths are entered in the third column of text in similar fashion to specifying the SPICE path. The various programs include the BERT pre- and post-processor, as well as a collection of programs that are used exclusively by the BERT shell script. All necessary executable files except SPICE are placed in the bert/exe directory once the makefile is executed (Section 2.2). Finally, the fourth modification is to set the paths for the location of the process files that will appear in the path selection menu (Fig. 2.4.4). The text after the equal sign in the "set PfDir*x* = " statements should appropriately be replaced by the desired paths. Note that double quotes must surround the path listing.

Once these additional items are done, BERT can be used immediately.

2.5 RESTRICTIONS

Hot-carrier degradation of transistors in which the source and drain are switched regularly in circuit operation (such as transmission gates) cannot be simulated properly in this version of CAS.

2.6 SUMMARY

We have described the installation and operating procedure of BERT in this section. Being able to separate the pre- and post-processing adds flexibility in use, but for convenience, a UNIX shell script program has been developed that automates the simulation process, as well as making iterative aging simulations possible. Finally, in this chapter we have discussed the detailed operation of BERT and methods which users can use to modify BERT to satisfy their own needs.

```
------------------------------------------------------------------------
                    BERKELEY RELIABILITY TOOLS (BERT)
------------------------------------------------------------------------

      (1) Simulation using SPICE.

      (2) Simulation using IRSIM.
          (*** This is only possible with the CORS module ***)

      (3) Exit Program.

Enter desired option.  >
```

Fig. 2.4.1 The opening menu of the shell script program.

```
------------------------------------------------------------------------
                              MAIN MENU
------------------------------------------------------------------------

     (1) Isub, Igate, Hot-Carrier Lifetime, Oxide (CORS), or
         Electromigration Analysis.

     (2) Hot-Carrier Circuit Aging Simulation (CAS/BiCAS).

     (3) Exit program.

Enter desired option.  >
```

Fig. 2.4.2 The initial main menu of the shell script program.

```
------------------------------------------------------------------------
  ISUB, IGATE, HOT-CARRIER LIFETIME, OXIDE AND ELECTROMIGRATION MENU
------------------------------------------------------------------------

            (1) Edit the input deck.

            (2) Specify new input and output files.

            (3) Specifiy new path for model parameter files.

            (4) Start simulation.

            (5) Return to MAIN MENU.

            (6) Exit program.

Enter desired option.  >
```

Fig. 2.4.3 The one-pass menu of the shell script program.

```
-------------------------------------------------------------------------
     Present Path for CAS process files =  /usr/cad/bert/CAS/Sample
-------------------------------------------------------------------------

Choose new path from the following :

(1)  /usr/cad/bert/CAS/Sample
(2)  ../../CAS/Pfiles
(3)
(4)
(5) Set new path.
(6) Remain with present path.

Enter desired option. >
```

```
-------------------------------------------------------------------------
     Present Path for BiCAS process files =  /usr/cad/bert/BiCAS/Sample
-------------------------------------------------------------------------

Choose new path from the following :

(1)  /usr/cad/bert/BiCAS/Sample
(2)  ../../CAS/Pfiles
(3)
(4)
(5) Set new path.
(6) Remain with present path.

Enter desired option. >
```

Fig. 2.4.4 Changing the path of the location of the model parameter files.

```
                <<<<<< SIMULATION IN PROGRESS >>>>>>>

            >>>> Pre-processing finished. Executing SPICE..... <<<<

            >>>> SPICE finished. Executing the post processor..... <<<<

   -------------------------------------------------------------------------
                              END OF SIMULATION
   -------------------------------------------------------------------------

         (1) Edit the output file.

         (2) Return to Hot-Carrier, Oxide and Electromigration Menu.

         (3) Return to Main Menu.

         (4) Exit program.

Enter desired option. >
```

Fig. 2.4.5 Example of a END OF SIMULATION menu.

```
------------------------------------------------------------------------
                          CAS/BiCAS MENU
------------------------------------------------------------------------

              (1) Edit the input deck.

              (2) Select the number of iterations desired for
                  intermediate aged process file calculations.
                  (Present Value = 1)

              (3) Specify new input and output files.

              (4) Specify new path for model parameter files.

              (5) Start simulation.

              (6) Return to MAIN MENU.

              (7) Exit program.

Enter desired option. >
```

Fig. 2.4.6 The CAS/BiCAS menu of the shell script program.

```
------------------------------------------------------------------------
          CIRCUIT OXIDE RELIABILITY SIMULATION USING IRSIM
------------------------------------------------------------------------

              (1) Edit the input decks.

              (2) Specify new input and output files.

              (3) Start simulation.

              (4) Return to MAIN MENU.

              (5) Exit program.

Enter desired option.  >
```

Fig. 2.4.7 The Irsim menu of the shell script program.

```
#!/bin/csh -f
#############################################################################
#                                                                           #
#                        BERKELEY RELIABILITY TOOLS (BERT)                  #
#                             SHELL SCRIPT PROGRAM                          #
#                                                                           #
#                                 Version 2.0                               #
#                               By Peter M. Lee                             #
#          Department of Electrical Engineering and Computer Sciences       #
#                        University of California, Berkeley                 #
#                                 July 10, 1990                             #
#                                                                           #
#     This program runs a shell script for use with the pre- and post-      #
# processing system of the BErkeley Reliability Tools (BERT).  With this    #
# shell script, automatic execution of the single-pass Isub, Igate, hot-    #
# carrier lifetime, oxide (CORS), and/or electromigration simulations can   #
# be done, as well as the multiple pass Circuit Aging Simulator (CAS) type  #
# simulations.  With the modification in version 2.0, this shell script     #
# can also do circuit oxide simulation (CORS) utilizing IRSIM.              #
#                                                                           #
# Copyright (c) 1988, 1989, 1990   Peter M. Lee  All rights reserved.       #
#                                                                           #
# Modified 1991 by W. Chan                                                  #
#                                                                           #
#############################################################################
#
# Edit the following ten lines to set the correct path for the various
# simulation programs.
#
alias spice    /usr/cad/spice3
#
# Set preflag to the argument of prebert corresponding to the spice
# version specified above according to the following:
#
# SPICE2G6: -> set preflag to "-2".
# SPICE3C1 or SPICE3D2 with level 4 as BSIM1 model, level 5 as BSIM2 model
#    or SPICE3E1 or SPICE3E2: -> set preflag to "-3".
# SPICE3C1 special version with BSIM2 model as level 4: -> set preflag to "-4".
#
set preflag = "-3"
#
alias prebert   prebert
alias postbert  postbert
alias CopyProc  copyproc
alias CopyProcBjt cpprocbjt
alias DelProc   delproc
alias AgeFilter agefilt
alias AgeConv   ageconv
alias ConvInp   convinp
alias irsim     ~wchan/Irsim/irsim
#
# Put the paths of the location of your process files equal to the
# variables PfDir1 through PfDir4 for CAS and PfDirBi1 through PfDirBi4
# for BiCAS.
#
# Paths for process files for CAS:-
#
set PfDir1 = "/usr/cad/bert/CAS/Sample"
set PfDir2 = "../../CAS/Pfiles"
set PfDir3 =
set PfDir4 =
#
# Paths for process files for BiCAS:-
#
set PfDirBi1 = "/usr/cad/bert/BiCAS/Sample"
set PfDirBi2 = "../../CAS/Pfiles"
set PfDirBi3 =
set PfDirBi4 =
#
#############################################################################
# Do not modify below this line.                                            #
#############################################################################
clear
set PfDir = $PfDir1
```

Fig. 2.4.8 The first several lines of the shell script program. To customize the shell, the user must modify the "alias" statements and the "set PfDirx" statements as described in the text.

3. Circuit Aging Simulator (CAS)

Hot-electron degradation is becoming a worrisome issue as device dimensions continue to shrink. Past research has concentrated on the device level to study and model hot-carrier degradation [21,22,23,24,25,26,28,4,31,32,34,6,36,37,38,40,42,44,13], and to design hot-carrier-resistant structures such as the Lightly-Doped-Drain (LDD) and Doubly-Diffused-Drain (DDD) devices. By understanding the mechanisms of hot-carrier degradation and using these degradation-resistant structures, devices with satisfactory hot-electron reliability can be designed.

So far, because hot-carrier degradation research has concentrated on the device level, researchers have used device-related parameters such as drain current degradation, transconductance degradation, or threshold voltage shift to quantify and judge hot-carrier reliability. It remains unclear, however, how these device-level degradation changes actual circuit behavior. Reference [7] has demonstrated the varying sensitivity of the circuit output on each transistor in the circuit. For example, a certain transistor M1 may experience 20% drain current degradation but affect the circuit output minimally, while another transistor M2 may suffer only 5% degradation and yet cause substantial degradation in circuit performance. Furthermore, each transistor may have varying sensitivity depending upon what aspect of the circuit performance is considered critical. Trying to define an arbitrary level of device degradation (such as 10% current degradation) can be misleading and overly conservative or dangerously optimistic. In addition, to predict degraded circuit behavior, the degradation of the individual devices of the circuit subjected to the dynamic circuit-determined waveforms must be predicted. This is in sharp contrast to the DC or uniform AC waveforms used to study device-level degradation behavior.

This chapter introduces the Circuit Aging Simulator (CAS) [11]. CAS predicts hot-carrier degradation of MOS devices undergoing dynamic operation in circuits and can therefore predict the degraded circuit behaviour after a user-specified operating time. Used in conjunction with the SPICE circuit simulator [20,46,19], CAS uses a parametric I_{sub} model with a device degradation model to calculate aged model parameters for each device in a circuit. With CAS, circuit output sensitivity to different transistors within the circuit no longer becomes an issue. Since raw circuit behavior is simulated, we can study the effect of hot-carrier degradation on any aspect of the circuit.

CAS is a successor to SCALE (Substrate Current And Lifetime Evaluator) [8,9,10,53] which computes device-level degradation information such as the I_{sub} waveform and device lifetime. CAS incorporates the structure and models of SCALE: 1) the system is configured in a pre- and post-processor fashion external to SPICE so that no modifications to SPICE is necessary; and 2) transient substrate current waveforms and device lifetimes can still be calculated. In fact, SCALE has become a wholly enclosed subset of CAS, so that SCALE commands will also work in CAS. A new quantity, *Age*, is introduced to quantify the amount of degradation each device suffers. The age includes extracted degradation parameters, the device width, and the substrate and drain currents. Device parameters corresponding to the user-specified future time point are then calculated by comparing the ages calculated for the circuit devices to that of devices at varying degrees of stress with model parameter sets associated with each different stress level. The newly created "aged" model parameters files are then used to simulate the circuit. In these simulations (as well as in the device lifetime simulations), two assumptions are made: 1) the SPICE analysis must be a transient analysis since aging is based on time; and 2) circuit behavior is assumed to be periodic with the period equal to the length of the SPICE analysis (i.e., the waveforms of the input, output, and all internal voltage nodes are assumed to repeat the pattern simulated in the SPICE run up to

the user-defined future time point).

CAS has been configured to use SPICE level 1, 2, and 3 models as well as the two BSIM models (BSIM1 [39] and BSIM2 [45,47])*. Any mixture of the models can be used in the SPICE input deck.

Below, is a general outline for the complete process of simulating the aging of a circuit using CAS.

(1) Extract fresh and stressed process files (a minimum of four).
(2) Extract degradation and I_{sub} and I_{gate} parameters.
(3) Reconcile experimental discrete device data (fresh and stressed) with SPICE I-V characterstics.
(4) Reconcile experimental and simulated I_{sub} and I_{gate}, i.e. confirm the extracted parameters.
(5) Simulate DC stress of discrete devices (n and p channel) and compare the result with experimental ΔI_d versus time in order to confirm the extracted parameters.
(6) Create SPICE deck (and check if circuit works in SPICE).
(7) Add CAS commands to SPICE input deck.
(8) Run BERT by using either the shell-script or line commands.

Sections 3.1 and 3.2 of this report will outline the NMOSFET aging models and device lifetime model, respectively, while Section 3.3 will describe a preliminary degradation and aging model for PMOSFET devices. Section 3.4 contains a CAS simulation example using the commands summarized in Appendix A.

3.1 NMOSFET CIRCUIT AGING AND LIFETIME MODELS

This section describes the models and formulations used to generate the aged model parameters at the user-specified future time point. A new parameter, Age, is introduced to quantify the amount of degradation each device experiences in a circuit environment. This Age parameter is then used as the basis in finding the aged model parameters.

3.1.1 Model Formulation

Device degradation is represented by the change in device model parameters, e.g. $\Delta\beta$, $\Delta\gamma$, etc. Here, we will generalize the degradation by using the symbol ΔD. ΔD may be interchangeably replaced by any of the three or other degradation parameters in the following equations.

Under DC static stressing conditions, the amount of degradation as a function of time is given by Ref[4]

$$\Delta D = f(At) \tag{3.1.1}$$

where

$$A = \frac{I_{ds}}{W}\exp(-\phi_{it}/q\lambda E_m) \tag{3.1.2}$$

*To use the BSIM1 model with SPICE2 requires a special version of SPICE2G.6 [8,30]

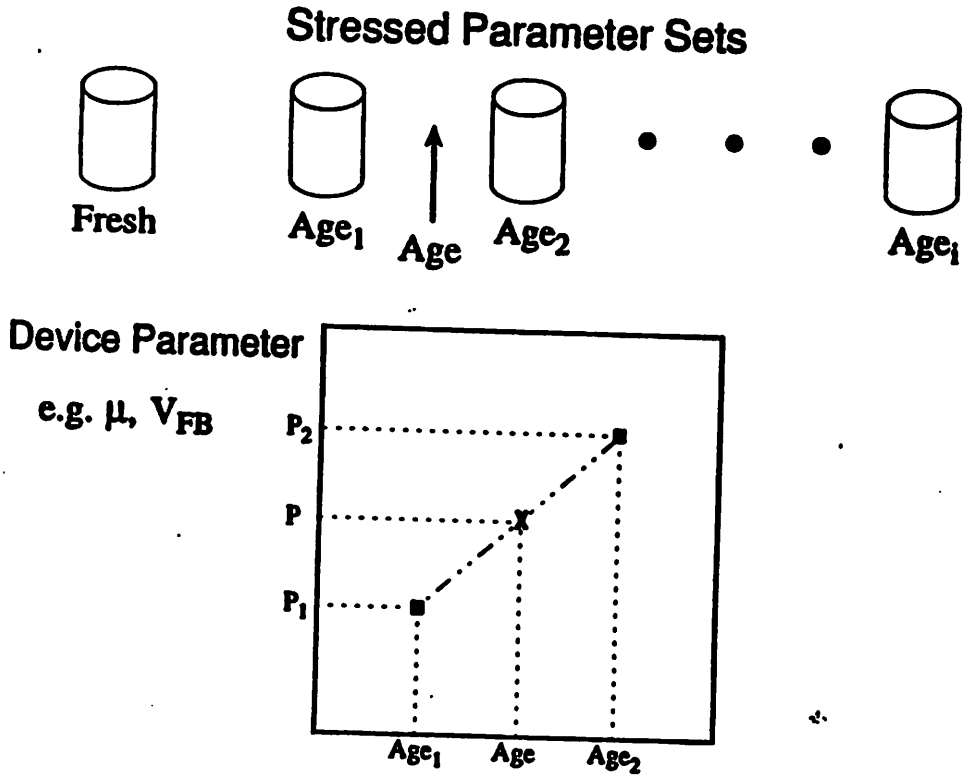


Fig. 3.2 Calculation of the aged parameter from pre-stressed model parameter sets. The barrels represent the various model parameter sets with different ages Age_1, Age_2, etc., while the in-circuit device suffers degradation represented by Age.

where ϕ_{it} is the critical energy required for the creation of interface traps, λ is the electron mean free path, E_m is the maximum lateral channel field, W is the device width, and n and C_1 are dependent on the processing technology. Here, we have generalized the relationship of the degradation by some monotonic function f (the aging concept does not require an explicit form for f, as we shall see shortly). Also from Ref[4],

$$\frac{I_{sub}}{I_{ds}} = C_2 \exp(-\phi_i / q\lambda E_m) \tag{3.1.3}$$

where ϕ_i is the critical energy required for impact ionization and C_2 is a process technology constant. Eq. 3.1.3 can be re-arranged in the following manner:

$$\exp(-\phi_{it}/q\lambda E_m) = \left[\exp(-\phi_i/q\lambda E_m)\right]^{\frac{\phi_{it}}{\phi_i}} = \left[\frac{I_{sub}}{C_2 I_{ds}}\right]^m \tag{3.1.4}$$

$$m = \frac{\phi_{it}}{\phi_i}$$

By substituting the exponential term in Eq. 3.1.2 with Eq. 3.1.4 and merging all constants into the parameter H, we can obtain

$$A = \frac{I_{ds}}{WH}\left[\frac{I_{sub}}{I_{ds}}\right]^m \tag{3.1.5}$$

where m and H are extracted parameters and are dependent on device processing technology. The degradation parameters m and H are also dependent on the gate-drain bias voltage V_{gd} Ref[10,53]. Thus, the expression for device degradation from Eq. 3.1.1 becomes

$$\Delta D = f\left[\frac{I_{ds}}{WH}\left[\frac{I_{sub}}{I_{ds}}\right]^m t\right] \tag{3.1.6}$$

3.1.2 NMOSFET Circuit Aging Model

A new parameter, Age, is now introduced to quantify the amount of degradation each device experiences in a circuit environment. After this parameter has been calculated, it is then used as the basis in finding the aged model for each transistor.

To determine the amount of degradation that occurs in a device, we must look back at the degradation equations above. Since the amount of degradation depends on the stressing condition as well as on time, an Age parameter solely based on time cannot be used. From Eq. 3.1.6, we can generalize an Age variable that is related to the degradation, ΔD:

$$\Delta D = f(Age) \tag{3.1.7}$$

where

$$Age = \int\left(\frac{I_{ds}}{WH}\left[\frac{I_{sub}}{I_{ds}}\right]^m\right)dt$$

Eq. 3.1.7 has all the information necessary - degradation parameters, currents, and time - and is geometry-independent. During circuit simulation, the Age is calculated for each device at each timestep, then integrated to obtain the total Age of the SPICE analysis,

$$Age_T = \int_{t=0}^{T} \frac{I_{ds}}{WH}\left[\frac{I_{sub}}{I_{ds}}\right]^m dt \tag{3.1.8}$$

where T is, as before, the length of the SPICE analysis. The age that each device would have at the user-specified time T_{age} is just

$$Age(T_{age}) = Age_T\left[\frac{T_{age}}{T}\right]$$

The list of ages for every device in the circuit is stored in an external file called "agetable" to be used for the creation of aged model parameters.

To create these aged model parameters, CAS needs a set of model parameter files extracted from the same device but at different levels of degradation. The principle behind the system is as follows:

(1) The user extracts model parameters from a fresh device, followed by extractions of the same device after it has been DC-stressed for different lengths of time.

(2) The user calculates the Age of each of the extracted set of model parameters by using Eq. 3.1.7. This is relatively straightforward since the stressing conditions are known.

(3) CAS simulates the desired circuit and calculates the Age that each device in the circuit would have if the SPICE analysis is repeated up to the user-specified future time point.

(4) CAS compares the Age of each device in the circuit with that of the stressed model parameter files of step (1), and calculates the new aged model parameters of the devices in the circuit by interpolation or regression.

The concept of calculating the aged parameter set is graphically given in Fig. 3.1. The barrels represent the fresh and pre-stressed model parameter files with ages Age_1, Age_2, etc., with the age of the circuit device (calculated by CAS) denoted by Age. Typically the age of the circuit device will lie between two of the pre-stressed model parameter sets. The user has the choice to specify whether interpolation is used (as shown in Fig. 3.1), or whether regression is desired. In both cases, the user also has a choice of whether to perform the analysis in the linear-linear, linear-log, or log-log domain. Generally, log-log is not recommended if the devices in the circuit have very small ages. If the age of the circuit device lies outside the range of the pre-stressed model parameter sets, the two closest parameter sets will be used to extrapolate the age. Extrapolation results in inaccurate output and is not recommended, because we did not assume any functional form for the model parameters versus Age. No explicit function f for Eq. 3.1.7 is assumed.

For BSIM1 and BSIM2 parameter extractions, if multiple device extractions are done to create size-independent process files, every device that will be used to construct one size-independent process file must be subject to the same amount of degradation when doing stressing so that the extraction of each set of stressed process files will consist of devices with the same Age. Since it is difficult to set the same current level for each device, it is recommended that once a current level is set, the stressing time for each device should be varied to obtain the same Age. In general, this method is not recommended because of its complexity.

3.1.3 NMOSFET Device Lifetime Prediction

This section outlines the models used to calculate dynamic NMOSFET device degradation. This is the aging output produced after the first pass of BERT. This model implementation differs from that in SCALE described in Ref[8] in that a bias-dependent n parameter can now be used; however, the basic workings of SCALE have been imbedded into BERT. Sometimes it is desirable to find, for a given stress time, ΔI_d (or ΔV_T) which is not one of the input aged parameters or, conversely, to find lifetime for a specified ΔI_d. If ΔD_f is the amount of degradation e.g. ΔI_d, at which device lifetime, τ, is defined, then for Eq. 3.1.6 and Eq. 3.1.8,

$$\Delta D_f = f(Age_f) \tag{3.1.9}$$

$$\tau = T\frac{Age_f}{Age_T} \tag{3.1.10}$$

For any user specified ΔD_f, BERT will first find Age_f. τ is determined from 3.1.10 and the calculated Age_T, Eq. 3.1.8. Device lifetime is predicted according to Eq. 3.1.10. To find ΔD at a specified time, t_{sp}, BERT generates a rough table of ΔD, e.g. ΔI_d, versus Age, i.e. find the f in Eq. 3.1.9. For device lifetime prediction, f is defined by a exponential function, $\Delta D = At^n$. Once that is done, ΔD_{sp} can be found from the table using $Age(t_{sp}) = Age_t\frac{t_{sp}}{T}$.

3.1.4 Enhanced AC Degradation

There have been several publications describing enhanced hot-carrier degradation from AC effects Ref[6,37,38,12]. These effects can be catagorized into two waveform cases, the "good" and the "bad" case. The "bad" waveform, known to cause the greatest enhanced AC degradation, corresponds to the case where there is a sudden and deep fall of V_{gs} in the presence of high V_{ds} Ref[37,38], while the "good" waveform corresponds to all other cases. Although certain published reports show enhanced degradation at the device level even in the "good" case Ref[6,12], this enhancement does not seem to have a great effect on the calculation of aged model parmaeters on the circuit level for inverter-class waveforms which are classified as the "good" case. We thus believe the quasi-static model presented here is adequate for this "good" class of waveforms (which represent the majority of waveforms encountered in circuits).

In case the "bad" waveform is encountered and the user so specifies, a warning is issued when the following criteria are met:

1) V_{gs} fall > 3 V
2) $V_{ds} - V_{dsat} > 4$ V during the V_{gs} fall.
3) $\frac{\partial V_{gs}}{\partial t} > 10$ V/μs

If this situation is encountered, the models presented in this section (as well as the aging model described in the next section) may be susceptible to underestimation of the hot-carrier degradation. A more recent publication has suggested that the enhanced degradation seen in the "bad" class of waveforms is actually due to capacitively-coupled spikes appearing in the drain voltage during device stressing Ref[51], in which case CAS would be able to predict degradation for both types of waveforms.

3.1.5 Substrate Current Model

The substrate current model is an empirical model developed for the BSIM1 parameter extraction program discussed in Ref[5,50,8]. This model is used for all SPICE models including BSIM2. The following is a summary of the equations and parameters used. Refer to the aforementioned references for more detailed information,

$$I_{sub} = \frac{A_i}{B_i} I_{ds}(V_{ds} - V_{dsat}) \exp\left[-\frac{B_i l_c}{V_{ds} - V_{dsat}}\right]$$

where

$$V_{dsat} = \frac{E_{crit} L (V_{gs} - V_{th})}{E_{crit} L + V_{gs} - V_{th}}$$

$$E_{crit} = E_{crit0} + E_{critg} V_{gs} + E_{critb} V_{bs}$$

and

$$l_c = \sqrt{t_{ox}}\left[l_1 + l_2\left[\frac{1}{V_{gs}+2}\right]\right]$$

$$l_1 = l_{c0} + l_{c1}\left[\frac{1}{V_{bs}-4}\right] + \left[l_{c2} + l_{c3}\left[\frac{1}{V_{bs}-4}\right]\right] V_{ds}$$

$$l_2 = l_{c4} + l_{c5}\left[\frac{1}{V_{bs}-4}\right] + \left[l_{c6} + l_{c7}\left[\frac{1}{V_{bs}-4}\right]\right] V_{ds}$$

$$A_i = \begin{cases} 2\times10^2 \text{ 1/µm for NMOS devices} \\ 1\times10^3 \text{ 1/µm for PMOS devices} \end{cases}$$

$$B_i = \begin{cases} 1.7\times10^2 \text{ V/µm for NMOS devices} \\ 3.4\times10^2 \text{ V/µm for PMOS devices} \end{cases}$$

Thus, there are 11 additional parameters (E_{crit0}, E_{critg}, E_{critb}, l_{c0}, l_{c1}, l_{c2}, l_{c3}, l_{c4}, l_{c5}, l_{c6}, l_{c7}). In its simplest form, however, only l_{c0} and E_{crit0} need to be specified, in which case the other parameters are set to zero and the model simplifies to the physical I_{sub} model Ref[1,27]:

$$I_{sub} = \frac{A_i}{B_i} I_{ds}(V_{ds} - V_{dsat}) \exp\left[-\frac{B_i l_{c0}\sqrt{t_{ox}}}{V_{ds} - V_{dsat}}\right]$$

where

$$V_{dsat} = \frac{E_{crit0} L (V_{gs} - V_{th})}{E_{crit0} L + V_{gs} - V_{th}}$$

See Appendix A for default values.

3.1.6 Device Degradation Parameters

The implementation of the degradation and device lifetime model introduces three parameters, each with two coefficients to model their behavior with respect to V_{gd}. The three parameters, H, m, and n, are implemented as follows:

$$H = H_0 + H_{gd} V_{gd}$$

$$m = m_0 + m_{gd} V_{gd}$$

$$n = n_0 + n_{gd} V_{gd}$$

The parameter n is used only for the device lifetime model. The bias dependence of H and m on V_{gd} implemented here correlate with the results found in Ref[10,53], while the bias dependence of n on V_{gd} has not been experimentally verified and is thus implemented only as an approximation to what it may be in reality. A constant n ($n_{gd} = 0$) value should be used for all aging simulations.

It should be mentioned that these degradation parameters must be extracted by separate device stressing measurements and added manually to the model parameter file when using any SPICE model (BSIM and BSIM2 included). More information on creating the modified model parameter file is given in Appendix A.

3.1.7 Device Stressing Methodology

There are several possible techniques in doing device stressing to extract the device degradation parameters listed in the previous section. The variations concern both the quantity that is monitored, such as I_{sub} or I_{sub}/I_{ds}, and what type of device stressing is used, such as constant voltage, constant I_{sub}, or constant I_{sub}/I_{ds}.

Eq. 3.1.6 of Section 3.1.1 suggests that constant field stressing (constant I_{sub}/I_{ds}) should be used to extract consistent degradation parameters. This condition implies that the rate of degradation is minimally affected by the degradation of device behavior as stressing proceeds Ref[36]. For instance, for the constant voltage case, as the device degrades, the current levels flowing in the device will change with time. Thus, the actual stressing condition, which depends on the electric field in the device, will also change with time.

Eq. 3.1.6 implies that in order to extract m and H, device lifetime should be plotted with the current ratio I_{sub}/I_{ds} rather than I_{sub} alone. By re-arranging Eq. 3.1.6, we can obtain the following expression:

$$\tau\left[\frac{I_{ds}}{W}\right] = Hf^{-1}(\Delta D)\left[\frac{I_{sub}}{I_{ds}}\right]^{-m} \qquad (3.1.11)$$

where ΔD is the amount of degradation.

If we plot Eq. 3.1.11 in log-log format, we can find m from the slope and H from the intercept. This method is preferred since it corresponds directly with theory Ref[4,6], correlates well with device degradation for a wider range of device sizes and stressing biases Ref[42], and relates directly to the amount of interface traps formed Ref[13].

n is the slope when device degradation is plotted against time in log-log format, $\Delta D = At^n$.

Because the parameters m and H are V_{gd} dependent as mentioned in the previous section, devices used to extract one m and H pair should be stressed at the same V_{gd} value. Separate sets of devices should then be stressed at different V_{gd} biases if the V_{gd}-sensitivity terms are desired.

Default values for m are expected to be good enough so that extraction of m is probably unnecessary. The default value for H' is also believed to be acceptable. H_0 then need be the only value to be extracted - an easy task.

3.1.8 Summary

In this section we have introduced the concept of Age to generate degraded model parameters for NMOSFETs for the simulation of circuits at a user-specified future time point. We have also discussed a lifetime prediction model. The next section will describe the equivalent models for the PMOSFET.

3.2 PMOSFET DEGRADATION AND AGING MODELS

This section presents a preliminary PMOS degradation and aging model that is implemented in similar fashion to that of the NMOS device. The difference is that now the gate current I_{gate} also enters the picture as I_{sub} has for the NMOSFET. Because of the disagreement in the literature on whether I_{gate} or I_{sub} correlates better with degradation Ref[14,31,33,35,41,43], we have incorporated both currents through a weighting coefficient that can be specified by the user.

3.2.1 Gate Current Model

The following PMOS gate current model developed by Ref[3] and Ref[14] is used in CAS:

$$I_{gate} = G_1 \frac{I_{sub} t_{ox}}{\lambda_r} \left(\frac{\lambda E_m}{\phi_b} \right)^2 P(E_{ox}) \exp\left(-\frac{\phi_b}{E_m \lambda} \right) \tag{3.2.1}$$

where (3.2.2)

$$P(E_{ox}) = \left[\left(\frac{5.66\times10^{-6} E_{ox}}{1+\frac{E_{ox}}{1.45\times10^{5}}} \right) \times \left(\frac{1}{1+\frac{2\times10^{-3}}{L_{eff}} \exp(-E_{ox} t_{ox}/1.5)} \right) + 2.5\times10^{-2} \right] \exp(-300/\sqrt{E_{ox}})$$

for $E_{ox} \geq 0$, and

$$P(E_{ox}) = 2.5\times10^{-2} \exp(-X_{ox}/\lambda_{ox}) \tag{3.2.3}$$

for $E_{ox} \leq 0$. $P(E_{ox})$ is essentially the probability that a scattered electron will surmount the oxide barrier and flow to the gate. $G_1 = 0.5$, $\lambda_{ox} = 320$ Å, $\lambda_r = 616$ Å is the re-direction scattering mean free path, and $\lambda = 105$ Å is the scattering mean free path of the electron Ref[14]. The oxide barrier height ϕ_b can be expressed by

$$\phi_b = 3.2 - 2.6\times10^{-4}\sqrt{E_{ox}} - \upsilon E_{ox}^{2/3} \tag{3.2.4}$$

where $\upsilon = 4\times10^{-5}$ in Ref[3]. The maximum electric field E_m can be described by the same formulation as that used for the substrate current in Section 3.1:

$$E_m = \frac{V_{ds} - V_{dsatg}}{l_{cg}}$$

where we have used the extra subscript "g" in V_{dsat} and I_c to differentiate them with those used in the substrate current. V_{dsatg} and I_{cg} can be again decomposed into the following parameters:

$$V_{dsatp} = \frac{E_{critp} L (V_{gs} - V_{th})}{E_{critp} L + V_{gs} - V_{th}}$$

$$E_{critp} = E_{critp0} + E_{critpg} V_{gs} + E_{critpb} V_{bs} \tag{3.2.5}$$

$$l_{cg} = \sqrt{t_{ox}} \left[l_{1g} + l_{2g} \left(\frac{1}{V_{gs}+2} \right) \right]$$

$$I_{1g} = I_{cg0} + I_{cg1}\left[\frac{1}{V_{bs}-4}\right] + \left[I_{cg2} + I_{cg3}\left[\frac{1}{V_{bs}-4}\right]\right] V_{ds} \tag{3.2.6}$$

$$I_{2g} = I_{cg4} + I_{cg5}\left[\frac{1}{V_{bs}-4}\right] + \left[I_{cg6} + I_{cg7}\left[\frac{1}{V_{bs}-4}\right]\right] V_{ds} \tag{3.2.7}$$

Recent studies have shown that this lucky-electron model can predict I_{gate} for both buried-channel (n+ poly gate) and surface-channel (p+ poly gate) PMOSFET's Ref[48,52].

3.2.2 Degradation and Aging Model Based on I_{gate}

This section outlines the models used to calculate dynamic PMOSFET By slightly modifying the expression from Ref[14] and paralleling Eq. 3.1.7, we can write for a PMOSFET,

$$\Delta D = f\left[\frac{1}{H}\left[\frac{I_g}{W}\right]^m t\right] \tag{3.2.8}$$

As for the Age expression, by looking at Eq. 3.2.8, we can parallel the NMOS analysis and propose the following generalized expression for Age,

$$\Delta D = f(\text{Age})$$

Then,

$$\text{Age} = \int \frac{1}{H}\left[\frac{I_{gate}}{W}\right]^m dt \tag{3.2.9}$$

3.2.3 Incorporation of I_{sub} and I_{gate} in Predicting Degradation

To conglomerate the substrate current and gate current degradation models, we can sum the contributions from each component linearly through weighting coefficients (W_g,W_b = 1 - W_g) that can be specified by the user,

$$\text{Age} = W_b \times \left[\text{Age from } I_{sub}\right] + W_g \times \left[\text{Age from } I_{gate}\right] \tag{3.2.10}$$

For the following, H_b and m_b denote the H and m parameter for I_{sub}, while H_g and m_g denote the H and m parameter associated with I_{gate}. Note that the n parameter is the same for both cases (since n depends only on the degradation behavior with time and not on what currents are used as a basis for degradation). Then, the following equation can be derived for the age:

$$\text{Age} = W_b \int \frac{I_{ds}}{WH_b}\left[\frac{I_{sub}}{I_{ds}}\right]^{m_b} dt + W_g \int \frac{1}{H_g}\left[\frac{I_{gate}}{W}\right]^{m_g} dt \tag{3.2.11}$$

Then $\Delta D = f(\text{Age})$ with Age defined as Eq 3.2.11.

Eq. 3.2.10 and 3.2.11 are then used in the degradation and aging calculations discussed in Sections 3.1.2 and 3.1.3.

3.2.4 PMOSFET Device Lifetime

This section outlines the models used to calculate PMOSFET lifetimes. The PMOS degradation and aging models closely parallel that of the NMOS case. For a given stress time, CAS can determine ΔI_d (or ΔV_T) or, conversely, find lifetime for a specified ΔI_d. If ΔD_f is the amount of degradation e.g. ΔI_d, at which device lifetime, τ, is defined, then Eq. 3.1.9 and Eq. 3.1.10 are also true for the PMOSFET.

For any user specified ΔD_f, BERT will first find Age_f. τ is determined from 3.1.10 and the calculated Age_T, integrating Eq.3.2.11 over period T. Device lifetime is predicted according to Eq. 3.1.10. To find ΔD at a specified time, t_{sp}, BERT generates a rough table of ΔD, e.g. ΔI_d, versus Age, i.e. find the f in Eq. 3.1.9. For device lifetime prediction, f is defined by a exponential function, $\Delta D = At^n$. Once that is done, ΔD_{sp} can be found from the table using $Age(t_{sp}) = Age_t \frac{t_{sp}}{T}$.

3.2.5 Parameters Necessary for Simulation

Because the I_{gate} model involves I_{sub} and E_m, substrate current parameters must be extracted. At present, we have made the following parameters user-specifiable: G_1 (Eq. 3.2.1), υ (Eq. 3.2.4), E_{critp0}, E_{critpg}, E_{critpb}, l_{cg0}, l_{cg1}, l_{cg2}, l_{cg3}, l_{cg4}, l_{cg5}, l_{cg6}, and l_{cg7} (Eqs. 3.2.5 - 3.2.7). The E_{critp} and l_{cg} parameters default to corresponding E_{crit} and l_c parameters specified for the substrate current.

As for the degradation parameters, H_g, m_g, and n should be extracted in similar manner as in the NMOS case. n is the slope when device degradation is plotted against time in log-log format ($\Delta D = At^n$).

As in the NMOS case, we can rearrange Eq. 3.2.8 and obtain the following expression:

$$\tau = Hf^{-1}(\Delta D_f)\left(\frac{I_{gate}}{W}\right)^{-m} \tag{3.2.12}$$

where the gate current is normalized by the device width W. Let us assume that the PMOS degradation follows the same power-law behavior as for the NMOS device. Then,

$$\Delta D = At^n \tag{3.2.13}$$

Denoting ΔD_f as the degradation level defining the device lifetime τ as before, we obtain

$$\Delta D_f = A\tau^n \tag{3.2.14}$$

Solving for the coefficient A using Eqs. 3.2.8 and 3.2.10, we get

$$A = \frac{\Delta D_f}{B^n}\left[\frac{I_{gate}}{W}\right]^{mn} \tag{3.2.15}$$

where

$$H = \frac{B}{\Delta D_f^{1/n}} \tag{3.2.16}$$

$-m_g$ and B_g are the slope and intercept respectively when device lifetime τ is plotted against I_{gate} in log-log format (Eq. 3.2.8). Using Eq. 3.2.16, B_g must then be converted to H_g to remove the dependency of the parameter set to the level of degradation defined at the device lifetime (Eq. 3.2.9). H_g and m_g are further divided into a constant and V_{gd}-sensitivity term as in the I_{sub} case:

$$H_g = H_{g0} + H_{ggd}V_{gd}$$

$$m_g = m_{g0} + m_{ggd}V_{gd}$$

Thus, to summarize, the following parameters must be added to the model parameter set to simulate PMOS degradation:

1) G_1: G1: constant coefficient for I_{gate} (default = 0.5)Ref[3].
2) υ: UPS: sensitivity of ϕ_b to the $E_{ox}^{2/3}$ term (default = 4×10^{-5} $V^{1/3}cm^{2/3}$).
3) E_{critp0}: ECRITP0: Constant term of E_{critp} (default = E_{crit0} of I_{sub}).
4) E_{critpg}: ECRITPG: V_{gs} depenedence of E_{critp} (default = E_{critg} of I_{sub}).
5) E_{critpb}: ECRITPB: V_{bs} depenedence of E_{critp} (default = E_{critb} of I_{sub}).
6) l_{cg0}: LCG0: Constant term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c0} of I_{sub}).
7) l_{cg1}: LCG1: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c1} of I_{sub}).
8) l_{cg2}: LCG2: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c2} of I_{sub}).
9) l_{cg3}: LCG3: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c3} of I_{sub}).
10) l_{cg4}: LCG4: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c4} of I_{sub}).
11) l_{cg5}: LCG5: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c5} of I_{sub}).
12) l_{cg6}: LCG6: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c6} of I_{sub}).
13) l_{cg7}: LCG7: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c7} of I_{sub}).
14) H_{g0}: HG0: intercept parameter of the lifetime versus I_{gate} plot (default = 10^4).
15) H_{ggd}: HGGD: V_{gd}-sensitivity term for H_g (default = 0).
16) m_{g0}: MG0: slope parameter of the lifetime versus I_{gate} plot (default = 1.5)Ref[14].
17) m_{ggd}: MGGD: V_{gd}-sensitivity term for m_g (default = 0).
18) W_g: WG: weighting coefficient for I_{gate}-based degradation (default = 0 or 1).
19) V_{fbg}: VFBG: Flat-band voltage for PMOS Gate current calculation.

Unlike the other model and degradation parameters, the PMOS I_{gate} and degradation parameters are declared in the input deck using the '.pmosdeg' command (See Appendix A). The default value of W_g is 1 if the '.pmosdeg' command is specified; otherwise W_g defaults to 0.

3.2.6 Summary

This section has introduced a preliminary PMOS degradation and aging model that parallels that of the NMOSFET case. The next section will present an example of a BERT deck running the CAS module.

3.3 CIRCUIT EXAMPLE: 21-STAGE CMOS INVERTER CHAIN

In Appendix A, the CAS commands are described. In this section, we will describe an example of a BERT deck which uses the CAS commands to invoke the CAS module.

Fig. 3.4.1 shows a SPICE3 input deck for a 21-stage CMOS inverter chain circuit with a 100 MHz clocked input and 0.1pF capacitive loading at each inverter output (sample input file located in the bert/CAS/Sample/ directory). For this example circuit, the period of the input waveform is 10 ns. However, to accurately predict circuit degradation, we need to make the SPICE analysis long enough for the signal to propagate through the last stage of the inverter chain. Thus, the SPICE analysis is doubled to 20ns, with no additional signal being inputted during the extra time. This effectively means that, for this particular case, we also need to double the ages we specify for the '.age' and '.agedid'-type of commands if we want to simulate a periodic waveform of 10ns. Thus, in this case, although we want circuit degradation at 10 years in the future, we need to specify 20 years for the commands. Also, the device lifetime results calculated by the simulator will need to be halved to obtain the correct value. Fig. 3.4.2 shows various degradation information for the NMOS (M202) and PMOS (M201) transistor of the 20th stage. As an example, to correctly interpret the results, the lifetime of M202 is 2.4 + 2 = 1.2 years, and $\Delta I_{ds}/I_{ds0} = 17.1\%$ after 10 years of operation. Fig. 3.4.3 shows the generated agetable with the Age that all the transistors would have after 10 years of operation. Fig. 3.4.4 shows the output waveform of the 20th inverter stage comparing the propagation delay difference between the fresh and 10-year aged inverter chain. As expected, with device degradation, propagation delay is longer for the aged case.

3.4 CONCLUSION

We have presented a hot-electron reliability simulator CAS which is a part of the BERT reliability simulator system. Used in conjunction with the SPICE circuit simulator, CAS can calculate various degradation information for individual devices in a circuit undergoing dynamic operation. For instance, by using the device lifetime option, hot spots in the circuit can be easily pinpointed. More importantly, CAS can predict the behavior of circuits that have undergone hot-carrier degradation for a user-specified length of time. With this tool, VLSI design engineers will be able to better understand the degradation and reliability performance of their circuits.

```
CMOS CLOCKED INVERTER CHAIN (21 STAGES)
*
* Power Supplies and Input Pulse.
*
vdd 40 0 dc 5.5
vin 1 0 pwl(0 0 0.02ns 5.5 5ns 5.5 5.2ns 0 )
vmeas 50 0 dc 0
*
* The Inverter Chain
*
m1 2 1 40 40 PC1_pm1_du2 w=60u L=1.4u
m2 2 1 0 0 PC2_nm1_du1 W=20u L=1.4u
m21 3 2 40 40 PC1_pm1_du2 w=60u L=1.4u
m22 3 2 0 0 PC2_nm1_du1 W=20u L=1.4u
m31 4 3 40 40 PC1_pm1_du2 w=60u L=1.4u
m32 4 3 0 0 PC2_nm1_du1 W=20u L=1.4u
m41 5 4 40 40 PC1_pm1_du2 w=60u L=1.4u
m42 5 4 0 0 PC2_nm1_du1 W=20u L=1.4u
m51 6 5 40 40 PC1_pm1_du2 w=60u L=1.4u
m52 6 5 0 0 PC2_nm1_du1 W=20u L=1.4u
m61 7 6 40 40 PC1_pm1_du2 w=60u L=1.4u
m62 7 6 0 0 PC2_nm1_du1 W=20u L=1.4u
m71 8 7 40 40 PC1_pm1_du2 w=60u L=1.4u
m72 8 7 0 0 PC2_nm1_du1 W=20u L=1.4u
m81 9 8 40 40 PC1_pm1_du2 w=60u L=1.4u
m82 9 8 0 0 PC2_nm1_du1 W=20u L=1.4u
m91 10 9 40 40 PC1_pm1_du2 w=60u L=1.4u
m92 10 9 0 0 PC2_nm1_du1 W=20u L=1.4u
m101 11 10 40 40 PC1_pm1_du2 w=60u L=1.4u
m102 11 10 0 0 PC2_nm1_du1 W=20u L=1.4u
m111 12 11 40 40 PC1_pm1_du2 w=60u L=1.4u
m112 12 11 0 0 PC2_nm1_du1 W=20u L=1.4u
m121 13 12 40 40 PC1_pm1_du2 w=60u L=1.4u
m122 13 12 0 0 PC2_nm1_du1 W=20u L=1.4u
m131 14 13 40 40 PC1_pm1_du2 w=60u L=1.4u
m132 14 13 0 0 PC2_nm1_du1 W=20u L=1.4u
m141 15 14 40 40 PC1_pm1_du2 w=60u L=1.4u
m142 15 14 0 0 PC2_nm1_du1 W=20u L=1.4u
m151 16 15 40 40 PC1_pm1_du2 w=60u L=1.4u
m152 16 15 0 0 PC2_nm1_du1 W=20u L=1.4u
m161 17 16 40 40 PC1_pm1_du2 w=60u L=1.4u
m162 17 16 0 0 PC2_nm1_du1 W=20u L=1.4u
m171 18 17 40 40 PC1_pm1_du2 w=60u L=1.4u
m172 18 17 0 0 PC2_nm1_du1 W=20u L=1.4u
m181 19 18 40 40 PC1_pm1_du2 w=60u L=1.4u
m182 19 18 0 0 PC2_nm1_du1 W=20u L=1.4u
m191 20 19 40 40 PC1_pm1_du2 w=60u L=1.4u
m192 20 19 50 0 PC2_nm1_du1 W=20u L=1.4u
m201 21 20 40 40 PC1_pm1_du2 w=60u L=1.4u
```

Fig. 3.4.1 SPICE3 input deck for a 21-stage CMOS inverter chain with substrate current, gate current, device lifetime, and circuit aging calculations requested (continued on next two pages).

```
m202 21 20 0 0 PC2_nm1_du1 W=20u L=1.4u
m211 22 21 40 40 PC1_pm1_du2 w=60u L=1.4u
m212 22 21 0 0 PC2_nm1_du1 W=20u L=1.4u
*
* Capacitive Loading.
*
c2 2 0 0.1pF
c3 3 0 0.1pF
c4 4 0 0.1pF
c5 5 0 0.1pF
c6 6 0 0.1pF
c7 7 0 0.1pF
c8 8 0 0.1pF
c9 9 0 0.1pF
c10 10 0 0.1pF
c11 11 0 0.1pF
c12 12 0 0.1pF
c13 13 0 0.1pF
c14 14 0 0.1pF
c15 15 0 0.1pF
c16 16 0 0.1pF
c17 17 0 0.1pF
c18 18 0 0.1pF
c19 19 0 0.1pF
c20 20 0 0.1pF
c21 21 0 0.1pF
c22 22 0 0.1pF
*
* Numerical Control.
*
.nodeset v(1)=0 v(2)=5 v(3)=0 v(4)=5 v(5)=0 v(6)=5
+ v(7)=0 v(8)=5 v(9)=0
+ v(10)=5 v(11)=0  v(12)=5 v(13)=0 v(14)=5 v(15)=0
+ v(16)=5 v(17)=0 v(18)=5 v(19)=0 v(20)=5 v(21)=0
*
* For uniform aging of all transistors, the period is 10ns, but
* the SPICE time window is 20ns long to allow the pulse to clear
* the last transistor. The age is modified accordingly so that 10years
* of aging is equivalent to 20years in the SPICE input deck.
*
.tran 0.02ns 20ns
*
* Output Control.
*
.print tran v(21)
.width out=80
*
* Model parameter file declarations.
*
.process PC1 filename = PM0OUT
.process PC2 filename = NM0OUT
```

Fig. 3.4.1 (cont) SPICE3 input deck for a 21-stage CMOS inverter chain with substrate current, gate current, device lifetime, and circuit aging calculations requested.

```
.pmosdeg PC1_pm1_du2 g1=0.7 ups=5e-5 hg0=1e3 mg0=1.5 wg=1
*
* Isub, Igate and lifetime commands.
*
.isubwidth=80
.deltaid 0.1
.agedid 20years
.plotisub m202 all
.plotigate m201
*
* Aging Commands.
*
.agemethod interp linlog
.age 20years
.ageproc PC2 filenames = NM0OUT, NM1OUT, NM2OUT, NM3OUT, NM4OUT
.end
```

Fig. 3.4.1 (cont) SPICE3 input deck for a 21-stage CMOS inverter chain with substrate current, gate current, device lifetime, and circuit aging calculations requested.

```
-------------------------------------------------------------------------
|
|       DEVICE DEGRADATION INFORMATION : TRANSISTOR M201
|
-------------------------------------------------------------------------
|
| AVERAGE IDRAIN                                =    1.8183675e-04 A
| MAXIMUM IDRAIN                                =    5.0063941e-03 A
| AVERAGE ISUB                                  =    1.1241229e-06 A
| MAXIMUM ISUB                                  =    1.2877306e-04 A
| AVERAGE IGATE                                 =    5.8945641e-10 A
| MAXIMUM IGATE                                 =    1.0475510e-07 A
| DELTA ID / ID0 IN THE FIRST TIME PERIOD       =    1.3104644e-12
|
-------------------------------------------------------------------------
|  DEVICE LIFETIME AT DELTA ID / ID0 = 0.1:
-------------------------------------------------------------------------
|
|   >>>>>> TAU(m201) = 3.75 YEARS (1.183e+08 SEC.) <<<<<<
|
-------------------------------------------------------------------------
|  DEGRADATION OF m201 AT 6.31152e+08 SEC. (20 YEARS):
-------------------------------------------------------------------------
|
|   >>>>>> DELTA ID0/ID = 0.317546 <<<<<<
|
-------------------------------------------------------------------------

-------------------------------------------------------------------------
|
|       DEVICE DEGRADATION INFORMATION : TRANSISTOR M202
|
-------------------------------------------------------------------------
|
| AVERAGE IDRAIN                                =    1.6686596e-04 A
| MAXIMUM IDRAIN                                =    4.2653863e-03 A
| AVERAGE ISUB                                  =    1.5277986e-07 A
| MAXIMUM ISUB                                  =    1.5201352e-05 A
| DELTA ID / ID0 IN THE FIRST TIME PERIOD       =    1.0239075e-05
|
-------------------------------------------------------------------------
|  DEVICE LIFETIME AT DELTA ID / ID0 = 0.1:
-------------------------------------------------------------------------
|
|   >>>>>> TAU(m202) = 2.439 YEARS (7.69e+07 SEC.) <<<<<<
|
-------------------------------------------------------------------------
|  DEGRADATION OF m202 AT 6.31152e+08 SEC. (20 YEARS):
-------------------------------------------------------------------------
|
|   >>>>>> DELTA ID0/ID = 0.171408 <<<<<<
|
-------------------------------------------------------------------------
```

Fig. 3.4.2 **Degradation information of the NMOS and PMOS devices of the 20th stage of the 21-stage inverter chain.**

Device Name	Model Name	Age
m1	pc1 pm1 du2	1.311304e-01
m2	pc2 nm1 du1	4.256910e-05
m21	pc1 pm1 du2	2.296307e-01
m22	pc2 nm1 du1	9.017232e-04
m31	pc1 pm1 du2	2.438348e-01
m32	pc2 nm1 du1	8.378833e-04
m41	pc1 pm1 du2	2.559721e-01
m42	pc2 nm1 du1	1.320404e-03
m51	pc1 pm1 du2	2.536003e-01
m52	pc2 nm1 du1	1.275502e-03
m61	pc1 pm1 du2	3.459318e-01
m62	pc2 nm1 du1	1.329472e-03
m71	pc1 pm1 du2	2.503702e-01
m72	pc2 nm1 du1	1.511937e-03
m81	pc1 pm1 du2	2.621047e-01
m82	pc2 nm1 du1	1.417721e-03
m91	pc1 pm1 du2	2.982601e-01
m92	pc2 nm1 du1	7.035680e-04
m101	pc1 pm1 du2	2.524995e-01
m102	pc2 nm1 du1	1.398250e-03
m111	pc1 pm1 du2	2.319539e-01
m112	pc2 nm1 du1	7.917197e-04
m121	pc1 pm1 du2	2.778566e-01
m122	pc2 nm1 du1	1.443027e-03
m131	pc1 pm1 du2	2.727511e-01
m132	pc2 nm1 du1	1.315843e-03
m141	pc1 pm1 du2	2.426205e-01
m142	pc2 nm1 du1	1.384947e-03
m151	pc1 pm1 du2	1.877004e-01
m152	pc2 nm1 du1	1.509574e-03
m161	pc1 pm1 du2	1.901972e-01
m162	pc2 nm1 du1	1.370279e-03
m171	pc1 pm1 du2	3.457375e-01
m172	pc2 nm1 du1	1.317324e-03
m181	pc1 pm1 du2	1.940385e-01
m182	pc2 nm1 du1	1.795126e-03
m191	pc1 pm1 du2	2.284004e-01
m192	pc2 nm1 du1	1.310316e-03
m201	pc1 pm1 du2	1.896623e-01
m202	pc2 nm1 du1	1.018439e-03
m211	pc1 pm1 du2	3.286558e-01
m212	pc2 nm1 du1	1.817707e-03

Fig. 3.4.3 The agetable generated by CAS of all the transistors in the circuit.

