DEVELOPMENT OF A NUMERICAL COMPUTER CODE AND CIRCUIT ELEMENT MODELS FOR SIMULATION OF FIRING SYSTEMS

Kenneth H. Carpenter

Department of Electrical and Computer Engineering
Kansas State University
Manhattan, Kansas 66506

July 2, 1990

FINAL REPORT

for the period April 15, 1988, to April 15, 1990
under subcontract 1507303 between the Regents of the University of California and Kansas State University

to

Dr. Ronald S. Lee, L-281
Lawrence Livermore National Laboratory
P. O. Box 808
Livermore, California 94550

DISCLAIMER

UCRL-CR--105603

DE91 007162

Abstract

Numerical simulation of firing systems requires both the appropriate circuit analysis framework and the special element models required by the application. We have modified the SPICE circuit analysis code (version 2G.6), developed originally at the Electronic Research Laboratory of the University of California, Berkeley, to allow it to be used on MSDOS-based, personal computers and to give it two additional circuit elements needed by firing systems – fuses and saturating inductances. An interactive editor and a batch driver have been written to ease the use of the SPICE program by system designers, and the interactive graphical post processor, NUTMEG, supplied by U. C. Berkeley with SPICE version 3B1, has been interfaced to the output from the modified SPICE. Documentation and installation aids have been provided to make the total software system accessible to PC users. Sample problems show that the resulting code is in agreement with the FIRESET code on which the fuse model was based (with some modifications to the dynamics of scaling fuse parameters).

In order to allow for more complex simulations of firing systems, studies have been made of additional special circuit elements – switches and ferrite cored inductances. A simple switch model has been investigated which promises to give at least a first approximation to the physical effects of a non ideal switch, and which can be added to the existing SPICE circuits without changing the SPICE code itself. The effect of fast rise time pulses on ferrites has been studied experimentally in order to provide a base for future modeling and incorporation of the dynamic effects of changes in core magnetization into the SPICE code.

This report contains detailed accounts of the work on these topics performed during the peroid it covers, and has appendices listing all source code written and documentation produced.

Contents

Forward

This report is the final report for the work performed during the period April 15, 1988, through April 15, 1990, under Subcontract 1507303 from the Regents of the University of California to Kansas State University. The work was performed by Professor Kenneth H. Carpenter, the principal investigator for Kansas State University. He was assisted by Graduate Research Assistant Damon Mick during part of the period, and by Steve Warren, both as an undergraduate student assistant and as a Graduate Research Assistant during part of the period.

Acknowledgements

The author gratefully acknowleges the direction, help, and encouragement given to him by Dr. Ronald S. Lee of Lawrence Livermore National Laboratory. The help and hospitality shown by all the Laboratory staff during his visits there is appreciated. Special thanks go to Ralph Boberg for helping Steve Warren with his experiments during the summer of 1989. Thanks also to the staff members of the KSU Department of Electrical and Computer Engineering for their help with accounting, mailing, computer matters, and so forth.

Preface

During the time period covered by this report, several different, although related, topics have been investigated. Furthermore, these investigations were often proceeding in parallel; and after a subject was essentially completed minor modifications were sometimes required at a later date. Thus rather than presenting a chronological history of the work in this report, it will be organized to present each subject at its current status. The details of how that status was reached will be omitted in most cases.

Chapter 1

The basis for a firing system simulation code

Design of the electrical circuitry needed for firing systems depends on having good circuit simulation software available. Further, the software needs to be adapted to the available computers. While there have been many general purpose circuit simulation codes developed, these have either been limited to specific computers, to specific types of circuits, or have required the user to have a detailed knowledge of the numerical techniques to obtain accurate results. Further, the specific nonlinear behavior of several of the circuit components needed in firing system simulations pose severe problems for a general numerical analysis. For these reasons, a study was undertaken to determine the best approach to numerical simulation of firing systems, prior to beginning the actual work of developing the simulation program.

1.1 Numerical approaches for firing system simulation

Simulation of the electrical circuitry used in firing systems is much the same as any electrical circuit analysis, with the addition of nonlinear circuit elements representing the special components required. However, the nature of the nonlinear elements can be such as to cause severe difficulties with numerical methods. Experience in computer modeling of the circuit elements themselves provided insight into this problem. Before beginning development of a specific circuit simulation program, several existing programs were examined for suitability as a basis for the work. A preliminary plan for a completely new code was also undertaken. The result of this investigation was to select SPICE, a widely used program developed and distributed by the Department of Electrical Engineering at the University of California at Berkeley[7], as the basis for the firing system simulation software. The remainder of this section of this report provides a record of this investigation.

1.1.1 Target computers

One of the goals set for the firing system simulation program was that it be useable on personal computers with moderate computational capability. If the program could only be run on large computer systems the engineers at small subcontractors would not be likely to have access to it. On the other hand, the computational hardware must be capable of executing simulations of a reasonable complexity in a reasonable amount of time. Keeping within these limitations, the target computers for the study were from, at a minimum, a MSDOS-based, "XT" class personal computer having a numerical coprocessor with 640 kilobytes of memory, to, at a maximum, a "Unix" or "Microvax" type of workstation, having floating point capability and virtual memory. Development of the code could be done on more powerful hardware, but would ultimately have to be moved to the target computers.

The code resulting from this work has met the goal for target computers. It runs on the minimal PC configuration described above. In addition, the modified SPICE program (but not the editor and

batch driver) can be run on the maximal systems.

1.1.2 Existing codes and limitations

There are a large number of circuit simulation codes in existence. One of the earliest was ECAP[1], which ran on IBM 1620 and comparable machines. SPICE[2] was one of the first simulation codes developed which had nonlinear device models included. It is now in its third major version and is widely used. Many commercial personal computer versions of SPICE[3] are now sold independently from the freely distributed version from UC Berkeley. There are also other general purpose codes, such as SCEPTRE[4], intended for large computer systems. In addition, there are many special purpose simulation codes that have been written in response to particular program needs, such as SCREAMER[5] and FIRESET[6]. Several of these existing codes have been studied to see if they could meet the need for firing system simulation, or if it would be possible to modify them so that they would. Finally, a feasibility study has been made of the writing of a new special purpose code to simulate firing systems. The result of these investigations has been to choose SPICE as a basis for firing system simulation.

The commercial codes, such as the many SPICE derivatives, were rejected from further consideration due to license costs and lack of access to the source code for the programs. While some of these products will allow user-supplied subroutines for incorporation of additional models, whether or not the flexibility would be present to allow representing all types of needed models could not be judged without actually doing the work. If failure to represent a peculiarity of a model then arose, without source code, one would be at a dead end.

The FIRESET[6] program by Ron Lee is actually a special purpose firing system simulation code. The model it contains has to be incorporated into any general code to be developed. It is written in BASIC, and serves as the starting point for coding its model, but it is not itself a candidate for building a more comprehensive simulation program.

The SCREAMER[5] code was supplied for use by Mark Kiefer of Sandia National Laboratories. It is a special purpose code, intended for the analysis of series-parallel circuits found in particle accelerator systems. The name implies its fast execution time on its design task. However, it achieves speed by using fixed time step integration techniques. This fails for "stiff" systems, such as represented by the nonlinear characteristics of fuses and saturating inductors. Since the numerical approach is interwoven with the entire code, it would not be feasible to try to use SCREAMER as the starting point for a general firing systems code.

The SPICE[2] code has many features in its favor. It has been the subject of many man years of development effort, and is reasonably modular in design. The numerical methods can handle severe nonlinearities, and many nonlinear models are included in the code (all were semiconductor device models). The source code for SPICE is distributed by UC Berkeley, and no license fees are required. These reasons made it the leading contender for the basis for a firing system simulation program.

1.1.3 Feasibility of a new code using a series-parallel circuit approach

There are advantages to writing a completely new code over adapting an existing one to a new use. The main advantage is having complete control over and knowledge of its structure. The main disadvantage is that one is "reinventing the wheel" for much of the effort. A second disadvantage is that in the time allowed to complete the work only more modest goals can be met if all work must be started fresh. Nevertheless, a study was made to determine the basic form that a code would need to take to represent firing systems, limiting their topography to series-parallel circuits. A written report of this study was submitted to Ron Lee in June 1988. It is included in Appendix A of this report.

The study concluded that such a ladder network approach could lead to a code that would be reasonable to use on personal computers, especially if the inductances were all linear. Introduction of nonlinear inductance in a circuit having many branches would greatly increase computational difficulty, however. While this approach could have led to a simple program, independent from SPICE and much smaller than SPICE, it would not have been capable of extension into other topologies or allowed introducing different classes of circuit elements. Thus the series-parallel circuit was not pursued any further.

1.1.4 SPICE

The SPICE[8] computer code is actually a family of codes that has been developed over a period of years by a number of different professors and students at the Electronics Research Laboratory, College of Engineering, University of California, Berkeley, California. At the time the work described in this report was begun, there were two versions of SPICE being distributed by the Industrial Liaison Program of the Department of Electrical Engineering and Computer Sciences of UC Berkeley[7]. These versions were SPICE2G.6, which has its source code in FORTRAN, and SPICE3B.1, which has its source code in "c". (At the time of this writing version SPICE3C.1 has replaced SPICE3B.1 as the "c" version distributed by UC Berkeley. However, only SPICE3B.1 was used for the work reported here.) These two versions have different features, as will be discussed below, and each has been utilized in developing the firing system simulation software.

The SPICE program was chosen as the basis for the firing system simulation code because of several attractive features. The first is its general purpose nature. This means that very general, nonlinear circuit analyses can be performed using SPICE without any modifications to the basis code. Further, the FORTRAN version of SPICE was written with forethought for overlaying the code in computer memory, thus making it possible to fit the code in to the memory limits of an MSDOS-based personal computer. The "c" version of SPICE has been written in a modular fashion to allow incorporation of portions of it in other analysis systems, and to allow removal of unneeded portions of code to save memory. The SPICE versions have been in use for a number of years, giving time for major difficulties to be discovered and corrected. The amount of work that has gone into the development of SPICE could not be duplicated in any special purpose code written to solve a special case situation, such as firing system simulation.

The final feature which made it possible to use SPICE as the basis of the firing system simulation code is the liberal distribution policy of the University of California. While the code is copywrited, it is freely distributed by the University, and no restrictions are placed on redistribution. (Many commercial firms have taken advantage of this to add their own enhancements and then sell derivative programs[3].) To be sure that there would be no restriction on redistribution of SPICE as used with the firing system simulation software, a letter was written to the distribution office at UC Berkeley requesting a definitive statement on the matter. The reply received was a form letter, declaring the SPICE code has always been considered "public domain." The letter and reply is reproduced in Appendix B.

In light of all the foregoing discussion, the work on developing a firing system simulation code was begun, using SPICE as a basis, with the intent of including fuse and saturating inductance models, and providing a convenient user interface. Details of this development are given in the following chapters of this report.

Chapter 2

Development of a numerical simulation code for firing systems based on SPICE

Once the decision was made to base the firing system simulation program on the SPICE code, there were several problems to be solved to carry this out. First, since the Berkeley versions of SPICE are not usually run on personal computers, there was the need to port the codes to run on MSDOS-based personal computers. Second, for SPICE to be used for firing system simulation, the models for the special nonlinear devices needed in these systems had to be added to the SPICE program. Third, the resulting codes needed extensive testing to be sure that correct simulations were being produced. These three aspects of the work were carried on in a parallel fashion, with the first addition of a fuse model done before porting the FORTRAN version to the PC but after porting the "c" version to the PC. Tests were performed in parallel with all aspects of the work to catch problems as soon as possible. A summary of these procedures follows.

2.1 Porting SPICE to the PC

SPICE is a large numerical program for any computer platform. The early versions of SPICE ran on time sharing computer systems using Unix, IBM, and CDC operating systems. With the move from FORTRAN to "c" with version SPICE3A.7, compilations were made for IBM-PC compatible personal computers using the Lattice "c" compiler. Difficulties with the "c" versions of SPICE led to the decision to switch to SPICE2G.6, the last FORTRAN version, as the basis for the firing system simulation code. This required a port of the FORTRAN version to the PC, using the Microsoft FORTRAN compiler. As part of this port, modifications were made to allow the FORTRAN version to output "raw data" files which could be used by the postprocessor part of the "c" version to allow interactive graphics examination of results from simulations.

2.1.1 Porting SPICE3B1

The source code of SPICE3B1 is in the "c" programming language. This language is the primary one used with Unix operating systems, and it was the Unix dialect of the language that was used for development of SPICE3B1. The "c" compilers available for MSDOS-based personal computers are slightly different from the Unix "c" compiler, which leaves a certain amount of "porting" to be done. In particular, the header files that are "included" to allow standard library functions to be used by a program, are somewhat different between PC's and Unix-based computers. The way numbers are stored in memory, the way lines of text are ended, and the binary length of integers and other mathematical objects also differ between different computer implementations of "c". The Lattice "c" compiler had

been used by UC Berkeley to make a PC version of SPICE3B1. The compiler available for use was the Borland Turbo C compiler, version 1.5. It was desirable to use it instead of the Lattice product because of additional library functions available, and so the necessary changes were made, rather than purchasing a Lattice compiler.

The source code for SPICE3B1 is about 5 megabytes long, and consists of a great many short files located in a tree structure of directories and subdirectories. Extensive use is made of structures containing pointers to other structures and pointers to functions. This, presumably, makes the code more modular and easier to change, but it makes it extremely difficult to understand for someone who has not been involved with its development from the beginning. The documentation provided is some help, but still one is essentially forced to follow the rules provided in the "makefile's" and then fix errors that show up due to the differences in the compilers. This procedure was followed, and eventually an executable for SPICE3B1 was produced which could be run on the PC.

Under a Unix operating system with virtual memory, SPICE can expand itself almost without limit, to accommodate a wide variety of circuits and circuit components. On the PC the memory is limited to 640 kilobytes, including MSDOS and the program plus its data structures. Thus to be able to have room for the data for a circuit of any practical size, the Unix version of SPICE3B1 has to be cut down and restricted for use on a PC. This is done by removing all the interactive code and producing a batch version (referred to in the documentation as BSPICE), and by removing any of the semiconductor models that are not anticipated to be needed in simulating circuits on the PC. Removing certain MOSFET and MESFET models provided enough memory space, but having only a batch mode capability posed another challenge. The batch mode only version, BSPICE, writes no output in ASCII, but only creates a "rawfile" of data to be fed to a postprocessor program. Thus the postprocessor program, called NUTMEG, also had to be made to work on the PC.

Large computer codes run on Unix operating systems usually include files called "makefile's". These contain the instructions to the compiler on which modules depend on which others, how they are to be compiled, and how to link them together to produce the executable binary versions. The Turbo C compiler provides a "make" facility that is a subset of the Unix version. Also, the convention for naming files is different under MSDOS that under Unix. Thus the "makefile's" had to be edited before the compilation under Turbo C could begin. (The Lattice compiler was intended to be invoked from a set of "batch" files rather than "makefile's".) There were separate "makefile's" for each subdirectory, and they were chained together by "makefile's" in the higher level directories. The big advantage of using the makefile method over the batch file method is that with "make", once a file has been successfully compiled it does not have to be compiled again, even when changes are made to other files.

When running the "make" of the BSPICE or NUTMEG programs, an error would stop the compile and leave an error message. Then the particular source of the error could be sought (usually something like a library header file needing a different name from Unix or Lattice), fixed, and "make" invoked again to restart where it had left off. Finally the executables would be produced without "fatal" errors, and could be tested.

The NUTMEG postprocessor program for SPICE3B1 provides interactive graphics displays of data as well as producing printed output similar that which can be output directly on large computer versions of SPICE. NUTMEG utilizes most of the same source code that would be included with SPICE3B1 itself on a computer with virtual memory. The PC version intended for use with the Lattice compiler has code included to drive a PC color graphics adapter (in 640 by 200 black and white mode only). The Turbo C compiler supplies graphics libraries that can drive other PC displays, so the code was modified to use the Turbo C libraries. Unfortunately, "bugs" in the code, prevented the version using the Turbo C libraries from ever functioning properly. Thus the CGA display only version was reverted to, and a version called NUTCGA was produced using Turbo C. (The "bugs" were most likely something within NUTMEG overwriting a portion of memory inadvertently. In such a large code it is very difficult to find the source of the overwriting. However, small changes in the program can often move the overwriting from a region where it is harmless to one where it destroys critical parts of the code. Thus with Borland graphics NUTMEG always "hung" the computer, while with the simpler CGA-only graphics it executes without this problem.)

The NUTMEG source, as distributed with SPICE3B1, contained a few noticeable "bugs" that were

corrected. One had to do with a failure of the mechanism to stop a listing of data to the screen before it was completed. Another had to do with label locations on the graphical displays. Whenever an annoying aspect was observed which could be easily corrected, it was.

One annoying aspect of NUTMEG has not been corrected, however. That is the behavior that after making several complicated plots the dynamic memory management runs out of space to do more work. When this occurs NUTMEG exits to the DOS prompt and must be restarted. No cure to this problem could be found, and it may be something inherent in the way the dynamic memory management of either MSDOS or the compiler library is carried out. (Discussions of such behavior have appeared on the electronic mail - USENET "news" network recently.) Scanning the source code did not turn up any use of memory that was not subsequently released. However, in such complex code, instances where this happened could have been missed. The time required to track down such a possibility was judged excessive for the return in convenience.

With working executable versions of BSPICE and NUTMEG, testing of them could be done. For small, simple circuits, BSPICE seemed to produce correct results. However, when more complicated programs were run, often the program would "hang" and no output be produced. A discussion of SPICE3B1 and SPICE in general was held at LLNL with W. G. Magnuson, Jr., on July 27, 1988. He reported difficulties encountered with using SPICE3B1 had been great enough that his group had returned to using SPICE2G6, and recommended that SPICE2G6 be used as the basis for the firing system simulation code rather than SPICE3B1. He also made available the FORTRAN source he was using on a LLNL VAX computer. This source was then used to begin the process of adding new circuit elements to SPICE (although a different, Unix based, source of SPICE2G6 was used to perform the port to the PC). Thus no further work on BSPICE was carried out, although NUTMEG was used with SPICE2G6 to provide the same kind of postprocessor capability that had been introduced with SPICE3B1.

2.1.2 Porting SPICE2G6

While work was begun on adding fuse and saturating inductance models to SPICE2G6 using the VAX version supplied by Magnuson, the port of SPICE2G6 was made to the PC from a different SPICE2G6 version. (This latter version was intended for using under Unix; it was available at Kansas State, and another copy of it was obtained via "ftp" over the Internet at a later date.) The porting of the Unix version to a version which could be compiled using Microsoft FORTRAN on a MSDOS personal computer required several steps. These were splitting the code into reasonable sized files, replacing the dynamic memory management code, writing low level interface routines, substituting certain constructs in a systematic way throughout the entire code, and setting up an overlay structure for the Microsoft linker. An explanation of each of these aspects is given in the following paragraphs. The result of the porting was a working, although limited in available storage, version of SPICE2G6 on a MSDOS PC. The changes required from the Unix version are given in detail in a set of "diff's" listed in Appendix C. The first step in the port to the PC was to make the files accessible and useable by the compiler. The source was copied to PC diskettes by using the KERMIT program on a PC connected to a computer running Unix. Next the source files were broken into modules small enough that they could be used with Microsoft FORTRAN. (The original Unix source contained several separate source files having names with ".f" extensions. These source files were often too large to be compiled directly by the Microsoft FORTRAN compiler. The source files that were too large were broken into two or more parts, separating at the start of subroutines. Each remaining file still contained many individual FORTRAN functions and subroutines.)

The dynamic memory management used in SPICE2G6 is internal to the FORTRAN code, rather than relying on the operating system, as would be done with a "c" language program. FORTRAN does not have memory management tools as a standard part of its libraries. Thus the needed routines had been written to work expediently, but only on large computers having virtual memories. The technique that had been used was to specify a large common block, containing only a single array, and then have subroutines to manage allocating variables within that array, using a function LOC that returns the address of its argument. This LOC function is not in standard FORTRAN, but is available with many

compilers, including MSFORTRAN for the PC. The common block declaration had to be included with almost every function and subroutine in the code. It had been "hard coded" to have a length of 200000 words in the Unix version. This is 1.6Mbytes, and far beyond a PC's capability.

The "fix" of the memory management problem for the PC involved several steps. First, all of the common block declarations "COMMON /BLANKC/..." were commented out and replaced by a MSFORTRAN "INCLUDE" instruction. The file to be included then contained the common block declaration and a parameter statement with its length. Thus different sizes could be tried by changing only this one file.

The second step was to change the code handling the common storage so that it could use the results of the LOC function in MSFORTRAN. (This is complicated by the segmented address space of the Intel 8088 architecture.)

The third step (which came about in order to fix "bugs" which occurred without it) was to order the variables in all the common blocks, and add dummy variables for padding when necessary, so that all variables start on word boundaries. Further, some local variables had to be placed into common blocks for the same reason.

The size of the array in COMMON/BLANKC/ was made one-tenth the value it had been under Unix. This allows 160Kbytes for dynamic storage on the PC, and this amount has been enough to handle quite large circuit simulations. Unfortunately, this memory has to be allocated whether used or not, so the resulting executable program needs to be run alone on the PC, not as a task under a controlling program (more complicated than the simple BATCH command).

The Unix version of SPICE2G6 contains a routine written in "c" which provides basic, low-level operations that are not available with the Unix f77 FORTRAN library. These functions had to be rewritten in MSFORTRAN. They were placed in a file, UNIX.FOR, which is listed in Appendix C. This file also contains a subroutine which was moved to it from one of the other FORTRAN source files in order to facilitate the setting up of the overlay structure.

The most significant change from the Unix version to the PC version of SPICE2G6 is the overlaying of sections of code to allow such a large program to run with 640Kbytes of real memory. Under most computers using the Unix operating system, virtual memory eliminates the need for overlaying. SPICE2G6, however, has been written so that it can be overlayed in a relatively straightforward manner, and there are no doubt other versions which use overlaying. Overlaying has been implemented with the Microsoft linker so that all the SPICE devices are retained while still allowing room for working storage for the circuit being analyzed.

Setting up the overlay mechanism with Microsoft FORTRAN and the Microsoft linker requires determining which sections of code are not needed in memory at the same time and then writing a command file for the linker to include this information. The files which are to remain in memory at all times are placed in the "root" of the overlay "tree." The large common block used by the local memory manager must be in the root, which is the critical size limitation of the code. The different analyses – ac, dc-transient, noise, etc., can be placed in different overlay "branches," as can the input and output routines. The latter is possible since SPICE2G6 stores up its output in the dynamically managed large common block, to be written to a file at the conclusion of the analyses. The resultant overlay structure is contained in file SPICE2G6.LNK, which is listed in Appendix C.

The final step of the porting of the code to run on a PC was actually an enhancement rather than just a change to allow a previous feature to work. The Unix version of SPICE2G6 contains binary write statements to output the results of the analyses, as they are in progress, to a file managed by one of the "c" routines, included in the file "unix.c." The PC implementation of this required writing this data to a FORTRAN unit (UNIT=15 was chosen) in binary mode. The problem was that the resulting output, while containing the information from all analyses from the SPICE run, was not accessible to the postprocessor program NUTMEG. For the data to be used by NUTMEG it first requires conversion from the "old binary format" output by SPICE2G6 to the new format understood by NUTMEG. This is accomplished by the program SCONVERT. However, the resulting new format file allows access to only the first analysis of the run. In order to make all analyses in a run accessible, the code in SPICE2G6 was changed to cause a new file to be opened, complete with the necessary header information, for each analysis in a run. The names for the raw data files are obtained from command line arguments given

on the PC, or from an interactive dialog. The changes in the source code needed to accomplish this consisted of calling for opening and closing of the raw data file in the correct places and for the writing of the header information to each new file when opened. This is the reason the header writing was moved to the file UNIX.FOR and placed in the root overlay. The details of these code changes may be seen by examining the "diff"s" in Appendix C.

At the conclusion of the porting process, a version of SPICE2G6 was running on a MSDOS PC with 640K memory and mathematics coprocessor. It was essentially identical to its Unix counterpart except for the limitation in working memory size.

2.2 Adding new circuit elements to SPICE2G6

Two new circuit elements were added to the code for SPICE2G6. These were a fuse element, based on the model by Ron Lee and used in his FIRESET program[6], and a saturating inductor element, based on a hyperbolic tangent model for saturation of magnetization[9]. These elements require the user to specify several data items in addition to the nodes. Further, the way they have been incorporated into the existing SPICE2G6 code is by making them special cases of already existing circuit elements. The details, which were different for each of the two devices, are described in the following.

2.2.1 Fuse element

The fuse model due to Ron Lee[6, pages 5–9] yields a value for the resistance between the two nodes of the element as a function of the action (integral of square of current with respect to time) of the current through the element. In addition, some of the parameters in the model are dependent on the initial rate of rise of the current when the fuse is placed in a simple RLC circuit. The equations of the model are[6, pages 7–8]:

$$\rho = A\left[1 - \tanh\left(\frac{g}{g_0}\right)\right] + B\exp\left[-\left(\frac{g-g_0}{s}\right)^2\right], \tag{2.1}$$

$$g_0 = G_0(V_0/KL)^P \qquad s = S_0(V_0/KL)^P. \tag{2.2}$$

Eq.2.1 gives the resistivity of the fuse in terms of four parameters:

A the resistivity after fusing

B the peak resistivity

g_0 the specific action to point of fusion

s the width of the resistivity peak.

The independent variable determining the resistivity ρ is the specific action g.

Specific action is the time integral of current density, just as resistivity is electric field intensity divided by current density. For use in SPICE, the geometry of the fuse element must also be supplied so that current and voltage information can be related to the normalized values of eq.2.1.

The constants G_0 and S_0, used in eqs.2.2, are material properties. The factor which multiplies each of them to get the parameters in eq.2.1 is determined by fitting to data for fuses placed in simple series RLC circuits. In such a circuit V_0 is the initial voltage on the capacitor while L is the inductance. The ratio of these is the initial rate of rise of current. K and P are parameters determined by fitting the data. (See the FIRESET paper for more details and typical values[6, pages 8–9].)

When the fuse is placed in a complex circuit, rather than a simple RLC one, it is not clear what value should be used in place of V_0/L. For the first implementation of the model in SPICE2G6, the dimensional equivalent of eq.2.1 was used, and the coefficients were calculated by hand for entry to the program. This allowed comparison of the results using SPICE to those using FIRESET. However,

a more robust method was needed for general circuit analysis. This more general method has been implemented in the version identified as SPICE2G6-SB2. Here the input line to SPICE for the fuse element must give the material constants A, B, G_0, S_0 in addition to the dimensions for the fuse element, which is assumed to be a rectangular box. The power P must also be input, as well as the constant K. In carrying out the calculations, the ratio of V_0/L is replaced dynamically by the following scheme:

- If the parameter P is omitted from the input line then the dynamic calculation of the V_0/L factor is not done and the parameter input as K is taken to be a constant scale factor to be used in place of the factor $[V_0/(KL)]^P$.
- When both K and P are included on the input line, then the code, contained in SUBROUTINE FUSE1 (which is listed in Appendix D), proceeds to replace V_0/L at each time step by CMAX/TMAX, where CMAX is the maximum magnitude of current through the element up to the present time, the time of the maximum being TMAX.

The code must update CMAX and TMAX at each step, and must also keep a numerical integration going for the action. Details of how this is accomplished may be seen in the code listing in Appendix D.

SPICE2G6-SB2 must be able to determine from the input line for a circuit element what routines to call to calculate currents and voltages. This was accomplished for the fuse element by making it a special case of a nonlinear voltage controlled current source. In the unmodified SPICE2G6, if the first parameter to a voltage controlled current source is POLY then the calculation proceeds as if the source function were a polynomial one. The modified SPICE2G6-SB2 has the input line tested to see if the first polynomial coefficient has the (unlikely otherwise) value of -1234. If so, it is taken to be a fuse element, the remaining parameters are the fuse parameters, and the correct subroutines are called to calculate current-voltage relationships. This interfacing to the original SPICE2G6 code required modifications to several code modules. These can be seen in the "diff's" given in Appendix D.

One additional feature was included with the fuse element. The FIRESET program writes out the information of the time when a fuse melts and also displays this point on graphical output. With the use of the NUTMEG program, the time of the fuse melting can be seen as a peak when the fuse resistance is plotted as a function of time on the same axes as the other variables of interest. To provide that information in the printed output, the program stores the time when the specific action first exceeds the value to melt, g_0, and then sets a flag to cause the routine doing the time stepping to print out the parameters at the time of the fuse's melting to the standard output file.

2.2.2 Saturating inductance

SPICE2G6-SB2 also contained a modified inductor circuit element which represented saturation of the magnetization of a ferromagnetic core as a hyperbolic tangent function. The equations of this model are as follows:

$$F = I_s(L_i - L_0)\left\{\tanh[(I/I_s)^N]\right\}^{1/N} + L_0 I, \tag{2.3}$$

where F is the effective flux linkage due to current I in the inductance, so that the voltage across the inductance is $V = dF/dt$, and where I_s is the current at the knee of the magnetization curve, L_i is the small signal inductance, L_0 is the inductance of the same winding with an air core, and N is the power that specifies the sharpness of the corner in the magnetization curve. (For more details on this model, see the discussion in section V of the previous report.[9])

The interface of this function to the original code for inductors in SPICE2G6 was made by treating the saturating inductance as a special case of a nonlinear inductor. SPICE2G6 allows nonlinear inductors to have their inductances given as polynomials in the current through them. In SPICE2G6-SB2, when the first polynomial coefficient of a nonlinear inductor has the (unlikely) value -1234 then the inductor is assumed to have the hyperbolic tangent model, and the following coefficients are the parameters of that model.

A few difficulties had to be worked out in the numerical calculations using eq.2.3. The derivative of the flux function must be calculated for use by SPICE. This involves a hyperbolic secant function, which in turn must be calculated as the inverse of a hyperbolic cosine. When the argument of the hyperbolic

cosine is too large a numerical overflow occurs. This condition has to be tested for and corrective action taken. In the same way, the raising of too small a number to a power causes an error in FORTRAN, so cases for small arguments must be tested for and handled separately. Details of how these numerical problems are handled may be seen in the listings given in Appendix D.

2.3 Testing SPICE

Any program as complicated as SPICE requires extensive testing to insure that the numbers it gives as output are meaningful. SPICE has been in use for a long time, and so has had extensive testing. In this work what remained to be tested was the correctness of the basic SPICE2G6 after porting to the PC, and the correctness of the implementation of the new circuit elements.

2.3.1 Testing SPICE2G6 on the PC

The SPICE2G6 code comes with a test input file, BENCHMARK.IN, and an accompanying output file, BENCHMARK.OUT. One tests the port of SPICE2G6 to a new computer system by using BENCHMARK.IN as the input file and comparing the output produced to that in BENCHMARK.OUT. This was done with the versions of SPICE2G6 on the VAX under VMS (the one from W. G. Magnuson, Jr., mentioned above), on a Unix system, and on the PC. The resulting output files contained identical results except for the differences attributable to different numerical precisions on the different computers.

The BENCHMARK.IN file contains several separate SPICE "jobs", separated by ".END" lines. Each of these produced separate raw data files on the implementations where the raw data output could not be suppressed. The raw data was not examined, but its output may have caused execution times to be increased.

The SPICE "jobs" in BENCHMARK.IN could not all be run on the demonstration version of the PSPICE program which was available to use, but those jobs which could be run on it were done so for comparison. Again the results were the same as in the BENCHMARK.OUT file.

The use of the same input for many different types of SPICE analyses on several different computers provided a way to compare computer speeds. A summary of these speed comparisons has been included in Appendix E.

2.3.2 Testing the fuse element implementation

The implementation of the fuse element was verified by using SPICE2G6-SB2 on the PC to analyze a circuit having the same components and parameters as used as an example for the FIRESET program. The parameters of this example are given in the paper by Lee[6, pages 11-13]. The corresponding input for SPICE2G6-SB2 is given in computer file TFUSE10.CIR, which also contains extensive comments explaining the input format. This file is included with the sample inputs near the end of the User's Guide, given in Appendix F. The output file produced when TFUSE10.CIR is run is listed in Appendix G (The output file contains a copy of the input.) The numerical values for the time of melt and the current through the fuse at that time agree to 1% with the values from FIRESET. For ease in comparing the results from SPICE to those from FIRESET, a plot of the current and voltage through the fuse versus time is shown in Fig.2.1. (This plot was produced by inputting the raw data file from SPICE to SCONVERT then NUTMEG. The "hardcopy" command of NUTMEG was used to output a file in Unix "plot" format. All this was done on the PC. The, using a Unix workstation, the "plot" format file was converted to "fig" format using "plot2fig." Then the "fig" file was edited to enhance the labeling using "xfig." Finally, "transfig" was used to provide a file to include with the LaTeXinput that typesets this report.)

A test was made of the dynamic adjustment of the scale factors for specific action to melt and peak resistivity by running the same problem using the fixed scale factor that is used in FIRESET. (This merely required changing the comments from one form of the fuse input line to the other in the sample input file.) The results from the fixed scale factor calculation were identical to within 0.1% to the results

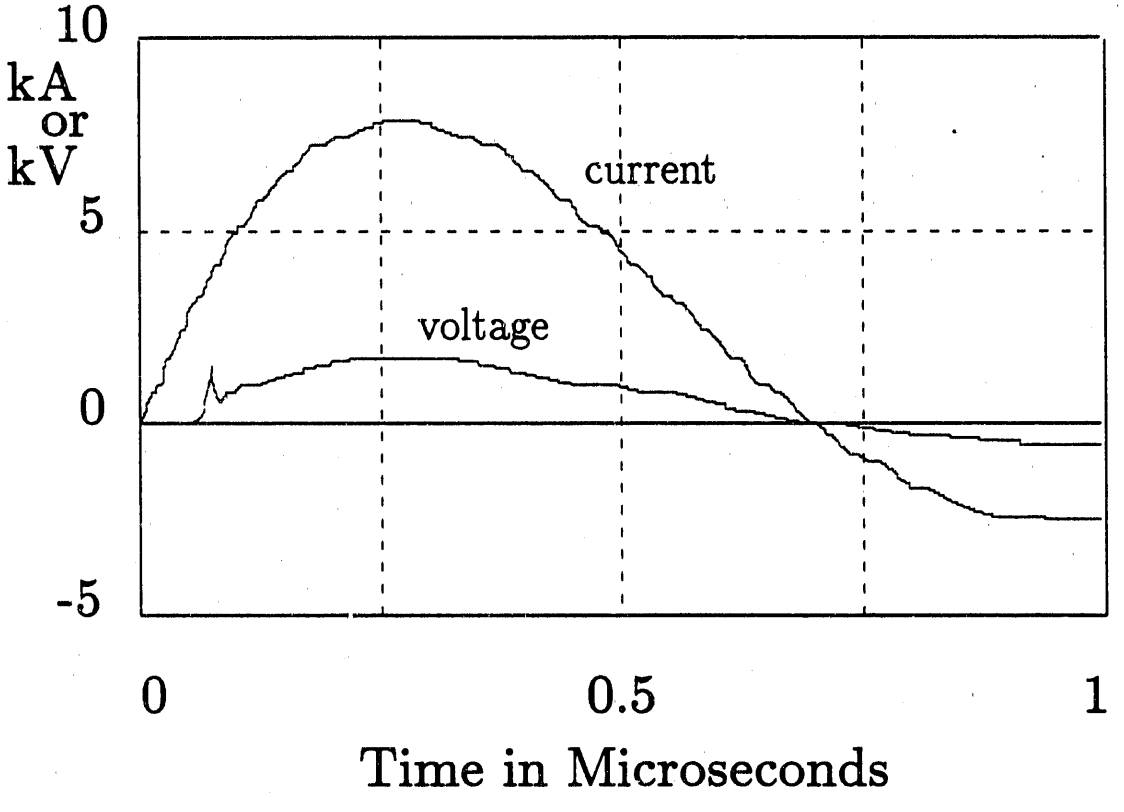


Figure 2.1. Plot of fuse current and voltage versus time from data of TFUSE10.CIR.

from the calculation using dynamic scaling. This close agreement is due to the melt in this example occuring early in the rise of the current, which causes V/L to be nearly equal to the ratio of current to time for all times prior to melt.

The input file for the test points out several problems that arise when using SPICE in this manner which had to be resolved in order to get satisfactory results. First, SPICE requires a direct current path to ground from every node in a circuit. One might think that a fuse would supply such a path, but the fact that the fuse is implemented as a nonlinear current source means that it appears as an open circuit on DC analysis. Thus the fuse must be paralleled with a large resistor (which has no effect on the results) to meet this requirement. Second, SPICE requires small time steps near times when parameter values are changing rapidly, as they do when a fuse melts. The dynamic choice of time steps can cause errors if limits are not set appropriately. Finally, SPICE2G6 has limitations which require one to add large resistors around reactive elements in some cases, and often experimentation must be done with error tolerances and step sizes to find values which allow solutions to proceed without error.

2.3.3 Testing the saturating inductor implementation

A test file LTANT1.CIR was created to use as SPICE input to check the saturating inductor implementation. This file is included with the sample inputs near the end of the User's Guide, given in Appendix F. The output file produced when LTANT1.CIR is run is listed in Appendix G (The output file contains a copy of the input.) The input file has extensive comments which explain how to use the saturating inductance element. Similar comments on choosing step sizes, placing large resistors in the circuits, and experimenting with various SPICE control parameters, apply to this case in the same way as is discussed for the fuse element in the preceding section.

The output of the test run with LTANT1.CIR as input has a qualitatively correct appearance. To give it a more specific test, the NUTMEG program was used on the raw data output (after conversion with SCONVERT) to plot the actual flux versus current for comparison to the theoretical values. (In order to make such a plot possible the flux was found in the circuit by integrating the voltage using a capacitor and a controlled current source.) Examination of the resulting plot showed that the circuit analysis was, in fact, giving correct output. The plot, produced with NUTMEG and edited with "xfig" is given in Figure 2.2.

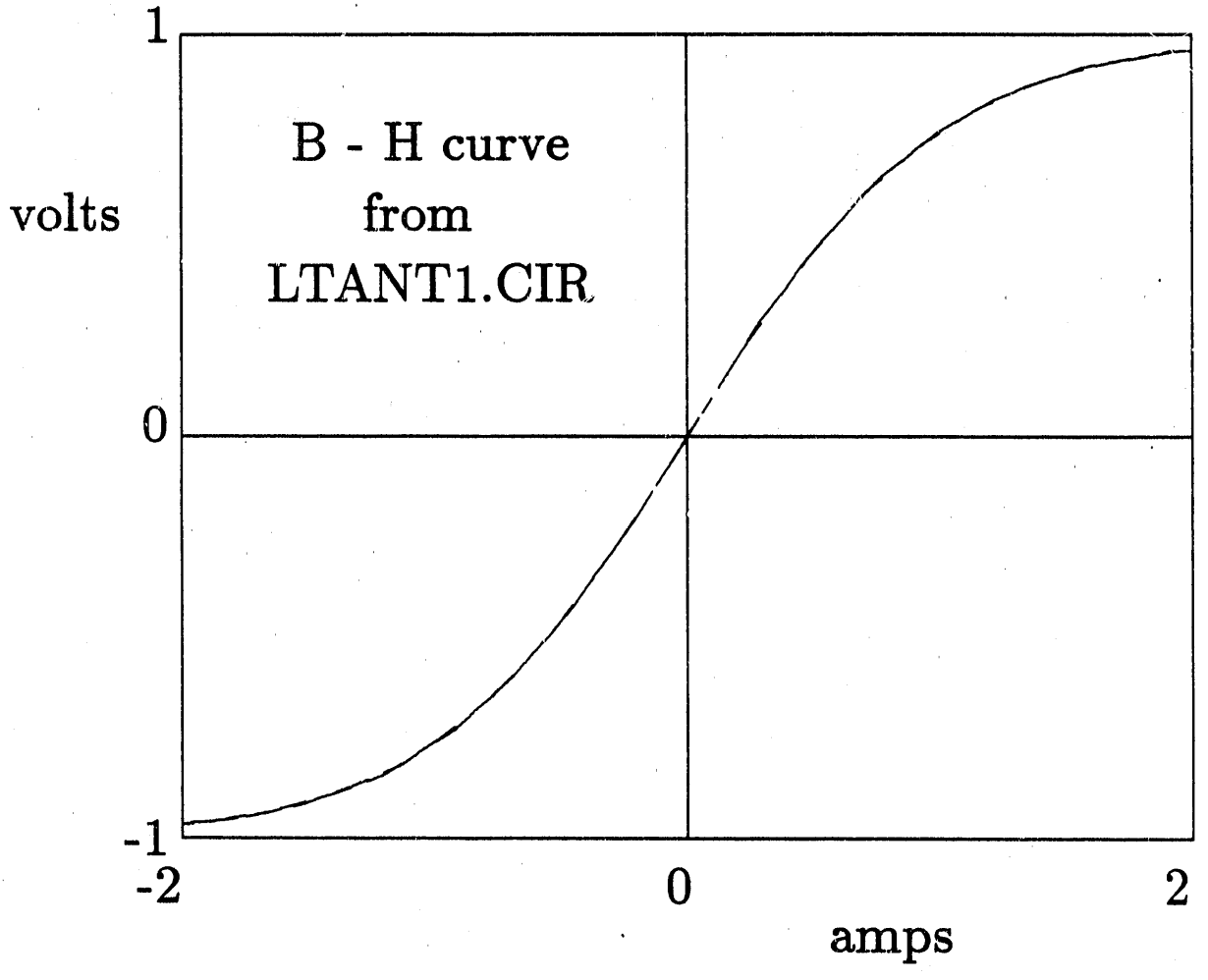


Figure 2.2. B-H curve simulation made with SPICE2G6 run on LTANT1.CIR and NUTMEG.

Chapter 3

Development of a PC user interface for SPICE

The SPICE circuit simulation code is capable of analysis of a great variety of circuits. With the enhancements made to allow fuse elements and saturating inductance elements, the possibilites for simulations are extended even further. Unfortunately, the user interface to this universe of simulation possibilities is quite minimal, especially for SPICE version 2. While SPICE version 3 has made interactive use and post-processing for graphics possible, the initial input file still must be a "net list" which has been created with an independent text editor. Since the system developed for firing system analysis makes use of parts of both SPICE version 2 and version 3, and since it is desirable for the firing system simulation code to be accessible to users who are not "experts" in the SPICE code itself, a flexible user interface has been designed and coded for use on personal computers running the MSDOS operating system (PC's).

The development of the user interface for SPICE involved three separate activites. The first, which has already been discussed in the preceding chapter, was to enable SPICE2G6 to output raw data files for each analysis made, and to insure that they could be converted, using the SCONVERT program, to a form suitable to input to the NUTMEG program. NUTMEG also had to be ported to run on a PC, and the bugs present in the port had to be found and fixed or circumvented. The second activity necessary for development of the user interface was the design and implementation of a menu-driven, interactive input-file editor for use with SPICE. This editor, called SPFEDIT, reduces the effort required to make up the "net list" for a circuit simulation, and allows changes to be made in existing SPICE input files with a minimum of effort. The third activity in the development of the software package was to produce a MSDOS batch control file which could integrate the separate executable programs so that they could be run as a single user command on the PC. This batch control file has been named SPDRIVER.BAT, so that one wishing to use SPICE on a PC equipped with this software package need only enter the command "SPDRIVER" and follow the instructions given with the menus. The final activity undertaken was not part of the development of the package itself, but was providing a way for automatically installing the package on a PC. Each of these activites is discussed in a following section.

3.1 SPFEDIT – A menu driven editor for creating and changing input files for SPICE

The SPICE program requires an input file, which contains a "net list," that is a set of instructions on how to form the circuit to be simulated, and also contains control lines to specify the analyses to be performed on the circuit. The SPICE program itself does not provide any editing facilites to be used to produce the input file, so one must use an independent editor to do it. (An editor may be run interactively from inside the main program of SPICE, version 3, on Unix operating systems, but the editor is still a separate program.) The syntax of the SPICE input file is simple, yet the number and

type of parameters varies with each different circuit element and control instruction, so that to write an input file for SPICE using a standard text editor requires constant reference to the User's Guide. Furthermore, the new fuse and saturating inductance elements that were introduced in SPICE2G6-SB2 require entry of several parameters having values that are not easily remembered. Thus a special purpose editor has been written that guides the user in the choice of parameters for SPICE elements and control instructions. This editor, named SPFEDIT, utilizes the windowing routines included with Borland's TurboC compiler[10] to provide a menu-driven dialog with the user. The entry of the fuse and saturating inductance elements is also simplified in that setting of many required parameters (to default values) is taken care of by the editor automatically. For all circuit elements and control instructions, standard default choices are provided for those parameters where they can be reasonably assumed. When nearly identical circuit elements are required, elements can be duplicated automatically. An on-line, context-sensitive "help" feature, and built-in consistency checking aids the user in writing valid input files without having to make continual reference to the written users' guides. While the editor cannot eliminate all errors the user might enter, it does prevent ones that could otherwise be troublesome, such as accidentally entering the same node twice for an element, or giving two elements the same name. At the conclusion of the menu-driven dialog the user has produced an input file that has the proper syntax and includes the minimally required control instructions. This file can be run with SPICE, and if errors occur, changes can be made easily by re-entering SPFEDIT again.

The SPFEDIT program is written entirely in the "c" programming language. Since functions in the TurboC library are used for producing the windows on the screen for the menus, the code is specialized to use on a PC. If porting to another operating system were desired, the windowing sections would have to be rewritten to use a library supported on that system. The program uses the dynamic memory management routines which are standard with "c" function libraries. This allows the SPFEDIT program to utilize as much as is needed of the PC's memory for storage of circuit information without having to specify a maximum circuit size in advance. The source code of the editor is listed in Appendix H.

The written documentation for SPFEDIT is provided by same file that is used to obtain the on-line help messages. This file, named SPFEDIT.HLP, is listed in Appendix H. The entries in the file are separated by lines beginning with a question mark. The word following the leading question mark is the key that allows the editor to find the help section appropriate for the context. The first section, headed by "?title", is the screen that is displayed when the program is started. The next section, headed "?begin", is an introduction to use of SPFEDIT. The other sections apply to specific menus. When a "help" item is too long to fit on one screen, one screen at a time is displayed, a prompt "–more–" given, and the next screen displayed when any key is pressed. While one can learn much about SPFEDIT by reading the "help" file, it is best to actually be using the program on a PC while viewing the help sections.

3.2 SPDRIVER – A batch driver for circuit simulations using SPICE

The software that has been written and collected for the purpose of simulation of firing systems is united by a control program called SPDRIVER. The structure of this set of software is described in the file SPDRIVER.DOC, which is listed in Appendix I. The MSDOS BATCH control file, SPDRIVER.BAT, was written to allow a user to execute all the separate programs of the circuit simulation software system through a single command to the PC. This control of the separate programs had to be done using BATCH rather than in a more sophistocated way, because of the memory limitations of PC's. The BATCH driver is built into the basic control program of the operating system for MSDOS. Thus it does not take up any extra memory. BATCH, in turn, runs each of the programs: SPFEDIT, SPICE2G6-SB2, SCONVERT, and NUTMEG, at appropriate times, with information being interchanged between them via disk files. The SPDRIVER program can have optional parameters given on its input line to bypass parts of the software that have been run before. Following the execution of the input file by SPICE, SPDRIVER runs DGREP (a public domain adaptation for PC's of a standard Unix utility) to find any error messages output by SPICE and writes them to the terminal. SPDRIVER also pauses

between major steps of the software operation to allow the user to control the progress. Because of the limitations of MSDOS BATCH, these controls outside of the programs themselves must be kept minimal. If one enters "SPDRIVER" on the PC, a short help message about its use is written on the terminal. A listing of the SPDRIVER.BAT file is given in Appendix J. The listing should be consulted for more details of SPDRIVER's operation.

3.3 The installation and use of the SPDRIVER system

Installation

In order for SPDRIVER.BAT to carry out its function, the separate programs and files making up the software system must be located in proper subdirectories in the MSDOS file system, and the proper PATH must be set in the "environment." This is accomplished if the software system is installed from the floppy diskette supplied as the software distribution medium, using the INSTALL.BAT batch control file included on the diskette. The file, README.INS, included on the diskette, should be printed and read before beginning the installation procedure. This file is listed in Appendix I. Some of the programs on the distribution diskette have been altered by a data compression and archiving program so that less disk space is required to store them. These files are restored to the forms needed for execution by other programs included on the distribution diskette. If an earlier version of SPICE has been installed on the PC system, the installation program will supercede it, provided the dialog questions in the INSTALL program are answered accordingly. After the installation has completed, the AUTOEXEC.BAT file must be changed by the user if the user wishes to have the PATH set automatically for subsequent use. The INSTALL.BAT file is listed in Appendix I. It can be consulted for more details of the installation procedure, and to see the various utility programs that are provided and utilized, in addition to the ones described above.

Use

When installation on the PC of the software system for circuit simulation has been completed, one begins its use by carrying out the following steps. First, if new to using SPICE, the user should become familiar with the User's Guide for SPICE2G6-SB2, which is included in this report as Appendix F. This Guide was modified from the one supplied by U.C.Berkeley[11, 12] in order to include the information on fuse elements and saturating inductances. In the process certain editoral changes and comments were added to make the guide easier to use and understand. If it is desired to use the NUTMEG program to view the raw data file and make general plots with it, then the documentation on that program, supplied with SPICE by U.C. Berkeley[7], should be reviewed. (The NUTMEG "man page" documentation has been reproduced here, for convenience, in Appendix K. The introduction to NUTMEG, supplied by U.C. Berkeley, has been edited to conform to usage on a PC. The edited version is given in Appendix L. Next, the circuit to be simulated should be sketched on paper, and the nodes numbered and elements labeled with names and values. Finally, a name should be chosen for this simulation problem, which can serve as a file name on the PC. Then one starts the software with the MSDOS command: "SDPRIVER name", where "name" is the one chosen for the project. The SPFEDIT program will be entered with the input file name set as "name.CIR", and the user can begin to enter the circuit, following the interactive instructions. A "man page" for SPDRIVER is given in Appendix K.

A sample SPICE input file is given for a problem using a fuse circuit element and for a problem using a saturating inductor circuit element. These files are listed at the end of the User's Guide in Appendix F. The (line printer) output SPICE produces for these inputs is given in Appendix G.

Due to the interactive and menu-driven nature of the SPFEDIT program, one cannot give a sample input/output in written form. The user is encouraged to experiment using the program and examining the output files produced, before making the first SPICE runs.

The computer codes comprising this software circuit simulation system based on SPICE provide a powerful tool for design of electrical networks containing fuse and saturating inductance elements as well as the standard elements of SPICE. Improvements will be made and additional elements added to

this system as experience in its use is gained. The author welcomes comments on these codes and their use, along with suggestings for further improvements and reports of difficulties or "bugs."

Chapter 4

Development of a simple model for gas discharge switches

Gas discharge switches are often used with firing systems. Thus one needs to be able to model such switches for simulation of a firing system circuit. A model for a gas discharge switch, where the resistance of the switch is given as a function of the action of the current through it, has been proposed by Martinez, Bryan, and Yactor[13], based on a fit to experimental data taken by the authors. While this model does fit the data they obtained, the model does not lend itself to use with codes such as SPICE due to infinities at the start. This could be remedied by modifying the dependence on action in a manner that would still retain the fit. However, rather than taking this approach, it was decided to try a much simpler model.

The simple gas discharge has essentially a constant potential drop across it, regardless of the current, provided the discharge has been started and that the current remains within limits. Thus the model for the switch, based on this concept, is that of an ideal switch in series with a constant voltage source. This model has been analyzed theoretically and calculations have been done with it to compare with the data in the paper referenced above. The details of this analysis were presented in a short paper sent privately to Ron Lee in February, 1990. This paper is reproduced in the following section. The short computer code used to make the calculations presented in the paper was written by Steve Warren, and is listed in Appendix M. The conclusion reached is that this simple approach to modeling a switch should be tried in simulation of firing systems, before introducing more complex models into SPICE.

4.1 Test of a switch model for RLC discharges

In a capacitor discharge through an inductance and resistance, the current can easily be calculated provided the discharge is initiated by an ideal switch. When the switch is not ideal, the non-linear characteristics of the switch must be taken into account, which makes the analysis more complex. Martinez, Bryan, and Yactor[13] have obtained experimental data on the effect of spark gap type switches on such RLC discharges. Their data shows a "switch voltage," which appears to be a constant regardless of circuit parameters and charging voltage, and a switch resistance proportional to the reciprocal of the square root of the action.

Since a simple model for a gas discharge is a constant voltage drop for all currents (once the discharge is initiated) and since this simple model retains linearity in the circuit, and since this model agrees with the "switch voltage" aspect of the experimental data, a calculation has been carried out to see if such a simple model can predict the experimental data as well as one that models switch resistance as proportional to inverse of square root of action. The conclusion is that a constant voltage drop across the switch can also fit the data reasonably well.

Consider the simple RLC circuit shown in Fig. 4.1. The switch is represented by a voltage of constant magnitude V_S for times $0 < t$ and by an open circuit for times $t < 0$. The initial value of voltage on

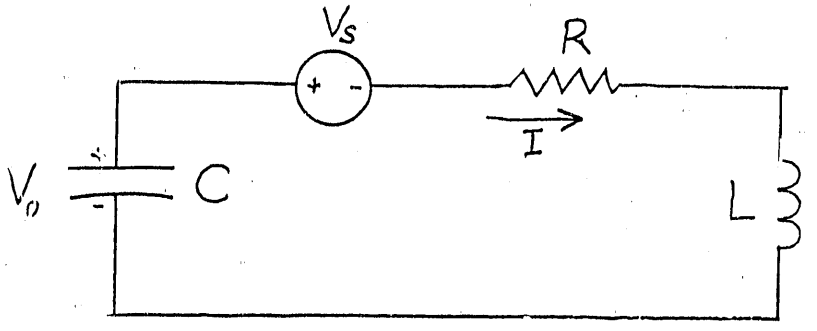


Figure 4.1. RLC circuit with constant voltage switch.

the capacitor is V_0. The solution of the differential equation for the current $i(t)$, for times $0 < t < T/2$, where T is the period of the oscillation (assuming the circuit is underdamped – if overdamped there is no upper limit for the time), is the same as if the switch voltage were absent and the capacitor initial voltage were reduced by V_S. If one defines the damping coefficient as $\alpha_0 = R/2L$ and the natural frequency as $\omega_0 = 1/\sqrt{LC}$, then the frequency of oscillation is $\omega_\tau = \sqrt{\omega_0^2 - \alpha_0^2}$. The solution for current for times $0 < t < T/2 = \pi/\omega_\tau$ is

$$I(t) = \frac{V_0 - V_S}{\omega_\tau L} e^{-\alpha_0 t} \sin(\omega_\tau t). \tag{4.1}$$

The maximum current occurs at a time slightly different from $t/4$ and has the value

$$I_{max} = (V_0 - V_S)\sqrt{\frac{C}{L}}\, e^{-\frac{\alpha_0}{\omega_\tau}\tan^{-1}\frac{\omega_\tau}{\alpha_0}}. \tag{4.2}$$

The expression for current in eq. 4.1 can be integrated to give the action as an analytic formula:

$$\begin{aligned} G = \int_0^t I^2(t')dt' \;=\; & \left(\frac{V_0 - V_S}{\omega_\tau L}\right)^2 \left\{ \frac{1}{4\alpha_0} - \frac{\alpha_0}{4(\alpha_0^2 + \omega_\tau^2)} \right. \\ & \left. + \frac{1}{4} e^{-2\alpha_0 t} \left[-\frac{1}{\alpha_0} + \frac{1}{\alpha_0^2 + \omega_\tau^2} \left\langle \alpha_0 \cos 2\omega_\tau t - \omega_\tau \sin 2\omega_\tau t \right\rangle \right] \right\}. \end{aligned} \tag{4.3}$$

The resistance of the switch in the model represented by these equations is $R_S = V_S/I(t)$. It will vary with time, and diverge rapidly near time zero and near time $T/2$. However, it will be nearly the same between these extremes of time as (a suitably chosen) constant divided by $\sqrt{G}$, which also diverges at time zero.