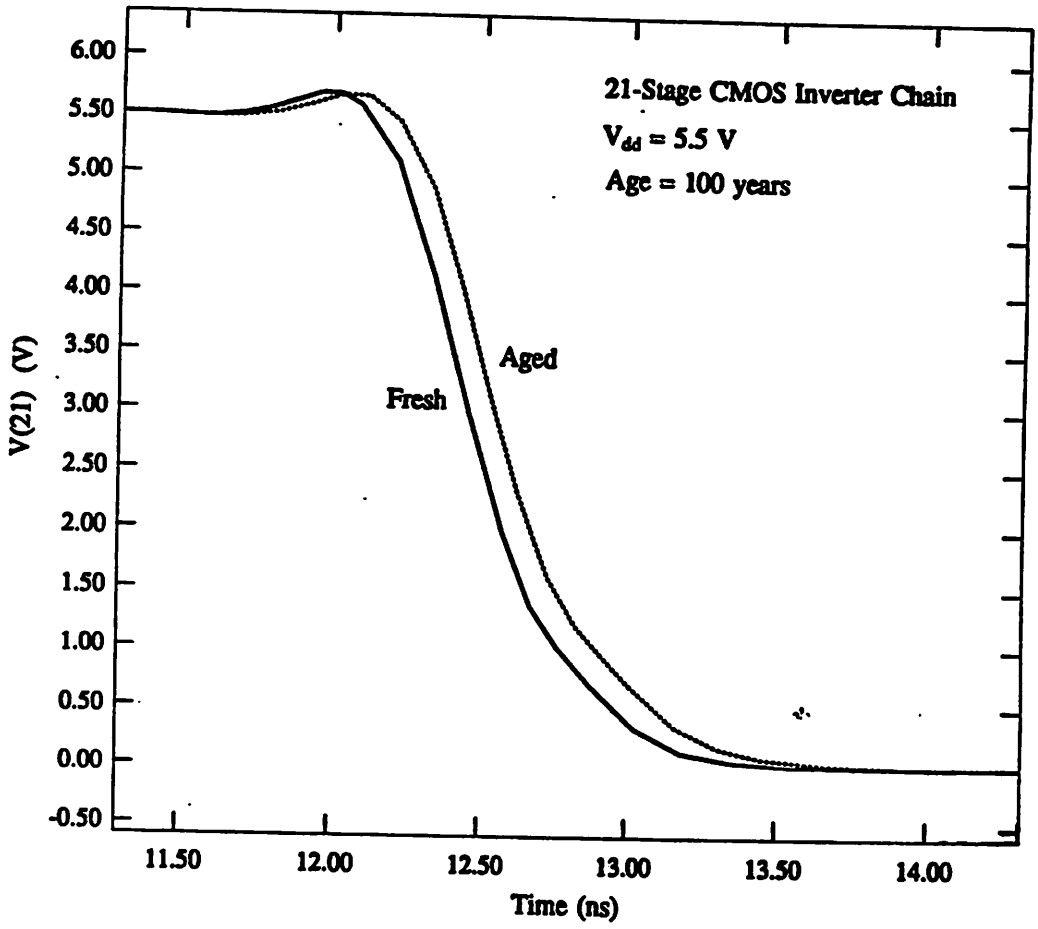


Fig. 3.4.4 The voltage waveform at the output of the 20th stage showing the propagation delay difference between the fresh and aged inverter chain.

4. Circuit Oxide Reliability Simulator - (CORS)

CORS (Circuit Oxide Reliability Simulator) is a fully integrated part of BERT (BErkeley Reliability Tools). CORS projects the probability of oxide breakdown induced circuit failure as a function of operating time, temperature, power supply voltage and input waveforms. CORS can also simulate the effects of burn-in on subsequent yield and lifetime. The user is required to provide the simulator with test capacitor breakdown statistics. Either SPICE or IRSIM may be used as the circuit simulator invoked between calls to the BERT pre-processor and post-processor, although only SPICE can be used if CAS, EM or BiCAS are being called concurrently.

Appendix C should be reviewed by the user unfamiliar with the capabilities of CORS. This appendix provides the user with several examples of CORS output selected to demonstrate CORS's various features. Section 4.1 discusses the equations for time-dependent dielectric breakdown (TDDB) implemented in CORS. Section 4.2 describes the procedure for obtaining test capacitor breakdown data and also details the use of programs DEFECT and EMODEL which are provided to the user along with BERT for the purpose of formatting this breakdown data so that it can be read by BERT. Section 4.3 contains examples of BERT input decks containing CORS commands and the corresponding output listings. Appendix A contains the user's guide for all of the CORS commands. The user who needs information on executing BERT, should consult Chapter 2 of this manual.

4.1 CIRCUIT FAILURE MODEL AND IMPLEMENTATION

4.1.1 Oxide Breakdown Model

The default model for oxide breakdown in CORS is based on [56] and is referred to as the "$\frac{1}{E}$ model." An alternate model based on [61] has also been implemented in CORS; this model is called the "E model" and is described in section 4.1.6.

According to the $\frac{1}{E}$ model, oxide intrinsic lifetime (t_{BD}) is

$$t_{BD} = \tau \exp\left[\frac{GX_{ox}}{V_{ox}}\right] \tag{4.1}$$

where X_{ox} is oxide thickness, V_{ox} is oxide voltage, G and τ are proportionality constants. Large area capacitors generally experience "defect-related" breakdown which causes them to have a non-deterministic lifetime. Defect-related breakdown is modeled as [57]

$$t_{BD} = \tau \exp\left[\frac{GX_{eff}}{V_{ox}}\right]. \tag{4.2}$$

Note that the random variable X_{eff} has been substituted for X_{ox} in the above expression. The probability that a device undergoes oxide breakdown before a given t_{BD} is related to the probability that the oxide contains a specific value of X_{eff}. This is discussed in greater depth later in this section.

Using a quasi-static approach, we have extended this model for use with time varying voltages [58],

$$1 = \frac{1}{\tau}\int_0^{t_{BD}} \exp\left[\frac{-GX_{eff}}{V_{ox}(t)}\right] dt. \tag{4.3}$$

Inside CORS, $V_{ox}(t)$ is derived from SPICE or IRSIM node The integral in (4.3) is evaluated numerically using the Trapezoidal Rule (following a SPICE simulation) or analytically (following an IRSIM simulation). The integral is not actually evaluated for large t_{BD}, such as 10 years; instead, it is evaluated for the period, T, of SPICE (or IRSIM) simulation and then multiplied by a factor of $\frac{t_{BD}}{T}$. This places a constraint on the simulation results; they are derived assuming repetitive input waveforms.

4.1.2 Temperature Dependence of Oxide Breakdown

Time-to-breakdown (t_{BD}) decreases with increasing operating temperature. The G and τ parameters in (4.3) are temperature dependent to account for this effect [59]. Specifically, G and τ are defined as follows.

$$G(T) = G_{300}\left[1 + \frac{\delta}{k_B}\left[\frac{1}{T} - \frac{1}{300}\right]\right] \tag{4.4}$$

$$\tau(T) = \tau_{300}\exp\left[\frac{-E_b}{k_B}\left[\frac{1}{T} - \frac{1}{300}\right]\right] \tag{4.5}$$

T has units of K°. Default values of G_{300}, τ_{300}, δ and E_b are provided or the user may provide values derived from intrinsic oxide studies of his/her technology (Appendix A).

4.1.3 Determination of $V_{ox}(t)$

The voltage drop across a capacitor oxide is set equal to

$$V_{ox}(t) = V_1(t) - V_2(t)$$

where $V_1(t)$ and $V_2(t)$ are the SPICE node voltages for the capacitor electrodes. It is assumed that both electrodes are made from similar materials (so there is no work function difference) and that both electrodes are highly conductive (so there is no voltage drop across a depletion region). These are reasonable assumptions for VLSI capacitors.

The voltage drop across a MOSFET oxide is not as straightforward to determine. The voltage drop across the substrate depletion region must be accounted for and work function differences must be considered. Furthermore, when current flows along the channel, the surface potential becomes a function of position and, consequently, so does the oxide electric field. In this release of CORS, the position dependence of the oxide field along a MOSFET channel is neglected and the oxide voltage is set equal to the maximum of V_{gs} and V_{gd}, minus the depletion region voltage drop. Specifically, the oxide voltage along the channel is calculated as follows.

N-channel MOSFET

Inversion ($V_{gs} > V_t$):
Surface potential is near the conduction band edge when there is appreciable gate current. Therefore,

$$V_{ox} = | V_{gs} - \phi_{ms} - \frac{E_g}{2} - \frac{\phi}{2} |$$

where ϕ equals the SPICE parameter PHI (surface potential) and ϕ_{ms} is the work function difference between the gate and substrate materials.

Accumulation ($V_g - V_b < V_{fb}$):
Surface potential is near the valence band edge when there is appreciable gate current. Therefore,

$$V_{ox} = | V_g - V_b - \phi_{ms} + \frac{E_g}{2} - \frac{\phi}{2} |$$

Depletion:
$V_{ox} \simeq 0.0$

P-channel MOSFET

Inversion ($V_{gs} < V_t$):
$$V_{ox} = | V_{gs} - \phi_{ms} + \frac{E_g}{2} + \frac{\phi}{2} |$$

Accumulation ($V_g - V_b > V_{fb}$):
$$V_{ox} = | V_g - V_b - \phi_{ms} - \frac{E_g}{2} + \frac{\phi}{2} |$$

Depletion:
$V_{ox} \simeq 0.0$

Examination of these equations will show that the difference between the applied and the oxide voltage is zero or E_g, depending on the gate material employed and the operating regime. Additionally, the oxide voltage along the device edges must be calculated for the user who provides the simulator with statistics for defect density per unit length along diffusion and field oxide edges. V_{ox} along the field oxide edges is identical to that in the channel. The field at the source/drain edges is different from that in the channel because these regions have a different surface potential from the channel region due to the different carrier concentrations and types. The voltage at a diffusion, (say, drain) edge is calculated as follows.

N-channel MOSFET

n+ poly gate:
$$V_{ox} = | V_g - V_d |$$

p+ poly gate:
$$V_{ox} = | V_d - V_g + E_g |$$

P-channel MOSFET

p+ poly gate:
$$V_{ox} = | V_g - V_d |$$

n+ poly gate:
$$V_{ox} = | V_g - V_d + E_g |$$

4.1.4 Calculation of X_{eff}

For a given time-to-breakdown (t_{BD}), X_{eff} is calculated from (4.3) using Newton-Raphson's method (following a SPICE simulation) or directly (following an IRSIM simulation). The X_{eff} thus derived is the maximum thickness (least severe defect) which is predicted to fail by t_{BD}. This calculation is repeated for eight different values of t_{BD}. When SPICE has been used and both integration and iteration are required to solve for X_{eff} then these calculations take up the bulk of the CORS computation time. An alternative algorithm, called "quick," has been added for the user who wishes to reduce execution time and is willing to sacrifice some accuracy (about 2% - see Section 4.3). The quick algorithm replaces an expression of the form f ($V_{ox}(t)$) with an

expression of the form $A*f(V_{max})$. When "quick" has been specified, CORS calculates X_{eff} corresponding the smallest t_{BD} from (4.3); these are denoted X_{eff}^0 and t_{BD}^0. The remaining X_{eff}'s are calculated directly using

$$1 = \frac{t_{BD}^i}{t_{BD}^0} \exp\left[\frac{G}{V_{max}} (X_{eff}^0 - X_{eff}^i) \right]. \tag{4.6}$$

4.1.5 Calculation of t_{BD} after Burn-In

Circuit lifetime after burn-in may be projected by using CORS in a two-pass mode (Appendix A). During the first pass, SPICE (or IRSIM) provides CORS with the node voltages during burn-in; during the second pass, SPICE (or IRSIM) provides CORS with the node voltages under normal operating conditions. If burn-in is performed, the expression for predicted lifetime, (4.3), must be modified to account for oxide wear incurred during the burn-in process [60].

$$1 = \frac{1}{\tau_{op}} \int_0^{t_{op}} \exp\left[\frac{-G_{op} X_{eff}}{V_{ox}(t)} \right] dt + \frac{1}{\tau_{bi}} \int_0^{t_{bi}} \exp\left[\frac{-G_{bi} X_{eff}}{V_{ox}(t)} \right] dt \tag{4.7}$$

This equation is implemented in CORS using an approximation to save storage space and computation time. With the approximation, the equation becomes

$$1 = \frac{1}{\tau_{op}} \int_0^{t_{op}} \exp\left[\frac{-G_{op} X_{eff}}{V_{ox}(t)} \right] dt + \exp\left[\frac{G_{bi}}{V_{max}^{bi}} (X_{bi} - X_{eff}) \right] \tag{4.8}$$

where X_{bi} is the maximum X_{eff} screened out by the burn-in trial (calculated during pass one of CORS). (4.8) is solved for X_{eff} using Newton-Raphson's method.

The user may choose to invoke a "quick" algorithm, rather than using (4.8), to calculate X_{eff}. Accuracy is sacrificed, typically on the order of 50% (see Section 4.3). This loss of accuracy will be acceptable the user who is only interested in an order-of-magnitude estimate of failure probability. "Quick" always errs on the conservative side. If "quick" has been invoked, the simulator calculates X_{bi} during the first pass. It also calculates a X_{no_bi} during the second pass, which is the X_{eff} that would correspond to t_{BD}^0 if no burn-in had been performed. CORS proceeds to calculate X_{eff} using

$$1 = \frac{t_{BD}^i}{t_{BD}^0} \exp\left[\frac{G_{op}}{V_{max}} (X_{no\_bi} - X_{eff}) \right] + \exp\left[\frac{G_{bi}}{V_{max}^{bi}} (X_{bi} - X_{eff}) \right]. \tag{4.9}$$

4.1.6 Failure Probability Calculations

CORS assumes that defects are distributed uniformly and independently across the test wafers and actual circuits. This allows the use of the Poisson distribution to describe the defect density (see Section 4.2). The probability that a device fails at or before a specified t_{BD} is equal to the probability that the device contains one or more defects of size X_{eff} or smaller, where X_{eff} was calculated from (4.3) for the specified t_{BD}. Using the Poisson distribution, this probability may be expressed as

$$P(\text{failure}) = 1 - e^{-AD(X_{eff})} \tag{4.10}$$

where $D(X_{eff})$ is the area density of defects size X_{eff} or smaller (recall smaller X_{eff}'s are more severe defects) which has been provided by program DEFECT (Section 4.2).

The probability that a circuit fails at or before time t_{BD} is equal to the probability that at least one device in the circuit fails by time t_{BD}. This may be expressed as

$$P\ (\text{circuit failure}) = 1 - \prod_{i=1}^{n} e^{-AD(X_{eff}^{i})} \tag{4.11}$$

where n is the number of MOS devices in the circuit, $D(X_{eff}^{i})$ is the defect density for the i^{th} device and X_{eff}^{i} yields t_{BD} for the particular waveform found at device i.

When a burn-in trial is simulated in addition to normal operation, (4.11) must be modified to account for the circuits lost during the burn-in test. The expression for failure probability after burn-in is

$$P\ (\text{circuit failure}) = \prod_{i=1}^{n} e^{-AD(X_{bi}^{i})} \tag{4.12}$$

where X_{bi}^{i} is calculated for device i from (4.3) with t_{BD} set equal to the burn-in duration, and X_{op}^{i} is calculated for device i from (4.8).

4.1.7 The E model

There is still disagreement in the engineering community as to whether time-dependent dielectric breakdown has a functional dependence on electric field of $\exp\left[\frac{1}{E_{ox}}\right]$ or $\exp\left[-E_{ox}\right]$. These are referred to as the "$\frac{1}{E}$ model" and "E model," respectively. CORS will calculate circuit failure probabilities based on the E model if so requested by the user. However, simulation of burn-in using the E model is not supported in CORS.

[61] proposes that

$$\left[\frac{\partial \ln t}{\partial E}\right]_F = -\gamma \tag{4.13}$$

where F denotes that this expression is to be evaluated at one particular value of fraction of devices failed. [61] also introduces the concept of effective time at a given field which allows us to derive the following equation for t_{BD} under time-varying field conditions.

$$\int_0^{t_{BD}} e^{\gamma E(t)} dt = e^{b(F)} \tag{4.14}$$

where b(F) is the "inverse distribution function." Equation 4.14 is solved for b. Similar to the $\frac{1}{E}$ model, CORS assumes Poisson statistics and finds P(failure) based on the area density of b (provided by program EMODEL - Section 4.2).

The E model takes temperature effects into account by allowing γ to vary with temperature according to

$$\gamma(T) = 10.36 - \frac{.62}{k_B T} \tag{4.15}$$

Recall that

$$t_{BD} \propto e^{-\gamma E_{ox}}$$

4.2 CHARACTERIZING DEFECT DENSITY USING PROGRAM DEFECT

Programs DEFECT and DEFECT2 (Sections 4.2.1 and 4.2.2) format test capacitor breakdown data for compatibility with the (default) $\frac{1}{E}$ model for oxide breakdown. If the user plans to use the "E model," instead, s/he should consult Section 4.2.3.

4.2.1 Using DEFECT to Obtain Area Density of Defects

DEFECT takes raw data from a test capacitor breakdown experiment and formats it for use by CORS. There are two commonly used methods for characterizing oxide breakdown statistics, the time-to-breakdown test and the ramp-voltage-breakdown test. DEFECT will accept data from either of these experiments. In the absence of experimental data, the user may provide DEFECT with parameters for a statistical distribution which is thought to represent the oxide breakdown statistics.

A time-to-breakdown test is performed by applying a constant voltage to an ensemble of identical large-area capacitors. The oxide voltage used is larger than the designed power supply voltage because one wishes to observe failures in a short amount of time (see Eq. (4.2)). The fraction of devices which have failed is recorded as a function of time. For each observed breakdown time, DEFECT derives X_{eff} using (4.2). Next, DEFECT obtains the area density, $D(X_{eff})$, from (10). An ordered list of X_{eff} versus $D(X_{eff})$ is generated by DEFECT and is stored in a user-named output file which will be referenced by a .XEFF card in the CORS input deck (Section IV.B). Following is a description of the format in which time-to-breakdown data should be placed in the DEFECT input file.

Line 1: Oxide Thickness (Å)

Line 2: Area of test structure (cm^2)

Line 3: Test temperature (C°)

Line 4: 0.0
The value 0.0 notifies DEFECT that time-to-breakdown data will be provided

Line 5: Voltage difference between $V_{applied}$ and V_{ox}.
This value will be subtracted from $V_{applied}$ *by DEFECT to yield* V_{ox}. *This value may be estimated using the equations in Section II.C.*

Line 6: $V_{applied}$ (+ volts, the voltage applied to the test capacitor terminals)

Lines 7-: t_{BD} (sec), cumulative percent failed
Lines 7- should consist of ordered pairs ranging from from minimum t_{BD} *on line 7 to the maximum on the final line of the file.*

A DEFECT input file listing containing time-to-breakdown data may be found in Example 1 and a DEFECT output file listing is provided in Example 2.

A ramp-voltage-breakdown test is performed by applying a linearly increasing voltage waveform to an ensemble of identical large-area capacitors. A record of breakdown voltage (V_{BD}) versus the fraction of capacitors failed is generated. V_{BD} may be related to X_{eff} by

$$1 = \frac{V_{BD}^2}{RG\tau X_{eff}} \exp\left[\frac{-GX_{eff}}{V_{BD}}\right] \qquad (4.16)$$

which can be derived from (4.3). DEFECT subsequently derives $D(X_{eff})$ using (10). The input file format needed for DEFECT to process ramp-voltage-breakdown data is as follows.

Line 1: Oxide Thickness (Å)

Line 2: Area of test structure (cm^2)

Line 3: Test temperature (C°)

Line 4: Ramp rate (V/sec)

Line 5: Voltage difference between $V_{applied}$ and V_{ox}.
$V_{applied}$ is the ramp voltage at a given time.
The difference between this value and V_{ox}
may be inferred from the equations in Section II.C.
The value listed on line 5 will be subtracted from V_{BD}.

Line 6: 0.0
This is a dummy number, not used by DEFECT when V_{BD}
is being provided.

Lines 7- V_{BD} (volts), cumulative percent failure
Lines 7- should consist of ordered pairs ranging
from minimum V_{BD} on line 7 to the
largest observed V_{BD} on the final line
of the file.

A DEFECT input file listing containing V_{BD} data may be found in Example 3. The DEFECT output file will be similar to that shown in Example 2.

If test capacitor breakdown data is not available, yet a CORS projection of circuit reliability is desired, the user may provide DEFECT with a statistical distribution which is believed to model the time-to-breakdown statistics for the oxide of interest. Specifically, the user may describe his/her hypothetical breakdown data with a one or two population lognormal or Weibull distribution. The parameters for these distributions are placed in a file which is then input to DEFECT to be properly formated for use by CORS. The input file should be formated as follows.

Line 1: Oxide Thickness (Å)

Line 2: Area of test structure (cm^2)

Line 3: Test temperature (C°)
The data on lines 1-3 is, of course, hypothetical.

Line 4: 0.0

Line 5: the voltage difference between $V_{applied}$ (hypothetical) and V_{ox}.
See format description for time-to-breakdown data (above) for an explanation.

Line 6: $V_{applied}$ (+ volts, the voltage applied to the hypothetical test capacitors)

Line 7: Lognormal
-or-
Weibull

Line 8: Number of populations (1 or 2)

Line 9: (if Lognormal) t50 *median*
-or-
(if Weibull) alpha *location parameter*

Line 10: (if Lognormal) sigma *shape parameter*
-or-
(if Weibull) beta " "
If line 8 reads "2", then the information contained on lines 9 and 10 pertains to one population of samples and the following lines must be included to describe the other population.

Line 11: Fraction of samples following distribution #1
A number between 0 and 1.

Line 12: (if Lognormal) t50 *for population #2*
-or-
(if Weibull) alpha " "

Line 13: (if Lognormal) sigma *for population #2*
-or-
(if Weibull) beta " "

*****All parameters have units of seconds.*****

A sample DEFECT input file following the above format is contained in Example 4 along with the generated output file.

4.2.2 Using DEFECT2 to Obtain Defect Density per Unit Length

The user may also wish to provide CORS with data about the defect density per unit length along diffusion and/or field oxide edges. There are two techniques for measuring these defect densities. The first method is to conduct a breakdown test on an ensemble of test capacitors which have a very large ratio of perimeter to area. If one assumes that the probability that one of these capacitors encompasses a severe area defect is negligible, the user may simply substitute the perimeter of the structures (cm) on line 2 of the DEFECT input file as described in Section 4.2.1. DEFECT will then output $D(X_{eff})$ values which have units of cm^{-1}.

A second method for determining the density of defects along edges is to conduct breakdown tests on two ensembles of capacitors having different perimeter, different area or both. DEFECT2 will create two output files for use by CORS, one containing area defect density, the other containing perimeter defect density. The breakdown data for each set of test capacitors should be stored in a separate file. Each file is formated according to the instructions for program DEFECT with just one modification; an extra line is to be added to the data file between Line 2 (Area of test structure) and Line 3 (Test temperature). This extra line should contain the test structure perimeter in cm.

DEFECT2 calculates defect density as follows.

$$Y = 1 - F \quad \textit{Y is yield, F is cumulative fraction failed}$$

$$Y = \exp\left[AD_A + PD_P \right]$$

4.2.3 Using EMODEL to Obtain Area Density of Defects

The data file which is input to EMODEL should be formated identically to one intended for program DEFECT (details in Section 4.2.1). The difference lies in the calculations. EMODEL calculates the inverse distribution function described in [61]. Similar to X_{eff} in the $\frac{1}{E}$ model, each value of b(F) is associated with a density which when multiplied by device area and placed in the Poisson equation for yield, gives the correct answer, *i.e.*,

$$Y = e^{-AD[b(f)]} \tag{4.17}$$

Y is yield measured at some value of t_{BD} (or V_{BD}), b(F) is corresponds to t_{BD} (V_{BD}) and is calculated by EMODEL, A is the test device area. D[b(f)] is calculated by EMODEL to complete the equality in (4.17).

4.3 EXAMPLES

Examples 1-4 are described in Section 4.2. Example 5 shows a CORS input deck for a simple RC circuit and the subsequent output. This example illustrates the proper use of the .ALTMODEL card. (The oxide defect density distribution used for this simulation is listed in Example 2.) Example 6 shows a CORS input deck and subsequent output for a 4-input nand gate. This example illustrates the proper use of the .EACHPROB and .LSI cards. The input deck shown in Example 7 is identical to that in Example 6 except that the QUICK option is specified on the .TTF card. Note that the failure projections are not very different from those in Example 6. Example 8 shows the proper use of the .BURNIN card. Example 9 is identical to Example 8 except that the QUICK option was requested. The user should compare the output listings in Example 8 and Example 9 to determine if the error introduced by using QUICK during a burn-in simulation is acceptable for his/her application.

```
.1
27
0.0
1.2
9
.2,13.3
.4,16.7
.6,20
1.6,23.3
1.8,25
2,26.7
2.4,28.3
3.4,30
4,31.7
4.2,33.3
6,35
6.4,36.7
7.8,38.3
9.4,40
12,41.7
12.2,43.3
13.4,45
13.8,46.7
14,50
16.6,51.7
17.2,53.3
20.8,55
24.4,56.7
32.4,58.3
36.6,60
39.4,61.7
50,63.3
51.8,65
53.4,66.7
68.2,68.3
68.6,70
84.4,71.7
87,73.3
138,75
230,76.7
275,78.3
650,81.7
673,83.3
1100,85
1165,86.7
1209,88.3
1250,90
```

Example 1

Listing of input file for DEFECT. Contains time-to-breakdown data.

```
55.307817 2.231436
57.493666 2.652685
57.756153 2.876821
57.990957 3.106096
58.397273 3.326794
59.173500 3.566749
59.535685 3.812604
59.644417 4.049652
60.439293 4.307829
60.583122 4.572849
61.023990 4.828863
61.439810 5.108256
61.984021 5.395681
62.020858 5.673960
62.229939 5.978370
62.295490 6.292339
62.327557 6.931472
62.707183 7.277386
62.786313 7.614260
63.209838 7.985077
63.565586 8.370176
64.197553 8.746691
64.469194 9.162907
64.633478 9.597203
65.164452 10.023934
65.243270 10.498221
65.311064 10.996128
65.856248 11.488535
65.869281 12.039728
66.331208 12.623084
66.398824 13.205066
67.426966 13.862944
68.565377 14.567168
68.963604 15.278579
70.880624 16.982691
70.958118 17.897615
72.053060 18.971200
72.181005 20.174062
72.263623 21.455813
72.337946 23.025851
```

Example 2

Listing of DEFECT output file. Input file is displayed in Ex. 1. This oxide has an unacceptably high defect density for commerical applications!

```
.1
27
1
1.2
0
7.8,1.8
8.4,3.6
8.6,8.9
8.8,10.7
9,19.6
9.4,26.8
9.6,37.5
9.8,39.3
10,41.1
10.2,46.4
10.4,50
10.6,57.1
10.8,66.1
11,69.6
11.2,76.8
11.4,78.6
13,80.4
13.6,82.1
13.8,83.9
14,87.5
15.2,89.3
15.6,91.1
```

Example 3.

Listing of DEFECT input file containing ramp-voltage-breakdown data.

```
125
.01
27
0
0
9
Lognormal
1
20
2

*** Output file ***


Temp= 27
Vox= 9.000000
Area= 0.010000
Lognormal
numpop= 1
t50_1/sigma1: 20 2
```

Example 4

Listings of DEFECT input and output files containing parameters for a lognormal time-to-breakdown distribution.

```
SAMPLE CIRCUIT
C1 2 0 1.4P
+ TBDMODEL=CMOD L=10U W=50U
R1 1 2 1K
VIN 1 0  PULSE(0 5 5n 5n 5n 10n 30n)
.ALTMODEL CMOD C TOX=12.5N
.TRAN .5N 60N
.XEFF CMOD FILENAME=data125
.TTF
.PRINT tran v(1) v(2)
.END
```

CORS Output

```
                                    SAMPLE CIRCUIT

TDDB STATISTICS
Operating Temperature=              27

number seconds 2.592e+06        fraction failed 2.131181e-05
number seconds 7.776e+06        fraction failed 2.779222e-05
number seconds 1.66752e+07      fraction failed 3.983974e-05
number seconds 3.1536e+07       fraction failed 4.368089e-05
number seconds 6.3072e+07       fraction failed 5.299287e-05
number seconds 1.5768e+08       fraction failed 6.692421e-05
number seconds 3.1536e+08       fraction failed 7.033430e-05
number seconds 6.3072e+08       fraction failed 7.523625e-05
```

Example 5

```
FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 5.5
VA 3 0 PWL(0 5.5 79N 5.5 80N 0 85N 0 87N 5.5 117N 5.5 119N 0 160N 0)
VB 4 0 5.5
VD 6 0 5.5
VC 5 0 PWL(0 0 5N 0 7N 5.5 37N 5.5 39N 0 80N 0 81N 5.5 160N 5.5)
.TRAN 1N 160N
.TTF
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 16K 64K
.EACHPROB all
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END
```

Example 6

TDDB STATISTICS
Operating Temperature= 27

number seconds 2.592e+06 (1 mo) fraction failed 7.834910e-06
number seconds 7.776e+06 (3 mo) fraction failed 1.079340e-05
number seconds 1.66752e+07 (6 mo) fraction failed 1.321533e-05
number seconds 3.1536e+07 (1 yr) fraction failed 1.605680e-05
number seconds 6.3072e+07 (2 yr) fraction failed 1.819782e-05
number seconds 1.5768e+08 (5 yr) fraction failed 1.970588e-05
number seconds 3.1536e+08 (10 yr) fraction failed 2.125051e-05
number seconds 6.3072e+08 (20 yr) fraction failed 2.322021e-05

TDDB STATISTICS FOR 16000 IDENTICAL CELLS

number seconds 2.592e+06 (1 mo) fraction failed 1.178199e-01
number seconds 7.776e+06 (3 mo) fraction failed 1.586061e-01
number seconds 1.66752e+07 (6 mo) fraction failed 1.905876e-01
number seconds 3.1536e+07 (1 yr) fraction failed 2.265628e-01
number seconds 6.3072e+07 (2 yr) fraction failed 2.526098e-01
number seconds 1.5768e+08 (5 yr) fraction failed 2.704279e-01
number seconds 3.1536e+08 (10 yr) fraction failed 2.882381e-01
number seconds 6.3072e+08 (20 yr) fraction failed 3.103201e-01

TDDB STATISTICS FOR 64000 IDENTICAL CELLS

number seconds 2.592e+06 (1 mo) fraction failed 3.943398e-01
number seconds 7.776e+06 (3 mo) fraction failed 4.988157e-01
number seconds 1.66752e+07 (6 mo) fraction failed 5.707804e-01
number seconds 3.1536e+07 (1 yr) fraction failed 6.421507e-01
number seconds 6.3072e+07 (2 yr) fraction failed 6.879748e-01
number seconds 1.5768e+08 (5 yr) fraction failed 7.166829e-01
number seconds 3.1536e+08 (10 yr) fraction failed 7.433513e-01
number seconds 6.3072e+08 (20 yr) fraction failed 7.737491e-01

Probability of Failure at 3.1536e+08 seconds

MNB 5.234728e-06
MNA 5.165112e-06
MND 4.922535e-06
MNC 4.902181e-06
MPC 5.446983e-07
MPA 4.814296e-07
MPB 0.000000e+00
MPD 0.000000e+00

Example 6

```
FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 5.5
VA 3 0 PWL(0 5.5 79N 5.5 80N 0 85N 0 87N 5.5 117N 5.5 119N 0 160N 0)
VB 4 0 5.5
VD 6 0 5.5
VC 5 0 PWL(0 0 5N 0 7N 5.5 37N 5.5 39N 0 80N 0 81N 5.5 160N 5.5)
.TRAN 1N 160N
.TTF QUICK
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 16K 64K
.EACHPROB all
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END
```

Example 7

FOUR INPUT CMOS NAND GATE

TDDB STATISTICS
Operating Temperature= 27

number seconds 2.592e+06 (1 mo) fraction failed 7.834910e-06
number seconds 7.776e+06 (3 mo) fraction failed 1.090249e-05
number seconds 1.66752e+07 (6 mo) fraction failed 1.344226e-05
number seconds 3.1536e+07 (1 yr) fraction failed 1.635161e-05
number seconds 6.3072e+07 (2 yr) fraction failed 1.836965e-05
number seconds 1.5768e+08 (5 yr) fraction failed 2.006943e-05
number seconds 3.1536e+08 (10 yr) fraction failed 2.169091e-05
number seconds 6.3072e+08 (20 yr) fraction failed 2.377342e-05

TDDB STATISTICS FOR 16000 IDENTICAL CELLS

number seconds 2.592e+06 (1 mo) fraction failed 1.178199e-01
number seconds 7.776e+06 (3 mo) fraction failed 1.600734e-01
number seconds 1.66752e+07 (6 mo) fraction failed 1.935211e-01
number seconds 3.1536e+07 (1 yr) fraction failed 2.302026e-01
number seconds 6.3072e+07 (2 yr) fraction failed 2.546618e-01
number seconds 1.5768e+08 (5 yr) fraction failed 2.746595e-01
number seconds 3.1536e+08 (10 yr) fraction failed 2.932359e-01
number seconds 6.3072e+08 (20 yr) fraction failed 3.163979e-01

TDDB STATISTICS FOR 64000 IDENTICAL CELLS

number seconds 2.592e+06 (1 mo) fraction failed 3.943398e-01
number seconds 7.776e+06 (3 mo) fraction failed 5.023027e-01
number seconds 1.66752e+07 (6 mo) fraction failed 5.769692e-01
number seconds 3.1536e+07 (1 yr) fraction failed 6.488394e-01
number seconds 6.3072e+07 (2 yr) fraction failed 6.913874e-01
number seconds 1.5768e+08 (5 yr) fraction failed 7.231990e-01
number seconds 3.1536e+08 (10 yr) fraction failed 7.504842e-01
number seconds 6.3072e+08 (20 yr) fraction failed 7.816197e-01

Probability of Failure at 3.1536e+08 seconds

MNB 5.523978e-06
MNA 5.198405e-06
MND 4.991622e-06
MNC 4.922559e-06
MPC 5.619827e-07
MPA 4.925431e-07
MPB 0.000000e+00
MPD 0.000000e+00

Example 7

```
**** Input deck used during pass #1. ****

FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 7
VA 3 0 PWL(0 7 79N 7 80N 0 85N 0 87N 7 117N 7 119N 0 160N 0)
VB 4 0 7
VD 6 0 7
VC 5 0 PWL(0 0 5N 0 7N 7 37N 7 39N 0 80N 0 81N 7 160N 7)
.TRAN 1N 160N
.TTF
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 64K
.BURNIN TIME=3600 TEMP=100
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END

**** Input deck used during pass #2. ****

FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 5.5
VA 3 0 PWL(0 5.5 79N 5.5 80N 0 85N 0 87N 5.5 117N 5.5 119N 0 160N 0)
VB 4 0 5.5
VD 6 0 5.5
VC 5 0 PWL(0 0 5N 0 7N 5.5 37N 5.5 39N 0 80N 0 81N 5.5 160N 5.5)
.TRAN 1N 160N
.TTF
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 64K
.BURNIN TIME=3600 TEMP=100
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END
```

Example 8

FOUR INPUT CMOS NAND GATE

TDDB STATISTICS
Operating Temperature= 27

Prior to Operation, Burn In was Conducted for
seconds= 3600.000000 temperature= 100
*** The Yield after Burn-In was 0.999976 ***

number seconds 2.592e+06 (1 mo) fraction failed 7.547313e-09
number seconds 7.776e+06 (3 mo) fraction failed 4.502429e-08
number seconds 1.66752e+07 (6 mo) fraction failed 9.561688e-08
number seconds 3.1536e+07 (1 yr) fraction failed 1.762994e-07
number seconds 6.3072e+07 (2 yr) fraction failed 3.344744e-07
number seconds 1.5768e+08 (5 yr) fraction failed 6.947581e-07
number seconds 3.1536e+08 (10 yr) fraction failed 1.149830e-06
number seconds 6.3072e+08 (20 yr) fraction failed 5.622876e-06

TDDB STATISTICS FOR 64000 IDENTICAL CELLS
*** The Yield after Burn-In for 64000 identical cells was 0.216911 ***

number seconds 2.592e+06 (1 mo) fraction failed 4.829114e-04
number seconds 7.776e+06 (3 mo) fraction failed 2.877407e-03
number seconds 1.66752e+07 (6 mo) fraction failed 6.100795e-03
number seconds 3.1536e+07 (1 yr) fraction failed 1.121975e-02
number seconds 6.3072e+07 (2 yr) fraction failed 2.117888e-02
number seconds 1.5768e+08 (5 yr) fraction failed 4.349047e-02
number seconds 3.1536e+08 (10 yr) fraction failed 7.094667e-02
number seconds 6.3072e+08 (20 yr) fraction failed 3.022295e-01

Example 8

```
**** Input deck used during pass *1. ****

FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 7
VA 3 0 PWL(0 7 79N 7 80N 0 85N 0 87N 7 117N 7 119N 0 160N 0)
VB 4 0 7
VD 6 0 7
VC 5 0 PWL(0 0 5N 0 7N 7 37N 7 39N 0 80N 0 81N 7 160N 7)
.TRAN 1N 160N
.TTF QUICK
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 64K
.BURNIN TIME=3600 TEMP=100
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END

**** Input deck used during pass #2. ****

FOUR INPUT CMOS NAND GATE
MPA 2 3 1 1 MODP W=20U L=1U
MPB 2 4 1 1 MODP W=20U L=1U
MPC 2 5 1 1 MODP W=20U L=1U
MPD 2 6 1 1 MODP W=20U L=1U
MNA 9 3 0 0 MODN W=32U L=1U
MNB 8 4 9 0 MODN W=32U L=1U
MNC 7 5 8 0 MODN W=32U L=1U
MND 2 6 7 0 MODN W=32U L=1U
CLOAD 2 0 30F
.MODEL MODP PMOS VTO=-.7 TOX=12.5N KP=8U GAMMA=.4 TPG=-1
.MODEL MODN NMOS VTO=.7 TOX=12.5N KP=20U GAMMA=.4
VDD 1 0 5.5
VA 3 0 PWL(0 5.5 79N 5.5 80N 0 85N 0 87N 5.5 117N 5.5 119N 0 160N 0)
VB 4 0 5.5
VD 6 0 5.5
VC 5 0 PWL(0 0 5N 0 7N 5.5 37N 5.5 39N 0 80N 0 81N 5.5 160N 5.5)
.TRAN 1N 160N
.TTF QUICK
.XEFF MODP FILENAME=data125
.XEFF MODN FILENAME=data125
.LSI 64K
.BURNIN TIME=3600 TEMP=100
.options reltol=.01 abstol=1e-9 vntol=.0001 itl1=2000 itl2=500
.END
```

Example 9

```
TDDB STATISTICS
Operating Temperature=          27

Prior to Operation, Burn In was Conducted for
seconds= 3600.000000     temperature= 100
*** The Yield after Burn-In was 0.999976 ***

number seconds 2.592e+06 (1 mo)          fraction failed 9.650230e-09
number seconds 7.776e+06 (3 mo)          fraction failed 5.340358e-08
number seconds 1.66752e+07 (6 mo)        fraction failed 1.132711e-07
number seconds 3.1536e+07 (1 yr)         fraction failed 2.080374e-07
number seconds 6.3072e+07 (2 yr)         fraction failed 3.917119e-07
number seconds 1.5768e+08 (5 yr)         fraction failed 7.987068e-07
number seconds 3.1536e+08 (10 yr)        fraction failed 2.683061e-06
number seconds 6.3072e+08 (20 yr)        fraction failed 1.006256e-05

 TDDB STATISTICS FOR 64000 IDENTICAL CELLS
*** The Yield after Burn-In for 64000 identical cells was 0.216911 ***

number seconds 2.592e+06 (1 mo)          fraction failed 6.174241e-04
number seconds 7.776e+06 (3 mo)          fraction failed 3.411995e-03
number seconds 1.66752e+07 (6 mo)        fraction failed 7.223140e-03
number seconds 3.1536e+07 (1 yr)         fraction failed 1.322615e-02
number seconds 6.3072e+07 (2 yr)         fraction failed 2.475794e-02
number seconds 1.5768e+08 (5 yr)         fraction failed 4.983275e-02
number seconds 3.1536e+08 (10 yr)        fraction failed 1.577818e-01
number seconds 6.3072e+08 (20 yr)        fraction failed 4.748161e-01
```

Example 9

5. Circuit Electromigration Simulator - (EM)

We have developed models for predicting interconnect and intermetallic contact reliability due to arbitrary current waveform stress. These models are incorporated in the Circuit Electromigration Simulator module. The result is a tool which can (1) advise the user, based on user-specified reliability requirements, of the required width and length of each interconnect, as well as give a safety factor for the contacts and vias; (2) estimate the overall circuit electromigration failure rate and the cumulative failure percentage of a layout design.

5.1 ELECTROMIGRATION RELIABILITY MODEL

5.1.1 Electromigration's Dependence on Current Density

The time-to-failure (TTF) due to electromigration caused by arbitrary current waveforms (valid for frequencies of waveforms greater than 1kHz) is given by [63]:

$$TTF = \frac{A_{DC}(T)}{\overline{|J|^{m-1}}\,\bar{J}\left[1 + \frac{A_{DC}}{A_{AC}}\frac{(\overline{|J|} - \bar{J})}{\bar{J}}\right]} \tag{5.1}$$

where $\bar{J}$ is the average current density, $\overline{|J|}$ is the average of the absolute current density. m, A_{DC} and A_{AC} are experimentally determined constants which need to be supplied by the user. $A_{AC}(T)$ and $A_{DC}(T)$ have an Arrhenius dependence (with the activation energy E_a) on temperature.

5.1.2 Electromigration's Dependence on Geometry

This simulator uses the independent element analysis described in [67] to find the dependence of electromigration on the length of the interconnect. The simulator asks for a single time-to-failure result for a long interconnect (longer than the longest interconnect in the circuit). Failure statistics are then calculated for shorter lines using the following assumption: a long interconnect is modeled by a series of shorter segments (see Appendix D) and its Time to Failure (TTF) is determined by the weakest segment. Therefore, the failure rate of a long metal line is the sum of the failure rates of the shorter segments. For example, if the user inputs a Median Time to Failure (MTF) of 7.5 hours for a 4.5 cm long interconnect, the simulator can determine the failure distribution for a line half the length of the original test line which has a failure probability of F(t). The 4.5 cm long line will fail if either or both of the two segments fail, the failure probability of a 2.25 cm line $G_2(t)$ is:

$$[1 - G_2(t)]^2 = [1 - F(t)]$$

$$G_2(t) = 1 - [1 - F(t)]^{\frac{1}{2}}$$

The first equation states that the probability that the long line will not fail is the product of the probability that the two shorter segments are good. Thus, at time t = 7.5, the failure probability for the 2.25 cm long line is 0.29. In general, for a line that is 1/x of the long test line, $G_x(t)$ is:

$$G_x(t) = 1 - [1 - F(t)]^{\frac{1}{x}} \tag{5.2}$$

The width dependence of TTF is obtained from an empirical fit of TTF versus linewidth data using a piecewise fit of two second order polynomial functions (see Appendix E). This allows the simulator to model the increase in TTF when the linewidth is decreased below the average grain

size [68]. The width dependence is specified by four parameters, A_w, B_w, C_w and D_w which are defined by the following equations:
for $W \geq B_w$:

$$TTF(W) = A_w \times (W-B_w)^2 + D_w \quad (5.3a)$$

for $W < B_w$:

$$TTF(W) = C_w \times (W-B_w)^2 + D_w \quad (5.3b)$$

B_w is the linewidth at the minimum of the TTF versus linewidth data. It is approximately equal to the average grain size in the interconnect.

5.1.3 Contact Electromigration Model

The lifetime of contact structures is modeled in a way that is similar to the way an interconnect is modeled (i.e. Eq.(5.1)). Our model assumes the use of barrier metal technology and does not consider failure due to Si migration at the Al-diffusion contact. The parameters in Equation 5.1 are extracted from lifetime experiments using contact chain structures. When designing the chain structures, the area of each contact opening in the test structure should be the same as the area of the contact opening that the user designs in the circuit layout. In the model, the current crowding effect and step coverage issues are already factored into the parameters A_{DC} and A_{AC} since the same contact size is used in the test structures and circuit layout. For the same reason, the simulator does not require a model for the dependence of lifetime on contact size. In the layout advisory table for contacts, the simulator calculates the number of contact openings (each with the same area) needed to meet the reliability specifications. Current density is assumed to be divided evenly among contact openings at a particular connection. Via contacts are similarly treated.

5.1.4 Electromigration's Dependence on the Statistical Distribution

The user is given the choice of using either the lognormal or Weibull distribution function to calculate the failure rate and the cumulative percent failure ([69] has an excellent discussion on the statistical failure distributions). Although the lognormal distribution function is commonly used to represent the experimental time-to-failure data, there is no physical reason to expect that the electromigration failures should be lognormally distributed. If the lognormal distribution is chosen, the length dependence model adopted in the simulator will result in distributions for shorter lines that are not lognormal. This inconsistency is illustrated in Appendix D. If the Weibull distribution is chosen, the inconsistency is removed, i.e. failure distributions for all lines are Weibull.

5.2 HOW TO USE THE EM SIMULATOR

5.2.1 General Operation

There are two modes of operation for the simulator, either one or both can be selected by the appropriate EM commands. In the first mode (Fig. 5.1(a)), the simulator is used as a layout advisor. The user inputs the desired reliability specification, which is a failure rate for the circuit after a specified number of operating hours. Based on this requirement, the simulator will generate layout guidelines for the width and length of each interconnect (up to three layers of interconnect are supported), and a safety factor for each contact or via in a circuit. 1/(safety factor) is the number of contacts/vias that will satisfy the reliability specification.

In the second mode of operation (Fig. 5.1(b)), the simulator calculates the failure rate and/or the cumulative percent failure of a circuit layout. The user can supply the layout geometry of all the interconnects, contacts and vias for the layout by hand, or more conveniently, use the layout

extractor which is provided with our simulator. This layout extractor reads in the Caltech Intermediate Format (CIF) layout description file, extracts the circuit elements, and generates the SPICE input deck for the circuit. At the same time, it also produces the geometry description file containing the length and width of each segment and the area of both the metal-to-metal vias and metal-to-silicon contacts.

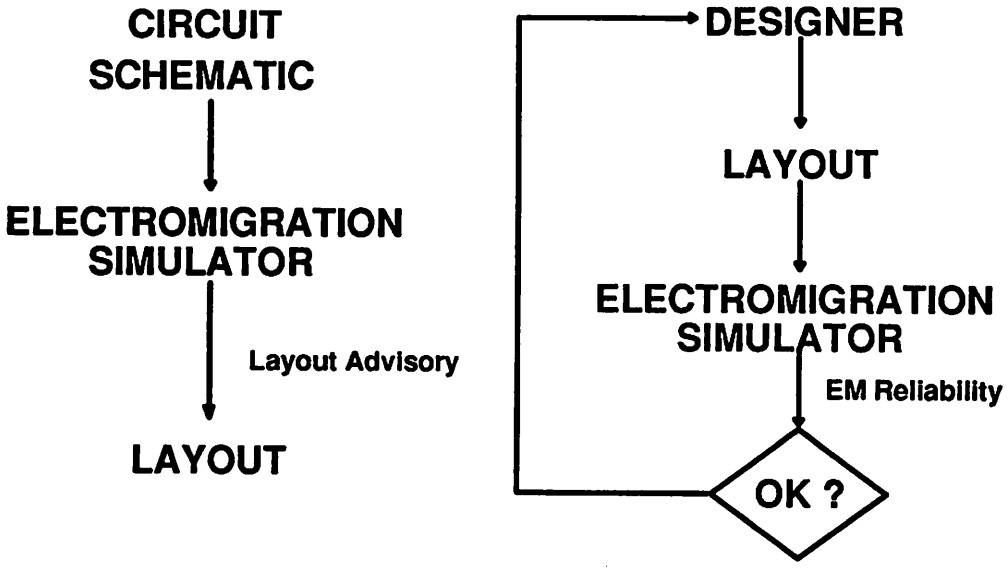


Fig. 5.1. The electromigration simulator can be used in two ways: (a) to generate layout advisory, (b) to compute circuit failure rate and cumulative percent failure.

5.2.2 Parameters needed by the Simulator

One set of physical and statistical parameters are needed for each layer of interconnects, contacts and vias. The parameters are entered by the user in the EM design rule file which will be read by the post-processor.

In the rule file, A_{DC}, A_{AC}, m and E_a are required parameters (in Eq.(5.1)). These parameters can be extracted from the TTF versus current density plot and the Arrhenius plot of TTF versus temperature. In addition, the length, width and thickness of the test structures are required. The user selects either lognormal or Weibull distribution to represent the TTF data. If TTF is required to be a function of width (the simulator defaults to constant width dependence), the width parameters (Eqs.(5.3a) and (5.3b)) have to be extracted from additional experiments using the same type of long line but with a number of different widths.

The parameter sets for contact and via are set up similarly. The test data for contact or via electromigration is obtained from contact/via chain test structures. The EM rule file will contain A_{DC}, A_{AC}, m, E_a as well as the area and the total number of contacts/vias in the chain.

Example of EM design rule file:

```
* Reliability parameters for electromigration simulator
PrintCurrent
* print the current in each connection in the circuit
ncurrent=3 0.50e-3 1.00e-3 2.00e-3
* requests simulator to generate current layout guidelines for 3 current values
SkipFailpercent
* Do not print cumulative percent failure table, only failure rate table will
* be printed
WorstList=0.5
* print half (50%) of the worst connections in failure rate/cumulative percent
* failure table.
AC_define=0.2
* Definition of Pure AC current
MinJcurrent=1e2
* ignore anything with current density less than 1e2 A/cm2
spec_time=1.0e+4 spec_failrate=1.0e-9
* this is the reliability specifications
nwidth=2 1.0 2.0
* Generate layout guidelines for two widths of interconnect
ncv=3 1.0 2.0 4.0
* Generate layout guidelines for three contacts/vias openings: 1, 2 and 4
* at a connection
* The following set for metal-one
metal1 length=4.5e+4 width=1.0 thick=0.1
Tdata=25.0 Ea=0.5
width_a=0.0 width_b=1.0
m=2.0 Adc=1.0e+16 Aac=1.0e+20
Lognormal logsigma=1.0 logmedian=7.52
* The following set for metal-two
metal2 length=4.5e+4 width=1.0 thick=0.1
Tdata=25.0 Ea=0.5
width_a=0.0 width_b=1.0
m=2.0 Adc=1.0e+15 Aac=1.0e+19
Lognormal logsigma=1.0 logmedian=7.52
* The following set for via
via area=1.0 nchain=10.0
* The test structure is a chain of 10 vias, each is 1μm^2
Tdata=25.0 Ea=0.5
area_a=0.0 area_b=1.0
m=2.0 Adc=1.0e+16 Aac=1.0e+20
Lognormal logsigma=1.0 logmedian=7.52
* Because set for contact and metal-three are not given
* Simulator will ignore them
```

The pre-processor will generate the geometry file, *deckfile*.geo (or optionally named *geometryfile* with the -G option). If the SPICE input deck was not generated by the layout extractor i.e. the deck was generated by hand, the pre-processor will add dummy voltages in the SPICE input deck. The dummy voltages have the prefix VEM and the added node numbers will start from 5000. Therefore, the original SPICE input deck must not have any voltage sources with the prefix VEM

or node number greater than or equal to 5000. The user can change the starting number (default 5000) of the dummy nodes by redefining STARTNODE in the pre-processor header file empredef.h.

5.3 OUTPUT FROM THE SIMULATOR

5.3.1 Current Table

Normally the current table will not be printed, unless the user specifically requests it by including **PrintCurrent** in the rule file. In this case, the table will be at the beginning of the output. This table lists average currents and average absolute currents in amps for each connection in the circuit. For some connections, where the average current < AC_DEFINE × average absolute current, the simulator performs the calculation assuming the current waveform is a pure AC waveform by setting the internal average current to zero. Those connections are indicated by **AC** in the current table. It is a good idea to check this table to make sure that the AC waveform assumptions are valid. This is particularly important for signal lines in MOS circuit where the gate current is assumed to be a pure AC waveform. If a particular connection to a purely capacitive node is not indicated by **AC** in the output, the layout guideline and the failure rate calculated for this connection will be too pessimistic (assuming **Aac** > **Adc** in the rule file).

5.3.2 Layout Advisory Tables

There are two layout advisory tables. The first table contains the following.

(1) the maximum interconnect length L (in μm) for metal-one, metal-two or metal-three. This is based on a given DC current (in A).

(2) the safety factor S for contacts or vias for a given DC current (in A). If a number less than 1.0 is printed, the number of contact/via openings carrying the given total DC current is not sufficient to satisfy the reliability specification.

This table, although generated for a given DC current, can be used as a guideline for circuits operating on pulse waveforms if **m** in Eq.(5.1) is equal to 2. This is possible because the only current density dependence in Eq.(5.1) is J. Therefore, the designer can just read off the average current from the top of the table. For the case of pure AC current, if **m** is equal to 2, Eq.(5.1) will simplify to:

$$TTF = \frac{A_{AC}}{|J|^2}$$

or

$$= \frac{A_{DC}}{\left[\sqrt{\frac{A_{DC}}{A_{AC}}}\,|J|\right]^2}$$

To use the table for pure AC current waveforms, the current at the top of the table will have to be multiplied by $\sqrt{\frac{A_{AC}}{A_{DC}}}$. For example, if **Aac**=1.0e20 and **Adc**=1.0e16, the result in the table for a DC current of 0.1mA will be the same as an AC current (the average of the absolute current) of 10.0mA.

The second layout advisory table is a list of maximum interconnect lengths L (in μm) and the safety factor S for contacts or vias. This guideline is given for a number of specified widths of interconnects and a number of openings of contacts and vias per connection which the user specifies by using **NWIDTH** and **NCV** in the rule file.

The safety factor S for contacts and vias is the maximum number of contacts/vias chains that is allowed along the path of the current flow. For example, let us say the user specified 2 via openings per terminal. If 3 is the result, the number of metal-to-metal via chains (2 vias at each terminal) along the current path can not exceed 3. A number less than 1.0 for a particular number of contact/via openings indicates that the number of openings per connection is not sufficient to satisfy the reliability specification.

If the user enters the "stacked" parameter for a connection (See Sec.5.6.3), the results generated only apply to one connection. If the current density in a connection is smaller than *MinJCurrent* (specified using **MINJCURRENT=** in the rule file), the table entry will be $J < MinJ$.

5.3.3 Failure Rate Statistics

This part of the output is requested by **EMSTAT** (if the user set up the geometry file by hand) or **EMSTATX** (if the geometry file comes from the layout extractor). In this table, the failure rate and/or cumulative percent failure for each connection is calculated at times specified using the **EMSTAT** or **EMSTATX** card. The user can select to print out some of the worst connections by using the **WorstList** card in the rule file, otherwise all connections will be printed. The first column of the table is the name given by the simulator for the connection (useful for identifying trouble spots in CIF file) followed by the node connection in the SPICE input deck. At the bottom of the table the total failure rate/cumulative percent failure for the design is given. If **EMSTATX** is specified, the simulator will print out the locations of connections that pose electromigration hazards in CIF format. In this output, the names of the connections are listed in descending order by failure rates. The ranking is attached to the name separated by a dash. The user can then cut out the CIF format output and superimpose the names on the original CIF layout file for printout.

5.4 OPERATION OF THE LAYOUT EXTRACTOR

This section describes the procedures to extract the SPICE input deck from a CIF layout file. First, **mextra** is used to extract the connectivity of transistor, interconnect, contact and via. The extracted information is written to a file in **sim** format. **sim2spice** is called next to construct the SPICE input deck and also produce the layout geometry database from the **sim** file. A CIF to PostScript plotting program **cif2ps** is provided to plot the CIF file. **cif2ps** can also be used to locate input and power supply nodes in the SPICE deck and also to view any reliability hazards in interconnects, vias and contacts.

The use of the layout extractor might become cumbersome because this process generates a lot of data and therefore will consume a considerable amount computation time in the SPICE simulation. If this is the case, setting up the geometry file by hand may be more appropriate (see section 5.5)

5.4.1 Manhattan Circuit Extractor for VLSI Simulation

The operation of the layout extractor: **mextra** is described in the accompanying manual page (see Appendix H). The technologies known to **mextra** are: nMOS ("nmos"), MOSIS P well CMOS/Bulk, also known as CBPM ("cmos-pw"), MOSIS Scalable CMOS/Bulk N-well, also known as SCN ("cmos-nw"), MOSIS Scalable CMOS/Bulk P-well, also known as SCP ("cmos-s"), and MOSIS Scalable CMOS/Bulk Generic, also known as SCG ("cmos-g"). The mask layer names for each technology are listed in Appendix F. If the CIF layers have different names than the ones listed in Appendix F, the user can set up his/her own CIF layer names and use the **-L** option of **mextra** (explained in Appendix G). An example of the command line is:

>mextra -t scmos circ.cif

which extracts the layout from *circ.cif* file. The technology is *scmos.*

5.4.2 Modifications in Mextra

Detailed description of the original version of **mextra** can be found in [70]. Modifications have been made to the original version to extract interconnect widths and lengths, metal-to-metal via and metal-to-silicon contact areas. The additional layout information is appended to the *basename*.**sim** file. The line describing the via or contact geometry has the following format:

type node1 node2 area xloc yloc;

where *type* is either **CONT** or **VIA** (for metal-one to silicon contact or metal-one to metal-two via), *node1* and *node2* are the connection nodes, *area* is in square centi-microns (=0.0001 μm^2). The location of the via or contact is given by *xloc* and *yloc* in cif coordinates.
The line describing the metal-one or metal-two interconnect has the following format:

type node1 node2 width length xloc yloc;

where *type* is either **M1** or **M2**, *node1* and *node2* are the connection nodes, *width* and *length* are in centi-micron (=0.01 μ). The location of the interconnect is given by *xloc* and *yloc* in cif coordinates.

5.4.3 Limitations of the Layout Extractor

The layout extractor has been tested for a number of circuit designs. Although it works well in most cases, the user has to be aware of its limitations:

(1) **mextra** can only handle Manhattan type structures. Non-manhattan polygons will be ignored by the extractor.

(2) **mextra** can only extract two layers of interconnects.

(3) The extracted interconnect length and width are accurate for long metal lines. **mextra** tends to err at irregular corners and where many metal lines join together.

(4) **mextra**, by default only recognizes the technologies listed in Appendix F. The user is advised to check the layer names in his/her CIF file with the layers listed in the table to prevent unknow layers error. If necessary, user can set up alternate CIF layer names (see Appendix G).

5.4.4 Converting .sim Format to SPICE Input Format

The program **sim2spice** converts the *basename*.**sim** file produced by **mextra** to a SPICE input deck and also produces the layout geometry database in file *basename*.**geo**. Two files are needed for a successful conversion: *basename*.**sim** which contains connectivity information and *basename*.**nodes** which contains node names and numbers. Both files are generated by **mextra**. **sim2spice** puts the spice input deck in *basename*.**spice** and the geometry information in *basename*.**geo**. In the SPICE deck, **sim2spice** will add for each interconnect and contact structure a dummy voltage source. For a metal-to-silicon contact and metal-to-metal via, a small series resistance (0.001 Ω) will also be added in the SPICE deck. The name of the dummy voltage source has the prefix **VEM** and the series resistor has the prefix **REM**. The resistor is needed to prevent a SPICE error when there is a closed loop consisting of voltage sources only.

The SPICE deck from **sim2spice** is incomplete. It does not have the input and power sources and it does not request any spice analysis. The user must insert the voltages and set up the transient analysis card. The user will also have to insert the following two cards to request EM analysis.

.EMMODEL *rulefile*
.EMSTATX *basename*.geo *1.0e4 1.0e5*

where *rulefile* is the EM design rule file and *basename*.geo is the extracted geometry file from **sim2spice.**

The user can locate the input, power and output nodes of the circuit by using the **cif2ps** program. This is described in Section 5.4.5.

The extracted SPICE deck may have a number of floating nodes where there is only one connection to the node. This will cause SPICE2G6 to abort simulation. Therefore, the user is advised to use SPICE3C or SPICE3E for the extracted SPICE input deck.

The geometry information of the layout in *basename*.geo has the following format:

node1 node2 VEMxxx type {width length} | area xloc yloc;

where *node1* and *node2* are the nodes in SPICE deck: *VEMxxx* is the dummy voltage source in the SPICE deck; type is one of **M1** (for metal-one); **M2** (for metal-two); **CO** (for contact) or **VI** (for via); *width* and *length* are in μm (for interconnects) and *area* (for contacts and vias) is in μm^2. The location of the connection is given in CIF coordinates *xloc* and *yloc*.

sim2spice also produces a file *basename*.spcnode containing node numbers in the SPICE deck and their locations in the layout. The list is written in CIF format. It can be read by **cif2ps** which will make a plot showing the circuit with the named nodes superimposed.

An example of this command line is:

```
>sim2spice circ.sim
```

After this, the program will read **circ.sim, circ.nodes** and produce **circ.spice, circ.spcnode** and **circ.geo.**

5.4.5 Plotting a CIF File

The program **cif2ps** converts a CIF layout file into a PostScript file which can be sent to a PostScript printer for printout. The operation of **cif2ps** is described in the accompanying manual page (see Appendix H). The following command line:

```
>cif2ps -t scmos circ.cif | lpr
```

will plot **circ.cif** (which contains CIF layer names of scmos technology). **cif2ps** is technology dependent; the layers known to cif2ps are listed in Appendix F. Appendix G describes how to set up different CIF layer names for **mextra** and **cif2ps.**

The **-m** option of **cif2ps** can be used to superimpose CIF labels on a CIF layout. This is useful in locating the SPICE node numbers assigned by **sim2spice.** Recall that **sim2spice** writes the locations of nodes in CIF format into *basename*.spcnode file. This file can be printed together with the original CIF layout file *basename*.cif using **cif2ps -m.** For example:

```
>cif2ps -m -t scmos circ.cif circ.spcnode | lpr
```

will print the SPICE node numbers on the layout *circ.cif*.

The EM simulator produces a listing of metal structures that pose reliability hazards in CIF format. This listing can be printed on the CIF layout using the same procedure:

```
>cif2ps -m -t scmos circ.cif circ.worst | lpr
```

circ.worst contains the listing of the worst metal structures.

5.5 SETTING UP A GEOMETRY FILE BY HAND

The user may create the geometry file by hand instead of using the layout extractor. For a layout which uses a large number of standard cells, such as the case with gate arrays and memory chips, the user is advised to set up the geometry file by hand. In this case, many standard cells are repeated, but only one cell's current waveforms are needed. The user can set up subcircuit elements (using the **SUBCKT** card in SPICE) for all but one of the standard cells in the circuit. This implicitly instructs the EM simulator not to calculate the failure probabilities for the metal connections in the subcircuit definition. As a result, the amount of cpu time will be reduced. The user can setup the geometry file by using the following procedures:

(1) The user enters the layout geometry of one cell by hand (or just have the layout extractor extract one cell).

(2) The user then sets up the subcircuit definition using the **SUBCKT** card in SPICE. All but one cell is replaced by the subcircuit call. (using the X*zzz* element in SPICE).

(3) The layout geometry information is entered into the geometry file for the original cell only. The user is required to identify any "stacked" connections in the circuit. "Stacked" connections are the connections that simultaneously feed current to a number of cells (e.g. power, ground and clock lines). An example of a "stacked" connection is explained in Sec. 5.5.2.

5.5.1 The Format of the User Entered Geometry File

The pre-processor of the EM simulator reads the user-entered geometry file *geometryfile* that is defined by the **.EMSTAT** card in the SPICE input deck. This geometry file will have a different format from the *geometryfile* produced by the extractor program. The format was designed to be more user-friendly.

Now, we will describe the organization of the user-entered geometry file. The format for capacitance, inductance, resistance, voltage and current (and other two-node) elements is:

element [*type* {*width length*} | *openings*] ...

Type is one of MF, MS, MT, VI, V2, CO which represent respectively metal-one, metal-two, metal-three, metal-one to metal-two via, metal-two to metal-three via, and contact. For interconnects, the user enters the width and length in μm. The number of contacts or via openings is entered for contact and vias (the area of the opening is taken to be the same as the area of contact/via opening entered in the **Area=** card for the test structure, see Appendix A). The information for each element must appear on a single line. For example, a capacitance element (labeled C10 in the SPICE deck) which has a first level metal connection 4μm wide 100μm long connected to the second level metal (2μm wide 20μm long)

through via with 2 via openings would be specified as:

C10 MS 2.0 20.0 MF 4.0 100.0 VI 2.0

The format for a transistor element and for subcircuit elements is:

element [**node** *nodenumber* [*type* {*width length*} | *openings*] ...]

Type is one of MF, MS, MT, VI, V2, CO which represent metal-one, metal-two, metal-three, metal-one to metal-two via, metal-two to metal-three via, and contact respectively. For interconnects, the user enters the width and length in μm. In the definition, the number of contact or via openings is entered for each connection (the area of the opening is taken to be the same as the area of contact/via opening entered in the **Area=** card, see Appendix A). For example, a bipolar transistor (three terminal device) (labeled Q1 in the SPICE input deck) with the following metal connections:

(1) Emitter (node number 1) contacting polysilicon,

(2) Base (node number 2) with 1 contact opening to first level metal connection 2μm wide 5μm long.

(3) Collector (node number 3) with 3 contact openings to first level metal connection 2.5μm wide 15μm long. The first level metal later connects to second level metal interconnect 5.0μm wide 100μm long through a via with 2 via openings

can be entered in the geometry file as follows:

Q1 node 3 CO 3.0 MF 2.5 15.0 VI 2.0 MS 5.0 100.0 node 2 CO 1.0 MF 2.0 5.0 node 1

Comments can be inserted in the file but typing * in the first column. For example:

* This is a comment line

5.5.2 "Stacked" Metal Connections

A "stacked" metal connection occurs when a power line (or signal line) successively connects to a number of identical cells (e.g. the 21-stage BiCMOS inverter chain in Fig. 5.3). The metal segments between two adjacent cells all have the same length and width. As a result, the average current density flowing in the metal segments "stack," with the greatest current being in the segments closest to the power or signal source and the least current in the segments connecting the farthest cell. As noted earlier (in Sec.5.1), the user can set up a subcircuit definition for the cell and repeat the subcircuit in the SPICE deck. But to account for the "stacked" connection, the user will have to set up a dummy resistor between two adjacent cells. This resistor will define the geometry information of the segement. This can be quite troublesome if the number of cells is very large.

When entering the geometry information by hand, the user needs to attach a suffix S after *type* followed by a "stacking" parameter *nSTACK* in order to simulate a "stacked" metal connection. For example:

M1 node 2 MFS 21 10.0 30.0

defines a metal-one connection of 21 segments stacked at node 2. Each segment is 10 μm wide 30.0 μm long. When calculating the failure statistics, the simulator will sum the failure rates of the first segment carrying current density J (the result of the SPICE simulation), the second segment carrying current density $2 \times J$, and so on, up to and including the *nStack*-th segment carrying current density $nStack \times J$.

5.6 EXAMPLES OF SIMULATION

In this section, we will demonstrate the operation of the EM simulator in three examples:

(1) We design a CMOS EPROM sense amplifier circuit using the layout guideline from the simulator and make an estimation of the failure rate for the 512K × 8 EPROM.

(2) We assess the failure rate of a 21-stage BiCMOS inverter chain. The layout information in entered by hand. The use of subcircuits and "stacked" connections is demonstrated.

(3) We project the reliability of a CMOS logic circuit designed using arrays of inverters, NOR and NAND gates. The SPICE input deck is extracted from the CIF layout file.

The SPICE input decks and CIF layout files are found in **BERT/EM/Sample** in the distribution tape.

5.6.1 Examples' Design Rules and Reliability Parameters

The EM design rule file for all the examples described later will be **emrule**. The line:

.EMMODEL emrule

is inserted in the SPICE input deck. The file *emrule* is given below:

```
* Reliability parameters for electromigration simulator
PrintCurrent
ncurrent=3 1.00e-4 1.00e-3 5.00e-3
SkipFailpercent
WorstList=0.5
AC_define=0.2
MinJcurrent=1e2
* this is the reliability specifications
nwidth=4 1.0 2.0 4.0 10.0
ncv=3 1.0 2.0 4.0
* The following set for metal-one interconnect
metal1 length=4.5e+4 width=1.0 thick=0.5
Tdata=200.0 Ea=0.5
width_a=6.25 width_b=1.0 width_c=40.0
m=2.0 Adc=7.52e+12 Aac=7.52e+16
Lognormal logsigma=1.0 logmedian=7.52
* The following set for metal-one to silicon contact
contact area=1.0 Nchain=10.0
Tdata=200.0 Ea=0.5
m=2.0 Adc=1.0e+13 Aac=9.0e+14
Lognormal logsigma=1.0 logmedian=7.52
spec_time=1.0e+4 spec_failrate=1.0e-9
```

By using this emrule file the following assumptions and declarations are made.

(1) This emrule file requests a list of currents at each connection.

(2) For the layout advisory, the simulator is instructed to generate guidelines for four interconnect widths: 1μm, 2μm, 4μm and 10μm, and three contact and via openings: 1, 2 and 4.

(3) The emrule file requests that layout guidelines be generated for three current values: 0.1 mA, 1.0 mA and 5.0 mA.

(4) The simulator is instructed to ignore any connections with current density less than 1e2 A/cm^2.

(5) A pure AC waveform is defined if the average current is less than 20% of the average absolute current.

(6) The width dependence was extracted from the example in Appendix E.

(7) The metal-one line used in the electromigration lifetime experiment was 4.5 cm long, 2 μm wide and 0.5 μm thick.

(8) A_{DC}, A_{AC} and m were obtained from accelerated testing at 200°C.

(9) A lognormal failure distribution will be used. For example, $\sigma = 1.0$ and MTF = 7.52 hours (using $A_{DC}=7.52\times10^{12}$, the DC stress current is 10^6 A/cm^2 at 200°C in the experiment).

(10) The contact chain structure used in the experiment had ten $1\mu m^2$ contact openings in series.

(11) The reliability requirement for the circuit is a failure rate of 1 FIT (10^{-9} failure / device hour) at 10^4 hours (1.1 year).

The parameters for metal-two, metal-three and metal-to-metal vias are not given. The simulator will not perform analysis for these structures.

5.6.2 CMOS EPROM Sense Amplifier

In this example, the designer wishes to generate a layout guideline for a CMOS EPROM sense amplifier circuit (see Fig. 5.2) and estimate the electromigration reliability of a 512K×8 EPROM design.

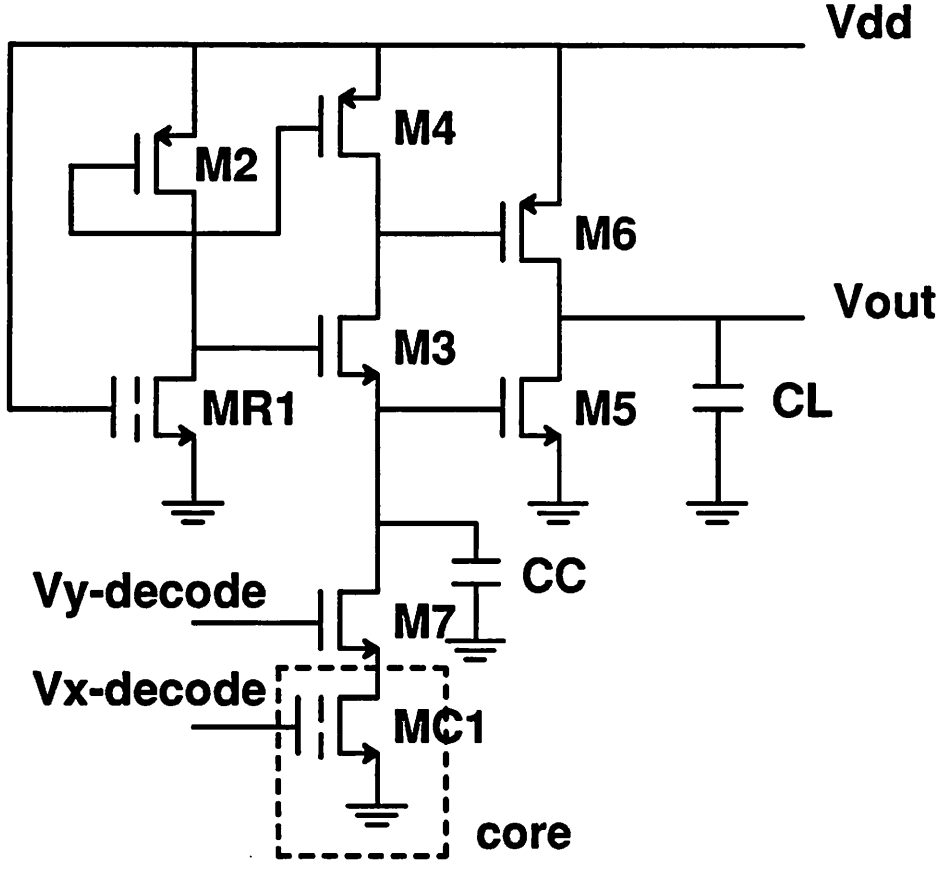


Figure 5.2. EPROM sense amplifier

The SPICE input deck (in file **eprom**) is set up as follows:

```
CMOS EPROM SENSE AMPLIFIER

VDD 5 0 DC 5
VX 6 0 DC 0 PWL(0 0 5NS 5 55NS 5 60NS 0 150NS 0 155NS 5 205NS 5
+ 210NS 0 300NS 0)
VY 7 0 DC 0 PWL(0 0 5NS 5 55NS 5 60NS 0 150NS 0 155NS 5 205NS 5
+ 210NS 0 300NS 0)


MR1 2 5 0 0 EPROM L=2U W=3U
+ AD=20P AS=20P PD=14U PS=14U NRD=1.25 NRS=1.25
M2 2 2 5 5 MODP L=2U W=11U
+ AD=60P AD=60P PD=22U PS=22U NRD=0.42 NRS=0.42
M3 3 2 1 0 MODN L=2U W=39U
+ AD=200P AD=200P PD=50U PS=50U NRD=0.13 NRS=0.13
M4 3 2 5 5 MODP L=2U W=7U
+ AD=40P AD=40P PD=18U PS=18U NRD=0.63 NRS=0.63
M5 4 1 0 0 MODN L=2U W=49U
+ AD=250P AD=250P PD=60U PS=60U NRD=0.1 NRS=0.1
M6 4 3 5 5 MODP L=2U W=11U
+ AD=60P AD=60P PD=22U PS=22U NRD=0.42 NRS=0.42
M7 1 7 8 0 MODN L=2U W=49U
+ AD=250P AD=250P PD=60U PS=60U NRD=0.1 NRS=0.1
MC1 8 6 0 0 EPROM L=2U W=3U
+ AD=20P AS=20P PD=14U PS=14U NRD=1.25 NRS=1.25

CC 1 0 2P
CL 4 0 0.3P

.MODEL MODN NMOS (LEVEL=2 VTO=0.9 KP=36U GAMMA=0.16 PHI=0.58 LAMBDA=0.04
+  CGSO=2.3E-10 CGDO=2.3E-10 CGBO=1.0E-10 RSH=30 CJ=1E-4 CJSW=3E-10
+  JS=3E-9 TOX=300E-10 NSUB=1E15 NFS=2E10 XJ=0.3E-6 LD=0.2E-6 UO=310
+  VMAX=6E4)

.MODEL MODP PMOS (LEVEL=2 VTO=-0.9 KP=17U GAMMA=0.27 PHI=0.63 LAMBDA=0.06
+  CGSO=3.4E-10 CGDO=3.4E-10 CGBO=1.0E-10 RSH=120 CJ=1.7E-4 CJSW=6.7E-10
+  JS=1E-9 TOX=300E-10 NSUB=3E15 NFS=2E10 XJ=0.4E-6 LD=0.4E-6 UO=150
+  VMAX=8E4)

.MODEL EPROM NMOS (LEVEL=2 VTO=2.0 KP=14U GAMMA=0.16 PHI=0.58 LAMBDA=0.04
+  CGSO=2.3E-10 CGDO=2.3E-10 CGBO=1.0E-10 RSH=30 CJ=1E-4 CJSW=3E-10
+  JS=3E-9 TOX=300E-10 NSUB=1E15 NFS=2E10 XJ=0.3E-6 LD=0.2E-6 UO=310
+  VMAX=6E4)

.EMModel emrule
.WIDTH OUT=80
.TRAN 1NS 300NS 150NS
.OPTIONS NODE METHOD=GEAR VNTOL=0.001 ABSTOL=1.0E-8
.PLOT TRAN V(4) V(1) V(3) V(6) (0,5)
.END
```

We have set up the input voltages and the transient analysis card so that SPICE will simulate the circuit for two read cycles (each read cycle is 150 ns). Keep in mind that in the actual read cycle of the EPROM, only one out of many cells will be accessed. Since only layout guidelines are desired, we have omitted the .EMSTAT card in the SPICE deck. The command to call the simulator (using SPICE2G6) is:

```
>prebert -2 eprom | spice2g6 | postbert > eprom.em
```

or using the "standalone" version of the EM simulator, which saves the SPICE output:

```
>preem eprom | spice2g6 > eprom.spcout
>postem -S2 -G eprom.geo -R emrule eprom.spcout > eprom.em
```

A partial listing of the output current table is given below:

<Current Table>

Connection	Avg Current(A)	Avg Abs Cur(A)	
VDD to 5	2.56e-04	2.56e-04	
MR1 to 2	1.12e-04	1.12e-04	
MR1 to 5	1.49e-10	2.22e-09	AC
MR1 to 0	1.12e-04	1.12e-04	
M2 to 2	1.11e-04	1.11e-04	
M2 to 2	2.98e-09	4.47e-08	AC
M2 to 5	1.11e-04	1.11e-04	
M5 to 4	1.09e-04	1.09e-04	
M5 to 1	4.80e-07	2.29e-06	
M5 to 0	1.09e-04	1.10e-04	
M6 to 4	1.07e-04	1.07e-04	
M6 to 3	3.48e-07	1.12e-06	
M6 to 5	1.06e-04	1.06e-04	
M7 to 1	2.98e-05	3.64e-05	
M7 to 7	2.19e-06	2.38e-05	AC
M7 to 8	2.98e-05	3.03e-05	
MC1 to 8	2.98e-05	3.03e-05	
MC1 to 6	9.04e-08	1.72e-06	AC
MC1 to 0	2.98e-05	3.02e-05	

From our results we can see that the currents in the gates of MR1, M2, M7, and MC1 are treated as pure AC current. But, note that this is not the case for the current in the gates of M5 and M6. Therefore, the SPICE simulation might need to be extended to more cycles in order to obtain steady-state current waveforms. Another solution would be to set **AC_Define** to a higher value.

The first layout advisory table gives the maximum allowable length for each interconnect, and the safety factor for each contacts and vias as a function of current passing through:

<Layout Advisory Table for Current>

Maximum Interconnect Length (in micron) or Safety Factor for Contacts or Vias to guarantee 1.00e-09(/h) failure rate at time 1.00e+04(h) for given Current(A)

DC Current(mA)	1.0e-04	1.0e-03	5.0e-03
M1 W = 1.0	2.2e+13	2.0e+02	5.6e-01

M1 W = 2.0	9.4e+20	8.9e+05	8.0e+00
M1 W = 4.0	1.9e+35	2.5e+14	1.9e+05
M1 W = 10.0	1.9e+60	3.8e+31	9.6e+16
No.of CONT=1.0	9.5e+15	3.9e+01	9.7e-04
No.of CONT=2.0	1.3e+22	9.0e+04	2.7e-02
No.of CONT=4.0	1.2e+29	1.4e+09	4.9e+00

Let us see interpret some of these results. The maximum length allowed for a 2μm wide metal-one interconnect carrying 5 mA current is 8μm long. And the safety factor for 4 contact openings is 4.9 if 5 mA of current flows through the contact (each contact opening carries 1.25 mA).

The second table gives layout guidelines for every connection in the circuit. The following is a partial listing (the least constraining guidelines have been omitted):

<Layout Advisory Table for all connections>
Maximum Interconnect Length (in micron) or Safety Factor for Contacts or Vias to guarantee 1.00e-09(/h) failure rate at time 1.00e+04(h) in each connection

Metal: (Width)	Type	1.0e+00	2.0e+00	4.0e+00	1.0e+01
Cont/Via:(#)	Type	1.0e+00	2.0e+00	4.0e+00	
VDD to 5	MF	5.4e+07	5.4e+13	4.5e+25	4.5e+30
VDD to 5	CO	1.0e+09	1.0e+14	7.0e+19	
MR1 to 2	MF	3.9e+12	1.1e+20	4.5e+30	4.5e+30
MR1 to 2	CO	1.2e+15	1.2e+21	1.0e+27	
MR1 to 5	MF	J <MinJ	J <MinJ	J <MinJ	J <MinJ
MR1 to 5	CO	J <MinJ	J <MinJ	J <MinJ	
MR1 to 0	MF	3.9e+12	1.1e+20	4.5e+30	4.5e+30
MR1 to 0	CO	1.2e+15	1.2e+21	1.0e+27	
M2 to 2	MF	4.2e+12	1.1e+20	4.5e+30	4.5e+30
M2 to 2	CO	1.2e+15	1.3e+21	1.0e+27	
M2 to 2	MF	J <MinJ	J <MinJ	J <MinJ	J <MinJ
M2 to 2	CO	J <MinJ	J <MinJ	J <MinJ	
M2 to 5	MF	4.2e+12	1.1e+20	4.5e+30	4.5e+30
M2 to 5	CO	1.2e+15	1.3e+21	1.0e+27	
M5 to 4	MF	6.1e+12	1.9e+20	4.5e+30	4.5e+30
M5 to 4	CO	2.0e+15	2.2e+21	1.0e+27	
M5 to 1	MF	4.5e+30	4.5e+30	4.5e+30	J <MinJ
M5 to 1	CO	1.0e+27	1.0e+27	J <MinJ	
M5 to 0	MF	5.4e+12	1.6e+20	4.5e+30	4.5e+30
M5 to 0	CO	1.7e+15	1.8e+21	1.0e+27	
M6 to 4	MF	8.2e+12	2.7e+20	4.5e+30	4.5e+30
M6 to 4	CO	2.8e+15	3.2e+21	1.0e+27	
M6 to 3	MF	4.5e+30	4.5e+30	J <MinJ	J <MinJ
M6 to 3	CO	1.0e+27	J <MinJ	J <MinJ	
M6 to 5	MF	8.6e+12	2.8e+20	4.5e+30	4.5e+30
M6 to 5	CO	3.0e+15	3.4e+21	1.0e+27	
M7 to 1	MF	9.1e+21	4.5e+30	4.5e+30	4.5e+30
M7 to 1	CO	1.6e+26	1.0e+27	1.0e+27	
M7 to 7	MF	4.5e+30	4.5e+30	4.5e+30	4.5e+30
M7 to 7	CO	1.0e+27	1.0e+27	1.0e+27	

M7 to 8	MF	5.8e+22	4.5e+30	4.5e+30	4.5e+30
M7 to 8	CO	1.0e+27	1.0e+27	1.0e+27	
MC1 to 8	MF	5.8e+22	4.5e+30	4.5e+30	4.5e+30
MC1 to 8	CO	1.0e+27	1.0e+27	1.0e+27	
MC1 to 6	MF	4.5e+30	4.5e+30	J <MinJ	J <MinJ
MC1 to 6	CO	1.0e+27	J <MinJ	J <MinJ	
MC1 to 0	MF	6.2e+22	4.5e+30	4.5e+30	4.5e+30
MC1 to 0	CO	1.0e+27	1.0e+27	1.0e+27	

As pointed out earlier, because the gate current in M5 and M6 is not a pure AC waveform, the recommendations for the gate of M5 (M5 to 1) and gate of M6 (M6 to 3) are too pessimistic.

From the table above, the designer can layout the circuit and also estimate the failure rate of the layout using the table. The interconnect failure rate for the design at 10^4 hours is:

$$\text{Designed FIT} = \frac{\text{Layout Length}}{\text{L}} \times 1 \text{ FIT}$$

where L is the maximum length allowed from the layout advisory table. 1 FIT is used on the R.H.S. of the equation because the layout guidelines are generated for the spec of 1 FIT in 10^4 hours. The contact/via failure rate for the design is:

$$\text{Designed FIT} = \frac{1}{S} \times 1 \text{ FIT}$$

where S is the safety factor from the layout advisory table.

We will calculate the failure rate at 10^4 hours for a 512K× 8 EPROM assuming the worst case scenario: the same 8 EPROM bits located furthest away from the sense amplifier are continuously accessed in each read cycle. The geometry of interconnects and contacts in the chip are:

(1) All the transistors in the sense amplifier circuit (a total of 8 sense amplifier circuits) have 20μm and 1μm wide metal-one lines connected to their source and drain.
Failure rate for these:

$$F_a = 8 \times 20 \times \left[\frac{1}{3.9e12} + \frac{1}{3.9e12} + \frac{1}{4.2e12} + \frac{1}{4.2e12} + \frac{1}{6.1e12} + \right.$$
$$\left. + \frac{1}{5.4e12} + \frac{1}{8.2e12} + \frac{1}{8.6e12} \right]$$
$$= 2.5 \times 10^{-10} = \text{FIT}$$

(2) The Vdd line for each (total of 8) sense amplifier is 200 μm long 2μm wide.

$$F_b = 8 \times \frac{200}{5.4e13}$$
$$= 3.0 \times 10^{-11} = \text{FIT}$$

(3) The bitline (M7 to 8) is 1 μm wide 1 cm long.

$$F_c = 8 \times \frac{10^4}{5.8e22}$$
$$= 1.4 \times 10^{-18} = \text{FIT}$$

(4) In the core cell of the EPROM (the same 8 bits are accessed all the time) the gate, source and drain connections are each 3μm long 1μm wide.

$$F_d = 8 \times 3 \times \left[\frac{1}{5.8e22} + \frac{1}{6.2e22} \right]$$

$$= 8.0 \times 10^{-22} = \text{FIT}$$

(5) The source and drain of each transistor has one contact opening (of area $1\mu m^2$).

$$F_e = 8 \times \left[\frac{1}{1.2e15} + \frac{1}{1.2e15} + \frac{1}{1.2e15} + \frac{1}{1.2e15} + \frac{1}{2.0e15} + \frac{1}{1.7e15} + \frac{1}{2.8e15} + \frac{1}{3.0e15} + \cdots \right]$$

$$= 4.0 \times 10^{-14} = \text{FIT}$$

Thus, the FIT of the design is:

$$\text{Designed FIT} = F_a + F_b + F_c + F_d + F_e \approx 0 \text{ FIT@ } 10^4 \text{ hours}$$

We see that this small circuit is not expected to suffer any electromigration failures in 10^4 hours. However, if this small circuit were repeated millions of times inside a single IC (a realistic scenario) then the FIT would rise to measurable numbers. We can make a better estimation by assuming a certain probability that the bit is accessed and the associated bit line is activated, and then run the SPICE simulation over a duration that covers both the active cycles and the idle cycles. This will produce a more accurate average current flowing in each connection. For example, if we know on the average that each EPROM cell will be accessed once every 100 read cycles and that each bit line is activated once every 10 read cycles, a SPICE deck can be set up consisting of three EPROM bits (bit A, bit B and bit C). Bit A and bit B are on the same bitline, bit C is on a different bitline (but connects to the same sense amplifier). Bit A is accessed once and left idle for 99 cycles, bit B is accessed for 9 cycles and left idle for 91 cycles, and bit C is accessed for 90 cycles and left idle during the 10 cycles when A and B are activated. The three bits are set up to continuously provide a signal to the sense amplifier and to activate bit A's bitline once every 10 cycles. The transient duration requested in the SPICE input deck is 100 cycles. The total failure rate can be calculated by summing the failure rate of all the bits (i.e. multiply the failure rate of bit A by 512K × 8), the failure rate of the bit lines (i.e. multiply the failure rate of bit A's bitline by the total number of bit lines) and the failure rate of the 8 sense amplifiers.