In order to make comparison of the values of R_S and $K/\sqrt{G}$ without introducing unnecessary parameter changes, the equations of the circuit can be normalized so that they depend on only two parameters. These normalized parameters are a normalized damping coefficient, $\alpha = \alpha_0/\omega_0$, and a normalized switch voltage, $v_s = V_S/V_0$. The normalizations that accomplish this parameterization are to normalize time to $\tau = \omega_0 t$ and to normalize current to $i(\tau) = I(t)\sqrt{L/C}/V_0$. (This latter normalization is actually impedance to $\sqrt{L/C}$ and voltage to V_0.) The action is correspondingly normalized to

$$g(\tau) = \frac{G(t)}{V_0^2 \frac{C}{L}\sqrt{LC}}.$$

By defining a normalized frequency, $\omega = \sqrt{1 - \alpha^2}$, the normalized solutions can be written as

$$i(\tau) = \frac{1 - v_s}{\omega} e^{-\alpha\tau} \sin(\omega\tau) \tag{4.4}$$

and

$$g(\tau) = \int_0^\tau i^2(\tau')d\tau = \frac{(1-v_s)^2}{\omega^2}\left\{\frac{1}{4\alpha} - \frac{\alpha}{4} + \frac{1}{4}e^{-2\alpha t}\left[-\frac{1}{\alpha} + \alpha\cos 2\omega\tau - \omega\sin 2\omega\tau\right]\right\}. \qquad (4.5)$$

If the resistance of the switch, R_S, were equal to $K/\sqrt{G}$ then one would have $K = R_S\sqrt{G}$ be a constant with time. The normalized equivalent of K is $k = \sqrt{g}v_s/i$ with normalizing factor $K = k[(LC)^{1/4}V_0]$. Instead, K will be a function of time. By calculating $k(\tau)$ and plotting it versus τ the two models for switch resistance can be compared.

Equations 4.4 and 4.5 have been programmed, and values of i, g, and k have been plotted on the same axes versus τ. The results for one case of α and v_s are shown in Fig. 4.2. One can see from the figure that k is not constant, but does not vary rapidly when away from time zero and time $T/2$.

One cannot compare the experimental data in the paper referenced above, directly with these calculations due to a lack of values for specific circuit parameters used with the experiment. However, one can compare the behavior of $\log(R_s)$ versus $\log(G)$ given in the experimental paper to that of $-\log(i)$ versus $\log(g)$. The experimental curve was somewhat noisy, but otherwise gave a straight line of slope $-1/2$ over the range shown. The calculated curve is shown in Fig. 4.3. Also on this figure is a line of slope $-1/2$ for comparison. The correct vertical position of the comparison line cannot be known without specific values of the voltage and attenuation factor in the experimental run. Note that a change in the normalized parameter v_s will only produce a vertical shift in this logarithmic plot. α must be changed to affect the slope.

In conclusion, it appears that one can model the effect of a spark gap switch in a firing system as a constant voltage drop, at least as a first approximation. This way of modeling a switch has the advantage that it can be used with existing circuit simulators without modification and with very little additional overhead in the calculations. It will be interesting to simulate firing circuits using both a simple voltage drop for a switch and a resistance modeled as the inverse square root of action to see if significant differences in burst time occur.

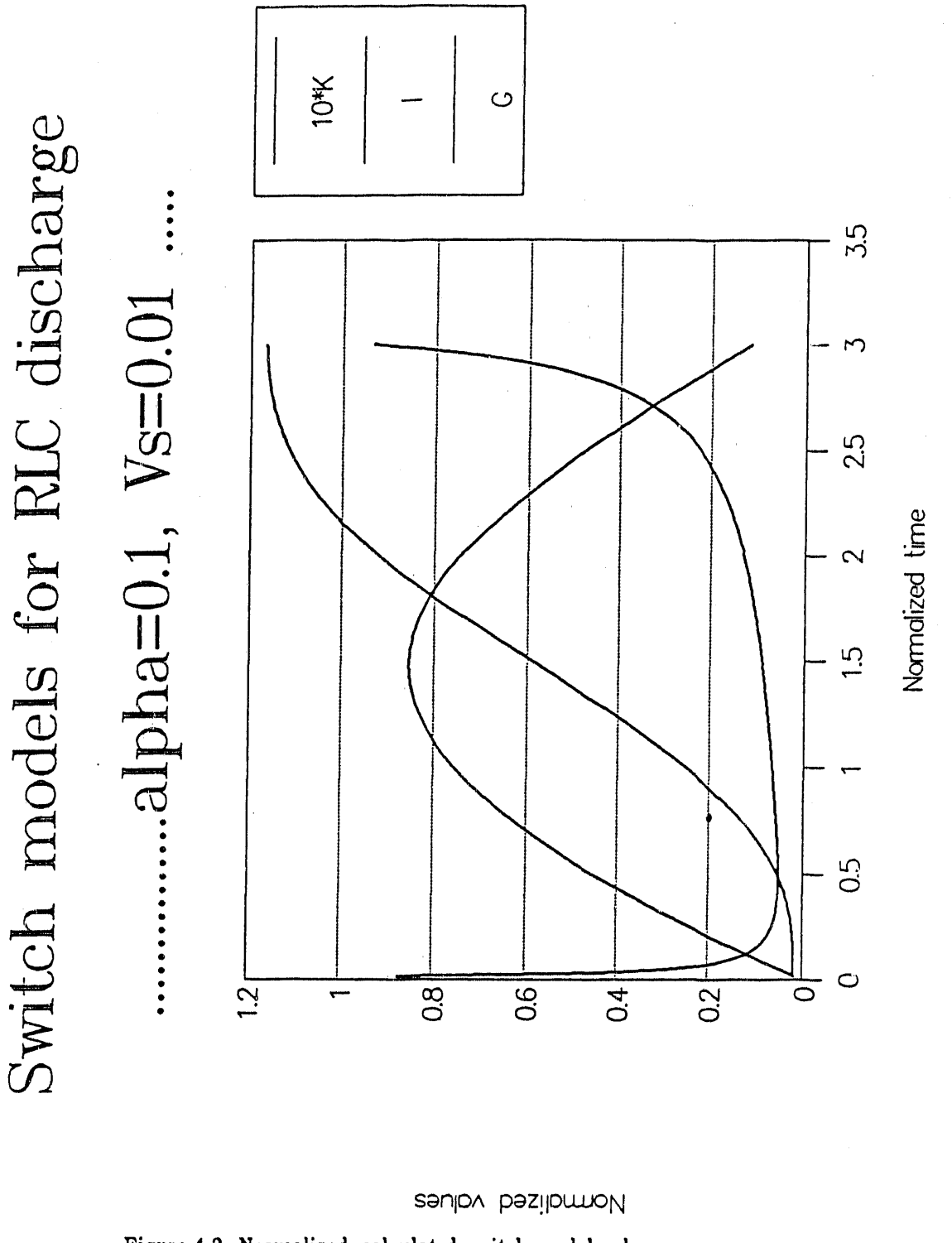


Figure 4.2. Normalized, calculated switch model values.

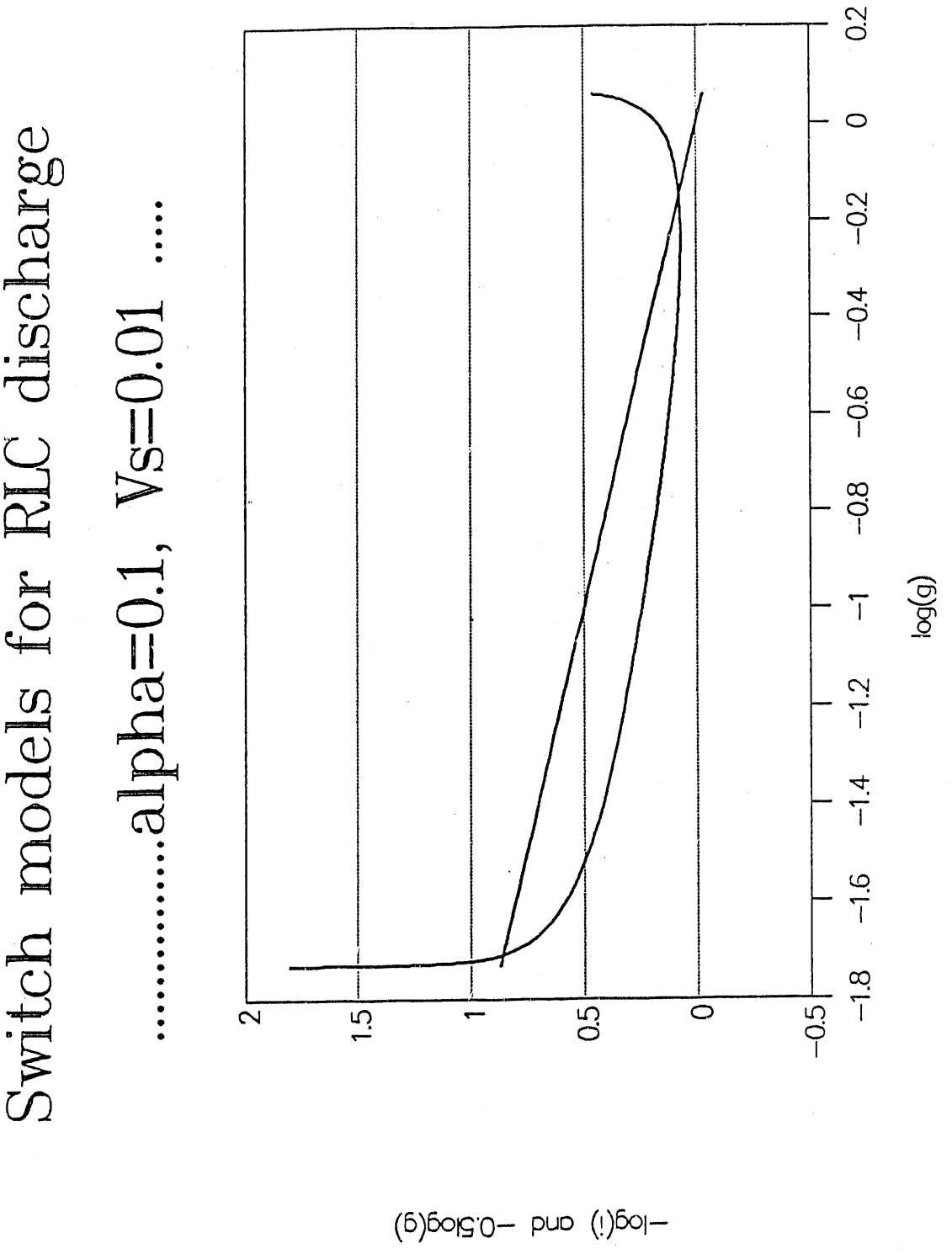


Figure 4.3. Log-log plot of resistance versus action.

Chapter 5

Study of ferrite properties under conditions of fast rise of magnetic field intensity

The saturating inductance model used in SPICE2G6-SB2 allows the effect of saturation of the core in a ferrite coupled transformer to be modeled by the program. However, in a practical firing system the current in a transformer has a complicated wave form, and dynamic effects in the core are likely to have an effect on its performance. Experimental measurements have been carried out to enable observation of the effect of fast changes in magnetic excitation of such cores. The first measurements were carried out by R. Boberg at Lawrence Livermore National Laboratory during 1987-88.[14] Boberg's measurements have been discussed in a previous report[9], and the attempts to fit his data with linear circuit elements, with or without saturation in the inductance, were not completely satisfactory. [It has recently been suggested[15] that the cause of some of the discrepancy between the measured current and the simulated ones might be due to skin effect in the CVR (current shunt) used in the measurements – but no further measurements have yet been made to clarify this point.] Due to the interest in having an accurate model for ferrite cored transformers, a systematic study of the effect of fast rise time pulses on ferrites has been undertaken.

During the summer of 1989, (Graduate Research Assistant) Steve Warren was employed directly by Lawrence Livermore National Laboratory to make measurments on a set of ferrite materials in the form of toroidal cores. The cores were excited by a few turns of wire connected to the output of a mercury pulser. The data was digitally recorded, using a fast response time Pearson probe to measure current. The data was subsequently reduced and examined for dynamic effects. The obvious effect is a delay in the response of the ferrite magnetization following the arrival of the exciting pulse. The details of this study have been reported in a paper given at the Joint XUSC/LFSC Meeting in November 1989.[16]. A written version of this paper is included in this report as Appendix N.

In order to reduce the digitized data from the measurements to the form presented in the paper in Appendix N it was necessary to write a computer program to interpret the data recorded by the Tektronix oscilloscope. This program output the data in a form suitable for input to the NUTMEG program, used to plot SPICE simulations. The program was named ADIFNUT, and a listing of it is included in Appendix O. The digitally recorded data included calibration information, but the format of the data was not well documented. The resulting code came about from "detective work" done to find the reference data in the files. (The converted data was actually plotted by using the Borland Quatro program rather than NUTMEG in order to get a better quality plots for the paper presented at the meeting.)

As noted in the conclusions in the paper in Appendix N, the magnetization is unable to follow a fast rise time pulse of magnetic field intensity, but is, in a qualitative sense, delayed in response. While a SPICE simulation of the measurement setup produces output in good agreement when the risetime

is below a threshold value, for faster risetimes, the simulation cannot follow the actual flux change in the ferrite. Further studies are being undertaken to quantify the nature of the response.

References

[1] *E*lectronic *C*ircuit *A*nalysis *P*rogram, IBM Corporation, White Plains, New York (circa 1970).

[2] Laurence W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memorandum No. ERL-M520, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, May 9, 1975.

[3] For example, PSpice (a registered trademark of) MicroSim Corporation, 20 Fairbanks, Irvine, California.

[4] Cited by Widner and Kiefer in their report on SCREAMER (see the following reference).

[5] M. M. Widner and M. L. Kiefer, "SCREAMER – A Pulsed Power Design Tool – User's Guide," Sandia National Laboratories, Albuquerque, New Mexico, April 25, 1985. (Privately distributed, not part of a technical report series as received, but may have been published later.)

[6] Ronald S. Lee, "Fireset," UCID-21322, Lawrence Livermore National Laboratory, Livermore, California, February 19, 1988.

[7] Software Distribution Office, Department of Electrical Engineering and Computer Sciences, Electronics Research Laboratory, 479 Cory Hall, University of California at Berkeley, Berkeley, CA 94720.

[8] Thomas L. Quarles, *Analysis of Performance and convergence Issues for Circuit Simulation*, a Ph.D. dissertation, with Appendices, UCB/ERL Memoranda M89/42, M89/43, M89/44, M89/45, M89/46, M89/47, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, April 24, 1989.

[9] Kenneth H. Carpenter, "Calculation of Mutual and Self Inductance and Coupling Coefficient for Closely Spaced Loops," Final Report for April 15, 1987, to April 15, 1988, to Dr. W. C. Von Holle, Lawrence Livermore National Laboratory; Department of Electrical and Computer Engineering, Kansas State University, Manhattan, Kansas, June 20, 1988.

[10] TURBO C, Version 1.5, Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA 95066.

[11] A. Vladimirescu, Kaihe Zhang, A. R. Newton, D. O. Pederson, A. Sangiovanni-Vincentelli, "SPICE Version 2G User's Guide," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, August 10, 1981.

[12] T. Quarles, and A. R. Newton, D. O. Peterson, A. Sangiovanni-Vincentelli, "SPICE 3B1 User's Guide," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, June 22, 1988.

[13] E. C. Martinez, C. S. Bryan, R. J. Yactor, "Evaluation of Switches for Use in Discharge Circuits," Paper presented at the Joint XUSC/LFSC Meeting, October 31 - November 2, Los Alamos National Laboratory, Los Alamos, NM.

[14] Ralph E. Boberg, Lawrence Livermore National Laboratory, private communication.

[15] Ronald S. Lee, private communication, May 1990.

[16] K. H. Carpenter, S. Warren, R. S. Lee, "An Empirical Study of Magnetization in Ferrites Excited by Fast Rise Time Pulese," Paper presented at the Joint XUSC/LFSC Meeting, October 31 - November 2, Los Alamos National Laboratory, Los Alamos, NM.

Appendix A

Feasibility study for a series-parallel circuit approach to firing system simulations

The following report is a reproduction of one submitted in May 1988.

BASIS FOR ANALYSIS OF A LADDER NETWORK OF CONSTANT INDUCTORS, NONLINEAR RESISTORS, AND A SINGLE CAPACITOR

Kenneth H. Carpenter

May 5, 1988

A ladder network consisting of N parallel and series branches containing inductance and resistance in each branch and having a single capacitor as the source of energy is shown in Fig. 1. It is assumed that the capacitance and inductances are constant values but the resistances may be arbitrary functions of the currents and actions (integrals of currents squared) of the network. The circuit equations for the ladder network may be written as

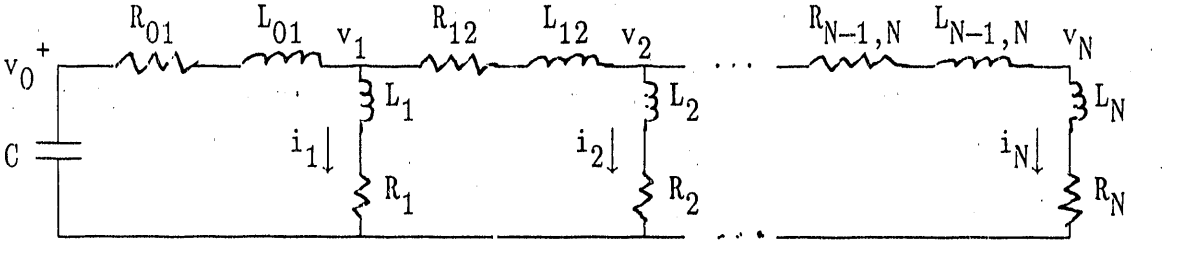


Fig. 1 Ladder circuit of resistances and inductances with source capacitor

$$v_k = R_k i_k + L_k \frac{di_k}{dt} \text{ , and}$$

$$v_k - v_{k+1} = R_{k,k+1} \sum_{p=k+1}^{N} i_p + L_{k,k+1} \sum_{p=k+1}^{N} \frac{di_p}{dt} \qquad (1)$$

which on eliminating the voltage variables gives

$$R_k i_k + L_k \frac{di_k}{dt} - R_{k+1} i_{k+1} - L_{k+1} \frac{di_{k+1}}{dt} = R_{k,k+1} \sum_{p=k+1}^{N} i_p + L_{k,k+1} \sum_{p=k+1}^{N} \frac{di_p}{dt}$$

where $k = 1,2...N-1$. (2)

Two more equations are needed to have a solvable set for the N currents and voltage v_0. These are

$$v_0 - R_1 i_1 - L_1 \frac{di_1}{dt} = R_{01} \sum_{p=1}^{N} i_p + L_{01} \sum_{p=1}^{N} \frac{di_p}{dt} \quad . \tag{3}$$

and

$$C \frac{dv_0}{dt} = -\sum_{p=1}^{N} i_p \, . \tag{4}$$

To obtain the complete set of differential equations needed to solve for the variables we make the definitions

$$x_0 = v_0 \text{ and } x_k = i_k \, , \; k = 1,2...N \tag{5}$$

along with the action variables x_k , $k = N+1,N+2...2N$ which satisfy

$$\frac{dx_k}{dt} = x_{k-N}^2 \, , \quad k = N+1,N+2...2N \, . \tag{6}$$

Thus the set of coupled, first order differential equations satisfied by the circuit becomes

$$\begin{bmatrix} C & 0 & 0 & \cdots & 0 & 0 \\ 0 & -(L_1+L_{01}) & -L_{01} & \cdots & -L_{01} & -L_{01} \\ 0 & L_1 & -(L_2+L_{12}) & & -L_{12} & -L_{12} \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & L_{N-1} & -(L_N+L_{N-1,N}) \end{bmatrix} \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_N \end{bmatrix} =$$

$$\begin{bmatrix} 0 & -1 & -1 & \cdots & -1 & -1 \\ -1 & (R_1+R_{01}) & R_{01} & \cdots & R_{01} & R_{01} \\ 0 & -R_1 & (R_2+R_{12}) & & R_{12} & R_{12} \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & -R_{N-1} & (R_N+R_{N-1,N}) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \qquad (7)$$

and

$$\begin{bmatrix} \dot{x}_{N+1} \\ \dot{x}_{N+2} \\ \dot{x}_{N+3} \\ \vdots \\ \dot{x}_{2N} \end{bmatrix} = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ \vdots \\ x_N^2 \end{bmatrix}. \qquad (8)$$

Equations (7) and (8) form the basis for the numerical solution for the currents. Once the values of capacitance and inductances are specified as constant values, the matrix on the left of eq.(7) can be inverted and stored. Then when the numerical differential equation solver calls for the time derivatives (the $\dot{x}$ values) to be computed, the R values can be computed as functions of the x values, then the R matrix multiplied by the inverted and stored matrix, and then the $\dot{x}$'s calculated by multiplying the resultant matrix times the supplied x values. The initial conditions are set as zero for all x_k values except for x_0 which has the initial capacitor voltage. Since the matrix inversion only has to be done once for each circuit the solution should be relatively fast. (If the inductances were functions of the currents then the matrix on the left of eq.(7) would have to be inverted on each call for the derivatives by the DE solver.)

Equations (7) and (8) above can be used as the basis for a ladder network circuit solution problem. The dimensions of the system of equations is just the number of parallel branches in the network (plus one). The resistances in each branch may be composed of a set of linear resistors in series with a set of nonlinear resistors. Even so, the storage requirements for reasonable sized circuits should be well within the capabilities of a personal computer.

Appendix B

Correspondence with U. C. Berkeley concerning use of the SPICE code

The following pages contain reproductions of

- A letter sent to the U.C.Berkeley, EECS/ERL Industrial Support Office, and returned by them along with:
- A letter from D. O. Pederson.
- The general information side of their order blank.

(The order blank was enclosed with a copy of the catalog of the Industrial Liaison Program - Public Domain Software, dated August 1987, which was also returned with the letter of inquiry.)

RECEIVED MAY 2 5 1988

KSU
KANSAS STATE UNIVERSITY

Department of Electrical and Computer Engineering
Durland Hall
Manhattan, Kansas 66506
913-532-5600

May 9, 1988

Cindy Manley
EECS/ERL Industrial Support Office
University of California
461 Cory Hall
Berkeley, CA 94720

Dear Ms Manly:

In January 1987, we ordered a tape from your office of SPICE3A7. Recently, we received a copy of SPICE3B1 from Lehigh University, which had been modified to allow it to run on an AT&T 3B15 computer. There are still some problems with the graphics interface in the versions we have, and before putting a significant effort into correcting them I wanted to find out if a later version than 3B1 is now available. Will you please send me information for ordering the most recent version of SPICE if it is newer than version 3B1?

I have another request for information regarding SPICE. The page of information, printed on your UC letterhead, about copying permissions for programs distributed by the Industrial Liaison Program states that "sale, resale, or use of the programs for profit" requires written permission from your department. I am doing research under subcontract from the University of California for Lawrence Livermore National Laboratory. As part of that research I need to write a computer program to do some specialized circuit analysis with circuit components (containing nonlinearities and hysteresis) significantly different from any currently included in SPICE. One approach for me to follow would be to modify the basic structure of SPICE to allow the type of elements needed and then interface the resulting structure to the desired input-output techniques. The resulting code would not be sold commercially, but would be distributed through the National Energy Software Center. It is not clear to me whether or not such a use of the SPICE software would require the written permission of your department, but to avoid any possible problem in that regard I will not use any of the SPICE code in my program unless I can get a written statement from the EECS Department of UC Berkeley that authorizes me to do so. Will you please let me know how to formally

request such a letter of permission? I would appreciate receiving the name and telephone number to call if there needs to be further communication from me in order to receive the letter of permission. Thank you for you help in this.

My telephone number is (913)532-5600 and my electronic mail addresses are:
khc@eecea.eece.ksu.edu (internet),
KHCARPEN AT KSUVM (bitnet), and
mfeee@ksv (mfenet).
I will look forward to hearing from you. Thanks again for your help.

Sincerely,

Kenneth H. Carpenter

Kenneth H. Carpenter
Professor

UNIVERSITY OF CALIFORNIA, BERKELEY

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO

SANTA BARBARA • SANTA CRUZ

COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCES

BERKELEY, CALIFORNIA 94720

June 1, 1984

To Whom It May Concern:

Programs SPICE have always been in the public domain. SPICE1 was released in May 1972 and the first issue of SPICE2 was releaseed in July 1975. We have never requested any royalties or rents with respect to the use of SPICE. When we issue a copy, we only make a small charge to cover handling and mailing.

Initially, we issued versions of the program which could be run only on CDC cyber-type computers. During the period 1980-81, we issued versions for CDC, IBM and VAX computers; however, this overwhelmed our staff who had to handle this task on an available-time basis. Presently, we issue only versions for VAX computers operating under the UNIX operating system. We have made arrangements with several other sources to distribute the program for other computers and operating systems. These sources supply the program as a professional courtesy. (Several companies have expressed a desire to help distribute SPICE as a "thank you" to us for making the program available.)

We have always stated that SPICE must not be sold. There should be no charge for the use of the program, except for computer usage, consulting services, etc. Several firms have made what they consider to be extensive modifications and/or additions to the program. These firms believe it proper to charge for these additions and modifications. Some firms make the modified programs available only on their own computer equipment and charge for the computer time which is used and a surcharge for the use of the modified program. Other firms have made the modified versions of the program available for outright sale. It is emphasized to us that charges are made only for their efforts; it is emphasized that there is no charge for the native SPICE code still residing in their versions. The cost for these programs varies from $15,000 to $100,000.

I hope that this note provides you with the information you need.

D. O. Pederson
Professor and Chairman,
Department of Electrical Engineering
and Computer Sciences

ORDERING INFORMATION

1. Remove the enclosed ordering form and fill it out.
2. Specify clearly which version you want.
3. Consult each information sheet for prices of documentation and programs and special licensing/distribution restrictions.
4. Mail the completed form with your payment (check or money order) to the address indicated on the order form.

GENERAL INFORMATION

Source codes and copyright information: The source codes for the programs distributed by the ILP are all in the public domain, except for the **1986 VLSI Tools, Ditroff/Gremlin,** and **Smalltalk,** which have special licensing requirements. The public domain software programs require no license to obtain a tape with the code. Almost all the source codes shipped from Berkeley recently have had a copyright notice placed on them that gives the University ownership of the code. The notices, which are part of a standard template used to start each file of code, are placed on the tape at the request of the professor in charge. All student authors include their names on what they have written. All the programs are available free of charge to any interested party; the small charge covers the cost of materials and handling only. The sale, resale, or use of the programs for profit without the express written consent of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, is forbidden.

Redistribution: Unless otherwise indicated, you are free to redistribute programs to other U.S. organizations without University permission. The ILP will not distribute tapes outside the United States. Should you wish to distribute these programs outside the U.S., it is your responsibility to ensure compliance with all government regulations restricting the export of high technology.

ERL memoranda: The ERL Memoranda distributed by the ILP are generally in the public domain. You and your staff may reproduce and circulate these reports for your own purposes. However, individual graduate students sometimes reserve the rights to their reports for a year; we recommend that you check the front page of the report for a cop ght. After the year has elapsed, you are free to reproduce the report.

Correspondence on technical matters: The Industrial Liaison Program is not a technical services office. The Department and the ERL do not have the resources to offer technical support, although the graduate student authors do appreciate being notified of bugs. If you have a technical question or bug to report, you may send a brief description of the problem *in writing* to the Industrial Liaison Program. Please include as a minimum the following items:

- Version of the program suffering the problem.
- Where you got your version.
- Whether you have made any local modifications.
- An example input exhibiting the problem **(as short as possible).**
- The resulting output from your input.
- The machine you are running the program on.

We will forward the letter to the appropriate professor or graduate student. However, we cannot promise any response, nor even an acknowledgement of receipt of your letter.

Refunds: There are no refunds on documentation. Defective tapes may be returned for a refund provided they are received by our office within 60 days of the date postmarked from UC Berkeley.

Appendix C

Source code differences between Unix and MSDOS versions of SPICE2G6

The following pages list the differences required when the Unix FORTRAN code for SPICE2G6 is ported to run on an MSDOS PC, and compiled using Microsoft FORTRAN, version 4.01. (The file UNIX.C had to be replaced entirely by the file UNIX.FOR, which is also listed.) In addition a listing is given of a README file,which contains information on how to use these difference files along with the Unix distribution to obtain the MSFORTRAN version. Also listed are the "make" and linker-response files needed to automatically compile the code on a PC.

The files of differences were made using the Unix "diff" program, context difference option. The files are listed in the following order. See the README listing first for more information.

```
README
blankc.for
unix.for
acan.f.diff
dcop.f.diff
dctran.f.diff
errchk.f.diff
ovtpvt.f.diff
readin.f.diff
setup.f.diff
spice.f.diff
spice2g6.lnk
spice2g6.mak
```

README - K.H.Carpenter - EECE Dept., Kansas State University - 06JUN90
  Instructions for making a version of SPICE2G6 that will
  compile and run using Microsoft Fortran (v4.01) on a 640Kbyte MSDOS PC.

1. Obtain a copy of the Unix version of SPICE2G6 -- at this writing
   the copy used for the "diffs" given here could be gotten via
   anonymous ftp over the internet to salmon.ee.ubc.ca (get file
   pub/vlsi/spice2g6.tar.Z). The ultimate source is UC Berkeley.
   For information write to EECS/ERL Industrial Liaison Program,
   479 Cory Hall, University of California, Berkeley, CA 94720.

2. (On a Unix system) apply the patches contained in the files included
   here, ending in .diff, to the files in the Unix version, using the
   "patch" program of Larry Wall (or by hand if "patch" is not available).
   (A version of "patch" is available for MSDOS pc's - with it the
   changes could be made on the PC entirely. But first be sure that the
   linefeeds from the Unix files are changed to CR-LF combinations.)

   The Unix version has files dctran1.f and dctran2.f. These need
   to be combined into a single file, dctran.f, before the patch
   dctran.f.diff is applied. The patched dctran.f will have to be
   split into five parts for use with MSFORTRAN.

3. Split the files spice.f, acan.f, dctran.f, errchk.f, and readin.f
   into smaller files, separating at the beginning of subroutines,
   to obtain files small enough for the MSFORTRAN program to accept.
   dctran.f must be split into five parts, dctran1.f,..., spice.f into
   three parts, while the others only need to be split into two parts each.
   The first line of each of the component files, at the recommended
   split points, is given below.

4. These split files, along with the other patched *.f files, along with
   the two files blankc.for and unix.for (which replaces unix.c in the
   Unix version) become the sources for the MSFORTRAN version. Convert
   any isolated linefeed characters in any of these files to CR-LF
   combinations before compiling with MSFORTRAN. Change the .f
   extensions to .for for all files.

5. Use the file SPICE2G6.MAK as the makefile for Microsoft "MAKE".
   The file SPICE2G6.LNK is used by it to drive the linker.
   The resulting SPICE2G6.EXE should be compacted using the
   Microsoft "EXEPACK" program. The file will still be too large
   for a 360Kbyte diskette, but can be compressed with an archive
   program, such as ZOO, ZIP, or ARC, to fit on one.

First lines of split files, at recommended separation points:

```
ACAN2.FOR:
      SUBROUTINE NOISE(LOCO)

DCTRAN2.FOR:
      SUBROUTINE LOAD

DCTRAN3.FOR:
      SUBROUTINE DIODE

DCTRAN4.FOR:
      SUBROUTINE MOSFET

DCTRAN5.FOR:
      SUBROUTINE MOSEQ1(VDS,VBS,VGS,GM,GDS,GMBS)

ERRCHK2.FOR:
      SUBROUTINE ELPRNT

READIN2.FOR:
      SUBROUTINE RUNCON(ID)

SPICE2.FOR:
      SUBROUTINE DCDCMP

SPICE3.FOR:
      SUBROUTINE CODGEN
```

```
CKHC     INCLUDE file for BLANK COMMON
CKHC     SPICE2G6-MS1  30AUG88  K.H.CARPENTER
C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
CKHC       COMMON /BLANK/ VALUE(200000)
      PARAMETER (MSMAX = 20000)
      COMMON /BLANK/ VALUE(MSMAX)
```

```
C UNIX.FOR - K.H.CARPENTER - 09SEP88
C
C The routines in this file are the MS FORTRAN replacements for the
C routines found in unix.c in the UNIX version of SPICE2G6.
C Also the PHEADR routine is moved here from DCTRAN1.FOR.
C
C*********************************
C    SUBROUTINE DBLSGL(CVAL,NUM)
C In this implementation of SPICE2G6 this routine is never used.  It
C was a dummy routine in unix.c.  The one call to it in the other code
C has been commented out.
C
C*********************************
      SUBROUTINE CLSRAW
C If the rawfile is closed via this call at the end of one analysis, and
C a second analysis occurs during the same job, then the unit (15) of the
C raw file will be opened again, with an interactive dialog.
C By making this a dummy routine, the raw data for the second analysis
C will be appended to that from the first.
C But, the input routines using the rawfile cannot work if a second analysis
C is appended, so this close must be used.  Supply enough rawfile names
C on the command line, or else supply them through the dialog.
      CLOSE(15)
      RETURN
      END
C
C*********************************
      SUBROUTINE XTIME(ATIME)
C     CHARACTER*(*) ATIME
      INTEGER*1 ATIME(8)
      INTEGER*1 LOCAL[NEAR](8)
      INTEGER*2 HR,MIN,SEC,HUN
      CALL GETTIM(HR,MIN,SEC,HUN)
      WRITE(LOCAL,10)HR,MIN,SEC
   10 FORMAT(I2,':',I2,':',I2)
      DO 20 K=1,8
   20 ATIME(K)=LOCAL(K)
      RETURN
      END
C
C*********************************
      SUBROUTINE XDATE(ADATE)
C     CHARACTER*(*) ADATE
      INTEGER*1 ADATE(8)
      INTEGER*1 LOCAL[NEAR](8)
CKHC     INTEGER*2 YR,MO,DAY
CKHC ADD UNNEEDED VALUE TO KEEP DATA ON 4 BYTE BOUNDARIES:
      INTEGER*2 YR,MO,DAY,JUNK
      CALL GETDAT(YR,MO,DAY)
      YR = MOD(YR,100)
CKHC USE JUNK TO KEEP OPTIMIZER FROM REMOVING IT:
      JUNK = YR
      WRITE(LOCAL,10)MO,DAY,YR
   10 FORMAT(I2,'/',I2,'/',I2)
      DO 20 K=1,8
   20 ADATE(K)=LOCAL(K)
      RETURN
      END
C
C*********************************
      SUBROUTINE FWRITE(VAL,NUM)
      INTEGER*2 VAL(1)
      WRITE(15) (VAL(K),K=1,NUM)
      RETURN
      END
C
C*********************************
      SUBROUTINE KHCCLEAR(D,N)
      INTEGER*1 D(1)
      DO 10 K=1,N
   10 D(K) = 0
      RETURN
      END
C
C*********************************
      SUBROUTINE KHCMOVE(FROM,TO,SIZE)
      INTEGER*1 FROM(1),TO(1)
      INTEGER SIZE
        IABSA=LOC(FROM(1))
      IHIG = IAND(IABSA,#FFFF0000)
      IABSA= IAND(IABSA,#FFFF)
      IABSA= IABSA + ISHFT(IHIG,-12)
        IABSB=LOC(TO(1))
      IHIG = IAND(IABSB,#FFFF0000)
      IABSB= IAND(IABSB,#FFFF)
      IABSB= IABSB + ISHFT(IHIG,-12)
      IF(IABSA.GE.IABSB)THEN
          DO 10 K=1,SIZE
   10     TO(K) = FROM(K)
      ELSE
          DO 20 K=SIZE,1,-1
   20     TO(K) = FROM(K)
      ENDIF
      RETURN
      END
C
C*********************************
      SUBROUTINE MOVE(ARRAY1, INDEX1, ARRAY2, INDEX2, LENGTH)
      INTEGER*1 ARRAY1(1),ARRAY2(1)
      CALL KHCMOVE(ARRAY2(INDEX2),ARRAY1(INDEX1),LENGTH)
      RETURN
      END
C
C*********************************
      SUBROUTINE ZERO4(ARRAY,LENGTH)
      CALL KHCCLEAR(ARRAY,LENGTH*4)
      RETURN
      END
      SUBROUTINE ZERO8(ARRAY,LENGTH)
      CALL KHCCLEAR(ARRAY,LENGTH*8)
      RETURN
      END
      SUBROUTINE ZERO16(ARRAY,LENGTH)
      CALL KHCCLEAR(ARRAY,LENGTH*8)
```

```
C Note that in this implementation, the 16 byte routines are
C the same as the 8.
      RETURN
      END
C
C*********************************
      SUBROUTINE COPY4(FROM,TO,LENGTH)
      CALL KHCMOVE(FROM,TO,LENGTH*4)
      RETURN
      END
      SUBROUTINE COPY8(FROM,TO,LENGTH)
      CALL KHCMOVE(FROM,TO,LENGTH*8)
      RETURN
      END
      SUBROUTINE COPY16(FROM,TO,LENGTH)
      CALL KHCMOVE(FROM,TO,LENGTH*8)
C Note that in this implementation, the 16 byte routines are
C the same as the 8.
      RETURN
      END
CKHC COMMON/CPHEAD/ INTRODUCED TO GET 4 BYTE ALIGNMENT ON IBUFF,ETC.
CKHC PHEADR MOVED HERE FROM DCTRAN1.FOR SO IT WILL BE IN ROOT
CKHC AND HENCE CALLABLE FROM ACAN AS WELL AS DCTRAN.
CKHC OPEN of unit 15 also moved here.
CKHC Output of plot title made more meaningful.  10SEP88
      SUBROUTINE PHEADR(AHEADR)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /CPHEAD/ IBUFF,INAMES,ITYPES,ISEQS
C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
      COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
     1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
     2   JUNODE,LSBKPT,NUMBKP,IORDER,JMNODE,IUR,IUC,ILC,ILR,NUMOFF,ISR,
     3   NMOFFC,ISEQ,ISEQ1,NEQN,NODEVS,NDIAG,ISWAP,IEQUA,MACINS,LVNIM1,
     4   LXO,LVN,LYNL,LYU,LYL,LX1,LX2,LX3,LX4,LX5,LX6,LX7,LDO,LD1,LTD,
     5   IMYNL,IMVN,LCVN,NSNOD,NSMAT,NSVAL,ICNOD,ICMAT,ICVAL,
     6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
     7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
C SPICE VERSION 2G.6  SCCSID=CIRDAT 3/15/83
      COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
     1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
      COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
     1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
     2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
C SPICE VERSION 2G.6  SCCSID=DC 3/15/83
      COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
     1   KINEL,KIDIN,KOVAR,KIDOUT
C SPICE VERSION 2G.6  SCCSID=MISCEL 3/15/83
      COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
     1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
CKHC      COMMON /BLANK/ VALUE(200000)
$INCLUDE:'BLANKC.FOR'
      INTEGER NODPLC(64)
      COMPLEX CVALUE(32)
C int3 (not used) is strictly for alignment.  f77 on Unix craps out.
      INTEGER*2 INT2,int3,NODPL2(128)
      EQUIVALENCE (VALUE(1),NODPL2(1))
      EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
      DIMENSION AHEADR(10)
C
C  PUT OUT THE HEADER RECORDS ONTO THE POST-PROCESSING FILE
C  ROUTINE IS USED FOR ALL ANALYSIS MODES (MODE=1,2,3)
C
      DIMENSION XTYPE(2)
CKHC
    DIMENSION HTRAN(2),HDC(2),HAC(2)
    DATA HTRAN /8H  Transi , 8Hent Anal /
    DATA HDC   /8H  DC Tra , 8Hnsfer An /
    DATA HAC   /8H  AC Ana , 8Hlysis    /
      DATA XTYPE /4HTIME,4HFREQ/
      DATA ABLNK,ALETV,ALETI /1H ,1HV,1HI/
CKHC
    DATA ACUR /2HI# /
C
C File structure for post-processor
C
C Record 1  Title card (80 bytes), date (8 bytes), time (8 bytes) TOTAL-96 BYTES
C Record 2  Number of output variables (including "sweep" variable)
C Record 3  Integer '4' (2 bytes)
C Record 4  Names of each output variable (8 bytes ea.)
C Record 5  Type of each output       0-no type
C                                     1-time
C                                     2-frequency
C                                     3-voltage
C                                     4-current
C                                     5-output noise
C                                     6-input noise
C                                     7-HD2     |
C                                     8-HD3     |
C                                     9-DIM2    }   distortion outputs
C                                    10-SIM2    |
C                                    11-DIM3    |
C Record 6  The location of each variable within each sweep point.
C           (Normally just 1,2,3,4,... but needed if outputs are mixed up)
C Record 6a 24 characters that are the plot title if Record 3 is a '4'.
C Record 7  Output at first sweep point
C Record 8  Output at second sweep point
C Record 9  .
C           .
C           .
C last record
C
C
CKHC
    OPEN(15,FILE=' ',FORM='BINARY')
      CALL GETM8(IBUFF,12)
      CALL COPY8(AHEADR(1),VALUE(IBUFF+1),10)
      VALUE(IBUFF+11)=ADATE
      VALUE(IBUFF+12)=ATIME
      CALL FWRITE(VALUE(IBUFF+1),48)
      NUMOUT=NUNODS+JELCNT(9)
C Force nused to be allocated by useless usage.
      int2 = NUMOUT
```

```
      int3 = NUMOUT
      INFO=4
      CALL GETM8(INAMES,NUMOUT)
      CALL GETM4(ITYPES,NUMOUT)
      CALL GETM4(ISEQS,NUMOUT)
      ITYPE2=ITYPES*2
      ISEQ2=ISEQS*2
      IKNT=1
      NODPL2(ISEQ2+1)=1
C
C DC Transfer curve (mode = 1):
C
      IF(MODE.NE.1) GO TO 10
      LOC=ITCELM(1)
      LOCV=NODPLC(LOC+1)
      VALUE(INAMES+1)=VALUE(LOCV)
      ANAM=ABLNK
      CALL MOVE(ANAM,1,VALUE(LOCV),1,1)
      ITYP=0
C Voltage transfer becomes type 3 and Current transfer becomes 4.
      IF(ANAM.EQ.ALETV) ITYP=3
      IF(ANAM.EQ.ALETI) ITYP=4
      NODPL2(ITYPE2+1)=ITYP
      GO TO 20
   10 VALUE(INAMES+1)=XTYPE(MODE-1)
      NODPL2(ITYPE2+1)=MODE-1
   20 DO 30 I=2,NUNODS
      NODPL2(ITYPE2+I)=3
      NODPL2(ISEQ2+I)=I
      VALUE(INAMES+I)=ABLNK
      IPOS=1
      CALL ALFNUM(NODPLC(JUNODE+I),VALUE(INAMES+I),IPOS)
   30 CONTINUE
      LOC=LOCATE(9)
      IKNT=NUNODS
   40 IF(LOC.EQ.0) GO TO 50
      IKNT=IKNT+1
      NODPL2(ITYPE2+IKNT)=4
      NODPL2(ISEQ2+IKNT)=IKNT
      LOCV=NODPLC(LOC+1)
CKHC       VALUE(INAMES+IKNT)=VALUE(LOCV)
CKHC Change here to give current and voltage different names
    ANAM = ACUR
    CALL MOVE(ANAM,3,VALUE(LOCV),1,6)
    VALUE(INAMES+IKNT)=ANAM
      LOC=NODPLC(LOC)
      GO TO 40
   50 INT2=NUMOUT
      CALL FWRITE(INT2,1)
      INT2=INFO
      CALL FWRITE(INT2,1)
      NWDS=NUMOUT*4
      CALL FWRITE(VALUE(INAMES+1),NWDS)
      CALL FWRITE(NODPL2(ITYPE2+1),NUMOUT)
      CALL FWRITE(NODPL2(ISEQ2+1),NUMOUT)
CKHC       CALL FWRITE(APROG(1),12)
      CALL FWRITE(APROG(1),4)
IF(MODE.EQ.3)CALL FWRITE(HAC(1),8)
IF(MODE.EQ.2)CALL FWRITE(HTRAN(1),8)
IF(MODE.EQ.1)CALL FWRITE(HDC(1),8)
  CALL CLRMEM(IBUFF)
  CALL CLRMEM(INAMES)
  CALL CLRMEM(ITYPES)
  CALL CLRMEM(ISEQS)
  RETURN
  END
```

```
*** ../spice2g6/acan.f  Tue Jun  2 14:19:52 1987
--- acan.f  Wed Jun  6 15:50:34 1990
***************
*** 1,3 ****
--- 1,4 ----
+ CKHC 09SEP88 - Placed IBUFF in a common block.
  C SPICE VERSION 2G.6  SCCSID=acan.ma 3/15/83
        SUBROUTINE ACAN
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
***************
*** 4,9 ****
--- 5,13 ----
  C
  C     THIS ROUTINE DRIVES THE SMALL-SIGNAL ANALYSES.
  C
+ CKHC THE FOLLOWING COMMON BLOCK HAS BEEN ADDED TO PUT WHAT WERE LOCALS
+ CKHC INTO A SEPARATE SEGMENT TO INSURE THEY WILL BE ON 4 BYTE BOUNDARIES
+   COMMON /KHCTAB/ITABLE,ITABID,IBUFF,ITEMP,LOCX,LOCY
  C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
        COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
       1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
***************
*** 36,42 ****
  C SPICE VERSION 2G.6  SCCSID=CJE 3/15/83
        COMMON /CJE/ MAXTIM,ITIME,ICOST
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        COMPLEX CENDOR
--- 40,47 ----
  C SPICE VERSION 2G.6  SCCSID=CJE 3/15/83
        COMMON /CJE/ MAXTIM,ITIME,ICOST
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        COMPLEX CENDOR
***************
*** 44,49 ****
--- 49,56 ----
        CALL SECOND(T1)
  C.. POST-PROCESSOR INITIALIZATION
        IF(IPOSTP.EQ.0) GO TO 1
+ CKHC
+   OPEN(15,FILE=' ',FORM='BINARY')
        NUMCUR=JELCNT(9)
        NUMPOS=NUNODS+NUMCUR
        CALL GETM16(IBUFF,NUMPOS)
***************
*** 114,120 ****
        CALL COPY16(CVALUE(LCVN+2),CVALUE(IBUFF+2),NUNODS-1)
        IF(NUMCUR.NE.0) CALL COPY16(CVALUE(LCVN+LOCCUR+1),
       1  CVALUE(IBUFF+NUNODS+1),NUMCUR)
!       CALL DBLSGL(CVALUE(IBUFF+1),NUMPOS)
        CALL FWRITE(CVALUE(IBUFF+1),NUMPOS)
  C
  C  NOISE AND DISTORTION ANALYSES
--- 121,128 ----
        CALL COPY16(CVALUE(LCVN+2),CVALUE(IBUFF+2),NUNODS-1)
        IF(NUMCUR.NE.0) CALL COPY16(CVALUE(LCVN+LOCCUR+1),
       1  CVALUE(IBUFF+NUNODS+1),NUMCUR)
! CKHC DBLSGL is just a dummy routine in this implementation
! CKHC      CALL DBLSGL(CVALUE(IBUFF+1),NUMPOS)
        CALL FWRITE(CVALUE(IBUFF+1),NUMPOS)
  C
  C  NOISE AND DISTORTION ANALYSES
***************
*** 208,214 ****
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPSO,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
  C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
        COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
--- 216,223 ----
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPSO,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
  C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
        COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
***************
*** 288,294 ****
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 297,304 ----
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 379,385 ****
        COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
       1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 389,396 ----
```

```
       COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
      1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1114,1120 ****
       COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
      1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1125,1132 ----
       COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
      1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1517,1523 ****
       COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
      1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1529,1536 ----
       COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
      1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1609,1615 ****
       COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
      1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1622,1629 ----
       COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
      1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1819,1825 ****
       COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
      1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1833,1840 ----
       COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
      1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
```

```
*** ../spice2g6/dcop.f  Tue Jun  2 14:19:53 1987
--- dcop.f  Wed Jun  6 15:52:33 1990
***************
*** 35,41 ****
        COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
       1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 35,42 ----
        COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
       1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 559,565 ****
        COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
       1   KINEL,KIDIN,KOVAR,KIDOUT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 560,567 ----
        COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
       1   KINEL,KIDIN,KOVAR,KIDOUT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 712,718 ****
        COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
       1   KINEL,KIDIN,KOVAR,KIDOUT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 714,721 ----
        COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
       1   KINEL,KIDIN,KOVAR,KIDOUT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1085,1091 ****
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1088,1095 ----
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
```

```
*** ../spice2g6/dctran.f    Wed Jun  6 14:52:28 1990
--- dctran.f    Wed Jun  6 14:53:17 1990
***************
*** 1,3 ****
--- 1,5 ----
+ CKHC 9SEP88 - Put IBUFF and ITEMP in a common block
+ CKHC 7SEP88 - Add close to rawfile after DC transfer output
  C SPICE VERSION 2G.6  SCCSID=dctran.ma 3/15/83
        SUBROUTINE DCTRAN
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
***************
*** 7,12 ****
--- 9,17 ----
  C AND TRANSIENT ANALYSES.  THE VARIABLES MODE AND MODEDC (DEFINED BELOW)
  C DETERMINE EXACTLY WHICH ANALYSIS IS PERFORMED.
  C
+ CKHC THE FOLLOWING COMMON BLOCK HAS BEEN ADDED TO PUT WHAT WERE LOCALS
+ CKHC INTO A SEPARATE SEGMENT TO INSURE THEY WILL BE ON 4 BYTE BOUNDARIES
+    COMMON /KHCTAB/ITABLE,ITABID,IBUFF,ITEMP,LOCX,LOCY
  C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
        COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
       1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
***************
*** 37,43 ****
  C SPICE VERSION 2G.6  SCCSID=CJE 3/15/83
        COMMON /CJE/ MAXTIM,ITIME,ICOST
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 42,49 ----
  C SPICE VERSION 2G.6  SCCSID=CJE 3/15/83
        COMMON /CJE/ MAXTIM,ITIME,ICOST
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 336,341 ****
--- 342,349 ----
        CALL RELMEM(ITEMPS,2)
        IF(IPOSTP.EQ.0) GO TO 1000
        CALL FWRITE(VALUE(IBUFF+1),NUMPOS)
+ CKHC
+       CALL CLSRAW
        GO TO 1000
  C
  C  ....  TRANSIENT ANALYSIS
***************
*** 662,803 ****
        RSTATS(LOCTIM)=RSTATS(LOCTIM)+T2-T1
        RETURN
        END
-       SUBROUTINE PHEADR(AHEADR)
-       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
- C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
-       COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
-      1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
-      2   JUNODE,LSBKPT,NUMBKP,IORDER,JMNODE,IUR,IUC,ILC,ILR,NUMOFF,ISR,
-      3   NMOFFC,ISEQ,ISEQ1,NEQN,NODEVS,NDIAG,ISWAP,IEQUA,MACINS,LVNIM1,
-      4   LXO,LVN,LYNL,LYU,LYL,LX1,LX2,LX3,LX4,LX5,LX6,LX7,LDO,LD1,LTD,
-      5   IMYNL,IMVN,LCVN,NSNOD,NSMAT,NSVAL,ICNOD,ICMAT,ICVAL,
-      6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
-      7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
- C SPICE VERSION 2G.6  SCCSID=CIRDAT 3/15/83
-       COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
-      1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
- C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
-       COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
-      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
-      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
- C SPICE VERSION 2G.6  SCCSID=DC 3/15/83
-       COMMON /DC/ TCSTAR(2),TCSTOP(2),TCINCR(2),ICVFLG,ITCELM(2),KSSOP,
-      1   KINEL,KIDIN,KOVAR,KIDOUT
- C SPICE VERSION 2G.6  SCCSID=MISCEL 3/15/83
-       COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
-      1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
- C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
-       COMMON /BLANK/ VALUE(200000)
-       INTEGER NODPLC(64)
-       COMPLEX CVALUE(32)
- C int3 (not used) is strictly for alignment.  f77 on Unix craps out.
-       INTEGER*2 INT2,int3,NODPL2(128)
-       EQUIVALENCE (VALUE(1),NODPL2(1))
-       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
-       DIMENSION AHEADR(10)
- C
- C  PUT OUT THE HEADER RECORDS ONTO THE POST-PROCESSING FILE
- C  ROUTINE IS USED FOR ALL ANALYSIS MODES (MODE=1,2,3)
- C
-       DIMENSION XTYPE(2)
-       DATA XTYPE /4HTIME,4HFREQ/
-       DATA ABLNK,ALETV,ALETI /1H ,1HV,1HI/
- C
- C File structure for post-processor
- C
- C Record 1  Title card (80 bytes), date (8 bytes), time (8 bytes) TOTAL-96 BYTES
- C Record 2  Number of output variables (including "sweep" variable)
- C Record 3  Integer '4' (2 bytes)
- C Record 4  Names of each output variable (8 bytes ea.)
- C Record 5  Type of each output       0-no type
- C                                     1-time
- C                                     2-frequency
- C                                     3-voltage
- C                                     4-current
- C                                     5-output noise
- C                                     6-input noise
- C                                     7-HD2    |
- C                                     8-HD3    |
- C                                     9-DIM2   }   distortion outputs
- C                                    10-SIM2   |
```

```
- C                                    11-DIM3   |
- C Record 6  The location of each variable within each sweep point.
- C           (Normally just 1,2,3,4,... but needed if outputs are mixed up)
- C Record 6a 24 characters that are the plot title if Record 3 is a '4'.
- C Record 7  Output at first sweep point
- C Record 8  Output at second sweep point
- C Record 9  .
- C           .
- C           .
- C last record
- C
- C
-       CALL GETM8(IBUFF,12)
-       CALL COPY8(AHEADR(1),VALUE(IBUFF+1),10)
-       VALUE(IBUFF+11)=ADATE
-       VALUE(IBUFF+12)=ATIME
-       CALL FWRITE(VALUE(IBUFF+1),48)
-       NUMOUT=NUNODS+JELCNT(9)
- C Force nused to be allocated by useless usage.
-       int2 = NUMOUT
-       int3 = NUMOUT
-       INFO=4
-       CALL GETM8(INAMES,NUMOUT)
-       CALL GETM4(ITYPES,NUMOUT)
-       CALL GETM4(ISEQS,NUMOUT)
-       ITYPE2=ITYPES*2
-       ISEQ2=ISEQS*2
-       IKNT=1
-       NODPL2(ISEQ2+1)=1
- C
- C DC Transfer curve (mode = 1):
- C
-       IF(MODE.NE.1) GO TO 10
-       LOC=ITCELM(1)
-       LOCV=NODPLC(LOC+1)
-       VALUE(INAMES+1)=VALUE(LOCV)
-       ANAM=ABLNK
-       CALL MOVE(ANAM,1,VALUE(LOCV),1,1)
-       ITYP=0
- C Voltage transfer becomes type 3 and Current transfer becomes 4.
-       IF(ANAM.EQ.ALETV) ITYP=3
-       IF(ANAM.EQ.ALETI) ITYP=4
-       NODPL2(ITYPE2+1)=ITYP
-       GO TO 20
-    10 VALUE(INAMES+1)=XTYPE(MODE-1)
-       NODPL2(ITYPE2+1)=MODE-1
-    20 DO 30 I=2,NUNODS
-       NODPL2(ITYPE2+I)=3
-       NODPL2(ISEQ2+I)=I
-       VALUE(INAMES+I)=ABLNK
-       IPOS=1
-       CALL ALFNUM(NODPLC(JUNODE+I),VALUE(INAMES+I),IPOS)
-    30 CONTINUE
-       LOC=LOCATE(9)
-       IKNT=NUNODS
-    40 IF(LOC.EQ.0) GO TO 50
-       IKNT=IKNT+1
-       NODPL2(ITYPE2+IKNT)=4
-       NODPL2(ISEQ2+IKNT)=IKNT
-       LOCV=NODPLC(LOC+1)
-       VALUE(INAMES+IKNT)=VALUE(LOCV)
-       LOC=NODPLC(LOC)
-       GO TO 40
-    50 INT2=NUMOUT
-       CALL FWRITE(INT2,1)
-       INT2=INFO
-       CALL FWRITE(INT2,1)
-       NWDS=NUMOUT*4
-       CALL FWRITE(VALUE(INAMES+1),NWDS)
-       CALL FWRITE(NODPL2(ITYPE2+1),NUMOUT)
-       CALL FWRITE(NODPL2(ISEQ2+1),NUMOUT)
-       CALL FWRITE(APROG(1),12)
-       CALL CLRMEM(IBUFF)
-       CALL CLRMEM(INAMES)
-       CALL CLRMEM(ITYPES)
-       CALL CLRMEM(ISEQS)
-       RETURN
-       END
        SUBROUTINE COMCOF
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
  C
--- 670,675 ----
***************
*** 821,827 ****
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 693,700 ----
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 920,926 ****
  C SPICE VERSION 2G.6  SCCSID=TRAN 3/15/83
        COMMON /TRAN/ TSTEP,TSTOP,TSTART,DELMAX,TDMAX,FORFRE,JTRFLG
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 793,800 ----
  C SPICE VERSION 2G.6  SCCSID=TRAN 3/15/83
        COMMON /TRAN/ TSTEP,TSTOP,TSTART,DELMAX,TDMAX,FORFRE,JTRFLG
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
```

```
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1025,1031 ****
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 899,906 ----
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1109,1115 ****
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 984,991 ----
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1302,1308 ****
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1178,1185 ----
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1384,1390 ****
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1261,1268 ----
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1475,1480 ****
--- 1353,1359 ----
        END
        SUBROUTINE LOAD
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
+ C      FOR INDUCTORS AND TO ALLOW BRIDGE ELEMENTS.
  C
  C     THIS ROUTINE ZEROES-OUT AND THEN LOADS THE COEFFICIENT MATRIX.
  C THE ACTIVE DEVICES AND THE CONTROLLED SOURCES ARE LOADED BY SEPARATE
***************
*** 1507,1513 ****
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1386,1393 ----
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1872,1878 ****
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1752,1759 ----
```

```
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2116,2122 ****
      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1997,2004 ----
      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2152,2158 ****
  C FICIENTS, AND LARG POINTS TO THE VALUES OF THE POLYNOMIAL ARGUMENT(S).
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 2034,2041 ----
  C FICIENTS, AND LARG POINTS TO THE VALUES OF THE POLYNOMIAL ARGUMENT(S).
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2307,2313 ****
      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 2190,2197 ----
      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2406,2412 ****
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 2290,2297 ----
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2612,2618 ****
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 2497,2504 ----
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 3173,3179 ****
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
       INTEGER NODPLC(64)
       COMPLEX CVALUE(32)
       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 3059,3066 ----
      1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
      2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
       INTEGER NODPLC(64)
```

```
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 3532,3538 ****
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 3419,3426 ----
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 5097,5103 ****
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 4985,4992 ----
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
```

```
49c49,50
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
675c676,677
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
719c721,722
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
770c773,774
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
871c875,876
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
930c935,936
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
984c990,991
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1103c1110,1111
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1202c1210,1211
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1269c1278,1279
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1701c1711,1712
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
2324c2335,2336
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
2328a2341,2343
> CKHC THE FOLLOWING COMMON BLOCK HAS BEEN ADDED TO PUT WHAT WERE LOCALS
> CKHC INTO A SEPARATE SEGMENT TO INSURE THEY WILL BE ON 4 BYTE BOUNDARIES
>   COMMON /KHCTAB/ITABLE,ITABID,IBUFF,ITEMP,LOCX,LOCY
```

```
0a1
> CKHC 9/10/88
8a10,12
> CKHC THE FOLLOWING COMMON BLOCK HAS BEEN ADDED TO PUT WHAT WERE LOCALS
> CKHC INTO A SEPARATE SEGMENT TO INSURE THEY WILL BE ON 4 BYTE BOUNDARIES
>    COMMON /KHCTAB/ITABLE,ITABID,IBUFF,ITEMP,LOCX,LOCY
11,12c15,16
<       INTEGER PSIG(3,20)
<       COMMON /PSIG/ PSIG
---
> CKHC      INTEGER PSIG(3,20)
> CKHC      COMMON /PSIG/ PSIG
47c51,52
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
109c114
<       WRITE(10,8886) PSIG(1,K),PSIG(2,K),PSIG(3,K),YVAR(K),XVAR
---
> CKHC      WRITE(10,8886) PSIG(1,K),PSIG(2,K),PSIG(3,K),YVAR(K),XVAR
277c282,283
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
368,369c374,375
<       INTEGER PSIG(3,20), OURBLK
<       COMMON /PSIG/ PSIG
---
> CKHC      INTEGER PSIG(3,20), OURBLK
> CKHC      COMMON /PSIG/ PSIG
398c404,405
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
443,446c450,453
<       PSIG(1,I) = OURBLK
<       PSIG(2,I) = OURBLK
<       PSIG(3,I) = OURBLK
<       CALL MOVE(PSIG(1,I),1,STRING,IPOS0,IPOS-IPOS0)
---
> CKHC      PSIG(1,I) = OURBLK
> CKHC      PSIG(2,I) = OURBLK
> CKHC      PSIG(3,I) = OURBLK
> CKHC      CALL MOVE(PSIG(1,I),1,STRING,IPOS0,IPOS-IPOS0)
503c510,511
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
591c599,600
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
844a854,856
> CKHC THE FOLLOWING COMMON BLOCK HAS BEEN ADDED TO PUT WHAT WERE LOCALS
> CKHC INTO A SEPARATE SEGMENT TO INSURE THEY WILL BE ON 4 BYTE BOUNDARIES
>    COMMON /KHCTAB/ITABLE,ITABID,IBUFF,ITEMP,LOCX,LOCY
877c889,890
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
```

```
53c53,54
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
893c894,895
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
959c961,962
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1619c1622,1623
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
1772c1776,1777
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
2083a2089,2098
> CKHC
> CKHC  IF FIRST CHARACTER IS * DON'T UPSHIFT - ALLOWS LOWER CASE
> CKHC  IN COMMENTS
> CKHC
> C write(5,999)line(1),line(1)/256
> C999  format(1x,a2,i8)
> C IF((LINE(1)/256).EQ.42)RETURN
> C  Above fails on PC - left most character is least sig 8 bits
> C  due to intel arch.  Use instead:
>   IF(IAND(LINE(1),255).EQ.42)RETURN
```

```
32c32,33
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
124c125,126
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
564c566,567
<       COMMON /BLANK/ VALUE(200000)
---
> CKHC      COMMON /BLANK/ VALUE(200000)
> $INCLUDE:'BLANKC.FOR'
```

```
*** ../spice2g6/spice.f Fri Oct  6 15:26:53 1989
--- spice.f Wed Jun  6 15:42:46 1990
***************
*** 1,3 ****
--- 1,18 ----
+ C SPICE VERSION SG6-KHC-MS0  9/10/88
+ C  SPICE2G6 for MSFORTRAN under MSDOS on 640K PC's.
+ C Modified from the Unix version to remove the plot output to
+ C   unit 10 (found at lines beginning with C UW/NW).
+ C   MSFORTRAN version has blank common specification placed
+ C   in an include file, BLANKC.FOR.
+ C   Input and output files are units 5 and 6 respectively,
+ C   and the specifications used to open them are the 1st and second
+ C command line arguments, respectively.
+ C   The rawfile is always written to the third command line argument.
+ C   It is in old binary format.
+ C See the MSFORTRAN manual "OPEN" entry for more information on
+ C how command line arguments are used.
+ C If a raw file or an output file are not wanted, they can be sent
+ C to the NULL device.
  C SPICE VERSION 2G.6  SCCSID=root.ma 3/15/83
        PROGRAM SPICE
  C...  CRAY NOTES:
***************
*** 73,81 ****
  C
  C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
  C UW/NW VLSI
!       INTEGER PSIG(3,20)
!       CHARACTER*14 MTPF
!       COMMON /PSIG/ PSIG

        COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
       1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
--- 88,96 ----
  C
  C SPICE VERSION 2G.6  SCCSID=TABINF 3/15/83
  C UW/NW VLSI
! CKHC      INTEGER PSIG(3,20)
! CKHC      CHARACTER*14 MTPF
! CKHC      COMMON /PSIG/ PSIG

        COMMON /TABINF/ IELMNT,ISBCKT,NSBCKT,IUNSAT,NUNSAT,ITEMPS,NUMTEM,
       1   ISENS,NSENS,IFOUR,NFOUR,IFIELD,ICODE,IDELIM,ICOLUM,INSIZE,
***************
*** 128,134 ****
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 143,150 ----
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 139,161 ****
        DIMENSION REMAIN(4)
        DATA ABLNK /1H  /
        DATA ACCTIT / 8HJOB STAT, 8HISTICS S, 8HUMMARY  , 8H         /
!       DATA AHDR1,AHDR2,AHDR3 / 8H  SPICE ,8H2G.6    ,8H3/15/83     /
  C
  C
!       IPOSTP=0
  C     CHECK IF A RAW DATA FILE SHOULD BE WRITTEN
  C     OPEN FILE IF SO REQUIRED
  C UW/NW VLSI
! C      CALL GETARG(1,MTPF)
  C SPECIFYING STATUS DOESN'T SEEM TO REALLY MATTER BUT THIS IS THE WAY IT
  C SHOULD BE IF EVERYTHING WORKED AS ADVERTISED
! C      OPEN(10,FILE=MTPF,STATUS='UNKNOWN')
!
! C      IF  (IARGC().GT.0) IPOSTP=IOPRAW()
!       MAXMEM=2*200000
        MAXTIM=1E8
        ICOST=0
        IOFILE=6
  C
  C  INITIALIZATION
  C
--- 155,184 ----
        DIMENSION REMAIN(4)
        DATA ABLNK /1H  /
        DATA ACCTIT / 8HJOB STAT, 8HISTICS S, 8HUMMARY  , 8H         /
! CKHC      DATA AHDR1,AHDR2,AHDR3 / 8H  SPICE ,8H2G.6    ,8H3/15/83     /
!       DATA AHDR1,AHDR2,AHDR3 / 8HSPICE2G6,8H-KHC-MS0,8H 9/10/88    /
  C
  C
! CKHC      IPOSTP=0
  C     CHECK IF A RAW DATA FILE SHOULD BE WRITTEN
  C     OPEN FILE IF SO REQUIRED
  C UW/NW VLSI
! CKHC      CALL GETARG(1,MTPF)
  C SPECIFYING STATUS DOESN'T SEEM TO REALLY MATTER BUT THIS IS THE WAY IT
  C SHOULD BE IF EVERYTHING WORKED AS ADVERTISED
! CKHC      OPEN(10,FILE=MTPF,STATUS='NEW')
! CKHC      IF  (IARGC().GT.0) IPOSTP=IOPRAW()
! CKHC      MAXMEM=2*200000
!       MAXMEM=2*MSMAX
        MAXTIM=1E8
        ICOST=0
        IOFILE=6
+ CKHC
+ CKHC  Now open the files, using the arguments from the command line:
+   OPEN(5,FILE=' ',STATUS='OLD')
+   OPEN(IOFILE,FILE=' ')
+ CKHC - MOVED TO ROUTINES WRITTING OPEN(15,FILE=' ',FORM='BINARY')
```

```
+   IPOSTP = 1
  C
  C  INITIALIZATION
  C
***************
*** 372,378 ****
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 395,402 ----
       1   GMIN,RELTOL,ABSTOL,VNTOL,TRTOL,CHGTOL,EPS0,EPSSIL,EPSOX,
       2   PIVTOL,PIVREL
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 634,640 ****
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 658,665 ----
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 789,795 ****
        COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
       1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
  C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
        COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
--- 814,821 ----
        COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
       1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
  C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
        COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
***************
*** 908,914 ****
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 934,941 ----
       1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 921,928 ****
  C
     30 IF (LEN.LE.80) GO TO 100
        WRITE (IOFILE,31) ADATE,APROG,ATIME,(ATITLE(I),I=1,10)
!    31 FORMAT(1H1,16(1H*),A8,1X,24(1H*),3A8,24(1H*),A8,16(1H*),//1H0,
!      1   15A8/)
        IF (ICOM.EQ.0) GO TO 40
        WRITE (IOFILE,36) COMENT,VALUE(ITEMPS+ITEMNO)
     36 FORMAT(5H0****,17X,4A8,21X,'TEMPERATURE =',F9.3,' DEG C'/)
--- 948,957 ----
  C
     30 IF (LEN.LE.80) GO TO 100
        WRITE (IOFILE,31) ADATE,APROG,ATIME,(ATITLE(I),I=1,10)
! CKHC   31 FORMAT(1H1,16(1H*),A8,1X,24(1H*),3A8,24(1H*),A8,16(1H*),//1H0,
! CKHC     1   15A8/)
!    31 FORMAT(1H1,16(1H*),A8,1X,20(1H*),3A8,8H-3/15/83,20(1H*),A8,
!      1 16(1H*),//1H0,15A8/)
        IF (ICOM.EQ.0) GO TO 40
        WRITE (IOFILE,36) COMENT,VALUE(ITEMPS+ITEMNO)
     36 FORMAT(5H0****,17X,4A8,21X,'TEMPERATURE =',F9.3,' DEG C'/)
***************
*** 932,938 ****
  C
  C
    100 WRITE (IOFILE,101) ADATE,APROG,ATIME,(ATITLE(I),I=1,10)
!   101 FORMAT(1H1,7(1H*),A8,1X,8(1H*),3A8,8(1H*),A8,5(1H*)//1H0,10A8/)
        IF (ICOM.EQ.0) GO TO 110
        WRITE (IOFILE,106) COMENT,VALUE(ITEMPS+ITEMNO)
    106 FORMAT(10H0****      ,4A8,' TEMPERATURE =',F9.3,' DEG C'/)
--- 961,969 ----
  C
  C
    100 WRITE (IOFILE,101) ADATE,APROG,ATIME,(ATITLE(I),I=1,10)
! CKHC  101 FORMAT(1H1,7(1H*),A8,1X,8(1H*),3A8,8(1H*),A8,5(1H*)//1H0,10A8/)
!   101 FORMAT(1H1,4(1H*),A8,1X,6(1H*),3A8,8H-3/15/83,6(1H*),A8,4(1H*)//
!      1  1H0,10A8/)
        IF (ICOM.EQ.0) GO TO 110
```

```
       WRITE (IOFILE,106) COMENT,VALUE(ITEMPS+ITEMNO)
    106 FORMAT(10H0****     ,4A8,' TEMPERATURE =',F9.3,' DEG C'/)
***************
*** 987,993 ****
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1018,1025 ----
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1272,1278 ****
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1304,1311 ----
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1374,1380 ****
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1407,1414 ----
  C SPICE VERSION 2G.6  SCCSID=DEBUG 3/15/83
        COMMON/DEBUG/ IDEBUG(20)
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1445,1451 ****
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1479,1486 ----
        COMMON /CIRDAT/ LOCATE(50),JELCNT(50),NUNODS,NCNODS,NUMNOD,NSTOP,
       1   NUT,NLT,NXTRM,NDIST,NTLIN,IBR,NUMVS,NUMALT,NUMCYC
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1492,1498 ****
       6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
       7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1527,1534 ----
       6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
       7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1677,1683 ****
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1713,1720 ----
        COMMON /FLAGS/ IPRNTA,IPRNTL,IPRNTM,IPRNTN,IPRNTO,LIMTIM,LIMPTS,
       1   LVLCOD,LVLTIM,ITL1,ITL2,ITL3,ITL4,ITL5,ITL6,IGOOF,NOGO,KEOF
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 1754,1760 ****
        COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
       1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
```

```
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 1791,1798 ----
        COMMON /AC/ FSTART,FSTOP,FINCR,SKW2,REFPRL,SPW2,JACFLG,IDFREQ,
       1   INOISE,NOSPRT,NOSOUT,NOSIN,IDIST,IDPRT
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2103,2109 ****
        COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
       1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 2141,2148 ----
        COMMON /MISCEL/ ATIME,APROG(3),ADATE,ATITLE(10),DEFL,DEFW,DEFAD,
       1  DEFAS,RSTATS(50),IWIDTH,LWIDTH,NOPAGE
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 2197,2203 ****
  C THE FIRST WORD OF THE BLOCK TABLE.
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
  C SPICE VERSION 2G.6  SCCSID=MEMMGR 3/15/83
        COMMON /MEMMGR/ CPYKNT,ISTACK(1),LORG,ICORE,MAXCOR,MAXUSE,MEMAVL,
       1   LDVAL,NUMBLK,LOCTAB,LTAB,IFWA,NWOFF,NTAB,MAXMEM,MEMERR,NWD4,
--- 2236,2243 ----
  C THE FIRST WORD OF THE BLOCK TABLE.
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
  C SPICE VERSION 2G.6  SCCSID=MEMMGR 3/15/83
        COMMON /MEMMGR/ CPYKNT,ISTACK(1),LORG,ICORE,MAXCOR,MAXUSE,MEMAVL,
       1   LDVAL,NUMBLK,LOCTAB,LTAB,IFWA,NWOFF,NTAB,MAXMEM,MEMERR,NWD4,
***************
*** 2268,2274 ****
--- 2308,2328 ----
       2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
        DIMENSION IVAR(1)
        IABSA=LOC(IVAR)
+ CKHC Following three lines needed with MSFORTRAN:
+    IHIG = IAND(IABSA,#FFFF0000)
+    IABSA= IAND(IABSA,#FFFF)
+    IABSA= IABSA + ISHFT(IHIG,-12)
        LOCF=IABSA/4
+ cdebug
+ cdebug     write(*,9)iabsa,iabsa
+ cdebug9    format(' LOC gives ',I12,2X,Z8)
+ CKHC With Microsoft FORTRAN and an MSDOS operating system, the
+ CKHC separate common blocks all are located on paragraph boundaries
+ CKHC by the linker.  When the .EXE file is loaded into real
+ CKHC memory, the program appears to start on an address divisible by 4.
+ CKHC Hence the following test will fail only for local variables that
+ CKHC may not be properly aligned (or in the event of a load not on a
+ CKHC paragraph boundary - which seems unlikely).
        IF (IABSA.EQ.4*LOCF) RETURN
        WRITE(IOFILE,100) IABSA
    100 FORMAT ('0*ERROR*: SYSTEM ERROR, ADDRESS ',I10,
***************
*** 2599,2605 ****
  C...  MEMORY NEEDS EXCEED MAXIMUM AVAILABLE SPACE
    300 WRITE (IOFILE,301) MAXMEM,MAXMEM
    301 FORMAT('0*ERROR*:  MEMORY REQUIREMENT EXCEEDS MACHINE CAPACITY',
!      1 /'0 MEMORY NEEDS EXCEED',I6,'(',O6,'B)'/)
        GO TO 900
  C...    *ISIZE* < 0
    410 WRITE(IOFILE,411)
--- 2653,2660 ----
  C...  MEMORY NEEDS EXCEED MAXIMUM AVAILABLE SPACE
    300 WRITE (IOFILE,301) MAXMEM,MAXMEM
    301 FORMAT('0*ERROR*:  MEMORY REQUIREMENT EXCEEDS MACHINE CAPACITY',
!      1 /'0 MEMORY NEEDS EXCEED',I6,'(',Z,'hex.)'/)
! CKHC     1 /'0 MEMORY NEEDS EXCEED',I6,'(',O6,'B)'/)
        GO TO 900
  C...    *ISIZE* < 0
    410 WRITE(IOFILE,411)
***************
*** 2947,2953 ****
       6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
       7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 3002,3009 ----
       6   LOUTPT,LPOL,LZER,IRSWPF,IRSWPR,ICSWPF,ICSWPR,IRPT,JCPT,
       7   IROWNO,JCOLNO,NTTBR,NTTAR,LVNTMP
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 3092,3098 ****
  C FACT THAT ONLY ONE POINTER IS ALLOWED PER TABLE.
  C
```

```
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 3148,3155 ----
  C FACT THAT ONLY ONE POINTER IS ALLOWED PER TABLE.
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 3112,3118 ****
  C FACT THAT ONLY ONE POINTER IS ALLOWED PER TABLE.
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
!       COMMON /BLANK/ VALUE(200000)
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
--- 3169,3176 ----
  C FACT THAT ONLY ONE POINTER IS ALLOWED PER TABLE.
  C
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
! CKHC      COMMON /BLANK/ VALUE(200000)
! $INCLUDE:'BLANKC.FOR'
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 3127,3134 ****
        END
        SUBROUTINE SECOND(T1)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
!       DIMENSION IBUFF(4)
!       CALL TIMES (IBUFF)
!       T1=DFLOAT (IBUFF(1))/60.0D0
        RETURN
        END
--- 3185,3195 ----
        END
        SUBROUTINE SECOND(T1)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
! CKHC      DIMENSION IBUFF(4)
! CKHC      CALL TIMES (IBUFF)
! CKHC      T1=DFLOAT (IBUFF(1))/60.0D0
!   INTEGER*2 HOUR,MIN,SEC,HUND
!   CALL GETTIM(HOUR,MIN,SEC,HUND)
!   T1 = SEC + 60.D0*(MIN + 60.D0*HOUR)
        RETURN
        END
```

```
SPICE1.OBJ+SPICE2.OBJ+SPICE3.OBJ+
(ACAN1.OBJ+ACAN2.OBJ)+
(DCOP.OBJ)+
(DCTRAN1.OBJ+DCTRAN2.OBJ+DCTRAN3.OBJ+DCTRAN4.OBJ+DCTRAN5.OBJ)+
(ERRCHK1.OBJ+ERRCHK2.OBJ)+
(OVTPVT.OBJ)+
(READIN1.OBJ+READIN2.OBJ)+
(SETUP.OBJ)+
UNIX.OBJ
SPICE2G6,SPICE2G6;
```

```
#Microsoft FORTRAN makefile for SPICE2G6
#K.H.Carpenter, EECE KSU, 02SEP88

#Use inline floating point calls, huge model:
FFLAGS= /FPi87 /AH /c

#Generic command for creating object files:
.FOR.OBJ:
    FL $(FFLAGS) $*.FOR

#Dependencies for the object files: (MSMAKE can't handle generic here!)
SPICE1.OBJ:     SPICE1.FOR  BLANKC.FOR

SPICE2.OBJ:     SPICE2.FOR  BLANKC.FOR

SPICE3.OBJ:     SPICE3.FOR  BLANKC.FOR

ACAN1.OBJ:      ACAN1.FOR   BLANKC.FOR

ACAN2.OBJ:      ACAN2.FOR   BLANKC.FOR

DCOP.OBJ:       DCOP.FOR    BLANKC.FOR

DCTRAN1.OBJ:    DCTRAN1.FOR BLANKC.FOR

DCTRAN2.OBJ:    DCTRAN2.FOR BLANKC.FOR

DCTRAN3.OBJ:    DCTRAN3.FOR BLANKC.FOR

DCTRAN4.OBJ:    DCTRAN4.FOR BLANKC.FOR

DCTRAN5.OBJ:    DCTRAN5.FOR BLANKC.FOR

ERRCHK1.OBJ:    ERRCHK1.FOR BLANKC.FOR

ERRCHK2.OBJ:    ERRCHK2.FOR BLANKC.FOR

OVTPVT.OBJ:     OVTPVT.FOR  BLANKC.FOR

READIN1.OBJ:    READIN1.FOR BLANKC.FOR

READIN2.OBJ:    READIN2.FOR BLANKC.FOR

SETUP.OBJ:      SETUP.FOR   BLANKC.FOR

UNIX.OBJ:       UNIX.FOR

#Dependencies for .EXE file, and link command:
SPICE2G6.EXE:   SPICE1.OBJ SPICE2.OBJ SPICE3.OBJ ACAN1.OBJ ACAN2.OBJ \
  DCOP.OBJ DCTRAN1.OBJ DCTRAN2.OBJ DCTRAN3.OBJ DCTRAN4.OBJ DCTRAN5.OBJ \
  ERRCHK1.OBJ ERRCHK2.OBJ OVTPVT.OBJ READIN1.OBJ READIN2.OBJ SETUP.OBJ UNIX.OBJ
#   LINK $**, $@, $@;
#Use the following line, along with the response file, to link.
    LINK /E @SPICE2G6.LNK
```