5.6.3 21-Stage BiCMOS Inverter Chain

In this example, the simulator is used to generate layout guidelines and to calculate the failure rate of a 21-stage BiCMOS inverter chain (Fig. 5.3). The SPICE deck is found in file **bicmos**:

```
21-stage BiCMOS output inverter
vdd 98 0 dc 5.5
vin 4 0 dc 0.0 pulse (0.0 5.0 1n 10p 10p 7n 15n)
rdummy1 98 1 0.001
rdummy2 99 0 0.001
q1 1 2 5 0 npn1
q2 5 3 99 0 npn1
m1 1 4 2 1 pmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
m2 1 1 2 1 pmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
m3 2 4 6 0 nmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
```

```
m4 6 1 99 0 nmos w=15u l=1.2u ad=30p as=60p pd=4u ps=23u
m5 5 4 7 99 nmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
m6 7 1 3 0 nmos w=15u l=1.2u ad=30p as=60p pd=4u ps=23u
m7 3 5 99 0 nmos w=15u l=1.2u ad=60p as=60p pd=23u ps=23u
cl 5 0 0.5p

x1 5 8 1 0 inverter
x2 8 9 1 0 inverter
x3 9 10 1 0 inverter
x4 10 11 1 0 inverter
x5 11 12 1 0 inverter
x6 12 13 1 0 inverter
x7 13 14 1 0 inverter
x8 14 15 1 0 inverter
x9 15 16 1 0 inverter
x10 16 17 1 0 inverter
x11 17 18 1 0 inverter
x12 18 19 1 0 inverter
x13 19 20 1 0 inverter
x14 20 21 1 0 inverter
x15 21 22 1 0 inverter
x16 22 23 1 0 inverter
x17 23 24 1 0 inverter
x18 24 25 1 0 inverter
x19 25 26 1 0 inverter
x20 26 27 1 0 inverter

.subckt inverter 4 5 1 99
* 4 is input 5 is output 1 is +supply 99 is -supply
qs1 1 2 5 0 npn1
qs2 5 3 99 0 npn1
ms1 1 4 2 1 pmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
ms2 1 1 2 1 pmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
ms3 2 4 6 0 nmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
ms4 6 1 99 0 nmos w=15u l=1.2u ad=30p as=60p pd=4u ps=23u
ms5 5 4 7 99 nmos w=15u l=1.2u ad=60p as=30p pd=23u ps=4u
ms6 7 1 3 0 nmos w=15u l=1.2u ad=30p as=60p pd=4u ps=23u
ms7 3 5 99 0 nmos w=15u l=1.2u ad=60p as=60p pd=23u ps=23u
cs1 5 0 0.5p
.ends
.nodeset v(5)=4.796 v(8)=0.531 v(9)=4.796 v(10)=0.531 v(11)=4.796 v(12)=0.531
+ v(13)=4.796 v(14)=0.531 v(15)=4.796 v(16)=0.531 v(17)=4.796 v(18)=0.531
+ v(19)=4.796 v(20)=0.531 v(21)=4.796 v(22)=0.531 v(23)=4.796 v(24)=0.531
+ v(25)=4.796 v(26)=0.531 v(27)=4.796
.width out=80

.model npn1 npn (is=5.0e-18 bf=86 br=10 cje=11f cjc=15f cjs=135f
+ tf=5p vaf=40 rb=50 re=25 rc=180 ikf=0.01)
.model nmos nmos (level=2 vto=0.58 kp=80u gamma=0.16 phi=0.58 lambda=0.04
+   cgso=2.3e-10 cgdo=2.3e-10 cgbo=1.0e-10 rsh=30 cj=1E-4 cjsw=3e-10
+   js=3e-9 tox=200e-10 nsub=1e15 nfs=2e10 xj=0.3e-6 ld=0.0e-6 uo=310
+   vmax=6e4)
.model pmos pmos (level=2 vto=-0.52 kp=27u gamma=0.27 phi=0.63 lambda=0.06
+   cgso=3.4e-10 cgdo=3.4e-10 cgbo=1.0e-10 rsh=120 cj=1.7e-4 cjsw=6.7e-10
```

```
+ js=1e-9 tox=200e-10 nsub=3e15 nfs=2e10 xj=0.4e-6 ld=0.0e-6 uo=150
+ vmax=8e4)
.emmodel emrule
.emstat bicmos.int 1.0e2 5.0e2 1.0e3 2.0e3

.tran 100p 20n
.plot tran v(27)
.end
```

Because the circuit has 21 identical inverter cells, we set up a subcircuit for the inverter stage. All but one of the 21 inverter cells are represented by the subcircuit elements. To account for the "stacked" Vcc power line, we inserted a dummy resistance rdummy1 in the first cell to represent the connection. In the geometry file, the connection of rdummy1 is indicated to be metal-one 7.5μm wide and 30μm long and with a "stack" parameter of 21. The same procedure is applied to the ground line of the first cell. The geometry file set up by the user in **bicmos.int** is:

```
rdummy1 MFS 21 7.5 30.0
rdummy2 MFS 21 7.5 30.0
q1 node 1 CO 2.0 node 2 CO 2.0 MF 2.0 10.0 node 5 CO 2.0 MF 2.0 10.0
q2 node 5 CO 2.0 MF 2.0 10.0 node 3 CO 2.0 MF 2.0 10.0 node 99 CO 2.0
m1 node 1 CO 2.0 MF 2.0 10.0 node 4 CO 2.0 MF 2.0 15.0 node 2 CO 2.0
m2 node 1 CO 2.0 MF 2.0 10.0 node 1 CO 2.0 node 2 CO 2.0
m3 node 2 CO 2.0 MF 2.0 10.0 node 4 CO 2.0 MF 2.0 10.0 node 6
m4 node 6 node 1 CO 2.0 node 99 CO 2.0 MF 2.0 10.0
m5 node 5 CO 2.0 MF 2.0 10.0 node 4 CO 2.0 MF 2.0 10.0 node 7
m6 node 7 node 1 CO 2.0 node 3 MF 4.0 25.0 CO 2.0
m7 node 3 CO 2.0 MF 2.0 10.0 node 5 CO 2.0 MF 2.0 10.0 node 99 CO 2.0
c1 MF 2.0 10.0
```

The command line to invoke the simulator is:

```
>prebert bicmos | spice3c1 | postbert > bicmos.em
```

or using the "standalone" version of the simulator:

```
>preem bicmos | spice3c1 > bicmos.spcout
>postem -G bicmos.geo -R emrule bicmos.spcout > bicmos.em
```

The failure rate table calculated by the simulator in **bicmos.em**:

<Failure Rate (/hour) Table>

No	Conn. at time	1.0e+02	1.0e+03	1.0e+04	1.0e+05
MFS0	rdummy1 to 98	2.1e-11	2.3e-08	2.4e-07	1.6e-07
MFS1	rdummy2 to 99	1.4e-25	3.4e-18	4.5e-13	3.6e-10
C5	q1 to 5	2.2e-33	6.4e-24	9.1e-17	6.4e-12
MF6	q1 to 5	6.6e-31	1.6e-22	2.0e-16	1.2e-12
MF15	m1 to 4	5.3e-36	1.6e-26	2.5e-19	1.9e-14
C16	m1 to 2	2.8e-38	6.9e-28	8.4e-20	5.1e-14

MF27	m4 to 99	2.4e-36	8.0e-27	1.3e-19	1.1e-14
MF29	m5 to 5	1.8e-37	1.0e-27	2.7e-20	3.7e-15
Total Failure Rate(/h):		2.1e-11	2.3e-08	2.4e-07	1.6e-07

The worst electromigration hazard in the design is in the Vcc and ground lines. Since we have only simulated one of the 21 stages. The failure rate at 10^4 hours (1.1 year) for the circuit is the sum of the failure rates for the supply and ground lines and 21× of the total failure rate for each cell:

$$\text{Total Failure Rate} = \Sigma_{cell}F_i + F_{vdd} + F_{ground}$$
$$= 21 \times 2.5 \times 10^{-16} + 2.4 \times 10^{-7} + 4.5 \times 10^{-13}$$
$$= 2.4 \times 10^{-7} = 2.4 \times 10^2 \text{ FIT @ } 10^4 \text{ hours}$$

Note that the failure rate of the Vcc line decreases after 10^4 hours. This is caused by use of the lognormal distribution. The failure rate for a lognormal distribution can first increase, reach a maximum and decrease. Therefore, the failure statistics beyond 10^4 hours may be invalid.

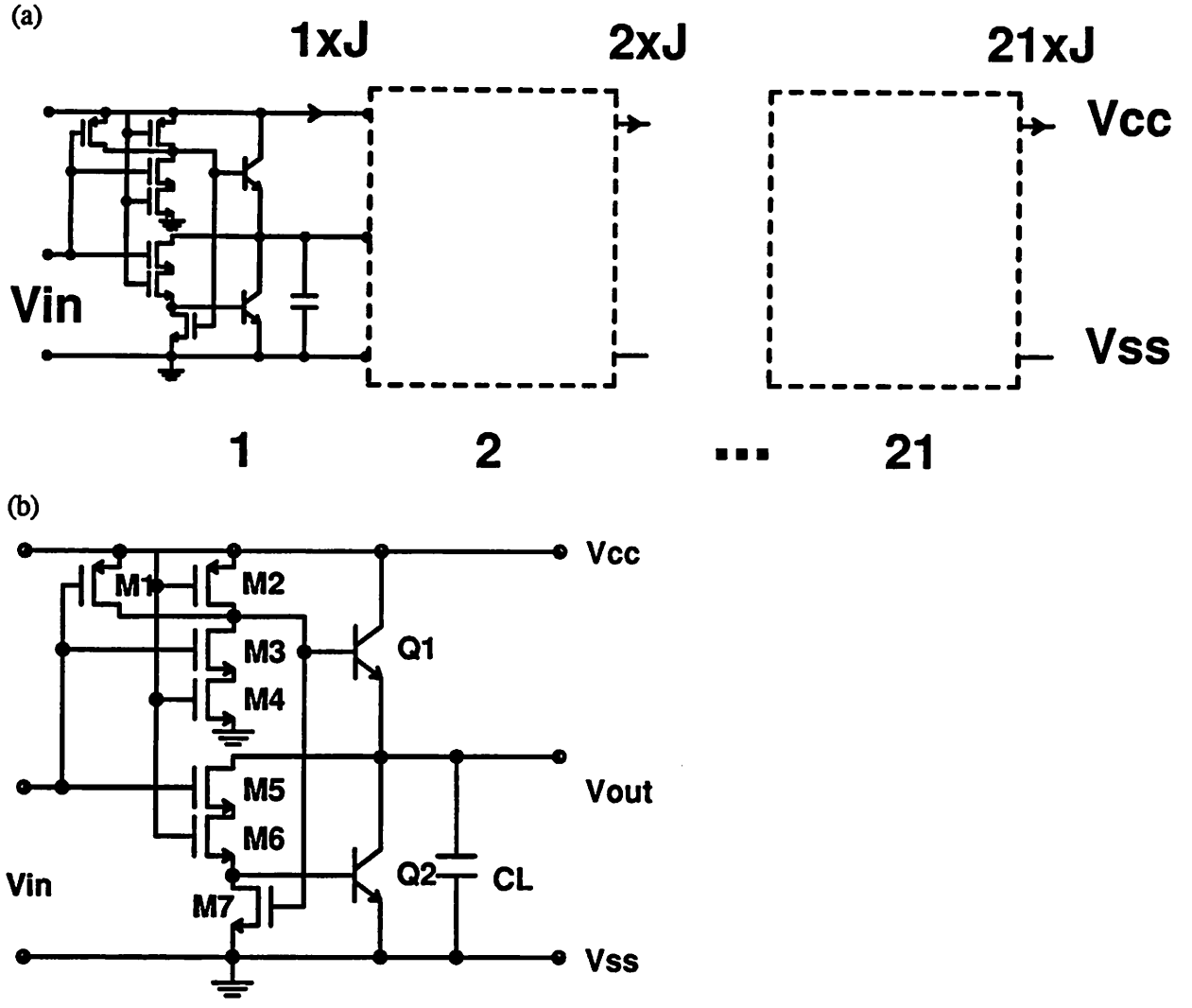


Figure 5.3 (a) 21-stage BiCMOS inverter chain. An example of "stacked" connection is shown. The first segment of the power line in the first cell on the left carries current density J, and the current density increases towards the Vcc source. The last segment of the power line carries 21× current density. (b) the circuit of one inverter stage.

5.6.4 CMOS Logic Circuit Using Inverters, NOR and NAND Gates.

The circuit shown in Figure 5.4 is designed using a scmos process. The CIF file for the layout has been generated (see Fig. 5.5) and is in file **circ.cif**. We will assess the reliability of the layout design using the EM simulator. First, the SPICE deck is extracted from the CIF layout. The following commands are used to invoke the extractor which also produces the layout geometry information:

```
>mextra -t scmos circ.cif
>sim2spice circ.sim
```

Input voltages, power supply voltages and ground nodes have to be added to complete the SPICE deck (in **circ.spice**). The completed SPICE deck is in file **circ.ospice**. In order to trace the input nodes and power buses, the extracted SPICE nodes (which are written to **circ.spcnode** by **sim2spice**) are printed on top the original CIF layout file using **cif2ps** with the option **-m**:

```
>cif2ps -m -t scmos circ.cif circ.spcnode | lpr
```

The output is sent to a PostScript printer and the printout is shown in Figure 5.5. Polysilicon, diffusion and well layers have one node number assigned to each electrically connected area. The number is usually printed at the center lower edge of the layer. Node numbers on metal layers are found at the corners, intersections or contacts where the extractor fractures the interconnect. At contacts and vias, two node numbers are assigned, the node number for the layer on top of the contact is printed at the upper right corner of the contact or via. The node number for the lower layer is usually at the lower right.

The commands for the simulator are:

```
>prebert circ.ospice | spice3c1 | postbert > circ.em
```

or using the "standalone" version of the EM simulator:

```
>preem circ.ospice | spice3c1 | postem -G circ.geo -R emrule > circ.em
```

The simulator prints a listing of the worst reliability hazards (the number of connections printed is determined by the **WorstList=** card in the rule file) after the failure rate or cumulative percent failure table. In the listing, the name of the connection (beginning with **C,V, MF** or **MS** for contact, via, metal-one, metal-two respectively) is given with its location in CIF coordinates.

The following is the list of the worst 10% (**WorstList=** 0.1) of the connections from **circ.em**. The connection name is followed by a dash and ranking number. The lines starting with **DS** and ending with **E** can be saved to a file and printed out on top of the original CIF file using **cif2ps**. The failure rate (/hour) of the connection is enclosed in brackets in the following line:

```
DS 94 1 1;
9 WORST_STAT;
94 C63-1 -1650 23700;
( 5.70e-39 )
94 C42-2 -1650 15750;
( 5.40e-39 )
94 C21-3 -1650 7800;
( 5.30e-39 )
94 C0-4 -1650 -175;
( 5.10e-39 )
94 C64-5 -450 23700;
( 1.40e-40 )
94 C43-6 -450 15750;
( 1.30e-40 )
94 C22-7 -450 7800;
( 1.30e-40 )
94 C1-8 -450 -175;
( 1.20e-40 )
94 MF215-9 187 24900;
( 8.90e-42 )
94 MF173-10 187 16950;
( 8.40e-42 )
94 MF133-11 187 9000;
( 8.10e-42 )
94 MF93-12 187 1050;
( 8.10e-42 )
94 C70-13 1050 24675;
```

```
( 1.10e-42 )
94 C49-14 1050 16725;
( 9.90e-43 )
94 C28-15 1050 8775;
( 9.60e-43 )
94 C7-16 1050 825;
( 9.50e-43 )
94 MF202-17 2550 22237;
( 5.60e-44 )
94 MF154-18 2550 14287;
( 5.30e-44 )
94 MF120-19 2550 6337;
( 5.20e-44 )
94 MF80-20 2550 -1612;
( 5.00e-44 )
94 MF124-21 16050 6337;
( 3.60e-44 )
94 MF206-22 16050 22237;
( 3.40e-44 )
94 MF158-23 16050 14287;
( 3.40e-44 )

DF;
C 94;
E
```

This listing is cut to file **circ4.worst** and printed together with the CIF file (see Fig. 5.6) by:

```
>cif2ps -m -t scmos circ.cif circ.worst | lpr
```

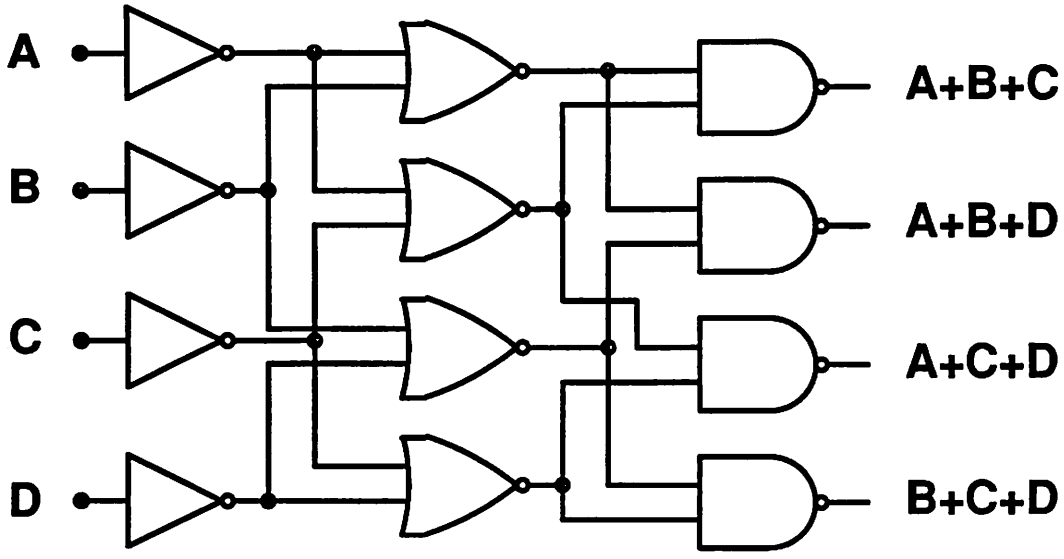


Figure 5.4 CMOS logic circuit using inverters, NOR and NAND gates

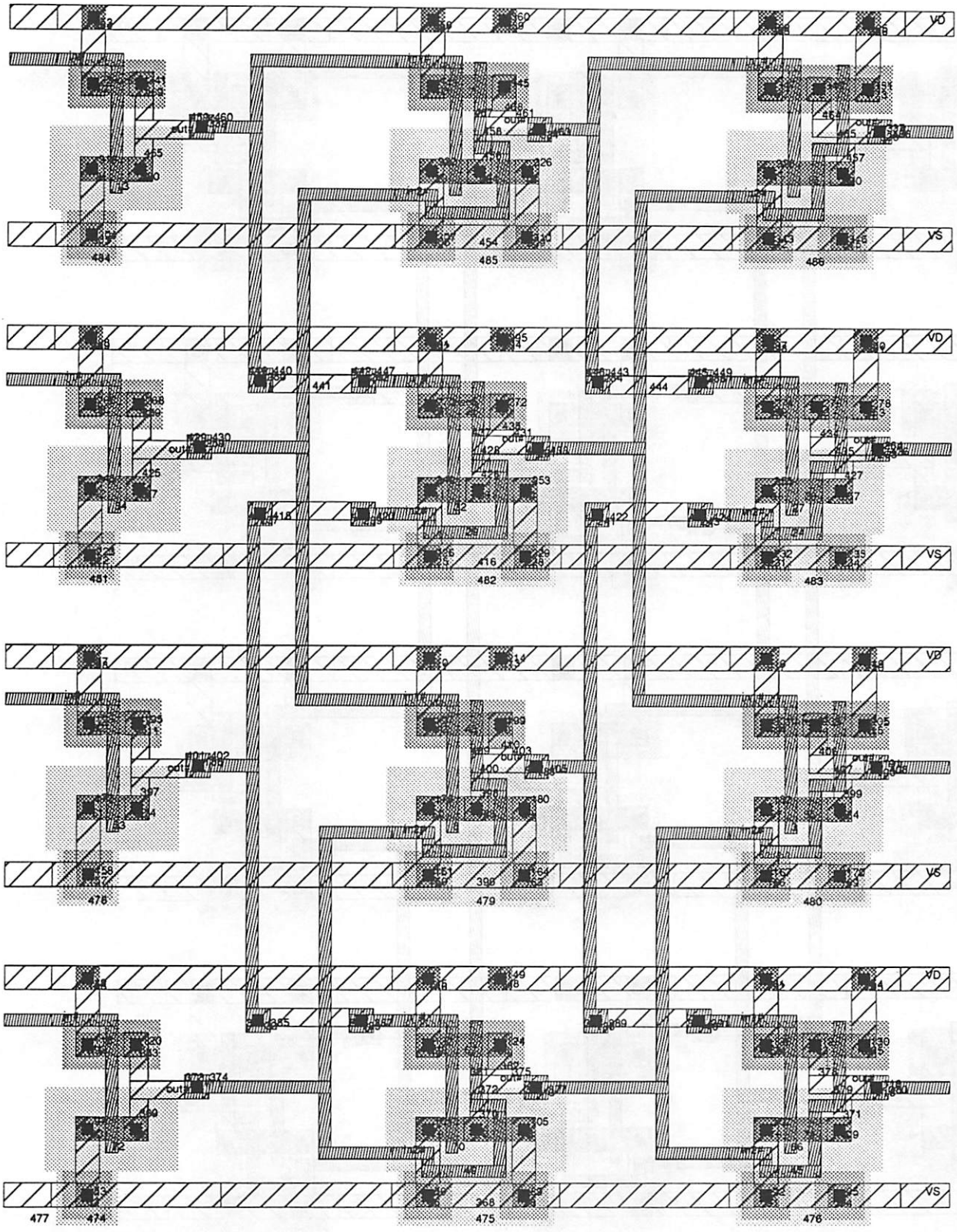


Figure 5.5 SPICE node numbers are plotted on top of the **circ.cif** file using **cif2ps**

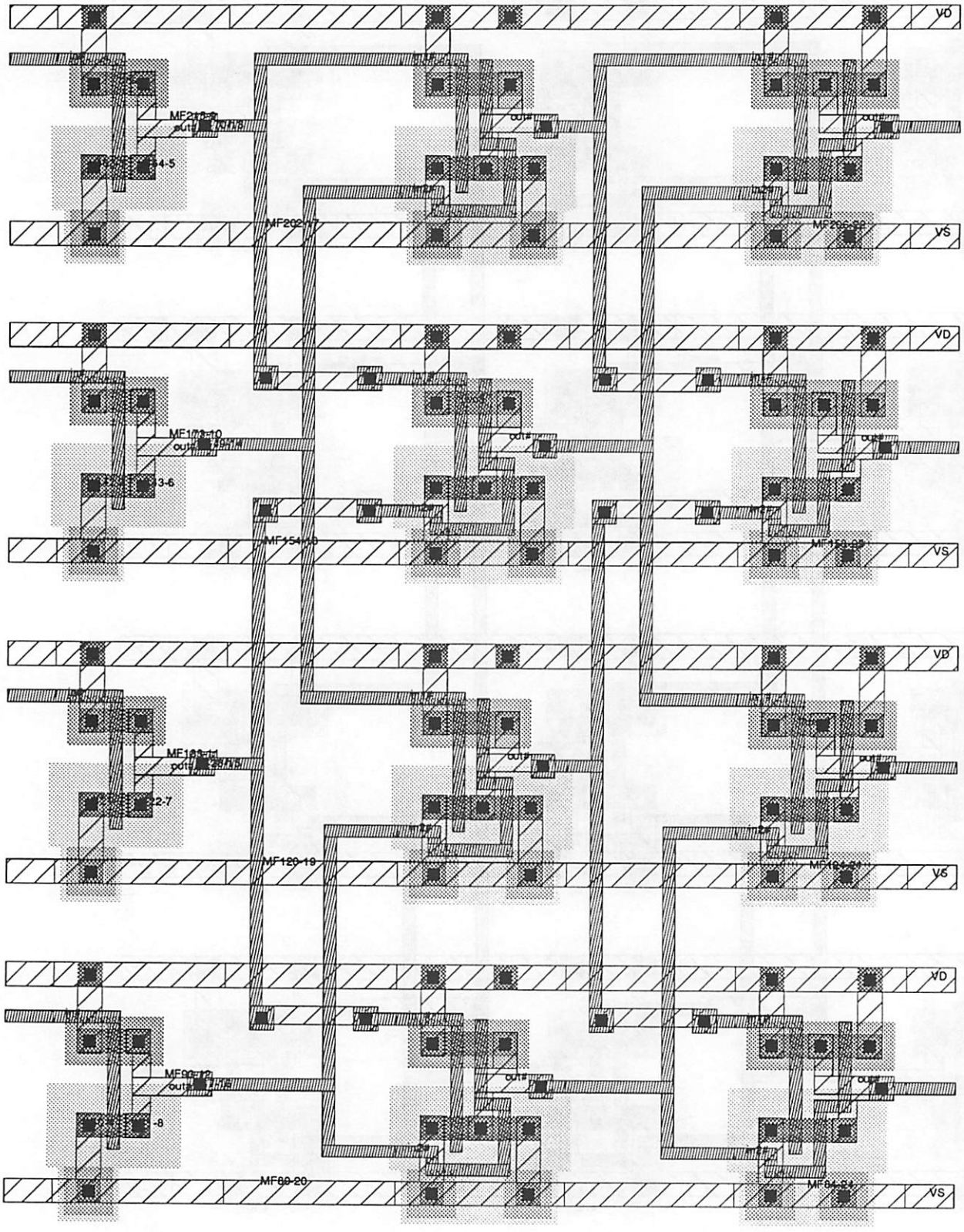


Figure 5.6 The worst 10% of the connections from the EM Simulator in **circ.cif**.

VI. Bipolar Circuit Aging Simulator - (BiCAS)

Since the last release, a new module, BiCAS (Bipolar Circuit Aging Simulator) has been developed. BiCAS, which is organized in the same manner as CAS [72], projects bipolar transistor degradation and simulates circuit aging due to this degredation. To use the module, the user needs to provide an input deck, which contains the circuit description, device models, and the BiCAS commands. CAS, EM, and CORS commands may also be included in the same input deck. Therefore, both bipolar and BiCMOS circuit aging can be simulated. BiCAS uses the regular bipolar junction transistor SPICE model parameters with the addition of a few degradation parameters.

Section 6.1 of this manual describes the equations and models implemented in BiCAS. Section 6.2 describes the procedure for obtaining the degradation parameters. Section 6.3 details the format of the process files and aged process files. Section 6.4 gives an example of a BiCAS simulation.

6.1 BIPOLAR CIRCUIT AGING MODEL

A model for BJT aging has been implemented in BiCAS. This aging can be modeled as an increase in I_B (ΔI_B) [76,77]. BiCAS can also generate "aged" BJT model cards, so that the operation of the aged circuit can be simulated by SPICE.

6.1.1 Device Degradation Model

As shown in [76,77], the bipolar degradation due to either periodic time-varying or constant reverse-bias emitter-base stressing can be modeled by the following equation.

$$\Delta I_B = D(I_S/A_E)^a \, e^{aV_{BE}/V_t} \left[(\#\text{ of cycles}) \cdot \int_{1\text{ cycle}} I_R^{b/n}(t)\, dt \right]^n \tag{6.1}$$

It is important to keep in mind that V_{BE} in Equation (6.1) is the internal base-emitter voltage and not the applied voltage. The parameters D and b change depending upon the level of the reverse bias [76,77]. If the reverse bias puts the transistor in the pre-avalanche region, then $b \approx 0.9$–1.0. If the transistor is in the heavy avalanche region ($V_R > BV_{EBO}$), then $b \approx n$. To account for this variation, Equation (6.1) can be rewritten as

$$\Delta I_B = I_S^a \left[(\#\text{ of cycles}) \cdot \int_{1\text{ cycle}} (D')^{1/n} I_R^{b/n}(t)\, dt \right]^n e^{aV_{BE}/V_t} \tag{6.2}$$

where $D' = D/A_E^a$. By defining (A_E in μm^2), we can reduce the number of parameters by one.

As in the CAS model [72], the variable Age is introduced for the SPICE analysis.

$$\Delta I_B(T) = g(\text{Age}(T)^n) \tag{6.3}$$

with

$$\text{Age}(T) = I_S^{a/n} \int_{t=0}^{T} (D')^{1/n} I_R^{b/n}(t)\, dt \tag{6.4}$$

and

$$g(x) = x \cdot e^{aV_{BE}/V_t} \tag{6.5}$$

where T is the length of the SPICE analysis. g(x) is defined such that ΔI_B can be easily calculated for different values of V_{BE}. Equation (6.4) is calculated for each bipolar device in the circuit. I_R is calculated at each timestep by first calculating V_R from the SPICE node voltages and then using a user provided reverse I-V table to interpolate the current corresponding to the reverse voltage. If V_R is greater than the largest voltage value in the table, then I_R is set to the current corresponding to the largest V_R value. If V_R is less than the smallest voltage value in the table, then $I_R = 0$. The interpolation is performed in a log-linear manner for the reverse current-voltage values.

The age of each device at the user-specified time T_{age} is then calculated as

$$\mathrm{Age}(T_{age}) = \mathrm{Age}(T)\left[\frac{T_{age}}{T}\right] \tag{6.6}$$

The list of ages for every bipolar device in the circuit is stored in an external file called "ageta-bleBJT", which is used for the creation of aged model parameters.

The lifetime to reach a certain $\Delta I_{B,F}$, requested by the user is calculated as

$$\tau = T\left[\frac{\Delta I_{B,F}}{\Delta I_B(T)}\right]^{1/n} \tag{6.7}$$

The user needs to specify the V_{BE} at which ΔI_B is to be calculated.

To calculate the aged model parameters deck, a number of aged parameter files are used as in the CAS [71] module. From the calculated value of Age for each device, a new set of parameters is calculated using either interpolation or regression, as specified by the user. It should be noted that because the SPICE Gummel-Poon parameters are not size independent, the degradation parameters need to be extracted for each device size. If size-independent capabilities are added to the SPICE bipolar equations, then the degradation modeling methodology can be extended with the addition of a few sizing parameters to provide a set of size-independent parameters.

It should be noted that all of the parameters need to be extracted in a consistent manner. For example, D′ should be extracted using the same values for I_S, n_F, the Early Voltage parameters, V_{AF} and V_{AR}, that are used in the SPICE decks. The easiest way to do this is to use Equation (6.1) in the extraction of a and D′ while keeping in mind that V_{BE} is the *internal* base-emitter voltage. Thus, V_{BE} is not equal to $V_t \ln(I_C/I_S)$ if $n_F \neq 1$, $V_{AF} \neq \infty$, or $V_{AR} \neq \infty$; or if I_C is large enough such that I_C begins to roll-off; or if the parasitic voltage drop across R_E and R_B becomes significant.

6.1.2 Parameter Extraction

The extraction of the degradation parameters are based on the following equations :

$$\Delta I_B = I_{SE} e^{V_{BE}/(n_E V_T)} \tag{6.9}$$

$$\Delta I_B = D' I_C^a I_R^b t^n \tag{6.10}$$

$$I_{SE} = D' I_S^a \left[(\#\ \text{of cycles}) \cdot \int_{1\ \text{cycle}} I_R^{b/n}(t)\, dt \right]^n \tag{6.11}$$

Three parameters need to be set by the user. They are

V0: lower V_{BE} value for regression fitting for effective I_{SE} and n_E (default=0.2)
V1: upper V_{BE} value for regression fitting for effective I_{SE} and n_E (default=$n_F V_t \ln(10\beta_F \Delta I_{B,SUM}/I_S)$)
I1: reverse current at which D′ and b changes from the pre- to heavy- avalanche region (default=∞)

Six more degradation parameters need to be extracted from the stressed devices. They are

D0: value of D′ for I_R<I1 (default=0)
A0: a, reciprocal of the non-ideality factor n_E (default=0.5)
B0: b, power dependence of ΔI_B on I_R for I_R<I1 (default =0.5)
C0: n, power dependence of ΔI_B on t (default=0.5)
D1: value of D′ for I_R>I1 (default=0)
B1: b, power dependence of ΔI_B on I_R for I_R>I1 (default =0.5)

The user also needs to make measurements for the reverse current ($I_R V_R$) table.

A. Measuring I_R

To measure I_R, apply a reverse bias to the emitter-base junction and measure I_{eb}, which is the reverse current I_R (see Figure 6.9).

B. Measuring A0

To measure A0,

1) measure I_b of the device at two different forward V_{be} (e.g. V_{be1} and V_{be2}).
2) DC stress the device for a certain time by applying a reverse bias to the emitter-base junction.
3) measure I_b at V_{be1} and V_{be2}.
4) compute ΔI_B from data collected in step 1 and 3.
5) plot $\ln(\Delta I_B)$ vs V_{be} and measure the slope.
The slope is $A0/V_T$. Note that V_T equals to kT/q. See Figure 6.10 and equation (6.9).

C. Measuring C0

To measure C0,

1) measure I_b of the device at a certain V_{be} (e.g. V_{be1}).
2) stress the device for a certain amount time (e.g. t_1 min) by applying a reverse bias (V_R) to the emitter-base junction.
3) measure I_b at V_{be1}.
4) stress the device again (e.g. to a total of t_2 min) using the same V_R.
5) measure I_b at V_{be1}.
6) compute ΔI_B from data collected in step 1, 3, and 5.
7) plot $\ln(\Delta I_B)$ vs ln(t) and measure the slope.
The slope is C0. See Figure 6.11 and equation (6.10).

D. Measuring B0 and B1

To measure B0,

1) two devices are needed.
2) measure I_b of the devices at a certain V_{be} (e.g. V_{be1}).
3) DC stress one device using V_{R1} at the emitter-base junction, and the other using V_{R2}. Both V_{R1} and V_{R2} must be in the low stress region (e.g. V_{R1}=4V, V_{R2}=5V).
4) measure I_b of the devices at V_{be1}.
5) compute ΔI_B from data collected in step 2 and 4, and look up the I_R's corresponding to the V_R's from the $I_R V_R$ table.
6) plot $\ln(\Delta I_B)$ vs $\ln(I_R)$ and measure the slope.
The slope is B0. See Figure 6.12 and equation (6.10).

To measure B1, follow the same procedure used in measuring B0, but set V_R's so that is in the high stress region (e.g. 7V and 8V).

E. Measuring D0 and D1

To measure D0, follow the same procedure used in meauring A0, but set the reverse stress V_R in the low stress region (e.g. 4V). The y-intercept is $\ln(I_{SE})$ (See equation 6.9). Since the device is being DC stressed, I_{SE} of (6.11) becomes :

$$I_{SE} = D' I_S^a I_R^b t^n \qquad (6.12)$$

Since we know, (1) I_{SE}, (2) *a*, *b*, and *n* from A0, B0, and C0, and (3) I_S and A_E from the spice parameter extraction, we can calculate D′ for the low stress region (D0).

To measure D1, follow the same procedure used in meauring D0, but set the reverse stress V_R so that it is in the high stress region (e.g. 8V).

6.2 FORMAT OF PROCESS FILES AND AGED PROCESS FILES

6.2.1 Format of Process Files

A process file is specified by a .BJTPROC card. For example,

.BJTPROC BJT1 revivbjt FILENAME = *bjt_01*

bjt_01 is the process file. A bipolar process file contains a regular SPICE BJT model card in .model format with the addition of a few degradation parameters. For example, *bjt_01* may contain the following :

```
.model bjt1 npn is=4.82e-16 bf=80 nf=1.02
+ vaf=65 ikf=550e-3 br=29.683 nr=1.0 var=1.522
+ isc=25.2e-17 nc=1.57 rb=9.8 irb=2.4e-3 rbm=2.45
+ re=0.48 rc=7.7 fc=.5 ise=0
+ cje=608f vje=0.805 mje=.5 cjc=476f xcjc=.28
+ vjc=0.554 mjc=.249 cjs=356f tr=800p
+ vjs=0.520 mjs=.391 tf=6.645p xtf=150 vtf=1.653 itf=1.452
+ D0=0.00234 a0=0.5098 b0=0.43 c0=0.43
```

The **bold** parameters are the degradation parameters.

6.2.2 Format of Aged Process Files

The aged process files are specified by a **.AGEPROCBJT** card. For example,

.AGEPROCBJT bjt1 FILENAMES = *bjt_0,bjt_1,bjt_2,bjt_3,bjt_4*

bjt_0, bjt_1, bjt_2, bjt_3, bjt_4 are the aged process files.

The format of the first aged process file (*bjt_0*) is different from the rest (*bjt_1, bjt_2, bjt_3, bjt_4*). The first line of *bjt_0* must be a comment line. The 2nd line is the Age and I_{SE}. The following lines are the regular SPICE parameters without the **.model** keyword. For example, *bjt_0* may look like :

Fresh process file , stress time = 0s, AGE=0, Set Ise= 0, Ir=1mA
0 0
+ npn
+ is=4.82e-16 bf=80 nf=1.02
+ vaf=65 ikf=550e-3 br=29.683 nr=1.0 var=1.522
+ ne=1.96
+ isc=25.2e-17 nc=1.57 rb=9.8 irb=2.4e-3 rbm=2.45
+ re=0.48 rc=7.7 fc=.5
+ cje=608f vje=0.805 mje=.5 cjc=476f xcjc=.28
+ vjc=0.554 mjc=.249 cjs=356f tr=800p
+ vjs=0.520 mjs=.391 tf=6.645p xtf=150 vtf=1.653 itf=1.452
+ D0=0.00234 a0=0.5098 b0=0.43 c0=0.43

The *italic* line is the comment line. The **bold** line is the aged data line. The first term is the Age. The second term is I_{SE}. Note that only I_{SE} is changing.
The rest of the process files (*bjt_1, bjt_2, bjt_3, bjt_4*) have only two lines. The first line must be a comment line. The second line must be the aged data line. For example, *bjt_1* may look like :

Aged process file #1, stress time = 1s, AGE=5.277e-28, Ise=1.865e-12, Ir=1mA
5.277e-28 1.865e-12

Since only I_{SE} changes, the rest of the SPICE parameters need not be repeated.

6.3 A DIFFERENTIAL PAIR EXAMPLE

The differential pair of the input comparator in flash analog-to-digital converters (ADCs) is a circuit subject to reverse stress signals. A typical emitter-coupled pair circuit is shown in Figure 6.1. From this circuit, the fresh BERT/SPICE input deck was written, and is shown in Figure 6.2. The resulting BERT output is in Figure 6.3. Figure 6.4 shows the aged input deck. Note that all aging and degradation commands have been removed by prebert in the aged input deck. Finally, Figure 6.5 compares the SPICE output of the aged and fresh differential pair when the input is a square pulse with a magnitude of 0.5V.

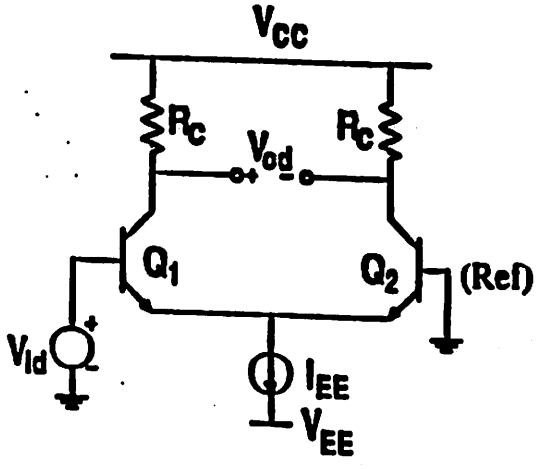


Fig. 6.1

```
Diff Pair for a flash ADC

* circuit description
vcc 10 0 6v
vb2 2 0 0v
iee 3 0 250e-6
vin 1 0 dc 0 pulse(0 0.5 10n 1n 1n 10n 22n)
r1 10 4 4k
r2 10 5 4k
q1 4 1 3 0 bjt1
q2 5 2 3 0 bjt1

* spice control card
.tran 0.5n 44n

.options vntol=1e-11 abstol=1e-19 chgtol=1e-20 cptime=1e4 pivtol=1e-29
+ itl1=10000 itl2=500 itl4=2000

* bert BJT control card
.bjtproc bjt1 revivbjt filename = bjt_01
.deltaib 100ua vbe=0.7
.agebjt 100y
.agedib 10y vbe=0.7
.degsortbjt
.ageprocbjt bjt1 filenames=bjt_0,bjt_1,bjt_2,bjt_3,bjt_4
.agemethodbjt interp linlog
.width out=80
.end
```

Fig. 6.2

```
Circuit: Diff Pair for a flash ADC

Circuit: Diff Pair for a flash ADC
Date: Thu Jun 13 14:12:43 PDT 1991


Total run time: 1.371 seconds.

Current data size = 326736,
Data limits: hard = 33554432, soft = 33554432.

Time since last call: 0.000 seconds.

-------------------------------------------------------------------------
|  DEVICE LIFETIME AT DELTA IB = 0.0001 AMPS:
-------------------------------------------------------------------------
|
|   >>>>>> TAU(q1) = 282400 YEARS (8.907e+12 SEC.) <<<<<<
|
-------------------------------------------------------------------------
|  DEGRADATION OF q1 AT 3.15576e+08 SEC. (10 YEARS):
-------------------------------------------------------------------------
|
|   >>>>>> DELTA IB = 1.21962e-06 <<<<<<
|
-------------------------------------------------------------------------

-------------------------------------------------------------------------
|  DEVICE LIFETIME AT DELTA IB = 0.0001 AMPS:
-------------------------------------------------------------------------
|
|   >>>>>> TAU(q2) = 111500 YEARS (3.517e+12 SEC.) <<<<<<
|
-------------------------------------------------------------------------
|  DEGRADATION OF q2 AT 3.15576e+08 SEC. (10 YEARS):
-------------------------------------------------------------------------
|
|   >>>>>> DELTA IB = 1.81881e-06 <<<<<<
|
-------------------------------------------------------------------------


-------------------------------------------------------------------------
| BIPOLAR TRANSISTORS LISTED IN ORDER OF DECREASING HOT CARRIER DEGRADATION:
-------------------------------------------------------------------------
|
|   Transistor Name          Lifetime     Degradation        Age
|                            (seconds)      delta Ib
|
|   Q2                     3.516500e+12   1.818809e-06   5.166034e-27
|   Q1                     8.907216e+12   1.219623e-06   2.039510e-27
-------------------------------------------------------------------------
delta Ib   :  change in Ib after 3.16e+08 sec
Lifetime   :  time needed for Ib to change by 1.00e-04 A
```

Fig. 6.3

```
Diff Pair for a flash ADC

* circuit description
vcc 10 0 6v
vb2 2 0 0v
iee 3 0 250e-6
vin 1 0 dc 0 pulse(0 0.5 10n 1n 1n 10n 22n)
r1 10 4 4k
r2 10 5 4k
q1 4 1 3 0 B0
q2 5 2 3 0 B1

* spice control card
.tran 0.5n 44n

.options vntol=1e-11 abstol=1e-19 chgtol=1e-20 cptime=1e4 pivtol=1e-29
+ itl1=10000 itl2=500 itl4=2000

* bert BJT control card
.width out=80


.bjtproc B0 revivbjt filename = AGEBJT0
.bjtproc B1 revivbjt filename = AGEBJT1
.end
```

Fig. 6.4

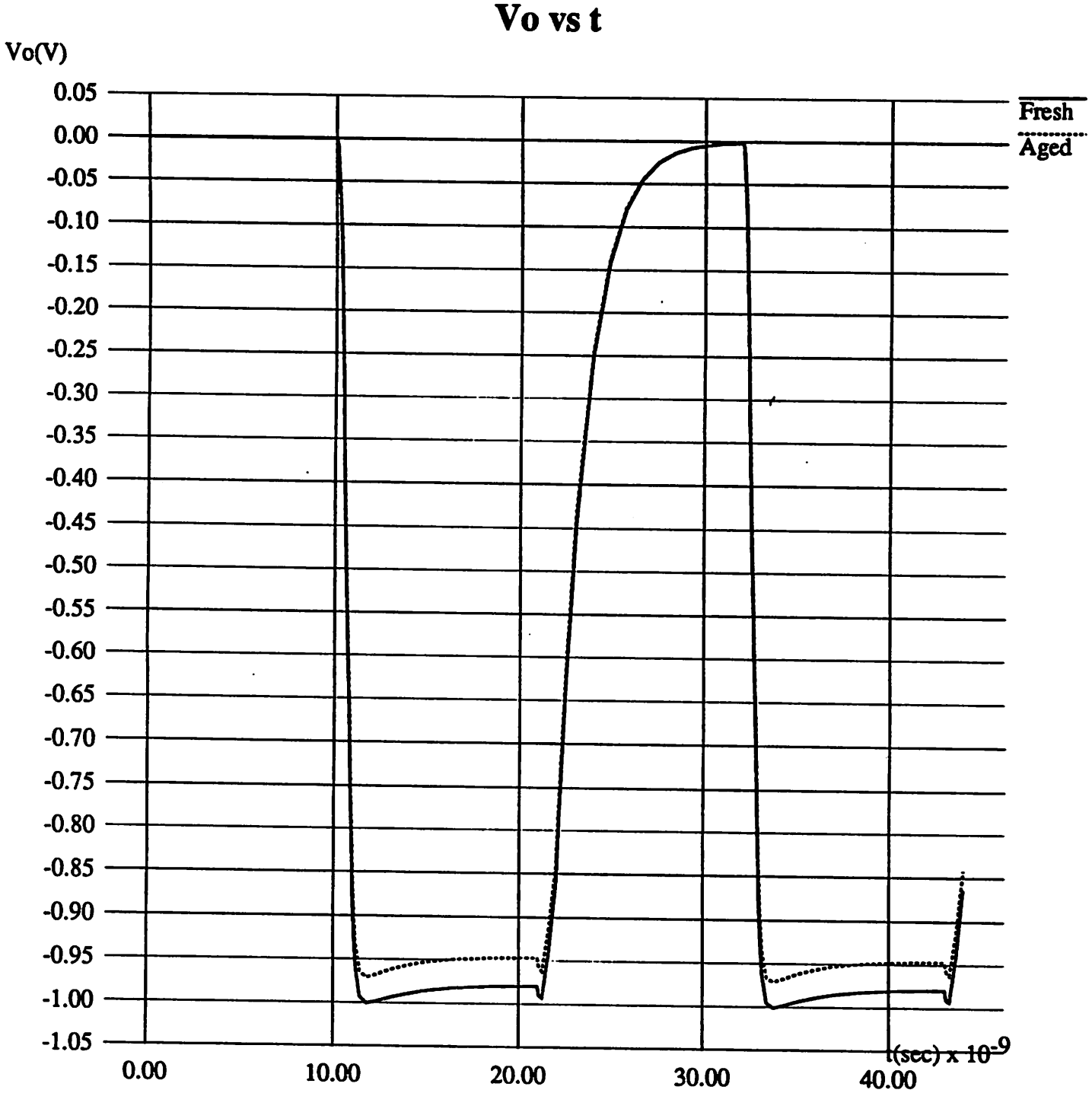

Vo vs t
Vo(V)
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
-0.30
-0.35
-0.40
-0.45
-0.50
-0.55
-0.60
-0.65
-0.70
-0.75
-0.80
-0.85
-0.90
-0.95
-1.00
-1.05
0.00
10.00
20.00
30.00
40.00
t(sec) x 10^-9
Fresh
Aged

Fig. 6.5

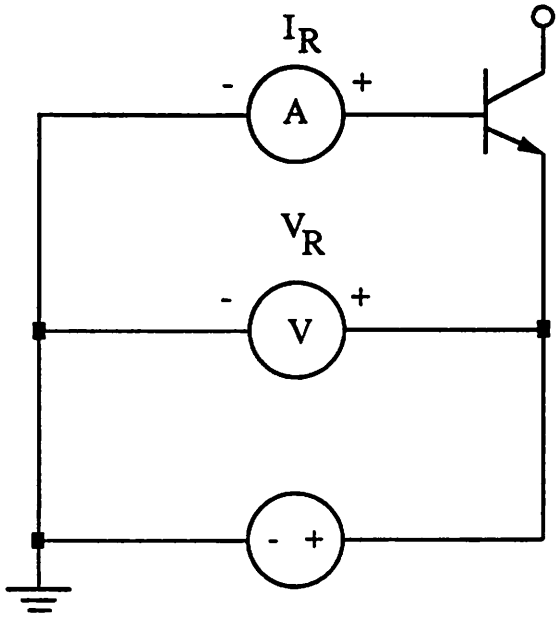


Fig 6.9

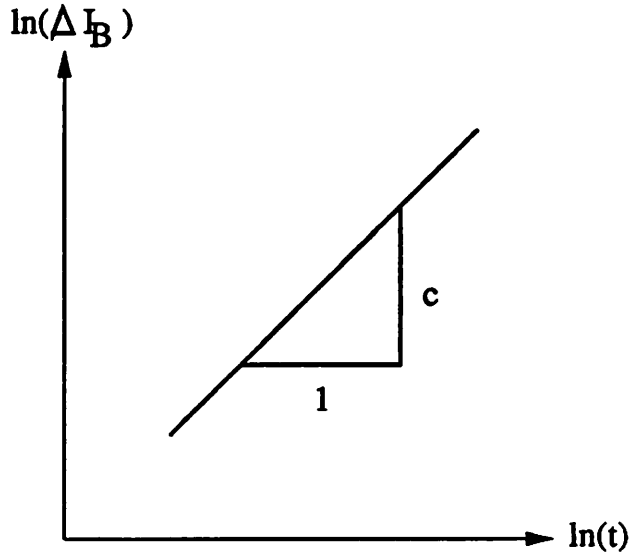


Fig 6.11

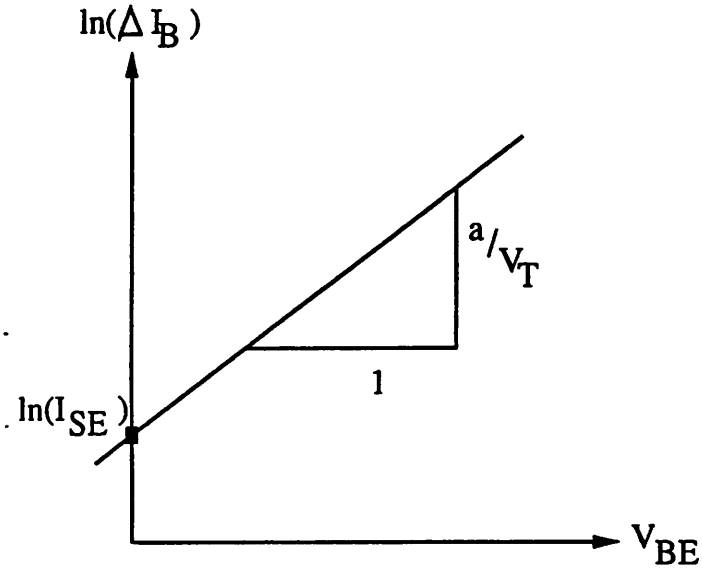


Fig 6.10

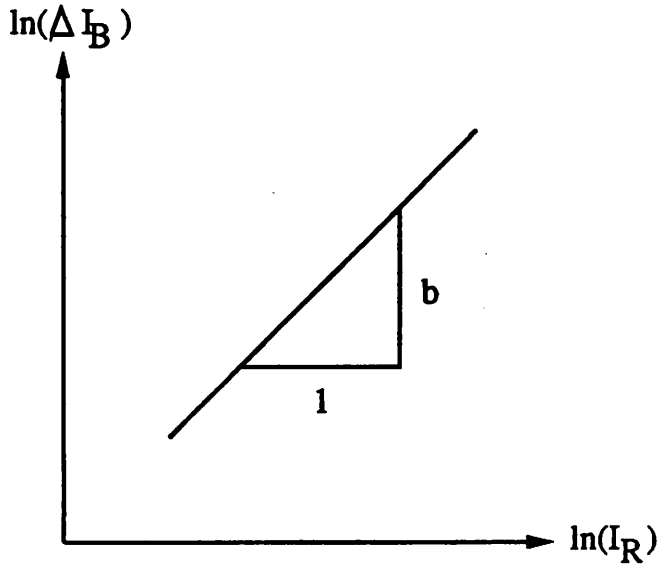


Fig 6.12

7. Using Irsim with Bert

Bert now works with Irsim, a timing simulator. The use of Irsim rather than SPICE reduces simulation time (especially important for large circuits), although accurate construction of circuit waveforms is sacrificed. Our current efforts have been directed towards making IRSIM compatible with both the CORS and EM modules. However, with the release of Bert 2.0, only the CORS module can be run using Irsim. The process of running Bert with Irsim is very similar to the process of running it with SPICE. The rest of this chapter will detail how to run BERT with Irsim.

7.1 HOW TO RUN PREBERT USING IRSIM

7.1.1 Running Prebert

Because of the differences between Irsim and SPICE, we needed to modify the way prebert is invoked. BERT specific commands are stored in a separate file, the *bertfile*, rather than being stored in the cmdfile from which they would have to be commented out by prebert before passing to Irsim. The shell-script (Section 2.4) may be used to run BERT with Irsim, or the user may execute prebert, Irsim and postbert manually in a sequential fashion.

To run prebert, type :

> prebert *[-b bertfile] [-c cmdfile] [-n] <infile>*

Option c is mandatory, however option b is not.

bertfile	The external BERT command file.
cmdfile	The Irsim cmdfile (will be modified by prebert).
n	The SPICE type, for spice only, will be ignored in an Irsim run.
infile	the input circuit description.

7.1.2 What Prebert Returns When Using Irsim

Prebert will return five (5) files for Irsim.

bert.sim the modified sim format circuit description file *(infile)*. The reason we need to modify *infile* is to collect all the equivalent node names (declared in the '=' statements) into one single node name. If we don't do that, Irsim will randomly use one of the equivalent node names, which will cause trouble for postbert.

bert.cmd the modified Irsim command file, which has trace (t) statements added in the beginning to trace all transistor nodes that are not in the constant-node list. The *model linear* statement is also added to the beginning of the file. The *inputs* statement is inserted right before the first running command (i.e. s, c, R, or p) to print out the value of all unchanged nodes.

infile.ref the cross reference file for the transistors in the sim file and their corresponding names used in BERT. Note that the suffix of *infile* is stripped away (i.e. if *infile* is *adder.sim*, *adder.ref* will be returned). Suppose we have the followings in *infile* :

```
n 1 2 3 4 5 6 7
p 2 3 4 5 6 7 8
p a b c d e f g
```

infile.ref will have :

n0 1 2 3 4 5 6 7
p1 2 3 4 5 6 7 8
p2 a b c d e f g

Note the transistor names with the numbers following the 'n' and 'p', they will be used in BERT.

rawsub the rawsub file for BERT. If it is an Irsim run, it will contain the following :

irsim = 1
emmodel = 0
dbmodel = 1
Voh = 1.200000
Vol = 0.100000
Vdd = 4.000000

Vdd, Voh, and Vol are specified in infile.bert.

rawtddb the connection between prebert and postbert for CORS. Its format is identical to its counterpart in a SPICE run.

7.1.3 Running Irsim

To understand how to run BERT using Irsim, one needs to know how Irsim is usually run by itself. To run Irsim in batch mode, the user types in the following command line:

> irsim *parafile simfile -cmdfile*

parafile the parameter file which is technology dependent.
simfile the sim format circuit description file
cmdfile the Irsim command file, which describes the circuit inputs and clocking.

For more information on how Irsim is used, see Appendix I, the Irsim user's guide.
After passing the input file through prebert, the user should type in the following to execute Irsim:

> irsim *parafile* bert.sim -bert.cmd > *irsim.out*

parafile the parameter file which is technology dependent.
bert.sim generated by prebert.
bert.cmd generated by prebert.
irsim.out output file from Irsim.

7.1.4 Running Postbert

After running through Irsim, type the followings to run postbert:

> postbert < *irsim.out* > *post.out*

irsim.out output file from Irsim.
post.out output of BERT using Irsim.

7.2 FORMAT OF THE BERTFILE

The *bertfile* contains BERT commands. Only CORS and some general commands are allowed in *bertfile* at the present time.

7.2.1 CORS Commands Availiable for Irsim

More information on each of the commands may be found in Appendix A.

emodel	Note that the SPICE version uses **.emodel**, but the Irsim version uses **emodel**.
ttf	The options are the same as the SPICE version. Note that the SPICE version uses **.ttf**, but the Irsim version uses **ttf**.
xeff	The options are the same as the SPICE version. Note that the SPICE version uses **.xeff**, but the Irsim version uses **xeff**.
.altmodel	The options are the same as the SPICE version. Note that the SPICE version uses **.altmodel**, and the Irsim version ALSO uses **.altmodel**.
.model	The options are the same as the SPICE version. Note that the SPICE version uses **.model**, and the Irsim version ALSO uses **.model**.
eachprob	The options are the same as the SPICE version. Note that the SPICE version uses **.eachprob**, but the Irsim version uses **eachprob**.
lsi	The options are the same as the SPICE version. Note that the SPICE version uses **.lsi**, but the Irsim version uses **lsi**.
burnin	The options are the same as the SPICE version. Note that the SPICE version uses **.burnin**, but the Irsim version uses **burnin**.
.options	The options are the same as the SPICE version. Note that the SPICE version uses **.options**, and the Irsim version ALSO uses **.options**. The point of putting a spice .options card in an Irsim run is to provide TNOM for postbert calculation.

7.2.2 Additional Commands

The following commands were added to meet the specific requirements imposed by the Irsim simulator. Just like the previous commands, these should be included in the *bertfile*. Each of these commands specify the values for a parameter used by BERT. The user should assign these parameters a value appropriate for the circuit's operation.

(1) vdd
Usage:

Vdd = <*voltage*>

Specifies the power supply voltage. If omitted, **vdd** will be set to 5V. For example :

Vdd=4.5

specifies the supply voltage to be 4.5V.

(2) voh
Usage:

Voh = <*voltage*>

Specifies the high output voltage. If omited, **voh** will be set to **vdd**. It is an error to set **voh** higher than **vdd**. For example :

Voh=4.0
Voh=5

If the Vdd is set to 4.5V, then the first line will set the high output voltage to be 4.0V, and the second line would cause an error to be flagged.

(3) vol
Usage:

Vol = *<voltage>*

Specifies the low output voltage. If omited, **vol** will be set to zero (0). For example :

Vol=0.3V

specifies the low output voltage to be 0.3V.

(4) consthi
Usage:

consthi = *<nodename1> <nodename2> <nodename3>* ...

Specifies the nodes which will always have a high output value. One such node is the node connected to the positive terminal of the power supply. For example :

consthi = vdd vg1 vg2 vg3
consthi = vg4 vg5

If both **consthi** statements are in the .bert file, vdd, vg1, vg2, vg3, vg4, and vg5 will be set to high for the BERT simulation.

Neither **consthi** nor **constlo** (see below) statements are necessary for successful execution of CORS; however, their inclusion speeds the execution time.

(5) constlo
Usage:

constlo = *<nodename1> <nodename2> <nodename3>* ...

Specifies the nodes which will always have a low output value. One of such nodes is the node connected to the negative terminal of the power supply. For example :

constlo = vss vs1 vs2 vs3
consthi = vs4 vs5

If both **constlo** statements are in the .bert file, vss, vs1, vs2, vs3, vs4, and vs5 will be set to low for BERT simulation.

(6) starttime
Usage:

starttime = *<time>*

Specifies the starting simulation time for bert calculations. If it is omitted, starttime will be set to zero. For example :

starttime = 10n

sets the starting simulation time to be 10 ns.

Because many nodes will be set to X (unknown state) until a (simulated) time when input voltages have propagated all the way through the circuit, we suggest setting starttime to be greater than the time for the initial signals to propagate through the circuit.

(7) endtime
Usage:

endtime = *<time>*

Specifies the ending simulation time for bert. For example :

endtime = 100n

sets the ending simulation time to be 100 ns.

(8) comments
Usage:

| comment line 1
* comment line 2
; comment line 3

If a line starts with the symbols |, *, or ; in the first column, it will be considered as a comment line and ignored.

8. References

Circuit Aging Simulator

[1] T.Y. Chan, P.K. Ko, and C. Hu, "A simple method to characterize substrate current in MOSFET's," *IEEE Electron Device Letters*, Vol. EDL-5, No. 12, pp. 505-507, December 1984.

[2] Charles Sodini, Ping-Keung Ko, and John L. Moll, "The effect of high fields on MOS device and circuit performance," *IEEE Trans. Electron Devices*, Vol. ED-31, No. 10, pp. 1386-1393, October 1984.

[3] Simon Tam, Ping-Keung Ko, and Chenming Hu, "Lucky-electron model of channel hot-electron injection in MOSFET's," *IEEE Trans. Electron Devices*, Vol. ED-31, No. 9, pp. 1116-1125, September 1984.

[4] C. Hu, S. Tam, F.-C. Hsu, P.K. Ko, T.Y. Chan, and K.W. Terrill, "Hot-electron-induced MOSFET degradation - model, monitor, and improvement," *IEEE Trans. Electron Devices, Vol. ED-32, pp. 375-385, February 1985.*

[5] Peter M. Lee, "BSIM - Substrate current modeling," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M86/49, June 1986.

[6] W. Weber, C. Werner, and A.V. Schwerin, "Lifetimes and substrate currents in static and dynamic hot-carrier degradation," in *IEDM Tech. Digest*, pp. 390-393, December 1986.

[7] Shian Aur, Dale E. Hocevar, and Ping Yang, "Circuit hot electron effect simulation," *IEDM Tech. Digest*, pp. 498-501, December 1987.

[8] M.-C. Jeng, P.M. Lee, M.M. Kuo, P.K. Ko, and C. Hu, "Theory, algorithms, and user's guide for BSIM and SCALP," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M87/35, May 1987.

[9] Mary Mei-Lin Kuo, "Implementation of the BSIM substrate current and degradation models in SCALP," Master's Thesis, May 1987.

[10] M.M. Kuo, K. Seki, P.M. Lee, J.Y. Choi, P.K. Ko, and C. Hu, "Quasi-static simulation of hot-electron-induced MOSFET degradation under AC (pulse) stress," *IEDM Tech. Digest*, pp. 47-50, December 1987.

[11] Peter M. Lee, Mary M. Kuo, Ping K. Ko, and Chenming Hu, "Circuit Aging Simulator (CAS)," *IEDM Tech. Digest*, pp. 134-137, December 1988.

[12] S. Aur, "Kinetics of hot carrier effects for circuit simulation," in *Proc. IEEE Rel. Phys. Symp.*, pp. 88-91, April 1989.

[13] R. Bellens, P. Heremans, G. Groeseneken, and H.E. Maes, "On the channel-length dependence of the hot-carrier degradation of n-channel MOSFETs," *IEEE Electron Device Letters*, Vol. 10, No. 12, pp. 553-555, December 1989.

[14] Tong-Chern Ong, Koichi Seki, Ping K. Ko, and Chenming Hu, "P-MOSFET gate current and device degradation", *Proc. IEEE Rel. Phys. Symp.*, pp. 178-182, March 1989.

[15] Elyse Rosenbaum, Peter M. Lee, Reza Moazzami, P.K. Ko, and C. Hu, "Circuit reliability simulator - oxide breakdown module," *IEDM Tech. Digest*, pp. 331-334, December 1989.

[16] B.K. Liew, Peng Fang, N.W. Cheung, and C. Hu, "BERT - Circuit Electromigration Reliability Simulator", University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M90/3, January 1990.

[17] B.K. Liew, N.W. Cheung, and C. Hu, "Reliability simulator for interconnect and intermetallic contact electromigration," to be presented at the *Proc. IEEE Rel. Phys. Symp.*, March 1990.

[18] Elyse Rosenbaum, Peter M. Lee, Reza Moazzami, Ping K. Ko, and Chenming Hu, "BERT - Circuit Oxide Reliability Simulator (CORS)", University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M90/4, January 1990.

[19] L. W. Nagel,"SPICE2: A computer program to simulate semiconductor ciruits," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M75/520, May 1975.

[20] Andrei Vladimirescu and Sally Liu, "The simulation of MOS integrated circuits using SPICE2," University of california, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M80/7, February 1980.

[21] Richard B. Fair and Robert C. Sun, "Threshold-voltage instability in MOSFET's due to channel hot-hole emission," IEEE Transactions on Electron Devices, Vol. ED-28, No.1, pp. 83-93, January 1981.

[22] Heihachi Matsumoto, Kokichi Sawada, Sotoju Asai, Makoto Hirayama, and Koichi Nagasawa, "Effect of long-term stress on IGFET degradations due to hot electron trapping," IEEE *Transactions on Electron Devices*, Vol. ED-28, No.8, pp. 923-928, August 1981.

[23] E. Takeda and N. Suzuki, "An empirical model for device degradation due to hot-carrier injection," IEEE Electron Device Letters, Vol. EDL-4, No.4, pp. 111-113, April 1983.

[24] E. Takeda, Y. Nakagome, H. Kume, and S. Asai, "New hot-carrier injection and device degradation in submicron MOSFETs," *IEEE Proceedings*, Vol. 130, Part I, No.3, pp. 144-149, June 1983.

[25] E. Takeda, A.Shimizu, and T. Hagiwara, "Role of hot-hole injection in hot-carrier effects and the small degraded channel region in MOSFET's, " *IEEE Electron Device Letters, Vol. EDL-4, No. 9, pp.329-331, September 1983.*

[26] F.-C. Hsu and S. Tam, "Relationship between MOSFET degradation and hot-electron-induced interface-state generation," *IEEE Electron Device Letters* Vol. EDL-5, No.2 pp. 50-52, February 1984.

[27] Charles Sodini, Ping-Keung Ko, and John L. Moll, "The effect of high fields on MOS device and circuit performance," *IEEE Electron Device Letters*, Vol. ED-31, No.10, pp. 1386-1393, October 1984.

[28] Karl R. Hofmann, Christoph Werner, Werner Weber, and Gerhard Dorda, "Hot-electron and hole-emission effects in short n-channel MOSFETs," *IEEE Transactions on Electron Devices*, Vol. ED-32, No.3, pp. 691-699, March 1985.

[29] Fu-chieh Hsu and Kuang Yi Chiu, "Hot-electron substrate-current generation during switching transients," *IEEE Transactions on Electron Devices*, Vol. ED-32, No.2, pp. 394-399, February 1985.

[30] Bing J. Sheu, D.L. Sharfetter, and P.K. Ko, "SPICE2 implementation of BSIM," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M85/42, May 1985.

[31] T. Tsuchiya and J. Frey, "Relationship between hot-electron/holes and degradation of p- and n-channel MOSFET's," *IEEE Electron Device Letters*, Vol. EDL-6, No.1, pp. 8-11, January 1985.

[32] Anjan Bhattacharyya and Sunhil N. Shabde, "Degradation of short-channel MOSFET's under constant current stress across gate and drain," *IEEE Transactions on Electron Devices*, Vol. ED-33, No.9, pp.1329-1333.

[33] Y. Hiruta, K. Maeguchi, and K. Kanzaki, "Impact of hot electron trapping on half micron PMOSFETs with p+ poly Si gate," IEDM Tech. Digest, pp. 718-721, December 1986.

[34] T. Horiuchi, H. Mikoshiba, K. Nakamura, and K. Hamano, "A simple method to evaluate device lifetime due to hot-carrier effect under dynamic stress," *IEEE Electron Device Letters*, Vol. EDL-7, No.6, pp. 337-339, June 1986.

[35] Joseph J. Tzou, C.C. Yao, R. Cheung, and Hugo W.K. Chan, "Hot-carrier-induced degradation in p-channel LDD MOSFET's" *IEEE Electron Device Letters*, Vol. EDL-7,

No.1, pp. 5-7, January 1986.

[36] J.Y. Choi, P.K. Ko, and C. Hu, "Effect of oxide field on hot-carrier-induced degradation of metal-oxide-semiconductor field-effect transistors," *Applied Physics Letters*, Vol. 50, pp. 1188-1190, April 1987.

[37] J.Y. Choi, P.K. Ko, and C. Hu, "Hot-carrier-induced MOSFET degradation: AC versus DC stressing," *Proceedings of the Symposium on VLSI Technology*, pp. 45-46, May 1987.

[38] J.Y. Choi, P.K. Ko, and C. Hu, "Hot-carrier-induced MOSFET degradation under AC stress," *IEEE Electron Device Letters*, Vol. EDL-8, No.8, pp. 333-335, August 1987.

[39] B.J. Sheu, D.L. Sharfetter, P.K. Ko, and M.-C. Jeng, "BSIM: Berkeley short-channel igfet model for MOS transistors," *IEEE Journal of Solid-State Circuits, vol SC-22, No. 4, August 1987.*

[40] Toshiaki Tsuchiya, Toshio Kobayashi, and Shigeru Nakajima, "Hot-carrier-injected oxide region and hot-electron trapping as the main cause in Si nMOSFET degradation," *IEEE Transactions on Electron Devices*, Vol. ED-34, No. 2, pp. 386-391, February 1987.

[41] Michael P. Brassington, Mark W. Poulter, and Monir El-Diwany, "Suppression of hot-carrier effects in submicrometer surface-channel PMOSFET's," *IEEE Transactions on Electron Devices*, Vol. 35, No. 7, pp.1149-1151, July 1988.

[42] T.Y. Chan, C.L. Chiang, and H. Gaw, "New insight into hot-electron-induced degradation of n-MOSFET's," *IEDM Technical Digest*, pp. 196-199, December 1988.

[43] T.-C. Ong, Kouichi Seki, P.K. Ko, and Chenming Hu, "Hot-carrier-induced degradation in p-MOSFET's under AC stress," *IEEE Electron Device Letters*, Vol. 9, No.5, pp. 211-213, May 1988.

[44] Werner Weber, "Dynamic stress experiments for understanding hot-carrier degradation phenomena," *IEEE Transactions on Electron Devices*, Vol. 35, No. 9, pp. 1476-1486, September 1988.

[45] M.-C. Jeng, "Design and modeling of deep-submicrometer MOSFET's," Ph.D. dissertation, University of California, Berkeley, November 1989.

[46] Thomas Quarles, "SPICE3 Version 3C1 user's guide," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M89/46, April 1989.

[47] J.S. Duster, M.-C. Jeng, P.K. Ko, and Chenming Hu, "User's guide for the BSIM2 parameter extraction program and SPICE3 with BSIM2 implementation," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M90, May 1990.

[48] Tom Garfinkel, "PMOSFET Hot-Carrier...," Master's Thesis, University of California, Berkeley, May 1990.

[49] Peter M. Lee, Ping K. Ko, and Chenming Hu, "Relating CMOS inverter lifetime to DC hot-carrier lifetime of NMOSFET's," *IEEE Electron Device Letters*, Vol. 11, No. 1, pp. 39-41, January 1990.

[50] Peter M. Lee, "Modeling and simulation of hot-carrier effects in MOS devices and circuits," Ph.D. Thesis, University of California, Berkeley, Electronics Research Laborartory Memorandum UCB/ERL M90/3, April 1990.

[51] Kaizad Mistry and Brian Doyle, "The role of electron trap creation in enhanced hot-carrier degradation during AC stress," *IEEE Electron Device Letters*, Vol. 11, No. 6, pp. 267-269, June 1990.

[52] Tong-Chern Ong, Ping K. Ko, and Chenming Hu, "Hot-carrier current modeling and device degradation in surface-channel p-MOSFET's," *IEEE Transactions on Electron Devices*, Vol. 37, No. 7, pp. 1658-1666, July 1990.

[53] M.M. Kuo, K. Seki, P.M. Lee, J.Y. Choi, P.K. Ko, and C. Hu, "Simulation of MOSFET lifetime under AC hot-electron stress." *IEEE Transactions on Electron Devices*, Vol. ED-35, No.7, pp. 1004-1011, July 1988.

Circuit Oxide Simulator

[54] P.M. Lee et al., "BERT - Circuit Aging Simulator (CAS)," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M90/2, January 1990.

[55] B. K. Liew et al., "BERT - Circuit Electromigration Reliability Simulator," University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M90/3, January 1990.

[56] I. C. Chen and C. Hu, "Accelerated testing of Time-Dependent Breakdown of SiO_2," *IEEE Electron Device Letters*, Vol. 8, no.4, p.140, April 1987.

[57] J. Lee et al., *IEEE Transactions on Electron Devices*, Vol. 35, p. 2268, Dec. 1988.

[58] E. Rosenbaum et al., *IEDM Technical Digest*, p. 331, Dec. 1989.

[59] R. Moazzami et al., *IEEE Transactions on Electron Devices*, Vol. 36, p. 2462, Nov. 1989.

[60] R. Moazzami et al., *VLSI Symposium Technical Digest*, May 1989.

[61] Arnold Berman, "Time-Zero Dielectric Reliability Test by a Ramp Method," *Proceedings of the IRPS* p. 204, 1981.

Electromigration Circuit Simulator

[62] B. K. Liew, N. W. Cheung, and C. Hu, "Electromigration interconnect lifetime under AC and pulse DC stress," *Proc. 29th Int. Reliab. Phys. Symp. IEEE*, p. 215, 1989.

[63] B. K. Liew, N. W. Cheung, and C. Hu, "Projecting interconnect electromigration lifetime for arbitrary current waveforms," to be published in *IEEE Trans. on Electron Devices*,

[64] B. K. Liew, P. Fang, N. W. Cheung, and C. Hu, "Circuit reliability simulator for interconnect and contact electromigration," to be published in *Proc. 30th Int. Reliab. Phys. Symp. IEEE*,

[65] P. M. Lee, M. M. Kuo, P. K. Ko, and C. Hu, "BERT - Circuit Aging Simulator (CAS)," *UCB/Electronics Research Lab. M90/2*, Electronics Research Laboratory of University of California, Berkeley, Jan 1990.

[66] E. Rosenbaum, P. M. Lee, R. Moazzami, P. K. Ko, and C. Hu, "BERT - Circuit Oxide Reliability Simulator (CORS)," *UCB/Electronics Research Lab. M90/4*, Electronics Research Laboratory of University of California, Berkeley, Jan 1990.

[67] B. N. Agarwala, M. J. Attardo, and A. P. Ingraham, "Dependence of electromigration-induced failure time on length and width of aluminum thin-film conductor," *J. Appl. Phys.*, vol. 41, no. 10, p. 3954, 1979.

[68] E. Kinsbron, "A model for the width dependence of electromigration lifetimes in aluminum thin-film stripes," *J. Appl. Phys.*, vol. 36. no. 12. p. 968, 1980.

[69] A. V. Ferris-Prabhu, "Reliability Modeling," *Int. Reliab. Phys. Symp. Tutorial Notes*, p. 3-1, Apr 1984.

[70] D. T. Fitzpatrick, "Mextra: A Manhattan Circuit Extractor," *UCB/Electronics Research Lab. M82/42*, Electronics Research Laboratory of University of California, Berkeley, May 1982.

Bipolar Circuit Aging Simulator

[71] P.M. Lee, M.M. Kuo, P.K. Ko, and C. Hu, "BERT-circuit aging simulator (CAS)," University of California, Berkeley, Electronics Research Laboratory Memorandum UCM/ERL M90/2, January 1990.

[72] P.M. Lee, M.M. Kuo, P.K. Ko, and C. Hu, "Circuit aging simulator (CAS)," in *IEDM Tech. Dig.*, 1988, p. 134.

[73] T. Quarles, "SPICE3 version 3C1 user's guide," University of California, Berkeley, Electronics Research Laboratory Memorandum UCM/ERL M89/46, April 1989.

[74] I.E. Getreu, *Modeling the Bipolar Transistor*. Amsterdam: Elsevier, 1976.

[75] B.A. Unger, "Electrostatic discharge failures of semiconductor devices," *Proc. Int. Rel. Phys. Symp.*, 1981, p. 193.

[76] James David Burnett, "Hot-Carrier Reliability of Bipolar Transistors and Circuits" (Ph.D. Thesis), University of California, Berkeley, July 90, Chapter 3.

[77] James David Burnett, Chenming Hu, "Hot-Carrier Reliability of Bipolar Transistors," in *IRPS* 1990, p. 164.

General Bert Sections

[78] Chemning Hu, "IC Reliability Simulation," *Proceedings of the CICC*, 1991.

Appendix A
Bert Command Summary

I. GENERAL BERT AND CAS COMMAND SUMMARY

The following new commands are for use specifically with CAS for substrate current, device degradation analysis, and circuit aging. CAS includes a revised SCALE command set that eliminates some of the redundancy and adds more flexibility to the ones listed in [8]. Note that many commands are similar to SPICE commands.

1.1 .AGE *time*

Examples:

.AGE 10years
.AGE 5minutes

This command specifies the future time at which to calculate the aged model parameter files for circuit simulation. The units for time can be in "y", "h", "m", or "s", corresponding to years, hours, minutes, and seconds, with no space between the number and the unit. Letters following the above four units of time are ignored. Thus **10years** and **10y** are interpreted identically.

1.2 .AGEDID *time*
.AGEDGM *time*
.AGEDVT *time*

Examples:

.AGEDID 10years
.AGEDVT 1year

These commands specify the future time at which drain current degradation $\Delta I_{ds}/I_{ds0}$ (**AGEDID**), transconductance degradation $\Delta g_m/g_{m0}$ (**AGEGM**), or threshold voltage shift ΔV_{th} (**AGEDVT**) is desired. This is the converse of the lifetime commands **DELTAID, DELTAGM, DELTAVT.** The format for *time* is identical to that of the **.AGE** command. Note that appropriate H, m, and n values must be given, since parameter values will differ depending upon the actual degradation specified ($\Delta I_{ds}/I_{ds0}$, $\Delta g_m/g_{m0}$, or ΔV_{th}). Setting H_0 and H_{gd} to 0 will disable the calculation for that particular model.

Note that the three commands differ only in nomenclature. Any other quantity other than the three mentioned above can be used as the degradation monitor, as long as the corresponding values for H, m, and n are used. For instance, the H, m, n parameters can be extracted while monitoring $\Delta R_{out}/R_{out0}$, in which case using any of the three commands will give you the $\Delta R_{out}/R_{out0}$ suffered up to the specified time.

1.3 .AGEMETHOD *method <domain>*

Examples:

.AGEMETHOD INTERP LINLOG
.AGEMETHOD LINLIN

This command specifies the method of numerical analysis used to calculate the aged parameter set from the pre-stressed model parameters. The first argument specifies the method of the regression analysis (LINLIN, LINLOG, or LOGLOG). The keyword INTERP should be placed in this position if interpolation rather than regression is desired. The keyword INTERP can be

followed by the method in which the interpolation will be performed (LINLIN, LINLOG, or LOGLOG). The default is linear-log interpolation if no **AGEMETHOD** command is present.

1.4 .AGEPROC *mname* FILENAMES=*fname1, fname2, fname3 <,fname4,...>*

Example:

.AGEPROC PC1 FILENAMES=DE0, DE1, DE2, DE3

This command specifies the names of the pre-stressed model parameter files *fname* associated with the model *mname*. The filenames should be ordered by increasing ages, with the fresh file first. At least one fresh and one aged model parameter file must be present for linear-linear analysis, while two aged model parameter files must be present for linear-log or log-log analysis. Note that unlike the **.PROCESS** statement, "**FILENAMES**" appears in plural form. The **.PROCESS** command is still needed. The format of the aged model parameter files is identical to the fresh model parameter files used in the **.PROCESS** command.

1.5 .DEGPRINT *trnname1 <trnname2 ...>*

Example:

.DEGPRINT M1 M4 M6

This command restricts degradation information printout (such as that shown in Fig. 3.4.2) to occur only for the specified transistors. Without this command, degradation information for all the transistors in the circuit will be printed out.

1.6 .DEGSORT

Example:

.DEGSORT

This command requests a printout in tabular form all the transistors in the circuit listed from the most degraded to least degraded. The corresponding device lifetime is given if one of the **.DELTA** commands (e.g. **.DELTAID**) is present, the amount of device degradation is given if one of the **.AGE** commands (e.g. **.AGEDID**) is present, and the age of each transistor is given if the **.AGE** command is present.

1.7 .DELTAID *value*
.DELTAGM *value*
.DELTAVT *value*

Examples:

.DELTAID 0.05
.DELTAGM 0.1
.DELTAVT 10mV

These commands specify either drain current degradation $\Delta I_{ds}/I_{ds0}$, transconductance degradation $\Delta g_m/g_{m0}$, or the threshold voltage shift ΔV_{th}, at which the device lifetime is defined. Like the **AGEDID, AGEDGM,** and **AGEDVT** commands, appropriate values of H, m, and n must be specified depending on which of the three criteria is used to determine device lifetime. Again, setting $H_0 = 0$ and $H_{gd} = 0$ will disable the calculation for that particular model.

Note that the three commands differ only in nomenclature, as in the **AGEDID**-type of commands. Any other quantity other than the three mentioned above can be used as the degradation monitor, as long as the corresponding values for H, m, and n are used.

1.8 .EXPCKT <*subckt call*> <*all*>

Example:

.expckt x1 x2 x3 x4 x5
.expckt all

If the *all* option is specfied, all subcircuit calls found in the deck will be expanded. Nested subckt calls up to 20 layers in depth are supported. If the nesting of the subckts exceeds 20 layers, a recursive subckt definition is assumed. A local device translation table "rawdevtab" will be generated for each expanded subckt call.

Restrictions

- No nested subckt *definitions* are allowed.
- No node number $\geq$ 200000.
- No symbol '%' in any device name
- Each subckt def must have less than 2000 local nodes.
- Each subckt def must have less than 2000 local devices.
- Each subckt def must have less than 512 formal nodes.
- Maximum depth of nested subckt call is 20.
- All local and formal nodes must be integers, like all modules of BERT.

1.9 .IDBADAC

Example:

.idbadac

The subroutine *BadAC()* of postbert.c in CAS takes up the most processor time in a CAS run. Therefore, with this command the user can determine if he/she wants to run *BadAC()*. *BadAC()* tells the users if *bad AC* waveforms are encountered. The default is **NOT** to run *BadAC()*. However, the user can force postbert to run *BadAC()* by including *.idbadac* in the input deck.

1.10 .ISUBWIDTH = *colwidth*

Example:

.ISUBWIDTH = 90

This command controls the width of the substrate current output printout in SPICE2. This is independent of the usual **.WIDTH** command. Permissible values for *colwidth* range from 80 to 200. The default value is 80.

1.11 .PMOSDEG *mname* <*keyword1=value*> <*keyword2=value*> ...

Example:

.PMOSDEG PMOSMODEL G1=0.6 UPS=1E-5 HG0=2E3 MG0=1.6 WG=0.9

This command specifies the gate current degradation parameters for the PMOS devices. *mname* is the model name that this parameter set is associated with. The following parameter keywords are recognized:

1) G1: constant coefficient for I_{gate} (default = 0.8).
2) UPS: sensitivity of ϕ_b to the $E_{ox}^{2/3}$ term (default = 4×10^{-5} $V^{1/3}cm^{2/3}$).
3) ECRITP0: Constant term of E_{critp} (default = E_{crit0} of I_{Sub}).
4) ECRITPG: V_{gs} dependence of E_{critp} (default = E_{critg} of I_{Sub}).
5) ECRITPB: V_{bs} dependence of E_{critp} (default = E_{critb} of I_{Sub}).
6) LCG0: Constant term of l_{cg} / $\sqrt{t_{ox}}$ (default = l_{c0} of I_{Sub}).

7) LCG1: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c1} of I_{Sub}).
8) LCG2: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c2} of I_{Sub}).
9) LCG3: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c3} of I_{Sub}).
10) LCG4: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c4} of I_{Sub}).
11) LCG5: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c5} of I_{Sub}).
12) LCG6: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c6} of I_{Sub}).
13) LCG7: Bias sensitivity term of $l_{cg} / \sqrt{t_{ox}}$ (default = l_{c7} of I_{Sub}).
14) HG0: intercept parameter of the lifetime versus I_{gate} plot (default = 10^4).
15) HGGD: V_{gd}-sensitivity term for H_g (default = 0).
16) MG0: slope parameter of the lifetime versus I_{gate} plot (default = 1.5).
17) MGGD: V_{gd}-sensitivity term for m_g (default = 0).
18) WG: weighting coefficient for I_{gate}-based degradation.
19) VFBG: Flat-band voltage for PMOS gate current calculation.

The default value for WG is 1 if the PMOSDEG command is present, 0 if not. See Section 3.2 for the model description.

1.12 .PRINTIGATE or .PLOTIGATE

.PRINTIGATE MXXXX <MYYYY ... MZZZZ> <ALL>
.PRINTIGATE SXXXX <SYYYY ... SZZZZ> <ALL>
.PLOTIGATE MXXXX <MYYYY ... MZZZZ> <ALL> <(MIN,MAX)>
.PLOTIGATE SXXXX <SYYYY ... SZZZZ> <ALL> <(MIN,MAX)>

Examples:

.PLOTISUB S1 S4 (0,7E-6)
.PRINTISUB M1 M4 ALL

These commands are used to either print or plot out the gate current of the specified PMOS transistors. SXXX is the transistor denotation for the BSIM1 model in SPICE2, while MXXX is that for non-BSIM1 models in SPICE2 and all models in SPICE3. Note that the format is similar to the normal **.PRINT** and **.PLOT** commands in SPICE, except that the TRAN keyword is unnecessary. MIN and MAX specify the minimum and maximum values for the plot. The keyword ALL is used if a printout or plotout of the total gate current of all the PMOS transistors in the circuit is desired.

1.13 .PRINTISUB or .PLOTISUB

.PRINTISUB MXXXX <MYYYY ... MZZZZ> <ALL>
.PRINTISUB SXXXX <SYYYY ... SZZZZ> <ALL>
.PLOTISUB MXXXX <MYYYY ... MZZZZ> <ALL> <(MIN,MAX)>
.PLOTISUB SXXXX <SYYYY ... SZZZZ> <ALL> <(MIN,MAX)>

Examples:

.PLOTISUB S1 S4 (0,7E-6)
.PRINTISUB M1 M4 ALL

These commands are used to either print or plot out the substrate current of the specified transistors. SXXX is the transistor denotation for the BSIM1 model in SPICE2, while MXXX is that for non-BSIM1 models in SPICE2 and all models in SPICE3. Note that the format is similar to the normal **.PRINT** and **.PLOT** commands in SPICE, except that the TRAN keyword is unnecessary. MIN and MAX specify the minimum and maximum values for the plot. The keyword ALL is used if a printout or plotout of the total substrate current of all the NMOS and PMOS transistors in the circuit is desired. This is useful to determine whether, for instance, the substrate bias generator used is adequate for the circuit.

1.14 .PROCESS *mname* FILENAME=*fname*

Examples:

.PROCESS PC1 FILENAME=TRN
.PROCESS MK1 FILENAME=NMOS5

This command specifies the model name *mname* and the corresponding model parameter filename *fname* which contains all the device parameters. This configuration is identical to that already implemented for the BSIM1 model in SPICE2, but is new for the other models and SPICE3. It is important to realize that **.MODEL** commands are no longer necessary in the input deck, but that a **.PROCESS** command is now mandatory. All model parameter filenames should be in capital letters if SPICE2 used.

For SPICE Level 1, 2, or 3 models, the model parameter file format contains **.MODEL** commands with the model parameters in the usual SPICE **.MODEL** format. The only restrictions are that the I_{sub} and degradation parameters must be on separate lines from the drain current parameters, and only one model per file is allowed. For the SPICE Level 4 (BSIM1) model, the model parameter file is the file created by the BSIM1 extraction program (see [8] and Fig. A1).

The following provides information concerning the format of these model parameter files.

Additional SPICE Level 1, 2, 3 Parameters:

The following shows the additional parameters and their keywords that can be added to the **.MODEL** parameter declarations.

	Name	parameter	units	default
43	ECRIT0	Constant term of E_{crit}	V/cm	1.0E4
44	ECRITG	V_{gs} dependence of E_{crit}	1/cm	0.0
45	ECRITB	V_{bs} dependence of E_{crit}	1/cm	0.0
46	LC0	Constant term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$	1.0E-7
47	LC1	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$–V	0.0
48	LC2	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$–V^{-1}	0.0
49	LC3	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$	0.0
50	LC4	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$–V	0.0
51	LC5	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$–V^2	0.0
52	LC6	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$	0.0
53	LC7	Bias-sensitivity term of $l_c/\sqrt{t_{ox}}$	$\mu m^{1/2}$–V	0.0
54	H0	Degradation plot intercept (H_0)	A sec / (m $V^{\frac{1}{n}}$)	1.0E4
55	HGD	Degradation plot intercept (H_{gd})	A sec / (m $V^{\frac{n+1}{n}}$)	0.0
56	NN0	Slope of degradation parameter (n_0)		0.5
57	NNGD	Slope of degradation parameter (n_{gd})	V^{-1}	0.0
59	M0	Slope of degradation plot (m_0)		3.5
59	MGD	Slope of degradation plot (m_{gd})	V^{-1}	0.0
60	AGE	Device Age	A sec / m	0.0

*For the parameters H0 and HGD, the unit "*x*" is the unit used for the degradation monitor (e.g. "x" would be volts if ΔV_{th} is monitored).

BSIM1 Process File Modifications (SPICE3 Level 4):

Fig. A1 shows the modified format of the BSIM1 parameter process file. The format is identical to the previous format except five rows have been added below the substrate current parameters.

Rows 35 through 37 contain the coefficients of the H, n, and m degradation parameters. The first column of Row 38 is the Age of the process file. This should be set to zero for a fresh process file. Columns two and three of Rows 38 and 39 are the minimum and maximum channel lengths and widths of the devices that were measured. For the single device case, set $L_{min} = L_{max}$ and $W_{min} = W_{max}$. All entries labeled "DUM" are dummy positions used as placeholders by the program.

The BSIM1 parameter extraction program includes a row of zeroes for Row 35, but no other rows are present. The user must add the extra rows manually and enter the appropriate values. As mentioned previously, the BSIM1 extraction program does not do DC stressing measurements; the degradation parameters must be obtained separately.

1.15 .TRAN *tstep tstop < tstart >*

Examples:

.TRAN 1NS 100NS
.TRAN 5NS 1000NS 2NS

Since this simulator system is designed to calculate transient substrate currents, the SPICE .TRAN command should always be included whenever BERT is used. In order for the degradation calculations to be meaningful, the difference between *tstop* and *tstart* should be equal to a multiple of the period of the input signal.

1.16 General form for MOSFETs :

SXXXX *nd ng ns nb mname* < W=*value* > < L=*value* > ...etc.
MXXXX *nd ng ns nb mname* < W=*value* > < L=*value* > ...etc.

Examples:

S1 1 2 3 4 PC1_NM1_DU1 W=20U L=1U
M1 1 2 3 4 PC1_NM1_DU1 W=20U L=1U
M2 1 2 3 4 MODP W=5U L=10U AD=100P AS=100P PD=40U PS=40U

To describe a MOSFET, the user should use SXXXX for the BSIM1 model in SPICE2, or MXXXX for all other models in SPICE2 and for all models in SPICE3. *mname* is the model name which should always be given. The format for the model name for the BSIM1 model is *pname_mt_dt*, where *pname* is the process name, *mt* is the MOSFET type, and *dt* is the source/drain junction type. The possible choices for *mt* are NM1 through NM5 for NMOSFETs, and PM1 through PM5 for PMOSFETs. DU1 to DU3 are the three available diffusion types. For users who are not familiar with SPICE commands, please consult the SPICE manual. For users who wish to learn more about the BSIM1 model implemented in SPICE or about the BSIM1 parameter extraction program, please refer to [8].

One other note about transistor names. BERT-CAS treats transistors labeled as M1 and S1 as having identical names. Thus, use transistor names that differ from the second character onwards (e.g. M1 and S2).

II. CORS COMMAND SUMMARY

2.1 .TTF <G=G_{300}> <TAU=τ_{300}> <EB=E_b> <DELTA= δ> <QUICK> <t_{BD}^1 ... t_{BD}^8>

Examples:

.TTF
.TTF G=300 TAU=10p QUICK
.TTF 3600 86400 2.59Meg 5.18Meg 7.77Meg 31.54Meg 63.07Meg .315G

The .TTF card must be present for CORS to be invoked by BERT. If it is not in the input deck, all other CORS commands will be ignored. G and TAU, which characterize time dependent dielectric breakdown for a particular technology, are obtained by conducting an intrinsic oxide breakdown study (instructions are included with Figure A2). EB and DELTA, which characterize the temperature acceleration of oxide breakdown, are obtained by conducting a study of the temperature dependence of breakdown (instructions are included with Figure A3). The inclusion of the QUICK option indicates that user wants the "quick" algorithm used (detailed in Sections 4.1.4 and 4.1.5) which decreases the CORS run-time with some loss of accuracy. t_{BD}^1 through t_{BD}^8 are the operating times for which the failure statistics will be generated. If the user wishes to specify the values of t_{BD}, he/she must specify all eight values. All parameter except $t_{BD}^1 \ldots t_{BD}^8$ will be ignored if the .EMODEL card is present in the input deck.

name	parameter	units	default
G	G_{300}	MV/cm	350
TAU	τ_{300}	sec	1.0E-11
EB	E_b	eV	.28
DELTA	δ	eV	.0167
QUICK	-	-	not quick
-	t_{BD}^1	sec	1 month
-	t_{BD}^2	sec	3 months
-	t_{BD}^3	sec	6 months
-	t_{BD}^4	sec	1 year
-	t_{BD}^5	sec	2 years
-	t_{BD}^6	sec	5 years
-	t_{BD}^7	sec	10 years
-	t_{BD}^8	sec	20 years

2.2 **.XEFF** *MNAME* **FILENAME=***areadata* **<DIFFEDGE=***diffdata***> <OXEDGE=***foxdata***>**
Examples:
.XEFF POLYCAP FILENAME=DEFDATA
.XEFF NMOS1 FILENAME=MOSDATA DIFFEDGE=DRAINDATA OXEDGE=OXDATA

MNAME is the model name listed on a .MODEL (SPICE) or .ALTMODEL (CORS) card for a MOSFET or capacitor. There must be one .XEFF card for each MOSFET model name defined in the input deck. Each capacitor which the user wishes included in the breakdown projections will be associated with a model name which must also be found in a .XEFF card. The file named after keyword FILENAME should contain defect density (cm^{-2}) data appropriate for devices described by MNAME. The file named after keyword DIFFEDGE should contain defect density (cm^{-1}) data for the diffusion (source/drain) edges of a MOSFET or for the width-wise edges of a capacitor. The file named after keyword OXEDGE should contain defect density (cm^{-1}) data for the field oxide edges of a MOSFET or the length-wise edges of a capacitor. All data files should have been formated by program DEFECT.

2.3 Capacitors

General form:
SPICE 2 or 3:
C*XXXXXXX N+ N-* **VALUE <IC=***INCOND***> <TBDMODEL=***MNAME* **L=***length* **W=***width***>**

SPICE 3 only:
C*XXXXXXX N+ N-* **MNAME L=***length* **<W=***width***> <IC=***INCOND***> <TBDMODEL=***MNAME***>**

Examples:
C1 4 0 1.4P TBDMODEL=CMODEL L=10U W=50U
CABC 10 3 CAPMOD L=10U W=50U IC=5V TBDMODEL=CAPMOD

A capacitor is included in the dielectric breakdown calculations only if the TBDMODEL keyword is included on the card which defines the capacitor. If the first capacitor card format illustrated above is used, then an .ALTMODEL card which defines MNAME must be included in the deck. Capacitor breakdown statistics can not be calculated for IRSIM runs.

2.4 MOSFETs

If the Level 1, 2 or 3 SPICE MOSFET model is being used, then no changes need be made to accommodate CORS. If the Level 4 model (BSIM) is being used, then an .ALTMODEL card must be created for each model (or "process") name which appears on a MOSFET definition card.