Appendix D

FORTRAN sources for fuse and saturating inductance modifications to SPICE

The following pages give listings of the additions and changes that must be made to the FORTRAN sources for SPICE2G6 to add fuse and saturating inductance circuit elements. The changes are differences from the version of SPICE2G6 ported to be compiled using MSFORTRAN on a PC. The complete listings of the subroutines implementing the new circuit elements are included in the difference files. The file README is listed first. It explains how to use the others to produce the complete source for the program. The "make" and linker-response files are also listed. The sample input files mentioned in README are given in the User's Guide in Appendix F.

The files are listed in the following order:

```
README
dctrn2s2.diff
readin1s.diff
spice1s2.diff
sp2g6sb2.lnk
sp2g6sb2.mak
```

README - K.H.Carpenter - EECE Dept., Kansas State University - 06JUN90

Instructions to make version of SPICE2G6 containing FUSE and SATURATING INDUCTANCE devices.

1. Make the MSFORTRAN version of SPICE2G6.

2. Use the files with extensions ".dif" given here to convert the corresponding files in the standard SPICE2G6 into the files needed for SP2G6SB2. (Use the "patch" program, if available.)

3. Use the file SP2G6SB2.MAK with Microsoft "MAKE" to compile the program under MSFORTRAN. Then compress it with EXEPACK to get the working .EXE file.

4. Example circuits LTANT1.CIR and TFUSE10.CIR illustrate the new devices.

```
*** dctran2.for Thu Sep  8 17:17:56 1988
--- dctrn2s2.for    Thu May 18 18:24:26 1989
***************
*** 1,5 ****
--- 1,17 ----
+ C DCTRN2S2  KHC 18MAY89
+ C This file modified by K.H.Carpenter, EECE Dept., KSU.
+ C Modifications are to:  (1) include a fuse model as a special
+ C  case of a Voltage controlled current source,
+ C  and (2) to include a saturation model for an inductor based on
+ C  the hyperbolic tangent approach.
+ C Changes are in routines LOAD, NLCSRC, and in the addition of new
+ C subroutires LTANH1, FUSE1.
+ C Changed lines contain tabs, and sections having changes are preceded
+ C by a comment line, CKHC.
+ C-----------------------------------------------------------------------
        SUBRCUTINE LOAD
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
+ CKHC - CHANGES 03AUG88 TO SUPPORT HYPERBOLIC SATURATION MODEL
  C      FOR INDUCTORS AND TO ALLOW BRIDGE ELEMENTS.
  C
  C     THIS ROUTINE ZEROES-OUT AND THEN LOADS THE COEFFICIENT MATRIX.
***************
*** 145,151 ****
--- 157,168 ----
        CIND=VALUE(LVNIM1+IPTR)
        IF ((INITF.EQ.5).AND.(NOSOLV.NE.0)) CIND=VALUE(LOCV+2)
        VALUE(LOCV+3)=CIND
+ CKHC
+   IF((NCOEF.EQ.5).AND.(VALUE(LCOEF+1).EQ.-1234))THEN
+       CALL LTANH1(FIND(LX0+LOCT),-1,LCOEF,NCOEF,LOCV+2,1,LOC+11)
+   ELSE
        CALL EVPOLY(FIND(LX0+LOCT),-1,LCOEF,NCOEF,LOCV+2,1,LOC+11)
+   ENDIF
    118 LOC=NODPLC(LOC)
        GO TO 110
    120 LOC=LOCATE(4)
***************
*** 183,189 ****
--- 200,211 ----
    205 IF (INITF.NE.5) GO TO 210
        FIND(LX1+LOCT)=FIND(LX0+LOCT)
    210 IF (IPOLY.EQ.1) GO TO 220
+ CKHC
+   IF((NCOEF.EQ.5).AND.(VALUE(LCOEF+1).EQ.-1234))THEN
+   CALL LTANH1(VALUE(LOCV+1),0,LCOEF,NCOEF,LOCV+2,1,LOC+11)
+   ELSE
        CALL EVPOLY(VALUE(LOCV+1),0,LCOEF,NCOEF,LOCV+2,1,LOC+11)
+   ENDIF
    220 IF (MODE.EQ.1) GO TO 250
        CALL INTGR8(REQ,VEQ,VALUE(LOCV+1),LOCT)
        IF (IPOLY.EQ.1) GO TO 250
***************
*** 401,406 ****
--- 423,430 ----
  C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
  CKHC      COMMON /BLANK/ VALUE(200000)
  $INCLUDE:'BLANKC.FOR'
+ CKHC
+   LOGICAL TFUSE
        INTEGER NODPLC(64)
        COMPLEX CVALUE(32)
        EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
***************
*** 428,434 ****
--- 452,464 ----
        LOCT=LOCT+2
        LNOD=LNOD+2
     20 CONTINUE
+ CKHC
+   TFUSE = (NDIM.EQ.1).AND.(VALUE(LCOEF+1).EQ.-1234.)
+   IF(TFUSE) THEN
+            CALL FUSE1(COLD,0,LCOEF,NCOEF,LARG,NDIM,NODE1,NODE2)
+   ELSE
        CALL EVPOLY(COLD,0,LCOEF,NCOEF,LARG,NDIM,LEXP)
+   ENDIF
        LOCT=NODPLC(LOC+12)
        IF (ICHECK.EQ.1) GO TO 30
        IF (INITF.EQ.6) GO TO 30
***************
*** 438,444 ****
--- 468,479 ----
     40 VALUE(LX0+LOCT)=COLD
        CEQ=COLD
        DO 50 I=1,NDIM
+ CKHC
+   IF(TFUSE) THEN
+       CALL FUSE1(GEQ,I,LCOEF,NCOEF,LARG,NDIM,NODE1,NODE2)
+   ELSE
        CALL EVPOLY(GEQ,I,LCOEF,NCOEF,LARG,NDIM,LEXP)
+   ENDIF
        LOCT=LOCT+2
        VALUE(LX0+LOCT)=GEQ
        CEQ=CEQ-GEQ*VALUE(LARG+I)
***************
*** 738,743 ****
--- 773,1148 ----
    320 CONTINUE
        RESULT=RESULT+VALUE(LCOEF+I)*ARG
    330 CONTINUE
+ C
+ C  FINISHED
+ C
+  1000 RETURN
+       END
+       SUBROUTINE FUSE1(RESULT,ITYPE,LCOEF,NCOEF,LARG,
+      1  NARG,NODE1,NODE2)
+       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
+ CKHC
+ C This routine added 13JAN89 to calculate resistance and current
+ C for a modification of the FIRESET model to allow dynamic
+ C calculation of the scaling factor based on current rate of rise.
+ C This routine calculates and saves its own action.
+ C Storage for saved variables is in the parameters passed to the
```

```
+ C program.
+ C
+ C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
+ CKHC      COMMON /BLANK/ VALUE(200000)
+ $INCLUDE:'BLANKC.FOR'
+ C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
+       COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
+      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
+      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
+ CKHC LOCAL VARIABLES:
+   LOGICAL FIXSC
+   DIMENSION STOV(16)
+   EQUIVALENCE (STOV(1),CMAX),(STOV(2),TMAX),(STOV(3),G)
+      1,(STOV(4),FLAG),(STOV(5),TLAST),(STOV(6),LASTI),(STOV(7),TOLD)
+      2,(STOV(8),OLDI),(STOV(9),CMOLD),(STOV(10),TMOLD)
+      3,(STOV(11),GOLD),(STOV(12),SCF),(STOV(13),RP),(STOV(14),RA)
+      4,(STOV(15),G1),(STOV(16),S1)
+       INTEGER NODPLC(64)
+       COMPLEX CVALUE(32)
+       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
+   SAVE DEV,EDEV,RP,RA,EXPGP,EXPGM,GO,SECHG,R
+ C
+ C
+       IF (ITYPE) 100,200,300
+ C
+ C  INTEGRATION NOT ALLOWED FOR FUSE1
+ C
+   100 WRITE(IOFILE,110)
+  110  FORMAT(/' ILLEGAL ITYPE IN FUSE1'/' JOB ABORTED')
+   STOP
+ C
+ C  EVALUATION OF FUSE CURRENT BASED ON A MODIFICATION OF
+ C  THE FIRESET MODEL OF RON LEE.
+ C
+ C  The use of this routine depends on its being called with
+ C  ITYPE = 0 to set the values of the model parameters before
+ C  being called with ITYPE=1 or 2 (for the same parameters).
+ C
+ C  First, a check is made to assure the correct number of parameters
+ C  are available, and the number is stored to indicate fixed or
+ C  dynamic scaling.
+ C
+   200     IF(NCOEF.EQ.25)THEN
+       FIXSC = .TRUE.
+       NPAR = 9
+   ELSEIF(NCOEF.EQ.26)THEN
+       FIXSC = .FALSE.
+       NPAR = 10
+   ELSE
+       WRITE(IOFILE,210)NCOEF-16
+  210  FORMAT(/I4,': ILLEGAL NUMBER OF PARAMETERS TO FUSE1'/
+      1  ' JOB ABORTED')
+       STOP
+   ENDIF
+ C
+ C  The input parameters are all assumed to be in SI units, since
+ C  SPICE2G6 assumes them.  Multiples, and even mils can be input with
+ C  SPICE unit suffixes.
+ C
+ C  The input parameters have the following meanings in the model:
+ C
+ C LCOEF+1  -  -1234 is the flag to indicate model FUSE1
+ C LCOEF+2  -  length (meters)
+ C LCOEF+3  -  width  (meters)
+ C LCOEF+4  -  thickness (meters)
+ C LCOEF+5  -  A = after melt resistivity (ohm-m)
+ C LCOEF+6  -  B = peak resistivity (ohm-m)
+ C LCOEF+7  -  G00 = normalized action at melt (A**2-s/m**4)
+ C LCOEF+8  -  S00 = normalized Gaussian width      "
+ C LCOEF+9  -  KOE = modified constant value (2E+11/exp(1))(A/s)
+ C LCOEF+10 -  POW = power for scale factor (0.19)
+ C  Aternately, if LCOEF+10 is not input, i.e., there are only 9 input
+ C  parameters, then LCOEF+9 is the fixed scale factor, SCF.
+ C
+ C  The following variables are stored with the input parameters, are not
+ C  included on the card specifiying the fuse:
+ C
+ C  NST is offset of storage locations into parameter memory:
+   NST = LCOEF + NPAR
+ C
+ C NST+1 -  CMAX = maximum absolute current in fuse up to TMAX
+ C NST+2 -  TMAX = time when max current occurred
+ C NST+3 -  G    = action of current in fuse
+ C NST+4 -  FLAG = set to one if fuse has blown, zero otherwise
+ C NST+5 -  TLAST= time of last call to this routine
+ C NST+6 -  LASTI= RESULT on last call to this routine
+ C NST+7 -  TOLD = time at last converged step
+ C NST+8 -  OLDI = RESULT on last call when converged
+ C NST+9 -  CMOLD= CMAX when last converged
+ C NST+10 - TMOLD= TMAX when last converged
+ C NST+11 - GOLD = value of G when last converged
+ C NST+12 - SCF  = scale factor for G0 and S0
+ C NST+13 - RP   = peak resistance
+ C NST+14 - RA   = after melt resistance
+ C NST+15 - G1   = action to melt
+ C NST+16 - S1   = Gaussian width of peak resistance
+ C
+ C  Since integration for action must assume a starting time,
+ C  it is assumed that each analysis starts at time zero.  Otherwise
+ C  the results will be meaningless.
+ C  Next test for time zero, and reset variables when it is:
+ C
+   IF(TIME.EQ.0.)THEN
+       DO 220 I=1,11
+           VALUE(NST+I) = 0.
+  220      CONTINUE
+ C SCF must have a non-zero initial value -
+       VALUE(NST+12) = 1.
+ C RP and RA can be calculated once and saved -
+       AREA          = VALUE(LCOEF+3) * VALUE(LCOEF+4)
+       VALUE(NST+13) = VALUE(LCOEF+6) * VALUE(LCOEF+2)/AREA
+       VALUE(NST+14) = VALUE(LCOEF+5) * VALUE(LCOEF+2)/AREA
+ C G1 and S1 can be calculated once and saved -
+       VALUE(NST+15) = VALUE(LCOEF+7)*AREA**2
```

```
+       VALUE(NST+16) = VALUE(LCOEF+8)*AREA**2
+ C  Write that reset has occured:
+       WRITE(IOFILE,222)NODE1-1,NODE2-1
+  222  FORMAT('OTIME=0 reset of parameters for Fuse from ',I3,' to',I3)
+    ENDIF
+ C
+ C  Copy saved variables to local variables:
+    DO 225 I=1,16
+       STOV(I) = VALUE(NST+I)
+  225  CONTINUE
+ C
+ C  Next check that the current time is an increase from the
+ C  previous call, and correct if it is not:
+ C The procedure here is complicated by the fact that
+ C the algorithms used in SPICE may cause
+ C this routine to be called several times for the same
+ C value of TIME while doing the nonlinear DC convergence,
+ C and may cause this routine to be called with a value
+ C of TIME smaller than a previous value due to dynamic
+ C step size determination.
+ C Examination of test output, printing out times called,
+ C leads to the conclusion that, at least for trapezoidal
+ C method, that once the TIME value increases over the
+ C value at the preceeding call, TIME will never be as small
+ C as on that preceeding call again.
+ C Thus the following procedure has
+ C been worked out to keep the CMAX, TMAX, and G values
+ C dependent only on fully converged and accepted values.
+ C
+    IF(TIME.LE.TLAST)THEN
+ C  Restore values from last converged step:
+       G = GOLD
+       CMAX = CMOLD
+       TMAX = TMOLD
+       TLAST = TOLD
+       LASTI = OLDI
+    ELSE
+ C  Save values from last step as converged, and
+ C  calculate new scale factor for G0 and S0:
+ C Test to see if SCF should change:
+       IF((.NOT.FIXSC).AND.
+      1              (CMAX.NE.CMOLD.OR.TMAX.NE.TMOLD))THEN
+ C Set scale factor based on stored current and time values:
+    SCF = (CMAX/(VALUE(LCOEF+9) * TMAX))**VALUE(LCOEF+10)
+ C WRITE(IOFILE,777)G,SCF,CMAX,TMAX,R,VALUE(LARG+1)
+ C777  FORMAT(' G=',1PE13.5,' SCF=',E13.5,' CMAX=',E13.5,' TMAX=',E13.5
+ C     1  ,' R=',E13.5,' V=',E13.5)
+       ENDIF
+ C Save other values as converged -
+       GOLD = G
+       CMOLD = CMAX
+       TMOLD = TMAX
+       TOLD = TLAST
+       OLDI = LASTI
+    ENDIF
+ C
+ C Now scale input values:
+    IF(FIXSC)SCF = VALUE(LCOEF+9)
+    G0 = SCF*G1
+    S0 = SCF*S1
+ C
+ C  Next find values needed in fuse model formulas:
+    GOG = G/G0
+ C  If GOG too large (far after burst, of course) then overflow
+ C  occurs on exponential evaluation.  Hence the following for
+ C  VAX use, which should be OK on other computers:
+    IF(GOG.GT.32)THEN
+       SECHG = 0.
+    ELSE
+       EXPGP = EXP(GOG)
+       EXPGM = EXP(-GOG)
+       SECHG = 2./(EXPGP+EXPGM)
+    ENDIF
+    DEV = (G-G0)/S0
+    R = RA*(1.-SECHG)
+    EDEV = EXP(-DEV*DEV)
+    R = R + RP*EDEV
+    RESULT = VALUE(LARG+1)/R
+ C
+ C  Next update action, maximum current and its time:
+ C Trapezoidal rule is used for integration for G.
+ C
+    G = G + 0.5*(TIME-TLAST)*(RESULT**2+LASTI**2)
+    IF(CMAX.LT.ABS(RESULT))THEN
+       CMAX = ABS(RESULT)
+       TMAX = TIME
+    ENDIF
+ C
+ C  Save current and time for next call:
+    LASTI = RESULT
+    TLAST = TIME
+ C
+ C  Next, test if G has exceeded the value for the fuse
+ C  to be "blown" according to the model, and output
+ C  a message if it has.  (Use old values here to be sure
+ C  the test is not made with a value that will be discarded
+ C  later.):
+    IF(FLAG.EQ.0..AND.GOLD.GT.G0)THEN
+       FLAG = 1.
+       WRITE(IOFILE,230)NODE1-1,NODE2-1,TOLD,OLDI,GOLD
+  230  FORMAT(/' FUSE from',I3,' to',I3,' blows at T='
+      1  ,1PE13.4,', I=',E13.4,', G=',E13.4/)
+ C  For debugging and checking on goodness of cmax/tmax relative to
+ C  Vo/L, output the following also:
+       WRITE(IOFILE,240)g0,cmold,tmold
+  240  FORMAT(' g0=',1pe13.4,' cmax=',e13.4,' tmax=',e13.4/)
+    ENDIF
+ C
+ C  Now load variables into global storage:
+    DO 250 I=1,12
+       VALUE(NST+I) = STOV(I)
+  250  CONTINUE
+ C
+ ckhc  test for sequences of TIME on call to FUSE1:
```

```
+ c if(time.le.1.e-8.and.time.ge.0.e-8)write(iofile,999)time
+ c999  format(' time = ',1pe13.5)
+       GO TO 1000
+ C
+ C  PARTIAL DERIVATIVE WITH RESPECT TO THE ITYPE*TH VARIABLE
+ C
+   300 IF(ITYPE.EQ.1) THEN
+           RESULT = 1./R
+       ELSE
+           RESULT = 2.*DEV*EDEV*RP -
+      1                      0.5*SECHG*SEGHG*RA*(EXPGP-EXPGM)/GO
+       RESULT = RESULT*VALUE(LARG+1)/(R*R)
+   ENDIF
+ C
+ C  FINISHED
+ C
+  1000 RETURN
+       END
+       SUBROUTINE LTANH1(RESULT,ITYPE,LCOEF,NCOEF,LARG,
+      1  NARG,LEXP)
+       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
+ CKHC
+ C This routine added 10AUG88 to calculate inductance and integral
+ C of inductance using the hyperbolic tangent saturation model.
+ C Modified 18MAY89 to prevent overflow in hyperbolic cosine calculation
+ C for large currents and large powers. (And corrected error for case
+ C with non-integer power.)
+ C
+ C SPICE VERSION 2G.6  SCCSID=BLANK 3/15/83
+ CKHC       COMMON /BLANK/ VALUE(200000)
+ $INCLUDE:'BLANKC.FOR'
+ C SPICE VERSION 2G.6  SCCSID=STATUS 3/15/83
+       COMMON /STATUS/ OMEGA,TIME,DELTA,DELOLD(7),AG(7),VT,XNI,EGFET,
+      1   XMU,SFACTR,MODE,MODEDC,ICALC,INITF,METHOD,IORD,MAXORD,NONCON,
+      2   ITERNO,ITEMNO,NOSOLV,MODAC,IPIV,IVMFLG,IPOSTP,ISCRCH,IOFILE
+       INTEGER NODPLC(64)
+       COMPLEX CVALUE(32)
+       EQUIVALENCE (VALUE(1),NODPLC(1),CVALUE(1))
+ C
+ C MAXIMUM ARGUMENT FOR COSH:
+   DATA CARGM/60/
+ C  If argument to COSH is greater than a value slightly over 700
+ C  then overflow occurs in MSFORTRAN double precision.  By keeping
+ C  the argument less than 60, the value of cosh(60) = 5.7D+25 will
+ C  not overflow even in single precision (or on a VAX).  The setting
+ C  of 1/cosh to zero in that case will give the exact same return from
+ C  this routine as would be obtained otherwise to double precision values
+ C  for relative permeablilities under 10**20 (which includes any physically
+ C  believable!).
+ C
+ C CALCULATE COMBINATIONS USED BY ALL CASES:
+   AI = VALUE(LARG+1)
+   AIA = ABS(AI)
+   AISAT = VALUE(LCOEF+4)
+   AIOIS = AIA/AISAT
+   ALFER = VALUE(LCOEF+2)
+   ALAIR = VALUE(LCOEF+3)
+   ALFMLA = ALFER - ALAIR
+   AN = VALUE(LCOEF+5)
+   N = AN
+ C  CALCULATE AIOSN HERE SO IT CAN BE USED TO TEST FOR ZERO
+ C  VALUE (ZERO TO A POWER CAUSES A MATH ERROR BELOW).
+   IF(AN.NE.1.)THEN
+       IF(AN.EQ.N)THEN
+           AIOSN = AIOIS**N
+       ELSE
+           AIOSN = AIOIS**AN
+       ENDIF
+   ELSE
+       AIOSN = AIOIS
+   ENDIF
+ C  SET AIOSN TO ZERO IF TOO SMALL TO AVIOD "INVALID" MATH ERROR
+ C  BUT, IF THIS TOLERANCE IS TOO LARGE, THEN CONVERGENCE WILL FAIL
+ C  FOR LARGE POWERS DUE TO A STEP OCCURING IN RESULT WHEN CHANGING
+ C  FROM AIOSN JUST BELOW THIS TOLERANCE TO JUST ABOVE IT.
+ C  CHOOSE POWER BELOW FOR MSFORTRAN.  IT WOULD BE TOO SMALL A TOLERANCE
+ C  TO USE ON A VAX WITH D-FLOATING.  THIS ALLOWS
+ C  POWERS UP TO 15 WITH ONLY A 10^-10 JUMP.
+   IF(AIOSN.LT.1.D-150)AIOSN = 0.
+       IF (ITYPE) 100,200,300
+ C
+ C  EVALUATION OF INTEGRAL OF INDUCTANCE FOR HYPERBOLIC TANGENT SATURATION
+ C  MODEL (Actually, the flux for the given current is returned.)
+ C
+  100  IF(AIOSN) 120, 110, 120
+  110  RESULT = 0.0
+   GOTO 1000
+  120  IF(AN.EQ.1.)THEN
+       RESULT = TANH(AIOIS)
+   ELSE
+       RESULT = (TANH(AIOSN))**(1./AN)
+   ENDIF
+   RESULT = RESULT*ALFMLA*AISAT + ALAIR*AIA
+   RESULT = SIGN(RESULT,AI)
+   GOTO 1000
+ C
+ C
+ C
+ C  EVALUATION OF INDUCTANCE BASED ON HYPERBOLIC TANGENT SATURATION MODEL
+ C   (Actually, the incremental inductance, dFLUX/dCURRENT, is returned.)
+ C
+ C
+  200  IF(AIOSN) 220, 210, 220
+  210  RESULT = ALFER
+   GOTO 1000
+  220  IF(AN.EQ.1.)THEN
+       IF(AIOIS.GT.CARGM)THEN
+           RESULT = 0.0
+       ELSE
+           RESULT = (1./COSH(AIOIS))**2
+       ENDIF
+       GOTO 250
+   ENDIF
+   IF(AIOSN.GT.CARGM)THEN
```

```
+       RESULT = 0.0
+    ELSE
+    RESULT =  (TANH(AIOSN))**(1./AN -1.) / (COSH(AIOSN))**2
+       1                     * AIOSN/AIOIS
+    ENDIF
+   250  RESULT = RESULT*ALFMLA + ALAIR
+    GOTO 1000
+ C
+ C  PARTIAL DERIVATIVE WITH RESPECT TO THE ITYPE*TH VARIABLE
+ C
+    300 WRITE(IOFILE,10)
+   10   FORMAT(/' ITYPE>0 ILLEGAL FOR HYPERBOLIC SAT. MODEL.',
+       1 /' JOB ABORTED')
+    STOP
  C
  C  FINISHED
  C
```

```
*** readin1.for Wed Aug 31 16:58:32 1988
--- readin1s.for    Fri Jan 13 16:19:30 1989
***************
*** 1,3 ****
--- 1,5 ----
+ C READIN1S.FOR - KHC - 13JAN89
+ C  Modified READIN1.FOR by adding extra storage for FUSE on input.
  C SPICE VERSION 2G.6  SCCSID=readin.ma 3/22/83
        SUBROUTINE READIN
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
***************
*** 397,403 ****
    166 ISPOT=NODPLC(LOC+LTAB+5)+I
        VALUE(ISPOT)=VALUE(IFIELD+IFLD)
    168 CONTINUE
!   170 IF (NDIM.NE.1) GO TO 50
        IF (IKNT.NE.1) GO TO 50
        CALL EXTMEM(NODPLC(LOC+LTAB+2),1)
        ISPOT=NODPLC(LOC+LTAB+2)
--- 399,412 ----
    166 ISPOT=NODPLC(LOC+LTAB+5)+I
        VALUE(ISPOT)=VALUE(IFIELD+IFLD)
    168 CONTINUE
! CKHC
! C  Here, check if input is a FUSE model.  If so, reserve an
! C  extra 16 words in the parameter space:
!  170  IF((IKNT.EQ.9.OR.IKNT.EQ.10)
!      1    .AND.VALUE(NODPLC(LOC+LTAB+2)+1).EQ.-1234.)THEN
!       CALL EXTMEM(NODPLC(LOC+LTAB+2),16)
!   ENDIF
!       IF (NDIM.NE.1) GO TO 50
        IF (IKNT.NE.1) GO TO 50
        CALL EXTMEM(NODPLC(LOC+LTAB+2),1)
        ISPOT=NODPLC(LOC+LTAB+2)
```

```
*** spice1.for  Wed Jun  6 16:58:39 1990
--- spice1s2.for    Wed Jun  6 17:08:00 1990
***************
*** 1,5 ****
! C SPICE VERSION SG6-KHC-MS0  9/10/88
! C  SPICE2G6 for MSFORTRAN under MSDOS on 640K PC's.
  C Modified from the Unix version to remove the plot output to
  C   unit 10 (found at lines beginning with C UW/NW).
  C   MSFORTRAN version has blank common specification placed
--- 1,6 ----
! C SPICE VERSION SG6-KHC-MS2 5/18/89
! C  SPICE2G6 for MSFORTRAN under MSDOS on 640K PC's -
! C   with Fuse and Saturating Inductance devices added.
  C Modified from the Unix version to remove the plot output to
  C   unit 10 (found at lines beginning with C UW/NW).
  C   MSFORTRAN version has blank common specification placed
***************
*** 156,162 ****
       DATA ABLNK /1H  /
       DATA ACCTIT / 8HJOB STAT, 8HISTICS S, 8HUMMARY  , 8H         /
  CKHC      DATA AHDR1,AHDR2,AHDR3 / 8H  SPICE ,8H2G.6    ,8H3/15/83     /
!       DATA AHDR1,AHDR2,AHDR3 / 8HSPICE2G6,8H-KHC-MS0,8H 9/10/88    /
  C
  C
  CKHC      IPOSTP=0
--- 157,163 ----
       DATA ABLNK /1H  /
       DATA ACCTIT / 8HJOB STAT, 8HISTICS S, 8HUMMARY  , 8H         /
  CKHC      DATA AHDR1,AHDR2,AHDR3 / 8H  SPICE ,8H2G.6    ,8H3/15/83     /
!       DATA AHDR1,AHDR2,AHDR3 / 8HSPICE2G6,8H-KHC-MS2,8H 18MAY89   /
  C
  C
  CKHC      IPOSTP=0
```

```
SPICE1S2.OBJ+SPICE2.OBJ+SPICE3.OBJ+
(ACAN1.OBJ+ACAN2.OBJ)+
(DCOP.OBJ)+
(DCTRAN1.OBJ+DCTRN2S2.OBJ+DCTRAN3.OBJ+DCTRAN4.OBJ+DCTRAN5.OBJ)+
(ERRCHK1.OBJ+ERRCHK2.OBJ)+
(OVTPVT.OBJ)+
(READIN1S.OBJ+READIN2.OBJ)+
(SETUP.OBJ)+
UNIX.OBJ
C:SP2G6SB2,C:SP2G6SB2;
```

```
#Microsoft FORTRAN makefile for SPICE2G6-SB2
#K.H.Carpenter, EECE KSU, 28DEC88

#Use inline floating point calls, huge model:
FFLAGS= /FPi87 /AH /c

#Generic command for creating object files:
.FOR.OBJ:
    FL $(FFLAGS) $*.FOR

#Dependencies for the object files: (MSMAKE can't handle generic here!)
SPICE1S2.OBJ:   SPICE1S2.FOR    BLANKC.FOR

SPICE2.OBJ:     SPICE2.FOR  BLANKC.FOR

SPICE3.OBJ:     SPICE3.FOR  BLANKC.FOR

ACAN1.OBJ:      ACAN1.FOR   BLANKC.FOR

ACAN2.OBJ:      ACAN2.FOR   BLANKC.FOR

DCOP.OBJ:       DCOP.FOR    BLANKC.FOR

DCTRAN1.OBJ:    DCTRAN1.FOR BLANKC.FOR

DCTRN2S2.OBJ:   DCTRN2S2.FOR BLANKC.FOR

DCTRAN3.OBJ:    DCTRAN3.FOR BLANKC.FOR

DCTRAN4.OBJ:    DCTRAN4.FOR BLANKC.FOR

DCTRAN5.OBJ:    DCTRAN5.FOR BLANKC.FOR

ERRCHK1.OBJ:    ERRCHK1.FOR BLANKC.FOR

ERRCHK2.OBJ:    ERRCHK2.FOR BLANKC.FOR

OVTPVT.OBJ:     OVTPVT.FOR  BLANKC.FOR

READIN1S.OBJ:   READIN1S.FOR    BLANKC.FOR

READIN2.OBJ:    READIN2.FOR BLANKC.FOR

SETUP.OBJ:      SETUP.FOR   BLANKC.FOR

UNIX.OBJ:       UNIX.FOR

#Dependencies for .EXE file, and link command:
SP2G6SB2.EXE:   SPICE1S2.OBJ SPICE2.OBJ SPICE3.OBJ ACAN1.OBJ ACAN2.OBJ \
  DCOP.OBJ DCTRAN1.OBJ DCTRN2S2.OBJ DCTRAN3.OBJ DCTRAN4.OBJ DCTRAN5.OBJ \
  ERRCHK1.OBJ ERRCHK2.OBJ OVTPVT.OBJ READIN1S.OBJ READIN2.OBJ SETUP.OBJ UNIX.OBJ
#   LINK $**, $@, $@;
#Use the following line, along with the response file, to link.
    LINK /E @SP2G6SB2.LNK
```

Appendix E

Comparisons of execution speed for SPICE test problems on different computers

The SPICE2G6 code comes with a set of test inputs included in a file named "benchmark.in". This file was used as input to test the port of SPICE2G6 to MSDOS PC's, by comparing its output when run on a PC to its output when run on Unix computers and on VAX computers using the VMS operating system. There was also an output file included with the source distribution, but the computer on which it was made was not known. All the outputs agreed within the limits of the numerical precision used.

As a result of making these runs of the test programs, speed of execution data was available for several computer systems. This speed data is of interest of itself, and so has been summarized in a table reproduced on the following page.

BENCHMARK - tests of SPICE2G6

COMPUTER: / TEST	Z158 8MHz	HP Vectra	GV386 /287	GV386 /387	Z386 cache	VAX750 EUNICE	VAX750 VMS	PSPICE2 on GV386	Berkeley Machine
Differential Pair	221.0	106.0	54.0	44.0	38.0	37.8	23.5	21.8	22.2
Tunnel Diode Oscillator	128.0	63.0	35.0	29.0	24.0	36.6	49.5	10.9	18.4
RTL inverter	147.0	68.0	37.0	29.0	25.0	29.5	19.1	15.3	15.9
Schmitt trigger	158.0	77.0	42.0	30.0	27.0	28.5	17.8	15.2	17.3
MOS memory	208.0	118.0	73.0	40.0	38.0	46.2	26.3	20.9	34.7
MOS amplifier - DC/AC	466.0	272.0	170.0	80.0	81.0	81.5	48.9	circuit	74.4
MOS amplifier - transient	748.0	483.0	322.0	130.0	138.0	163.7	94.9	too large	169.1
Total Time (Seconds):	2076.0	1187.0	733.0	382.0	371.0	423.8	279.9	****	351.9
Ratios to VAX750/VMS:	7.4	4.2	2.6	1.4	1.3	1.5	1.0		1.3
Ratios for 1st 5 times:	6.3	3.2	1.8	1.3	1.1	1.3	1.0	0.6	0.8

Breakdown of time for MOS amplifier - transient:

Readin	17.0	6.0	3.0	2.0	2.0	2.7	1.5		1.8
Setup	10.0	3.0	1.0	1.0	1.0	1.2	0.8		0.8
DCAN	29.0	18.0	11.0	5.0	5.0	5.5	3.5		5.0
DCDCMP	66.0	22.0	5.0	9.0	5.0	7.4	5.4		4.7
DCSOL	43.0	12.0	14.0	5.0	11.0	4.2	2.9		3.1
TRANAN	667.0	443.0	300.0	111.0	123.0	150.4	87.1		159.1
Output	9.0	3.0	2.0	2.0	1.0	3.6	1.9		2.1
Load	515.0	387.0	274.0	91.0	99.0	130.5	72.4		141.5
Overhead	16.0	10.0	5.0	9.0	6.0	0.3	0.2		0.2

Appendix F

SPICE User's Guide – for version 2G6-SB2

The following pages contain a reproduction of the User's Guide to SPICE, as distributed by U.C.Berkeley, with modifications added to clarify certain points, and to add documentation for the fuse and saturating inductance circuit elements.

SPICE User's Guide - for version 2G6-SB2 [1]

Adapted by

Kenneth H. Carpenter
Department of Electrical and Computer Engineering
Kansas State University
Manhattan, Kansas 66506

from the User's Guide for version 3B1 by

T. Quarles, A.R.Newtor, D.O.Pederson, A.Sangiovanni-Vincentelli

and from the User's Guide for version 2G by

A.Vladimirescu, Kaihe Zhang, A.R.Newton,
D.O.Pederson, A.Sangiovanni-Vincentelli

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

June 15, 1989

[1]Developed for Lawrence Livermore National Laboratory under subcontract from the University of California to Kansas State University.

Contents

INTRODUCTION

SPICE is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analyses. In SPICE version 2G6, circuits may contain resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines, switches, and four of the most common semiconductor devices: diodes, BJTs, JFETs, and MOSFETs. In addition, version 2G6-SB2 adds a model for saturation of magnetization in inductors and adds a fuse circuit element.

SPICE has built-in models for the semiconductor devices, and the user need specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon; however, if the Gummel-Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The diode model can be used for either junction diodes or Schottky barrier diodes. The JFET model is based on the FET model of Shichman and Hodges. Three MOSFET models are implemented in SPICE 2G6: MOS1 is described by a square-law I-V characteristic, MOS2[1] is an analytical model, while MOS3[1] is a semi-empirical model. Both MOS2 and MOS3 include second-order effects such as channel length modulation, subthreshold conduction, scattering limited velocity saturation, small-size effects, and charge-controlled capacitances.

Circuits and analyses on them to be performed by SPICE are specified by statements in an input file submitted to the program. Results from the analyses are written on output files for printing or post processing. This document serves as a reference manual for preparation of the input file.

1 TYPES OF ANALYSES

1.1 DC Analysis

The dc analysis portion of SPICE determines the dc operating point of the circuit with inductors shorted and capacitors opened. A dc analysis is automatically performed prior to a transient analysis to determine the transient initial conditions [unless suppressed by the UIC parameter on the .TRAN control statement], and prior to an ac small-signal analysis to determine the linearized, small-signal models for nonlinear devices. If requested, the dc small-signal value of a transfer function (ratio of output variable to input source), input resistance, and output resistance will also be computed as a part of the dc solution. The dc analysis can also be used to generate dc transfer curves: a specified independent voltage or current source is stepped over a user-specified range and the dc output variables are stored for each sequential source value. If requested, SPICE also will determine the dc small-signal sensitivities of specified output variables with respect to circuit parameters. The dc analysis options are specified on the .DC, .TF, .OP, and .SENS control statements.

If one desires to see the small-signal models for nonlinear devices in conjunction with a transient analysis operating point, then the .OP statement must be provided. The dc bias conditions will be identical for each case, but the more comprehensive operating point information is not available to be printed when transient initial conditions are computed.

1.2 AC Small-Signal Analysis

The ac small-signal portion of SPICE computes the ac output variables as a function of frequency. The program first computes the dc operating point of the circuit and determines linearized, small-signal models for all of the nonlinear devices in the circuit. The resultant linear circuit is then analyzed over a user-specified range of frequencies. The desired output of an ac small- signal analysis is usually a transfer function (voltage gain, transimpedance, etc). If the circuit has only one ac input, it is convenient to set that input to unity and zero phase, so that output variables have the same value as the transfer function of the output variable with respect to the input.

The generation of white noise by resistors and semiconductor devices can also be simulated with the ac small-signal portion of SPICE. Equivalent noise source values are determined automatically from the small signal operating point of the circuit, and the contribution of each noise source is added at a given summing point. The total output noise level and the equivalent input noise level are determined at each frequency point. The output and input noise levels are normalized with respect to the square root of the noise bandwidth and have the units Volts/$\sqrt{\text{Hz}}$ or Amps/$\sqrt{\text{Hz}}$. The output noise and equivalent input noise can be printed or plotted in the same fashion as other output variables. No additional input data are necessary for this analysis.

Flicker noise sources can be simulated in the noise analysis by including values for the parameters KF and AF on the appropriate device model statements.

The distortion characteristics of a circuit in the small-signal mode can be simulated as a part of the ac small-signal analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input.

The frequency range and the noise and distortion analysis parameters are specified on the .AC, .NOISE, and .DISTO control statements.

1.3 Transient Analysis

The transient analysis portion of SPICE computes the transient output variables as a function of time over a user-specified time interval. The initial conditions can be specified for circuit parameters or can be automatically determined by a dc analysis. All sources which are not time dependent (for example, power supplies) are set to their dc values. For large-signal sinusoidal simulations, a Fourier analysis of the output waveform can be specified to obtain the frequency domain Fourier coefficients. The transient time interval and the Fourier analysis options are specified on the .TRAN and .FOURIER control statements.

1.4 Analysis at Different Temperatures

All input data for SPICE is assumed to have been measured at 27 deg C (300 deg K). The simulation also assumes a nominal temperature of 27 deg C. The circuit can be simulated at other temperatures by using a .TEMP control statement.

Temperature appears explicitly in the exponential terms of the BJT and diode model equations. In addition, saturation currents have a built-in temperature dependence. The temperature dependence of the saturation current in the BJT models is determined by:

```
IS(T1) = IS(T0)*((T1/T0)**XTI)*exp(q*EG*(T1-T0),(k*T1*T0))
```


where k is Boltzmann's constant, q is the electronic charge, EG is the energy gap which is a model parameter, and XTI is the saturation current temperature exponent (also a model parameter, and usually equal to 3). The temperature dependence of forward and reverse beta is according to the formula:

```
beta(T1)=beta(T0)*(T1/T0)**XTB
```

where T1 and T0 are in degrees Kelvin, and XTB is a user-supplied model parameter. Temperature effects on beta are carried out by appropriate adjustment to the values of BF, ISE, BR, and ISC. Temperature dependence of the saturation current in the junction diode model is determined by:

```
IS(T1) = IS(T0)*((T1/T0)**(XTI/N))*exp(q*EG*(T1-T0)/(k*N*T1*T0))
```

where N is the emission coefficient, which is a model parameter, and the other symbols have the same meaning as above. Note that for Schottky barrier diodes, the value of the saturation current temperature exponent, XTI, is usually 2.

Temperature appears explicitly in the value of junction potential, PHI, for all the device models. The temperature dependence is determined by:

```
PHI(TEMP) = k*TEMP/q*log(Na*Nd/Ni(TEMP)**2)
```

where k is Boltzmann's constant, q is the electronic charge, Na is the acceptor impurity density, Nd is the donor impurity density, Ni is the intrinsic concentration, and EG is the energy gap.

Temperature appears explicitly in the value of surface mobility, UO, for the MOSFET model. The temperature dependence is determined by:

```
UO(TEMP) = UO(TNOM)/(TEMP/TNOM)**(1.5)
```

The effects of temperature on resistors is modeled by the formula:

```
value(TEMP) = value(TNOM)*(1+TC1*(TEMP-TNOM)+TC2*(TEMP-TNOM)**2))
```

where TEMP is the circuit temperature, TNOM is the nominal temperature, and TC1 and TC2 are the first- and second-order temperature coefficients.

2 CONVERGENCE

Both dc and transient solutions are obtained by an iterative process which is terminated when both of the following conditions hold:

1. The nonlinear branch currents converge to within a tolerance of 0.1 percent or 1 picoamp (1.0E-12 Amp), whichever is larger.

2. The node voltages converge to within a tolerance of 0.1 percent or 1 microvolt (1.0E-6 Volt), whichever is larger.

[The tolerances can be changed by using the .OPTIONS statement – see below.] Although the algorithm used in SPICE has been found to be very reliable, in some cases it will fail to converge to a solution. When this failure occurs, the program will print the node voltages at the last iteration and terminate the job. In such cases, the node voltages that are printed are not necessarily correct or even close to the correct solution.

Failure to converge in dc analysis is usually due to an error in specifying circuit connections, element values, or model parameter values. Regenerative switching circuits or circuits with positive feedback probably will not converge in the dc analysis unless the OFF option is used for some of the devices in the feedback path, or the .NODESET statement is used to force the circuit to converge to the desired state.

[Experience has shown that convergence problems may occur during transient analyses if reactive elements do not have losses associated with them. Placing a large resistor in parallel with reactive elements may help convergence problems without changing the theoretical values of any results by significant amounts.]

3 INPUT FORMAT

The input format for SPICE is of the free format type. Each SPICE statement begins on a new line in the input file. Fields in a statement are separated by one or more blanks, a comma, an equal (=) sign, or a left or right parenthesis; extra spaces are ignored. A statement may be continued by entering a + (plus) in column 1 of the following line; SPICE continues reading beginning with column 2. [The "newline" and following "+" are effectively replaced by a single space.] A single line may not exceed the length given by the .WIDTH statement (or its default value). Any characters beyond this length are discarded. [In some implementations the length may not exceed 72 characters – it is best to limit lines to 72 characters.]

A name field must begin with a letter (A through Z) and cannot contain any delimiters. Only the first eight characters of the name are used.

A number field may be an integer field (12, -44), a floating point field (3.14159), either an integer or floating point number followed by an integer exponent (1E-14, 2.65E3), or either an integer or a floating point number followed by one of the following scale factors:

```
T=1E12    G=1E9     MEG=1E6    K=1E3     MIL=25.4E-6
M=1E-3    U=1E-6    N=1E-9     P=1E-12   F=1E-15
```

Letters immediately following a number that are not scale factors are ignored, and letters immediately following a scale factor are ignored. Hence, 10, 10V, 10VOLTS, and 10HZ all represent the same number, and M, MA, MSEC, and MMHOS all represent the same scale factor. Note that 1000, 1000.0, 1000HZ, 1E3, 1.0E3, 1KHZ, and 1K all represent the same number.

[Note: there are pitfalls for the unwary when using scale factors! Especially, with regard to assuming that letters following the scale factor are ignored. As a ridiculous example, suppose there were a unit called "EGGS". Then if one were to wish to specify a number of "milli-eggs" as, e.g., 2.5MEGGS, then the result within SPICE would be 9 orders of magnitude larger than intended.]

4 CIRCUIT DESCRIPTION

The circuit to be analyzed is described to SPICE by a set of element statements, which define the circuit topology and element values, and a set of control statements, which define the model parameters and the run controls. The first line in the input file must be a title statement, and the last line must be a .END statement. The order of the remaining lines is arbitrary (except, of course, that continuation lines must immediately follow the line being continued).

Each element in the circuit is specified by an element statement that contains the element name, the circuit nodes to which the element is connected, and the values of the parameters that determine the electrical characteristics of the element. The first letter of the element name specifies the element type. The format for the SPICE element types is given in what follows. The strings XXXXXXX, YYYYYYY, and ZZZZZZZ denote arbitrary alphanumeric strings. For example, a resistor name must begin with the letter R and can contain one or more characters. Hence, R, R1, RSE, ROUT, and R3AC2ZY are valid resistor names.

Data fields that are enclosed in less than and greater than signs "< >" are optional. All indicated punctuation (parentheses, equal signs, etc.) is optional and merely indicates the presence of any delimiter. [Unfortunately, the previous statement is not universally true. The parentheses around the node numbers on the .PRINT and .PLOT statements are required, and the parentheses following POLY are required on the nonlinear dependent source statements described in Appendix B.] A consistent style such as that shown here will make the input easier to understand. With respect to branch voltages and currents, SPICE uniformly uses the associated reference convention (current flows in the direction of voltage drop). [Also known as passive sign convention in circuit theory.]

Nodes must be nonnegative integers but need not be numbered sequentially. The datum (ground) node must be numbered zero. The circuit cannot contain a loop of voltage sources and/or inductors and cannot contain a cutset of current sources and/or capacitors. Each node in the circuit must have a dc path to ground. [Note that current sources, both dependent and independent, are open circuits to dc, and thus do not provide dc paths to ground. So also is the fuse element defined in version 2G6-SB2.] Every node must have at least two connections except for transmission line nodes (to permit unterminated transmission lines) and MOSFET substrate nodes (which have two internal connections anyway).

5 TITLE, COMMENT, AND .END STATEMENTS

5.1 Title Statement

Examples:

```
POWER AMPLIFIER CIRCUIT
TEST OF CAM CELL
```

This statement must be the first line in the input file. Its contents are printed verbatim as the heading for each section of output.

5.2 .END Statement

Example:

```
.END
```

This statement must always be the last line in the input file. Note that the period is an integral part of the name.

5.3 Comment Line

General Form:

```
* <any comment>
```

Examples:

```
* RF=1K      GAIN SHOULD BE 100
* MAY THE FORCE BE WITH MY CIRCUIT
```

The asterisk in the first column indicates that this line is a comment line. Comment lines may be placed anywhere in the circuit description.

6 ELEMENT STATEMENTS

6.1 Resistors

General Form:

```
RXXXXXXX N1 N2 VALUE <TC=TC1<,TC2>>
```

Examples:

```
R1 1 2 100
RC1 12 17 1K
```

N1 and N2 are the two element nodes. VALUE is the resistance (in ohms) and may be positive or negative but not zero. TC1 and TC2 are the (optional) temperature coefficients; if not specified, zero is assumed for both. The value of the resistor as a function of temperature is given by:

```
value(TEMP) = value(TNOM)*(1+TC1*(TEMP-TNOM)+TC2*(TEMP-TNOM)**2))
```

6.2 Capacitors and Inductors

General Form:

```
CXXXXXXX N+ N- VALUE <IC=INCOND>
LYYYYYYY N+ N- VALUE <IC=INCOND>
```

Examples:

```
CBYP 13 0 1UF
COSC 17 23 10U IC=3V
LLINK 42 69 1UH
LSHUNT 23 51 10U IC=15.7MA
```


N+ and N– are the positive and negative element nodes, respectively. VALUE is the capacitance in Farads or the inductance in Henries.

For the capacitor, the (optional) initial condition is the initial (time-zero) value of capacitor voltage (in Volts). For the inductor, the (optional) initial condition is the initial (time-zero) value of inductor current (in Amps) that flows from N+, through the inductor, to N–. Note that the initial conditions (if any) apply 'only' if the UIC option is specified on the .TRAN statement.

Nonlinear capacitors and inductors can be described.

General Form:

```
CXXXXXXX N+ N- POLY C0 C1 C2 ... <IC=INCOND>
LYYYYYYY N+ N- POLY L0 L1 L2 ... <IC=INCOND>
```

C0 C1 C2 ...(and L0 L1 L2 ...) are the coefficients of a polynomial describing the element value. The capacitance is expressed as a function of the voltage across the element while the inductance is a function of the current through the inductor. The value is computed as

```
value=C0+C1*V+C2*V**2+...
value=L0+L1*I+L2*I**2+...
```

where V is the voltage across the capacitor and I the current flowing in the inductor.

Saturating inductors, based on a hyperbolic tangent model for the magnetization curve, can be defined as a special case of nonlinear inductors in SPICE version 2G6-SB2.

General Form:

```
LXXXXXXX N+ N- POLY -1234 LI LA ISAT POW
```

where -1234 is the flag to indicate the hyperbolic tangent saturation model is being used, and LI is the initial inductance (in Heneries), LA is the inductance after saturation (in Heneries), ISAT is the current at the knee of the saturation curve (in Amperes), and POW is the power-root used for the hyperbolic tangent and its argument in the model. POW must be positive. Values greater than one give sharper knees in the curve, values less than one give more gradual transitions from saturated to unsaturated magnetization. (Excessively large values for POW lead to a sharp corner in the magnetization, and can cause problems in the numerical analysis.) The inductance is calculated by assuming that the flux linkage of the winding carrying current of magnitude I is given by

```
FLUX = I*LA + (LI - LA)*ISAT*((TANH((I/ISAT)**POW))**(1/POW))
```

Note that a saturating inductor may not be used as one of the inductances coupled via the mutual coupling described next. (SPICE will accept such an input, but the results will not be meaningful.) To use the saturating inductor model to simulate a transformer having a ferromagnetic core, a "T" equivalent circuit for the transformer should be employed in which the magnetizing inductance is the saturating one.

6.3 Coupled (Mutual) Inductors

General Form:

```
KXXXXXXX LYYYYYYY LZZZZZZZ VALUE
```

Examples:

```
K43 LAA LBB 0.999
KXFRMR L1 L2 0.87
```

LYYYYYYY and LZZZZZZZ are the names of the two coupled inductors, and VALUE is the coefficient of coupling, K, which must be greater than 0 and less than or equal to 1. Using the 'dot' convention, place a 'dot' on the first node of each inductor.

6.4 Transmission Lines (Lossless)

General Form:

```
TXXXXXXX N1 N2 N3 N4 Z0=VALUE <TD=VALUE> <F=FREQ <NL=NRMLEN>>
+                    <IC=V1,I1,V2,I2>
```

Example:

```
T1 1 0 2 0 Z0=50 TD=10NS
```

N1 and N2 are the nodes at port 1; N3 and N4 are the nodes at port 2. Z0 is the characteristic impedance. [Note that the second character in Z0 is a zero.] The length of the line may be expressed in either of two forms. The transmission delay, TD, may be specified directly (as TD=10ns, for example). Alternatively, a frequency F may be given, together with NL, the normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency F. If a frequency is specified but NL is omitted, 0.25 is assumed (that is, the frequency is assumed to be the quarter-wave frequency). Note that although both forms for expressing the line length are indicated as optional, one of the two must be specified.

Note that this element models only one propagating mode. If all four nodes are distinct in the actual circuit, then two modes may be excited. To simulate such a situation, two transmission-line elements are required. (See the example in Appendix A for further clarification.)

The (optional) initial condition specification consists of the voltage and current at each of the transmission line ports. Note that the initial conditions (if any) apply 'only' if the UIC option is specified on the .TRAN statement.

One should be aware that SPICE will use a transient time step which does not exceed 1/2 the minimum transmission line delay. Therefore very short transmission lines (compared with the analysis time frame) will cause long run times.

6.5 Linear Dependent Sources

SPICE allows circuits to contain linear dependent sources characterized by any of the four equations


```
i=g*v        v=e*v        i=f*i        v=h*i
```

where g, e, f, and h are constants representing transconductance, voltage gain, current gain, and transresistance, respectively.

6.5.1 Linear Voltage-Controlled Current Sources

General Form:

```
GXXXXXXX N+ N- NC+ NC- VALUE
```

Example:

```
G1 2 0 5 0 0.1MMHO
```

N+ and N− are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. NC+ and NC− are the positive and negative controlling nodes, respectively. VALUE is the transconductance (in mhos).

6.5.2 Linear Voltage-Controlled Voltage Sources

General Form:

```
EXXXXXXX N+ N- NC+ NC- VALUE
```

Example:

```
E1 2 3 14 1 2.0
```

N+ is the positive node, and N− is the negative node. NC+ and NC− are the positive and negative controlling nodes, respectively. VALUE is the voltage gain.

6.5.3 Linear Current-Controlled Current Sources

General Form:

```
FXXXXXXX N+ N- VNAM VALUE
```

Example:

```
F1 13 5 VSENS 5
```

N+ and N− are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the current gain.

6.5.4 Linear Current-Controlled Voltage Sources

General Form:

```
HXXXXXXX N+ N- VNAM VALUE
```

Example:

```
HX 5 17 VZ 0.5K
```

N+ and N− are the positive and negative nodes, respectively. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the transresistance (in ohms).

6.6 Independent Sources

General Form:

```
VXXXXXXX N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
IYYYYYYY N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
```

Examples:

```
VCC 10 0 DC 6
VIN 13 2 0.001 AC 1 SIN(0 1 1MEG)
ISRC 23 21 AC 0.333 45.0 SFFM(0 1 10K 5 1K)
VMEAS 12 9
```

N+ and N− are the positive and negative nodes, respectively. Note that voltage sources need not be grounded. Positive current is assumed to flow from the positive node, through the source, to the negative node. A current source of positive value, will force current to flow out of the N+ node, through the source, and into the N− node. Voltage sources, in addition to being used for circuit excitation, are the 'ammeters' for SPICE, that is, zero valued voltage sources may be inserted into the circuit for the purpose of measuring current. They will, of course, have no effect on circuit operation since they represent short-circuits.

DC/TRAN is the dc and transient analysis value of the source. If the source value is zero both for dc and transient analyses, this value may be omitted. If the source value is time-invariant (e.g., a power supply), then the value may optionally be preceded by the letters DC.

ACMAG is the ac magnitude and ACPHASE is the ac phase. The source is set to this value in the ac analysis. If ACMAG is omitted following the keyword AC, a value of unity is assumed. If ACPHASE is omitted, a value of zero is assumed. If the source is not an ac small-signal input, the keyword AC and the ac values are omitted.

Any independent source can be assigned a time-dependent value for transient analysis. If a source is assigned a time-dependent value, the time-zero value is used for dc analysis. There are five independent source functions: pulse, exponential, sinusoidal, piece-wise linear, and single-frequency FM. If parameters other than source values are omitted or set to zero, the default values shown will be assumed. (TSTEP is the printing increment and TSTOP is the final time (see the .TRAN statement for explanation)).

6.6.1 Pulse source

General Form:

```
VXXXXXXX N+ N- PULSE(V1 V2 TD TR TF PW PER)
IXXXXXXX N+ N- PULSE(V1 V2 TD TR TF PW PER)
```

Example:

```
VIN 3 0 PULSE(-1 1 2NS 2NS 2NS 50NS 100NS)
```

parameters	default values	units
V1 (initial value)		Volts or Amps
V2 (pulsed value)		Volts or Amps
TD (delay time)	0.0	seconds
TR (rise time)	TSTEP	seconds
TF (fall time)	TSTEP	seconds
PW (pulse width)	TSTOP	seconds
PER(period)	TSTOP	seconds

A single pulse so specified is described by the following table:

time	value
0	V1
TD	V1
TD+TR	V2
TD+TR+PW	V2
TD+TR+PW+TF	V1
TSTOP	V1

Intermediate points are determined by linear interpolation.

6.6.2 Sinusoidal source

General Form:

```
VXXXXXXX N+ N- SIN(VO VA FREQ TD THETA)
IXXXXXXX N+ N- SIN(VO VA FREQ TD THETA)
```

Example:

```
VIN 3 0 SIN(0 1 100MEG 1NS 1E10)
```

parameters		default values	units
VO	(offset)		Volts or Amps
VA	(amplitude)		Volts or Amps
FREQ	(frequency)	1/TSTOP	Hz
TD	(delay)	0.0	seconds
THETA	(damping factor)	0.0	1/seconds

The shape of the waveform is described by the following table:

```
time            value
0 to TD         V0
TD to TSTOP     V0 + VA*exp(-(time-TD)*THETA)*sine(twopi*FREQ*(time+TD))
```

6.6.3 Exponential source

General Form:

```
VXXXXXXX N+ N- EXP(V1 V2 TD1 TAU1 TD2 TAU2)
IXXXXXXX N+ N- EXP(V1 V2 TD1 TAU1 TD2 TAU2)
```

Examples:

```
VIN 3 0 EXP(-4 -1 2NS 30NS 60NS 40NS)
```

parameters		default values	units
V1	(initial value)		Volts or Amps
V2	(pulsed value)		Volts or Amps
TD1	(rise delay time)	0.0	seconds
TAU1	(rise time constant)	TSTEP	seconds
TD2	(fall delay time)	TD1+TSTEP	seconds
TAU2	(fall time constant)	TSTEP	seconds

The shape of the waveform is described by the following table:

```
time            value

0 to TD1        V1
TD1 to TD2      V1+(V2-V1)*(1-exp(-(time-TD1)/TAU1))
TD2 to TSTOP    V1+(V2-V1)*(1-exp(-(time-TD1)/TAU1))
                  +(V1-V2)*(1-exp(-(time-TD2)/TAU2))
```

6.6.4 Piece-Wise Linear source

General Form:

```
VXXXXXXX N+ N- PWL(T1 V1 <T2 V2 T3 V3 T4 V4 ...>)
IXXXXXXX N+ N- PWL(T1 V1 <T2 V2 T3 V3 T4 V4 ...>)
```

Example:

```
VCLOCK 7 5 PWL(0 -7 10NS -7 11NS -3 17NS -3 18NS -7 50NS -7)
```

Parameters and default values:

Each pair of values (T_i, V_i) specifies that the value of the source is V_i (in Volts or Amps) at time=T_i. The value of the source at intermediate values of time is determined by using linear interpolation on the input values.

6.6.5 Single-Frequency FM source

General form:

```
VXXXXXXX N+ N- SFFM(V0 VA FC MDI FS)
IXXXXXXX N+ N- SFFM(V0 VA FC MDI FS)
```

Example:

```
V1 12 0 SFFM(0 1M 20K 5 1K)
```

parameters		default values	units
V0	(offset)		Volts or Amps
VA	(amplitude)		Volts or Amps
FC	(carrier frequency)	1/TSTOP	Hz
MDI	(modulation index)		
FS	(signal frequency)	1/TSTOP	Hz

The shape of the waveform is described by the following equation:

```
value = V0 + VA*sine((twopi*FC*time) + MDI*sine(twopi*FS*time))
```

6.7 Fuses

General Form:

```
GXXXXXXX N1 N2 POLY(1) N1 N2 -1234 LEN WIDTH THICK A B G0 S0 KOE POW
```

Example:

```
GFUSE 4 0 POLY(1) 4 0 -1234 20MIL 20MIL 0.175MIL 0.9U 1.4U
+ 8.9E16 1.8E16 20.E10 0.19
```

N1 and N2 are the nodes between which the fuse is connected. These node numbers must be repeated after the 'POLY(1)' descriptor, and must be followed by the value -1234 which flags this special case of a nonlinear voltage controlled current source as the fuse model. LEN is the length of the (assumed rectangular) fuse element, WIDTH is its width, and THICK is its thickness (all in meters - but note the use of the MIL unit in the example). The values A, B, G0, S0, KOE, and POW are the resistivity parameters from the 'FIRESET' code, written by Ronald S. Lee,[2] which has been adapted for use in SPICE version 2G6-SB2 by including a dynamic calculation of the rate of current rise used in scaling the resistivity parameters. See Lee's report for the details. It is recommended that the special editor program SPFEDIT be used to prepare SPICE input files containing fuse elements. This program aids in constructing the rather complicated input statement.

Note that the fuse element is intended for use in transient analyses only. Attempts to use it in other analyses in SPICE will give output corresponding to the low initial resistance value. Validity of results in such cases has not been checked!

Note also that since the fuse is implemented as a controlled current source it presents an OPEN CIRCUIT to DC! This is counter-intuitive to the behaviour of a fuse, and requires special attention in constructing a circuit model for SPICE to be sure there is a dc path from each node to ground.

6.8 Nonlinear Dependent Sources

SPICE allows circuits to contain dependent sources characterized by any of the four equations

```
i=f(v)        v=f(v)        i=f(i)        v=f(i)
```

where the functions must be polynomials, and the arguments may be multidimensional. The nature of these multidimensional polynomial functions and the way their coefficients are entered on SPICE input statements are explained in Appendix B.

7 SEMICONDUCTOR DEVICES

The elements described to this point typically require only a few parameter values. However, the models for the semiconductor devices that are included in the SPICE program require many parameter values. Often, many devices in a circuit are defined by the same set of device model parameters. For these reasons, a set of device model parameters is defined on a separate .MODEL statement and assigned a unique model name. The device element statements in SPICE then refer to the model name. This scheme alleviates the need to specify all of the model parameters on each device element statement.

Each device element statement contains the device name, the nodes to which the device is connected, and the device model name. In addition, other optional parameters may be specified for some devices: geometric factors and an initial condition.

The area factor used on the diode, BJT, and JFET device statements determines the number of equivalent parallel devices of a specified model. The affected parameters are marked with an asterisk under the heading 'area' in the model descriptions below. Several geometric factors associated with the channel and the drain and source diffusions can be specified on the MOSFET device statement.

Two different forms of initial conditions may be specified for some devices. The first form is included to improve the dc convergence for circuits that contain more than one stable state. If a device is specified OFF, the dc operating point is determined with the terminal voltages for that device set to zero. After convergence is obtained, the program continues to iterate to obtain the exact value for the terminal voltages. If a circuit has more than one dc stable state, the OFF option can be used to force the solution to correspond to a desired state. If a device is specified OFF when in reality the device is conducting, the program will still obtain the correct solution (assuming the solutions converge) but more iterations will be required since the program must independently converge to two separate solutions. The .NODESET statement serves a similar purpose as the OFF option. The .NODESET option is easier to apply and is the preferred means to aid convergence.

The second form of initial conditions are specified for use with the transient analysis. These are true 'initial conditions' as opposed to the convergence aids above. See the description of the .IC statement and the .TRAN statement for a detailed explanation of initial conditions.

7.1 Junction Diodes

General Form:

```
DXXXXXXX N+ N- MNAME <AREA> <OFF> <IC=VD>
```


Examples:

```
DBRIDGE 2 10 DIODE1
DCLMP 3 7 DMOD 3.0 IC=0.2
```

N+ and N− are the positive and negative nodes, respectively. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) starting condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using IC=VD is intended for use with the UIC option on the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point.

7.2 Bipolar Junction Transistors (BJT's)

General Form:

```
QXXXXXXX NC NB NE <NS> MNAME <AREA> <OFF> <IC=VBE,VCE>
```

Examples:

```
Q23 10 24 13 QMOD IC=0.6,5.0
Q50A 11 26 4 20 MOD1
```

NC, NB, and NE are the collector, base, and emitter nodes, respectively. NS is the (optional) substrate node. If unspecified, ground is used. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for the dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using IC=VBE,VCE is intended for use with the UIC option on the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point. See the .IC statement description for a better way to set transient initial conditions.

7.3 Junction Field-Effect Transistors (JFET's)

General Form:

```
JXXXXXXX ND NG NS MNAME <AREA> <OFF> <IC=VDS,VGS>
```

Example:

```
J1 7 2 3 JM1 OFF
```

ND, NG and NS are the drain, gate, and source nodes, respectively. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification, using IC=VDS,VGS is intended for use with the UIC option on the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point. See the .IC statement for a better way to set initial conditions.

7.4 MOSFET's

General Form:

```
MXXXXXXX ND NG NS NB MNAME <L=VAL> <W=VAL> <AD=VAL> <AS=VAL>
+ <PD=VAL> <PS=VAL> <NRD=VAL> <NRS=VAL> <OFF> <IC=VDS,VGS,VBS>
```

Examples:

```
M1 24 2 0 20 TYPE1
M31 2 17 6 10 MODM L=5U W=2U
M1 2 9 3 0 MOD1 L=10U W=5U AD=100P AS=100P PD=40U PS=40U
```

ND, NG, NS, and NB are the drain, gate, source, and bulk (substrate) nodes, respectively. MNAME is the model name. L and W are the channel length and width, in meters. AD and AS are the areas of the drain and source diffusions, in sq-meters. Note that the suffix U specifies microns (1E-6 m) and P sq-microns (1E-12 sq-m). If any of L, W, AD, or AS are not specified, default values are used. The use of defaults simplifies input file preparation, as well as the editing required if device geometries are to be changed. PD and PS are the perimeters of the drain and source junctions, in meters. NRD and NRS designate the equivalent number of squares of the drain and source diffusions; these values multiply the sheet resistance RSH specified on the .MODEL statement for an accurate representation of the parasitic series drain and source resistance of each transistor. PD and PS default to 0.0 while NRD and NRS to 1.0. OFF indicates an (optional) initial condition on the device for dc analysis. The (optional) initial condition specification using IC=VDS,VGS,VBS is intended for use with the UIC option on the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point. See the .IC statement for a better and more convenient way to specify transient initial conditions.

7.5 .MODEL Statement

General Form:

```
.MODEL MNAME TYPE(PNAME1=PVAL1 PNAME2=PVAL2 ... )
```

Example:

```
.MODEL MOD1 NPN (BF=50 IS=1E-13 VBF=50)
```

The .MODEL statement specifies a set of model parameters that will be used by one or more devices. MNAME is the model name, and type is one of the following seven types:

D	diode model
NPN	NPN BJT model
PNP	PNP BJT model
NJF	N-channel JFET model
PJF	P-channel JFET model
NMOS	N-channel MOSFET model
PMOS	P-channel MOSFET model

Parameter values are defined by appending the parameter name, as given below for each model type, followed by an equal sign and the parameter value. Model parameters that are not given a value are assigned the default values given below for each model type.

7.6 Diode Model

The dc characteristics of the diode are determined by the parameters IS and N. An ohmic resistance, RS, is included. Charge storage effects are modeled by a transit time, TT, and a nonlinear depletion layer capacitance which is determined by the parameters CJO, VJ, and M. The temperature dependence of the saturation current is defined by the parameters EG, the energy, and XTI, the saturation current temperature exponent. Reverse breakdown is modeled by an exponential increase in the reverse diode current and is determined by the parameters BV and IBV (both of which are positive numbers).

	name	parameter	units	default	example	area
1	IS	saturation current	A	1.0E-14	1.0E-14	*
2	RS	ohmic resistance	Ohm	0	10	*
3	N	emission coefficient	-	1	1.0	
4	TT	transit-time	sec	0	0.1Ns	
5	CJO	zero-bias junction capacitance	F	0	2PF	*
6	VJ	junction potential	V	1	0.6	
7	M	grading coefficient	-	0.5	0.5	
8	EG	activation energy	eV	1.11	1.11 Si	
					0.69 Sbd	
					0.67 Ge	
9	XTI	saturation-current temp. exp	-	3.0	3.0 jn	
					2.0 Sbd	
10	KF	flicker noise coefficient	-	0		
11	AF	flicker noise exponent	-	1		
12	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		
13	BV	reverse breakdown voltage	V	infinite	40.0	
14	IBV	current at breakdown voltage	A	1.0E-3		

7.7 BJT Models (both NPN and PNP)

The bipolar junction transistor model in SPICE is an adaptation of the integral charge control model of Gummel and Poon. This modified Gummel-Poon model extends the original model to include several effects at high bias levels. The model will automatically simplify to the simpler Ebers-Moll model when certain parameters are not specified. The parameter names used in the modified Gummel-Poon model have been chosen to be more easily understood by the program user, and to reflect better both physical and circuit design thinking.

The dc model is defined by the parameters IS, BF, NF, ISE, IKF, and NE which determine the forward current gain characteristics, IS, BR, NR, ISC, IKR, and NC which determine the reverse current gain characteristics, and VAF and VAR which determine the output conductance for forward and reverse regions. Three ohmic resistances RB, RC, and RE are included, where RB can be high current dependent. Base charge storage is modeled by forward and reverse transit times, TF and TR, the forward transit time TF being bias dependent if desired, and nonlinear depletion layer capacitances which are determined by CJE, VJE, and MJE for

the B-E junction , CJC, VJC, and MJC for the B-C junction and CJS, VJS, and MJS for the C-S (Collector-Substrate) junction. The temperature dependence of the saturation current, IS, is determined by the energy-gap, EG, and the saturation current temperature exponent, XTI. Additionally base current temperature dependence is modeled by the beta temperature exponent XTB in the new model.

The BJT parameters used in the modified Gummel-Poon model are listed below. The parameter names used in earlier versions of SPICE2 are still accepted.

Modified Gummel-Poon BJT Parameters:

	name	parameter	units	default	example	area
1	IS	transport saturation current	A	1.0E-16	1.0E-15	*
2	BF	ideal maximum forward beta	-	100	100	
3	NF	forward current emission coefficient	-	1.0	1	
4	VAF	forward Early voltage	V	infinite	200	
5	IKF	corner for forward beta high current roll-off	A	infinite	0.01	*
6	ISE	B-E leakage saturation current	A	0	1.0E-13	*
7	NE	B-E leakage emission coefficient	-	1.5	2	
8	BR	ideal maximum reverse beta	-	1	0.1	
9	NR	reverse current emission coefficient	-	1	1	
10	VAR	reverse Early voltage	V	infinite	200	
11	IKR	corner for reverse beta high current roll-off	A	infinite	0.01	*
12	ISC	B-C leakage saturation current	A	0	1.0E-13	*
13	NC	B-C leakage emission coefficient	-	2	1.5	
14	RB	zero bias base resistance	Ohms	0	100	*
15	IRB	current where base resistance falls halfway to its min value	A	infinite	0.1	*
16	RBM	minimum base resistance at high currents	Ohms	RB	10	*
17	RE	emitter resistance	Ohms	0	1	*
18	RC	collector resistance	Ohms	0	10	*
19	CJE	B-E zero-bias depletion capacitance	F	0	2PF	*
20	VJE	B-E built-in potential	V	0.75	0.6	
21	MJE	B-E junction exponential factor	-	0.33	0.33	
22	TF	ideal forward transit time	sec	0	0.1Ns	
23	XTF	coefficient for bias dependence of TF	-	0		
24	VTF	voltage describing VBC dependence of TF	V	infinite		
25	ITF	high-current parameter for effect on TF	A	0		*
26	PTF	excess phase at freq=1.0/(TF*2PI) Hz	deg	0		
27	CJC	B-C zero-bias depletion capacitance	F	0	2PF	*
28	VJC	B-C built-in potential	V	0.75	0.5	
29	MJC	B-C junction exponential factor	-	0.33	0.5	
30	XCJC	fraction of B-C depletion capacitance connected to internal base node	-	1		
31	TR	ideal reverse transit time	sec	0	10Ns	
32	CJS	zero-bias collector-substrate capacitance	F	0	2PF	*
33	VJS	substrate junction built-in potential	V	0.75		
34	MJS	substrate junction exponential factor	-	0	0.5	
35	XTB	forward and reverse beta temperature exponent	-	0		
36	EG	energy gap for temperature effect on IS	eV	1.11		
37	XTI	temperature expcnent for effect on IS	-	3		
38	KF	flicker-noise coefficient	-	0		
39	AF	flicker-noise exponent	-	1		
40	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		

7.8 JFET Models (both N and P Channel)

The JFET model is derived from the FET model of Shichman and Hodges. The dc characteristics are defined by the parameters VTO and BETA, which determine the variation of drain current with gate voltage, LAMBDA, which determines the output conductance, and IS, the saturation current of the two gate junctions. Two ohmic resistances, RD and RS, are included. Charge storage is modeled by nonlinear depletion layer capacitances for both gate junctions which vary as the -1/2 power of junction voltage and are defined by the parameters CGS, CGD, and PB.

	name	parameter	units	default	example	area
1	VTO	threshold voltage	V	-2.0	-2.0	
2	BETA	transconductance parameter	A/V**2	1.0E-4	1.0E-3	*
3	LAMBDA	channel length modulation parameter	1/V	0	1.0E-4	
4	RD	drain ohmic resistance	Ohm	0	100	*
5	RS	source ohmic resistance	Ohm	0	100	*
6	CGS	zero-bias G-S junction capacitance	F	0	5PF	*
7	CGD	zero-bias G-D junction capacitance	F	0	1PF	*
8	PB	gate junction potential	V	1	0.6	
9	IS	gate junction saturation current	A	1.0E-14	1.0E-14	*
10	KF	flicker noise coefficient	-	0		
11	AF	flicker noise exponent	-	1		
12	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		

7.9 MOSFET Models (both N and P channel)

SPICE2 provides three MOSFET device models, which differ in the formulation of the I-V characteristic. The variable LEVEL specifies the model to be used:

LEVEL=1 -> Shichman-Hodges
LEVEL=2 -> MOS2 (as described in [1])
LEVEL=3 -> MOS3, a semi-empirical model(see [1])

The dc characteristics of the level 1 through level 3 MOSFETs are defined by the device parameters VTO, KP, LAMBDA, PHI and GAMMA. These parameters are computed by SPICE if process parameters (NSUB, TOX, ...) are given, but user-specified values always override. VTO is positive (negative) for enhancement mode and negative (positive) for depletion mode N-channel (P-channel) devices. Charge storage is modeled by three constant capacitors, CGSO, CGDO, and CGBO which represent overlap capacitances, by the nonlinear thin-oxide capacitance which is distributed among the gate, source, drain, and bulk regions, and by the nonlinear depletion-layer capacitances for both substrate junctions divided into bottom and periphery, which vary as the MJ and MJSW power of junction voltage respectively, and are determined by the parameters CBD, CBS, CJ, CJSW, MJ, MJSW and PB. There are two built-in models of the charge storage effects associated with the thin-oxide. The default is the piecewise linear voltage-dependent capacitance model proposed by Meyer. The second choice is the charge-controlled capacitance model of Ward and Dutton [1]. The XQC model parameter acts as a flag and a coefficient at the same time. As the former it causes the program to use Meyer's model whenever larger than 0.5 or not specified, and the charge-controlled model when between 0 and 0.5. In the latter case its value defines the share of the channel charge associated with the drain terminal in the saturation region. The thin-oxide charge storage effects are treated slightly different for the LEVEL=1 model. These voltage-dependent capacitances are included only if TOX is specified in the input description and they are represented using Meyer's formulation.

There is some overlap among the parameters describing the junctions, e.g. the reverse current can be input either as IS (in A) or as JS (in A/m**2). Whereas the first is an absolute value the second is multiplied by AD and AS to give the reverse current of the drain and source junctions respectively. This methodology has been chosen since there is no sense in relating always junction characteristics with AD and AS entered on the device statement; the areas can be defaulted. The same idea applies also to the zero-bias junction capacitances CBD and CBS (in F) on one hand, and CJ (in F/m**2) on the other. The parasitic drain and source series resistance can be expressed as either RD and RS (in ohms) or RSH (in ohms/sq.), the latter being multiplied by the number of squares NRD and NRS input on the device statement.

SPICE level 1 to level 3 parameters:

	name	parameter	units	default	example	area
1	LEVEL	model index	-	1		
2	VTO	zero-bias threshold voltage	V	0.0	1.0	
3	KP	transconductance parameter	A/V**2	2.0E-5	3.1E-5	
4	GAMMA	bulk threshold parameter	V**0.5	0.0	0.37	
5	PHI	surface potential	V	0.6	0.65	

6	LAMBDA	channel-length modulation (MOS1 and MOS2 only)	1/V	0.0	0.02	
7	RD	drain ohmic resistance	Ohm	0.0	1.0	
8	RS	source ohmic resistance	Ohm	0.0	1.0	
9	CBD	zero-bias B-D junction capacitance	F	0.0	20FF	
10	CBS	zero-bias B-S junction capacitance	F	0.0	20FF	
11	IS	bulk junction saturation current	A	1.0E-14	1.0E-15	
12	PB	bulk junction potential	V	0.8	0.87	
13	CGSO	gate-source overlap capacitance per meter channel width	F/m	0.0	4.0E-11	
14	CGDO	gate-drain overlap capacitance per meter channel width	F/m	0.0	4.0E-11	
15	CGBO	gate-bulk overlap capacitance per meter channel length	F/m	0.0	2.0E-10	
16	RSH	drain and source diffusion sheet resistance	Ohm/sq.	0.0	10.0	
17	CJ	zero-bias bulk junction bottom cap. per sq-meter of junction area	F/m**2	0.0	2.0E-4	
18	MJ	bulk junction bottom grading coef.	-	0.5	0.5	
19	CJSW	zero-bias bulk junction sidewall cap. per meter of junction perimeter	F/m	0.0	1.0E-9	
20	MJSW	bulk junction sidewall grading coef.	-	0.50(level1) 0.33(level2,3)		
21	JS	bulk junction saturation current per sq-meter of junction area	A/m**2		1.0E-8	1.0E-9
22	TOX	oxide thickness	meter	1.0E-7	1.0E-7	
23	NSUB	substrate doping	1/cm**3	0.0	4.0E15	
24	NSS	surface state density	1/cm**2	0.0	1.0E10	
25	NFS	fast surface state density	1/cm**2	0.0	1.0E10	
26	TPG	type of gate material: +1 opp. to substrate -1 same as substrate 0 Al gate	-	1.0		
27	XJ	metallurgical junction depth	meter	0.0	1U	
28	LD	lateral diffusion	meter	0.0	0.8U	
29	UO	surface mobility	cm**2/V-s	600	700	
30	UCRIT	critical field for mobility degradation (MOS2 only)	V/cm	1.0E4	1.0E4	
31	UEXP	critical field exponent in mobility degradation (MOS2 only)	-	0.0	0.1	
32	UTRA	transverse field coef (mobility) (deleted for MOS2)	-	0.0	0.3	
33	VMAX	maximum drift velocity of carriers	m/s	0.0	5.0E4	

34	NEFF	total channel charge (fixed and mobile) coefficient (MOS2 only)	-	1.0	5.0
35	XQC	thin-oxide capacitance model flag and coefficient of channel charge share attributed to drain (0-0.5)	-	1.0	0.4
36	KF	flicker noise coefficient	-	0.0	1.0E-26
37	AF	flicker noise exponent	-	1.0	1.2
38	FC	coefficient for forward-bias depletion capacitance formula	-	0.5	
39	DELTA	width effect on threshold voltage (MOS2 and MOS3)	-	0.0	1.0
40	THETA	mobility modulation (MOS3 only)	1/V	0.0	0.1
41	ETA	static feedback (MOS3 only)	-	0.0	1.0
42	KAPPA	saturation field factor (MOS3 only)	-	0.2	0.5

8 SUBCIRCUITS

A subcircuit that consists of SPICE elements can be defined and referenced in a fashion similar to device models. The subcircuit is defined in the input file by a grouping of element statements; the program then automatically inserts the group of elements wherever the subcircuit is referenced. There is no limit on the size or complexity of subcircuits, and subcircuits may contain other subcircuits. An example of subcircuit usage is given in Appendix A.

8.1 .SUBCKT Statement

General Form:

```
.SUBCKT subnam N1 <N2 N3 ...>
```

Example:

```
.SUBCKT OPAMP 1 2 3 4
```

A circuit definition is begun with a .SUBCKT statement. SUBNAM is the subcircuit name, and N1, N2, ... are the external nodes, which cannot be zero. The group of element statements which immediately follow the .SUBCKT statement define the subcircuit. The last statement in a subcircuit definition is the .ENDS statement (see below). Control statements may not appear within a subcircuit definition; however, subcircuit definitions may contain anything else, including other subcircuit definitions, device models, and subcircuit calls (see below). Note that any device models or subcircuit definitions included as part of a subcircuit definition are strictly local (i.e., such models and definitions are not known outside the subcircuit definition). Also, any element nodes not included on the .SUBCKT statement are strictly local, with the exception of 0 (ground) which is always global.

8.2 .ENDS Statement

General Form:


```
.ENDS <SUBNAM>
```

Example:

```
.ENDS OPAMP
```

This statement must be the last one for any subcircuit definition. The subcircuit name, if included, indicates which subcircuit definition is being terminated; if omitted, all subcircuits being defined are terminated. The name is needed only when nested subcircuit definitions are being made.

8.3 Subcircuit Calls

General Form:

```
XYYYYYYY N1 <N2 N3 ...> SUBNAM
```

Example:

```
X1 2 4 17 3 1 MULTI
```

Subcircuits are used in SPICE by specifying pseudo-elements beginning with the letter X, followed by the circuit nodes to be used in expanding the subcircuit. .bp .sp .5i

9 CONTROL STATEMENTS

9.1 .TEMP Statement

General Form:

```
.TEMP T1 <T2 <T3 ...>>
```

Example:

```
.TEMP -55.0 25.0 125.0
```

This statement specifies the temperatures at which the circuit is to be simulated. T1, T2, ... are the different temperatures, in degrees C. Temperatures less than -223.0 deg C are ignored. Model data are specified at TNOM degrees (see the .OPTIONS statement for TNOM); if the .TEMP statement is omitted, the simulation will be performed at a temperature equal to TNOM.

9.2 .WIDTH Statement

General Form:

```
.WIDTH IN=COLNUM OUT=COLNUM
```

Examples:

```
.WIDTH IN=72 OUT=133
```

COLNUM is the last column read from each line of input; the setting takes effect with the next line read. The default value for COLNUM is 80. The out parameter specifies the output print width. Permissible values for the output print width are 80 and 133.

9.3 .OPTIONS Statement

General Form:

```
.OPTIONS OPT1 OPT2 ... (or OPT=OPTVAL ...)
```

Examples:

```
.OPTIONS RELTOL=.005 TRTOL=8
```

This statement allows the user to reset program control and user options for specific simulation purposes. Any combination of the following options may be included, in any order. 'x' (below) represents some positive number.

option effect

ACCT causes accounting and run time statistics to be printed

LIST causes the summary listing of the input data to be printed

NOMOD suppresses the printout of the model parameters.

NOPAGE suppresses page ejects

NODE causes the printing of the node table.

OPTS causes the option values to be printed.

GMIN=x resets the value of GMIN, the minimum conductance allowed by the program. The default value is 1.0E-12.

RELTOL=x resets the relative error tolerance of the program. The default value is 0.001 (0.1 percent).

ABSTOL=x resets the absolute current error tolerance of the program. The default value is 1 picoamp.

VNTOL=x resets the absolute voltage error tolerance of the program. The default value is 1 microvolt.

TRTOL=x resets the transient error tolerance. The default value is 7.0. This parameter is an estimate of the factor by which SPICE overestimates the actual truncation error.

CHGTOL=x resets the charge tolerance of the program. The default value is 1.0E-14.

PIVTOL=x resets the absolute minimum value for a matrix entry to be accepted as a pivot. The default value is 1.0E-13.

PIVREL=x resets the relative ratio between the largest column entry and an acceptable pivot value. The default value is 1.0E-3. In the numerical pivoting algorithm the allowed minimum pivot value is determined by EPSREL=AMAX1(PIVREL*MAXVAL,PIVTOL) where MAXVAL is the maximum element in the column where a pivot is sought (partial pivoting).

NUMDGT=x resets the number of significant digits printed for output variable values. X must satisfy the relation $0 < x < 8$. The default value is 4. Note: this option is independent of the error tolerance used by SPICE (i.e., if the values of options RELTOL, ABSTOL, etc., are not changed then one may be printing numerical 'noise' for NUMDGT > 4.

TNOM=x resets the nominal temperature. The default value is 27 deg C (300 deg K).

ITL1=x resets the dc iteration limit. The default is 100.

ITL2=x resets the dc transfer curve iteration limit. The default is 50.

ITL3=x resets the lower transient analysis iteration limit. the default value is 4.

ITL4=x resets the transient analysis timepoint iteration limit. the default is 10.

ITL5=x resets the transient analysis total iteration limit. the default is 5000. Set ITL5=0 to omit this test.

CPTIME=x the maximum cpu-time in seconds allowed for this job.

LIMTIM=x resets the amount of cpu time reserved by SPICE for generating plots should a cpu time-limit cause job termination. The default value is 2 (seconds).

LIMPTS=x resets the total number of points that can be printed or plotted in a dc, ac, or transient analysis. The default value is 201.

LVLCOD=x if x is 2 (two), then machine code for the matrix solution will be generated. Otherwise, no machine code is generated. The default value is 2. Applies only to CDC computers.

LVLTIM=x if x is 1 (one), the iteration timestep control is used. if x is 2 (two), the truncation-error timestep is used. The default value is 2. If method=Gear and MAXORD>2 then LVLTIM is set to 2 by SPICE.

METHOD=name sets the numerical integration method used by SPICE. Possible names are Gear or trapezoidal. The default is trapezoidal.

MAXORD=x sets the maximum order for the integration method if Gear's variable-order method is used. X must be between 2 and 6. The default value is 2.

DEFL=x resets the value for MOS channel length; the default is 100.0 micrometer.

DEFW=x resets the value for MOS channel width; the default is 100.0 micrometer.

DEFAD=x resets the value for MOS drain diffusion area; the default is 0.0.

DEFAS=x resets the value for MOS source diffusion area; the default is 0.0.

9.4 .OP Statement

General Form:

```
.OP
```

The inclusion of this statement in an input file will force SPICE to determine the dc operating point of the circuit with inductors shorted and capacitors opened. Note: a dc analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an ac small-signal analysis to determine the linearized, small-signal models for nonlinear devices.

SPICE performs a dc operating point analysis if no other analyses are requested.

9.5 .DC Statement

General Form:

```
.DC SRCNAM VSTART VSTOP VINCR [SRC2 START2 STOP2 INCR2]
```

Examples:

```
.DC VIN 0.25 5.0 0.25
.DC VDS 0 10 .5 VGS 0 5 1
.DC VCE 0 10 .25 IB 0 10U 1U
```

This statement defines the dc transfer curve source and sweep limits. SRCNAM is the name of an independent voltage or current source. VSTART, VSTOP, and VINCR are the starting, final, and incrementing values respectively. The first example will cause the value of the voltage source VIN to be swept from 0.25 Volts to 5.0 Volts in increments of 0.25 Volts. A second source (SRC2) may optionally be specified with associated sweep parameters. In this case, the first source will be swept over its range for each value of the second source. This option can be useful for obtaining semiconductor device output characteristics. See the second example data file in that section of the guide.

9.6 .NODESET Statement

General Form:

```
.NODESET V(NODNUM)=VAL V(NODNUM)=VAL ...
```

Example:

```
.NODESET V(12)=4.5 V(4)=2.23
```

This statement helps the program find the dc or initial transient solution by making a preliminary pass with the specified nodes held to the given voltages. The restriction is then released and the iteration continues to the true solution. The .NODESET statement may be necessary for convergence on bistable or astable circuits. In general, this statement should not be necessary.

9.7 .IC Statement

General Form:

```
.IC V(NODNUM)=VAL V(NODNUM)=VAL ...
```

Example: Ni

```
.IC V(11)=5 V(4)=-5 V(2)=2.2
```

This statement is for setting transient initial conditions. It has two different interpretations, depending on whether the UIC parameter is specified on the .TRAN statement. Also, one should not confuse this statement with the .NODESET statement. The .NODESET statement is only to help dc convergence, and does not affect final bias solution (except for multi-stable circuits). The two interpretations of this statement are as follows:

1. When the UIC parameter is specified on the .TRAN statement, then the node voltages specified on the .IC statement are used to compute the capacitor, diode, BJT, JFET, and MOSFET initial conditions. This is equivalent to specifying the IC=... parameter on each device statement, but is much more convenient. The IC=... parameter can still be specified and will take precedence over the .IC values. Since no dc bias (initial transient) solution is computed before the transient analysis, one should take care to specify all dc source voltages on the .IC statement if they are to be used to compute device initial conditions.

2. When the UIC parameter is not specified on the .TRAN statement, the dc bias (initial transient) solution will be computed before the transient analysis. In this case, the node voltages specified on the .IC statement will be forced to the desired initial values during the bias solution. During transient analysis, the constraint on these node voltages is removed.

9.8 .TF Statement

General Form:

```
.TF OUTVAR INSRC
```

Examples:

```
.TF V(5,3) VIN
.TF I(VLOAD) VIN
```

This statement defines the small-signal output and input for the dc small- signal analysis. OUTVAR is the small-signal output variable and INSRC is the small-signal input source. If this statement is included, SPICE will compute the dc small-signal value of the transfer function (output/input), input resistance, and output resistance. For the first example, SPICE would compute the ratio of V(5,3) to VIN, the small-signal input resistance at VIN, and the small-signal output resistance measured across nodes 5 and 3.

9.9 .SENS Statement

General Form:

```
.SENS OV1 <OV2 ... >
```

Example:

```
.SENS V(9) V(4,3) V(17) I(VCC)
```

If a .SENS statement is included in the input file, SPICE will determine the dc small-signal sensitivities of each specified output variable with respect to every circuit parameter. Note: for large circuits, large amounts of output can be generated.

9.10 .AC Statement

General Form:

```
.AC DEC ND FSTART FSTOP
.AC OCT NO FSTART FSTOP
.AC LIN NP FSTART FSTOP
```

Examples:

```
.AC DEC 10 1 10K
.AC DEC 10 1K 100MEG
.AC LIN 100 1 100HZ
```

DEC stands for decade variation, and ND is the number of points per decade. OCT stands for octave variation, and NO is the number of points per octave. LIN stands for linear variation, and NP is the number of points. FSTART is the starting frequency, and FSTOP is the final frequency. If this statement is included in the file, SPICE will perform an ac analysis of the circuit over the specified frequency range. Note that in order for this analysis to be meaningful, at least one independent source must have been specified with an ac value.

9.11 .DISTO Statement

General Form:

```
.DISTO RLOAD <INTER <SKW2 <REFPWR <SPW2>>>>
```

Example:

```
.DISTO RL 2 0.95 1.0E-3 0.75
```

This statement controls whether SPICE will compute the distortion characteristic of the circuit in a small-signal mode as a part of the ac small-signal sinusoidal steady-state analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input; let the two frequencies be f1 (the nominal analysis frequency) and f2 (=SKW2*f1). The program then computes the following distortion measures:


```
HD2  - the magnitude of the frequency component 2*f1 assuming that f2
       is not present.
HD3  - the magnitude of the frequency component 3*f1 assuming that f2
       is not present.
SIM2 - the magnitude of the frequency component f1 + f2.
DIM2 - the magnitude of the frequency component f1 - f2.
DIM3 - the magnitude of the frequency component 2*f1 - f2.
```

RLOAD is the name of the output load resistor into which all distortion power products are to be computed. INTER is the interval at which the summary printout of the contributions of all nonlinear devices to the total distortion is to be printed. If omitted or set to zero, no summary printout will be made. REFPWR is the reference power level used in computing the distortion products; if omitted, a value of 1 mW (that is, dbm) is used. SKW2 is the ratio of f2 to f1. If omitted, a value of 0.9 is used (i.e., f2 = 0.9*f1). SPW2 is the amplitude of f2. If omitted, a value of 1.0 is assumed.

The distortion measures HD2, HD3, SIM2, DIM2, and DIM3 may also be be printed and/or plotted (see the description of the .PRINT and .PLOT statements).

9.12 .NOISE Statement

General Form:

```
.NOISE OUTV INSRC NUMS
```

Example:

```
.NOISE V(5) VIN 10
```

This statement controls the noise analysis of the circuit. The noise analysis is performed in conjunction with the ac analysis (see .AC statement). OUTV is an output voltage which defines the summing point. INSRC is the name of the independent voltage or current source which is the noise input reference. NUMS is the summary interval. SPICE will compute the equivalent output noise at the specified output as well as the equivalent input noise at the specified input. In addition, the contributions of every noise generator in the circuit will be printed at every NUMS frequency points (the summary interval). If NUMS is zero, no summary printout will be made.

The output noise and the equivalent input noise may also be printed and/or plotted (see the description of the .PRINT and .PLOT statements).