2.5 .ALTMODEL *MNAME TYPE <parameters>*

Examples:

```
.ALTMODEL NM1 NMOS VTO=.7 GAMMA=.4 TOX=20N PHI=.6 TPG=1
.ALTMODEL CAP1 C TOX=12.5N
```

MNAME is a model name which is also found on one or more MOSFET or capacitor definition cards. TYPE is the type of device, NMOS, PMOS or C (capacitor).

name	parameter	units	default
VTO	zero-bias threshold voltage	V	1.0
TOX	oxide thickness	meter	1.0e-7
GAMMA	bulk threshold parameter	V	0.0
PHI	surface potential	V	.6
TPG	type of gate material: +1 opposite of substrate -1 same as substrate 0 Al gate	-	1

2.6 .EACHPROB *<NUM, ALL>*

Examples:

```
.EACHPROB ALL
.EACHPROB 10
```

The .EACHPROB card appends to the CORS output a listing of the failure probability for each of the NUM devices most likely to fail. These individual failure probabilities are calculated for an operating time of t_{BD}^{7} (default 10 years).

2.7 .LSI *<num1> <num2> <num3>*

Example:

```
.LSI 256K 1M
```

The .LSI card may be used to print out failure probabilities for large circuits which are constructed of cells identical to the circuit described in the input deck. The arguments (up to three) specify the number of cells in the large circuit(s).

2.8 .BURNIN <TIME=*burntime*> <TEMP=*burntemp*> Example:

```
.BURNIN TIME=14400 TEMP=150
```

The presence of a .BURNIN card indicates that the user wishes to have the effects of burn-in simulated. *The user may not request CAS or BiCAS analysis along with burn-in. All other CORS commands are compatible with CAS and BiCAS commands.* TIME is the duration of the burn-in

trial, it has units of seconds. TEMP is the temperature (C°) during the burn-in trial.

To simulate burn-in, CORS must be run in a two-pass mode. During the first pass, the user should adjust the power supply voltage values defined in the input deck to the values used during burn-in. If the user typically performs burn-in using different signal waveforms from those during normal operation, the burn-in waveforms should be defined in the input deck during the first pass. During the second pass, the power supply voltage values defined in the input deck should be set to their normal operating values and signal waveforms should be those expected under normal conditions. *Except for voltage source cards, the input deck should be identical during both simulator passes.* Output is only generated after the second pass. Two temporary files are created during the first pass and not erased until the completion of the second pass; these are named rawtddb and rawburn.

2.8 .OPTIONS OPT1=*optval* OPT2=*optval* . . .

Example:

.OPTIONS TNOM=55

Simulation of circuit reliability at a given ambient temperature can be specified by using the SPICE .OPTIONS card as illustrated above. TNOM has units of C°.

2.9 .EMODEL

Example:

.EMODEL

The .EMODEL card instructs CORS to use the "E model" for time-dependent dielectric breakdown (described in Section 4.1.6) rather than the default "$\frac{1}{E}$ model."

III. EM COMMAND SUMMARY

The following commands are inserted in the SPICE input deck to request EM analysis. Only one command can appear in each line.

3.1 .EMMODEL *filename*

Example:

.EMMODEL emrulefile

This command tells the simulator that reliability parameters are in emrulefile.

3.2 .EMSTAT *filename hour* [*hour...*]

Example:

.EMSTAT bicmos.int 100.0 1.0e3 2e4

This command specifies the failure rates are to be calculated at times: 100 hours, 1000 hours and 20000 hours. The geometry file which contains the length and width of interconnects, number of contact and via openings that the user set up (by hand) is *bicmos.int.*

3.3 .EMSTATX *filename hour* [*hour...*]

Example:

.EMSTATX fulladder.geo 200.0

This command specifies the failure rates are to be calculated at times: 200 hours. The geometry file extracted from the layout by the extractor is *fulladder.geo.*

Either one of **.EMSTAT** or **.EMSTATX**, but not both is needed if user requests failure rate calculation for a circuit layout.

3.4 RELIABILITY PARAMETERS IN THE EM RULE FILE

The simulator needs as input the necessary reliability parameters to perform electromigration analysis. The parameters are given in a file (which will be referred to as the EM design rule file). There are two types of parameters: numerical and logical. The parameter name must be typed in full.

- Logical parameters are entered without assignment. It is set true if it appears. Example:
  PRINTCURRENT
- Numerical parameters are assigned values by following the name of the parameter immediately by an equal sign, and the value. The value must also immediately follow the equal sign. There can be no blank space in between. Example:
  LENGTH=1.2e4
- Comments can be entered by entering * in the first column of the comment line. Entries in that line will not be read by the simulator. Example:
  *This is a comment line

3.4.1 LOGICAL PARAMETERS IN THE EM RULE FILE

3.4.1a **PRINTCURRENT**

Request simulator to print out the current in each connection.

3.4.1b **SKIPLAYOUTCUR**

Normally, the simulator generates the layout advisory table containing the width and length of the interconnects, the safety factor of the contacts and vias for specified current values. This option requests the simulator not to generate this table.

3.4.1c **SKIPLAYOUTGEO**

Normally, the simulator generates the layout advisory table containing the width and length of each interconnect in the circuit, and the safety factor of contacts and vias at every connection of the circuit to meet the reliability specifications. This option requests the simulator not to generate this table.

3.4.1d **SKIPFAILRATE**
3.4.1e **SKIPFAILPERCENT**

The simulator will not print out the cumulative percent failure table (if **SKIPFAILPERCENT** is specified) or the failure rate table (if **SKIPFAILRATE** is specified). The default is both tables will be printed if either the **.EMSTAT** or **.EMSTATX** card is found in the SPICE input deck.

3.4.1f **METAL1**

The reliability parameters that follow are for metal-one.

3.4.1g **METAL2**

The reliability parameters that follow are for metal-two.

3.4.1h **METAL3**

The reliability parameters that follow are for metal-three.

3.4.1i **CONTACT**

The reliability parameters that follow are for metal-one contact to diffusion.

3.4.1j **VIA or VIA1**

The reliability parameters that follow are for metal-one to metal-two vias.

3.4.1k **VIA2**

The reliability parameters that follow are for metal-two to metal-three vias.

3.4.2 NUMERICAL PARAMETERS IN THE EM RULE FILE

3.4.2a **NCURRENT=***ncurrent current1 current2 ... currentn*
Example:
NCURRENT=4 1.0e-4 2.0e-4 1.0e-3 2.0e-3

ncurrent is the number of current values the simulator uses to generate the current layout advisory table. *current1..currentn* are given in Amperes. There must be exactly *ncurrent* fields following *ncurrent* and they must all be in one line. The default is *ncurrent=3 .50e-3 1.00e-3 2.00e-3.*

3.4.2b **AC_DEFINE=***ac_define*
Example:
AC_DEFINE=0.1

ac_define is used as a criterion for pure AC waveforms. The simulator will treat a current waveform as a pure AC if the average current is less than or equal to *ac_define* × the average of the absolute current. This parameter is important because quite often the transient analysis in SPICE does not result in pure AC waveforms when charging and discharging of a capacitor node. This happens when the time duration requested in the .TRAN card does not cover one period under steady-state operation. Usually a value of 0.1-0.2 is reasonable. To obtain steady-state waveforms, user is advised to set up SPICE to run for more than one cycle of the waveform. The default is *AC_define=0.0*

3.4.2c **MINJCURRENT=***minJcurrent*
Example:
MINJCURRENT=1.0e3

In order to save computation time, when the current density in the interconnect, contact or via is below **minJcurrent** (in A/cm^2), the simulator will skip failure calculation and print **J <MinJ** in the output. The default is *MinJCurrent=1e2.*

3.4.2d **WORSTLIST=***WorstList*
Example: WORSTLIST=0.5

The simulator will print out worst *WorstList* fraction of the connections in the failure rate/cumulative percent failure tables. In the example, half (50%) of the worst connections will be printed. The default is *WorstList=1.0.*

3.4.2e **NWIDTH=***nwidth width1 width2 ... widthn*
Example:
NWIDTH=3 0.8 1.2 2.0

nwidth is the number of widths the simulator will use to generate the layout advisory table for interconnects. *width1..widthn* are given in μm. There must be exactly *nwidth* fields following *nwidth*, and they must all be in one line. The default is *NWidth=4 1.0 2.0 4.0 10.0.*

3.4.2f **NCV=***ncv N1 N2 ... Nn*
Example:
NCV=4 1.0 2.0 3.0 4.0

ncv tells the the simulator that the user wishes a layout advisory table for contacts and vias containing *ncv* columns for *N1..Nn* openings. In each column a safety factor is generated for the number of contact or via openings (in parallel) at that connection. There must be exactly *ncv* fields following *ncv* and they must all be in one line. The example shown above will request the simulator to calculate the safety factor for connections having: one contact/via, two contact/via, three contact/via and four contact/via openings. The default is *NCV=2 1.0 2.0*.

3.4.2g **SPEC_TIME=***spec_time*
Example:
SPEC_TIME=1.0e6

spec_time is the number of operating hours when the failure rate is *spec_failrate* defined by **SPEC_FAILRATE=**. *spec_time* is in hours. The simulator will generate a layout advisory to meet this spec. The default is *Spec_Time=1.0e4*.

3.4.2h **SPEC_FAILRATE=***spec_failrate*
Example:
SPEC_FAILRATE=1.0e-3

spec_failrate (in 1/hour) is the failure rate at *spec_time* (hours) defined in **SPEC_TIME**. Note that 1 FIT = 0.1% percent (10^{-3}) failures per million (10^{6}) device hours is equal to a **SPEC_FAILRATE** of 10^{-9}. The simulator will generate a layout advisory to meet this spec. The default is *Spec_Failrate=1.0e-9*.

3.4.2i **TOP=**T_{op}
Example:
TOP=25

The circuit operates at temperature a T_{op} (in °C). The accelerated testing data entered by the user will be extrapolated to this temperature. The default is *Top=25.0*.

3.4.3 PARAMETERS SPECIFYING LINES AND CONTACTS/VIAS

The following parameters apply to metal-one, metal-two, metal-three, metal-one to diffusion contact and metal-to-metal via. To associate the parameters to one of the connection types, one of the logical parameters: **METAL1, METAL2, METAL3, CONTACT, VIA** and **VIA2** must be set before any of the following appears. Items (a) to (n) must be given to complete the parameter set for each connection type. The simulator will skip any connection type with an incomplete parameter set.

3.4.3a **ADC=**A_{DC}
3.4.3b **AAC=**A_{AC}
3.4.3c **M=m**
3.4.3d **TDATA=**T_{data}
3.4.3e **EA=**E_a

These are the parameters in Eq.(5.1). Note that the user will have to enter the experimentally determined values of A_{DC} and A_{AC} (both are in units of hours × $(A/cm^2)^2$). These values are obtained from experiments at temperature T_{data} (in °C). E_a is the activation energy for A_{DC} and A_{AC}.

3.4.3f **THICK=***thickness*
3.4.3g **WIDTH=***width*
3.4.3h **LENGTH=***length*

These parameters specify the *thickness*, *width* and *length* (all in μm) of the interconnects used in the electromigration lifetime experiment to extract A_{DC}, A_{AC} and m.

3.4.3i **AREA=***area*
3.4.3j **NCHAIN=***nchain*

The first parameter specifies the *area* (in μm^2) of each contact or via in the chain test structure used in lifetime experiment to extract A_{DC}, A_{AC} and m. *nchain* is the number of contacts or vias in series in the test structure. The default value is *nchain=1.0.*

3.4.3k **LOGMEDIAN=***MTF*
3.4.3l **LOGSIGMA=**σ

These parameters select either the lognormal distribution with median *MTF* (in hours) or the log-normal standard deviation σ. The MTF is the experimental median-time-to-failure at temperature T_{data}.

3.4.3m **WEIBULL_A=**α
3.4.3n **WEIBULL_B=**β

These parameters select the Weibull distribution and specify the parameters α and β. The failure data is collected at a temperature T_{data}. The Weibull cumulative distribution function is described by:

$$F(t) = 1 - \exp(-\frac{1}{\alpha} t^{\beta}) \qquad (5.4)$$

where t is in hours.

3.4.3o **WIDTH_A=**A_w
3.4.3p **WIDTH_B=**B_w
3.4.3q **WIDTH_C=**C_w

These parameters specify the width dependence of the time-to-failure for interconnect (see Appendix E for an example). The parameters A_w, B_w, C_w are defined by Eqs.(5.3a) and (5.3b.) for $W \geq B_w$:

$$TTF(W) = A_w \times (W - B_w)^2 + D_w \qquad (5.3a)$$

for $W < B_w$:

$$TTF(W) = C_w \times (W - B_w)^2 + D_w \qquad (5.3b)$$

D_w is found using the previously entered experimental time-to-failure and *width*. W is in μm. Note that by setting, $A_w = C_w = 0$, TTF will be a constant, independent of width. The default values are *Width_A=0.0 Width_B=0.0 Width_C=0.0.*

IV. BIPOLAR DEGRADATION COMMAND SUMMARY

4.1 **.AGEBJT** *time*
Examples:
.AGEBJT 10years
.AGEBJT 5hours

This command specifies the future time at which to calculate the aged bipolar model parameter files for circuit simulation. The units for time can be in "y","h","m", or "s", corresponding to years, hours, minutes, and seconds, with no space between the number and the unit. Letters following the above four units of time are ignored. Thus **10minutes** and **10m** are interpreted identically.

4.2 **.AGEDIB** *time* **< VBE=** *vbevalue***>**
Example:
.AGEDIB 1year VBE=0.8

This command specifies the future time at which base current degradation is desired. The

vbevalue value is used to calculate ΔI_B at *time*. If **VBE=***vbevalue* is not specified, then a default value of 0.6 V is used. If *vbevalue*'s are specified for both **AGEDIB** and **DELTAIB**, then the second *vbevalue* will be used for both **AGEDIB** and **DELTAIB** computation.

4.3 .AGEMETHODBJT *method <domain>*
Example:
.AGEMETHODBJT INTERP LINLOG
.AGEMETHODBJT LINLIN

This command specifies the method of numerical analysis used to calculate the aged parameters from the pre-stressed model parameters. The first argument specifies the method of the regression analysis (LINLIN, LINLOG, or LOGLOG). The keyword INTERP should be placed in this position if interpolation rather than regression is desired. The keyword INTERP can be followed by the method in which the inpterpolation will be performed (LINLIN, LINLOG, or LOGLOG). The default is linear-log interpolation if no **AGEMETHODBJT** card is found.

4.4 .AGEPROCBJT *mname* **FILENAMES=***fname1,fname2,fname3 <,fname4,...>*
Example:
.AGEPROCBJT NPN1 FILENAMES=NBJT0,NBJT1,NBJT2,NBJT3

This command specifies the names of the pre-stressed model parameter files *fname* associated with the model *mname*. The filenames should be ordered by increasing ages, with the fresh file first. At least one fresh and one aged model parameter file must be present for linear-linear analysis, while two aged model parameter files must be present for linear-log or log-log analysis. Note that unlike the **.BJTPROC** statement, " **FILENAMES** " appears in plural form. The **.BJTPROC** command is still needed.

4.5 .BJTPROC *mname fname1* **FILENAME=***fname2*
Example:
.BJTPROC NPN1 IRDATA FILENAME=TRN

This command specifies the model name *mname*, the reverse I-V data file *fname1*, and the corresponding model parameter filename *fname2* which contains all the bipolar device parameters. It should be noted that **.BJTPROC** and the information stored in *fname2* are in lieu of the SPICE **.MODEL** card.

fname1 should contain two columns, the left one being the reverse voltage and the right one the corresponding reverse current. The values for reverse voltage should be in increasing order, with the lowest value listed on the first line. There should not be any additional lines such as comments in *fname1*. If V_R is greater than the largest value in *fname1*, then I_R is set to the value corresponding to the largest V_R value in *fname1* (the value on the last line). If V_R is smaller than the smallest value in *fname1*, then I_R is set to 0. Because the stress parameters are size dependent, the bipolar model does not recognize the area scaling parameter that is optional in the normal bipolar transistor line corresponding to **QXXXX**. The area scaling factor should be omitted, and SPICE decks provided for each device geometry.

The following shows the keywords for the bipolar degradation parameters:

I1: reverse current at which D′ and b change from pre- to heavy- avalanche region (default=∞)
D0: value of D′ for I_R<I1 (default=0)
A0: a, reciprocal of the non-ideality factor (default=0.5)
B0: b, power dependence of ΔI_B on I_R for I_R<I1 (default =0.5)
C0: n, power dependence of ΔI_B on t (default=0.5)
D1: value of D′ for I_R>I1 (default=0)
B1: b, power dependence of ΔI_B on I_R for I_R>I1 (default =0.5)
V0: lower V_{BE} value for regression fitting for effective I_{SE} and n_E (default=0.2)
V1: upper V_{BE} value for regression fitting for effective I_{SE} and n_E

(default=$n_F V_t \ln(10\beta_F \Delta I_{B,SUM}/I_S)$)
AGE: device age (default=0)

4.6 **.DEGPRINTBJT** *trnname1 <trnname2...>*
Example:
.DEGPRINTBJT Q1 Q5 Q11

This command restricts the degradation information printout to only the specified bipolar transistors. Without this command, degradation information for all the bipolar transistors in the circuit will be printed out.

4.7 **.DEGSORTBJT**
Example:
.DEGSORTBJT

This command requests a printout in tabular form of all the transistors in the circuit listed from the most degraded to the least degraded. The corresponding device lifetime is given if the **.DELTAIB** command is present, the amount of device degradation is given if the **.AGEDIB** command is present, and the age of each transistor is given if the **.AGEBJT** command is present.

4.8 **.DELTAIB** *value* **<VBE=***vbevalue***>**
Example:
.DELTAIB 1uA VBE=0.8

This command specifies the base current degradation, ΔI_B, at which the device lifetime is defined. Like the **AGEDIB** command, the *vbevalue* value is used in the calculation. If **VBE=***vbevalue* is not specified, then a default value of 0.6 V is used. If *vbevalue*'s are specified for both **AGEDIB** and **DELTAIB**, then the second *vbevalue* will be used for both **AGEDIB** and **DELTAIB** computation.

4.9 **.PLOTIREV** *QXXX <QYYY...QZZZ><(MIN,MAX)>*
Example:
.PLOTIREV Q1 Q4 (0,5E-6)

This command is used to plot the reverse current of the specified bipolar transistors. MIN and MAX specify the minimum and maximum values for the plot.

4.10 **.PRINTIREV** *QXXX <QYYY...QZZZ>*
Example:
.PRINTIREV Q2 Q6

This command is used to print the reverse current of the specified bipolar transistors.

4.11 **QXXXX** *nc nb ne ns mname*
Example:
Q1 1 2 3 4 NPN1
Q2 1 2 3 NPN2

mname (defined in **.BJTPROC**) is the model name which should always be given. The substrate node *ns* is optional. If *ns* is not specified, **GND** is assumed.

	Name	L sens. factor	W sens. factor	Units of basic parameter
1	V_{FB} (VFB)	V_{FBl} (LVFB)	V_{FBw} (WVFB)	V
2	ϕ_S (PHI)	ϕ_{Sl} (LPHI)	ϕ_{Sw} (WPHI)	V
3	K_1 (K1)	K_{1l} (LK1)	K_{1w} (WK1)	$V^{1/2}$
4	K_2 (K2)	K_{2l} (LK2)	K_{2w} (WK2)	-
5	η_0 (ETA)	η_{0l} (LETA)	η_{0w} (WETA)	-
6	μ_Z (MUZ)	δ_l (DL)	δ_w (DW)	$cm^2/V\text{-}s$, μm, μm
7	U_{0Z} (U0)	U_{0Zl} (LU0)	U_{0Zw} (WU0)	V^{-1}
8	U_{1Z} (U1)	U_{1Zl} (LU1)	U_{1Zw} (WU1)	μm V^{-1}
9	μ_{ZB} (X2MZ)	μ_{ZBl} (LX2MZ)	μ_{ZBw} (WX2MZ)	$cm^2/V^2\text{-}s$
10	η_B (X2E)	η_{Bl} (LX2E)	η_{Bw} (WX2E)	V^{-1}
11	η_D (X3E)	η_{Dl} (LX3E)	η_{Dw} (WX3E)	V^{-1}
12	U_{0B} (X2U0)	U_{0Bl} (LX2U0)	U_{0Bw} (WX2U0)	V^{-2}
13	U_{1B} (X2U1)	U_{1Bl} (LX2U1)	U_{1Bw} (WX2U1)	μm V^{-2}
14	μ_S (MUS)	μ_{Sl} (LMS)	μ_{Sw} (WMS)	$cm^2/V^2\text{-}s$
15	μ_{SB} (X2MS)	μ_{SBl} (LX2MS)	μ_{SBw} (WX2MS)	$cm^2/V^2\text{-}s$
16	μ_{SD} (X3MS)	μ_{SDl} (LX3MS)	μ_{SDw} (WX3MS)	$cm^2/V^2\text{-}s$
17	U_{1D} (X3U1)	U_{1Dl} (LX3U1)	U_{1Dw} (WX3U1)	μm V^{-2}
18	T_{ox} (TOX)	T_{emp} (TEMP)	V_{dd} (VDD)	μm, °C, V
19	CGDO	CGSO	CGBO	F/m
20	XPART	DUM1	DUM2	-
21	N0	LN0	WN0	-
22	NB	LNB	WNB	-
23	ND	LND	WND	-
24	E_{crit0} (ECRIT0)	E_{crit0l} (LECRIT0)	E_{crit0w} (WECRIT0)	V/cm
25	E_{critg} (ECRITG)	E_{critgl} (LECRITG)	E_{critgw} (WECRITG)	1/cm
26	E_{critb} (ECRITB)	E_{critbl} (LECRITB)	E_{critbw} (WECRITB)	1/cm
27	l_{c0} (LC0)	l_{c0l} (LLC0)	l_{c0w} (WLC0)	$\mu m^{1/2}$
28	l_{c1} (LC1)	l_{c1l} (LLC1)	l_{c1w} (WLC1)	$\mu m^{1/2}$_ V
29	l_{c2} (LC2)	l_{c2l} (LLC2)	l_{c2w} (WLC2)	$\mu m^{1/2}$_ V^{-1}
30	l_{c3} (LC3)	l_{c3l} (LLC3)	l_{c3w} (WLC3)	$\mu m^{1/2}$
31	l_{c4} (LC4)	l_{c4l} (LLC4)	l_{c4w} (WLC4)	$\mu m^{1/2}$_ V
32	l_{c5} (LC5)	l_{c5l} (LLC5)	l_{c5w} (WLC5)	$\mu m^{1/2}$_ V^2
33	l_{c6} (LC6)	l_{c6l} (LLC6)	l_{c6w} (WLC6)	$\mu m^{1/2}$
34	l_{c7} (LC7)	l_{c7l} (LLC7)	l_{c7w} (WLC7)	$\mu m^{1/2}$_ V
35	H_0 (H0)	H_{gd} (HGD)	DUM3	*
36	n_0 (NN0)	n_{gd} (NNGD)	DUM4	-
37	m_0 (M0)	m_{gd} (MGD)	DUM5	-
38	AGE	L_{min} (LMIN)	W_{min} (WMIN)	**, μm, μm
39	DUM6	L_{max} (LMAX)	W_{max} (WMAX)	-, μm, μm

*: A sec / (m $x^{1/n}$), where x is the unit used for the degradation monitor.
**: A sec / m

Fig. A1 The modified BSIM1 model parameter file format to be used with BERT.

Measuring G_{300} and τ_{300}

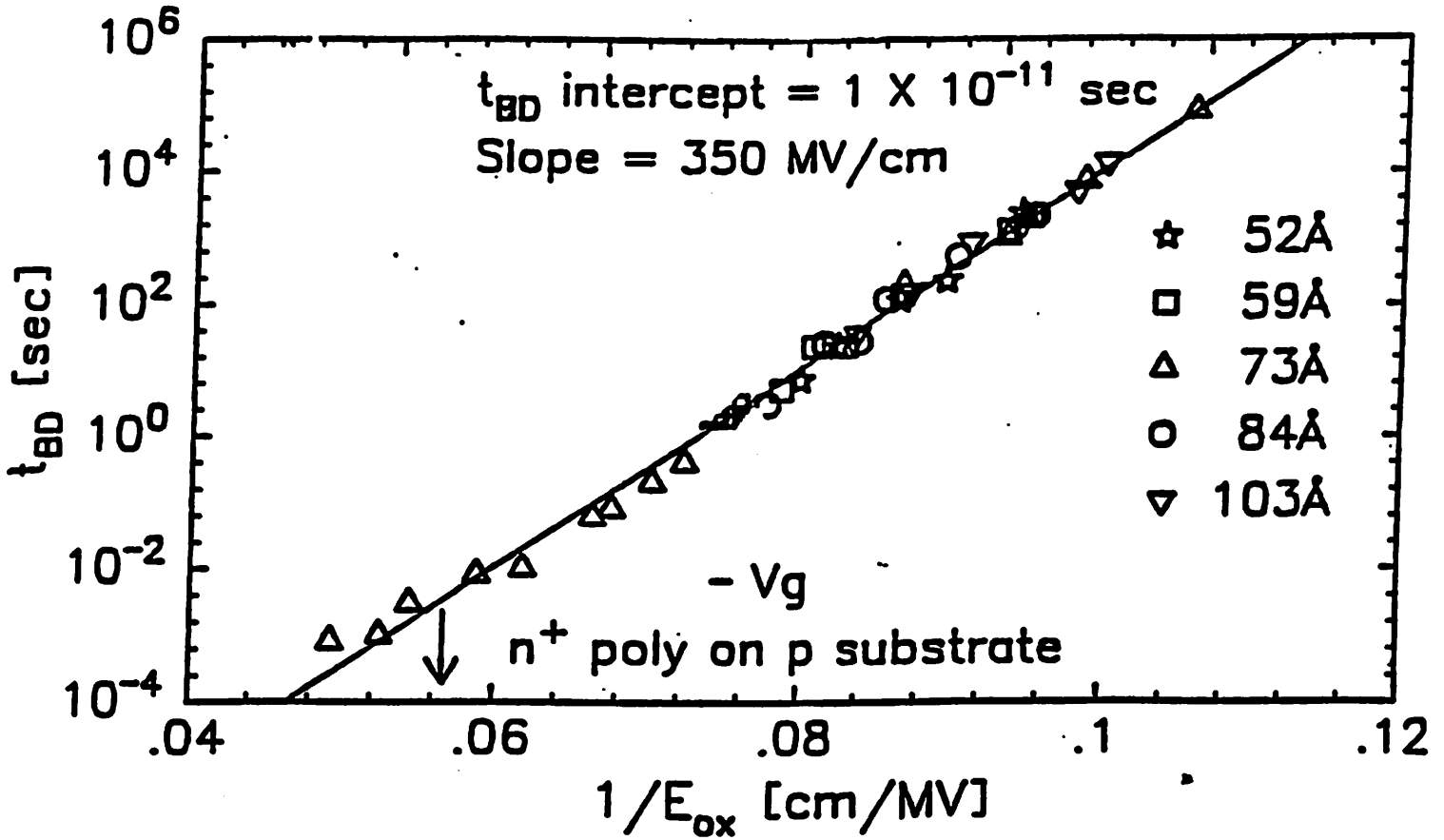
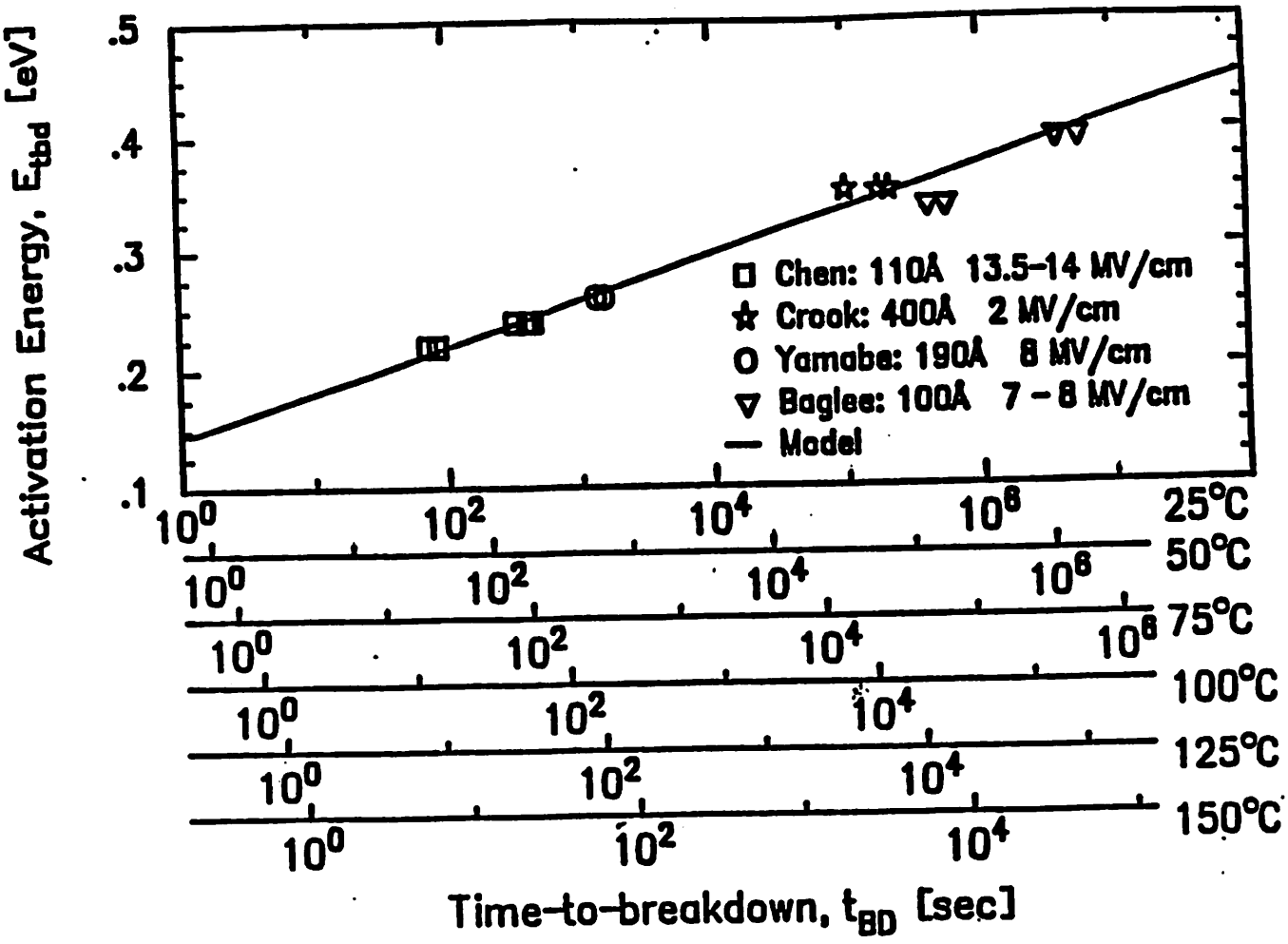


Fig. A2

Take an ensemble of *small* area capacitors and measure intrinsic time-to-breakdown as a function of electric field at room temperature. Plot $\log(t_{BD})$ vs $\frac{1}{E_{ox}}$ shown above. The slope is G_{300} and the y-intercept is τ_{300}.

Determination of E_b and ò



$$\text{Activation Energy: } E_{tbd} = k_B \frac{d(\ln(t_{BD}))}{d(\frac{1}{T})}$$

The t_{BD} in the above equation is measured for some arbitrary breakdown test yield. E_{tbd} was plotted (above) for several researchers' data and consistent results were found. Solving (4) and (5) for E_{tbd}, one finds

$$E_{tbd} = \frac{\ln\left[\frac{t_{BD}(T)}{\tau}\right] + \frac{E_b}{k_B}\left[\frac{1}{T} - \frac{1}{300}\right]}{1 + \frac{\delta}{k_B}\left[\frac{1}{T} - \frac{1}{300}\right]} * \delta - E_b$$

δ=.0167eV and E_b=.28eV are obtained from the linear fit shown in the above figure. These are believed to be valid for all oxides at temperatures 25–150° C. However, the user may repeat this experiment and provide CORS with different values of δ and E_b.

Fig. A3

Appendix B

Bert Error Messages

The following list contains the error messages of the pre- and post-processor of BERT and the BERT shell script including the routine name in which they occur. Error codes in the 'Cxx:' format are CAS errors; those in the 'Bxx:' format are BERT shell script errors. Furthermore, for CAS, two-digit codes represent pre-processing errors while three-digit codes represent post-processing errors. Error codes for CORS, the electromigration simulator, and BiCAS are in the format 'Txx:,' 'EMxx:,' and 'Bixx:', respectively. Furthermore, Error codes from the subcircuit expansion module are in the format 'EXxx:'

CAS Pre-processor Errors:

prebert.c:

Main:

C93: No transistor is listed in input deck!

ArgU:

C01: No input file specified!

C02: Cannot open *<input filename>* !

C03: Specified option not valid!

C04: Incorrect option or file specification!

C87: Missing filename after -b !

C88: Missing filename after -c !

C89: You must must have an external BERT cmd file to run bert-irsim !

PreFilter:

C05: Missing .process command in the input deck!

C06: Missing .ageproc command in the input deck!

C07: Missing .age command in the input deck!

FindIsub:

C08: Invalid .printisub or .plotisub command!

FindIgate:

C09: Invalid .printigate or .plotigate command!

GetDelta:

C10: No lifetime criteria given for the .deltavt command!

C11: No lifetime criteria given for the .deltaid command!

C12: No lifetime criteria given for the .deltagm command!

GetAge:

C13: No future time given for the .age command!

C14: Incorrect format for the future time given in the .age command!

AgeDeg:

C15: No future time given for the <*command*> command!

C16: Incorrect format for the future time given in the <*command*> command!

FindProc:

C17: Cannot open rawinp1 file!

C18: Insufficient memory space. Reduce the number of model parameter files!

C19: Incorrect .process command format!

C20: Too many model parameter files!

C21: No model parameter file(s) specified!

C22: Missing or incorrect model parameter filename specified!

CreateInpFile:

C23: Insufficient memory space. Too many transistors!

C24: Cannot open the <*model name*> rawmodel file !

FindIsubIgateOut:

C25: Invalid .printisub, .plotisub, .printigate, or .plotigate command!

SubstituteLine:

C26: No transistor model name specified!

procsub.c:

Proc2ModSub:

C27: Insufficient memory space!

C28: Cannot open model parameter file <*model parameter filename*>!

C29: Error in reading model parameter file!

C30: Illegal header line in model parameter file!

CreateRawprocess:

C31: Insufficient memory space!

C32: Illegal header line in process file!

C33: Parameters for BSIM1 model missing in model parameter file!

ChkModel:

C34: Cannot open model parameter file <*model parameter filename*>!

C35: Cannot write into temporary model parameter file <*model parameter filename*>!

C36: No MOS model parameters in the <*process filename*> model parameter file!

getdata:

C37: Premature end of file reading BSIM1 model parameter file!

premisc.c:

OpenInpFile:

C38: Cannot open rawinp1 file!

OpenRaw:

C39: Cannot open rawsub file!

getvalue:

C40: Insufficient memory space in reading in BSIM1 parameters!

C41: Premature end of file reading BSIM1 model parameter file!

age.c:

MemAlloc:

C42: Insufficient memory space!

GetAgecards:

C43: Insufficient memory space!

C44: Incorrect .ageproc command format!

C45: Insufficient memory space. Too many model parameter files!

GetDevAge:

C46: Cannot open agetable!

C47: Insufficient memory space!

C48: Insufficient memory space. Too many aged transistors!

C91: No aged transistor data. Check agetable file!

ReadAgePar:

C49: Insufficient memory space!

C50: Incorrect .process command format!

C51: Cannot open model parameter file <*model parameter filename*>!

C52: Cannot open .ageproc model parameter file <*model parameter filename*>!

C53: Illegal header line in model parameter file <*model parameter filename*>!

C54: Mixture of SPICE and BSIM models in same .ageproc command not allowed!

BSIMGetParm:

C55: Insufficient memory space!

C56: Illegal header line in model parameter file <*model parameter filename*>!

SPICEGetParm:

C58: Invalid model name declared in model parameter file!

C59: Invalid model type declared in model parameter file!

ParmExt:

C60: Not enough pre-stressed model parameter files for model <*model name*>!

C61: Not enough pre-stressed model parameter files for model <*model name*>!

GenInpDeck:

C62: Insufficient memory space!

bsimext.c:

GetWLparm:

C63: Insufficient memory space. Too many model parameter files!

BSRegress:

C64: Not enough pre-stressed model parameter files!

PreRegress:

C65: Unable to do log-log regression to find aged parameters!

BSInterp:

C66: Insufficient memory space!

PreInterp:

C67: Pre-stressed model parameter files not ordered from least to most aged!

C68: Method of interpolation not specified!

leastsq2:

C69: Least square approximation failed due to bad parameter data!

leastsq2_2vars:

C70: Least square approximation failed due to bad parameter data!

leastsq3b:

C71: Least squares approximation reduction failed due to small pivot!

spext.c:

SPICERegress:

C72: Insufficient memory space!

C73: Not enough pre-stressed model parameter files!

SPICEInterp:

C74: Insufficient memory space!

SPGenAgeParm:

C75: Cannot write aged model parameter files in present directory!

preirsim.c:

FindConstNode: (Also in Writerawtddb, WriteIrsimTran, FindXeff)

C90: Cannot open *<filename>* for reading !

FindXeff:

C92: No .XEFF card in input deck!

CAS Post-processor errors:

postbert.c:

main:

C101:Could not create 'rawout' file!

C102:Cannot open rawout1 file!

ErrorCheck:

C103:Cannot open rawsub file!

SubAnalysis:

C104:Cannot open agetable!

C105:Insufficient memory space!

AddSubParam:

C107:Insufficient memory space. Too many model parameter files!

C108:BSIM1 interconnect model parameters could not be found!

C109: .END command is missing from the input file!

C146:No process files listed in input deck!

MemAlloc:

C110:Timestep too small in reading voltage values!

C111:Insufficient memory space. Too many timesteps!

ReadVoltage:

C112:Voltage printout for substrate current analysis not found!

C113:Division by zero in reading voltages!

C114:Timesteps too small in reading voltages!

FindInfo:

C115:Invalid spice type specification!

C116:The .tran card missing!

readpar.c:

ObtainTrans:

C117:Insufficient memory. Too many Isub- or Igate-requested transistors!

C118:Insufficient memory space. Too many transistors!

C147:No transistor is listed in rawsub file!

ObtainModelCards:

C119:Cannot open rwmd<x> file!

C120:Insufficient memory!

C121:Insufficient memory. Too many model parameter files!

C148:No process files listed in input deck!

C149:No MOSFET models listed in input deck!

ObtainPMOSDegParam:

C143:Insufficient memory space!

mos.c:

BSIM1evaluate:

C126:Phi is negative in BSIM1 (Level 4) model!
C127:Phi = 0 in BSIM1 (Level 4) model!
C128:Vdd = 0 in BSIM1 (Level 4) model!
C129:Non-positive mobility given in BSIM1 (Level 4) model!

BSIM1setup:
C122:Division by zero in BSIM1 parameter calculation!

SPICEsetup:
C123:Nsub < Ni!
C124:Effective channel length less than zero!

degcalc.c:

BSIMDeltaVth:
C130:Degradation of transistor m<xx> too large!

output.c:

PlotSubCurrent:
C131:Insufficient memory space. Too many timesteps!
C132:Timestep too small to plot substrate current!
C133:Substrate current too large to plot!

PrintSubCurrent:
C134:Insufficient memory space. Too many timesteps!
C135:Timestep too small to print substrate current!

PlotGateCurrent:
C136:Insufficient memory space. Too many timesteps!
C137:Timestep too small to plot gate current!
C138:Gate current too large to plot!

PrintGateCurrent:
C139:Insufficient memory space. Too many timesteps!
C140:Timestep too small to print gate current!

postmisc.c:

OpenRaw:
C141:Cannot open the rawsub file!

OpenFile:

C142:Empty <*filename*> file!

bsim2.c:

BSIM2setup:

C144:BSIM2 MOSFET m<*transistor name*>, model <*process name*>: Effective channel length <= 0!

C145:BSIM2 MOSFET m<*transistor name*>, model <*process name*>: Effective channel width <= 0!

BERT Shell Script Errors:

ageconv.c:

main:

B01: Cannot open agetable!

B02: Cannot open rawagetable!

B03: Cannot create rawtempage file!

agefilt.c:

main:

B04: Cannot open SPICE input file!

B05: Cannot create SPICE input file!

B06: Temporary file is missing!

B07: Cannot open temporary file!

B20: Not enough memory!

convinp.c:

main:

B08: Cannot open input file!

B09: Cannot open inpdeck file!

B10: Cannot create intermediate file!

B11: Not enough memory!

B12: Improper age given in .age command!

B23: Input file has no transistor!

ChangeModelName:

B13: Model parameter file not found!

B14: Cannot write into directory!

B25: Not enough memory!

PrintFreshProcess:

B24: Not enough memory!

copyproc.c:

main:

B15: Cannot open input file!

B16: Cannot create intermediate file!

B17: Not enough memory!

B18: Premature end-of-file in input deck!

delproc.c:

main:

B19: Cannot open 'rawpfile'!

mainmisc.c:

NextWord:

B21: Not enough memory!

NextWord2:

B22: Not enough memory!

CORS Pre-processor Errors:

T01: Cannot open rawtddb file!
tddb.c
This error usually occurs when the user who wishes to do a burn-in simulation accidently deletes file rawtddb between the first and second CORS runs.

T02: Cannot open rawtddb file!
tddb.c
User's computer will not allow CORS to open a new file for writing.

T03: Cannot open rawinp2 file!
tddb.c
User's computer will not allow CORS to open a new file for writing.

T04: Not enough time-to-breakdown values on .TTF card!
tddb.c
CORS requires that zero or eight time-to-breakdown values be specified on a .TTF card.

T05: Burnin card formated incorrectly.
tddb.c
CORS found a keyword on a .BURNIN card that was neither TIME nor TEMP.

T06: No .TRAN card. Cannot do tddb analysis!
tddb.c
CORS did not find a .TRAN card in the input deck. This SPICE command must be specified for CORS or CAS to run.

T07: Insufficient memory space. Reduce the number of MOS models!
tddb.c
User's file system ran out of memory during execution of pre-processor. The user should try to free additional memory space. Simulation of a smaller circuit may be feasible.

T08: Not enough arguments on a .XEFF card!
tddb.c

T09: Not enough arguments on a .XEFF card!
tddb.c

T10: .XEFF card formated incorrectly!
tddb.c
CORS did not find keyword FILENAME in its expected position.

T11: .XEFF card formated incorrectly!
tddb.c
CORS did not find a user provided file name after keyword FILENAME.

T12: .XEFF card formated incorrectly!
tddb.c
CORS did not find a "=" after keyword FILENAME.

T13: .XEFF card formated incorrectly!
tddb.c
CORS did not find a user provided file name after keyword FILENAME.

T14: Insufficient memory space. Reduce the number of transistors.
tddb.c
See T07.

T15: No model name specified on a MOSFET definition card!
tddb.c

T16: Capacitor card in deck not formated correctly.
tddb.c
Some capacitor definition card does not contain the minimum number of parameters required by SPICE.

T17: Capacitor card in deck not formated correctly.
tddb.c
Keyword TBDMODEL may be spelled incorrectly.

T18: Insufficient memory space. Reduce # of tddb capacitors.
tddb.c
See T07.

T19: Missing a .altmodel/.model card for model *<cap model name>*
tddb.c

T20: A tddb data file can not be found!
tddb.c
CORS can not find a file which was named on .XEFF card. Spelling errors are a common cause of this problem or the user may have the file stored in a directory different from the one CORS is searching.

T21: A tddb data file can not be found!
tddb.c
See T20.

T22: A tddb data file can not be found!
tddb.c
See T20.

T23: Insufficient memory space. Reduce the number of transistors.
tddb.c
See T07.

T24: Insufficient memory space. Reduce the number of capacitors.
tddb.c
See T07.

T25: Missing an .XEFF card for model *<mos model name>*
tddb.c

T26: Missing a .altmodel/.model card for model *<mos model name>*
tddb.c

T27: Missing an .XEFF card for model *<cap model name>*
tddb.c

T28: Insufficient memory space.
tddb.c

T29: Insufficient memory space.
tddb.c

T30: Not enough memory for timebd !
preirsim.c

T31: Not enough memory for lsinum!
preirsim.c

T32: Incorrect format of STARTTIME card !
preirsim.c

T33: Incorrect format of ENDTIME card !
preirsim.c

T34: Missing '=' in CONSTHI card !
preirsim.c

T35: Missing '=' in CONSTLO card !
preirsim.c

T36: Not enough memory for tmpconst !
preirsim.c

T37: Missing '=' in Vol card !
preirsim.c

T38: Missing '=' in Voh card !
preirsim.c

T39: Missing '=' in Vdd card !
preirsim.c

T40: ENDTIME card not found, Run abort !
preirsim.c

T41: Node <*node name*> in both CONSTHI and CONSTLO card !
preirsim.c

T42: Cannot open rawsub for writing !
preirsim.c

T43: The .XEFF card for <*transistor type*> type transistor not found !
preirsim.c

T44: Illegal type name for IRSIM run !
preirsim.c

T45: Cannot open <*filename*> !
preirsim.c

T46: Cannot open <*filename*> for writing !
preirsim.c

T47: No .XEFF card was found in rawinp1 file !
tddb.c

CORS Post-processor Errors:

T101: Can not open file rawtddb!
postddb.c
User may have deleted file rawtddb between execution of pre- and post-processor or maybe never ran the pre-processor.

T102: Insufficient memory space. Reduce the number of transistors.
postddb.c
User's file system ran out of memory during execution of post-processor. The user should try to free additional memory space. Simulation of a smaller circuit may be feasible.

T103: Insufficient memory space. Reduce the number of models.
postddb.c
See T102.

T104: Insufficient memory space. Reduce the number of capacitors.
postddb.c
See T102.

T105: Insufficient memory space. Reduce the number of MOS devices.
postddb.c
See T102.

T106: Cannot open file rawburn!!!
postddb.c
There are two causes of this error. The user may have deleted file rawburn between runs 1 and 2 of a burn-in simulation. Alternately, CORS may have detected a rawtddb file from a previous failed run (this file is automatically deleted after successful runs). CORS now assumes it is doing pass 2 of a burn-in simulation. The solution is to delete all the raw**** files and retry CORS.

T107: Rawtddb not formated correctly!
postddb.c

T108: Cannot open file rawburn for writing.
postddb.c

User's computer will not allow CORS to open a new file for writing.

T109: Can not open rawout1 file!
postddb.c
CORS can not find file rawout1 which should have been created in postbert.c (CAS).

T110: Can not create rawout2 file.
postddb.c
See T108.

T111: Can not create outtddb file.
postddb.c
See T108.

T112: Voltage node printout for tddb analysis not found
postddb.c
Header in SPICE output can not be located. User may be using a version of SPICE other than those CORS was developed for.

T113: Division by zero in reading times.
postddb.c
Step size specified on .TRAN card is so small that a divide-by-zero error is anticipated.

T114: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T115: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T116: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T117: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T118: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T119: Voltage node printout for tddb analysis not found.
postddb.c
See T112.

T120: Can not open a defect density data file!
postddb.c
See T113.