9.13 .TRAN Statement

General Form:

```
.TRAN TSTEP TSTOP <TSTART <TMAX>> \"<UIC>
```

Examples:

```
.TRAN 1NS 100NS
.TRAN 1NS 1000NS 500NS
.TRAN 10NS 1US\" UIC
```

TSTEP is the printing or plotting increment for line-printer output. For use with the post-processor, TSTEP is the suggested computing increment. TSTOP is the final time, and TSTART is the initial time. If TSTART is omitted, it is assumed to be zero. The transient analysis always begins at time zero. In the interval <zero, TSTART>, the circuit is analyzed (to reach a steady state), but no outputs are stored. In the interval <TSTART, TSTOP>, the circuit is analyzed and outputs are stored. TMAX is the maximum stepsize that SPICE will use. (For default, the program chooses either TSTEP or (TSTOP-TSTART)/50.0, whichever is smaller.) TMAX is useful when one wishes to guarantee a computing interval which is smaller than the printer increment, TSTEP. [Otherwise the printed values will be linearly interpolated between the calculated ones. The "rawfile" output will contain the values at the times they are calculated, from 0 to TSTOP, regardless of the value of any of these parameters (except TSTOP).]

UIC (use initial conditions) is an optional keyword which indicates that the user does not want SPICE to solve for the quiescent operating point before beginning the transient analysis. If this keyword is specified, SPICE uses the values specified using IC=...on the various elements as the initial transient condition and proceeds with the analysis. If the .IC statement has been specified, then the node voltages on the .IC statement are used to compute the intitial conditions for the devices. Look at the description on the .IC statement for its interpretation when UIC is not specified.

9.14 .FOUR Statement

General Form:

```
.FOUR FREQ OV1 <OV2 OV3 ...>
```

Example:

```
.FOUR 100K  V(5)
```

This statement controls whether SPICE performs a Fourier analysis as a part of the transient analysis. FREQ is the fundamental frequency, and OV1, ..., are the output variables for which the analysis is desired. The Fourier analysis is performed over the interval <TSTOP-period, TSTOP>, where TSTOP is the final time specified for the transient analysis, and period is one period of the fundamental frequency. The dc component and the first nine components are determined. For maximum accuracy, TMAX (see the .TRAN statement) should be set to period/100.0 (or less for very high-Q circuits).

9.15 .PRINT Statements

General Form:

```
.PRINT PRTYPE OV1 <OV2 ... OV8>
```

Examples:

```
.PRINT TRAN V(4) I(VIN)
.PRINT AC VM(4,2) VR(7) VP(8,3)
```



```
.PRINT DC V(2) I(VSRC) V(23,17)
.PRINT NOISE INOISE
.PRINT DISTO HD3 SIM2(DB)
```

This statement defines the contents of a tabular listing of one to eight output variables. PRTYPE is the type of the analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The form for voltage or current output variables is as follows:

V(N1<,N2>) specifies the voltage difference between nodes N1 and N2. If N2 (and the preceding comma) is omitted, ground (0) is assumed. For the ac analysis, five additional outputs can be accessed by replacing the letter V by:

VR -	real part
VI -	imaginary part
VM -	magnitude
VP -	phase
VDB -	20*log10(magnitude)

I(VXXXXXXX) specifies the current flowing in the independent voltage source named VXXXXXXX. Positive current flows from the positive node, through the source, to the negative node. For the ac analysis, the corresponding replacements for the letter I may be made in the same way as described for voltage outputs.

Output variables for the noise and distortion analyses have a different general form from that of the other analyses, i.e.

OV<(X)> where OV is any of ONOISE (output noise), INOISE (equivalent input noise), D2, HD3, SIM2, DIM2, or DIM3 (see description of distortion analysis), and X may be any of:

R -	real part
I -	imaginary part
M -	magnitude (default if nothing specified)
P -	phase
DB -	20*log10(magnitude)

thus, SIM2 (or SIM2(M)) describes the magnitude of the SIM2 distortion measure, while HD2(R) describes the real part of the HD2 distortion measure.

There is no limit on the number of .PRINT statements for each type of analysis.

9.16 .PLOT Statements

General Form:

```
.PLOT PLTYPE OV1 <(PLO1,PHI1)> <OV2 <(PLO2,PHI2)> ... OV8>
```

Examples:

```
.PLOT DC V(4) V(5) V(1)
.PLOT TRAN V(17,5) (2,5) I(VIN) V(17) (1,9)
.PLOT AC VM(5) VM(31,24) VDB(5) VP(5)
.PLOT DISTO HD2 HD3(R) SIM2
.PLOT TRAN V(5,3) V(4) (0,5) V(7) (0,10)
```

This statement defines the contents of one plot of from one to eight output variables. PLTYPE is the type of analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The syntax for the OVI is identical to that for the .PRINT statement, described above.

The optional plot limits (PLO,PHI) may be specified after any of the output variables. All output variables to the left of a pair of plot limits (PLO,PHI) will be plotted using the same lower and upper plot bounds. If plot limits are not specified, SPICE will automatically determine the minimum and maximum values of all output variables being plotted and scale the plot to fit. More than one scale will be used if the output variable values warrant (i.e., mixing output variables with values which are orders-of-magnitude different still gives readable plots).

The overlap of two or more traces on any plot is indicated by the letter X.

When more than one output variable appears on the same plot, the first variable specified will be printed as well as plotted. If a printout of all variables is desired, then a companion .PRINT statement should be included.

There is no limit on the number of .PLOT statements specified for each type of analysis.

References

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

[2] Ronald S. Lee, Fireset, UCID-21322, Lawrence Livermore National Laboratory, February 19, 1988.

10 APPENDIX A: EXAMPLE DATA FILES

10.1 Circuit 1

The following file determines the dc operating point and small-signal transfer function of a simple differential pair. In addition, the ac small-signal response is computed over the frequency range 1Hz to 100MEGHz.

```
SIMPLE DIFFERENTIAL PAIR
VCC 7 0 12
VEE 8 0 -12
VIN 1 0 AC 1
RS1 1 2 1K
RS2 6 0 1K
Q1 3 2 4 MOD1
Q2 5 6 4 MOD1
RC1 7 3 10K
RC2 7 5 10K
RE 4 8 10K
.MODEL MOD1 NPN BF=50 VAF=50 IS=1.E-12 RB=100 CJC=.5PF TF=.6NS
.TF V(5) VIN
.AC DEC 10 1 100MEG
.PLOT AC VM(5) VP(5)
.PRINT AC VM(5) VP(5)
.END
```

10.2 Circuit 2

The following file computes the output characteristics of a MOSFET device over the range 0-10V for VDS and 0-5V for VGS.

```
MOS OUTPUT CHARACTERISTICS
.OPTIONS NODE NOPAGE
VDS 3 0
VGS 2 0
M1 1 2 0 0 MOD1 L=4U W=6U AD=10P AS=10P
.MODEL MOD1 NMOS VTO=-2 NSUB=1.0E15 UO=550
* VIDS MEASURES ID, WE COULD HAVE USED VDS, BUT ID WOULD BE NEGATIVE
VIDS 3 1
.DC VDS 0 10 .5 VGS 0 5 1
.PRINT DC I(VIDS) V(2)
.PLOT DC I(VIDS)
.END
```

10.3 Circuit 3

The following file determines the dc transfer curve and the transient pulse response of a simple RTL inverter. The input is a pulse from 0 to 5 Volts with delay, rise, and fall times of 2ns

and a pulse width of 30ns. The transient interval is 0 to 100ns, with printing to be done every nanosecond.

```
SIMPLE RTL INVERTER
VCC 4 0 5
VIN 1 0 PULSE 0 5 2NS 2NS 2NS 30NS
RB 1 2 10K
Q1 3 2 0 Q1
RC 3 4 1K
.PLOT DC V(3)
.PLOT TRAN V(3) (0,5)
.PRINT TRAN V(3)
.MODEL Q1 NPN BF 20 RB 100 TF .1NS CJC 2PF
.DC VIN 0 5 0.1
.TRAN 1NS 100NS
.END
```

10.4 Circuit 4

The following file simulates a four-bit binary adder, using several subcircuits to describe various pieces of the overall circuit.

```
ADDER - 4 BIT ALL-NAND-GATE BINARY ADDER

    *** SUBCIRCUIT DEFINITIONS

.SUBCKT NAND 1 2 3 4
*   NODES:  INPUT(2), OUTPUT, VCC
Q1 9 5 1 QMOD
D1CLAMP 0 1 DMOD
Q2 9 5 2 QMOD
D2CLAMP 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDROP 10 3 DMOD
Q5 3 8 0 QMOD
.ENDS NAND
.SUBCKT ONEBIT 1 2 3 4 5 6
*   NODES:  INPUT(2), CARRY-IN, OUTPUT, CARRY-OUT, VCC
X1 1 2 7 6 NAND
X2 1 7 8 6 NAND
X3 2 7 9 6 NAND
```



```
X4 8 9 10 6 NAND
X5 3 10 11 6 NAND
X6 3 11 12 6 NAND
X7 10 11 13 6 NAND
X8 12 13 4 6 NAND
X9 11 7 5 6 NAND
.ENDS ONEBIT
.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
*   NODES:  INPUT - BIT0(2) / BIT1(2), OUTPUT - BIT0 / BIT1,
*           CARRY-IN, CARRY-OUT, VCC
X1 1 2 7 5 10 9 ONEBIT
X2 3 4 10 6 8 9 ONEBIT
.ENDS TWOBIT

.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
*   NODES:  INPUT - BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
*           OUTPUT - BIT0 / BIT1 / BIT2 / BIT3, CARRY-IN, CARRY-OUT, VCC
X1 1 2 3 4 9 10 13 16 15 TWOBIT
X2 5 6 7 8 11 12 16 14 15 TWOBIT
.ENDS FOURBIT

    *** DEFINE NOMINAL CIRCUIT

.MODEL DMOD D
.MODEL QMOD NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
VCC 99 0 DC 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS   10NS   50NS)
VIN1B 2 0 PULSE(0 3 0 10NS 10NS   20NS  100NS)
VIN2A 3 0 PULSE(0 3 0 10NS 10NS   40NS  200NS)
VIN2B 4 0 PULSE(0 3 0 10NS 10NS   80NS  400NS)
VIN3A 5 0 PULSE(0 3 0 10NS 10NS  160NS  800NS)
VIN3B 6 0 PULSE(0 3 0 10NS 10NS  320NS 1600NS)
VIN4A 7 0 PULSE(0 3 0 10NS 10NS  640NS 3200NS)
VIN4B 8 0 PULSE(0 3 0 10NS 10NS 1280NS 6400NS)
X1 1 2 3 4 5 6 7 8 9 10 11 12 0 13 99 FOURBIT
RBIT0 9 0 1K
RBIT1 10 0 1K
RBIT2 11 0 1K
RBIT3 12 0 1K
RCOUT 13 0 1K
.PLOT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PLOT TRAN V(9) V(10) V(11) V(12) V(13)
.PRINT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PRINT TRAN V(9) V(10) V(11) V(12) V(13)
```

```
    *** (FOR THOSE WITH MONEY (AND MEMORY) TO BURN)
.TRAN 1NS 6400NS

.OPTIONS ACCT LIST NODE LIMPTS=6401
.END
```

10.5 Circuit 5

The following file simulates a transmission-line inverter. Two transmission-line elements are required since two propagation modes are excited. In the case of a coaxial line, the first line (T1) models the inner conductor with respect to the shield, and the second line (T2) models the shield with respect to the outside world.

```
TRANSMISSION-LINE INVERTER
V1 1 0 PULSE(0 1 0 0.1N)
R1 1 2 50
X1 2 0 0 4 TLINE
R2 4 0 50
.SUBCKT TLINE 1 2 3 4
T1 1 2 3 4 Z0=50 TD=1.5NS
T2 2 0 4 0 Z0=100 TD=1NS
.ENDS TLINE
.TRAN 0.1NS 20NS
.PLOT TRAN V(2) V(4)
.END
```

10.6 Saturating inductor circuit

```
LTANT1 - test of saturating inductance model in SPICE2SB
*Saturating inductance model is valid only for transient analysis.
*
*The following line specifies a transient analysis, saving results
*every 0.01S and continuing from time 0 to time 1S.
.TRAN 0.01 1
*
*The following line specifies an independant current source, connected
*between node 1 and ground (node 0 is always ground), with the current
*going through the source from node 1 to node 0 and out node 0.
*The current is a sine wave of amplitude -2amps, frequency 1Hz, zero
*phase shift, no offset and no damping:
I1 1 0 SIN(0 -2 1 0 0)
*
*The following line specifies the saturating inductor is connected
*between nodes 3 and 0.  The word "POLY" signals a nonlinear inductance,
*and the first number, -1234, is the flag to indicate the hyperbolic
*tangent saturation model is the one to use.  The numbers 1, .001, 1,1
```



```
*mean, respectively, initial inductance(in Henries), inductance after
*saturation, saturation current (in amps), and power (and root) used
*for the hyperbolic tangent and argument in the model.
L1 3 0 POLY -1234 1 0.001 1 1
*
*The next line specifies an ammeter measuring current flowing from node 3
*to node 3 through it.  The current is referenced later as I(VCUR):
VCUR 1 3 DC 0
*
*The next line specifies a large resistance connected from node 3 to node
*0.  It should not be needed, but a bug in SPICE2G6 makes it necessary to
*avoid an error of "time step too small".  Note that SPICE uses the
*abbreviation M for milli, but uses MEG for mega.  The "EG" is required
*or the value will be off six orders of magnetude!
R1 3 0 10MEG
*
*The following lines place a voltage controlled current source, having
*controlling voltage equal to the voltage across the inductor, in series
*with a one farad capacitor.  (The resistor in parallel is required since
*SPICE requires a DC path to ground from each node.)  The voltage across
*the capacitor is then numerically equal to the flux through the inductor.
*(Note that one cannot write 1F for the capacitance value, as SPICE
*interprets the F as a unit of 10^-15.)
C2 2 0 1
G2 2 0 1 0 -1
R2 2 0 10MEG
*
*The following line calls for a printout of the current through the
*inductor and the flux.  A plot of V(2) versus I(VCUR) will be a scaled
*one of the model B-H curve.
.PRINT TRAN I(VCUR) V(2)
*
*All SPICE input files must end with the following line:
.END
```

10.7 Fuse circuit

```
TFUSE10 - test circuit for SPICE2SB fuse model
*Same values are used below as in 1st example in FIRESET paper by Ron Lee.
*
*Transient analysis is the only one valid when the fuse model
*is used in SPICE2SB.  The following line calls for a transient
*analysis, saving output every 5 nanoseconds and running for a
*total of one microsecond.  The starting time is zero and the
*minimum time step for numerical integration is 1 nanosecond.
```

```
*Initial conditions specified on the element lines are to be used.
*(Note: If the SPICE error message,
* "ERROR*:  INTERNAL TIMESTEP TOO SMALL IN TRANSIENT ANALYSIS",
* results from an analysis, the first thing to try is to reduce
* the minimum time step.)
.TRAN 5NS 1US 0 1NS UIC
*
*The following line calls for a printout of the current in the
*fuse, I(VDC), and the voltage across it, V(4).
.PRINT TRAN I(VDC) V(4)
*
*The following line calls for a plot versus time of these same variables:
.PLOT TRAN I(VDC) V(4)
*
*The following line specifies that a capacitor of value 0.5microfarads
*with initial voltage 5killovolts is located between nodes 1 and 0
*(node 0 is ground), with node 1 positive:
CSOURCE 1 0 0.5UF IC=5KV
*
*The following line specifies a resistor of value 75milliohms from
*node 1 to node 2:
RS 1 2 0.075
*
*(Note, the following line DOES NOT GIVE THE CORRECT VALUE:
*RS 1 2 75MILLIOHM
*as SPICE takes MIL as a suffix for multiplication by 25.4E-6.
*Be VERY CAREFUL when using unit suffixes with SPICE!)
*
*The following line specifies an inductor of value 90nanohenries from
*node 2 to node 3:
LS 2 3 90NANOHENRY
*
*The following line specifies an ammeter for positive current flowing
*from node 3 to node 4 through it.  This current is I(VDC):
VDC 3 4 DC 0
*
*The next line specifies the fuse model is connected between nodes
*4 and 0.  The model is implemented as a nonlinear voltage controlled
*current source.  "POLY(1)" specifies that the source is input as if
*a one dimensional polynomial controlled the source.
*The 4 0 entries following "POLY(1) set the voltage from node 4 to
*node 0 as the controlling voltage.(This is the voltage between
*the terminals across which the fuse is attached.)
*The next number, -1234, is a flag telling the current source routine
*that this is the fuse model.  The last nine numbers are the fuse
```



```
*parameters:
*In this example - 20MIL means .02in length
*                  20MIL means .02in width
*                  0.175MIL means .000175in thickness
*                  0.9U means 90 microohm-cm for parameter A
*                  1.4U means 140 microohm-cm for parameter B
*                  8.9E16 means .89x10^9A^2-s/cm^4 for G0
*                  1.8E16 means .18 "              for S0
*                  20.E10 is the value of K0E
*                  0.19 is the value of POW
*                  [G0 and S0 are dynamically scaled by
*                   multiplying by a scale factor SCF.
*                   SCF = (max.cur/(time of max.current)/K0E)^POW]
GFUSE 4 0 POLY(1) 4 0 -1234 20MIL 20MIL 0.175MIL 0.9U 1.4U
+ 8.9E16 1.8E16 20.E10 0.19
*(The + in col. 1 in the line above makes it a continuation of
*the line before it.)
*
*The dynamic scaling for G0 and S0 in the fuse model above, can be
*replaced by a fixed scale factor.  In that case replace the
*value of K0E in the GFUSE line by the value to be used for SCF and
*do not include a value for POW.  The FUSE1 subroutine determines
*whether or not to use dynamic scaling by the number of parameters
*entered on the GFUSE command line.
*
*The following line is present because SPICE makes a
*dc path to ground required for nodes 1,2,3,and 4:
RDCP2 1 0 10MEG
*
*The following line is required at the end of SPICE input:
.END
```

11 APPENDIX B: NONLINEAR DEPENDENT SOURCES

SPICE allows circuits to contain dependent sources characterized by any of the four equations

```
i=f(v)        v=f(v)        i=f(i)        v=f(i)
```

where the functions must be polynomials, and the arguments may be multidimensional. The polynomial functions are specified by a set of coefficients p0, p1, ..., pn. Both the number of dimensions and the number of coefficients are arbitrary. The meaning of the coefficients depends upon the dimension of the polynomial, as shown in the following examples:

Suppose that the function is one-dimensional (that is, a function of one argument). Then the function value fv is determined by the following expression in fa (the function argument):

```
fv = p0 + (p1*fa) + (p2*fa**2) + (p3*fa**3) + (p4*fa**4)
     + (p5*fa**5) + ...
```

Suppose now that the function is two-dimensional, with arguments fa and fb. Then the function value fv is determined by the following expression:

```
fv = p0 + (p1*fa) + (p2*fb) + (p3*fa**2) + (p4*fa*fb) + (p5*fb**2)
     + (p6*fa**3) + (p7*fa**2*fb) + (p8*fa*fb**2) + (p9*fb**3) + ...
```

Consider now the case of a three-dimensional polynomial function with arguments fa, fb, and fc. Then the function value fv is determined by the following expression:

```
fv = p0 + (p1*fa) + (p2*fb) + (p3*fc) + (p4*fa**2) + (p5*fa*fb)
     + (p6*fa*fc) + (p7*fb**2) + (p8*fb*fc) + (p9*fc**2) + (p10*fa**3)
     + (p11*fa**2*fb) + (p12*fa**2*fc) + (p13*fa*fb**2)
     + (p14*fa*fb*fc)
     + (p15*fa*fc**2) + (p16*fb**3) + (p17*fb**2*fc) + (p18*fb*fc**2)
     + (p19*fc**3) + (p20*fa**4) + ...
```

Note: if the polynomial is one-dimensional and exactly one coefficient is specified, then SPICE assumes it to be p1 (and p0 = 0.0), in order to facilitate the input of linear controlled sources.

For all four of the dependent sources described below, the initial condition parameter is optional. If not specified, SPICE assumes zero for its value. The initial condition for dependent sources is an initial 'guess' for the value of the controlling variable. The program uses this initial condition to obtain the dc operating point of the circuit. After convergence has been obtained, the program continues iterating to obtain the exact value for the controlling variable. Hence, to reduce the computational effort for the dc operating point (or if the polynomial specifies a strong nonlinearity), a value fairly close to the actual controlling variable should be specified for the initial condition.

11.1 Voltage-Controlled Current Sources

General Form:

```
GXXXXXXX N+ N- <POLY(ND)> NC1+ NC1- ... P0 <P1 ...> <IC=...>
```

Examples:

```
G1 1 0 5 3 0 0.1M
GR 17 3 17 3 0 1M 1.5M IC=2V
GMLT 23 17 POLY(2) 3 5 1 2 0 1M 17M 3.5U IC=2.5, 1.3
```

N+ and N− are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). [Note: If 'POLY' is present, and the source is one-dimensional, the '1' must be explicitly included between parentheses as shown. The word 'POLY' alone cannot be used, in contrast to the case of non-linear inductors and capacitors.] If specified, ND is the number of dimensions, which must be positive. NC1+, NC1−, ...are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling voltage(s). The second example above describes a current source with value

```
I = 1E-3*V(17,3) + 1.5E-3*V(17,3)**2
```

Note that since the source nodes are the same as the controlling nodes, this source actually models a nonlinear resistor.

11.2 Voltage-Controlled Voltage Sources

General Form:

```
EXXXXXXX N+ N- <POLY(ND)> NC1+ NC1- ... P0 <P1 ...> <IC=...>
```

Examples:

```
E1 3 4 21 17 10.5 2.1 1.75
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.35
```

N+ and N− are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. NC1+, NC1−, ... are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling voltage(s). The second example above describes a voltage source with value

```
V = V(13,0) + V(15,0) + V(17,0)
```

(in other words, an ideal voltage summer).

11.3 Current-Controlled Current Sources

General Form:

```
FIXXXXXX N+ N- <POLY(ND)> VN1 <VN2 ...> P0 <P1 ...> <IC=...>
```

Examples:

```
F1 12 10 VCC 1MA 1.3M
FIFER 13 20 VSENS 0 1
```

N+ and N− are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in Amps). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling current(s). The first example above describes a current source with value

```
I = 1E-3 + 1.3E-3*I(VCC)
```

11.4 Current-Controlled Voltage Sources

General Form:

```
HXXXXXXX N+ N- <POLY(ND)> VN1 <VN2 ...> P0 <P1 ...> <IC=...>
```

Examples:

```
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5 1.3
HR 4 17 VX 0 0 1
```

N+ and N− are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in Amps). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling current(s). The first example above describes a voltage source with value

```
V = I(VIN1)*I(VIN2)
```

Appendix G

Output files produced by SPICE when running test files for fuse and saturating inductance elements

The following listings give the output files TFUSE10.OUT and LTANT1.OUT that were produced from using the correspondinly named input files with SPICE2G6-SB2 on a PC.

G.1 TFUSE10.OUT

(The output files contain FORTRAN carriage control characters in the first column. These were removed before printing the following pages by means of the utility program FORLPT.)

**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83****** 9:44: 1****

TFUSE10 - TEST CIRCUIT FOR SPICE2SB FUSE MODEL

****   INPUT LISTING   TEMPERATURE =   27.000 DEG C

```
*********************************************************************



*Same values are used below as in 1st example in FIRESET paper by Ron Lee.
*
*Transient analysis is the only one valid when the fuse model
*is used in SPICE2SB.  The following line calls for a transient
*analysis, saving output every 5 nanoseconds and running for a
*total of one microsecond.  The starting time is zero and the
*minimum time step for numerical integration is 1 nanosecond.
*Initial conditions specified on the element lines are to be used.
*(Note: If the SPICE error message,
* "ERROR*:  INTERNAL TIMESTEP TOO SMALL IN TRANSIENT ANALYSIS",
* results from an analysis, the first thing to try is to reduce
* the minimum time step.)
.TRAN 5NS 1US 0 1NS UIC
*
*The following line calls for a printout of the current in the
*fuse, I(VDC), and the voltage across it, V(4).
.PRINT TRAN I(VDC) V(4)
*
*The following line calls for a plot versus time of these same variables:
.PLOT TRAN I(VDC) V(4)
*
*The following line specifies that a capacitor of value 0.5microfarads
*with initial voltage 5killovolts is located between nodes 1 and 0
*(node 0 is ground), with node 1 positive:
CSOURCE 1 0 0.5UF IC=5KV
*
*The following line specifies a resistor of value 75milliohms from
*node 1 to node 2:
RS 1 2 0.075
*
*(Note, the following line DOES NOT GIVE THE CORRECT VALUE:
*RS 1 2 75MILLIOHM
*as SPICE takes MIL as a suffix for multiplication by 25.4E-6.
*Be VERY CAREFUL when using unit suffixes with SPICE!)
*
*The following line specifies an inductor of value 90nanohenries from
*node 2 to node 3:
LS 2 3 90NANOHENRY
*
*The following line specifies an ammeter for positive current flowing
*from node 3 to node 4 through it.  This current is I(VDC):
VDC 3 4 DC 0
*
*The next line specifies the fuse model is connected between nodes
*4 and 0.  The model is implemented as a nonlinear voltage controlled
*current source.  "POLY(1)" specifies that the source is input as if
*a one dimensional polynomial controlled the source.
*The 4 0 entries following "POLY(1) set the voltage from node 4 to
*node 0 as the controlling voltage.(This is the voltage between
*the terminals across which the fuse is attached.)
*The next number, -1234, is a flag telling the current source routine
*that this is the fuse model.  The last nine numbers are the fuse
*parameters:
*In this example - 20MIL means .02in length
*                  20MIL means .02in width
*                  0.175MIL means .000175in thickness
*                  0.9U means 90 microohm-cm for parameter A
*                  1.4U means 140 microohm-cm for parameter B
*                  8.9E16 means .89x10^9A^2-s/cm^4 for G0
*                  1.8E16 means .18 "              for S0
*                  20.E10 is the value of KOE
*                  0.19 is the value of POW
*                  [G0 and S0 are dynamically scaled by
*                   multiplying by a scale factor SCF.
*                   SCF = (max.cur/(time of max.current)/KOE)^POW]
GFUSE 4 0 POLY(1) 4 0 -1234 20MIL 20MIL 0.175MIL 0.9U 1.4U
+ 8.9E16 1.8E16 20.E10 0.19
*(The + in col. 1 in the line above makes it a continuation of
*the line before it.)
*
*The dynamic scaling for G0 and S0 in the fuse model above, can be
*replaced by a fixed scale factor.  In that case replace the
*value of KOE in the GFUSE line by the value to be used for SCF and
*do not include a value for POW.  The FUSE1 subroutine determines
*whether or not to use dynamic scaling by the number of parameters
*entered on the GFUSE command line.
*
*The following two lines have the fixed scale factor implemented for
*this example.
*GFUSE 4 0 POLY(1) 4 0 -1234 20MIL 20MIL 0.175MIL 0.9U 1.4U
*+ 8.9E16 1.8E16 0.78397562
*
*The following line is present because SPICE makes a
*dc path to ground required for nodes 1,2,3,and 4:
RDCP2 1 0 10MEG
*
*The following line sets the width for the printed output to 80 columns:
.WIDTH IN 80 OUT 80
*The following line is required at the end of SPICE input:
.END

TIME=0 reset of parameters for Fuse from   4 to  0

FUSE from  4 to  0 blows at T=   7.2280E-08, I=   3.7350E+03, G=   3.5982E-01

g0=   3.5090E-01 cmax=   3.7351E+03 tmax=   7.2280E-08
```

**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83****** 9:44: 1****

TFUSE10 - TEST CIRCUIT FOR SPICE2SB FUSE MODEL

**** TRANSIENT ANALYSIS TEMPERATURE = 27.000 DEG C

****

TIME	I(VDC)	V(4)
0.000E+00	4.632E-06	3.520E-17
5.000E-09	2.772E+02	1.206E-06
1.000E-08	5.530E+02	2.599E-04
1.500E-08	8.274E+02	5.307E-03
2.000E-08	1.100E+03	4.334E-02
2.500E-08	1.371E+03	2.167E-01
3.000E-08	1.640E+03	7.979E-01
3.500E-08	1.908E+03	2.380E+00
4.000E-08	2.172E+03	6.077E+00
4.500E-08	2.434E+03	1.375E+01
5.000E-08	2.694E+03	2.817E+01
5.500E-08	2.949E+03	5.332E+01
6.000E-08	3.200E+03	1.026E+02
6.500E-08	3.442E+03	3.104E+02
7.000E-08	3.655E+03	1.107E+03
7.500E-08	3.827E+03	1.263E+03
8.000E-08	4.024E+03	5.217E+02
8.500E-08	4.237E+03	5.128E+02
9.000E-08	4.443E+03	6.172E+02
9.500E-08	4.640E+03	7.175E+02
1.000E-07	4.828E+03	8.092E+02
1.050E-07	5.008E+03	8.905E+02
1.100E-07	5.180E+03	9.611E+02
1.150E-07	5.345E+03	1.022E+03
1.200E-07	5.503E+03	1.074E+03
1.250E-07	5.655E+03	1.119E+03
1.300E-07	5.801E+03	1.158E+03
1.350E-07	5.940E+03	1.193E+03
1.400E-07	6.074E+03	1.224E+03
1.450E-07	6.203E+03	1.252E+03
1.500E-07	6.326E+03	1.279E+03
1.550E-07	6.444E+03	1.304E+03
1.600E-07	6.556E+03	1.327E+03
1.650E-07	6.663E+03	1.349E+03
1.700E-07	6.764E+03	1.369E+03
1.750E-07	6.860E+03	1.389E+03
1.800E-07	6.951E+03	1.407E+03
1.850E-07	7.037E+03	1.425E+03
1.900E-07	7.118E+03	1.441E+03
1.950E-07	7.193E+03	1.456E+03
2.000E-07	7.264E+03	1.471E+03
2.050E-07	7.329E+03	1.484E+03
2.100E-07	7.389E+03	1.496E+03
2.150E-07	7.444E+03	1.507E+03
2.200E-07	7.494E+03	1.517E+03
2.250E-07	7.540E+03	1.527E+03
2.300E-07	7.580E+03	1.535E+03
2.350E-07	7.616E+03	1.542E+03
2.400E-07	7.647E+03	1.548E+03
2.450E-07	7.673E+03	1.554E+03
2.500E-07	7.695E+03	1.558E+03
2.550E-07	7.712E+03	1.561E+03
2.600E-07	7.724E+03	1.564E+03
2.650E-07	7.732E+03	1.566E+03
2.700E-07	7.736E+03	1.566E+03
2.750E-07	7.736E+03	1.566E+03
2.800E-07	7.731E+03	1.565E+03
2.850E-07	7.722E+03	1.563E+03
2.900E-07	7.708E+03	1.561E+03
2.950E-07	7.691E+03	1.557E+03
3.000E-07	7.670E+03	1.553E+03
3.050E-07	7.645E+03	1.548E+03
3.100E-07	7.616E+03	1.542E+03
3.150E-07	7.583E+03	1.535E+03
3.200E-07	7.547E+03	1.528E+03
3.250E-07	7.507E+03	1.520E+03
3.300E-07	7.463E+03	1.511E+03
3.350E-07	7.417E+03	1.502E+03
3.400E-07	7.366E+03	1.491E+03
3.450E-07	7.313E+03	1.481E+03
3.500E-07	7.256E+03	1.469E+03
3.550E-07	7.196E+03	1.457E+03
3.600E-07	7.133E+03	1.444E+03
3.650E-07	7.067E+03	1.431E+03
3.700E-07	6.998E+03	1.417E+03
3.750E-07	6.926E+03	1.402E+03
3.800E-07	6.852E+03	1.387E+03
3.850E-07	6.775E+03	1.372E+03
3.900E-07	6.695E+03	1.356E+03
3.950E-07	6.613E+03	1.339E+03
4.000E-07	6.529E+03	1.322E+03
4.050E-07	6.442E+03	1.304E+03
4.100E-07	6.353E+03	1.286E+03
4.150E-07	6.262E+03	1.268E+03
4.200E-07	6.169E+03	1.249E+03
4.250E-07	6.074E+03	1.230E+03
4.300E-07	5.977E+03	1.210E+03
4.350E-07	5.878E+03	1.190E+03
4.400E-07	5.777E+03	1.170E+03
4.450E-07	5.675E+03	1.149E+03
4.500E-07	5.571E+03	1.128E+03
4.550E-07	5.466E+03	1.107E+03
4.600E-07	5.359E+03	1.085E+03
4.650E-07	5.251E+03	1.063E+03
4.700E-07	5.142E+03	1.041E+03
4.750E-07	5.032E+03	1.019E+03
4.800E-07	4.920E+03	9.962E+02
4.850E-07	4.808E+03	9.734E+02
4.900E-07	4.694E+03	9.505E+02
4.950E-07	4.580E+03	9.273E+02
5.000E-07	4.465E+03	9.040E+02
5.050E-07	4.349E+03	8.806E+02

5.100E-07	4.233E+03	8.570E+02
5.150E-07	4.116E+03	8.333E+02
5.200E-07	3.998E+03	8.096E+02
5.250E-07	3.881E+03	7.857E+02
5.300E-07	3.762E+03	7.618E+02
5.350E-07	3.644E+03	7.378E+02
5.400E-07	3.525E+03	7.138E+02
5.450E-07	3.407E+03	6.897E+02
5.500E-07	3.288E+03	6.657E+02
5.550E-07	3.169E+03	6.416E+02
5.600E-07	3.050E+03	6.176E+02
5.650E-07	2.932E+03	5.936E+02
5.700E-07	2.813E+03	5.696E+02
5.750E-07	2.695E+03	5.457E+02
5.800E-07	2.577E+03	5.218E+02
5.850E-07	2.460E+03	4.980E+02
5.900E-07	2.343E+03	4.743E+02
5.950E-07	2.226E+03	4.508E+02
6.000E-07	2.110E+03	4.273E+02
6.050E-07	1.995E+03	4.039E+02
6.100E-07	1.880E+03	3.807E+02
6.150E-07	1.766E+03	3.576E+02
6.200E-07	1.653E+03	3.347E+02
6.250E-07	1.541E+03	3.120E+02
6.300E-07	1.429E+03	2.894E+02
6.350E-07	1.319E+03	2.670E+02
6.400E-07	1.209E+03	2.448E+02
6.450E-07	1.100E+03	2.228E+02
6.500E-07	9.929E+02	2.010E+02
6.550E-07	8.864E+02	1.795E+02
6.600E-07	7.811E+02	1.582E+02
6.650E-07	6.770E+02	1.371E+02
6.700E-07	5.741E+02	1.162E+02
6.750E-07	4.724E+02	9.566E+01
6.800E-07	3.721E+02	7.533E+01
6.850E-07	2.730E+02	5.528E+01
6.900E-07	1.754E+02	3.551E+01
6.950E-07	7.910E+01	1.602E+01
7.000E-07	-1.574E+01	-3.188E+00
7.050E-07	-1.091E+02	-2.209E+01
7.100E-07	-2.010E+02	-4.070E+01
7.150E-07	-2.914E+02	-5.900E+01
7.200E-07	-3.802E+02	-7.699E+01
7.250E-07	-4.675E+02	-9.465E+01
7.300E-07	-5.532E+02	-1.120E+02
7.350E-07	-6.372E+02	-1.290E+02
7.400E-07	-7.196E+02	-1.457E+02
7.450E-07	-8.004E+02	-1.621E+02
7.500E-07	-8.795E+02	-1.781E+02
7.550E-07	-9.569E+02	-1.937E+02
7.600E-07	-1.033E+03	-2.091E+02
7.650E-07	-1.107E+03	-2.240E+02
7.700E-07	-1.179E+03	-2.387E+02
7.750E-07	-1.249E+03	-2.529E+02
7.800E-07	-1.318E+03	-2.668E+02
7.850E-07	-1.385E+03	-2.804E+02
7.900E-07	-1.450E+03	-2.936E+02
7.950E-07	-1.513E+03	-3.064E+02
8.000E-07	-1.575E+03	-3.188E+02
8.050E-07	-1.634E+03	-3.309E+02
8.100E-07	-1.692E+03	-3.427E+02
8.150E-07	-1.748E+03	-3.540E+02
8.200E-07	-1.803E+03	-3.650E+02
8.250E-07	-1.855E+03	-3.756E+02
8.300E-07	-1.906E+03	-3.859E+02
8.350E-07	-1.955E+03	-3.958E+02
8.400E-07	-2.002E+03	-4.053E+02
8.450E-07	-2.047E+03	-4.144E+02
8.500E-07	-2.090E+03	-4.232E+02
8.550E-07	-2.132E+03	-4.316E+02
8.600E-07	-2.171E+03	-4.396E+02
8.650E-07	-2.209E+03	-4.473E+02
8.700E-07	-2.245E+03	-4.546E+02
8.750E-07	-2.280E+03	-4.615E+02
8.800E-07	-2.312E+03	-4.681E+02
8.850E-07	-2.343E+03	-4.744E+02
8.900E-07	-2.372E+03	-4.802E+02
8.950E-07	-2.399E+03	-4.857E+02
9.000E-07	-2.425E+03	-4.909E+02
9.050E-07	-2.448E+03	-4.957E+02
9.100E-07	-2.470E+03	-5.002E+02
9.150E-07	-2.491E+03	-5.043E+02
9.200E-07	-2.509E+03	-5.081E+02
9.250E-07	-2.526E+03	-5.115E+02
9.300E-07	-2.542E+03	-5.146E+02
9.350E-07	-2.556E+03	-5.174E+02
9.400E-07	-2.568E+03	-5.199E+02
9.450E-07	-2.578E+03	-5.220E+02
9.500E-07	-2.587E+03	-5.238E+02
9.550E-07	-2.594E+03	-5.253E+02
9.600E-07	-2.600E+03	-5.265E+02
9.650E-07	-2.605E+03	-5.273E+02
9.700E-07	-2.607E+03	-5.279E+02
9.750E-07	-2.609E+03	-5.282E+02
9.800E-07	-2.608E+03	-5.281E+02
9.850E-07	-2.607E+03	-5.278E+02
9.900E-07	-2.604E+03	-5.272E+02
9.950E-07	-2.599E+03	-5.263E+02
1.000E-06	-2.594E+03	-5.252E+02

**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83****** 9:44: 1****

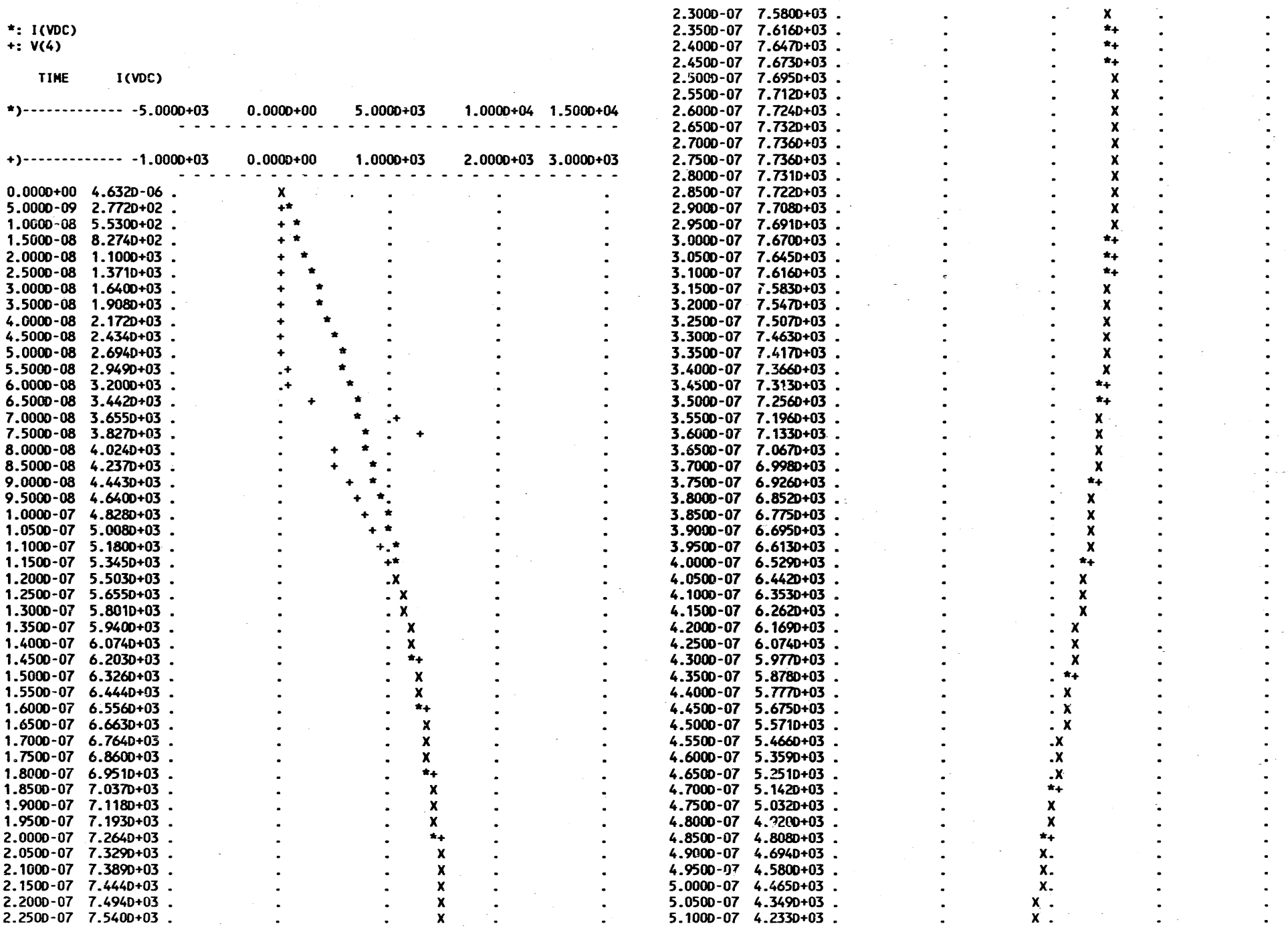
TFUSE10 - TEST CIRCUIT FOR SPICE2SB FUSE MODEL

**** TRANSIENT ANALYSIS TEMPERATURE = 27.000 DEG C

****

LEGEND:

*: I(VDC)
+: V(4)


TIME I(VDC)
*)---------- -5.000D+03 0.000D+00 5.000D+03 1.000D+04 1.500D+04
+)---------- -1.000D+03 0.000D+00 1.000D+03 2.000D+03 3.000D+03
0.000D+00 4.632D-06
5.000D-09 2.772D+02
1.000D-08 5.530D+02
1.500D-08 8.274D+02
2.000D-08 1.100D+03
2.500D-08 1.371D+03
3.000D-08 1.640D+03
3.500D-08 1.908D+03
4.000D-08 2.172D+03
4.500D-08 2.434D+03
5.000D-08 2.694D+03
5.500D-08 2.949D+03
6.000D-08 3.200D+03
6.500D-08 3.442D+03
7.000D-08 3.655D+03
7.500D-08 3.827D+03
8.000D-08 4.024D+03
8.500D-08 4.237D+03
9.000D-08 4.443D+03
9.500D-08 4.640D+03
1.000D-07 4.828D+03
1.050D-07 5.008D+03
1.100D-07 5.180D+03
1.150D-07 5.345D+03
1.200D-07 5.503D+03
1.250D-07 5.655D+03
1.300D-07 5.801D+03
1.350D-07 5.940D+03
1.400D-07 6.074D+03
1.450D-07 6.203D+03
1.500D-07 6.326D+03
1.550D-07 6.444D+03
1.600D-07 6.556D+03
1.650D-07 6.663D+03
1.700D-07 6.764D+03
1.750D-07 6.860D+03
1.800D-07 6.951D+03
1.850D-07 7.037D+03
1.900D-07 7.118D+03
1.950D-07 7.193D+03
2.000D-07 7.264D+03
2.050D-07 7.329D+03
2.100D-07 7.389D+03
2.150D-07 7.444D+03
2.200D-07 7.494D+03
2.250D-07 7.540D+03
2.300D-07 7.580D+03
2.350D-07 7.616D+03
2.400D-07 7.647D+03
2.450D-07 7.673D+03
2.500D-07 7.695D+03
2.550D-07 7.712D+03
2.600D-07 7.724D+03
2.650D-07 7.732D+03
2.700D-07 7.736D+03
2.750D-07 7.736D+03
2.800D-07 7.731D+03
2.850D-07 7.722D+03
2.900D-07 7.708D+03
2.950D-07 7.691D+03
3.000D-07 7.670D+03
3.050D-07 7.645D+03
3.100D-07 7.616D+03
3.150D-07 7.583D+03
3.200D-07 7.547D+03
3.250D-07 7.507D+03
3.300D-07 7.463D+03
3.350D-07 7.417D+03
3.400D-07 7.366D+03
3.450D-07 7.313D+03
3.500D-07 7.256D+03
3.550D-07 7.196D+03
3.600D-07 7.133D+03
3.650D-07 7.067D+03
3.700D-07 6.998D+03
3.750D-07 6.926D+03
3.800D-07 6.852D+03
3.850D-07 6.775D+03
3.900D-07 6.695D+03
3.950D-07 6.613D+03
4.000D-07 6.529D+03
4.050D-07 6.442D+03
4.100D-07 6.353D+03
4.150D-07 6.262D+03
4.200D-07 6.169D+03
4.250D-07 6.074D+03
4.300D-07 5.977D+03
4.350D-07 5.878D+03
4.400D-07 5.777D+03
4.450D-07 5.675D+03
4.500D-07 5.571D+03
4.550D-07 5.466D+03
4.600D-07 5.359D+03
4.650D-07 5.251D+03
4.700D-07 5.142D+03
4.750D-07 5.032D+03
4.800D-07 4.920D+03
4.850D-07 4.808D+03
4.900D-07 4.694D+03
4.950D-07 4.580D+03
5.000D-07 4.465D+03
5.050D-07 4.349D+03
5.100D-07 4.233D+03

```
5.1500-07  4.1160+03 .                .                 X .                .                .
5.2000-07  3.9980+03 .                .                X  .                .                .
5.2500-07  3.8810+03 .                .                X  .                .                .
5.3000-07  3.7620+03 .                .                X  .                .                .
5.3500-07  3.6440+03 .                .               X   .                .                .
5.4000-07  3.5250+03 .                .               X   .                .                .
5.4500-07  3.4070+03 .                .               X   .                .                .
5.5000-07  3.2880+03 .                .              X    .                .                .
5.5500-07  3.1690+03 .                .              X    .                .                .
5.6000-07  3.0500+03 .                .              X    .                .                .
5.6500-07  2.9320+03 .                .             X     .                .                .
5.7000-07  2.8130+03 .                .             X     .                .                .
5.7500-07  2.6950+03 .                .             X     .                .                .
5.8000-07  2.5770+03 .                .            X      .                .                .
5.8500-07  2.4600+03 .                .            X      .                .                .
5.9000-07  2.3430+03 .                .            X      .                .                .
5.9500-07  2.2260+03 .                .           X       .                .                .
6.0000-07  2.1100+03 .                .           X       .                .                .
6.0500-07  1.9950+03 .                .           X       .                .                .
6.1000-07  1.8800+03 .                .          X        .                .                .
6.1500-07  1.7660+03 .                .          X        .                .                .
6.2000-07  1.6530+03 .                .          X        .                .                .
6.2500-07  1.5410+03 .                .         X         .                .                .
6.3000-07  1.4290+03 .                .         X         .                .                .
6.3500-07  1.3190+03 .                .         X         .                .                .
6.4000-07  1.2090+03 .                .        X          .                .                .
6.4500-07  1.1000+03 .                .        X          .                .                .
6.5000-07  9.9290+02 .                .        X          .                .                .
6.5500-07  8.8640+02 .                .       *+          .                .                .
6.6000-07  7.8110+02 .                .       X           .                .                .
6.6500-07  6.7700+02 .                .       X           .                .                .
6.7000-07  5.7410+02 .                .       X           .                .                .
6.7500-07  4.7240+02 .                .      X            .                .                .
6.8000-07  3.7210+02 .                .      X            .                .                .
6.8500-07  2.7300+02 .                .      X            .                .                .
6.9000-07  1.7540+02 .                .     X             .                .                .
6.9500-07  7.9100+01 .                .     X             .                .                .
7.0000-07 -1.5740+01 .                .     X             .                .                .
7.0500-07 -1.0910+02 .                .     X             .                .                .
7.1000-07 -2.0100+02 .                .    X.             .                .                .
7.1500-07 -2.9140+02 .                .    X.             .                .                .
7.2000-07 -3.8020+02 .                .    X.             .                .                .
7.2500-07 -4.6750+02 .                .    X.             .                .                .
7.3000-07 -5.5320+02 .                .   X .             .                .                .
7.3500-07 -6.3720+02 .                .   X .             .                .                .
7.4000-07 -7.1960+02 .                .   X .             .                .                .
7.4500-07 -8.0040+02 .                .   X .             .                .                .
7.5000-07 -8.7950+02 .                .   X .             .                .                .
7.5500-07 -9.5690+02 .                .  X  .             .                .                .
7.6000-07 -1.0330+03 .                .  X  .             .                .                .
7.6500-07 -1.1070+03 .                .  X  .             .                .                .
7.7000-07 -1.1790+03 .                .  X  .             .                .                .
7.7500-07 -1.2490+03 .                .  +* .             .                .                .
7.8000-07 -1.3180+03 .                .  X  .             .                .                .
7.8500-07 -1.3850+03 .                .  X  .             .                .                .
7.9000-07 -1.4500+03 .                .  X  .             .                .                .
7.9500-07 -1.5130+03 .                .  X  .             .                .                .
8.0000-07 -1.5750+03 .                .   X .             .                .                .
8.0500-07 -1.6340+03 .                .  X  .             .                .                .
8.1000-07 -1.6920+03 .                .  X  .             .                .                .
8.1500-07 -1.7480+03 .                .  X  .             .                .                .
8.2000-07 -1.8030+03 .                .  X  .             .                .                .
8.2500-07 -1.8550+03 .                .  X  .             .                .                .
8.3000-07 -1.9060+03 .                .  X  .             .                .                .
8.3500-07 -1.9550+03 .                .  +* .             .                .                .
8.4000-07 -2.0020+03 .                .  X  .             .                .                .
8.4500-07 -2.0470+03 .                .  X  .             .                .                .
8.5000-07 -2.0900+03 .                .  X  .             .                .                .
8.5500-07 -2.1320+03 .                .  X  .             .                .                .
8.6000-07 -2.1710+03 .                .  X  .             .                .                .
8.6500-07 -2.2090+03 .                .  X  .             .                .                .
8.7000-07 -2.2450+03 .                .  X  .             .                .                .
8.7500-07 -2.2800+03 .                .  X  .             .                .                .
8.8000-07 -2.3120+03 .                .  +* .             .                .                .
8.8500-07 -2.3430+03 .                .  X  .             .                .                .
8.9000-07 -2.3720+03 .                .  X  .             .                .                .
8.9500-07 -2.3990+03 .                .  X  .             .                .                .
9.0000-07 -2.4250+03 .                .  X  .             .                .                .
9.0500-07 -2.4480+03 .                .  X  .             .                .                .
9.1000-07 -2.4700+03 .                .  X  .             .                .                .
9.1500-07 -2.4910+03 .                .  X  .             .                .                .
9.2000-07 -2.5090+03 .                .  X  .             .                .                .
9.2500-07 -2.5260+03 .                .  X  .             .                .                .
9.3000-07 -2.5420+03 .                .  X  .             .                .                .
9.3500-07 -2.5560+03 .                .  X  .             .                .                .
9.4000-07 -2.5680+03 .                .  X  .             .                .                .
9.4500-07 -2.5780+03 .                .  X  .             .                .                .
9.5000-07 -2.5870+03 .                .  X  .             .                .                .
9.5500-07 -2.5940+03 .                .  X  .             .                .                .
9.6000-07 -2.6000+03 .                .  X  .             .                .                .
9.6500-07 -2.6050+03 .                .  X  .             .                .                .
9.7000-07 -2.6070+03 .                .  X  .             .                .                .
9.7500-07 -2.6090+03 .                .  X  .             .                .                .
9.8000-07 -2.6080+03 .                .  X  .             .                .                .
9.8500-07 -2.6070+03 .                .  X  .             .                .                .
9.9000-07 -2.6040+03 .                .  X  .             .                .                .
9.9500-07 -2.5990+03 .                .  X  .             .                .                .
1.0000-06 -2.5940+03 .                .  X  .             .                .                .
```

JOB CONCLUDED

TOTAL JOB TIME 37.00

G.2 LTANT1.OUT

(The following pages were printed without conversion of the FORTRAN carriage control characters.)

```
1**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83******10:10:39****

OLTANT1 - TEST OF SATURATING INDUCTANCE MODEL IN SPICE2SB

0****     INPUT LISTING                    TEMPERATURE =   27.000 DEG C

0*****************************************************************************



 *Saturating inductance model is valid only for transient analysis.
 *
 *The following line specifies a transient analysis, saving results
 *every 0.01S and continuing from time 0 to time 1S.
 .TRAN 0.01 1
 *
 *The following line specifies an independent current source, connected
 *between node 1 and ground (node 0 is always ground), with the current
 *going through the source from node 1 to node 0 and out node 0.
 *The current is a sine wave of amplitude -2amps, frequency 1Hz, zero
 *phase shift, no offset and no damping:
 I1 1 0 SIN(0 -2 1 0 0)
 *
 *The following line specifies the saturating inductor is connected
 *between nodes 3 and 0.  The word "POLY" signals a nonlinear inductance,
 *and the first number, -1234, is the flag to indicate the hyperbolic
 *tangent saturation model is the one to use.  The numbers 1, .001, 1,1
 *mean, respectively, initial inductance(in Henries), inductance after
 *saturation, saturation current (in amps), and power (and root) used
 *for the hyperbolic tangent and argument in the model.
 L1 3 0 POLY -1234 1 0.001 1 1
 *
 *The next line specifies an ammeter measuring current flowing from node 1
 *to node 3 through it.  The current is referenced later as I(VCUR):
 VCUR 1 3 DC 0
 *
 *The next line specifies a large resistance connected from node 3 to node
 *0.  It should not be needed, but a bug in SPICE2G6 makes it necessary to
 *avoid an error of "time step too small".  Note that SPICE uses the
 *abbreviation M for milli, but uses MEG for mega.  The "EG" is required
 *or the value will be off six orders of magnetude!
 R1 3 0 10MEG
 *
 *The following lines place a voltage controlled current source, having
 *controlling voltage equal to the voltage across the inductor, in series
 *with a one farad capacitor.  (The resistor in parallel is required since
 *SPICE requires a DC path to ground from each node.)  The voltage across
 *the capacitor is then numerically equal to the flux through the inductor.
 *(Note that one cannot write 1F for the capacitance value, as SPICE
 *interprets the F as a unit of 10^-15.)
 G2 2 0 1
 G2 2 0 1 0 -1
 R2 2 0 10MEG
 *
 *The following line calls for a printout of the current through the
 *inductor and the flux.  A plot of V(2) versus I(VCUR) will be a scaled
 *one of the model B-H curve.
 .PRINT TRAN I(VCUR) V(2)
 *
 *The following line sets the width for the printed output to 80 columns:
 .WIDTH IN 80 OUT 80
 *All SPICE input files must end with the following line:
 .END
1**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83******10:10:39****

OLTANT1 - TEST OF SATURATING INDUCTANCE MODEL IN SPICE2SB

0****     INITIAL TRANSIENT SOLUTION       TEMPERATURE =   27.000 DEG C

0*****************************************************************************

  NODE   VOLTAGE     NODE   VOLTAGE     NODE   VOLTAGE

(  1)     .0000    (  2)     .0000    (  3)     .0000



    VOLTAGE SOURCE CURRENTS

    NAME        CURRENT

    VCUR       0.000D+00

    TOTAL POWER DISSIPATION   0.00D+00  WATTS
1**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83******10:10:39****

OLTANT1 - TEST OF SATURATING INDUCTANCE MODEL IN SPICE2SB

0****     OPERATING POINT INFORMATION      TEMPERATURE =   27.000 DEG C

0*****************************************************************************


0
0**** VOLTAGE-CONTROLLED CURRENT SOURCES


0            G2
 I-SOURCE  0.00E+00
1**** 7/ 2/90 ******SPICE2G6-KHC-MS2 18MAY89-3/15/83******10:10:39****

OLTANT1 - TEST OF SATURATING INDUCTANCE MODEL IN SPICE2SB

0****     TRANSIENT ANALYSIS               TEMPERATURE =   27.000 DEG C

0*****************************************************************************
```

TIME	I(VCUR)	V(2)
X		
0.000E+00	0.000E+00	0.000E+00
1.000E-02	1.256E-01	1.250E-01
2.000E-02	2.505E-01	2.447E-01
3.000E-02	3.741E-01	3.534E-01
4.000E-02	4.969E-01	4.571E-01
5.000E-02	6.169E-01	5.445E-01
6.000E-02	7.355E-01	6.238E-01
7.000E-02	8.500E-01	6.875E-01
8.000E-02	9.625E-01	7.434E-01
9.000E-02	1.070E+00	7.870E-01
1.000E-01	1.174E+00	8.245E-01
1.100E-01	1.273E+00	8.533E-01
1.200E-01	1.368E+00	8.778E-01
1.300E-01	1.455E+00	8.965E-01
1.400E-01	1.539E+00	9.122E-01
1.500E-01	1.615E+00	9.242E-01
1.600E-01	1.687E+00	9.344E-01
1.700E-01	1.750E+00	9.420E-01
1.800E-01	1.808E+00	9.485E-01
1.900E-01	1.856E+00	9.533E-01
2.000E-01	1.900E+00	9.573E-01
2.100E-01	1.934E+00	9.602E-01
2.200E-01	1.962E+00	9.625E-01
2.300E-01	1.981E+00	9.639E-01
2.400E-01	1.994E+00	9.648E-01
2.500E-01	1.996E+00	9.650E-01
2.600E-01	1.994E+00	9.648E-01
2.700E-01	1.981E+00	9.639E-01
2.800E-01	1.962E+00	9.625E-01
2.900E-01	1.934E+00	9.602E-01
3.000E-01	1.900E+00	9.574E-01
3.100E-01	1.856E+00	9.533E-01
3.200E-01	1.808E+00	9.486E-01
3.300E-01	1.750E+00	9.420E-01
3.400E-01	1.687E+00	9.345E-01
3.500E-01	1.615E+00	9.241E-01
3.600E-01	1.539E+00	9.124E-01
3.700E-01	1.455E+00	8.963E-01
3.800E-01	1.368E+00	8.780E-01
3.900E-01	1.273E+00	8.531E-01
4.000E-01	1.174E+00	8.249E-01
4.100E-01	1.070E+00	7.868E-01
4.200E-01	9.624E-01	7.438E-01
4.300E-01	8.501E-01	6.872E-01
4.400E-01	7.354E-01	6.240E-01
4.500E-01	6.170E-01	5.445E-01
4.600E-01	4.968E-01	4.569E-01
4.700E-01	3.742E-01	3.539E-01
4.800E-01	2.503E-01	2.432E-01
4.900E-01	1.254E-01	1.236E-01
5.000E-01	-6.110E-05	-3.136E-04
5.100E-01	-1.253E-01	-1.224E-01
5.200E-01	-2.505E-01	-2.435E-01
5.300E-01	-3.741E-01	-3.530E-01
5.400E-01	-4.969E-01	-4.566E-01
5.500E-01	-6.169E-01	-5.440E-01
5.600E-01	-7.355E-01	-6.233E-01
5.700E-01	-8.500E-01	-6.870E-01
5.800E-01	-9.625E-01	-7.430E-01
5.900E-01	-1.070E+00	-7.866E-01
6.000E-01	-1.174E+00	-8.241E-01
6.100E-01	-1.273E+00	-8.529E-01
6.200E-01	-1.368E+00	-8.774E-01
6.300E-01	-1.455E+00	-8.960E-01
6.400E-01	-1.539E+00	-9.118E-01
6.500E-01	-1.615E+00	-9.238E-01
6.600E-01	-1.687E+00	-9.339E-01
6.700E-01	-1.750E+00	-9.416E-01
6.800E-01	-1.808E+00	-9.480E-01
6.900E-01	-1.856E+00	-9.529E-01
7.000E-01	-1.900E+00	-9.569E-01
7.100E-01	-1.934E+00	-9.597E-01
7.200E-01	-1.962E+00	-9.620E-01
7.300E-01	-1.981E+00	-9.634E-01
7.400E-01	-1.994E+00	-9.644E-01
7.500E-01	-1.996E+00	-9.646E-01
7.600E-01	-1.994E+00	-9.644E-01
7.700E-01	-1.981E+00	-9.634E-01
7.800E-01	-1.962E+00	-9.620E-01
7.900E-01	-1.934E+00	-9.597E-01
8.000E-01	-1.900E+00	-9.569E-01
8.100E-01	-1.856E+00	-9.528E-01
8.200E-01	-1.808E+00	-9.481E-01
8.300E-01	-1.750E+00	-9.415E-01
8.400E-01	-1.687E+00	-9.340E-01
8.500E-01	-1.615E+00	-9.237E-01
8.600E-01	-1.539E+00	-9.119E-01
8.700E-01	-1.455E+00	-8.958E-01
8.800E-01	-1.368E+00	-8.776E-01
8.900E-01	-1.273E+00	-8.527E-01
9.000E-01	-1.174E+00	-8.244E-01
9.100E-01	-1.070E+00	-7.863E-01
9.200E-01	-9.624E-01	-7.433E-01
9.300E-01	-8.501E-01	-6.868E-01
9.400E-01	-7.354E-01	-6.235E-01
9.500E-01	-6.170E-01	-5.440E-01
9.600E-01	-4.968E-01	-4.564E-01
9.700E-01	-3.742E-01	-3.534E-01
9.800E-01	-2.503E-01	-2.427E-01
9.900E-01	-1.254E-01	-1.234E-01
1.000E+00	-6.060E-15	2.221E-04
Y		
0		
	JOB CONCLUDED	
0	TOTAL JOB TIME	10.00

Appendix H

Source listings for the SPFEDIT program

The following pages contain the source listings for the SPFEDIT program. This program is an interactive editor for circuit input files for the SPICE program, and allows easy entry of fuse and saturating inductance circuit elements. The first section below is a listing of the "c" source of the program. The second section is a listing of the "help" file used by the program.

H.1 The "c" source

The source for SPFEDIT was compiled using version 1.5 of the Borland TurboC compiler. The source was split into several separate files on the PC. All of these source files are listed on the following pages, with the code for each new file beginning on a new page. The files are listed using the line printer font on a Hewlett Packard LaserJet printer so that the characters of the PC screen are accurately reproduced.

The first file listed is the TurboC "project file" which shows the dependence relationships between the source files, much as is done with a "makefile" under Unix. The second file listed is the ".h" file, which is included by most of the other source files. The remaining files contain the "c" source code. The one additional file needed to use the TurboC integrated environment to build the executable SPFEDIT.EXE is the TurboC configuration file, which sets compiler options and locations for libraries, etc. It is a binary file, and must be tailored for the machine on which it is used. The essential option to give the compiler is "large model."

The files follow in the following order:

```
SPFEDIT  PRJ      289   8-15-89  12:36p
SPEDIT   H       3876   5-02-89   6:14p
SPFEDIT  C       3527   6-25-89   5:35p
MENUCALL C       2185   5-04-89   6:05p
TMALLOC  C       1724   5-22-89  11:43a
GETOPT   C       1983  11-09-88  12:12p
SWCHAR   C        236  11-07-88   1:25p
BRANCH   C      13122   5-22-89  11:21a
BRANUTIL C       7892   5-22-89  11:24a
FUSE     C       8906   5-22-89  11:16a
CVSOURCE C      10074   6-26-89  12:24p
READWRIT C       6388   6-25-89   5:40p
ELUTIL   C       7390   5-22-89  11:27a
MODCIR   C      20340   8-15-89  12:27p
TCMORE   C       2916   5-02-89   7:28p
DISPLAY  C       1763   5-02-89   4:51p
```

```
CREATCIR C          780   5-08-89   4:32p
MOREHELP C         9102   5-22-89  11:25a
TLINE    C         5426   8-15-89  12:42p
```