T121: Timesteps too small in reading voltages.
postddb.c
Try a larger step-size on .TRAN card.

T122: Voltage printout for TDDB analysis not found.
postddb.c
See T112.

T123: Voltage printout for TDDB analysis not found.
postddb.c

See T112.

T124: Voltage printout for TDDB analysis not found.
postddb.c
See T112.

T125: Can not open a defect density data file!
postddb.c
See T113.

T126: Voltage node printout for tddb analysis not found.
postddb.c
See T112.

T127: Can not open a defect density data file!
postddb.c
See T113.

T128: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T129: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T130: Insufficient memory space. Reduce the number of timesteps.
postddb.c
See T102.

T131: Voltage node printout for tddb analysis not found.
postddb.c
See T112.

T132: Could not find model name.
postddb.c

T133: Could not find model name.
postddb.c

T134: Can not open a defect density data file!
postddb.c
CORS can not find one of the file names originally listed on a .XEFF card with keyword FILENAME.

T135: Can not open a defect density data file!
postddb.c
CORS can not find one of the filenames originally listed on a .XEFF card with keyword DIFFEDGE.

T136: Can not open a defect density data file!
postddb.c
CORS can not find one of the filenames originally listed on a .XEFF card with keyword OXEDGE.

T137: Can not open file rawout1 for reading.
postddb.c

T138: Can not open file rawout2 for writing.
postddb.c

T139: Can not open file <*sim filename*>
postddb.c

T142: Insufficient memory space. Try reducing number of devices.
postddb.c

T143: Insufficient memory space. Try reducing number of devices.
postddb.c

T144: Insufficient memory space. Try reducing number of devices.
postddb.c

T145: Insufficient memory space. Try reducing number of devices.
postddb.c

T146: Insufficient memory space. Try reducing number of devices.
postddb.c

T147: Insufficient memory space. Try reducing number of devices.
postddb.c

T148: For duration of simulation, drain value is 'X' for transistor <*transistor name*>
postddb.c

T149: For duration of simulation, gate value is 'X' for transistor <*transistor name*>
postddb.c

T150: For duration of simulation, source value is 'X' for transistor <*transistor name*>
postddb.c

T151: For all times less than endtime, gate value is 'X' for transistor <*transistor name*>
postddb.c

T152: For all times less than endtime, drain value is 'X' for transistor <*transistor name*>
postddb.c

T153: For all times less than endtime, source value is 'X' for transistor <*transistor name*>
postddb.c

T154: The number of .XEFF card is 0. Check rawtddb file.
postddb.c

T155: Insufficient memory space.
postddb.c

T201: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations had not converged.

T202: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations had not converged.

T203: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations had not converged.

T204: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations

had not converged.

T205: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations had not converged.

T206: N-R calcs for tddb did not converge.
nrcalcs.c
CORS solves for Xeff iteratively, using Newton-Raphson's method. The calculations had not converged.

T300: Cannot open rawout1 file!
consort.c
Consort.c could not open the rawout1 file which contains the output of the Irsim Simulator

T301: Cannot create tmp file!
consort.c
A temporary file used by consort.c could not be opened.

T302: Cannot open tmp file!
consort.c
Consort.c could not open a tmp file that was previously created.

T303: Cannot create filterout file!
consort.c
Consort.c could not create the filterout file which passes its results to the main postbert routine. method. The calculations had not converged after 100 iterations.

T304: Cannot find node: *<nodename>*.
consort.c
Consort.c has tried to look up a node name that doesn't exist.

EM Pre-processor Errors:

EM00: Subcircuit Error

EM01: Could not open input file in EMPreFilter
Input file is not available to prebert

EM10: Could not open input file
Could not open input file in EMSetup

EM20: Could not open user set up geometry file
Could not open user set up geometry file when **.EMSTAT** is specified

EM30: Spice Deck error at element *xxx*
User has set up SPICE deck incorrectly at element *xxx*

EM31: Exceeding MAXXNODE at element *xxx*
Pre-processor can not handle a subcircuit card with more than MAXXNODE, recompile with higher value of MAXXNODE

EM40: Could not write to temporary geometry file
Pre-processor fails to open a temporary geometry file for writing

EM41: Could not read geometry file
Pre-processor fails to open a temporary geometry file for reading

EM50: Error in user set up geometry file at element *xxx*
User makes an error in setting up the geometry file at element *xxx*

EM60: Node not found in node file

EM61: Could not open rulefile

EM62: There is no interconnect file

EM63: Could not open interconnect file

EM64: Must have input file in command line

EM65: Error in command line option

EM66: Insufficient memory space

EM Post-processor Errors:

EM101: Could not open EM design rule file
Could not open design rule file in EMPost

EM102: Could not open geometry file
Could not open geometry file in EMPost

EM103: Could not open input file
Could not open input file in EMPost

EM104: Could not write to temporary output file
Could not write to temporary output file in EMPost

EM105: Could not write to simulator output file
Could not write to simulator output file in EMPost

EM106: Could not write to ciffile
Could not write to ciffile file in EMPost

EM107: Could not write to rate statfile
Could not write to rate statfile in EMPost

EM108: Could not write to percent statfile
Could not write to percent statfile in EMPost

EM109: Could not write to current table file
Could not write to current table file in EMPost

EM120: Exceeding MAXXNODE at element
Post-processor can not handle a subcircuit card with more than MAXXNODE, recompile with higher value of MAXXNODE

EM130: Node *x* not found in geometry file (LookUpNode)
This is unusual because the geometry file is set up by the pre-processor. Check to see if the geometry file is the right one.

EM140: Error in EM design rule file in line *x* at field *y*
User has made an error in EM design rule file in line *x* at field *y*.

EM141: No one complete parameters set in EM design rule file
No one complete parameter set is found in the rule file

EM142: Error in EM design rule file. Inconsistency between the definition of the m and Adc field(s)
Check the emrule file for an inconsistency between the definition of the m and Adc field(s)

EM150: Node *x* not found in geometry file (Failstat)
This is unusual because the geometry file is set up by the pre-processor. Check to see if the geometry file is the right one.

EM151: Error in geometry file at *xxx* (Failstat)
This is unusual because the geometry file is set up by the pre-processor. Check to see if the geometry file is the right one.

EM160: Error in command line

EM161: Error in command line option T

EM162: Error in command line option

EM163: Node not found in geometry file

EM164: Insufficient memory space

BiCAS Pre-processor Errors:

Bi01: Cannot open rawbjt file!
This error happens when there is not enough disk space for writing a new file or some other disk I/O problems.

Bi02: Cannot open rawinp2 file!
This error happens when there is not enough disk space for writing a new file or some other disk I/O problems.

Bi03: Incorrect format for the future time given in the .agebjt command!
Wrong format of age given in .agebjt card.

Bi04: No future time given for the .agebjt command!
BERT cannot find the the time argument in the .agebjt card. Maybe cause by a missing "+".

Bi05: No future time given for the AGEDIB command!
BERT cannot find the the time argument in the .agedib card. Maybe cause by a missing "+".

Bi06: Incorrect format for the future time in the AGEDIB command!
Wrong format of age given in .agedib card.

Bi07: Invalid .printirev or .plotirev command!
The argument of .printirev or .plotirev has to be the names of bipolar transistors starting with a "Q" or "q".

Bi08: Missing .ageprocbjt command in the input deck!
When the file *agetableBJT* is found in the current directory, BERT assume this is a pre-bert run to age the transistors and generate the aged input deck *inpdeck*. An **.AGEPROCBJT** card is required to age the transistor. This error may be caused by some *left-over* agetableBJT from previous BERT run.

Bi09: Cannot open rawinp1 file!
This error may be caused by the disk I/O error.

Bi10: Insufficient memory space. Reduce the number of model parameter files!
Not enough memory (RAM) for parameters. Simulation of a smaller circuit should be feasible.

Bi11: Incorrect .bjtproc command format!
The first argument should be the process name. The second argument should be the

reverse current file. The third (last) argument should be the process/parameter file name. For example : .BJTPROC *proc_name rev_file* filename=*proc_file*.

Bi12: Too many model parameter files!
You have more than 100000 parameters files. The upper limit is 100000 files

Bi13: No model parameter file(s) specified!
You may have forgot to put in **.bjtproc** cards.

Bi14: Missing or incorrect model parameter filename specified!
BERT cannot match each transistor to their specified process file.

Bi15: Insufficient memory space. Too many transistors!
Not enough memory (RAM) to store transistor data. Simulation of a smaller circuit should be feasible.

Bi16: Cannot open the *rwmd*n rawmodel file!
BERT cannot find the *rwmd*n file in the current directory. **n** is an integer. *rwmd*n contains the information from the process file.

Bi17: Incorrect BJT card format !
Your bipolar transistor statement is in the wrong format. It should look like:
Q1 1 2 3 4 bjt1
The bulk node (4) can be omitted.

Bi18: Bipolar process *PROC_NAME* not defined !
BERT cannot find the process *PROC_NAME* in your **.bjtproc** statements.

Bi19: Cannot open model parameter file *PROC_FILENAME*!
PROC_FILENAME does not exist in your current directory.

Bi20: Cannot write into temporary model parameter file rwmdn!
n is an integer. This error happens when there are disk I/O error, such as not enough disk space.

Bi21: NON-Spice model parameters in the specified model parameter file!
One or more of your process files specified in **.BJTPROC** cards have missed a **.model** keyword.

Bi22: No current given for the DELTAIB command!
BERT cannot find the current argument in the **.DELTAIB** card.

Bi23: Incorrect format of current in DELTAIB command!
BERT doesn't recognize the floating point format of the current argument in the **.DELTAIB** card.

Bi24: Non BJT transistor in .DEGPRINTBJT card!
BERT finds devices other than bipolar transistors in the **.DEGPRINTBJT** card. Maybe a typo.

Bi25: Empty process file!
One or more of your process files is empty or only contain comment.

Bi26: Insufficient memory space!
Memory allocation for malloc or calloc has failed.

BiCAS Post-processor Errors:

Bi100: Cannot open agetableBJT!
This error happens when your disk is full or there is writing problem on your disk.

Bi101: Insufficient memory space!
Not enough memory (RAM) to run postbert. Simulation of a smaller circuit should be feasible.

Bi102: Timestep too small in reading voltage values!
This error happens when your SPICE timestep is smaller than 10^{-38} sec, which is highly unlikely.

Bi103: Insufficient memory space. Too many timesteps!
Not enough memory to hold SPICE output. The user should increase the size of timestep and/or simulate a smaller circuit.

Bi104: Voltage printout for reverse current analysis not found!
BERT cannot find the SPICE voltage output for the reverse current analysis.

Bi105: Insufficient memory. Too many Irev requested transistors!
Not enough memory (RAM) to compute all reverse current. The user can either reduce the size of the circuit, increase the timestep size, or reduce the number of I_{rev} requested. The goal is to free up more memory.

Bi106: Insufficient memory space. Too many transistors!
Not enough memory (RAM). The user can either reduce the size of the circuit, increase the timestep size, or reduce the number of I_{rev} requested. The goal is to free up more memory.

Bi107: Cannot open rwmdn file!
n is a integer. BERT cannot find the required rwmdn parameter file. They may have been accidentally removed. You can run prebert again to generate the rwmdn file.

Bi109: Timestep too small to plot reverse current!
This error happens when your SPICE timestep is smaller than 10^{-38} sec, which is highly unlikely. You can increase the timestep size to avoid this error.

Bi110: Reverse current too large to plot!
This error happens when the computed I_{rev} is larger than 10^{35} Amp, which is highly unlikely.

Bi111: Timestep too small to print reverse current!
This error happens when your SPICE timestep is smaller than 10^{-38} sec, which is highly unlikely. You can increase the timestep size to avoid this error.

Bi112: Cannot open the rawbjt file!
BERT cannot find rawbjt in the current directory. rawbjt may have been accidentally removed. You can run prebert again to generate rawbjt.

Bi113: Cannot open reverse current file : **REV_FILE** !!
BERT cannot find **REV_FILE** in the current directory. You may have misspelled the **REV_FILE** name, or **REV_FILE** has been accidentally deleted.

Subcircuit Pre-processor Errors:

EX01: Cannot open file rawinp1 for reading !!
EX02: Missing .end card in file rawinp1 !!
EX03: Not enough memory !!
EX04: Term is longer than 30 !!
EX05: Missing subckt name in .subckt card !!
EX06: Recursive subckt call found !!
EX07: Missing .ends card !!

EX08: Incorrect subckt call format !!
EX09: Too many args after .ends !!
EX10: Mismatch subckt name in .ends card !!
EX11: Cannot open file rawinp2 for writing !!
EX12: Incorrect number of arg in subckt call <*subckt call name*> !!
EX13: More than 2000 local node in subckt def !!
EX14: More than 2000 local device in subckt def !!
EX15: Incorrect number of node in <*dev name*> !!
EX16: Incorrect number of device in <*dev name*> !!
EX17: Unknown subckt def <*subckt def name*> found !!

EX18: Cannot open output file !!
EX19: Empty input file <filename> !!
EX20: Cannot open file <filename> !!
EX21: Cannot open file for reading/writing/appending !!

If you are missing a *.ends* card, you may get *EX07 or EX10*. If you get *EX17* and you are sure that you have defined that subckt definition, you may have misspelled the key word *.subckt*.

Appendix C

CORS Simulations

This section provides the user with examples of the various kinds of studies which may performed with CORS. All simulations were performed for CMOS circuits operating at 5.5V and 125C, unless otherwise stated.

Figure C1 shows simulated reliability of 10K gate array after burn-in. (Each cell in the gate array is identical to the one described by the input deck listed in Example 4.6.) Power supply voltage during burn-in was 7 volts. These studies are useful for balancing conflicting goals such as cost, avoidance of hot electron degradation and reduction of field failures due to oxide breakdown in choosing the burn-in condition for a product.

CORS has an option for printing out the breakdown probability for individual devices. This is illustrated in Figure C2.

Figure C3 shows the effect of operating temperature on the reliability of a SRAM cell array. The results shown are for a circuit in which every memory cell is undergoing continuous read operations. Previous simulations were performed for a SRAM cell in the idle state as well as one undergoing continuous precharge and read operations. Both cells have nearly identical failure statistics. The simulator showed that under each of these conditions the same two transistors in the SRAM cell dominate circuit breakdown. The oxide field across those two transistors does not change appreciably during a read operation.

Figure C4 shows the dramatic effect of oxide quality on the expected circuit lifetime of a 10K gate array (the input deck for this simulation was similar to the one listed in Example 4.6). Figure C5 shows that the inclusion of edge defects can change the predicted lifetime depending on the relative density and severity of the edge defects (circuit simulated was the same as in Figure C4). If the reader is unfamiliar with the variable X_{eff} plotted in Figure C5, he/she can find a discussion of it in Section 4.1.

Lifetime after Burn-In

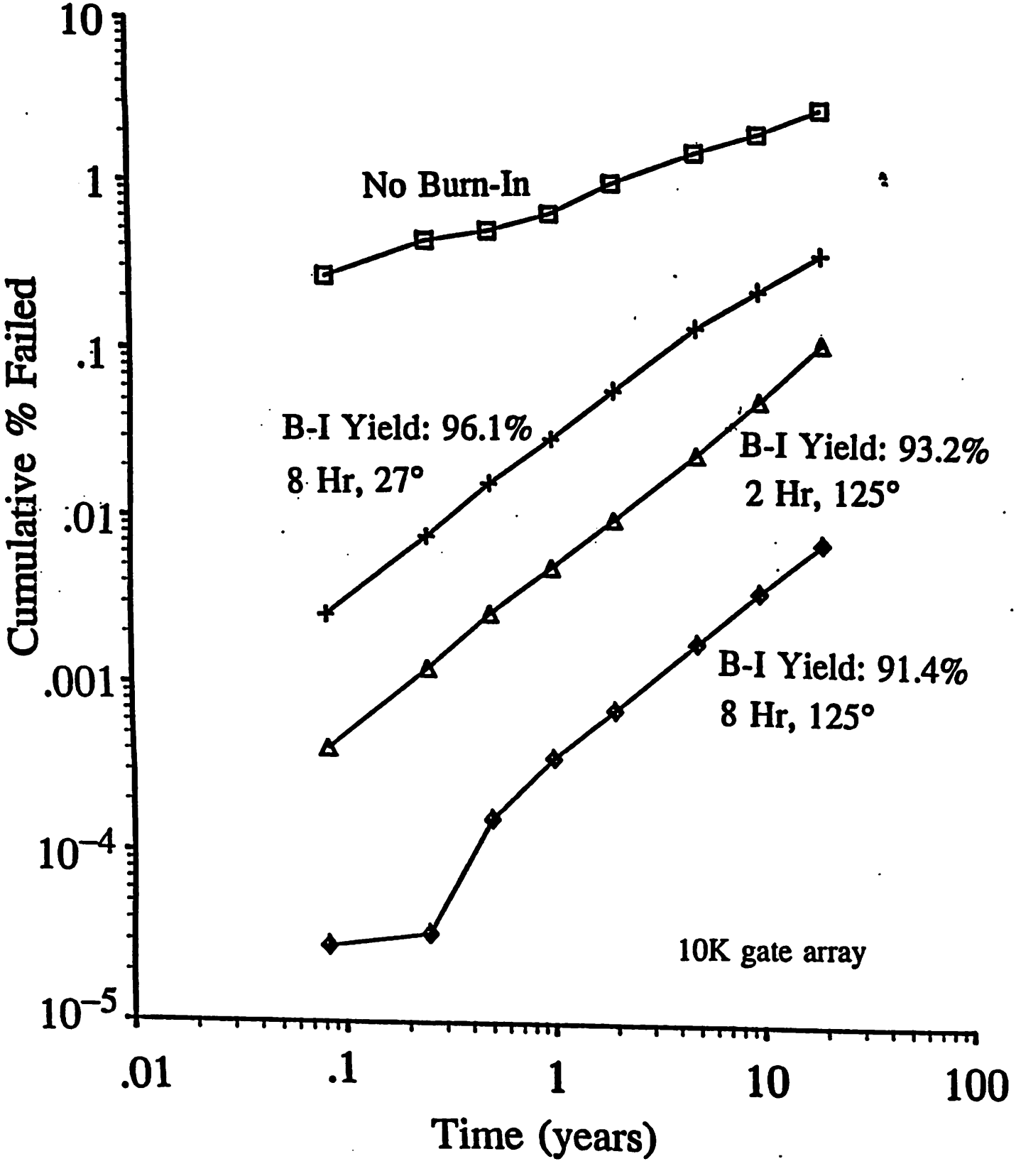


Fig. C1

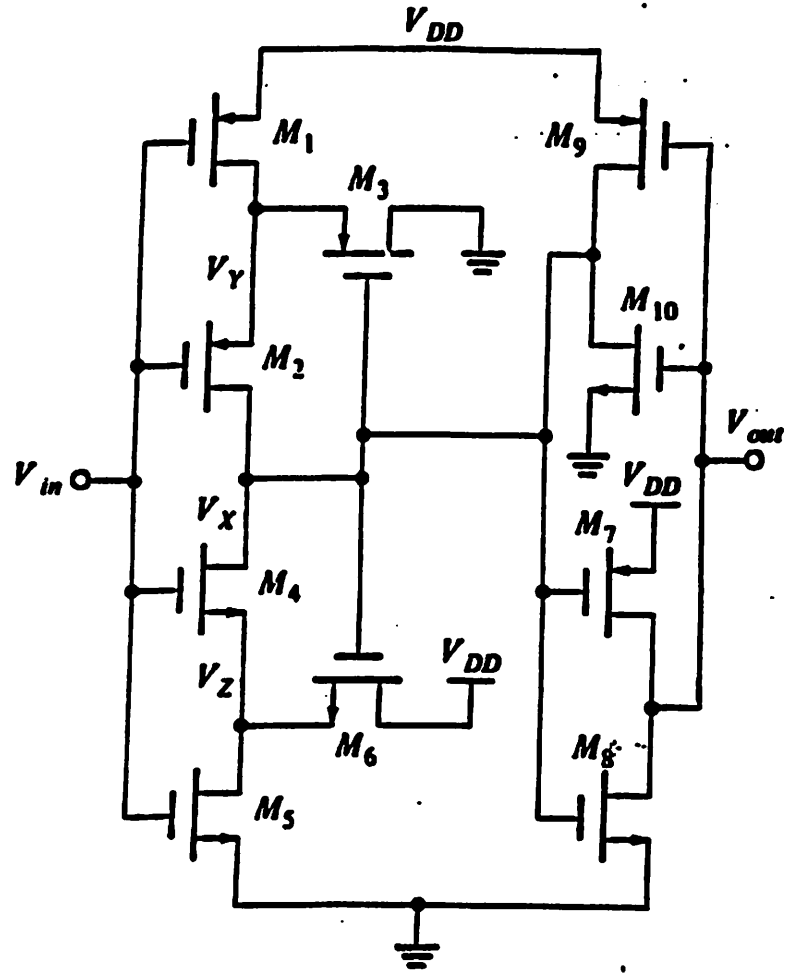
Device Failure Probabilities

$$\left(\frac{W}{L}\right)_{4,5,8} = \frac{10}{1}$$

$$\left(\frac{W}{L}\right)_{6,10} = \frac{1}{1}$$

$$\left(\frac{W}{L}\right)_{1,2,7} = \frac{20}{1}$$

$$\left(\frac{W}{L}\right)_{3,9} = \frac{2}{1}$$



Schmitt Trigger

cumulative % circuits f

	CF% at 10 yrs
M1	1.08×10^{-8}
M2	1.11×10^{-8}
M3	1.43×10^{-9}
M4	5.10×10^{-9}
M5	5.40×10^{-9}
M6	3.53×10^{-13}
M7	1.43×10^{-8}
M8	1.23×10^{-8}
M9	2.80×10^{-9}
M10	1.23×10^{-9}

Fig. C2

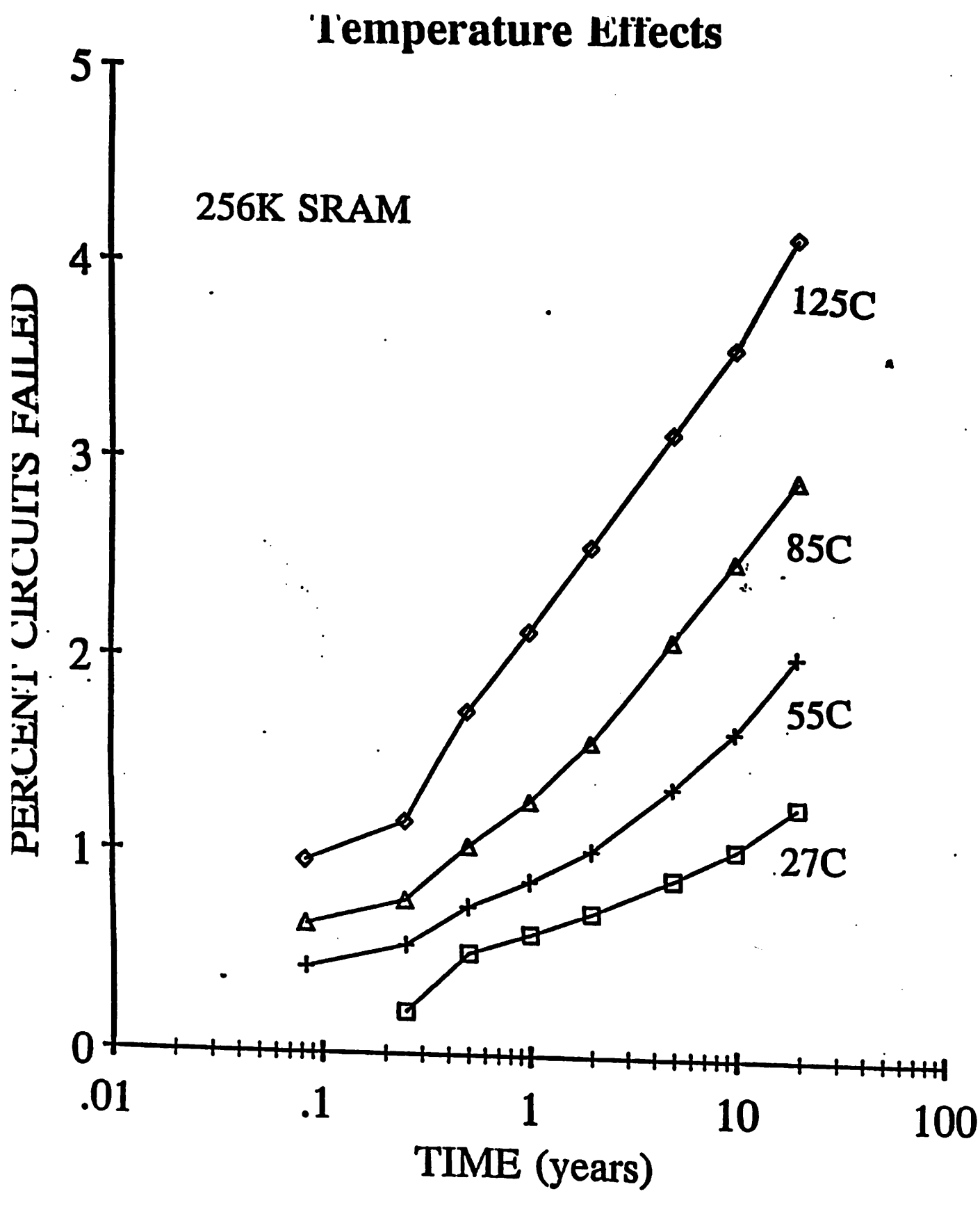
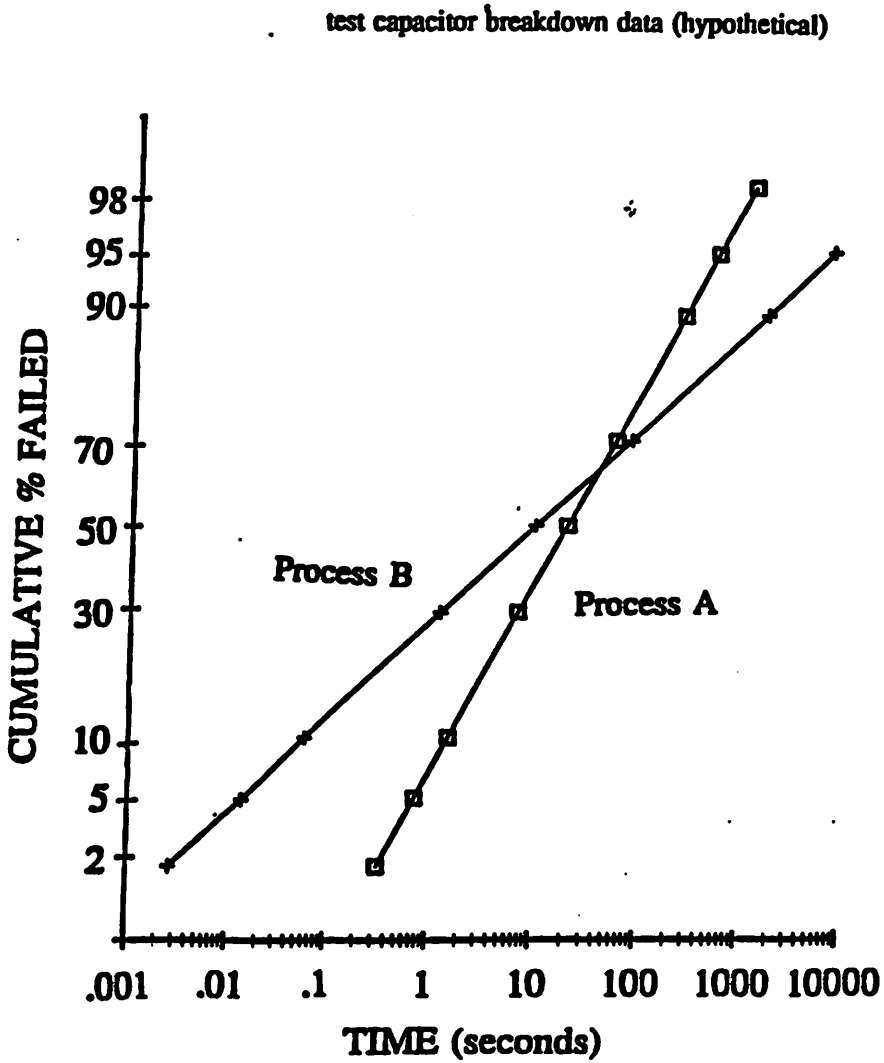

Temperature Effects
256K SRAM
PERCENT CIRCUITS FAILED
0
1
2
3
4
5
125C
85C
55C
27C
.01
.1
1
10
100
TIME (years)

Fig. C3

Effect of Oxide Quality on Lifetime



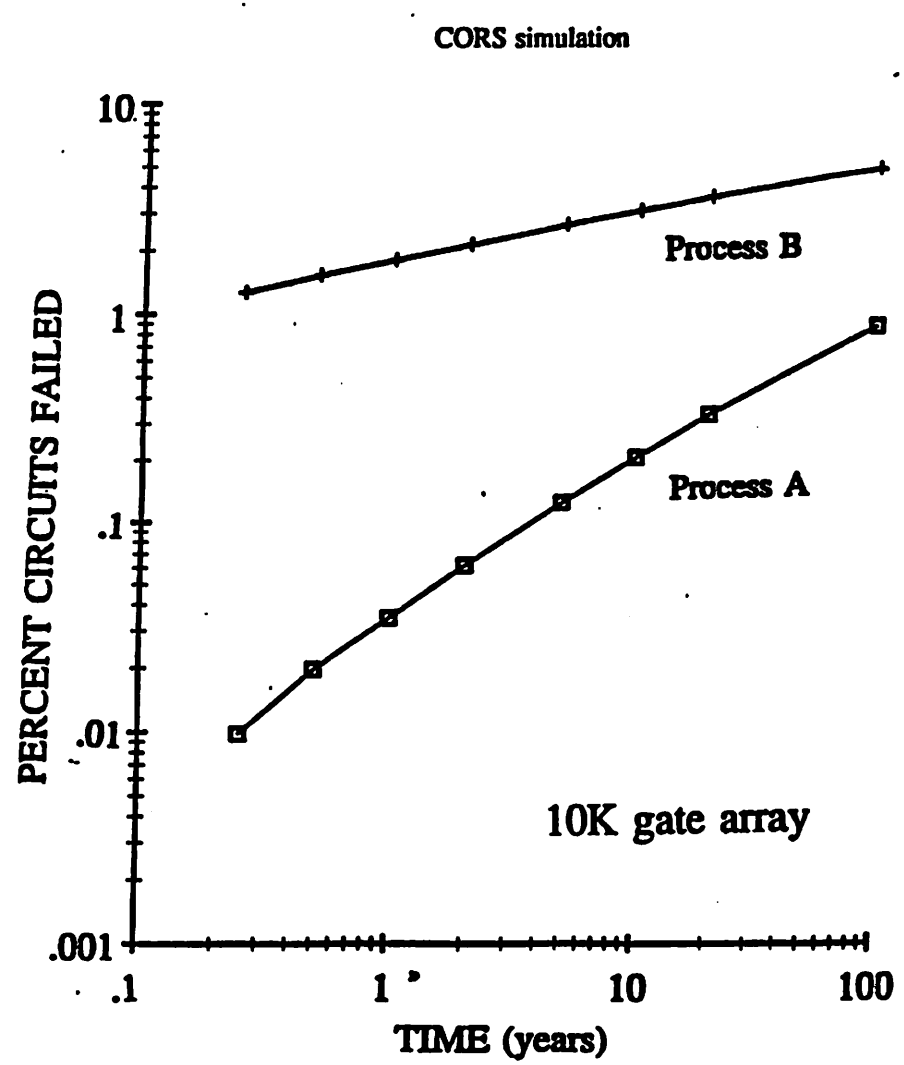
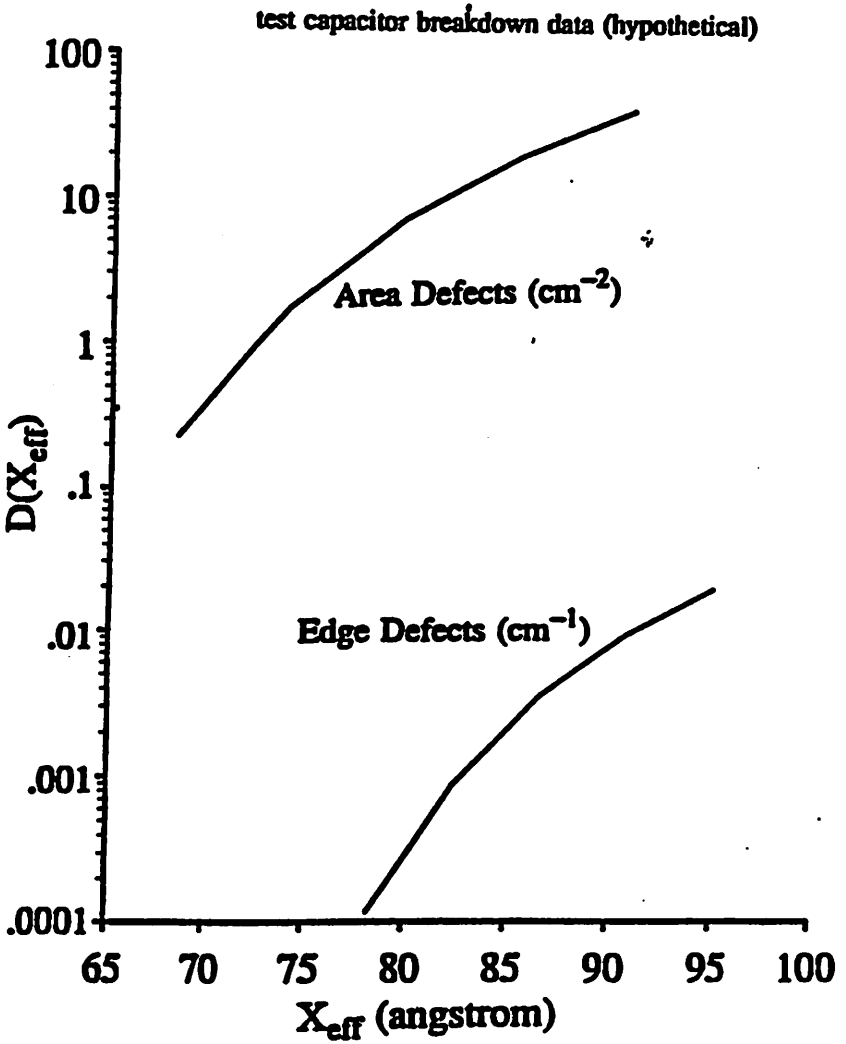


Fig. C4

Effect of Device Edge Defects



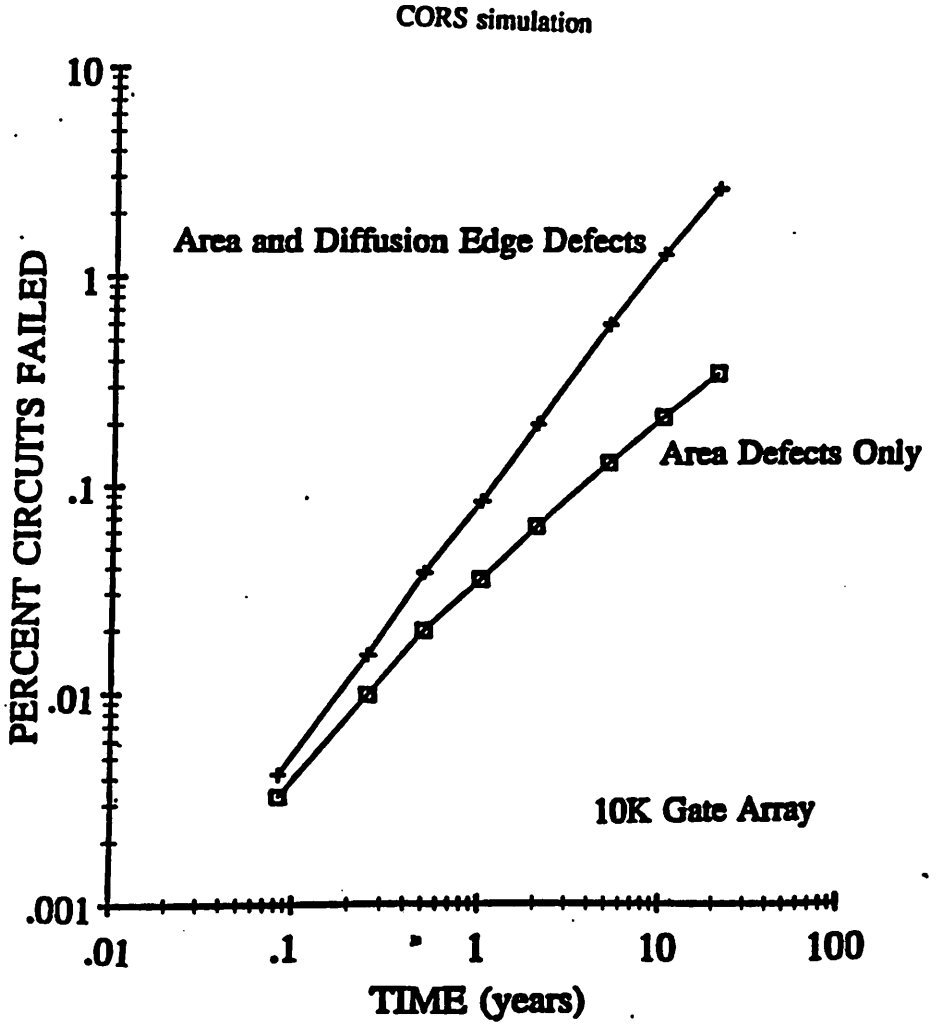


Fig. C5

Appendix D

Lognormal Distribution and Length Dependence Model

The following example illustrates the delima in choosing a lognormal distribution while using the length dependence model. The experimental data (indicated by closed circles) for a long test line ($2\times10^4\mu$) is given in Figure D. The failure probability for this line can be described by a lognormal distribution function F(t) with σ=1.0 and MTF=1000. From this data, the simulator can construct a failure distribution for a line half the length of the test line, i.e. at any time t the $1\times10^4\mu$ line has a failure probability $G_2(t)$ equal to:

$$G_2(t) = 1 - [1 - F(t)]^{1/2}$$

For example at time t = 1000, the failure probability for this line is 0.29. Similarly, the failure distribution of lines 1/3 of the long test line $G_3(t)$ can be constructed. In general, for a line that is 1/x of the long test line, $G_x(t)$ is:

$$G_x(t) = 1 - [1 - F(t)]^{1/x}$$

Figure E shows the plot of $G_2(t)$, $G_3(t)$, $G_5(t)$ and $G_{10}(t)$ on lognormal paper. We can see that as the lines become shorter, the extrapolated failure distribution deviates more from the lognormal function.

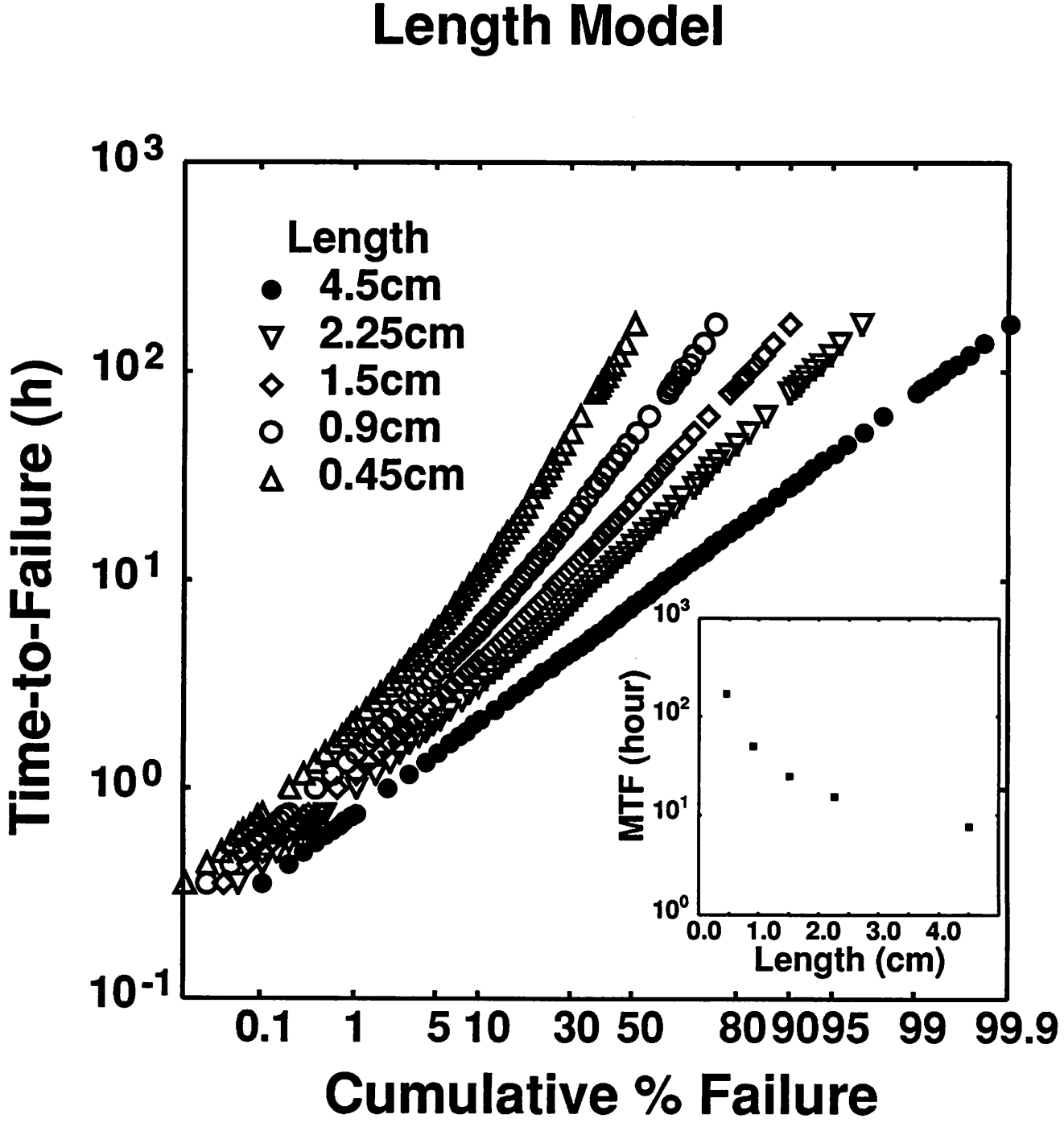


Figure D. The failure data from the 4.5 cm long interconnect is assumed to be lognormally distributed. The plot shows deviation from the lognormal distribution when failure statistics for shorter lines are calculated from the length dependence model. The calculated MTF versus length is plotted in the inset.

Appendix E

Width Dependence Parameters

The simulator accepts three parameters defining the dependence of interconnect time-to-failure on width: A_w, B_w and C_w which are entered using **WIDTH_A**, **WIDTH_B** and **WIDTH_C** in the rule file. B_w is the linewidth where the test data shows an increase in TTF. The parameters are defined by Eqs.(5.3a) and (5.3b) as shown in section 3.4.3(o,p,q) of Appendix A. The equations are repeated as follows:-
for $W \geq B_w$:

$$TTF(W) = A_w \times (W-B_w)^2 + D_w \qquad (5.3a)$$

for $W < B_w$:

$$TTF(W) = C_w \times (W-B_w)^2 + D_w \qquad (5.3b)$$

We will demonstrate the extraction of the three parameters:

From the fit shown in Figure E, the width dependence is:-
for $W > 1.0\mu$

$$MTF(W) = 6.25 \times (W-1.0)^2 + 7.5$$

and for $W < 1.0\mu$

$$MTF(W) = 40.0 \times (W-1.0)^2 + 7.5$$

where $A_w = 6.25$, $B_w = 1.0$, $C_w = 40.0$, and $D_w = 7.5$.

The parameters are entered in the rule file as shown below:
metal1 length=4.5e4 width=1.0 thick=0.5
log_median=7.5 log_sigma=1.0
width_a=6.25 width_b=1.0 width_c=40.0

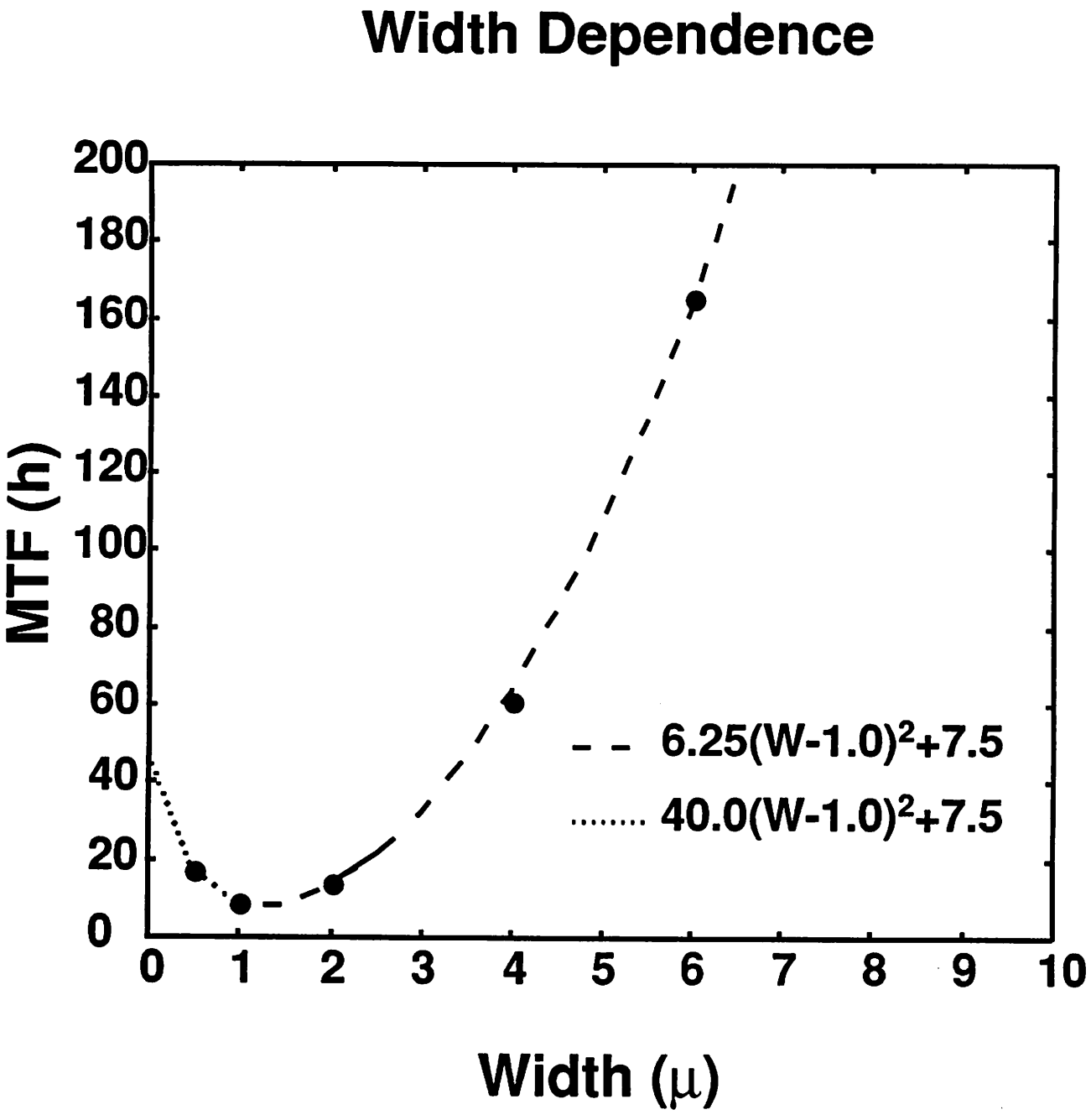


Figure E. Extracting width dependence parameters for the simulator

Appendix F

CIF Layer Names in the Layout Extractor

The technologies known to **mextra** are: nMOS ("nmos"), MOSIS P well CMOS/Bulk, also known as CBPM ("cmos-pw" or "cmos-p"), MOSIS Scalable CMOS/Bulk N-well, also known as SCN ("cmos-nw" or "cmos-n"), MOSIS Scalable CMOS/Bulk P-well, also known as SCP ("cmos-s" or "scmos"), and MOSIS Scalable CMOS/Bulk Generic, also known as SCG ("cmos-g"). The CIF layer names for each technology are listed in the two tables below:

Technology					
nmos		cmos-p/cmos-pw		cmos-n/cmos-nw	
cif layer	mextra/cif2ps internal names	cif layer	mextra/cif2ps internal names	cif layer	mextra/cif2ps internal names
NM	METAL	CW	WELL	CWN	WELL
NP	POLY	CM	METAL	CMS	METAL2
ND	DIFF	CM2	METAL2	CMF	METAL
NC	CUT	CP	POLY	CPG	POLY
NB	BURIED	CP2	POLY2	CAA	DIFF
NI	ION	CD	DIFF	CVA	CUT2
NG	GLASS	CC	CUT	CCA	CUT
		CC2	CUT2	CSP	CION
		CS	CION	CSN	GLASS
		CG	GLASS	COG	GLASS
				XP	GLASS

Technology					
cmos-s/scmos		cmos-g		isocmos	
cif layer	mextra/cif2ps internal names	cif layer	mextra/cif2ps internal names	cif layer	mextra/cif2ps internal names
CWP	WELL	CWN	WELL	CPW	WELL
CMS	METAL2	CMS	METAL2	CM	METAL
CMF	METAL	CMF	METAL	CM2	METAL2
CPG	POLY	CPG	POLY	CP	POLY
CAA	DIFF	CAA	DIFF	CP2	NOP
CVA	CUT2	CVA	CUT2	CD	DIFF
CCA	CUT	CCA	CUT	CC	CUT
CCP	CUT	CCP	CUT	CC2	CUT2
CSP	CION	CSP	CION	CPP	CION
COG	GLASS	CWP	GLASS	CNP	NOP
XP	GLASS	CSN	GLASS	CG	GLASS
		COG	GLASS	CS	GLASS
		XP	GLASS		

Appendix G

Setting up an Alternative Technology for the Layout Extractor

The mask names that **mextra** uses in generating geometry information for the electromigration simulator are: **METAL** for metal-one, **METAL2** for metal-two, **CUT** for metal-one to poly or diffusion contact, **CUT2** for metal-one and metal-two via. To determine whether a transistor is PMOS or NMOS in a CMOS process, **mextra** uses: **POLY** for polysilicon, **DIFF** for diffusion (both p+ and n+), and **WELL** for either p-well or n-well. A PMOS transistor for example can be found when a polysilicon line (**POLY**) is found adjacent to the diffusion (**DIFF**) in the n-well (**WELL**) in a n-well CMOS process. For a NMOS process, the mask **ION** will be checked to see if the transistor has been subjected to a depletion implant. Buried contacts are found from **BURIED** mask.

User can create a set of new CIF layer names by using the **-L** *ciflayerfile* option in both **mextra** and **cif2ps**. The format for *ciflayerfile* is:

(1) **wellohmic** *type*
Example:

wellohmic P

type is one of **N**, **P** or **0** for n-well, p-well or nmos process.

(2) *internal_layer_name cif_layer_name*
Example:

METAL CM5

The *cif_layer_name* is associated with a **mextra/cif2ps** *internal_layer_name*.

One example of *ciflayerfile* is:

```
wellohmic N
WELL CW1
POLY CP2
DIFF CD3
CUT CC4
METAL CM5
CUT2 CV6
METAL2 CM7
GLASS CSN
GLASS COG
GLASS XP
```

Note that internal layers **GLASS** and **NOP** are dummy layers in **mextra** and **cif2ps**.
The command line:

```
>cif2ps -L cifnames fulladder.cif | lpr
```

will use the CIF layer names defined in **cifnames** to plot **fulladder.cif**.

Technology defined by user:

Technology name	
WellOhmic	
WELL	
METAL	
METAL2	
POLY	
POLY2	
DIFF	
CUT	
CUT2	
BURIED	
ION	
CION	
GLASS	
GLASS	
GLASS	

Appendix H

Manual Pages for mextra, sim2spice and cif2ps

The following manual pages for **mextra**, **sim2spice** and **cif2ps** are available on line.

NAME

mextra, valtbs – Manhattan circuit extractor for VLSI simulation

SYNOPSIS

mextra [**-t** *tech*] [**-g**] [**-d** *temp_dir*] [**-c** *cadrc_line*] [**-f** *cadrc_file*] [**-u** *scale*] [**-O**] [**-o**] [**-W**] [**-w**] *basename*

valtbs *basename*.**tbs**

DESCRIPTION

Mextra will read the file *basename*.**cif** and create a circuit description. From this circuit description various electrical checks can be done on your circuit. The circuit description is directly compatible with *esim*, *powest*, and *erc*. There are translation programs to convert *mextra* output to acceptable *spice* input (see *sim2spice*, *pspice* and *spcpp*).

Mextra creates several new files, *basename*.**log**, *basename*.**al**, *basename*.**sim** and *basename*.**nodes**. It also creates *basename*.**tbs** (if -w option is used) or *basename*.**ohm** (if no -w option). After *mextra* finishes it is a good idea to read the *.log* file. This contains general information about the extraction. It has a count of the number of transistors and the number of nodes, and it contains messages about possible errors. The *.al* file is a list of aliases which can be used by *esim*. The *.tbs* file is a list of transistors in the *.sim* file that contains the substrate/well node name for each transistor (as the 5th field - see sim file description below). This file can be used to manually create spice input for analog circuits, such as sense amps, and to check substrate/well connectivity (see *valtbs* below). The *.ohm* file is used by *ohmics*. The *.nodes* file is a list of node names and their CIF locations listed in CIF format. It can be read by *cifplot* to make a plot showing the circuit with the named nodes superimposed. The form of this cifplot command is:

cifplot *basename*.**nodes** *basename*.**cif**

The .sim file is the circuit description for use with simulation programs and electrical rule checkers.

Names

Mextra uses the CIF label construct to implement node names and attributes. The form of the CIF label command is as follows:

94 *name x y* [*layer*];

This command attaches the label to the mask geometry on the specified layer crossing the point (x, y). If no layer is present then any geometry crossing the point is given the label.

Mextra interprets these labels as node names. These names are used to describe the extracted circuit. When no name is given to a node, a number is assigned to the node. A label may contain any ASCII character except space, tab, newline, double quote, comma, semi-colon, and parenthesis. To avoid conflict with extractor generated names, names should not be numbers or end in '*#n*' where *n* is a number.

A problem arises when two nodes are given the same name although they are not connected electrically. Sometimes we want these nodes to have the same names, other times we don't. This frequently happens when a name is specified in a cell which is repeated many times. For instance, if we define a shift register cell with the input marked 'SR.in' then when we create an 8 bit shift register we could have 8 nodes names 'SR.in'. If this happens it would appear as though all 8 of the shift register cells were shorted together. To resolve this the extractor recognizes three different types of names: *local*, *global*, and *unspecified*. Any time a local name appears on more than one node it is appended with a unique suffix of the form '*#n*' where *n* is a number. The numbers are assigned in scanline order and starting at 0. In the shift register example, the names would be 'SR.in#0' through 'SR.in#7'. Global names do not have suffixes appended to them. Thus unconnected nodes with global names will appear connected after extraction. (The -g causes the extractor to append unique suffixes to unconnected nodes with the same global name.) Names are made local by ending them with a sharp sign, '#'. Names are global if they end with an exclamation mark, '!'. These terminating characters are not considered part of the name, however. Names which do not end with these characters are considered unspecified. Unspecified names are treated similar to locals. Multiple occurrences are appended with unique suffixes. By convention, unspecified names signify the designer's intention that this name is a local name, but is connected to only one node. It is illegal to have a

name that is declared two different types. The extractor will complain if this is so and make the name local.

It makes no difference to the extractor if the same name is attached to the same node several times. However, if more than one name is given to a node then the extractor must choose which name it will use. Whenever two names are given to the same node the extractor will assign the name with the highest type priority, global being the highest, unspecified next, local lowest. If the names are the same type then the extractor takes the shortest name. At the end of the .log file the extractor lists nodes with more than one name attached. These lines start with an equal sign and are readable by *esim* so that it will understand these aliases.

Attributes

In addition to naming nodes *mextra* allows you to attach attributes to nodes. There are two types of attributes, *node attributes*, and *transistor attributes*. A node attribute is attached to a node using the CIF 94 construct, in the same way that a node name is attached. The node attribute must end in an at-sign, '@'. More than one attribute may be attached to a node. *Mextra* does not interpret these attributes other than to eliminate duplicates. For each attribute attached to a node there appears a line in the .sim file in the following form:

A *node attribute*

Node is the node name, and *attribute* is the attribute attached to that node with the at-sign removed.

Transistor attributes can be attached to the gate, source, or drain of a transistor. Transistor attributes must end in a dollar sign, '$'. To attach an attribute to a transistor gate the label must be placed inside the transistor gate region. To attach an attribute to a source or drain of a transistor the label must be placed on the source or drain edge of a transistor. Transistor attributes are recorded in the transistor record in the .sim file.

Transistors

For each transistor found by the exractor a line is added to the .sim file. The form of the line is:

type gate source drain length width x y
g=*attributes* s=*attributes* d=*attributes*

Type can be one of three characters, 'e' for enhancement, 'd' for depletion, or 'u' for unusual implant. (Unusual implant refers to transistors which are only partially in an implanted area. It will be necessary to write a filter to replace these transistors with the appropriate model in terms of enhacement and depletion transistors.) *Gate*, *source*, and *drain* are the gate, source, and drain nodes of the transistors. *Length* and *width* are the channel length and width in CIF units. *X* and *y* are the x and y coordinates of the bottom left corner of the transistor. *Attributes* is a comma seperated list of attributes. If no attribute is present for the gate, source, or drain, the g=, s=, or d= fields may be omitted.

The extractor guesses the length and width of a transistor by knowing the area, perimeter, and length of diffusion terminals. For rectangular transistors and butting transistors the reported length and width is accurate. For transistors with corners or for unusually shaped transistors the length and width is not as accurate.

It is possible to design a transistor with three or more diffusion terminals. The extractor considers these as *funny transistors*. They are entered in the .sim file in the form:

f*type gate node1 node2 ... nodeN xloc*

The 'f' is followed by the type : 'e', 'd' or 'u'. *Node1 ... nodeN* are the diffusion terminal nodes. As with any circuit with 'u' transistors, any circuit with 'f' transistors must be run through a filter replacing each of the funny transistors with the appropriate model in terms of enhancement and depletion transistors.

Capacitance

The .sim file also has information about capacitance in the circuit. The lines containing capacitance information are of the form:

C *node1 node2 cap-value*

cap-value is the capacitance betweens a node and substrate is in femto-farads. Capacitance values below a certain threshold are not reported. The default threshold is 50 femto-farads.

Transistor capacitances are not included since most of the tools that work on the .sim file calculate them from the width and length information.

The capacitance for each layer is calculated separately. The reported node capacitance is the total of the layer capacitances of the node. The layer capacitance is calculated by taking the area of a node on that layer and multiplying it by a constant. This is added to the product of the perimeter and a constant. The default constants are given below. Area constants are in femto-farads per square micron. Perimeter constants are femto-farads per micron.

Layer	Area	Perimeter
metal	0.03	0.0
metal2	0.015	0.0
poly	0.05	0.0
diff	0.10	0.1
poly/diff	0.40	0.0

Poly/diffusion capacitance is calculated similar to layer capacitance. The area is multiplied by constant and this is added to the perimeter multiplied by a constant. Poly/diffusion capacitance is not threshold, however.

The –o and the –O options are complementary. The –o option is the default. It supresses the calculation of capacitance, and instead, gives for each node in the circuit the area and perimeter of that node on the diffusion, poly, and metal layers. (The -O option causes the calculation of capacitance.) The lines containing this information look like this:

N *node diffArea diffPerim polyArea polyPerim metalArea metalPerim*

Node is the node name. *x y* is the position of a point on the node. Currently this is always '0 0'. *DiffArea* through *metalPerim* are the area and perimeter of the diffusion, poly, and metal layers in user defined units. (In addtion the –o [default] option causes transistors with only one terminal to be recorded in the .sim file as a transistor with source connected to drain.)

If the network is being extracted from the .cif file we suggest the node capacitance not be computed by mextra. Rather the *–o* [default] option should be used. This puts the burden of computing node capcitance on the programs *presim* and *sim2spice*. We feel this is advantageous because *presim* and *sim2spice* are filter programs linked directly to the type of simulation that is to be done. This will hopefully reduce some of the confusion associated with calibration.

Normally, mextra ignores connectivity through wells and substrate. If this connectivity is desired to give base connections of transistors, the *–w* option can be used. This allows one to check for proper connections of substrate and to detect shorts through wells and/or substrate. Since wells and substrate are highly resistive, it is desirable to detect what are, essentially, breaks in signals that travel through wells and substrate. This check should be part of the final check of a circuit before submission and should not be used for simulation, since, for instance, cmos-pw input pad circuitry is connected through a well. TO REITERATE: designs to be submitted should be simulated/compared using the –w and then reextracted normally and resimulated/compared to check for possible breaks in power/ground and other signals.

Changing Default Values

As part of its start up procedure *mextra* tries to read two files. It reads ~cad/.cadrc and then searches through the list of (colon separated) directories defined in environment variable CADRC until it finds a .cadrc file. This file is also read. If environment variable CADRC is not defined, *mextra* tries instead to read the .cadrc file in the home directory. *Mextra* reads these files to set up constants to be changed

without recompiling. The keywords for *mextra* are contained within the mextra environment of the .cadrc file. Declaration of environments in the .cadrc file are described in .cadrc(5). Cadrc lines may also be included on the command line by entering them as double quoted strings after the "-c" option. An entire cadrc file may also be read by including the option "-f <file>" on the command line, where the string <file> is the name of the cadrc file.

The temporary directory used may be set using the "-d" option. The string following this option is the temporary directory that will be used by mextra. The default is "/usr/tmp".

By default, *mextra* reports locations in CIF coordinates. A more convenient form of units may be specified either in the .cadrc file or on the command line. The form of the line in the .cadrc file is:

units *scale*

where *scale* is in centi-microns. The user may type in the chosen value for the scale directly.

To set units on the command line use the **–u** option.

mextra –u *scale basename*

The parameters used to compute node capacitance may be changed by including the following commands in your .cadrc file.

areatocap *layer value*
perimtocap *layer value*

value is atto-farads per square micron for area, and atto-farads per micron for perimeter. *layer* may be "poly", "diff", "metal", "metal2", or "poly/diff".

To set the capacitor values to those given in Mead and Conway the following lines would appear in the .cadrc file:

areatocap poly 40
areatocap diff 100
areatocap metal 30
areatocap poly/diff 400
perimtocap poly 0
perimtocap diff 0
perimtocap diff 0
perimtocap metal 0
perimtocap poly/diff 0

The threshold for reporting capacitance may be set in the .cadrc file with the following line.

capthreshold *value*

A negative value sets the threshold to infinity.

Mextra knows of the technologies, nMOS ("nmos"), MOSIS P well CMOS/Bulk (3.0 micron), also known as CBPM ("cmos-pw"), MOSIS Scalable CMOS/Bulk N-well, also known as SCN ("cmos-nw"), MOSIS Scalable CMOS/Bulk P-well, also known as SCP ("cmos-s"), and MOSIS Scalable CMOS/Bulk Generic, also known as SCG ("cmos-g"). The technology can also be set by use of the "-t <tech>" option; the string <tech> is the technology. Nmos is assumed by default. To set an alternate technology, include a line similar to the following in your .cadrc file with the appropriate string:

tech cmos-pw

Valtbs is a simple sed script that extracts all transistor records from the .tbs file except for 'p' transistors whose base is connected to Vdd, vdd, or VDD, and except for 'e' transistors whose base is connected to Gnd, gnd, or GND. Input files are named explicitly on the command line or via standard input. Output is on standard out.

FILES

~/.cadrc
basename.cif

basename.al
basename.log
basename.nodes
basename.sim
basename.ohm
basename.tbs

SEE ALSO

powest(1.vlsi), pspice(1.vlsi), spcpp(1.vlsi), sim2spice(1.vlsi), spice(1.vlsi), drc(1.vlsi), erc(1.vlsi), valtbs(1.vlsi), caesar(cad1), cadrc(cad5)

AUTHOR

Dan Fitzpatrick (UCB)

MODIFICATIONS

Wayne E. Winder (Northwest LIS, University of Washington)

BUGS

Accepts manhattan simple CIF only, use cifplot –O to convert complicated CIF. For unusually shaped transistors the UW/NW modified *mextra* should be used, otherwise values will be quite inaccurate. The modified *mextra* will either yield accurate values or a "reasonable" guess, depending on the complexity of the unusual transistor. The modified *mextra* will tell you when the output values are only best estimates. The length/width ratio for unusually shaped transistors may be inaccurate. This is true for snake transistors. Attributes for funny transistors are not recorded. Node attributes are ignored unless the –o switch is present.

NAME

sim2spice – convert from .sim format to spice format

SYNOPSIS

sim2spice [–d defs] file.sim

DESCRIPTION

Sim2spice reads a file in .sim format and creates a new file in spice format. The file contains just a list of transistors and capacitors, the user must add the transistor models and simulation information. The new file is appended with the tag .spice. One other file is created, which is a list of .sim node names and their corresponding spice node numbers. This file is tagged .names.

Defs is a file of definitions. A definition can be used to set up equivelences between .sim node names and spice node numbers. The form of this type of definition is:

set *sim_name spice_number* [*tech*]

The *tech* field is optional. In NMOS, a special node, 'BULK', is used to represent the substrate node. For CMOS, two special nodes, 'NMOS' and 'PMOS', represent the substrate nodes for the 'n' and 'p' transistors, repectively. For example, for NMOS the .sim node 'GND' corresponds to spice node 0, 'Vdd' corresponds to spice node 1, and 'BULK' corresponds to spice node 2. The *defs* file for this set up would look like this:

```
set GND 0 nmos
set Vdd 2 nmos
set BULK 3 nmos
```

A definition also allows you to set a correspondence between .sim transistor types and and spice transistor types. The form of this definition is:

def *sim_trans spice_trans* [*tech*]

Again, the *tech* field is optional. For NMOS these definitions would look as follows:

```
def e ENMOS nmos
def d DNMOS nmos
```

Definitions may also be placed in the '.cadrc' file, but the definitions in the *defs* file overrides those in the '.cadrc' file.

SEE ALSO

ext2sim(1), magic(1), spice(1), cadrc(5), ext(5), sim(5)

AUTHOR

Dan Fitzpatrick
CMOS fixes by Neil Soiffer

BUGS

The only pre-defined technologies are nmos, cmos-pw, and cmos (the same as cmos-pw). Only one definition file is allowed.

NAME

cif2ps – CIF to PostScript output

SYNOPSIS

cif2ps [-w *width*] [-h *height*] [-t *technology*] [-L *ciflayernames*] [-o *output.ps*] *input1.cif input2.cif* ...

DESCRIPTION

cif2ps takes a CIF file that has been produced by Magic or EGS graphics editor and creates a PostScript file that can be sent to the PostScript printer of choice. The code was written with the CIF layer names hard coded in. The technologies known are: nMOS ("nmos"), MOSIS P well CMOS/Bulk, also known as CBPM ("cmos-pw"), MOSIS Scalable CMOS/Bulk N-well, also known as SCN ("cmos-nw"), MOSIS Scalable CMOS/Bulk P-well, also known as SCP ("cmos-s"), and MOSIS Scalable CMOS/Bulk Generic, also known as SCG ("cmos-g").

Normally (without the -m option), upon invoking cif2ps, the program waits for user to enter the symbol number to be plotted. Entering invalid number will cause the program to list the valid symbol numbers. To terminate the input loop, enter a blank line.

Cif2ps options are :

–w specify width of plot (in pages, default is 1)

–h specify height of plot (in pages, default is 1)

Cif2ps will rescale a user-specific dimension if neccesary to avoid producing blank pages. The largest dimesion allowed is 5 pages by 5 pages (25 pages of output).

–t specify technology.

–m specify that plots are to be superimposed. This is useful for plotting node numbers generated from layout extractor onto the cif file. *cif2ps* will not wait for user input and will proceed and plot all symbols in one page.

–o specify the output file, otherwise cif2ps writes to standard output.

–L specify that the CIF layer names are defined in the file *ciflayernames*.

magic

AUTHOR

Arthur Simoneau wrote the version 'cifp'.
Marc Lesure modified 'cifp' to produce 'cif2ps'.
Boon-Khim Liew added -m, -o and -l options, support for other technologies and ability to draw polygons, wires.

FILES

ciflayernames, input.cif and output.ps

NOTES

PostScript is a trademark of Adobe Systems, Inc.

Appendix I

Irsim Documentation

The following documents describe Irsim.

NAME

irsim - An event-driven logic-level simulator for MOS circuits

SYNOPSIS

irsim [-s] prm_file sim_file ... [+hist_file] [-cmd_file ...]

DESCRIPTION

IRSIM is an event-driven logic-level simulator for MOS (both N and P) transistor circuits. Two simulation models are available:

switch
: Each transistor is modeled as a voltage-controlled switch. Useful for initializing or determining the functionality of the network.

linear
: Each transistor is modeled as a resistor in series with a voltage-controlled switch; each node has a capacitance. Node values and transition times are computed from the resulting RC network, using Chorng-Yeoung Chu's model. Chris Terman's original model is not supported any more.

If the -s switch is specified, 2 or more transistors of the same type connected in series, with no other connections to their common source/drain will be stacked into a compound transistor with multiple gates.

The prm_file is the electrical parameters file that configure the devices to be simulated. It defines the capacitance of the various layers, transistor resistances, threshold voltages, etc... (see presim(1)).
If prm_file does not specify an absolute path then IRSIM will search for the prm_file as follows (in that order):

1) ./<prm_file> (in the current directory).
2) ~cad/lib/<prm_file>
3) ~/cad/lib/<prm_file>.prm

If ~cad/ does not exist, IRSIM will try to use directory /projects/cad. The default search directory (~cad) can be overriden by setting the environment variable CAD_HOME to the appropriate directory prior to running IRSIM (i.e. setenv CAD_HOME /usr/beta/mycad).

IRSIM first processes the files named on the command line, then (assuming the exit command has not been processed) accepts commands from the user, executing each command before reading the next.

File names NOT beginning with a '-' are assumed to be sim files (see sim(5)), note that this version does not require to run the sim files through presim. These files are read and added to the network database. There is only a single name space for nodes, so references to node "A" in different network files all refer to the same node. While this feature allows one to modularize a large circuit into several network files, care must be taken to ensure that no unwanted node merges happen due to an unfortunate clash in names.

File names prefaced with a '-' are assumed to be command files: text files which contain command lines to be pro-

cessed in the normal fashion. These files are processed line by line; when an end-of-file is encountered, processing continues with the next file. After all the command files have been processed, and if an "exit" command has not terminated the simulation run, IRSIM will accept further commands from the user, prompting for each one like so:

```
irsim>
```

The hist_file is the name of a file created with the dumph command (see below). If it is present, IRSIM will initilize the network to the state saved in that file. This file is different from the ones created with the ">" command since it saves the state of every node for all times, including any pending events.

This version supports changes to the network through the update command. Also, the capability to incrementally resimulate the network up to the current time is provided by the isim command.

COMMAND SUMMARY

@ filename	take commands from command file
? wnode...	print info about node's source/drain connections
! wnode...	print info about node's gate connections
< filename	restore network state from file
> filename	write current network state to file
<< filename	same as "<" but restores inputs too
| comment...	comment line
activity from [to]	graph circuit activity in time interval
ana wnode...	display nodes in analyzer window
analyzer wnode...	display nodes in analyzer window
assert wnode [m] val	assert that wnode equals val
back [time]	move back to time
c [n]	simulate for n clock cycles (default:1)
changes from [to]	print nodes that changed in time interval
clock [node [val]]	define value sequence for clock node
clear	clear analyzer window (remove signals)
d [wnode]...	print display list or specified node(s)
debug [debug_level..]	set debug level (default: off)
decay [n]	set charge decay time (0 => no decay)
display [arg]...	control what gets displayed when
dumph filename...	write net history to file
exit [status]	return to system
flush [time]	flush out history up to time (default:now)
h wnode...	make node logic high (1) input
has_coords	print YES if transistor coordinates are available
inputs	print current list of input nodes
ires [n]	set incremental resolution to n ns
isim [filename]	incrementally resimulate changes form filename
l wnode...	make node logic low (0) input
logfile [filename]	start/stop log file
model [name]	set simulation model to name
p	step clock one simulation step (phase)
path wnode...	display critical path for last transition of a node
print comment...	print specified text
printp	print a list of all pending events
printx	print all undefined (X) nodes
q	terminate input from current stream
R [n]	simulate for n cycles (default:longest sequence)
readh filename	read history from filename
report[level]	set/reset reporting of decay events
s [n]	simulate for n ns. (default: stepsize)
stepsize [n]	set simulation step size to n ns.

```
set vector value        assign value to vector
setlog[file|off]        log net changes to file (off -> no log)
setpath                 set search path for cmd files
stats                   print event statistics
t [-]wnode...           start/stop tracing of specified nodes
tcap                    print list of shorted transistors
time [command]          print resource utilization summary
u wnode...              make node undefined (X) input
unitdelay [n]           force transitions to take n ns. (0 disables)
update filename         read net changes from file
V [node [value...]]     define sequence of inputs for a node
vector label node...    define bit vector
w [-]wnode...           add/delete nodes from display list
wnet [filename]         write network to file
x wnode...              remove node from input lists
Xdisplay[host:n]        set/show X display (for analyzer)
```

COMMAND DESCRIPTIONS

Commands have the following simple syntax:

    cmd arg1 arg2 ... argn <newline>

where cmd specifies the command to be performed and the argi are arguments to that command. The arguments are separated by spaces (or tabs) and the command is terminated by a <newline>.

If cmd is not one of the built-in commands documented below, IRSIM appends ".cmd" to the command name and tries to open that file as a command file (see "@" command). Thus the command "foo" has the same effect as "@ foo.cmd".

Notation:

... indicates zero or more repetitions

[ ] enclosed arguments are optional

node name of node or vector in network

wnode
: name of node or vector in network, can include '*' wildcard which matches any sequence of zero or more characters. The pair of characters '{' and '}' denote iteration over the limits enclosed by it, for example: name{1:10} will expand into name1, name2 ... name10. A 3rd optional argument sets the stride, for example: name{1:10:2} will expand into name1, name3, ... name7, name9.

| comment...
: Lines beginning with vertical bar are treated as comments and ignored -- useful for comments or temporarily disabling certain commands in a command file.

Most commands take one or more node names as arguments. Whenever a node name is acceptible in a command line, one can also use the name of a bit vector. In this case, the command will be applied to each node of the vector (the "t" and "d" treat vectors specially, see below).

vector label node...
: Define a bit vector named "label" which includes the specified nodes. If you redefine a bit vector, any

special attributes of the old vector (e.g., being on the display or trace list) are lost. Wild cards are not accepted in the list of node names since you would have no control over the order in which matching nodes would appear in the vector.

The simulator performs most commands silently. To find out what's happened you can use one of the following commands to examine the state of the network and/or the simulator.

set vector value
Assign value to vector. For example, the following sequence of commands:

```
vector BUS bit.1 bit.2 bit.3
set BUS 01x
```

The first command will define BUS to be a vector composed of nodes bit.1, bit.2, and bit.3. The second command will assign the following values:

```
bit.1 = 0
bit.2 = 1
bit.3 = X
```

Value can be any sequence of [0,1,h,H,l,L,x,X], and must be of the same length as the bit vector itself.

d [wnode]...
Display. Without arguments displays the values all nodes and bit vectors currently on the display list (see w command). With arguments, only displays the nodes or bit vectors specified. See also the "display" command if you wish to have the display list printed out automatically at the end of certain simulation commands.

w [-]wnode...
Watch/unwatch one or more nodes. Whenever a "d" command is given, each watched node will displayed like so:

```
node1=0 node2=X ...
```

To remove a node from the watched list, preface its name with a '-'. If wnode is the name of a bit vector, the values of the nodes which make up the vector will be displayed as follows:

```
label=010100
```

where the first 0 is the value of first node in the list, the first 1 the value of the second node, etc.

assert wnode [mask] value
Assert that the boolean value of the node or vector wnode is value. If the comparison fails, an error message is printed. If mask is given then only those bits corresponding to zero bits in mask take part in the comparison, any character other than 0 will skip that bit. The format of the error message is the following:

```
(tty, 3): assertion failed on 'name' 10X10 (1010X)
```

Where name is the name of the vector, followed by the actual value and the expected value enclosed in

parenthesis. If a mask is specified, then bits that were not compared are printed as '-'.

ana wnode...
This is a shorthand for the analyzer command (described below).

analyzer wnode...
Add the specified node(s)/vector(s) to the analyzer display list (see irsim-analyzer(3) for a detailed explanation). If the analyzer window does not exist, it will be created. If no arguments are given and the analyzer window already exists, nothing happens.

Xdisplay [host:display]
You must be able to connect to an X-server to start the analyzer. If you haven't set up the DISPLAY environment variable properly, the analyzer command may fail. If this is the case you can use the Xdisplay command to set it from within the simulator. With no arguments, the name of the current X-server will be printed.

clear
Removes all nodes and vectors from the analyzer window. This command is most useful in command scripts for switching between different signals being displayed on the analyzer.

"?" and "!" allow the user to go both backwards and forwards through the network. This is a useful debugging aid.

? wnode...
Prints a synopsis of the named nodes including their current values and the state of all transistors that affect the value of these nodes. This is the most common way of wandering through the network in search of what went wrong.
The output from the command ? out looks like

```
out=0 (vl=0.3 vh=0.8) (0.100 pf) is computed from:
n-channel phi2=0 out=0 in=0 [1.0e+04, 1.3e+04, 8.7e+03]
pulled down by (a=1 b=1)  [1.0e+04, 1.3e+04, 8.8e+03]
pulled up [4.0e+04, 7.4e+04, 4.0e+04]
```

The first line gives the node's name and current value, its low and high logic thresholds, user-specifed low-to-high and high-to-low propagation delays if present, and its capacitance if nonzero. Succeeding lines list the transistor whose sources or drains connect to this node: the transistor type ("pulled down" is an n-channel transistor connected to gnd, "pulled up" is a depletion pullup or p-channel transistor connected to vdd), the values of the gate, source, and drain nodes, and the modeling resistances. Simple chains of transistors with the same implant type are collapsed by the -s option into a single transistor with a "compound" gate; compound gates appear as a parenthesized list of nodes (e.g., the pulldown shown above). The three resistance values -- static, dynamic high, dynamic low -- are given in Kilo-ohms.

Finally, any pending events for a node are listed after the electrical information.

! wnode...
For each node in the argument list, print a list of

transistors controlled by that node.

tcap
: Prints a list of all transistors with their source/drain shorted together or whose source/drain are connected to the power supplies. These transistors will have no effect on the simulation other than their gate capacitance load. Although transistors connected across the power supplies are real design errors, the simulator does not complain about them.

Any node can be made an input -- the simulator will not change an input node's value until it is released. Usually on specific nodes -- inputs to the circuit -- are manipulated using the commands below, but you can fool with a subcircuit by forcing values on internal nodes just as easily.

h wnode...
: Force each node on the argument list to be a high (1) input. Overrides previous input commands if necessary.

l wnode...
: Like "h" except forces nodes to be a low (0) input.

u wnode...
: Like "h" except forces nodes to be a undefined (X) input.

x wnode...
: Removes nodes from whatever input list they happen to be on. The next simulation step will determine the correct node value from the surrounding circuit. This is the default state of most nodes. Note that this does not force nodes to have an "X" value -- it simply removes them from the input lists.

inputs
: prints the high, low, and undefined input lists.

It is possible to define a sequence of values for a node, and then cycle the circuit as many times as necessary to input each value and simulate the network. A similar mechanism is used to define the sequence of values each clock node goes through during a single cycle.

Each value is a list of characters (with no intervening blanks) chosen from the following:

1, h, H	logic high (1)
0, l, L	logic low (0)
u, U	undefined (X)
x, X	remove node from input lists

Presumably the length of the character list is the same as the size of the node/vector to which it will be assigned. Blanks (spaces and tabs) are used to separate values in a sequence. The sequence is used one value at a time, left to right. If more values are needed than supplied by the sequence, IRSIM just restarts the sequence again.

V [node [value...]]
: Define a vector of inputs for a node. After each cycle of an "R" command, the node is set to the next value specified in the sequence.

With no arguments, clears all input sequences (does not affect clock sequences however). With one argument, "node", clears any input sequences for that node/vector.

clock [node [value...]]

Define a phase of the clock. Each cycle, each node specified by a clock command must run through its respective values. For example,

```
clock phi1 1 0 0 0
clock phi2 0 0 1 0
```

defines a simple 4-phase clock using nodes phi1 and phi2. Alternatively one could have issued the following commands:

```
vector clk phi1 phi2
clock clk 10 00 01 00
```

With no arguments, clears all clock sequences. With one argument, "node", clears any clock sequences for that node/vector.

After input values have been established, their effect can be propagated through the network with the following commands. The basic simulated time unit is 0.1ns; all event times are quantized into basic time units. A simulation step continues until stepsize ns. have elapsed, and any events scheduled for that interval are processed. It is possible to build circuits which oscillate -- if the period of oscillation is zero, the simulation command will not return. If this seems to be the case, you can hit <ctrl-C> to return to the command interpreter. Note that if you do this while input is being taken from a file, the simulator will bring you to the top level interpreter, aborting all pending input from any command files.

When using the linear model (see the "model" command) transition times are estimated using an RC time constant calculated from the surrounding circuit. When using the switch model, transitions are scheduled with unit delay. These calculations can be overridden for a node by setting its tplh and tphl parameters which will then be used to determine the time for a transition.

s [n]

Simulation step. Propogates new values for the inputs through the network, returns when n (default: stepsize) ns. have passed. If n is specified, it will temporarily override the stepsize value. Unlike previous versions, this value is NOT remembered as the default value for the stepsize parameter. If the display mode is "automatic", the current display list is printed out on the completion of this command (see "display" command).

c [n]

Cycle n times (default: 1) through the clock, as defined by the "clock" command. Each phase of the clock lasts stepsize ns. If the display mode is "automatic", the current display list is printed out on the completion of this command (see "display" command).

p Step the clock through one phase (or simulation step).

For example, if the clock is defined as above

```
clock phi1   1 0 0 0
clock phi2   0 0 1 0
```

then "p" will set phi1 to 1 and phi2 to 0, and then propagate the effects for one simulation step. The next time "p" is issued, phi1 and phi2 will both be set to 0, and the effects propagated, and so on. If the "c" command is issued after "p" has been used, the effect will be to step through the next 4 phases from where the "p" command left off.

R [n]

Run the simulator through n cycles (see the "c" command). If n is not present make the run as long as the longest sequence. If display mode is automatic (see "display" command) the display is printed at the end of each cycle. Each "R" command starts over at the beginning of the sequence defined for each node.

back time

Move back to the specified time. This command restores circuit state as of "time", effectively undoing any changes in between. Note that you can not move past any previously flushed out history (see flush command below) as the history mechanism is used to restore the network state. This command can be useful to undo a mistake in the input vectors or to re-simulate the circuit with a different debug level.

path wnode...

display critical path(s) for last transition of the specified node(s). The critical path transistions are reported using the following format:

```
node -> value @ time (delta)
```

where node is the name of the node, value is the value to which the node transitioned, time is the time at which the transistion occurred, and delta is the delay through the node since the last transition. For example:

```
critical path for last transition of Hit_v1:
     phi1-> 1 @ 2900.0ns , node was an input
     PC_driver-> 0 @ 2900.4ns    (0.4ns)
     PC_b_q1-> 1 @ 2904.0ns    (3.6ns)
     tagDone_b_v1-> 0 @ 2912.8ns    (8.8ns)
     tagDone1_v1-> 1 @ 2915.3ns    (2.5ns)
     tagDone1_b_v1-> 0 @ 2916.0ns    (0.7ns)
     tagDone_v1-> 1 @ 2918.4ns    (2.4ns)
     tagCmp_b_v1-> 0 @ 2922.1ns    (3.7ns)
     tagCmp_v1-> 1 @ 2923.0ns    (0.9ns)
     Vbit_b_v1-> 0 @ 2923.2ns    (0.2ns)
     Hit_v1-> 1 @ 2923.5ns    (0.3ns)
```

activity from_time [to_time]

print histogram showing amount of circuit activity in the specified time inteval. Actually only shows number of nodes which had their most recent transition in the interval.

changes from_time [to_time]

print list of nodes which last changed value in the specified time interval.

printp
    print list of all pending events sorted in time. The node associated with each event and the scheduled time is printed.

printx
    print a list of all nodes with undefined (X) values.

Using the trace command, it is possible to get more detail about what's happening to a particular node. Much of what is said below is described in much more detail in "Logic-level Simulation for VLSI Circuits" by Chris Terman, available from Kluwer Academic Press. When a node is traced, the simulator reports each change in the node's value:

```
[event #100] node out.1: 0 -> 1 @ 407.6ns
```

The event index is incremented for each event that is processed. The transition is reported as

```
old value -> new value @ report time
```

Note that since the time the event is processed may differ from the event's report time, the report time for successive events may not be strictly increasing.

Depending on the debug level (see the "debug" command) each calculation of a traced node's value is reported:

```
[event #99] node clk: 0 -> 1 @ 400.2ns
 final_value( Load )  V=[0.00, 0.04]  => 0
 ..compute_tau( Load )
    {Rmin=2.2K  Rdom=2.2K  Rmax=2.2K}  {Ca=0.06  Cd=0.17}
    tauA=0.1  tauD=0.4 ns
[event #99: clk->1] transition for Load: 1 -> 0 (tau=0.5ns, delay=0.6ns)
```

In this example, a calculation for node Load is reported. The calculation was caused by event 99 in which node clk went to 1. When using the linear model (as in this example) the report shows

```
current value -> final value
```

The second line displays information regarding the final value (or dc) analysis for node "Load"; the minimun and maximum voltages as well as the final logical value (0 in this case).

The next three lines display timing analysis information used to estimate the delays. The meaning of the variables displayed can be found Chu's thesis: "Improved Models for Switch-Level Simulation".

When the final value is reported as "D", the node is not connected to an input and may be scheduled to decay from its current value to X at some later time (see the "decay" command).

"tau" is the calculated transition time constant, "delta" is when any consequences of the event will be computed; the difference in the two times is how IRSIM accounts for the shape of the transition waveform on subsequent stages (see reference given above for more details). The middle lines of the report indicate the Thevenin and capacitance parameters of the surrounding networks, i.e., the parameters on

which the transition calculations are based.

debug [ev dc tau taup tw spk][off][all]
Set debugging level. Useful for debugging simulator and/or circuit at various levels of the computation. The meaning of the various debug levels is as follows:

ev — display event enqueueing and dequeueing.

dc — display dc calculation information.

tau — display time constant (timing) calculation.

taup — display second time constant (timing) calculation.

tw — display network parameters for each stage of the tree walk, this applies to dc, tau, and taup. This level of debugging detail is usually needed only when debugging the simulator.

spk — displays spike analysis information.

all — This is a shorthand for specifying all of the above.

off — This turns off all debugging information.

If a debug switch is on then during a simulation step, each time a watched node is encounted in some event, that fact is indicated to the user along with some event info. If a node keeps appearing in this prinout, chances are that its value is oscillating. Vice versa, if your circuit never settles (ie., it oscillates) , you can use the "debug" and "t" commands to find the node(s) that are causing the problem.

Without any arguments, the debug command prints the current debug level.

t [-]wnode...
set trace flag for node. Enables the various printouts described above. Prefacing the node name with '-' clear its trace flag. If "wnode" is the name of a vector, whenever any node of that vector changes value, the current time and the values of all traced vectors is printed. This feature is useful for watching the relative arrival times of values at nodes in an output vector.

System interface commands:

> filename
Write current state of each node into specified file. Useful for making a breakpoint in your simulation run. Only stores values so isn't really useful to "dump" a run for later use, i.e., the current input lists, pending events, etc. are NOT saved in the state file.

< filename
Read from specified file, reinitializing the value of each node as directed. Note that network must already exist and be identical to the network used to create the dump file with the ">" command. These state saving commands are really provided so that complicated initializing sequences need only be simulated once.

<< filename
Same as "<" command, except that this command will restore the input status of the nodes as well. It does not, however, restore pending events.

dumph [filename]
Write the history of the simulation to the specified file, that is; all transistions since time = 0. The resulting file is a machine-independent binary file, and contains all the required information to continue simulation at the time the dump takes place. If the filename isn't specified, it will be constructed by taking the name of the sim_file (from the command line) and appending ".hist" to it.

readh filename
Read the specified history-dump file into the current network. This command will restore the state of the circuit to that of the dump file, overwriting the current state.

flush [time]
If memory consumption due to history maintanance becomes prohibitive, this command can be used to free the memory consumed by the history up to the time specified. With no arguments, all history up to the current point in the simulation is freed. Flushing out the history may invalidate an incremental simulation and the portions flushed will no longer appear in the analyzer window.

setpath [path...]
Set the search-path for command files. Path should be a sequence of directories to be searched for ".cmd" files, "." meaning the current directory. For eaxmple:

```
setpath . /usr/me/rsim/cmds /cad/lib/cmds
```

With no arguments, it will print the current search-path. Initially this is just ".".

print text...
Simply prints the text on the user's console. Useful for keeping user posted of progress through a long command file.

logfile [filename]
Create a logfile with the specified name, closing current log file if any; if no argument, just close current logfile. All output which appears on user's console will also be placed in the logfile. Output to the logfile is cleverly formatted so that logfiles themselves can serve as command files.

setlog [filename | off]
Record all net changes, as well as resulting error messages, to the specified file (see "update" command). Net changes are always appended to the log-file, preceding each sequence of changes by the current date. If the argument is off then net-changes will not be logged. With no arguments, the name of the current log-file is printed.

The default is to always record net changes; if no filename is specified (using the "setlog" command) the default filename irsim_changes.log will be used. The

log-files are formatted so that log-files may themselves be used as net-change files.

wnet [filename]
Write the current network to the specified file. If the filename isn't specified, it will be constructed by taking the name of the sim_file (from the command line) and appending ".inet" to it. The resulting file can be used in a future simulation run, as if it were a sim file. The file produced is a machine independent binary file, which is typically about 1/3 the size of the sim file and about 8 times faster to load.

time [command]
With no argument, a summary of time used by the simulator is printed. If arguments are given the specified command is timed and a time summary is printed when the command completes. The format of the time summary is Uu Ss E P% M, where:

U => User time in seconds
S => System time in seconds
E => Elapsed time, minutes:seconds
P => Percentage of CPU time (((U + S)/E) * 100)
M => Median text, data, and stack size use

q Terminate current input stream. If this is typed at top level, the simulator will exit back to the system; otherwise, input reverts to the previous input stream.

exit [n]
Exit to system, n is the reported status (default: 0).

Simulator parameters are set with the following commands. With no arguments, each of the commands simply prints the current value of the parameter.

decay [n]
Set decay parameter to n ns. (default: 0). If non-zero, it tells the number of ns. it takes for charge on a node to decay to X. A value of 0 implies no decay at all. You cannot specify this parameters separately for each node, but this turns out not to be a problem. See "report" command.

display [-][cmdfile][automatic]
set/reset the display modes, which are

cmdfile commands executed from command files are displayed to user before executing. The default is cmdfile = OFF.

automatic print out current display list (see "d" command) after completion of "s" or "c" command. The default is automatic = ON.

Prefacing the previous commands with a "-" turns off that display option.

model [name]
Set simulation model to one of the following:

switch
Model transistors as voltage controlled switches. This model uses interval logic levels, without accounting for transistor resistances, so circuits

with fighting transistors may not be accurately modelled. Delays may not reflect the true speed of the circuit as well.

linear
Model transistors as a resistor in series with a voltage controlled switch. This model uses a single-time-constant computed from the resulting RC network and uses a two-time-constant model to analyze charge sharing and spikes.

The default is the linear model. You can change the simulation model at any time -- even with events pending -- as only new calculations are affected. Without arguments, this command prints the current model name.

report [level]
When level is nonzero, report all nodes which are set to X because of charge decay, regardless on whether they are being traced. Setting level to zero disables reporting, but not the decay itself (see "decay" command).

stepsize [n]
Specify duration of simulation step or clock phase. n is specified in ns. (nanoseconds). Floating point numbers with up to 1 digit past the decimal point are allowed. Further decimals are trucated (i.e. 10.299 == 10.2).

unitdelay [n]
When nonzero, force all transitions to take n ns. Setting the parameter to zero disables this feature. The resolution is the same as for the "stepsize" command.

stats
Print event statitistics, as follows:

```
changes = 26077
punts (cns) = 208 (34)
punts = 0.79%, cons_punted = 16.35%
nevents = 28012; evaluations = 27972
```

Where changes is the total number of transistions recorded, punts is the number of punted events, (cns) is the number of consecutive punted events (a punted event that punted another event). The penultimate line shows the percentage of punted events with respect to the total number of events, and the percentage of consecutive punted events with respect to the number of punted events. The last line shows the total number of events (nevents) and the number of net evaluations.

Incremental simulation commands:

Irsim supports incremental changes to the network and resimulation of the resulting network. This is done incrementally so that only the nodes affected by the changes, either directly or indirectly, are re-evaluated.

update filename
Read net-change tokens from the specified file. The following net-change commands are available:

add type gate source drain length width [area]

```
delete  type gate source drain length width [area]
move    type gate source drain length width [area] g s d
cap     node value
N       node metal-area poly-area diff-area diff-perim
M       node M2A M2P MA MP PA PP DA DP PDA PDP
thresh  node low high
Delay   node tplh tphl
```

For a detailed dscription of this file see netchange(5). Note that this is an experimental interface and is likely to change in the future.

Note that this command doesn't resimulate the circuit so that it may leave the network in an inconsistent state. Usually this command will be followed by an isim command (see below), if that is not the case then it's up to the user to initilize the state of the circuit. This command exists only for historical reasons and will probably disappear in the future. It's use is discouraged.

isim [filename]

Read net-change tokens from the specified file (see netchange(5)) and incrementally resimulate the circuit up to the current simulation time (not supported yet).

ires n

The incremental algorithm keeps track of nodes deviating from their past behavior as recorded in the network history. During resimulation, a node is considered to deviate from its history if it's new state is found to be different within n ns of its previous state. This command allows for changing the incremental resolution. With no arguments, it will print the current resolution. The default resolution is 0 ns.

SEE ALSO

presim(1) (now obsolete)
rsim(1)
irsim-analyzer(3)
sim(5)
netchange(5)