spfedit (spedit.h)
menucall (spedit.h)
tmalloc
getopt
swchar
branch (spedit.h)
branutil (spedit.h)
fuse (spedit.h)
cvsource (spedit.h)
tline (spedit.h)
readwrit (spedit.h)
elutil (spedit.h)
modcir (spedit.h)
tcmore (spedit.h)
display (spedit.h)
creatcir (spedit.h)
morehelp

```
/*spedit.h - include file for spicedriver program
    K.H.Carpenter - 29APR89
 */

/*other files to be included*/

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <dir.h>
#include <errno.h>

/*symbols defined for all modules including this file*/

#define MAXSCRHT 25
#define makzero(A) makdefault(A,"0")

/*typedefs for storage structures for circuit*/

typedef struct element{
    char *elem; /*string giving line(s) in spice file for this entry)*/
    char *comm; /*string giving comment lines following this entry*/
    struct element *next;   /*pointer to next element*/
    struct element *last;   /*pointer to previous element*/
}Element;       /*Note: can't recursively use the typedef inside itself*/

typedef struct circuit{
    char *path; /*file spec for this circuit*/
    Element *title; /*first line and following comments*/
    Element *branch;/*first element card in linked list*/
    Element *lbr;   /*last element card in linked list*/
    Element *option;/*first option card in linked list*/
    Element *lop;   /*last option card in linked list*/
}Circuit;

/*function prototypes: */
char menucall(char* title, int num, char **items);
char *pardup(char *dest, char *sour);
void *tmalloc(unsigned length);
void *trealloc(void *pointer, unsigned length);
void freeparr:t(char **par, int number);
void spedituse(void);
void modifycir(void);
void loadcir(void);
void loadc(char *path);
void savecir(void);
void renamecir(void);
void newtitle(void);
void chcomments(void);
void copyelq(Element *newel, char *type, char first);
void addunit(char **place, char *unit);
void Displaycir();
void displaycir(int part, Circuit *cir, int flag, int height);
void exitcir(void);
void delel(Element *drop);
void upel(Element *el);
void clearcir(Circuit *cir);
void delcir(Circuit *cir);
void moreopen(int lines, char *title);
void moreclose(void);
int forceend(Circuit *cir);
int issep(char ch);
int makdefault(char **ze, char *string);
int morehelp(char *topic, char *title);
int morehlpini(char *name, char *envir);
int moreput(char *text, int init);
int ncomp(char *a, char *b);
int newdouble(char *prompt, char **place);
int newname(char first, char *type, char **place);
int newnode(char *where, char **place, char *test);
int Readcir(Circuit *cir);
int Readnext(FILE *fp, Element *ele);
int uniqnam(char *inname,int n);
int waitforkey(char *prompt);
char **spparse(char *string, int *number);
char *spunparse(char **pt);
void Creatcir(void);
int writecir(Circuit *cir);
Element *Findlist(char *type, char first, int dflag);
Element *Find(char *name, Element *start);
Element *addel(Element *place);
Circuit *creatcir(char *inpath);


/*global data*/

#ifdef MAIN
char Version[] = "SPFEDIT version 0.9, Copyright 1989, K.H.Carpenter";

Circuit *maincir;   /*circuit being edited*/
Element *firstel;   /*element in linked list*/
Element *firstco;   /*command in linked list*/
char *cirpath, *maintitle;
char cirname[9],cirdir[80],cirext[5],cirdriv[3];
char maintitl1[] = "Main Menu for circuit ";
char defaultcirpath[] = "spicein.cir";
char defaultcirtitle[]= "SPICE input file\n";
char Pressany[] = "(Press any key to continue)";

int circuitstat;    /*status bits for circuit   */
          /*=0 means undefined        */
          /*=1 means not modified since write*/
          /*=2 means modified since write */
int alcu = 1;       /*=0 means al defaults for fuse; =1 means cu*/
int helpav; /*true if help available*/

#else
extern Circuit *maincir;
extern Element *firstel;
extern Element *firstco;
extern char *cirpath, *maintitle;
extern char cirname[9],cirdir[80],cirext[5],cirdriv[3];
```

```
extern char maintitl1[];
extern char defaultcirpath[];
extern char defaultcirtitle[];
extern char Pressany[];
extern int circuitstat;
extern int alcu;
extern int helpav;

#endif
```

```
/*spfedit.c - main routine for SPFEDIT program
    K.H.Carpenter - 25JUN89
 */

#define MAIN
/* defining MAIN to place global data in this module */
#include "spedit.h"

#ifdef MDEBUG
/*global definition of stream for logging memory allocation*/
FILE *mfile;
#endif

/*menus*/

char *mainmen[7] = {
    "Modify circuit",
    "Load circuit from file",
    "Save circuit to file",
    "Create a new circuit",
    "Rename circuit",
    "Display circuit",
    "Exit SPFEDIT"
    };

/*****************************************************************/

/*main - main function for spfedit program*/

main(int argc, char **argv)
{

    char *tmp;
    int optch, hlpflg = 0, verflag = 0;
    extern int optind;

    circuitstat = 0;
    maincir = NULL;
    firstco = NULL;
    firstel = NULL; /*shows no circuit yet input*/

#ifdef MDEBUG
/*open file for logging memory allocations*/
    mfile = fopen("spfmemal.log","w");
#endif
/*process command line*/
    while((optch = getopt(argc, argv, "hHvV")) != EOF)
        switch(optch)
        {
            case 'v':
            case 'V':
                    verflag = 1;
                    break;

            case 'h':
            case 'H':
                    hlpflg = 1;
                    break;
            default:
                    spedituse();
                    exit(2);
        }

    if(optind < argc)
    {
        cirpath = argv[optind];
    }
/*Default circuit file name:*/
    else
    {
        cirpath = defaultcirpath;
    }
/*Put cirpath in an allocated string so it can be freed later*/
    tmp = (char*)tmalloc(strlen(cirpath)+1);
    strcpy(tmp,cirpath);
    maincir = creatcir(tmp);
    if(access(cirpath, 0) == 0)
    {   if(Readcir(maincir))circuitstat = 1;
        else clearcir(maincir);
    }
/*To allow newtitle to work*/
    if(maincir->title == NULL)
    {
        maincir->title = addel(NULL);
        maincir->title->elem = (char*)tmalloc(strlen(defaultcirtitle)+1);
        strcpy(maincir->title->elem,defaultcirtitle);
    }

/*Split path into parts to make menu title*/
    fnsplit(cirpath,cirdriv,cirdir,cirname,cirext);
    maintitle = (char*)tmalloc(strlen(maintitl1) + strlen(cirname) + 1);
    strcpy(maintitle,maintitl1);
    strcat(maintitle,cirname);
    if(verflag)
    {
        puts(Version);
        waitforkey(Pressany);
    }
    helpav = morehlpini("SPFEDIT.HLP","SPICEHLP");
    if(helpav)if(!morehelp("title\n",0))
    {
        gotoxy(1,24);
        if((waitforkey(Pressany)|0x20) == 'h')hlpflg = 1;
    }
    if(hlpflg && helpav)if(!morehelp("begin\n","SPICE EDITOR INSTRUCTIONS"))
        waitforkey(Pressany);
/*main menu loop*/

    while(1)switch(menucall(maintitle,7,mainmen))
    {
        case '1':   modifycir();
                    break;

        case '2':   loadcir();
```

```
                break;

        case '3':   savecir();
                break;

        case '4':   Creatcir();
                break;

        case '5':   renamecir();
                break;

        case '6':   Displaycir();
                break;

        case 'R':
        case '7':   exitcir();
                break;

        case 'H':
        default:    if(helpav)
                  if(!morehelp("main\n","HELP for MAIN MENU"))
                    waitforkey(Pressany);
    }
}

/*****************************************************************/

/*exitcir - function to call to exit program*/

void exitcir(void)
{
    if(circuitstat)if(Find(".TRAN",maincir->option)==NULL)
    {
        puts("\n\
WARNING - no transient analysis command included in circuit.\n\
Exit anyhow?(y or n):");
        if(getch() != ('y'|'Y'))return;
    }
    if(circuitstat == 2)
    {
        puts("\nCircuit has not been saved since last modified.\
   Exit anyhow?(y or n):");
        if(getch() != ('y'|'Y'))return;
    }
    exit(0);
}

/*****************************************************************/

void spedituse(void)
{
    puts("Use: spfedit [-hv] [pathtocircuitfile]\n");
}
```

```
/*MENUCALL.C - K.H.Carpenter - 04MAY89
 *This routine puts a menu on the screen, using TURBOC windowing calls.
 *Must be called with a full screen width window with bottom at 25, but
 *top can be greater than 1.
 *title argument is centered.  num is number of items in menu. items is
 *the array of menu items to display.
 *Return is the character (not the integer) of the number of the item
 *selected.
 *(If over 9 items, character returned is hexidecimal one.  Over 15
 * results in no way to return extra values. (Menu too long anyhow if over 15.)
 *If H, h, or ? is entered then the character H is returned (to allow
 *a "help" feature).
 */

#include <conio.h>
#include <string.h>

char menucall(char* title, int num, char **items){
    int i,j,k;
    char ch;
    struct text_info inwin;
    gettextinfo(&inwin);
    clrscr();
    gotoxy(j = ((80 - strlen(title))/2),1);
    cputs(title);
    gotoxy(j,2);
    for(i=0; i< strlen(title); i++)putch('-');
    j = wherey() + inwin.wintop - 1;
    for(k=0,i=0; i<num; i++){
        window(5,j+k+2,80,25);
        cprintf("%X. ",i+1);
        window(8,j+k+2,80,25);
        cputs(items[i]);
        k += wherey();
    }
    window(1,inwin.wintop,80,25);
#if 0
cprintf("ht=%d, top=%d, bot=%d\n",inwin.screenheight,
inwin.wintop,inwin.winbottom);
/*This test print shows that TurboC1.5 has a bug in gettextinfo.
 *The screenheight entry has the bottom value (when 25) not bot-top.
 */
#endif
    while(1){
        int scrht;
        scrht = inwin.winbottom - inwin.wintop + 1;
        gotoxy(1,scrht-3);
        if(num > 9)cputs("Enter number or letter of selection:");
        else cputs("Enter number of selection:");
        ch = getche();
#ifdef DEBUG
        if(ch == 3)exit(1); /*To allow ^c break for testing.  Remove when
                              bugs are out.*/
#else
        if(ch == 3 || ch == 'r' || ch == 'R')return('R');
         /*^c causes return to previous level*/
#endif
        if(ch <= ('0'+num) && ch > '0')return(ch);
        if(ch == 'h' || ch == 'H' || ch == '?')return('H');
        if(num > 9)
        {
            ch |= 0x20;
            if(ch == 'a' || ch == 'b' || ch == 'c' || ch == 'd' || ch == 'e'
                || ch == 'f')return(ch);
        }
        gotoxy(27,scrht-3);
        cprintf(" \n%c is not a valid selection, try again.",ch);
    }
}
```

```
/*tmalloc.c - K.H.Carpenter - 22MAY89
 *malloc and realloc with error checks for circuit editor program.
 *Optional recording of pointers allocated and freed if MDEBUG is
 *defined to the compiler.  In that case, output stream, mfile,
 *must have been defined and opened before calls to these routines,
 *and calls to free must have been replaced by calls to tfree.
 */

#include "spedit.h"
#include <alloc.h>
#ifdef MDEBUG
    extern FILE *mfile;
#endif

void MRerror(unsigned length)
{
    if(!maincir)fprintf(stderr,"\nUnable to allocate %u bytes\
 - aborting\n",length);
    else
    {
        fprintf(stderr,"\nUnable to allocate %u bytes.\n\
Enter file specification to use to save circuit before aborting:\n",length);
        {   char buffer[81], *nl;
            fgets(buffer,81,stdin);
            nl = strchr(buffer,'\n');
            *nl = 0;
            maincir->path = buffer;
#if 0
            if(writecir(maincir))fprintf(stderr,"Saved circuit\
 to %s\n- aborting -\n",buffer);
            else fprintf(stderr,"\nUnable to save circuit\
 - aborting -\n");
#endif
            savecir();
            fprintf(stderr,"\nAborted edit\n");
        }
    }
    exit(3);
}

void *tmalloc(unsigned length)
{
    void *pointer;
    if((pointer = malloc(length))==NULL)MRerror(length);
#ifdef MDEBUG
    fprintf(mfile,"malloced %u bytes at %p\n",length,pointer);
#endif
    return(pointer);
}

void *trealloc(void *ptr, unsigned length)
{
    void *pointer;
    if((pointer = realloc(ptr, length))==NULL)MRerror(length);
#ifdef MDEBUG
    fprintf(mfile,"realloced %u bytes at %p from %p\n",length,pointer,ptr);
#endif
    return(pointer);
}

void tfree(void *pointer)
{
    if(pointer == NULL)return;
    free(pointer);
#ifdef MDEBUG
    fprintf(mfile,"freed bytes at %p\n",pointer);
#endif
}
```

```
/*GETOPT.C - for PRTC.C - from pr - 07NOV88 - KHC*/
/*  Modified to use either '-' or the MSDOS value as switch character*/
#include <string.h>
#define index(a,b) strchr(a,b)
/*LINTLIBRARY*/
#define NULL    0
#define EOF (-1)
#ifndef NAME
#define NAME    "options"
#endif
/*
#define ERR(s, c)   if(opterr){\
    extern int strlen(), write();\
    char errbuf[2];\
    errbuf[0] = c; errbuf[1] = '\n';\
    (void) write(2, argv[0], (unsigned)strlen(argv[0]));\
    (void) write(2, s, (unsigned)strlen(s));\
    (void) write(2, errbuf, 2);}
*/
/*
extern int strcmp();
extern char *index();
*/
int opterr = 1;
int optind = 1;
int optopt;
char    *optarg;

int
getopt(argc, argv, opts)
int argc;
char    **argv, *opts;
{
    static int sp = 1;
    register int c;
    register char *cp;

    if(sp == 1)
        if(optind >= argc ||
           (argv[optind][0] != swchar() && argv[optind][0] != '-')
                   || argv[optind][1] == '\0')
            return(EOF);
/*      else if(strcmp(argv[optind], "--") == NULL) {
            optind++;
            return(EOF);
*/      else if(argv[optind][1] == '-' && argv[optind][2] == '\0') {
            optind++;
            return(EOF);
        }
    optopt = c = argv[optind][sp];
    if(c == ':' || (cp=index(opts, c)) == NULL) {
        ERR(": illegal option -- ", c);
        if(argv[optind][++sp] == '\0') {
            optind++;
            sp = 1;
        }
        return('?');
    }
    if(*++cp == ':') {
        if(argv[optind][sp+1] != '\0')
            optarg = &argv[optind++][sp+1];
        else if(++optind >= argc) {
            ERR(": option requires an argument -- ", c);
            sp = 1;
            return('?');
        } else
            optarg = argv[optind++];
        sp = 1;
    } else {
        if(argv[optind][++sp] == '\0') {
            sp = 1;
            optind++;
        }
        optarg = NULL;
    }
    return(c);
}

char name[] = NAME;

static ERR(s, c)
char s[];
int c;
{
    extern unsigned int strlen(), write();
    char errbuf[3];

    if(opterr){
    errbuf[0] = c; errbuf[1] = '\r'; errbuf[2] = '\n';
    write(2, name, (unsigned)strlen(name));
    write(2, s, (unsigned)strlen(s));
    write(2, errbuf, 3);}
}
```

```
/*SWCHAR.C - 07NOV88 - K.H.Carpenter
  TurboC version to return MSDOS switch character*/
#include <dos.h>
int swchar(void){
    union REGS regs;
    regs.x.ax = 0x3700;
    intdos(&regs,&regs);
    return(regs.x.cflag ? 0 : regs.h.dl);
}
```

```
/*branch.c - K.H.Carpenter - 22MAY89
 *Module for adding and changing elements and commands.
 */
/*TurboC v1.5 has a bug in its cprintf routine that causes it to
 *print garbage whenever a correct printing would cause the line to
 *wrap.  Can use printf when this is anticipated to be a problem and
 *when printf is otherwise ok.
 */

#include "spedit.h"
#include <ctype.h>

/*menus*/

char *capmenu[7] = {
    "Change name",
    "Change value",
    "Change first node",
    "Change second node",
    "Specify initial condition (or change it)",
    "Delete initial condition",
    "Save present values and return"
    };

char *indmenu[8] = {
    "Change name",
    "Change value",
    "Change first node",
    "Change second node",
    "Specify initial condition (or change it)",
    "Delete initial condition",
    "Change to saturation model",
    "Save present values and return"
    };

char *satindmenu[9] = {
    "Change name",
    "Change first node",
    "Change second node",
    "Change initial inductance",
    "Change inductance after saturation",
    "Change saturation current",
    "Change saturation model exponent",
    "Change to linear model",
    "Save present values and return"
    };

char *resmenu[7] = {
    "Name",
    "Value",
    "First node",
    "Second node",
    "First temperature coefficient",
    "Second temperature coefficient",
    "Save present values and return"
    };

/*************************************************************/

/*resistor - function to add or change a resistor element.
 *Call with pointer to element.
 */

void resistor(Element *el)
{
    int numpar, len, change = 0;
    char **par;
    int number;
    par = spparse(el->elem, &number);
    if(number < 7)
    {
        int i;
        par = trealloc(par, 8*sizeof(char *));
        for(i = number + 1; i < 8 ; i++)par[i] = NULL;
        number = 7;
    }
    /*Force values to be non-null, for use when creating a new
     *element.  Defaults values not specified to zero.
     */
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    change |= makzero(&par[3]);
/*Now loop on menu for changing values:*/
  while(1)
  {
    clrscr();
    cprintf("Resistor present values:\nName: %s      Value: %s\n\n\
First node:  %s     Second node:  %s\n\n",par[0],par[3],par[1],par[2]);
    if(par[4] != NULL)
    {
        strupr(par[4]);
        if(strcmp(par[4],"TC"))
        {
            cprintf("Illegal option %s for resistor - ignored\n"\
                ,par[4]);
            tfree(par[4]);
            par[4] = NULL;
            tfree(par[5]);
            par[5] = NULL;
            tfree(par[6]);
            par[6] = NULL;
        }
        else
        {
            if(((par[5] == NULL) || !strcmp(par[5],"0"))
             && ((par[6] == NULL) || !strcmp(par[6],"0")))
            {
                tfree(par[4]);
                par[4] = NULL;
                tfree(par[5]);
                par[5] = NULL;
                tfree(par[6]);
                par[6] = NULL;
                continue;
```

```
            }
            else
            {
                makzero(&par[5]);
                makzero(&par[6]);
            cprintf("First temp coef.:  %s     Second temp coef.:  \
  %s\n\n",par[5],par[6]);
            }
        }
    }
    else
    {
        if(par[5]!=NULL || par[6]!=NULL)
        {
            par[4] = (char *)tmalloc(3);
            strcpy(par[4],"TC");
            continue;
        }
        else    cputs("No temperature coefficient specified.\n\n");
    }
    window(1,9,80,25);
    switch(menucall("Choose item to change (or to keep all):",7,resmenu))
    {
        case '1':
            if(newname('R',"resistor",&par[0]) > 0)change = 1;
            break;

        case '2':
    change |= newdouble("Enter new resistance value, in Ohms:",&par[3]);
            break;

        case '3':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '4':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '5':
            change |= newdouble("Enter new value for first \
temperature coefficient:",&par[5]);
            break;

        case '6':
            change |= newdouble("Enter new value for second \
temperature coefficient:",&par[6]);
            break;

        case '7':
        case 'R':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'H':
        default:
            if(helpav)if(imorehelp("resistor\n","HELP for RESISTOR"))
                waitforkey(Pressany);
    }   /*end of switch*/
    window(1,1,80,25);
   }
 }

/*********************************************************/

/*capacitor - function to add or change a capacitor element.
 *Call with pointer to element.
 */

void capacitor(Element *el)
{
    int numpar, len, change = 0;
    char **par;
    int number;
    par = spparse(el->elem, &number);
    if(number < 6)
    {
        int i;
        par = trealloc(par, 7*sizeof(char *));
        for(i = number + 1; i < 7 ; i++)par[i] = NULL;
        number = 6;
    }
    /*Force values to be non-null, for use when creating a new
     *element.  Defaults values not specified to zero.
     */
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    change |= makzero(&par[3]);
/*Now loop on menu for changing values:*/
  while(1)
  {
    clrscr();
    cprintf("Capacitor present values:\nName: %s      Value: %s\n\n\
First node:  %s     Second node:  %s\n\n",par[0],par[3],par[1],par[2]);
    if(par[4] != NULL)
    {
        strupr(par[4]);
        if(strcmp(par[4],"IC"))
        {
            cprintf("Illegal option %s for capacitor - ignored\n"\
                ,par[4]);
            tfree(par[4]);
            par[4] = NULL;
            tfree(par[5]);
            par[5] = NULL;
        }
```

```
        else
        {
            makzero(&par[5]);
            cprintf("Initial voltage: %s\n\n",par[5]);
        }
    }
    else
    {
        if(par[5]!=NULL)
        {
            par[4] = (char *)tmalloc(3);
            strcpy(par[4],"IC");
            continue;
        }
        else    cputs("No initial voltage specified.\n\n");
    }
    window(1,9,80,25);
    switch(menucall("Choose action:",7,capmenu))
    {
        case '1':
            if(newname('C',"capacitor",&par[0]) > 0)change = 1;
            break;

        case '2':
    change |= newdouble("Enter new capacitance value, in Farads:",&par[3]);
            break;

        case '3':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '4':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '5':
            change |= newdouble("Enter new value for \
initial voltage, in volts:",&par[5]);
            break;

        case '6':
            if(par[4] == NULL)break;
            change = 1;
            tfree(par[4]);
            tfree(par[5]);
            par[4] = par[5] = NULL;
            break;

        case '7':
        case 'R':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'N':
        default:
            if(helpav)if(!morehelp("capacitor\n","HELP for CAPACITOR"))
                waitforkey(Pressany);
    }   /*end of switch*/
    window(1,1,80,25);
  }
}

/*********************************************************/

/*inductor - function to add or change an inductor element.
 *Call with pointer to element.
 */

void inductor(Element *el)
{
    int numpar, len, poly = 0, change = 0;
    char **par;
    int number;
    par = spparse(el->elem, &number);
    if(number < 9)
    {
        int i;
        par = trealloc(par, 10*sizeof(char *));
        for(i = number + 1; i < 10 ; i++)par[i] = NULL;
        number = 9;
    }
    /*Force values to be non-null, for use when creating a new
     *element.  Defaults values not specified to zero.
     */
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    change |= makzero(&par[3]);
    if(!strcmp(par[3],"POLY"))
    {
        poly = 1;
        if(!par[4] || strcmp(par[4],"-1234"))
        {
            cputs("General polynomial inductor elements cannot \
be edited using this program.\n\
Save the circuit to a file and use a text editor to edit this element.\n");
            waitforkey(Pressany);
            freeparret(par,number);
            return;
        }
        change |= makzero(&par[5]);
        change |= makzero(&par[6]);
        change |= makdefault(&par[7],"1");
        change |= makdefault(&par[8],"1");
    }
    else if(par[6])
    {
```

```
        cputs("Excess items in this element.  Remove them?(y or n):");
        if((getche() | 0x20) == 'n')
        {
            freeparret(par,number);
            return;
        }
        tfree(par[6]);
        par[6] = NULL;
    }
/*Now loop on menu for changing values:*/
  while(1)
  {
     clrscr();
     if(!poly)
     {
    cprintf("Inductor present values:\nName: %s      Value: %s\n\n\
First node:  %s     Second node:  %s\n\n",par[0],par[3],par[1],par[2]);
    if(par[4] != NULL)
    {
        strupr(par[4]);
        if(strcmp(par[4],"IC"))
        {
            cprintf("Illegal option %s for inductor - ignored\n"\
                ,par[4]);
            tfree(par[4]);
            par[4] = NULL;
            tfree(par[5]);
            par[5] = NULL;
        }
        else
        {
            makzero(&par[5]);
            cprintf("Initial current:  %s\n\n",par[5]);
        }
    }
    else
    {
        if(par[5]!=NULL)
        {
            par[4] = (char *)tmalloc(3);
            strcpy(par[4],"IC");
            continue;
        }
        else    cputs("No initial current specified.\n\n");
    }
    window(1,9,80,25);
    switch(menucall("Choose action:",8,indmenu))
    {
        case '1':
            if(newname('L',"inductor",&par[0]) > 0)change = 1;
            break;

        case '2':
    change |= newdouble("Enter new inductance value, in Henries:",&par[3]);
            break;

        case '3':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '4':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '5':
            change |= newdouble("Enter new value for \
initial current, in Amperes:",&par[5]);
            break;

        case '6':
            if(par[4] == NULL)break;
            change = 1;
            tfree(par[4]);
            tfree(par[5]);
            par[4] = par[5] = NULL;
            break;

        case '7':
            tfree(par[4]);
            tfree(par[5]);
            tfree(par[6]);
            tfree(par[7]);
            tfree(par[8]);
            par[4]=par[5]=par[6]=par[7]=par[8]=NULL;
            makdefault(&par[4],"-1234");
            makdefault(&par[5],par[3]);
            tfree(par[3]);
            par[3] = NULL;
            change |= makdefault(&par[3],"POLY");
            makzero(&par[6]);
            makdefault(&par[7],par[5]);
            makdefault(&par[8],"1");
            poly = 1;
            break;

        case '8':
        case 'R':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'H':
        default:
            if(helpav)if(!morehelp("inductor\n","HELP for INDUCTOR"))
                waitforkey(Pressany);
    }   /*end of switch*/
     }      /*end of !poly*/
```

```
     else   /*poly is true - hence saturatable inductor*/
     {
    cprintf("Inductor present values:\n\nName: %s      \
First node:  %s     Second node:  %s\n\nInitial inductance: %s       \
Inductance after saturation: %s\n\nSaturation current: %s       \
Power for model: %s\n\n",par[0],par[1],par[2],par[5],par[6],par[7],par[8]);
    window(1,9,80,25);
    switch(menucall("Choose action:",>,satindmenu))
    {
        case '1':
            if(newname('L',"inductor",&par[0]) > 0)change = 1;
            break;

        case '2':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '3':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '4':
change |= newdouble("Enter new value for initial inductance, in Henries:",
            &par[5]);
            break;

        case '5':
    change |= newdouble("Enter new value for inductance after \
saturation, in Henries:",&par[6]);
            break;

        case '6':
            change |= newdouble("Enter new value for \
saturation current, in Amperes:",&par[7]);
            break;

        case '7':
            change |= newdouble("Enter new value for \
power in hyperbolic tangent model:",&par[8]);
            break;

        case '8':
            change = 1;
            tfree(par[3]);
            tfree(par[4]);
            tfree(par[5]);
            tfree(par[6]);
            tfree(par[7]);
            tfree(par[8]);
            par[3]=par[4]=par[5]=par[6]=par[7]=par[8]=NULL;
            makzero(&par[3]);
            poly = 0;
            break;

        case '9':
        case 'R':
            /*here to save element, free storage allocated in
              this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'H':
        default:
            if(helpav)if(!morehelp("inductor\n","HELP for INDUCTOR"))
                waitforkey(Pressany);
    }   /*end of switch*/
     }      /*end of poly*/
     window(1,1,80,25);
   }
 }

/*********************************************************/

/*defaultel - function to add or change an unspecified element.
 *Call with pointer to element.
 */

void defaultel(Element *el)
{
    int strpos;
    char buffer[72];
    if(el->elem == NULL)
    {
        cputs("\nEnter a line to be placed in the SPICE input file\
 as the \"card\" for the element\nas specified in the SPICE users' guide.\
\n(If the desired card needs a continuation line, save this circuit to a\n\
 file and use a text editor to add the continuation line.\n\
 Only 71 characters may be entered here.)\n>");
        fgets(buffer,72,stdin);
        fflush(stdin);
        strpos = strlen(buffer);
        if(buffer[strpos-1]!='\n')buffer[strpos-1] = '\n';
        el->elem = (char *)tmalloc(strpos+1);
        strcpy(el->elem,buffer);
    }
    else
    {
        /*c*/printf("SPICE element:\n%s\nThe element above is not one\
 of the special types that can be edited\nby this program.\n\
To change this element, save the circuit to a file and use a text editor.\n",
            el->elem);
            waitforkey(Pressany);
            putch('\n');
    }
}
```

```
/*branutil.c - K.H.Carpenter - 22MAY89
 *Module for utility routines relating to branch routines.
 */

#include "spedit.h"
#include <ctype.h>

/*********************************************************/

/*makdefault - function to test pointer of first argument for NULL,
 *and if so to allocate a string and copy the second argument to it
 *and change the NULL pointer to point it.
 *Returns 0 if pointer was not NULL, 1 if it was NULL on call.
 */

int makdefault(char **ze, char *string)
{
    if(*ze != NULL)return(0);
    *ze = (char *)tmalloc(strlen(string)+1);
    strcpy(*ze,string);
    return(1);
}

/*********************************************************/

/*waitforkey - function to give a prompt and wait for a key from console.
 *Call with prompt string.  Returns key pressed.
 */

int waitforkey(char *prompt)
{
    cputs(prompt);
    return(getch());
}

/*********************************************************/

/*newname - function to get new name for an element.
 *Call with first character required for name, string giving
 *class of element, pointer to pointer to present name storage.
 *Returns 1 if name changed, -1 if only a <CR> was entered, and
 *0 otherwise.
 */

int newname(char first, char *type, char **place)
{
    int chang = 0;
    char *blk, inname[41];
    first &= 0xdf;
    clrscr();
    cprintf("Enter new name (must begin with '%c'):",first);
    fgets(inname,41,stdin);
    fflush(stdin);
    if(inname[0] == '\n')return(-1);
    if((blk=strchr(inname,'\n'))!=NULL)
        *blk = '\0';
    strupr(inname);
    if(inname[0] != first)
    {   cprintf("Sorry, %s name must begin with %c\
, try again.\n",type,first);
    }
    else if(uniqnam(inname,1))
    {
        cprintf("Name '%s' is already in use.  Try\
 again.",inname);
    }
    else
    {
        tfree(*place);
        *place = (char*)tmalloc(strlen(inname)+1);
        strcpy(*place,inname);
        chang = 1;
    }
    if(!chang)waitforkey(Pressany);
    return(chang);
}

/*********************************************************/

/*newnode - function to get new number for a node.
 *Call with string giving which node is being changed, pointer
 *to pointer to location storing the present node as a string,
 *pointer to string having node to be compared to insure different
 *value.
 *Returns 1 if node changed, 0 otherwise.
 */

int newnode(char *where, char **place, char *test)
{
    int n, cs;
    char nbuf[10];
    clrscr();
    cprintf("Enter new node number for %s node:",where);
    cs = scanf("%d",&n);
    fflush(stdin);
    if(!cs || cs == EOF || n < 0 || n > 32000)
    {   cputs("Illegal value, try again.\n");
        waitforkey(Pressany);
        return(0);
    }
    sprintf(nbuf,"%d",n);
    if(!strcmp(nbuf,test))
    {   cputs("Nodes must be different, try again.\n");
        waitforkey(Pressany);
        return(0);
    }
    tfree(*place);
    *place = (char*)tmalloc(strlen(nbuf)+1);
    strcpy(*place,nbuf);
    return(1);
}

/*********************************************************/
```

```
/*newdouble - function to replace a float value in element.
 *Call with pointer to prompt string, pointer to place to store
 *resulting string.  Returns 1 if result string changed, 0 if not.
 */

int newdouble(char *prompt, char **place)
{
    double res;
    char resb[30];
    int cs;
    clrscr();
    cputs(prompt);
    cs = scanf("%E",&res);
    fflush(stdin);
    if(!cs || cs == EOF)
    {   cputs("Illegal value, try again.\n");
        waitforkey(Pressany);
        return(0);
    }
    sprintf(resb,"%G",res);
    tfree(*place);
    *place = (char*)tmalloc(strlen(resb)+1);
    strcpy(*place,resb);
    return(1);
}

/*********************************************************/

/*addunit - function to add a unit after the number placed
 *in a parameter by a call to newdouble.
 *Call with pointer to pointer to string, unit as string.
 */

void addunit(char **place, char *unit)
{
    *place = trealloc(*place, strlen(*place) + strlen(unit) + 1);
    strcat(*place,unit);
}

/*********************************************************/

/*issep - function to determine if a character is a SPICE separator.
 *Call with character to be tested.  Returns true if a separator.
 */

int issep(char ch)
{
    if(!ch||isspace(ch)||ch == '=' || ch == ',' || ch == '(' || ch == ')')
        return(1);
    return(0);
}

/*********************************************************/

/*spparse - function to separate a SPICE input card (and its
 *continuations) into substrings.
 *Call with pointer to input string, and pointer to integer
 *to store number of substrings found.
 *Returns pointer array of pointers to the substrings.  The
 *last pointer in the array is followed by a NULL pointer.
 *Also returns the number of substrings set.
 *Storage is allocated to the pointers in this routine.
 *It should be freed by the caller when no longer needed.
 */
/*This version combines continuation cards by replacing the
 *+ in the first column with a space.  This seems to be
 *compatible with how SPICE2 reads them.
 */

char **spparse(char *instring, int *number)
{
    char *string, *buf, **pt, ch;
    int i,j;
    unsigned len = 0;
    i = strlen(instring) + 1;
    string = tmalloc(i);
    pt = tmalloc(sizeof( char *));
    buf = tmalloc(i);
    buf[0] = 0;
    *pt = NULL;
    /*first clear continuations*/
    ch = 0;
    for(j=0;j<i;j++)
    {
        string[j] = instring[j];
        if((ch == '\n') && (string[j] == '+'))string[j] = ' ';
        ch = string[j];
    }
    for(i=0,j=0;;)
    {
        if(issep(string[i]) || !string[i])
        {
            if(len)
            {
                pt[j] = tmalloc(++len);
                strcpy(pt[j],buf);
            }
            ++j;
            len = 0;
            buf[0] = 0;
            pt = trealloc(pt, (j+1)*sizeof(char *));
            pt[j] = NULL;
/*discard multiple spaces*/
            while(isspace(string[i]))i++;
            if(!string[i])
            {
                *number = j;
                tfree(buf);
                tfree(string);
#ifdef DEBUG
    for(i=0;i<j;i++)printf("\npar[%d] = %s",i,pt[i]);
    cputs("\nPress any key to continue");
    getche();
#endif
```

```
                return(pt);
            }
            if(issep(string[i]))
            {
                i++;
                while(isspace(string[i]))i++;
            }
        }
        else
        {
            buf[len++] = string[i];
            buf[len] = 0;
            i++;
        }
    }
}

/*********************************************************/

/*spunparse - function to combine separate strings into a SPICE
 *input card with separation of substrings by spaces and creation
 *of continuation lines when needed.
 *Call with pointer to array of strings; last string is indicated
 *by null pointer.
 *Returns pointer to SPICE input card (and continuations as a single
 *string).  Memory is allocated for the returned string by this routine.
 */

char *spunparse(char **pt)
{
    int i;
    unsigned sl, len, total;
    char *buf;
    buf = tmalloc(sl = strlen(pt[0]) + 2);
    strcpy(buf,pt[0]);
    len = total = sl;
    for(i = 1; pt[i]; i++)
    {
        sl = strlen(pt[i]) + 1;
        len += sl;
        total += sl;
        if(len > 73)
        {
            total++;
            len = 2;
            buf = trealloc(buf,total);
            strcat(buf,"\n+");
        }
        else
        {
            buf = trealloc(buf,total);
            strcat(buf," ");
        }
        strcat(buf,pt[i]);
    }
    strcat(buf,"\n");
    return(buf);
}

/*********************************************************/

/*freeparret - function to free space allocated by spparse function,
 *reset the window to full screen, and clear it.
 *Call with pointer parameter array, and number of items in array.
 */

void freeparret(char **par, int number)
{
    int numpar;
    for(numpar = 0; numpar <= number; numpar++)
        tfree(par[numpar]);
    tfree(par);
    window(1,1,80,25);
    clrscr();
}

/*********************************************************/

/*pardup - function to duplicate a one parameter in place of another.
 *Does the memory freeing and allocation.
 *Call with pointer to destination first, then to source.
 *Returns pointer to new destination string.
 */

char *pardup(char *dest, char *sour)
{
    char *new;
    tfree(dest);
    new = tmalloc(strlen(sour)+1);
    strcpy(new,sour);
    return(new);
}
```

```
/*fuse.c - K.H.Carpenter - 22MAY89
 *Module for routines relating to fuse elements.
 */

#include "spedit.h"
#include <ctype.h>

#define CUA "1U"
#define CUB "1U"
#define CUG "25E16"
#define CUS "5E16"

#define ALA "0.9U"
#define ALB "1.4U"
#define ALG "8.9E16"
#define ALS "1.8E16"

#define DKOE "20E10"
#define DPOW "0.19"

/*menus*/

char *fusmenu[8] = {
    "Change name",
    "Change first node",
    "Change second node",
    "Change length",
    "Change width",
    "Change thickness",
    "Change resistivity",
    "Save present values and return"
    };

char *fusmenu2[11] = {
    "New A (resistivity after melt)",
    "New B (peak resistivity)",
    "New G0 (normalized action to melt)",
    "New S0 (normalized action peak width)",
    "Replace values with copper defaults",
    "Replace values with aluminum defaults",
    "Change KOE (constant for scaling g0 and s0)",
    "Change POW (power for scaling g0 and s0)",
    "Replace KOE and POW with default values",
    "Change default material for new fuse elements",
    "Return to Fuse menu"
    };

/*********************************************************/

/*Defmat - function to set fuse resistivity parameters to default values.
 *Call with pointer to paramters, flag for type of material.
 *Flag is 1 if copper defaults, 0 if aluminum defaults, error otherwise.
 *Returns 1 on success, 0 on error.
 */

int Defmat(char **par, int matflg)
{
    if((matflg == 1) || !matflg)
    {
        int i;
        for(i = 11; i < 15; i++)
        {
            tfree(par[i]);
            par[i] = NULL;
        }
    }
    if(matflg == 1)
    {
        makdefault(&par[11],CUA);
        makdefault(&par[12],CUB);
        makdefault(&par[13],CUG);
        makdefault(&par[14],CUS);
    }
    else if(!matflg)
    {
        makdefault(&par[11],ALA);
        makdefault(&par[12],ALB);
        makdefault(&par[13],ALG);
        makdefault(&par[14],ALS);
    }
    else return(0);
    return(1);
}

/*********************************************************/

/*isalcu - function to test fuse resistivity parameters for default values.
 *Call with pointer to paramters.
 *Returns 1 if copper defaults, 0 if aluminum defaults, -1 otherwise.
 */

int isalcu(char **par)
{
    int i;
    if(!strcmp(par[11],CUA) && !strcmp(par[12],CUB) &&
        !strcmp(par[13],CUG) && !strcmp(par[14],CUS))
            return(1);
    if(!strcmp(par[11],ALA) && !strcmp(par[12],ALB) &&
        !strcmp(par[13],ALG) && !strcmp(par[14],ALS))
            return(0);
    return(-1);
}

/*********************************************************/

/*fusres - function to change fuse resistivity parameters.
 *Call with pointer to paramters.
 *Returns 1 if any changed, 0 otherwise.
 */

int fusres(char **par)
{
   char copper[] = "copper";
   char alum[]   = "aluminum";
```

```
    int tmp, change = 0;
    while(1)
    {
     window(1,1,80,25);
     clrscr();
     cprintf("Present resistivity model parameters:\n\
A = %s (ohm-m)          G0 = %s A^2-s/m^4         KOE = %s\n\n\
B = %s (ohm-m)          S0 = %s A^2-s/m^4         POW = %s\n\n\
                                          Default material: %s\n\n",
     par[11],par[13],par[15],par[12],par[14],par[16],alcu ? copper : alum);
     window(1,8,80,25);
     switch(menucall("Choose item to change (or to keep all):",11,fusmenu2))
     {
         case '1':
             tmp = newdouble("Enter new value for resistivity \
after melt, in microohm-meters:",&par[11]);
             if(tmp)addunit(&par[11],"U");
             change |= tmp;
             break;

         case '2':
             tmp = newdouble("Enter new value for peak \
resistivity, in microohm-meters:",&par[12]);
             if(tmp)addunit(&par[12],"U");
             change |= tmp;
             break;

         case '3':
             tmp = newdouble("Enter new value for normalized \
action to melt, in A^2-s/m^4:",&par[13]);
             if(tmp)addunit(&par[13],"U");
             change |= tmp;
             break;

         case '4':
             change |= newdouble("Enter new value for normalized \
action peak width, in A^2-s/m^4:",&par[14]);
             break;

         case '5':
             change |= Defmat(par,1);
             break;

         case '6':
             change |= Defmat(par,0);
             break;

         case '7':
             change |= newdouble("Enter new value for scaling \
constant KOE:",&par[15]);
             break;

         case '8':
             change |= newdouble("Enter new value for scaling \
power POW:",&par[16]);
             break;

         case '9':
             tfree(par[15]);
             tfree(par[16]);
             par[15] = par[16] = NULL;
             makdefault(&par[15],DKOE);
             makdefault(&par[16],DPOW);
             change = 1;
             break;

         case 'a':
             cputs("\nTo make new elements default to copper, enter \
'c'; to aluminum, enter 'a':");
             if((getche() | 0x20) == 'a')alcu = 0;
             else alcu = 1;
             break;

         case 'R':
         case 'b':
             return(change);

         case 'H':
         default:
             if(helpav)
              if(!morehelp("fusres\n","HELP for FUSE RESISTIVITY"))
                 waitforkey(Pressany);

     }
    }
}

/*********************************************************/

/*fuse - function to add or change a fuse element.
 *Call with pointer to element.
 */

void fuse(Element *el)
{
    int numpar, len, change = 0, tmp;
    char **par;
    static char *mat;
    char copper[] = "copper";
    char alum[]   = "aluminum";
    char other[]  = "other";
    int number;
    par = spparse(el->elem, &number);
    if(number < 17)
    {
        int i;
        par = trealloc(par, 18*sizeof(char *));
        for(i = number + 1; i < 18 ; i++)par[i] = NULL;
        number = 17;
    }
    /*Force values to be non-nil for use when creating a new
     *element.  Defaults the values not specified.
     */
    change |= makdefault(&par[3],"POLY");
```

```
    change |= makdefault(&par[4],"1");
    change |= makdefault(&par[7],"-1234");
/*check to see if this is a FUSE*/
    if(strcmp(par[3],"POLY") || strcmp(par[4],"1") ||
        strcmp(par[7],"-1234"))
    {
        cputs("General voltage controlled current sources cannot\
 be edited\nby this program.  Save the circuit to a file and use a text \
editor to\nchange this element.\n");
        waitforkey(Pressany);
        freeparret(par, number);
        return;
    }
    /*default node numbers to zero*/
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    change |= makzero(&par[5]);
    change |= makzero(&par[6]);
    /*default sizes set to zero to force entry if not copied*/
    change |= makzero(&par[8]);
    change |= makzero(&par[9]);
    change |= makzero(&par[10]);
    /*default resistivity paramters set according to last
     *value for alcu flag
     */
    if(alcu)
    {
        change |= makdefault(&par[11],CUA);
        change |= makdefault(&par[12],CUB);
        change |= makdefault(&par[13],CUG);
        change |= makdefault(&par[14],CUS);
    }
    else
    {
        change |= makdefault(&par[11],ALA);
        change |= makdefault(&par[12],ALB);
        change |= makdefault(&par[13],ALG);
        change |= makdefault(&par[14],ALS);
    }
    /*default values for dynamic scaling factors:*/
    change |= makdefault(&par[15],DKOE);
    change |= makdefault(&par[16],DPOW);
    /*check if material is to be defaulted or not*/

/*First and second set of nodes must be the same - force this*/
    if(!strcmp(par[1],par[5]))
    {
        tfree(par[5]);
        par[5] = tmalloc(strlen(par[1])+1);
        strcpy(par[5],par[1]);
    }
    if(!strcmp(par[2],par[6]))
    {
        tfree(par[6]);
        par[6] = tmalloc(strlen(par[2])+1);
        strcpy(par[6],par[2]);
    }
```

```
/*Now loop on menu for changing values:*/
  while(1)
  {
    int ismat;
    clrscr();
    /*check to see if a default material and set mat to it*/
    ismat = isalcu(par);
    if(!ismat)mat = alum;
    else if(ismat ==1)mat = copper;
    else mat = other;
    cprintf("Fuse present values:\nName: %s      Material: %s       \
Length: %s       Width: %s\n\n\
First node:  %s     Second node:  %s       Thickness: %s\n\n",
        par[0],mat,par[8],par[9],par[1],par[2],par[10]);
    window(1,9,80,25);
    switch(menucall("Choose item to change (or to keep all):",8,fusmenu))
    {
        case '1':
            if(newname('G',"fuse",&par[0]) > 0)change = 1;
            break;

        case '2':
            tmp = newnode("first",&par[1],par[2]);
            if(tmp)par[5] = pardup(par[5],par[1]);
            change |= tmp;
            break;

        case '3':
            tmp = newnode("second",&par[2],par[1]);
            if(tmp)par[6] = pardup(par[6],par[2]);
            change |= tmp;
            break;

        case '4':
            tmp = newdouble("Enter new value for length in \
mils:",&par[8]);
            if(tmp)addunit(&par[8],"MIL");
            change |= tmp;
            break;

        case '5':
            tmp = newdouble("Enter new value for width in \
mils:",&par[9]);
            if(tmp)addunit(&par[9],"MIL");
            change |= tmp;
            break;

        case '6':
            tmp = newdouble("Enter new value for thickness in \
mils:",&par[10]);
            if(tmp)addunit(&par[10],"MIL");
            change |= tmp;
            break;

        case '7':
            change |= fusres(par);
```

```
            break;

        case '8':
        case 'R':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'H':
        default:
            if(helpav)
             if(!morehelp("fuse\n","HELP for FUSE"))
                waitforkey(Pressany);
    }   /*end of switch*/
   window(1,1,80,25);
  }
}
```

```
/*cvsource.c - K.H.Carpenter - 26JUN89
 *Module for routines relating to independent source elements.
 */

#include "spedit.h"
#include <ctype.h>

/*menus*/

char *cvmenu[6] = {
    "Change name",
    "Change first node",
    "Change second node",
    "Change value",
    "Change to PWL (piecewise linear)",
    "Save present values and return"
    };

char *cvmenu2[6] = {
    "Change name",
    "Change first node",
    "Change second node",
    "Add or delete a time-value pair",
    "Change to DC (constant voltage)",
    "Save present values and return"
    };

char *cvmenu3[3] = {
    "Add a time-value pair",
    "Delete a time-value pair",
    "Return to previous menu"
    };

/*********************************************************/

/*parsort - function to compare times in time value pairs
 *for use with qsort on PWL element.
 *Call with pointers to time parameters.  Returns integer needed
 *by qsort.
 */

int parsort(char **pair1, char **pair2)
{
    double t1, t2;
    if(*pair1 == *pair2)return(0);
    if(!(*pair1))return(1);
    if(!(*pair2))return(-1);
    t1 = atof(*pair1);
    t2 = atof(*pair2);
    if(t1 > t2)return(1);
    if(t1 < t2)return(-1);
    return(0);
}

/*********************************************************/

/*pwlvalch - function to add or change time-value pair in a PWL source element.
 *Call with pointer to parameters, pointer to variable with number of
 *parameters.
 *Returns 1 if any change made, 0 otherwise.
 */

int pwlvalch(char ***par, int *number)
{
    int tmp;
    char *tchar = NULL;
    clrscr();
    switch(menucall("Choose action:",3,cvmenu3))
    {
    case '1':   /*add a pair*/
        if(!newdouble("Enter time for new pair, in seconds:"
            ,&tchar))return(0);
        if(tchar[0] == '-')
        {
            cputs("Time must not be negative, try again.\n");
            waitforkey(Pressany);
            tfree(tchar);
            return(0);
        }
        for(tmp=4;(*par)[tmp] && strcmp(tchar,(*par)[tmp]);
            tmp += 2);
        /*now tmp has index to the place in the par array where this
         *time for a pair is stored, or else it is the index to the
         *end of the array, pointing to a NULL entry
         */
        if((*par)[tmp])
        {
            printf("Time %s already in use, choose another.\n",
                tchar);
            waitforkey(Pressany);
            tfree(tchar);
            return(0);
        }
        *number += 2;
        *par = trealloc(*par,(*number+1)*sizeof(char *));
        (*par)[*number] = (*par)[*number-2] = tchar;
        (*par)[*number] = (*par)[*number-1] = NULL;
        if((*(*par)[0] | 0x20) == 'v')
            while(!newdouble("Enter value for new pair in volts:",
                &((*par)[*number-1])));
        else
            while(!newdouble("Enter value for new pair in amperes:"
                ,&((*par)[*number-1])));
        qsort(*par+4, (*number-4)>>1, 2*sizeof(char *), *parsort);
        return(1);

    case '2':   /*delete a pair*/
        tchar = NULL;
        if(!newdouble("Enter time of pair to delete:",&tchar))
            return(0);
/*khc 26jun89 - must not delete value at time zero*/
        if(!strcmp(tchar,"0"))
        {
            tfree((*par)[5]);
```

```
            (*par)[5] = NULL;
            if(!newdouble("Entry at time zero may not be deleted.\n\
Enter new value for entry at time zero:"
                ,&((*par)[5]))makzero(&((*par)[5]));
            tfree(tchar);
            return(1);
        }
        for(tmp=4; (*par)[tmp] && strcmp(tchar,(*par)[tmp]);
            tmp += 2);
        /*now tmp has index to the place in the par array where this
         *pair is stored, or else it is the index to the end of the
         *array, pointing to a NULL entry
         */
         if((*par)[tmp])
         {
            tfree((*par)[tmp]);
            tfree((*par)[tmp+1]);
            (*par)[tmp] = (*par)[tmp+1] = NULL;
        qsort(*par+4, (*number-4)>>1, 2*sizeof(char *), *parsort);
            *par =trealloc(*par,(--*number)*sizeof(char*));
            *number--;
            tfree(tchar);
            return(1);
        }
        else
        {
            cprintf("Pair not found with time = %s\n", tchar);
            waitforkey(Pressany);
            tfree(tchar);
            return(0);
        }

    case 'H':
    default:
        if(helpav)
         if(!morehelp("cvsadp\n","HELP for ADD/DELETE PAIRS"))
            waitforkey(Pressany);
    case 'R':
    case '3':
    }
    return(0);
}

/*********************************************************/

/*cvsource - function to add or change an independent source element.
 *Only DC and PWL sources are supported.  Others must be editied outside
 *this program.
 *Call with pointer to element.
 */

void cvsource(Element *el)
{
    int numpar, len, change = 0, tmp;
    char **par;
    int number;
    par = spparse(el->elem, &number);
    if(number < 6)
    {
        int i;
        par = trealloc(par, 7*sizeof(char *));
        for(i = number + 1; i < 7 ; i++)par[i] = NULL;
        number = 6;
    }
    /*Force values to be non-null for use when creating a new
     *element.  Defaults the values not specified.
     */
    /*default node numbers to zero*/
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    /*default element to a DC source*/
    change |= makdefault(&par[3],"DC");
/*check to see if this is DC or PWL, otherwise can't handle it*/
    if(strcmp(par[3],"DC") && strcmp(par[3],"PWL"))if(par[4])
    {
        cputs(
"Independent sources other than DC or PWL types cannot be edited \n\
by this program.  Save the circuit to a file and use a text editor \n\
to change this element.\n");
        waitforkey(Pressany);
        freeparret(par, number);
        return;
    }
    else
    {
/*If we get here with par[4] still NULL then it is a DC source without the
 *DC indicator in the card.  Change to place DC in it.
 *(Note that if par[4] is NULL then par[i] for i>4 will be also.)
 */
        makdefault(&par[4],par[3]);
        tfree(par[3]);
        par[3] = NULL;
        makdefault(&par[3],"DC");
    }
/*Complete the defaults as an ammeter or PWL with first time at zero*/
    change |= makzero(&par[4]);
/*Now loop on menu for changing values:*/
  while(1)
  {
    int isvolt;
    isvolt = ((*par[0] | 0x20) == 'v');
    clrscr();
/*check if DC or PWL and use different menus*/
     if(strcmp(par[3],"PWL"))
     {
    if(isvolt)cputs(
"DC Voltage source present values:\n\n");
    else cputs(
"DC Current source present values:\n\n");
    cprintf(
"Name: %s       First node %s       Second node %s\n\n\
Value %s",
    par[0],par[1],par[2],par[4]);
    if(isvolt)cputs("       (zero value implies an ammeter)\n\n");
```

```
    else cputs("\n\n");
    window(1,9,80,25);
    switch(menucall("Choose item to change (or to keep all):",6,cvmenu))
    {
        case '1':
            if(newname(*par[0],"source",&par[0]) > 0)change = 1;
            break;

        case '2':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '3':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '4':
            if(isvolt) change |= newdouble(
        "Enter new value for constant potential in volts:",&par[4]);
            else       change |= newdouble(
        "Enter new value for constant current in amperes:",&par[4]);
            break;

        case '5':
            /*make PWL have initial value same as DC*/
            makdefault(&par[5],par[4]);
            tfree(par[4]);
            par[4] = NULL;
            makzero(&par[4]);
            tfree(par[3]);
            par[3] = NULL;
            makdefault(&par[3],"PWL");
            change = 1;
            break;

        case 'H':
        default:
            if(helpav)
             if(!morehelp("cvsdc\n","HELP for DC CVSOURCE"))
                waitforkey(Pressany);
            break;

        case 'R':
        case '6':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;
    }   /*end of switch*/
     }
     else /*must be PWL*/
     {
        int i, j, numpairs = 0;
        char lbuf[81], pbuf[25];
    #ifdef DEBUG
    printf("numpairs = %d   number = %d\n",numpairs,number);
    #endif
        while(par[(++numpairs) + 3]);
        numpairs--;
        if((numpairs & 1)!=0)
        {
             cputs("No value for last time - setting last value to zero\n");
             waitforkey(Pressany);
             clrscr();
             par = trealloc(par, (numpairs + 6)*sizeof(char *));
             par[numpairs + 5] = NULL;
             mekzero(&par[numpairs + 4]);
             number++;
             numpairs++;
             change = 1;
        }
        numpairs >>= 1;
    #ifdef DEBUG
    printf("numpairs = %d   number = %d\n",numpairs,number);
    #endif
        /*sort pairs before displaying*/
    #ifndef DEBUG
        qsort((void *)(par+4), numpairs, 2*sizeof(char *), *parsort);
    #endif
        cprintf(
    "Name: %s       First node %s       Second node %s\n\n",
        par[0],par[1],par[2]);
        if(isvolt)cprintf(
    "Piecewise linear voltage source presently has %d values:\n",numpairs);
        else cprintf(
    "Piecewise linear current source presently has %d values:\n",numpairs);
        for(i = 1; i < 5; i++)
        {
            printf("%6.6s   %6.6s","time","value");
            if((i % 4))cputs("      ");
            else cputs("\n");
        }
        window(1,5,80,12);
        moreopen(8,0);
        moreput(0,0);
        lbuf[0] = 0;
        for(i = j = 1; i <= numpairs; i++, j += 2)
        {
            sprintf(pbuf,"%6.6s   %6.6s",par[j+3],par[j+4]);
            strcat(lbuf,pbuf);
            if((i % 4))strcat(lbuf,"      ");
            else
            {
                strcat(lbuf,"\n");
                if(moreput(lbuf,1))goto morebreak;
                lbuf[0] = 0;
            }
        }
```

```
    if(--i % 4)moreput(lbuf,1);
 morebreak:
    moreclose();
    window(1,13,80,25);
    switch(menucall("Choose item to change (or to keep all):",6,cvmenu2))
    {
        case '1':
            if(newname(*par[0],"source",&par[0]) > 0)change = 1;
            break;

        case '2':
            change |= newnode("first",&par[1],par[2]);
            break;

        case '3':
            change |= newnode("second",&par[2],par[1]);
            break;

        case '4':
        /*here to change set of time-value pairs*/
            change |= pwlvalch(&par,&number);
            break;

        case '5':
            /*make DC have value same as initial PWL*/
            tfree(par[3]);
            par[3] = NULL;
            makdefault(&par[3],"DC");
            tfree(par[4]);
            par[4] = NULL;
            makdefault(&par[4],par[5]);
            tfree(par[5]);
            par[5] = NULL;
            change = 1;
            break;

        case 'H':
        default:
            if(helpav)
             if(!morehelp("cvspwl\n","HELP for PWL CVSOURCE"))
                waitforkey(Pressany);
            break;

        case 'R':
        case '6':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;
    }   /*end of switch*/
     }
            window(1,1,80,25);
            }
        }
```

```
/*readwrit.c - functions for reading a SPICE input file into SPFEDIT
  and for writing a SPICE input file from a circuit in SPFEDIT.
  K.H.Carpenter - 25JUN89
 */

#include "spedit.h"

/*************************************************************************/

/*loadcir - function called from menu to load the current path into
 *the circuit structure.
 */

void loadcir(void)
{
    if(circuitstat == 2)
    {   cprintf("\nCurrent circuit has not been written to file.\n\
Load circuit from file %s anyhow?(y or n):",maincir->path);
        if((getch()|'y')!='y')return;
    }
    clearcir(maincir);
    if(Readcir(maincir))circuitstat = 1;
    else
    {   circuitstat = 0;
        clearcir(maincir);
        maincir->title = addel(NULL);
        maincir->title->elem = (char*)tmalloc(strlen(defaultcirtitle)+1);
        strcpy(maincir->title->elem,defaultcirtitle);
        printf("\nError attempting to load %s.\n\
Current circuit has been cleared.\n\
Press any key to continue",maincir->path);
        getch();
    }
}

/*************************************************************************/

/*Readcir - function to read a SPICE input file into a circuit structure.
 *Call with cir pointing to the circuit structure.
 *Returns 1 on successful read, 0 if unsuccessful.
 *If return is 0 then space may have been alloted for a partial circuit;
 *calling program should issue Clearcir(cir) after such a return.
 */

int Readcir(Circuit *cir)
{
    FILE *fp;
    Element *Title, *Branch, *Option;
    int first;

    if((fp = fopen(cir->path,"r")) == NULL)return(0);
    /*check to see title card does not begin with + or *. */
    if((first = getc(fp)) == EOF){fclose(fp); return(0);}
    if((first == '+')||(first == '*'))ungetc(' ',fp);
    else ungetc(first,fp);

    /*Input the title.  The only reason to allow for the
     *title passed to this function to be non-null is to
     *allow for merging of circuits.
     */
    if((Title = cir->title) == NULL)
    {   Title = addel(NULL);
        cir->title = Title;
    }
    else Title = addel(Title);
    if(!Readnext(fp,Title)){fclose(fp); return(0);}

    /*Loop to input the rest of the file, constructing the
     *linked lists
     */
    while((first = getc(fp))!=EOF)
    {
        ungetc(first,fp);
        if(first == '.')
        {
            if(cir->lop == NULL)
            {   Option = addel(NULL);
                cir->option = Option;
            }
            else Option = addel(cir->lop);
            cir->lop = Option;
            if(!Readnext(fp,Option)){fclose(fp); return(0);}
        }else
        {
            if(cir->lbr == NULL)
            {   Branch = addel(NULL);
                cir->branch = Branch;
            }
            else Branch = addel(cir->lbr);
            cir->lbr = Branch;
            if(!Readnext(fp,Branch)){fclose(fp); return(0);}
        }
    }
    fclose(fp);
    return(1);
}

/*************************************************************************/

/*Readnext - function to read in next data line and its continuations
 *and comments.
 *Call with fp pointing to input file opened as a stream, ele pointing
 *to the object being read into.
 *Returns 1 on success, 0 on error or end-of-file before any data input.
 */

int Readnext(FILE *fp, Element *ele)
{
    char buffer[82], *in, next;
    int len;

    if(fgets(buffer,80,fp) == NULL)return(0);
    if(buffer[0] == '+')return(0);  /*must not begin with continuation*/
    if(buffer[0] == '*')return(0);  /*must not begin with comment*/
```

```
    if((in = strchr(buffer,'\n')) == NULL)return(0);    /*line too long*/
    len = (int)(in - buffer + 2);   /*leave room for NL and NULL*/
    ele->elem = (char*)tmalloc(len);
    strcpy(ele->elem,buffer);

    /*Now check for continuation lines and process any found.*/
    if((next = getc(fp)) == EOF)return(1);
    while(next == '+')
    {
        ungetc(next,fp);
        fgets(buffer,80,fp);
        if((in = strchr(buffer,'\n')) == NULL)return(0);
        len += in - buffer + 1;
        if((ele->elem = (char*)trealloc(ele->elem,len)) == NULL)
            return(0);
        strcat(ele->elem,buffer);
        if((next = getc(fp)) == EOF)return(1);
    }
    /*Now check for comment lines and process any found.*/
    if(next != '*'){ungetc(next,fp); return(1);}
    ungetc(next,fp);
    fgets(buffer,80,fp);
    if((in = strchr(buffer,'\n')) == NULL)return(0);    /*line too long*/
    len = in - buffer + 2;  /*leave room for NL and NULL*/
    ele->comm = (char*)tmalloc(len);
    strcpy(ele->comm,buffer);
    if((next = getc(fp)) == EOF)return(1);
    while(next == '*')
    {
        ungetc(next,fp);
        fgets(buffer,80,fp);
        if((in = strchr(buffer,'\n')) == NULL)return(0);
        len += in - buffer + 1;
        if((ele->comm = (char*)trealloc(ele->comm,len)) == NULL)
            return(0);
        strcat(ele->comm,buffer);
        if((next = getc(fp)) == EOF)return(1);
    }
    ungetc(next,fp);
    return(1);
}

/*************************************************************************/

/*savecir - driver function to call writecir from menu
 */

void savecir(void)
{
    if(maincir == NULL)
    {   cputs("circuit undefined - can't write! (Press any key)");
        getch();
        return;
    }
    if(!access(maincir->path,0))
    {   cprintf("\n%s exists, overwrite?(y or n):",maincir->path);
        if((getch()|'y')!= 'y')return;
    }
    if(writecir(maincir))return;
    cprintf("Error writing %s.  File may not be valid. \n\
    Press any key to continue.",maincir->path);
    getch();
    return;
}

/*************************************************************************/

/*writecir - function to write the information in the circuit
 *structure to the file given in the path entry of the structure.
 *Call with pointer to circuit.  Returns 1 on successful write,
 *0 on some failure.
 */

int fps(char *string, FILE *fp)
{
    if(string == NULL)return(0);
    return(fputs(string,fp));
}

int writecir(Circuit *cir)
{
    FILE *fp;
    Element *out;
    if((fp = fopen(cir->path,"w")) == NULL)return(0);
    if(fps(cir->title->elem,fp)==EOF){fclose(fp); return(0);}
    if(fps(cir->title->comm,fp)==EOF){fclose(fp); return(0);}
/*In case the circuit does not have an .END card, force one into it
 *now.
 */
    forceend(cir);
/*Since .END must be the last card in the SPICE input file,
 *the options must be written out last, after the elements.
 */
    for(out = cir->branch; out != NULL; out = out->next)
    {
        if(fps(out->elem,fp)==EOF){fclose(fp); return(0);}
        if(fps(out->comm,fp)==EOF){fclose(fp); return(0);}
    }
    for(out = cir->option; out != NULL; out = out->next)
    {
        if(fps(out->elem,fp)==EOF){fclose(fp); return(0);}
        if(fps(out->comm,fp)==EOF){fclose(fp); return(0);}
    }
    fclose(fp);
    circuitstat = 1;
    return(1);
}
```

```
/*elutil.c - utility functions for element handling for SPFEDIT
   K.H.Carpenter - 22MAY89
 */

#include "spedit.h"

/*****************************************************************/

/*Find - function to return pointer to particular element or command.
 *Call with name set to first word of card, start to first element
 *in linked list.  Returns pointer to item in list having that name.
 *Name must match exactly to first location where both strings have
 *a blank or other SPICE input separator.
 */

Element *Find(char *name, Element *start)
{
    Element * item;
    for(item = start; item != NULL; item = item->next)
    {
        if(ncomp(name,item->elem))return(item);
    }
    return(NULL);
}

/*****************************************************************/

/*ncomp - function to compare case-folded, SPICE-separator delimited words.
 *Call with pointers to words as arguments.  Returns 0 if different,
 *returns 1 if equal.
 */

int ncomp(char *a, char *b)
{
    for(;;a++, b++)
    {
        if(issep(*a) && issep(*b)) return(1);
        if((*a | 0x20) != (*b | 0x20))return(0);
    }
}

/*****************************************************************/

/*addel - function to add a new element entry in a linked list.
 *Call with pointer to place in list to insert it.  Returns pointer
 *to the new entry.  If argument is null, an new list is begun.
 *(Space for elem and comm items in structure
 *must be allocated by calling routine when used.)
 */

Element *addel(Element *place)
{
    Element *new;

    new = (Element *)tmalloc(sizeof(Element));
    new->elem = new->comm = NULL;
    new->last = place;
    if(place == NULL)new->next = NULL;
    else
    {   new->next = place->next;
        place->next = new;
        if(new->next != NULL)new->next->last = new;
    }
    return(new);
}

/*****************************************************************/

/*upel - function to move an element to the next higher location
 *in the linked list.  If the element is already the head of the
 *list, no action is taken.  Call with a pointer the the element
 *to move up.
 */

void upel(Element *el)
{
    Element *tmp;
    if(el->last == NULL)return;
    tmp = el->next;
    el->next = el->last;
    el->last = el->next->last;
    el->next->next = tmp;
    if(el->last != NULL)el->last->next = el;
    el->next->last = el;
    return;
}

/*****************************************************************/

/*delel - function to delete an element entry in a linked list
 *and to free the storage used in it.
 */

void delel(Element *drop)
{
    if(drop == NULL)return;
    tfree(drop->elem);
    tfree(drop->comm);
    if(drop->last != NULL)drop->last->next = drop->next;
    if(drop->next != NULL)drop->next->last = drop->last;
    tfree(drop);
}

/*****************************************************************/

/*creatcir - function to allocate a circuit structure.
 *Call with pointer to path string.  Returns pointer to
 *the circuit structure.  Path is initialized to input
 *argument.  Other items in structure are initialized to NULL.
 */

Circuit *creatcir(char *inpath)
{
```

```
    Circuit *cir;
    cir = (Circuit *)tmalloc(sizeof(Circuit));
    cir->path = inpath;
    cir->title = cir->branch = cir->lbr = cir->option = cir->lop = NULL;
    return(cir);
}

/*****************************************************************/

/*clearcir - function to remove all the components of a circuit
 *structure, leaving the pointers in the structure NULL's
 */

void clearcir(Circuit *cir)
{
    Element *work;
    for(work = cir->title; work !=NULL; work = work->next)
        delel(work);
    cir->title = NULL;
    for(work = cir->option; work!=NULL; work = work->next)
        delel(work);
    cir->option = cir->lop = NULL;
    for(work = cir->branch; work!=NULL; work = work->next)
        delel(work);
    cir->branch = cir->lbr = NULL;
}

/*****************************************************************/

/*delcir - function to delete a circuit structure, freeing all its
 *storage.
 */

void delcir(Circuit *cir)
{
    clearcir(cir);
    tfree(cir->path);
    tfree(cir);
}

/*****************************************************************/

/*newtitle - function to change title in circuit file
 */

void newtitle(void)
{
    char buf[81];
    clrscr();
    cprintf("\nTitle is now:\n%s\n\
Enter new title(default - keep same title):\n",maincir->title->elem);
    fgets(buf,81,stdin);
    if(buf[0]&&(buf[0]!='\n'))
    {   if(strchr(buf,'\n')==NULL)
        {   cputs("Title must not be over 80 characters long.\
  Truncated.");
            sleep(1);
            buf[79] = '\n'; buf[80] = '\0';
            fflush(stdin);
        }
        tfree(maincir->title->elem);
        maincir->title->elem = (char*)tmalloc(strlen(buf)+1);
        strcpy(maincir->title->elem,buf);
        circuitstat = 2;    /*flags a modification*/
    }
}

/*****************************************************************/

/*renamecir - function to change path to circuit file
 */

void renamecir(void)
{
    char *nlch, buf[81];
    clrscr();
    cprintf("\nCircuit with title:\n%s\nhas path to file:\n%s\n\
Enter new path(default - keep same path):\n",maincir->title->elem,
        maincir->path);
    fgets(buf,81,stdin);
    if(buf[0]&&(buf[0]!='\n'))
    {   if((nlch = strchr(buf,'\n'))==NULL)
        {   cputs("Path must be less than 80 characters long.\
\nRetaining old path.");
            fflush(stdin);
            return;
        }
        *nlch = '\0';   /*Must not have path end with newline*/
        tfree(maincir->path);
        maincir->path = (char*)tmalloc(strlen(buf)+1);
        strcpy(maincir->path,buf);
/*Split path into parts to make menu title*/
        fnsplit(maincir->path,cirdriv,cirdir,cirname,cirext);
        tfree(maintitle);
        maintitle = (char*)tmalloc(strlen(maintitl1)+strlen(cirname)+1);
        strcpy(maintitle,maintitl1);
        strcat(maintitle,cirname);

    }
}

/*****************************************************************/

/*uniqnam - function to determine if name for an element or command
 *is unique.
 *Call with pointer to name to test, 1 for element, 2 for command.
 *Returns 0 if unique, 1 if not.
 */

int uniqnam(char *inname,int n)
{
    char *buf;
    Element *el;
    buf = (char*)tmalloc(strlen(inname)+2);
```

```
    if(n == 1)  /*case for element*/
    {
        strcpy(buf,inname);
        strcat(buf," ");
        el = Find(buf,maincir->branch);
    }
    else    /*case for command*/
    {
        if(inname[0] == '.')buf[0]=0;
        else {buf[0] = '.'; buf[1]=0;}
        strcat(buf,inname);
        strcat(buf," ");
        el = Find(buf,maincir->option);
    }
    tfree(buf);
    if(el == NULL)return(0);
    return(1);
}

/*****************************************************************/

/*copyelq - function to copy values from one element into another
 *if query is answered y or Y.
 *Call with pointer to element to be copied into, string giving type
 *of element (for prompt), character that must begin the element name.
 *Sets pointer in element to new string containing copied values.
 */

void copyelq(Element *newel, char *type, char first)
{
    Element *el;
    char **new, **old;
    int i, numold, numnew;
    cprintf("\nCopy values for this %s from another %s? ",type,type);
    if((waitforkey("(y or n):") | 0x20) != 'y')return;
    if((el = Findlist(type,first,2)) == NULL)return;
    old = spparse(el->elem, &numold);
    new = spparse(newel->elem, &numnew);
    tfree(*old);
    *old = *new;
    tfree(newel->elem);
    newel->elem = spunparse(old);
    tfree(*old);
    for(i=1; i < numold; i++)tfree(old[i]);
    for(i=1; i < numnew; i++)tfree(new[i]);
}
```

```
/*modcir.c - K.H.Carpenter - 15AUG89
  Module for entering and changing circuit elements and commands.
 */

#include "spedit.h"

/*prototypes*/
void modifycir(void);
void addcomm(void);
void addele(void);
void changele(void);
void deleteel(void);
void resistor(Element *el);
void capacitor(Element *el);
void inductor(Element *el);
void mutual(Element *el);
void tline(Element *el);
void fuse(Element *el);
void vcvs(Element *el);
void cccs(Element *el);
void ccvs(Element *el);
void cvsource(Element *el);
void defaultel(Element *el);
int trans(void);
int printcard(void);
int plotcard(void);
int width(void);
int option(void);
Element *addelloc(void);

/*menus*/

char *modmen[7] = {
    "Add an element",
    "Change an element",
    "Delete an element",
    "Add or modify a command",
    "Change title line",
    "Change general comments",
    "Return to main menu"
    };

char *addelmen1[3] = {
    "Add element before first element in file",
    "Add element after last element in file",
    "Add element after another element"
    };

char *addcomnd1[6] = {
    "TRAN    (analysis to perform)",
    "PRINT   (values to output)",
    "PLOT    (printer plot output)",
    "WIDTH   (input/output line lengths)",
    "OPTION  (SPICE numerical method)",
    };

char *addtran[5] = {
    "Change output time step",
    "Change analysis stop time",
    "Change output start time",
    "Change analysis maximum time step size",
    "Save present values and return"
    };

char *addelmen2[8] = {
    "Resistor",
    "Capacitor",
    "Inductor",
#if 0
    "Mutual coupling between two linear inductors",
#endif
    "Fuse",
    "Independent voltage source (or ammeter)",
    "Independent current source",
    "Lossless transmission line",
    "Other SPICE element"
    };

/*global data*/

    char *inname;

/*************************************************************/

/*modcir*/

void modifycir(void)
{

/*loop on menu*/

    while(1)switch(menucall("Modify Menu",7,modmen)){

        case '1':   addele();
                    break;

        case '2':   changele();
                    break;

        case '3':   deleteel();
                    break;

        case '4':   addcomm();
                    break;

        case '5':   newtitle();
                    break;

        case '6':   chcomments();
                    break;

        case '7':
        case 'R':
                    return;
```

```
        case 'H':
        default:    if(helpav)
                  if(!morehelp("modify\n","HELP for MODIFY MENU"))
                    waitforkey(Pressany);

    }
}

/*********************************************************/

/*Findlist - function to find a circuit element by name, displaying
 *a list to aid in choosing.
 *Call with (type: a string giving the class of element (used in prompt))
 *(first: a character that must be matched by the first character of the
 *name - if null, then any first character is all right)
 *(dflag: integer giving part of circuit to display and search -
 * = 1 for commands, = 2 for branches).
 *Returns pointer to the element structure having that name for its elem.
 */

Element *Findlist(char *type, char first, int dflag)
{
    Element *el;
    char cname[81], inname[81], *cr;
    clrscr();
    while(1){
        cprintf("Enter name of %s to select (= for list, 0 to abort):",
            type);
        fgets(inname,81,stdin);
        fflush(stdin);
        if(inname[0] == '=')
        {
            clrscr();
            displaycir(dflag, maincir, 0, 20);
            continue;
        }
        else if(inname[0] == '0' || inname[0] == '\n')return(NULL);
        if(dflag == 1)
        {
            if(inname[0] == '.')strcpy(cname,inname);
            else
            {   cname[0] = '.';
                strncpy(cname+1,inname,80);
            }
        }
        else strcpy(cname,inname);
        if((cr = strchr(cname,'\n'))!=NULL)*cr = ' ';
        if(first && (cname[0] != first))
        {
            cprintf("%s must begin with \"%c\"\n",type,first);
            continue;
        }
        if(dflag == 1)el = Find(cname,maincir->option);
        else el = Find(cname,maincir->branch);
        if(el == NULL)
        {
            cprintf("\n%s %sdoes not exist.\n",type,cname);
            continue;
        }
        return(el);
    }
}

/*********************************************************/

/*change element*/

void changele(void)
{
    while(1)
    {
        Element *el;
        if((el = Findlist("element",0,2))==NULL)return;
        switch(*el->elem|0x20)
        {
            case 'r':
                    resistor(el);
                    break;
            case 'c':
                    capacitor(el);
                    break;
            case 'l':
                    inductor(el);
                    break;
            case 'k':
                    mutual(el);
                    break;
            case 't':
                    tline(el);
                    break;
            case 'g':
                    fuse(el);
                    break;
            case 'e':
                    vcvs(el);
                    break;
            case 'f':
                    cccs(el);
                    break;
            case 'h':
                    ccvs(el);
                    break;
            case 'i':
            case 'v':
                    cvsource(el);
                    break;

            default:
                    defaultel(el);

        }
    }
}
```

```
/*********************************************************/

/*delete element*/

void deleteel(void)
{
    Element *el;
    if((el = Findlist("element",0,2))==NULL)return;
/*If element is either the first or the last in the linked
 *list of the circuit, then the pointer in the circuit
 *structure must be changed.
 */
    if(el == maincir->branch)maincir->branch = el->next;
    if(el == maincir->lbr)maincir->lbr = el->last;
    delel(el);
}

/*********************************************************/

/*chcomments - routine to allow editing of comment lines
 *not implemented in this version
 */

void chcomments()
{
    clrscr();
    puts("Comment lines may be placed in a SPICE input file after\
 the title\n(the first line) or after any element or command line.\n\
All comment lines must begin with an asterisk (*) in the first column.\n\
To add or change comments, save the circuit to a file and modify the file\n\
with a text editor.\n\n(Press any key to continue.)");
    getch();
}

/*********************************************************/

/*addele - routine to add new element to circuit
 */

void addele(void)
{
    Element *new;
    int ab;
    char cs;
    unsigned len;
    switch(menucall("Choose location in file for element:",3,addelmen1))
    {

        case '1':   new = addel(maincir->branch);
                    upel(new);
                    maincir->branch = new;
                    if(!maincir->lbr)maincir->lbr = new;
                    break;

        case '2':   new = addel(maincir->lbr);
                    maincir->lbr = new;
                    if(!maincir->branch)maincir->branch = new;
                    break;

        case '3':   new = addelloc();
                    break;

        case 'R':   return;

        case 'H':
        default:    if(helpav)
                      if(!morehelp("addloc\n","HELP for ADD LOCATION MENU"))
                        waitforkey(Pressany);
                    return;
    }
    if(new == NULL)return;
    if(maincir->branch == NULL)maincir->branch = maincir->lbr = new;
    /*here put code to get values into element "new" -
    get type of element from a menu, then get name and force it
    to begin with the correct letter, then call the modify routine.
    If only <CR> entered as name, abort adding element and free its
    storage.
    */
    switch(menucall("Choose type of element:",8,addelmen2))
    {
        case '1':   /*resistor*/
            while(!(ab=newname('R',"resistor",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "resistor", 'r');
            resistor(new);
            break;

        case '2':   /*capacitor*/
            while(!(ab=newname('C',"capacitor",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "capacitor", 'c');
            capacitor(new);
            break;

        case '3':   /*inductor*/
            while(!(ab=newname('L',"inductor",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "inductor", 'l');
            inductor(new);
            break;

#if 0
        case '4':   /*mutual coupling*/
            defaultel(new);
```

```
            break;
#endif

        case '4':   /*fuse*/
            while(!(ab=newname('G',"fuse",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "fuse", 'g');
            fuse(new);
            break;

        case '5':   /*independent voltage source*/
            while(!(ab=newname('V',"voltage source",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "voltage source", 'v');
            cvsource(new);
            break;

        case '6':   /*independent current source*/
            while(!(ab=newname('I',"current source",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "current source", 'i');
            cvsource(new);
            break;

        case '7':   /*transmission line*/
            while(!(ab=newname('T',"transmission line",&(new->elem))));
            if(ab == -1)break;
            len = strlen(new->elem);
            new->elem = trealloc(new->elem,len + 2);
            strcat(new->elem,"\n");
            copyelq(new, "transmission line", 't');
            tline(new);
            break;

        case '8':   /*other element*/

            defaultel(new);
            putch('\n');

        case 'R':
            return;

        case 'H':
        default:    if(helpav)
                  if(!morehelp("addtype\n","HELP for SELECT ELEMENT MENU"))
                    waitforkey(Pressany);
    }
    if(ab == -1)
    {
        if(new == maincir->branch)maincir->branch = new->next;
        if(new == maincir->lbr)maincir->lbr = new->last;
        delel(new);
    }
}

/*********************************************************/

/*addelloc - function to return the new element, located as
 *desired by dialog.
 */

Element *addelloc(void)
{
    char inname[81];
    clrscr();
    while(1){
        Element *el, *tmp;
        cputs("Enter name of element to follow (= for list, 0 to abort):");
        fgets(inname,81,stdin);
        fflush(stdin);
        if(inname[0] == '=')
        {
            displaycir(2, maincir, 0, 20);
        }else if(inname[0] == '0' || inname[0] == ' ')return(NULL);
        else if((el=Find(inname,maincir->branch)) == NULL)
        {
            cprintf("\nElement %sdoes not exist.\n",inname);
            continue;
        }else
        {
            tmp = addel(el);
            if(maincir->lbr == el) maincir->lbr = tmp;
            return(tmp);
        }
    }
}

/*********************************************************/

/*addcomm - routine to add new command to circuit or modify existing one.
 */

void addcomm(void)
{
    int change;
/*Always add commands at the end of the circuit, but before the .END command
 *which must always be present.
 */
    change = forceend(maincir);
    switch(menucall("Choose command to add or modify:",5,addcomnd1))
    {
        case '1':
            change |= trans();
            break;
```

```
        case '2':
            change |= printcard();
            break;
        case '3':
            change |= plotcard();
            break;
        case '4':
            change |= width();
            break;
        case '5':
            change |= option();
            break;
        case 'H':
        default:    if(helpav)
                  if(!morehelp("addcom\n","HELP for COMMANDS MENU"))
                    waitforkey(Pressany);
        case 'R':
    }
    if(change)circuitstat = 2;
}



/*********************************************************/

/*forceend - function to force the last element card in Spice circuit
 *to be the .END card.  Call with a pointer to the circuit.
 *Returns 1 if the card had to be added, 0 if it was already present
 *(and at the correct location).
 */

int forceend(Circuit *cir)
{
    Element *new;
    new = Find(".END",cir->option);
    if(new != NULL)
    {
        if(new == cir->lop)return(0);
        if(new == cir->option)cir->option = new->next;
        delel(new);
    }
    new = addel(cir->lop);
    cir->lop = new;
    if(cir->option == NULL)cir->option = new;
    makdefault(&(new->elem),".END\n");
    return(1);
}

/*********************************************************/

/*trans - function to add or modify the transient analysis input
 *card for SPICE.
 */

int trans(void)
{
    int change = 0;
    char **par;
    int number, i;
    Element *el;
    if((el = Find(".TRAN",maincir->option))==NULL)
    {
        el = addel(maincir->lop);
        upel(el);   /*so .END will remain the last card*/
        if(maincir->lop == maincir->option) maincir->option = el;
        makdefault(&(el->elem),".TRAN\n");
        change = 1;
    }
    clrscr();
    par = spparse(el->elem, &number);
    if(number < 6)
    {
        par = trealloc(par, 7*sizeof(char *));
        for(i = number + 1; i < 7 ; i++)par[i] = NULL;
        number = 6;
    }
    /*Force values to defaults*/
    makdefault(&par[1],"0.01");
    makdefault(&par[2],"1");
    /*In case tstart or tmax were not given but UIC was:*/
    if(!strcmp(par[3],"UIC")){tfree(par[3]); par[3] = NULL;}
    if(!strcmp(par[4],"UIC")){tfree(par[4]); par[4] = NULL;}
    makzero(&par[3]);
    makdefault(&par[4],"0.01");
    makdefault(&par[5],"UIC");
/*Now loop on menu for changing values:*/
  while(1)
  {
    clrscr();
    cprintf("TRAN command present values:\nTSTEP: %s      TSTOP: %s\n\n\
TSTART:  %s     TMAX:  %s\n\n",par[1],par[2],par[3],par[4]);
    window(1,9,80,25);
    switch(menucall("Choose action:",5,addtran))
    {
        case '1':
    change |= newdouble("Enter new step size (for output), in seconds:"
                ,&par[1]);
            break;

        case '2':
    change |= newdouble("Enter new stop time for analysis, in seconds:"
                ,&par[2]);
            break;

        case '3':
    change |= newdouble("Enter new start time (for output), in seconds:"
                ,&par[3]);
            break;

        case '4':
    change |= newdouble("Enter new maximum step size for analysis, \
in seconds.\n\
(Too large a value will cause numerical problems - try the lesser of\n\
 (TSTOP-TSTART)/100 and TSTEP as a first estimate.):"
```

```
            ,&par[4]);
            break;

        case 'R':
        case '5':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return(change);

        case 'H':
        default:
            if(helpav)
              if(!morehelp("trans\n",
                "HELP for TRANSIENT ANALYSIS MENU"))
                    waitforkey(Pressany);
                break;

    }   /*end of switch*/
    window(1,1,80,25);
   }
}
/*********************************************************/

/*printcard - function to modify .PRINT card in SPICE input file.*/

int printcard(void)
{
    int change = 0;
    char buf[63];
    Element *el;
    if((el = Find(".PRINT",maincir->option))==NULL)
    {
        el = addel(maincir->lop);
        upel(el);   /*so .END will remain the last card*/
        if(maincir->lop == maincir->option) maincir->option = el;
        makdefault(&(el->elem),".PRINT TRAN\n");
        change = 1;
    }
    clrscr();
cputs("PRINT command lists the variables (node voltages and currents in\n\
through independent voltage sources) to be placed in the output file as\n\
a set of numerical values.  See the SPICE User's Guide for details.\n\n");
    printf("Current contents of PRINT command:\n\n%s\n\n\
Enter a line to replace all the contents of the command, after TRAN,\n\
or enter <cr> to keep the present contents,\n\
or enter -<cr> (a single minus) to delete the command:\n\n.PRINT TRAN"
        ,el->elem);
    fgets(buf,63,stdin);
    if((buf[0] == '-')||(buf[0] == '\n' &&
        !strcmp(el->elem,".PRINT TRAN\n")))
    {
        if(el == maincir->option)maincir->option = el->next;
        delel(el);
        change = change ? 0 : 1;
    }
    else if(buf[0] == '\n')change = 0;
    else
    {
        tfree(el->elem);
        el->elem = tmalloc(strlen(buf) + 9);
        strcpy(el->elem,".PRINT TRAN");
        strcat(el->elem,buf);
        while(!strchr(buf,'\n'))
        {
            fgets(buf,63,stdin);
        el->elem = trealloc(el->elem,strlen(el->elem)+strlen(buf)+2);
            strcat(el->elem,"+");
            strcat(el->elem,buf);
        }
        change = 1;
    }
    fflush(stdin);
    return(change);
}

/*********************************************************/

/*plotcard - function to modify .PLOT card in SPICE input file.*/

int plotcard(void)
{
    int change = 0;
    char buf[63];
    Element *el;
    if((el = Find(".PLOT",maincir->option))==NULL)
    {
        el = addel(maincir->lop);
        upel(el);   /*so .END will remain the last card*/
        if(maincir->lop == maincir->option) maincir->option = el;
        makdefault(&(el->elem),".PLOT TRAN\n");
        change = 1;
    }
    clrscr();
cputs("PLOT command lists the variables (node voltages and currents in\n\
through independent voltage sources) to be placed in the output file in\n\
a \"printer plot\".  See the SPICE User's Guide for details.\n\n");
    printf("Current contents of PLOT command:\n\n%s\n\n\
Enter a line to replace all the contents of the command, after TRAN,\n\
or enter <cr> to keep the present contents,\n\
or enter -<cr> (a single minus) to delete the command:\n\n.PLOT TRAN"
        ,el->elem);
    fgets(buf,63,stdin);
    if((buf[0] == '-')||(buf[0] == '\n' &&
        !strcmp(el->elem,".PLOT TRAN\n")))
    {
        if(el == maincir->option)maincir->option = el->next;
        delel(el);
        change = change ? 0 : 1;
```

```
    }
    else if(buf[0] == '\n')change = 0;
    else
    {
        tfree(el->elem);
        el->elem = tmalloc(strlen(buf) + 9);
        strcpy(el->elem,".PLOT TRAN");
        strcat(el->elem,buf);
        while(!strchr(buf,'\n'))
        {
            fgets(buf,63,stdin);
        el->elem = trealloc(el->elem,strlen(el->elem)+strlen(buf)+2);
            strcat(el->elem,"+");
            strcat(el->elem,buf);
        }
        change = 1;
    }
    fflush(stdin);
    return(change);
}

/*********************************************************/

/*width - function to modify .WIDTH card in SPICE input file.*/

int width(void)
{
    int change = 0;
    char **par;
    int number;
    Element *el;
    if((el = Find(".WIDTH",maincir->option))==NULL)
    {
        el = addel(maincir->lop);
        upel(el);   /*so .END will remain the last card*/
        if(maincir->lop == maincir->option) maincir->option = el;
        makdefault(&(el->elem),".WIDTH\n");
        change = 1;
    }
    clrscr();
    par = spparse(el->elem, &number);
    if(number < 5)
    {
        int i;
        par = trealloc(par, 6*sizeof(char *));
        for(i = number + 1; i < 6 ; i++)par[i] = NULL;
        number = 5;
    }
    /*Force values to defaults*/
    if(strcmp(par[1],"IN"))if(par[1]){tfree(par[1]); par[1] = NULL;}
    if(strcmp(par[2],"80"))if(par[2]){tfree(par[2]); par[2] = NULL;}
    if(strcmp(par[3],"OUT"))if(par[3]){tfree(par[3]); par[3] = NULL;}
    if(strcmp(par[4],"80") && strcmp(par[4],"133"))if(par[1])
        {tfree(par[1]); par[1] = NULL;}
    change |= makdefault(&par[1],"IN");
    change |= makdefault(&par[2],"80");
    change |= makdefault(&par[3],"OUT");
    change |= makdefault(&par[4],"80");
    printf(
"Printer width must be 80 columns or 133 columns.\n\
Present setting is %s columns.  Change to the other?(y or n):",par[4]);
    if(getche() | 0x20 == 'y')
    {
        int t;
        t = strcmp(par[4],"80");
        tfree(par[4]);
        par[4] = NULL;
        if(t)makdefault(&par[4],"133");
        else makdefault(&par[4],"80");
        change = 1;
    }
    if(change)
    {
    tfree(el->elem);
    el->elem = spunparse(par);
    }
    freeparret(par, number);
    return(change);
}


/*********************************************************/

/*option - function to modify .OPTION card in SPICE input file.*/

int option(void)
{
    int change = 0;
    char buf[63];
    Element *el;
    if((el = Find(".OPTION",maincir->option))==NULL)
    {
        el = addel(maincir->lop);
        upel(el);   /*so .END will remain the last card*/
        if(maincir->lop == maincir->option) maincir->option = el;
        makdefault(&(el->elem),".OPTION\n");
        change = 1;
    }
    clrscr();
cputs("OPTION command allows setting many parameters associated with SPICE.\n\
See the SPICE User's Guide for details.  Usually one should omit this\n\
command and allow the default values to be used.\n\n");
    printf("Current contents of OPTION command:\n\n%s\n\n\
Enter a line to replace all the contents of the command, after its name,\n\
or enter <cr> to keep the present contents,\n\
or enter -<cr> (a single minus) to delete the command:\n\n.OPTION ",el->elem);
    fgets(buf,63,stdin);
    if((buf[0] == '-')||(buf[0] == '\n' && !strcmp(el->elem,".OPTION\n")))
    {
        if(el == maincir->option)maincir->option = el->next;
        delel(el);
        change = change ? 0 : 1;
    }
    else if(buf[0] == '\n')change = 0;
```

```
    else
    {
        tfree(el->elem);
        el->elem = tmalloc(strlen(buf) + 9);
        strcpy(el->elem,".OPTION ");
        strcat(el->elem,buf);
        while(!strchr(buf,'\n'))
        {
            fgets(buf,63,stdin);
        el->elem = trealloc(el->elem,strlen(el->elem)+strlen(buf)+2);
            strcat(el->elem,"+");
            strcat(el->elem,buf);
        }
        change = 1;
    }
    fflush(stdin);
    return(change);
}

/*********************************************************/
/*stubs*/
void mutual(Element *el){defaultel(el);}
void vcvs(Element *el){defaultel(el);}
void cccs(Element *el){defaultel(el);}
void ccvs(Element *el){defaultel(el);}
```

```
/*tcmore.c - K.H.Carpenter - 29APR89
 *Files to implement a Unix-like "more" of text to screen under TurboC.
 */

/*********************************************************************/
/*defs and globals*/

#include <conio.h>
#include <string.h>
#define NULL 0
#define Morew oscr.screenwidth
#define TATT 0x70
#define TBUFL 133
struct text_info oscr;
int Mcount, Moreh, Morebot, Morend;
int morewait(int flag);

/*********************************************************************/

/*moreopen - function to set the window for scrolled text.
 *Call with lines of window height to use, pointer to a one line
 *title to fit in first line (NULL if no title wanted).
 */

void moreopen(int lines, char *title)
{
    gettextinfo(&oscr); /*get and save current window info*/
    if(title != NULL)lines++;
    Morend=Moreh = lines < oscr.screenheight ? lines : oscr.screenheight;
    Morebot = Moreh + oscr.wintop - 1;
    window(oscr.winleft,oscr.wintop,oscr.winright,Morebot);
/*Next write title along top line of window, if given.*/
    if(title != NULL)
    {   unsigned len;
        Moreh--;
        textattr(TATT);
        if((len = strlen(title)) > Morew)
        {   int i;
            for(i=0; i <= Morew; i++)putch(title[i]);
        }
        else
        {   int i,j;
            for(i=0,j=0; i <= Morew - len; i+=2,j++)putch(' ');
            for(i=0; title[i]!='\n'&&title[i]!='\0';i++,j++)putch(title[i]);
            while(j++ < Morew)putch(' ');
        }
        textattr(oscr.attribute);
        window(oscr.winleft,oscr.wintop+1,oscr.winright,Morebot);
    }
}

/*********************************************************************/

/*moreclose - function to close the "more" window*/

void moreclose(void)
{
    window(oscr.winleft,oscr.wintop,oscr.winright,oscr.winbottom);
    gotoxy(oscr.winleft,Morend);
/*  clreol();*/
}

/*********************************************************************/

/*moreput - function to add lines to the screen.  Call with
 *pointer to text to write, and with init set to 0 if this is
 *the first call, or if screen is to be cleared before writing,
 *and with init set to 1 to continue writing after last text,
 *and with init set to -1 to continue writing after last text, but
 *to clear screen after each "more" prompt.
 *Returns 1 if "q" or "Q" was typed to "more" prompt, zero otherwise.
 */

int moreput(char *text, int init)
{
    int i, col;
    if(!init){Mcount = 1; clrscr();}
    if(text == NULL)return(0);
    for(i=0,col=0; text[i]!='\0'; i++,col++)
    {
        if((text[i] == '\n')||(col > Morew))
        {
            putch('\r'); putch('\n'); Mcount++; col = 0;
            if(Mcount == Moreh){if(morewait(init))return(1);}
        }
        if(text[i] == '\n')if(text[++i] == '\0')return(0);
        putch(text[i]);
    }
    return(0);
}

int morewait(int flag)
{
    Mcount = 1;
    textattr(TATT);
    cputs("  --more--  ");
    putch('\r');
    textattr(oscr.attribute);
    if(getch() == ('q'|'Q')){clreol(); return(1);}
    if(flag < 0)clrscr();
    clreol();
    return(0);
}
```

```
/*display.c - K.H.Carpenter - 02MAY89
 *Functions to display circuits and elements of
 *circuits to the screen.
 */

#include "spedit.h"

/*************************************************************/

/*Displaycir - function called from menu to drive displaycir.
 */

void Displaycir(void)
{
    displaycir(0, maincir, -1, MAXSCRHT);
    cputs("Press any key to continue:");
    getch();
}

/*************************************************************/

/*displaycir - function to display a circuit on the screen.
 *First argument gives part of circuit to display.
 *part=0 for entire circuit.  Then
 *title goes to window title, comments on title are displayed
 *first, then elements with comments, then commands with comments.
 *part=1 for commands with comments only;
 *part=2 for elements with comments only.
 *Call with next arguments: pointer to circuit, flag for scroll (negative for
 *clear screen before each new page, scroll otherwise),
 *height for screen window (in lines - starts at top).
 *(Height must be >2.  Set to 2 if not.)
 */

void displaycir(int part, Circuit *cir, int flag, int height)
{
    Element *next; int i;
    if(!flag)flag = 1;
    if(height < 2)height = 2;
    moreopen(height,cir->title->elem);
    if(!part)moreput(cir->title->comm,0);
    else moreput(0,0);  /*needed to set initial count to zero*/
    if((!part)||(part==2))
        for(next = cir->branch; next != NULL; next = next->next)
    {
        if(moreput(next->elem,flag)){moreclose(); return;}
        if(moreput(next->comm,flag)){moreclose(); return;}
    }
    if((part-1)<=0)for(next = cir->option; next != NULL; next = next->next)
    {
        if(moreput(next->elem,flag)){moreclose(); return;}
        if(moreput(next->comm,flag)){moreclose(); return;}
    }
    moreclose();
}
```

```
/*creatcir.c - routine for starting a new circuit
  K.H.Carpenter - 08MAY89
 */

#include "spedit.h"

/*************************************************************/

/*Creatcir - called from menu to create a new circuit
 */

void Creatcir(void)
{
    clrscr();
    if(circuitstat == 2)
    {   cprintf("Circuit with title:\n%shas been modified since saved\
 to a file.  Discard it? (y or n):",maincir->title->elem);
        if(getch() != ('y'|'Y'))return;
    }
    clearcir(maincir);
    maincir->title = addel(NULL);
    maincir->title->elem = (char*)tmalloc(strlen(defaultcirtitle)+1);
    strcpy(maincir->title->elem,defaultcirtitle);
    newtitle();
    renamecir();
    cputs("Circuit has been created.  Add elements from the Modify menu.\
\n(Press any key to continue.)");
    getch();
}
```

```
/*morehelp.c - KHC 22MAY89
 *file to support help messages from external files in a program.
 *"more" routine is used to allow multiple pages of help on one
 *topic, and help file is searched for in the environment.
 */

/*function prototypes*/

void *tmalloc(unsigned length);
void moreopen(int lines, char *title);
void moreclose(void);
int moreput(char *text, int init);
    /*A subtle bug was introduced by not having this prototype for
     *moreput included.  The call below with arguments (0,0) was
     *interpreted by the compiler as requiring two integers to be
     *pushed onto the stack, when it should be a pointer and one
     *integer.  As pointers are 4 bytes while integers are 2 in
     *TurboC large model, this caused the value of init received
     *to be non-zero, and so the initial count for the help screen
     *was not reset.
     */

/*********************************************************************/

/*help - function to read help segments from a file and send to screen.
 *This routine was taken from the distribution of the "gnutex"
 *plotting package - the original author of this file is
 *unknown, but the plotting package makefile contains the
 *following attribution:
# Makefile for gnutex
#   A plotting program supporting LaTeX output
#   David Kotz
#   Duke University
#   dfk@cs.duke.edu
 *Two minor changes were required to (1) allow compile under TurboCv1.5
 *and (2) fix a bug if alternate files not found.
 *A major change was introduced to call the "more" routines in file
 *tcmore.c to do the screen writing to thus allow multiple pages of help
 *for a single topic.  To see the extent of the changes from the
 *Kotz-distributed "help", do a "diff" with his version (or look near
 *the comments containing "KHC").
 */

#include <stdio.h>
#include <errno.h>
#include <string.h>
#define SAME    0   /* for strcmp() */

/*#include "help.h"*/   /* values passed back */
/*included in-line here - KHC*/
/* Exit status returned by help() */
#define H_FOUND     0   /* found the keyword */
#define H_NOTFOUND  1   /* didn't find the keyword */
#define H_ERROR     (-1)    /* didn't find the help file */
/* help -- help subsystem that understands defined keywords
**
** Looks for the desired keyword in the help file at runtime, so you
** can give extra help or supply local customizations by merely editing
** the help file.
**
** The original (single-file) idea and algorithm is by John D. Johnson,
** Hewlett-Packard Company.  Thanx and a tip of the Hatlo hat!
**
** The help file looks like (the question marks are really in column 1):
**
**  ?topic
**  This line is printed when the user wants help on "topic".
**  ?keyword
**  ?Keyword
**  ?KEYWORD
**  These lines will be printed on the screen if the user wanted
**  help on "keyword", "Keyword", or "KEYWORD".  No casefolding is
**  done on the keywords.
**  ?subject
**  ?alias
**  This line is printed for help on "subject" and "alias".
**  ?
**  ??
**  Since there is a null keyword for this line, this section
**  is printed when the user wants general help (when a help
**  keyword isn't given).  A command summary is usually here.
**  Notice that the null keyword is equivalent to a "?" keyword
**  here, because of the '?' and '??' topic lines above.
**  ?last-subject
**  Note that help sections are terminated by the start of the next
**  '?' entry or by EOF.  So you can't have a leading '?' on a line
**  of any help section.  You can re-define the magic character to
**  recognize in column 1, though, if '?' is too useful.  (Try ^A.)
*/

#define KEYFLAG '?' /* leading char in help file topic lines */

/*
** Calling sequence:
**  int result;     # 0 == success
**  char *keyword;      # topic to give help on
**  char *pathname;     # path of help file
**  result = help(keyword, pathname);
** Sample:
**  cmd = "search\n";
**  helpfile = "/usr/local/lib/program/program.help";
**  if (help(cmd, helpfile) != H_FOUND)
**      printf("Sorry, no help for %s", cmd);
**
**
** Speed this up by replacing the stdio calls with open/close/read/write.
*/

#ifdef  WDLEN
#  define   PATHSIZE    WDLEN
#else
#  define   PATHSIZE    BUFSIZ
#endif
```

```
help(keyword, path) /* print a help message */
    char *keyword;  /* on this topic */
    char *path; /* from this file */
{
/*KHC   static char oldpath[PATHSIZE] = '\0';*/ /* previous help file */
    static char oldpath[PATHSIZE] = {'\0'};
    char *oldpathp = oldpath;   /* pointer to same */

    static FILE *helpfp = NULL;
    FILE *fopen();

    char buf[BUFSIZ];   /* line from help file */
    char *bufp = buf;   /* pointer to same */
    char *bufkeyp = bufp + 1;   /* start of key in help line */

/*
** Open the help file if necessary (say, first time we enter this routine,
** or if the help file changes from the last time we were called).
*/
    errno = 0;
    if (strcmp(oldpathp, path) == SAME)
        rewind(helpfp); /* start at the beginning each time */
    else
    {   /* first time the user asked for help using this file */
        if ((helpfp = fopen(path, "r")) == NULL)
        {   /* can't open help file, so error exit */
/*KHC 2MAY89 - If switching between help files, and one is not found
 *then next call to the previous will fail because it will not be
 *reopened.  To fix this, the following line was added:
 */
            oldpath[0] = 0;
            return H_ERROR;
        }
        /* save the new path in oldpath */
        if (strlen(path) < sizeof oldpath)
            strcpy(oldpathp, path);
        else
        {   /* not enough room in oldpath, sigh */
            strncpy(oldpathp, path, sizeof oldpath);
            oldpath[sizeof oldpath] = NULL;
        }
    }

/*
** The correct help file is open.  Look in there for the keyword.
*/
    while (fgets(buf, sizeof buf, helpfp) != NULL)
    {   /*
        ** If we find the keyword:
        **  skip to the first non-keyword line
        **  print lines until you find another keyword line or EOF
        **  return success
        */
        if (buf[0] == KEYFLAG && strcmp(keyword, bufkeyp) == SAME)
        {   /* found the key */
            while (fgets(buf, sizeof buf, helpfp) != NULL
                && buf[0] == KEYFLAG)
                ;   /* eat additional keyword lines */

/*KHC substitute "moreput" for "fputs"*/
/*          fputs(bufp, stdout);    */  /* print help */
            if(moreput(bufp,1))return H_FOUND;
            while (fgets(buf, sizeof buf, helpfp) != NULL
                && buf[0] != KEYFLAG)
                if(moreput(bufp,1))return H_FOUND;
/*              fputs(bufp, stdout);*/  /* print help */
            return H_FOUND;
        }
    }

/*
** Didn't find anything in the help file for the poor user.  Return failure.
** Don't close the help file!  The user may want further help, and
** re-opening an already-open help file is wasteful of time.
*/
    return H_NOTFOUND;
}

/*********************************************************************/

#include <stdlib.h>
#include <io.h>
#include <dir.h>

/*accdep - function to check for, and return path to a file, searching
 *first in the default path, then in the path pointed to by the
 *string in the environment, then in the paths in the PATH environment.
 *Call with pointer to file name and to environment name.
 *returns pointer to path to file, or NULL if file not found.
 *(The memory to the file path has been allocated in this routine.
 *It should be freed when no longer needed.)
 */

char *accdep(char *name, char *envir)
{
    char *fpath, *fp;
    if(!access(name,0))
    {
        fpath = tmalloc(strlen(name)+1);
        strcpy(fpath,name);
        return(fpath);
    }
    if((fp = getenv(envir)) != NULL)
    {
        unsigned i;
        i = strlen(fp);
        fpath = tmalloc(strlen(name) + i + 2);
        strcpy(fpath,fp);
        if(fpath[--i] != '\\')
        {
            fpath[++i] = '\\';
            fpath[++i] = '\0';
        }
        strcat(fpath,name);
```

```
        if(!access(fpath,0))return(fpath);
        tfree(fpath);
    }
    if(!(fp = searchpath(name)))return(NULL);
    if(!access(fp,0))
    {
        fpath = tmalloc(strlen(fp) + 1);
        strcpy(fpath,fp);
        return(fpath);
    }
    return(NULL);
}

/*****************************************************************/

/*morehelp & morehlpini - functions to search a file  for a topic,
 *open a window, and display the help there, using "more".
 *morehlpini must be called before a call is made to morehelp.
 *The first argument to morehlpini, name, is the name of the help file.
 *It is searched for in the default directory first,
 *then in the directory pointed to by the environment variable in the second
 *argument, envir, then in the PATH environment.
 *If the help file is found, morehlpini returns 1, otherwise 0.
 *
 *morehelp is called with the first argument, topic, being the string
 *that is searched for in the help file to be the start of the section.
 *The second argument, title, is the string to use as the title on the
 *window showing the help.  If NULL, no title is used.
 */

    static char *hpnam = NULL;      /*To save path to help file*/

int morehlpini(char *name, char *envir)
{
    if((hpnam = accdep(name, envir))==NULL)return(0);
    return(1);
}

int morehelp(char *topic, char *title)
{
    int i;
    if(!hpnam)return(0);
    window(1,1,80,25);
    clrscr();
    moreopen(23,title);
    moreput(0,0);
    i = help(topic,hpnam);
    moreclose();
    return(i);
}
```

```
/*tline.c - K.H.Carpenter - 15AUG89
 *Module for lossless transmission line element.
 */
/*TurboC v1.5 has a bug in its cprintf routine that causes it to
 *print garbage whenever a correct printing would cause the line to
 *wrap.  Can use printf when this is anticipated to be a problem and
 *when printf is otherwise ok.
 */

#include "spedit.h"
#include <ctype.h>
#include <math.h>

/*menus*/

char *tlmenu[12] = {
    "Change name",
    "Change characteristic impedance",
    "Change delay time",
    "Change positive node at port 1",
    "Change negative node at port 1",
    "Change positive node at port 2",
    "Change negative node at port 2",
    "Change initial voltage at port 1",
    "Change initial current at port 1",
    "Change initial voltage at port 2",
    "Change initial current at port 2",
    "Save present values and return"
    };

/*********************************************************/

/*tline - function to add or change a lossless transmission line element.
 *Call with pointer to element.
 */

void tline(Element *el)
{
    double frequency, length, delay;
    int numpar, len, freq = 0, change = 0;
    char **par;
    char *z0, *td, *f, *nl, *icv1, *ici1, *icv2, *ici2;
    int number, j;
    par = spparse(el->elem, &number);
    if(number < 14)
    {
        int i;
        par = trealloc(par, 15*sizeof(char *));
        for(i = number + 1; i < 15 ; i++)par[i] = NULL;
        number = 14;
    }
    /*Force values to be non-null, for use when creating a new
     *element.  Defaults nodes to zero.
     */
    change |= makzero(&par[1]);
    change |= makzero(&par[2]);
    change |= makzero(&par[3]);
    change |= makzero(&par[4]);
    /*Order of keyword parameters is unimportant.  Sort order in
     *input element to standard order.  Then default missing ones.
     */
    z0 = td = f = nl = icv1 = ici1 = icv2 = ici2 = NULL;
    for(j = 5; j < 14 ; j++)
    {
        if(!stricmp(par[j],"Z0"))makdefault(&z0,par[++j]);
        if(!stricmp(par[j],"TD"))makdefault(&td,par[++j]);
        if(!stricmp(par[j],"IC"))
        {
            makdefault(&icv1,par[++j]);
            makdefault(&ici1,par[++j]);
            makdefault(&icv2,par[++j]);
            makdefault(&ici2,par[++j]);
        }
        if(!stricmp(par[j],"F"))
        {
            freq = 1;
            makdefault(&f,par[++j]);
        }
        if(!stricmp(par[j],"NL"))makdefault(&nl,par[++j]);
    }
    change |= makdefault(&z0,"50");
    change |= makdefault(&td,"1");
    change |= makzero(&icv1);
    change |= makzero(&ici1);
    change |= makzero(&icv2);
    change |= makzero(&ici2);
    /*Now free parsed parameters and recreate them in standard order.*/
    for(j = 5; j <= number; j++)
    {
    tfree(par[j]);
    par[j] = NULL;
    }
    makdefault(&par[5],"Z0");
    par[6] = z0;
    /*If both TD and F were given, F takes precedence.  But always
     *convert to TD form for output.
     */
    if(freq)
    {
        change = 1;
        frequency = atof(f);
        length = atof(nl);
        if(length <= 0.)length = 0.25;
        if(frequency <= 0.)frequency = 1;
        delay = length/frequency;
        tfree(td);
        td = (char *)tmalloc(30);
        tfree(f);
        tfree(nl);
        sprintf(td,"%G",delay);
    }
    makdefault(&par[7],"TD");
    par[8] = td;
    makdefault(&par[9],"IC");
```

```
    par[10] = icv1;
    par[11] = ici1;
    par[12] = icv2;
    par[13] = ici2;
    par[14] = NULL;
/*Now have input values in standard order, ready to display and change.*/
/*Now loop on menu for changing values:*/
  while(1)
  {
     clrscr();
    printf("Transmission line present values:\n\
Name: %s      Impedance (Ohms): %s      Delay (Sec): %s\n\
Port 1:  Positive node: %s      Negative node: %s\n\
         Initial potential (Volts): %s     Initial current (Amps): %s\n\
Port 2:  Positive node: %s      Negative node: %s\n\
         Initial potential (Volts): %s     Initial current (Amps): %s\n",
    par[0],par[6],par[8],par[1],par[2],par[10],par[11],par[3],par[4],
    par[12],par[13]);

    window(1,7,80,25);
    switch(menucall("Choose action:",12,tlmenu))
    {
        case '1':
            if(newname('T',"transmission line",&par[0]) > 0)change = 1;
            break;

        case '2':
    change |= newdouble("Enter new characteristic impedance value, in Ohms:"
                ,&par[6]);
            break;

        case '3':
    change |= newdouble("Enter new time delay value, in seconds:"
                ,&par[8]);
            break;

        case '4':
            change |= newnode("port 1 positive",&par[1],par[2]);
            break;

        case '5':
            change |= newnode("port 1 negative",&par[2],par[1]);
            break;

        case '6':
            change |= newnode("port 2 positive",&par[3],par[4]);
            break;

        case '7':
            change |= newnode("port 2 negative",&par[4],par[3]);
            break;

        case '8':
            change |= newdouble("Enter new value for port 1\
initial potential, in Volts:",&par[10]);
            break;

        case '9':
            change |= newdouble("Enter new value for port 1\
initial current, in Amperes:",&par[11]);
            break;

        case 'a':
            change |= newdouble("Enter new value for port 2\
initial potential, in Volts:",&par[12]);
            break;

        case 'b':
            change |= newdouble("Enter new value for port 2\
initial current, in Amperes:",&par[13]);
            break;

        case 'R':
        case 'c':
            /*here to save element, free storage allocated in
            this module, and return*/
            if(change)
            {
                circuitstat = 2;
                tfree(el->elem);
                el->elem = spunparse(par);
            }
            freeparret(par, number);
            return;

        case 'H':
        default:
            if(helpav)if(!morehelp("tline\n","HELP for TRANSMISSION LINE"))
                waitforkey(Pressany);
    }   /*end of switch*/
     window(1,1,80,25);
  }    /*end of while*/
}
```

H.2 The "help" file

The context sensitive help is provided in SPFEDIT by the program's reading through the file SPFEDIT.HLP, which must be in the current directory or in one given in the PATH environment variable. SPFEDIT finds the section of the file to use by the key word on a line beginning with a question mark. The order of these sections in the file is approximately the same as the order in which they are referenced in the code. The file is listed on the following pages using the line printer font on a Hewlett Packard LaserJet printer so that what is seen on the PC screen is accurately reproduced.

?title

SPFEDIT

SPFEDIT

An interactive editor for input files for the
SPICE circuit simulation program
for transient analysis of ciruits containing fuse elements.

26 JUNE 1989

For help on any menu, press "h" to the menu's prompt.
For introductory help, press "h" now.

?begin

SPFEDIT helps one to write an input file for the SPICE circuit simulator program to cause it to perform a transient analysis on a circuit containing fuse elements and/or saturating inductances. Since only transient analyses are useful when such elements are included, no other type of analysis can be entered via this program. Also, there are many other SPICE element types, such as semiconductors of various sorts, that cannot be entered directly via this program. Any of these elements can be added to the circuit with an ordinary text editor before the file is input to SPICE, however.

Before beginning to enter or edit a SPICE input file with SPFEDIT, one should have a sketch of the circuit at hand with the nodes numbered. The node numbered 0 is the ground (or reference) node to SPICE.

SPICE requires that all nodes have a DC path to the ground node. The only elements that provide DC paths are voltage sources, resistances, and inductors. (Fuses do not, since they are implemented internally as dependent sources.) Whenever the circuit sketch indicates a node that would not have a DC path to ground, one should add a large resistor (say 100Megohm) from that node to ground.

SPFEDIT is menu driven. For help on the current menu, enter an "h" at its prompt. To return to the previous menu, enter "r".

Menus require that single characters be entered to make a selection. Do not enter a carriage return following the character or it will be taken as the answer to the next query.

Some menu items require entry of a name or a numerical value. Since these contain a variable number of characters, a carriage return must be entered to complete the item. While entering the item, the DOS editing facilities are active (e.g., backspace deletes last character left, escape deletes the entire line). To abort the entry of a name, enter just a carriage return. To abort the entry of a numerical value, enter a semicolon.
WARNING - Do not enter control-c when a numerical value or name is required or else the run of SPFEDIT will be aborted without saving the circuit!

The numerical methods used in SPICE may have convergence difficulties on a transient analysis if reactive elements do not have resistances associated with them. Thus, even if the DC path to ground is present, it may be necessary to add a large resistor in parallel with an inductor or capacitor to get stable numerical behaviour. Also the maximum step size indicated on the transient analysis line must be sufficiently small relative to the rate of change of circuit voltages to avoid numerical problems. If error messages indicating numerical problems result from a run of SPICE, try smaller step sizes and/or large resistances added to the circuit before making any changes in precision from the defaults (via the .OPTION card).

?main

The MAIN menu allows chosing between the basic operations of SPFEDIT. To its prompt:

Enter "1" to either add lines to the SPICE input file or edit lines in it.

Enter "2" to read into the program from the disk the SPICE input file pointed to by the current path entry.

Enter "3" to save to disk, in the file pointed to by the current path entry, the SPICE input file contained in the program memory.

Enter "4" to clear the SPICE input file contained in the program memory and initialize a new file in memory.

Enter "5" to display the current path entry pointing to the disk file used by choices "2" and "3" and to change it if desired.

Enter "6" to display the SPICE input file in the program memory on the screen.

Enter "7" to exit the SPFEDIT program.

?modify

The choices on the MODIFY menu allow adding or changing lines in the SPICE input file in the program memory. (Any addition or changes will mark the memory contents as modified so that a warning will be issued if an attempt is made to exit the program before saving the file in memory to disk.)
To the MODIFY menu prompt:

Enter "1" to add a new circuit element to the SPICE input file.

Enter "2" to modify an existing element in the SPICE input file.

Enter "3" to modify or add a command (a line in the file beginning with a period, such as .TRAN for transient analysis or .PLOT for printer plots).

Enter "4" to change the title line in the SPICE input file.

Enter "5" for an explanation of how to add comments to a SPICE input file. (This program does not implement editing of comments itself.)

Enter "6" to return to the MAIN menu.

?addcom

This menu allows inclusion and modification of some SPICE control lines.

Enter "1" to add or modify the .TRAN control line. This line is required or else SPICE will not perform an analysis.

Enter "2" to add or modify the .PRINT control line. This line specifies the voltages and/or currents that are to be

listed as numbers in the output listing file.

Enter "3" to add or modify the .PLOT control line. This line specifies the voltages and/or currents that are to be plotted (as printer plots) in the output listing file.

Enter "4" to add or modify the .WIDTH control line. This line allows selection of 80 or 132 column width for the output listing file.

Enter "5" to add or modify the .OPTION control line. This line gives options and error tolerances to SPICE. Refer to the User's Guide before entering this line.

Or enter "r" to abort this menu and return to the previous level.
?addloc
This menu chooses the position in the SPICE input file in which to locate the new element. To the prompt:

Enter "1" to cause the new element to become the first element in the file.

Enter "2" to cause the new element to become the last element in the file.

Enter "3" to allow placing the new element after another element already included in the file. The program will ask for the name of the element for the new element to follow, and will allow display of the current elements to make the choice.

Enter "r" to abort adding a new element and return to the previous menu.
?addtype
Enter the number corresponding to the type of element to be added to the circuit at the prompt.

Or enter an "r" to abort adding the element and return to the previous menu.

After the element type is selected you will be prompted to enter the name for the element. To abort from the name entry level, just enter a <CR> alone.
?resistor
The parameters required for a resistor element are the nodes between which it is to be connected, its value, and, optionally, two temperature coefficient parameters.

The first and second nodes must be different, and must be entered as integers. Any integer less than 32000 is acceptable to this editor (but SPICE itself limits the total number of nodes to far less than this!).

The value of resistance must be positive (SPICE does not allow zero values for resistors) and less than the maximum value the SPICE program will admit. (See the SPICE User's Guide for the actual minimum and maximum limits.) If a true zero resistance is required, eliminate one of the nodes or use a voltage source of value zero.

The temperature coefficients are included for completeness with the SPICE program, which modifies the actual resistance values, using the coefficients if they are present, according to the TEMPERATURE value given to SPICE. For most fuse circuit applications the temperature coefficient should be omitted.

If a temperature coefficient is present and it is desired to remove it, this is done by setting it to zero.
?capacitor
The parameters required for a capacitor element are the nodes between which it is to be connected, its value, and, optionally, the initial voltage across the capacitor at time zero.

The first and second nodes must be different, and must be entered as integers. Any integer less than 32000 is acceptable to this editor (but SPICE itself limits the total number of nodes to far less than this!).

The value of capacitance must be positive and less than the maximum value the SPICE program will admit.
(See the SPICE User's Guide for the actual minimum and maximum limits.)

The initial voltage will be used for transient analysis of the circuit when UIC is present on the .TRAN control statement - this will be the case if the statement is entered using SPFEDIT. The voltage is taken to be positive on the first node. If no initial voltage is specified, the default assumed by SPICE is zero.

If an initial voltage is present and it is desired to remove it, choose the menu item of "delete initial condition".
?inductor
There are two models for inductance - a linear model for constant inductance, and a hyperbolic tangent model for saturating inductance.
The default inductance menu is for the linear model. To change to the saturating inductance model select item "7" on the linear inductance menu. To change from the saturating to linear model, select item "8" on the menu for the saturating model.

Both models require entry of the element name, which must begin with "L", and the nodes between which it is connected. The first and second nodes must be different, and must be entered as integers. Any integer less than 32000 is acceptable to this editor (but SPICE itself limits the total number of nodes to far less than this!).

The linear model requires entry of the constant inductance value, and, optionally, entry of the current entering the first node of the inductor at time zero. The inductance value must be a positive number.
The time zero initial condition can be positive, negative, or zero. The default if no value is entered is zero. The time zero value is only used if UIC appears on the .TRAN control statement. If an initial value is present, it can be removed by selecting menu item "6".

The saturating inductance model requires entry of the initial inductance, which is the small signal value -- the inductance when the current is small. It also requires the entry of the inductance after saturation which is the value that the same winding geometry would have if air-cored.
The value of the saturation current must also be entered. This is the value obtained by dividing the flux through the winding due to the magnetization of the magnetic material by the initial inductance less the inductance after saturation. (It is the current when at the "knee" of the magnetization curve.)

The saturation model exponent can be changed by choosing menu item "7" for

the saturating inductor. This exponent is used to raise the argument of the hyperbolic tangent, representing the magnetization curve of the material, to a power and then to take the same root of the result. This causes the saturation to come into effect more abruptly for exponent values greater than one and to come into effect more gradually for exponent values less than one. It should be chosen to match the actual magnetic properties of the core material, but should never be far from one in value.

There is no provision for entering a non-zero time zero current for a saturating inductor.

Choose the last menu item to save the inductor element and return to the previous editor level.
?fuse
The fuse element requires specification of the element name, which must begin with a "G" because internally to SPICE it is a voltage controlled current source. The other parameters which must always be specified are the nodes between which it is connected and the length, width, and thickness of the assumed rectangular shape of the fuse material.

The resistivity parameters for the model must be present. They default to those for the material shown above the menu. If changes are desired from the default, then the resistivity menu must be entered by selecting item "7".
?fusres
The fuse model used in SPICE2G6-SB2 is a modification of the "Fireset" model by Ronald S. Lee. Menu items "1" through "4" allow changing the parameters of the "Fireset" model. See the report by Lee for more information on these.

Menu items "5" and "6" allow replacing the "Fireset" parameters by their default values for either copper or aluminum material. These should be the only menu choices selected unless the user understands the operation of the "Fireset" model.

Menu items "7" and "8" allow changing the parameters that control the dynamic calculation of the "V/L" parameter in the "Fireset" model. Do not change these values unless you understand how the dynamic scaling is accomplished. If the values should be changed, the defaults can be restored by selecting menu item "9".

Selecting menu item "A" allows selecting the material to be used as the default for future fuse elements to be entered (ones entered after saving the one currently being described to the circuit in program memory.)

Selecting menu item "B" returns to the first fuse menu.
?trans
The .TRAN control line requires entry of four parameters.

The output time step is the spacing between time values for data in the output listing file. The numerical method may use smaller or larger step sizes than this. (If larger, results are interpolated for the output listing file.)

The output start time is the time at which output values are first written to the output listing file. The analysis always starts at time zero.

The analysis stop time is the time at which the analysis is halted.

The analysis maximum step size is the largest time step that will ever be used by the numerical method. This value must always be small enough to give the amount of detail needed in the ouput listing file. If this value is too large, the numerical method may fail to converge at some time. If such an error is encountered try reducing this time and running the circuit with SPICE again.

The raw data output file, which is input to the postprocessor, NUTMEG, contains values for all times at which they are calculated, regardless of the values of output start time and output time step given on this line.
?cvsadp
Select menu item "1" to add a new time-value pair for the piecewise linear source.

Select menu item "2" to delete a present time-value pair for the piecewise linear source. You will then be asked to supply the "time" of the pair. The pair at time zero cannot be deleted. If you specify zero for the time, then you will be asked to enter a new value for time zero.

To change the value for times other than zero, first delete the pair, then add it with the new value.
?cvsdc
Independent current or voltage sources can have either constant, dc values or piecewise linear specifications of values versus time.
The first menu displayed is for dc sources. To change to piecewise linear sources, select menu item "5".

The element name must be present for an independent source, and must begin with "V" for a voltage source and "I" for a current source.

The first node is the positive node and the second node is the negative node. For a current source, the current specified is assumed to enter the positive node, flow through the source, and exit the negative node.

The value for the source can be positive, negative, or zero.
Voltage sources with zero value are ammeters for SPICE, since SPICE can output current values only for currents through independent voltage sources. Current sources with zero value are open circuits and should not be used.
?cvspwl
The piecewise linear menu allows specifying the source value at any (non-negative) time. The value at time zero must always be present, but time-value pairs for other times may be added or deleted at will. The value at time zero may be changed by selecting time zero as the time for the choice of deleting a pair.

Values at times between specified pairs are linearly interpolated. Values after the last specified time retain the last specified value.

One may return to a dc model for the source from the piecewise linear one by selecting menu item "5" from the piecewise linear menu.
?tline
Lossless transmission lines are specified by giving the nodes at each end. One end is designated "port 1" the other "port 2".
The positive nodes at each port correspond to the same conductor;

similarly for the negative nodes.
The characteristic impedance is a single real number, since the line is assumed to be lossless.
The length of the line must be specified to this editor as its time delay in seconds. This is equivalent to the length in wavelengths divided by the frequency for cases of sinusoidal excitation.
Initial conditions must be specified for the current entering the positive node at each port and for the potential drop from the positive node to the negative node at each port. (These are required for transient analysis with UIC on the .TRAN control line. Otherwise they are ignored.)

Note that for the menu items above 9 the selection must be made by entering a letter rather than a number. (Actually the number in base 16).

Appendix I

Listing of the installation files for the firing circuit simulation code

The following listings, first of file README.INS, then of file INSTALL.BAT, and finally of file SPDRIVER.DOC, explain how the family of programs making up the firing system simulation code are loaded onto a PC. One obtains a distribution diskette having all the required files on it, including these two. One first reads the README file, and follows the instructions in it. The INSTALL.BAT file utilizes the MSDOS BATCH method to automate nearly all of the installation.

I.1 README.INS

```
This is the distribution diskette for the SPDRIVER programs:
SPFEDIT  - a SPICE input file editor for use with circuits
           containing fuse elements
SP2G6SB2 - SPICE2G6 as modified to allow fuse elements and
           saturating inductors
NUTCGA   - NUTMEG for MSDOS pc's having CGA graphics
SPDRIVER - The batch file to invoke the files above.

Several additional auxiliary files are also included, as well
as a "shareware" program, SIMCGA, which allows running NUTCGA
on a system with a "Hercules" type monographic card.

To install the software, be sure your hard drive has at least
2 Megabytes free space.  Then make the current drive the one
on which the files are to be installed.  Then type the command:
A:INSTALL, to install if you have CGA graphics, or
A:INSTALLH, to install if you have monographics.

After installation is complete a screenful of information will
be present regarding changes needed to your AUTOEXEC.BAT file.
Make these changes before attempting to use the programs.

K.H.Carpenter - 15AUG89 - KSU Dept. of EECE - (913)532-5600
```

I.2 INSTALL.BAT

```
echo off
```

```
rem INSTALL.BAT - installation file for SPDRIVER programs
rem K.H. Carpenter - 15AUG89
rem To use:  Place floppy with distributed files in drive a:,
rem make the default drive the one on which the files are to be
rem installed, and enter the command, A:INSTALL
echo SPDRIVER installation program -
echo  This program will make a subdirectory \SPDRIVER in the current drive
echo  (if it is not already present) and also subdirectories \SPICE3B1
echo  and \SPICE2G6.
echo  Enter control-c now to avoid creating these.
pause
mkdir \spdriver
mkdir \spdriver\util
mkdir \spice2g6
mkdir \spice3b1
echo Copying files to \SPDRIVER
copy a:spdriver.* \spdriver
copy a:spfedit.* \spdriver
copy a:forlpt.com \spdriver\util
copy a:forlpt.doc \spdriver\util
copy a:dgrep.com \spdriver\util
copy a:tlip.com \spdriver\util
copy a:autoexec.bat \spdriver\util
copy a:config.sys \spdriver\util
rem copy a:nansi.sys \spdriver\util
copy a:zoo201.exe \spdriver\util
rem copy a:lftocrlf.exe \spdriver\util
rem copy a:tbooz.com \spdriver\util
echo Creating ZOO archiver in \SPDRIVER\UTIL
echo To get versions from installation disk, answer "y" to any queries that follow:
cd \spdriver\util
zoo201
cd \spice2g6
copy a:spice2.bat
rem copy a:usersgui.de2
echo Decompressing SPICE into \SPICE2G6
\spdriver\util\zoo eSO a:sp2g6sb2
rem \spdriver\util\tbooz a:sp2g6sb2.zoo
echo Decompressing NUTMEG into \SPICE3B1
cd \spice3b1
\spdriver\util\zoo xSO a:nutmeg
rem \spdriver\util\tbooz a:nutmeg.zoo
cd \
echo Installation from distribution disk is now complete.
echo Before using these programs you must see that the DOS PATH is
echo set to access the correct files.
echo A sample AUTOEXEC.BAT file has been placed in \SPDRIVER\UTIL.
echo This file contains a line PATH=... The paths in this line must
echo be included in the AUTOEXEC.BAT file in the root directory on
echo the boot disk for access to the SPDRIVER programs to be enabled.
echo See your MSDOS User's Guide for how to use AUTOEXEC files and
echo set environments if you do not know what all this means!
echo .
```

```
rem echo You must also have ANSI.SYS included in your CONFIG.SYS file in
rem echo the root directory on the boot disk.  Alternatively, if you do
rem echo not have ANSI.SYS, you can use the line in the CONFIG.SYS file found
rem echo in \SPDRIVER\UTIL (or move this file itself to the root directory
rem echo on the boot disk) and move the file ANSI.SYS in \SPDRIVER\UTIL
rem echo to the root directory on the boot disk.
rem echo .
echo You need to have a buffers= and files= line in your config.sys
echo file in the root directory to be able to use the SPDRIVER system.
echo Suggested values for these lines can be found in the file
echo \SPDRIVER\UTIL\CONFIG.SYS.
echo If you do not already have a config.sys file, copy this sample
echo one to your root directory on the boot disk.
echo .
echo When this has been done and the system rebooted, you may use
echo SPDRIVER by making the current directory the one on which you
echo wish to store circuit files and outputs, and giving the command,
echo SPDRIVER.
echo For further information see \SPDRIVER\SPDRIVER.DOC.
rem pause
```

I.3 SPDRIVER.DOC

```
SPDRIVER.DOC - 15AUG89 -- K.H.CARPENTER

SPDRIVER is a system for simplified use of the SPICE2G6 circuit
simulator program as modified to include fuse and saturating inductor
elements.  The system consists of the following segments:
1. The batch driver - SPDRIVER.BAT
2. The input file editor - SPFEDIT.EXE
3. The circuit simulator - SP2G6SB2.EXE
4. The interactive graphics postprocessor - HUTCGA.EXE
5. The context sensitive, online help file for the editor - SPFEDIT.HLP
6. Other documentation (files ending in .DOC or .MAN).
7. Utility programs utilized either in installation of the software
   or while running it.

This is a "beta test" version of the software.  Updates will be
produced when experience using the current version warrants.
Please keep the author informed of problems encountered with these
programs and with suggestions for improvement.  Send your suggestions,
comments, and a description of the problems encountered to:
            Kenneth H. Carpenter
            Department of Electrical and Computer Engineering
            Durland Hall
            Kansas State University
            Manhattan, Kansas 66506
```

Specific instructions for use of the SPDRIVER.BAT driver program will be found in the file SPDRIVER.MAN. The following is a general description of the documentation available for the SPDRIVER software package.

The documentation for the SPDRIVER software is in three parts. The first is a set of files suitable for reading on the PC using the DOS "MORE" command. These files have extensions of .DOC or

.MAN. These files also have printed versions (where boldface and underscores emphasize items in a way that cannot be done on the screen). The second part of the documentation consists of printed reference documents which do not have versions for reading on the PC. (They do have source documents for a word processing program available, however, if one wishes to print their own, or to make changes in the contents.) The third part of the documentation is on-line help available while using the programs. Some of this "help" comes from files which could be read directly, but in so doing the context would be lost.

The information in the .DOC files is descriptive in nature, but not to the level of detail that will be found in the printed reference manuals. The information in the .MAN files consists of terse explanations of the commands used with the programs, in the form of the Unix "manual" pages. The Unix manual format will be familiar to users of that system. For ones not familiar with Unix, the only warning is that Unix systems are case sensitive, and commands shown in lower or upper case must be entered the same way to work properly. Also, some of the .MAN files are merely the ones supplied with the Berkeley distribution of SPICE, and may not reflect the changes needed to allow operation on a PC. In particular, not all the commands listed in NUTMEG.MAN are available in the PC version. (Trying to use these commands will result in harmless error messages.)

The SPFEDIT interactive circuit entry and editing program is primarily documented through its context sensitive "help" feature. Merely read the first screen when running the program and follow the directions to obtain "help" at any point in the program. (This information is all contained in the SPFEDIT.HLP file. In this file, lines beginning with a "?" are the keys that allow the program to find the appropriate section. Do not edit this file or you will, in all likelihood, "mess up" the help system.)

The SPICE program, version 2G6-SB2, is documented by the User's Guide. It is available as a printed document, not in a form for reading on a PC. It should be read for information on how to use features of SPICE that cannot be accessed through the SPFEDIT program, and to explain problems encountered with circuits entered through SPFEDIT. There are textbooks available on use of the SPICE circuit simulator, but none of them will refer to version 2G6-SB2 specifically, and they will have no information on the fuse element. Actually, the User's Guide is all one needs to have, along with a knowledge of circuit theory, but some patience is needed to read and understand it.

The NUTMEG program is an interactive graphics post-processor for data output by SPICE. It is not mentioned in the SPICE User's Guide, but is documented by NUTMEG.MAN and the printed Introduction to Nutmeg. The version supplied with this software is named NUTCGA.EXE on the disk because it requires an IBM-PC Color Graphics adapter compatible graphics card to be used. (This can be simulated on a system having only a Hercules-type monochrome graphics adapter by use of the SIMCGA software– which is pointed out in the README.INS file, which you should have read before this one!) NUTCGA is merely a compiler and graphics specific version of the program distributed by UC Berkeley with SPICE3B1. The .MAN file is the unaltered one from Berkeley. The "help" command inside NUTMEG requires that external "help" files be available on disk. These take up considerable disk space, and are not being included with this distribution, with the result that no on-line help is available. The help files from Berkeley can be made available on request to those who would like to have them.

The SCONVERT utility program is needed to convert the format of the raw data files output by SPICE version 2G6 to the format produced by version 3B1 so that NUTMEG can read them. The SCONVERT.MAN file contains a description of both formats, and this information is all that is needed to write other programs to output data that could be processed by NUTMEG.

SPICE2G6 printer output files contain FORTRAN carriage control characters. To convert these control characters to the linefeed or formfeeds needed by most PC printers, the FORLPT utility program can be used. See FORLPT.DOC for details.

The DGREP utility program is used to search the printer output file from a run of SPICE2G6-SB2 for information on "fuse blowing" and print them for inspection before running NUTMEG. It is called by the SPDRIVER.BAT file, and no documentation is provided. (DGREP is a public domain version of the Unix "grep" utility, from DECUS). If you already have a "grep" program on your PC system, it can be used in place of DGREP by removing the "D" in "DGREP" in the SPDRIVER.BAT file.

The ZOO201.EXE file on the distribution disk is a self extracting family of archiving programs. See the file, ZOO.MAN, for more information concerning these programs, all of which are placed in \SPDRIVER\UTIL by executing the INSTALL batch file. ZOO.EXE is the archiver needed to unpack the large files on the distribution diskette. This ZOO software was written by Rahul Dhesi. It is freely distributable (no "shareware" payments requred). The utility FLIP is also by the same author, and can be used to "flip" between UNIX and MSDOS line end conventions for text files.

One last warning. The SPICE program uses lots of memory. Your PC must have 640K or more to run it. If you get "not enough memory" type errors, then remove any memory resident utilities and try again.

Happy SPICEing! – Ken Carpenter.

Appendix J

Listing of the SPDRIVER.BAT file

```
echo off
rem Driver batch file for SPFEDIT, SPICE2G6, and NUTMEG
rem K.H.Carpenter - 25MAY89 - 28SEP89
if %1. == . goto help
if %1. == ?. goto help
if %2. == RUN. goto run
if %2. == run. goto run
if %2. == PLOT. goto plot
if %2. == plot. goto plot
echo Driver program for SPICE - entering editor:
spfedit %1.cir
echo .
echo To submit %1.cir to SPICE, press any key except cntrl-c.
echo To quit SPDRIVER now, press cntrl-c.
pause
:run
if not exist %1.cir goto help2
sp2g6sb2 %1.cir %1.out %1.r1 %1.r2 %1.r3 %1.r4 %1.r5 %1.r6 %1.r7
goto done
:help2
echo Cannot find %1.CIR - cannot run SPICE now.
goto end
:done
rem last argument to sconvert limits output file size
sconvert o %1.r1 b %1.r1b 512
if not errorlevel 1 del %1.r1
dgrep "1*ERROR" %1.out
echo To plot output from %1 using NUTMEG, press any key except cntrl-c.
echo To quit SPDRIVER now, press cntrl-c.
pause
:plot
if not exist %1.r1b goto help3
dgrep " FUSE from" %1.out
echo .
echo Basic NUTMEG commands:     "display" to show a list of variables
echo  "plot ..." to plot variables.  See NUTMEG.MAN for more details.
echo To exit from NUTMEG enter "quit"
echo .
```

```
pause
rem setcga
nutcga -n %1.r1b
rem setmono
goto end
:help3
echo Cannot find %1.R1B - cannot run NUTMEG now.
goto end
:help
echo .
echo Correct use is SPDRIVER NAME (option)
echo where NAME is the name for the SPICE input file without any extension,
echo and (option) can be either RUN or PLOT or may be omitted.
echo .
echo If (option) is omitted, SPFEDIT is entered for file NAME.CIR.
echo If (option) is RUN, SPICE is run on file NAME.CIR.
echo If (option) is PLOT, NUTMEG is run on output from previous run of
echo SPICE on NAME.CIR.
echo .
echo SPDRIVER proceeds to SPICE after SPFEDIT and proceeds to NUTMEG after SPICE.
echo .
:end
```

Appendix K

Unix-style manual pages

The following pages contain reproductions of the Unix-style manual pages for the programs SPDRIVER, SCONVERT, and NUTMEG. (The latter two are as received with the U.C. Berkeley distribution of SPICE3B1[7], the first was written in the "man page" style for consistency.)

NAME

spdriver - MSDOS batch driver for SPFEDIT, SPICE version 2G6-SB2, and NUTMEG

SYNOPSIS

spdriver circuitfile [<run , plot>]

DESCRIPTION

Spdriver is a MSDOS batch driver for the SPICE, version 2G6-SB2, family of software. It runs the interactive editor, SPFEDIT, on "circuitfile.cir", then submits it to SPICE, converts the raw data to the form required by NUTMEG and submits the raw data to NUTMEG. The edit and/or run steps can be bypassed, and descriptive prompts are issued to the user.

Arguments are:

circuitfile

This is the file specification of the SPICE input and output files without their trailing "." and extensions.

run (or plot but not both)

If "run" is present as the second argument, then bypass the editing of the input file and submit it to SPICE as the first step. If "plot" is present as the second argument, then bypass the editing and running of SPICE on the input file and go directly to the post-processor, NUTMEG, as the first step.

SEE ALSO

SPDRIVER.DOC and documentation mentioned therein

AUTHOR

Kenneth H. Carpenter, Dept. of EECE, Kansas State University, Manhattan, KS 66506

BUGS

The programs called by the SPDRIVER.BAT file must be in the default path or in a path given by the MSDOS PATH environment string.

NAME

sconvert - convert spice formats

SYNOPSIS

sconvert fromtype fromfile totype tofile
sconvert fromtype totype
sconvert

DESCRIPTION

Sconvert translates spice output files among three formats: the old binary format, a new binary format, and a new ascii format. The formats are specified by the **fromtype** and **totype** arguments: 'o' for the old format, 'b' for the new binary format, and 'a' for the new ascii format. **Fromtype** specifies the format to be read, and **totype** specifies the format to be written. If **fromfile** and **tofile** are given, then they are used as the input and output, otherwise standard input and output are used. (Note that this second option is only available on UNIX systems - on VMS and other systems you must supply the filenames.) If no arguments are given, the parameters are prompted for.

Binary format is the preferred format for general use, as it is the most economical in terms of space and speed of access, and ascii is provided to make it easy to modify data files and transfer them between machines with different floating-point formats. The old format is provided only for backward compatibility. The three formats are as follows:

Old:

What	Size in Bytes
title	80
date	8
time	8
numoutputs	2
the integer 4	2
variable names --	
char[numoutputs][8]	numoutputs * 8
types of output	numoutputs * 2
node index	numoutputs * 2
plot title	numoutputs * 24
the actual data	numpoints * numoutputs * 8

Ascii:

Title: *Title Card String*
Date: *Date*
[Plotname: *Plot Name*
Flags: *complex* or *real*
No. Variables: *numoutputs*
No. Points: *numpoints*
Command: *nutmeg command*
Variables: 0 *vamame1 typename1*
1 *vamame2 typename2*
etc...
Values:
0 n n n n ...
1 n n n n ...
And so forth...
] repeated one or more times

If one of the flags is *complex*, the points look like r,i where r and i are floating point (in %e format). Otherwise they are in %e format. Only one of *real* and *complex* should appear.

The lines are guaranteed to be less than 80 columns wide (unless the plot title or variable names are very long), so this format is safe to mail between systems like CMS.

Any number of **Command:** lines may appear between the **No. Points:** and the **Variables:** lines, and whenever the plot is loaded into **nutmeg** they will be executed.

Binary:

Title Card (a NULL terminated string)
Date, Time (a NULL terminated string)
[
Plot title (a NULL terminated string)
Number of variables (an int)
Number of data points (an int)
flags (a short)
variable header struct (repeated numoutputs times)
variable name (a NULL terminated string)
variable type (an int)
set of outputs (repeated numpoints times)
] repeated one or more times.

A set of outputs is a vector of doubles of length numoutputs, or a vector of real-imaginary pairs of doubles if the data is complex.

SEE ALSO

nutmeg(1), spice(1), writedata(3)

AUTHOR

Wayne Christopher (faustus@cad.berkeley.edu)

BUGS

If variable names and the title and plotname strings have trailing blanks in them they will be stripped off when the file is read, if it is in ascii format.

If a plot title begins with "Title:" **nutmeg** will be fooled into thinking that this is an ascii format file. Sconvert always requires the type to be specified, however.

NAME

nutmeg - spice post-processor

SYNOPSIS

nutmeg [-] [-n] [-t term] [datafile ...]

DESCRIPTION

Nutmeg is a post processor for SPICE - it takes the raw output file created by **spice -r** and plots the data on a graphics terminal or a workstation display. Note that the raw output file is different from the data that SPICE writes to the standard output.

Arguments are:

-
Don't try to load the default data file ("rawspice") if no other files are given.

-n (or -N)
Don't try to source the file ".spiceinit" upon startup. Normally **nutmeg** tries to find the file in the current directory, and if it is not found then in the user's home directory.

-t term (or -T term)
The program is being run on a terminal with *mfb* name **term**.

Further arguments are taken to be data files in binary or ascii format (see **sconvert(1)**) which are loaded into nutmeg. If the file is in binary format, it may be only partially completed (useful for examining SPICE ouput before the simulation is finished). One file may contain any number of data sets from different analyses.

Nutmeg data is in the form of vectors: time, voltage, etc. Each vector has a type, and vectors can be operated on and combined algebraicly in ways consistent with their types. Vectors are normally created when a data file is read in (see the **load** command below), and when the initial datafile is loaded. They can also be created with the **let** command.

An expression is an algebraic formula involving vectors and scalars (a scalar is a vector of length 1), and the following operations:

+, -, *, %, /, ^, and ,.

% is the modulo operator, and the comma operator has two meanings: if it is present in the argument list of a user-definable function, it serves to seperate the arguments. Otherwise, the term **x , y** is synonymous with **x + j(y)**.

Also available are the logical operations & (and), | (or), ! (not), and the relational operations <, >, >=, <=, =, and <> (not equal). If used in an algebraic expression they work like they would in C, producing values of 0 or 1. The relational operators have the following synonyms: "gt" is >, "lt" is <, "ge" is >=, "le" is <=, "ne" is <>, "eq" is =, "and" is &, "or" is |, and "not" is !. These are useful when < and > might be confused with IO redirection (which is almost always).

The following functions are available:

mag(vector) - The magnitude of vector.

ph(vector) - The phase of vector.

j(vector) - *i* (sqrt(-1)) times vector.

real(vector) - The real component of vector.

imag(vector) - The imaginary part of vector.

db(vector) - 20 * log10(mag(vector)).

log(vector) - The logarithm (base 10) of the vector.

ln(vector) - The natural logarithm (base e) of vector.

exp(vector) - e to the vector power.

abs(vector) - The absolute value of vector.

sqrt(vector) - The square root of vector.

sin(vector) - The sin of vector.

cos(vector) - The cosine of vector.

tan(vector) - The tangent of vector.

atan(vector) - The inverse tangent of vector.

norm(vector) - The vector normalized to 1 (i.e, the largest magnitude of any component will be 1).

rnd(vector) - A vector with each component a random integer between 0 and the absolute value of the vectors's corresponding component.

mean(vector) - The result is a scalar (a length 1 vector) that is the mean of the elements of vector.

vector(number) - The result is a vector of length number, with elements 1, 2, ... number. If number is a vector then just the first element is taken, and if it isn't an integer then the floor of the magnitude is used.

length(vector) - The length of vector.

interpolate(plot.vector) - The result of interpolating the named vector onto the scale of the current plot. This function uses the variable **polydegree** to determine the degree of interpolation.

A vector may be either the name of a vector already defined, a floating- point number (a scalar), or a list like **[elt1 elt2 ... eltn]**, which is a vector of length n. A number may be written in any format acceptable to SPICE, such as **14.6MEG** or **-1.231E-4**. Note that you can either use scientific notation or one of the abbreviations like *MEG* or *G*, but not both. As with SPICE, a number may have trailing alphabetic characters after it.

The notation **expr [lower upper]**, where lower and upper are numbers, denotes the range of elements from expr between lower and upper. The notation **expr [num]** denotes the num'th element of expr. If upper is lower than lower, the order of the elements in the vector is reversed. In all other cases, [and] serve to surround literal vectors as described above. (You may have to use a lot of parentheses to make sure that you get what you want. For instance, you have to type **print (foo) ([1 2])** to print the two vectors. Otherwise it will be interpreted as a function call or a vector with an index.) Note that the expression **foo[10 20][5]** will *not* yield the 15th element of **foo**, but rather the 5th. In general only the last index suffix on an expression will take effect.

To reference vectors in a plot that is not the *current plot* (see the **setplot** command, below), the notation **plotname.vecname** can be used.

Either a plotname or a vector name may be the wildcard **all**. If the plotname is all, matching vectors from all plots are specified, and if the vector name is all, all vectors in the specified plots are referenced. Note that you may not use binary operations on expressions involving wildcards - it is not obvious what **all + all** should denote, for instance.

Thus some (contrived) examples of expressions are:

```
cos(TIME) + db(v(3))

sin(cos(log([1 2 3 4 5 6 7 8 9 10])))

TIME * rnd(v(9)) - 15 * cos(vin#branch) ^ [7.9e5 8]

not ((ac3.FREQ[32] & tran1.TIME[10]) gt 3)
```

Nutmeg commands are as follows:

plot exprs [ylimit ylo yhi] [xlimit xlo xhi] [xindices xilo xihi]
[xcompress comp] [xdelta xdel] [ydelta ydel] [xlog] [ylog] [vs xname]
[xlabel word] [ylabel word] [title word] [samep]

Plot the given exprs on the screen (if you are on a graphics terminal). The xlimit and ylimit arguments determine the high and low x- and y-limits of the axes, respectively. The xindices arguments determine what range of points are to be plotted - everything between the xilo'th point and the xihi'th point is plotted. The xcompress argument specifies that only one out of every comp points should be plotted. If an xdelta or a ydelta parameter is present, it specifies the spacing between grid lines on the X- and Y-axis. These parameter names may be abbreviated to xl, yl, xind, xcomp, xdel, and ydel respectively. The xname argument is an expression to use as the scale on the x-axis. If xlog or ylog are present, the X or Y scale respectively will be logarithmic. The xlabel and ylabel arguments cause the specified labels to be used for the X and Y axes, respectively. If samep is given, the values of the other parameters (other than xname) from the previous plot, hardcopy, or asciiplot command will be used unless redefined on the command line. Finally, the title argument will be used in the place of the plot name at the bottom of the graph.

hardcopy file *plotargs*

Just like plot, except creates a file called file containing the plot. The file is an image in *plot*(5) format, and can be printed by either the plot(1) program or lpr with the -g flag.

asciiplot *plotargs*

Produce a line printer plot of the vectors. The plot is sent to the standard output, so you can put it into a file with *asciiplot args ... > file*. The set options width, height, and nobreak determine the width and height of the plot, and whether there are page breaks, respectively. Note that you will have problems if you try to asciiplot something with an X-scale that isn't monotonic (i.e, something like *sin(TIME)*), because asciiplot uses a simple-minded sort of linear interpolation.

define function(arg1, arg2, ...) expression

Define the *user-definable function* with the name *function* and arguments *arg1, arg2, ...* to be *expression*, which may involve the arguments. When the function is later used, the arguments it is given are substituted for the formal arguments when it is parsed. If *expression* is not present, any definition for *function* is printed, and if there are no arguments to *define* then all currently active definitions are printed. Note that you may have different functions defined with the same name but different arities. Some useful definitions are:

```
define max(x,y) (x > y) * x + (x <= y) * y
define min(x,y) (x < y) * x + (x >= y) * y
```

undefine function ...

Definitions for the named user-defined functions are deleted.

let name = expr

Creates a new vector called name with the value specified by expr, an expression as described above. If expr is [] (a zero-length vector) then the vector becomes undefined. If there are no arguments, let is the same as display.

print [col] [line] expr ...

Prints the vector described by the expression expr. If the col argument is present, print the vectors named side by side. If line is given, the vectors are printed horizontally. col is the default, unless all the vectors named have a length of one, in which case line is the default. The options width, length, and nobreak are effective for this command (see asciiplot). If the expression is all, all of the vectors available are printed. Thus print col all > file will print everything in the file in SPICE2 format. The scale vector (time, frequency) will always be in the first column unless the variable noprintscale is true.

load [filename] ...

Loads the raw data in either binary or ascii format from the files named. The default filename is rawspice, or the argument to the -r flag if there was one.

source filename

Reads commands from the file filename. Lines beginning with the character * are considered comments and ignored.

help [all] [command ...]

Prints help. If the argument all is given, a short description of everything you could possibly type is printed. If commands are given, descriptions of those commands are printed. Otherwise help for only a few major commands is printed.

display [varname ...]

Prints a summary of currently defined vectors, or of the names specified. The vectors are sorted by name unless the variable nosort is set. The information given is the name of the vector, the length, the type of the vector, and whether it is real or complex data. Additionally, one vector will be labeled [scale]. When a command such as *plot* is given without a *vs* argument, this scale is used for the X-axis. It is always the first vector in a rawfile, or the first vector defined in a new plot. If you undefine the scale (i.e, *let TIME* = []), a random remaining vector will become the scale.

setplot [plotname]

Set the current plot to the plot with the given name, or if no name is given, prompt the user with a menu. (Note that the plots are named as they are loaded, with names like tran1 or op2. These names are shown by the setplot and display commands and are used by diff, below.) If the "New plot" item is selected, the current plot will become one with no vectors defined. Note that here the word "plot" refers to a group of vectors that are the result of one SPICE run. When more than one file is loaded in, or more than one plot is present in one file, nutmeg keeps them seperate and only shows you the vectors in the current plot.

settype type vector ...

Change the type of the named vectors to type. Type names can be found in the manual page for sconvert.

diff plot1 plot2 [vec ...]

Compare all the vectors in the specified *plots*, or only the named vectors if any are given. There are different vectors in the two plots, or any values in the vectors differ significantly the difference is reported. The variables abstol, reltol, and vntol are used to determine what "significantly" means (see the SPICE3 User's Manual).

quit Quit nutmeg.

bug Send a bug report. (If you have defined BUGADDR, the mail will go there.)

write [file] [exprs]

Writes out the expr's to file. First vectors are grouped together by plots, and written out as such. (I.e, if the expression list contained three vectors from one plot and two from another, then two plots will be written, one with three vectors and one with two.) Additionally, if the scale for a vector isn't present, it is automatically written out as well. The default format is ascii, but this can be changed with the set filetype command. The default filename is rawspice, or the argument to the -r flag on the command line, if there was one, and the default expression list is all.

shell [args ...]

Fork a shell, or execute the arguments as a command to the shell.

alias [word] [text ...]

Causes word to be aliased to text. History substitutions may be used, as in C-shell aliases.

unalias [word ...]
Removes any aliases present for the words.

history [number]
Print out the history, or the last number commands typed at the keyboard. *Note:* in version 3a7 and earlier, all commands (including ones read from files) were saved.

set [word] [word = value] ...
Set the value of word to be value, if it is present. You can set any word to be any value, numeric or string. If no value is given then the value is the boolean 'true'. The value of *word* may be inserted into a command by writing *$word*. If a variable is set to a list of values that are enclosed in parentheses (which must be seperated from their values by white space), the value of the variable is the list. The variables meaningful to **nutmeg** (of which there are too many) are:

abstol
The absolute tolerance used by the diff command.

appendwrite
Append to the file when a write command is issued, if one already exists.

color*N*
These variables determine the colors used, if X is being run on a color display. N may be between 0 and 15. Color 0 is the background, color 1 is the grid and text color, and colors 2 through 15 are used in order for vectors plotted. The value of the color variables should be names of colors, which may be found in the file **/usr/lib/rgb.txt**.

combplot
Plot vectors by drawing a vertical line from each point to the X-axis, as opposed to joining the points. Note that this option is subsumed in the *plottype* option, below.

cpdebug
Print *cshpar* debugging information. (Must be complied with the -DCPDEBUG flag.)

debug
If set then a lot of debugging information is printed. (Must be compiled with the -DFTEDEBUG flag.)

device
The name (/dev/tty??) of the graphics device. If this variable isn't set then the user's terminal is used. To do plotting on another monitor you will probably have to *set both the device and term variables. (If device is set to the name of a file, nut*meg will dump the graphics control codes into this file -- this is useful for saving plots.)

dontplot
No graphics control codes are actually sent. (Useful for debugging on non-graphics terminals.)

echo
Print out each command before it is executed.

filetype
This can be either ascii or binary, and determines what the format of *rawfiles* will be. The default is ascii.

fourgridsize
How many points to use for interpolating into when doing fourier analysis.

geometry*N*
The size and positioning information for X windows. *N* may be any positive integer, in which case it is the information for the *N*'th window on the screen, or may be omitted, in which case it is used whenever there is no information for the window. The geometry information is a string of the form *=heightxwidth+xoff+yoff*, where the window will be of size *height* by *width* and be positioned at (*xoff, yoff*), where (0,0) is the upper left hand corner of the screen. Either the positioning information or the size information may be omitted, in which case the window will be opened interactively (as will happen if no geometry information is given). The method of interactive sizing is the same as for other X utilities. A typical use for the geometry variables might be to set **maxwins** to 3 and set **geometry1, geometry2**, and **geometry3** to position three plot windows in a row across the top of the screen.

gridsize
If this variable is set to an integer, this number will be used as the number of equally spaced points to use for the Y-axis when plotting. Otherwise the current scale will be used (which may not have equally spaced points). If the current scale *isn't strictly monotonic, then this option will have no effect.*

hcopydev If this is set, when the **hardcopy** command is run the resulting file is automatically printed on the printer named **hcopydev** with the command *lpr -Phcopydev -g file*.

height
The length of the page for asciiplot and print col.

history
The number of events to save in the history list.

maxwins
The maximum number of windows X should have on the screen at one time. If it has **maxwins** or more windows, it will begin re-using them for plots in an oldest-first manner.

nfreqs
The number of frequencies to compute in the fourier command. (Defaults to 10.)

nobreak
Don't have **asciiplot** and **print col** break between pages.

noasciiplotvalue
Don't print the first vector plotted to the left when doing an asciiplot.

noclobber
Don't overwrite existing files when doing IO redirection.

noglob
Don't expand the global characters '', '?', '[', and ']'. This is the default.*

nogrid
Don't plot a grid when graphing curves (but do label the axes).

nomoremode
If **nomoremode** is not set, whenever a large amount of data is being printed to the screen (e.g. the **print** or **asciiplot** commands), the output will be stopped every screenful and will continue when a carriage return is typed. If **nomoremode** is set then data will scroll off the screen without hesitation.

nonomatch
If **noglob** is unset and a global expression cannot be matched, use the global characters literally instead of complaining.

nosort
Don't have display sort the variable names.

noprintscale
Don't print the scale in the leftmost column when a print col command is given.

numdgt
The number of digits to print when printing tables of data (fourier, print col). The default precision is 6 digits. On the VAX, approximately 16 decimal digits are available using double precision, so numdgt should not be more than 16. If the number is negative, one fewer digit is printed to ensure constant widths in tables.

plottype
This should be one of *normal*, *comb*, or *point*:chars. *normal*, the default, causes points to be plotted as parts of connected lines. *comb* causes a comb plot to be done (see the description of the *combplot* variable above). *point* causes each point to be plotted seperately - the chars are a list of characters that will be used for each vector plotted. If they are omitted then a default set is used.

polydegree
The degree of the polynomial that the plot command should fit to the data. If *polydegree* is N, then nutmeg will fit a degree N polynomial to every set of N points and draw 10 intermediate points in between each endpoint. If the points aren't monotonic, then it will try rotating the curve and reducing the degree until a fit is achieved.

polysteps
The number of points to interpolate between every pair of points available when doing curve fitting. The default is 10. (This should really be done automatically.)

program
The name of the current program (*argv[0]*).

prompt
The prompt, with the character '!' replaced by the current event number.

rawfile
The default name for rawfiles created.

reltol
The relative tolerance used by the diff command.

rhost
The machine to use for remote SPICE-3 runs, instead of the default one. (See the description of the rspice command, below.)

rprogram
The name of the remote program to use in the rspice command.

savewins
If true, then don't get rid of the X window after the plot is done (X only). The window may be removed by clicking any mouse button inside of it.

slowplot
Stop between each graph plotted and wait for the user to type return before continuing.

sourcepath
A list of the directories to search when a source command is given. The default is the current directory and the standard spice library (*/usr/local/lib/spice*, or whatever LIBPATH is #defined to in the source.

spicepath
The program to use for the aspice command. The default is /cad/bin/spice.

term
The *mfb* name of the current terminal.

units
If this is degrees, then all the trig functions will use degrees instead of radians.

unixcom
If a command isn't defined, try to execute it as a UNIX command. Setting this option has the effect of giving a rehash command, below. This is useful for people who want to use nutmeg as a login shell.

verbose
Be verbose. This is midway between echo and debug / cpdebug.

vntol
The absolute voltage tolerance used by the diff command.

width
The width of the page for asciiplot and print col.

xbrushheight
The height of the brush to use if X is being run.

xbrushwidth
The width of the brush to use if X is being run.

xfont
The name of the X font to use when plotting data and entering labels. The plot may not look entirely great if this is a variable-width font.

unset [word] ...
Unset the variables word.

shift [varname] [number]
If *varname* is the name of a list variable, it is shifted to the left by *number* elements. (I.e, the *number* leftmost elements are removed.) The default *varname* is argv, and the default *number* is 1.

rusage [resource ...]
Print resource usage statistics. If any resources are given, just print the usage of that resource. Currently valid resources are:

elapsed
The amount of time elapsed since the last rusage elapsed call.

faults
Number of page faults and context switches (BSD only).

space
Data space used.

time
CPU time used so far.

everything
All of the above.

cd [directory] Change the current working directory to directory, or to the user's home directory if none is given.

aspice [output-file]
Start a SPICE-3 run, and when it is finished load the data. The raw data is kept in a

temporary file. If *output-file* is specified then the diagnostic output is directed into that file, otherwise it is thrown away.

Jobs Report on the asynchronous SPICE-3 jobs currently running. Nutmeg checks to see if the jobs are finished every time you execute a command. If it is done then the data is loaded and becomes available.

rspice [input file] Runs a SPICE-3 remotely taking the input file as a SPICE-3 input deck, or the current circuit if no argument is given. Nutmeg waits for the job to complete, and passes output from the remote job to the user's standard output. When the job is finished the data is loaded in as with aspice. If the variable *rhost* is set, nutmeg will connect to this host instead of the default remote SPICE-3 server machine. Note that this command will only work if your system administrator is running a SPICE-3 daemon on the remote host. If the variable *rprogram* is set, then rspice will use this as the pathname to the program to run.

echo [stuff...] Echos the arguments.

fourier fundamental_frequency [value ...]
Does a fourier analysis of each of the given values, using the first 10 multiples of the fundamental frequency (or the first *nfreqs*, if that variable is set - see below). The output is like that of the .four card. The values may be any valid expression. The values are interpolated onto a fixed-space grid with the number of points given by the fourgridsize variable, or 200 if it is not set. The interpolation will be of degree polydegree if that variable is set, or 1. If polydegree is 0, then no interpolation will be done. This is likely to give erroneous results if the time scale is not monotonic, though.

version [version id]
Print out the version of nutmeg that is running. If there are arguments, it checks to make sure that the arguments match the current version of SPICE. (This is mainly used as a Command: line in rawfiles.)

rehash Recalculate the internal hash tables used when looking up UNIX commands, and make all UNIX commands in the user's PATH available for command completion. This is useless unless you have set unixcom first (see above).

The following control structures are available:

while *condition*
statement
...
end

While *condition*, an arbitrary algebraic expression, is true, execute the statements.

repeat [*number***]**
statement
...
end

Execute the statements *number* times, or forever if no argument is given.

dowhile *condition*
statement
...
end

The same as while, except that the *condition* is tested after the statements are executed.

foreach *var value ...*
statement
...
end

The statements are executed once for each of the *values*, each time with the variable *var* set to the current one. (*var* can be accessed by the *$var* notation - see below).

if *condition*
statement
...
else
statement
...
end

If the *condition* is non-zero then the first set of statements are executed, otherwise the second set. The **else** and the second set of statements may be omitted.

label *word*

If a statement of the form **goto** *word* is encountered, control is transfered to this point, otherwise this is a no-op.

goto *word*

If a statement of the form **label** *word* is present in the block or an enclosing block, control is transfered there. Note that if the label is at the top level, it *must* be before the goto statement (i.e, a forward goto may occur only within a block).

continue

If there is a **while**, **dowhile**, or **foreach** block enclosing this statement, control passes to the test, or in the case of **foreach**, the next value is taken. Otherwise an error results.

break

If there is a **while**, **dowhile**, or **foreach** block enclosing this statement, control passes out of the block. Otherwise an error results.

Of course, control structures may be nested. When a block is entered and the input is the terminal, the prompt becomes a number of >'s equalling the number of blocks the user has entered. The current control structures may be examined with the debugging command cdump.

If a word is typed as a command, and there is no built-in command with that name, the directories in the *sourcepath* list are searched in order for the file. If it is found, it is read in as a command file (as if it were sourced). Before it is read, however, the variables *argc* and *argv* are set to the number of words following the filename on the command line, and a list of those words respectively. After the file is finished, these variables are unset. Note that if a command file calls another, it must save its *argv* and *argc* since they will get altered. Also, command files may not be re-entrant since there are no local variables. (Of course, the procedures may explicitly manipulate a stack...) This way one can write scripts analogous to shell scripts for nutmeg and with a blank line (or whatever you like, since it will be thrown away) and then a line with .control on it. This is an unfortunate result of the source command being used for both circuit input and command file execution. Note also that this allows the user to merely type the name of a circuit file as a command, and it will be automatically run.

There are various command scripts installed in */usr/local/lib/spice/scripts* (or whatever the path is on your machine), and the default *sourcepath* includes this directory, so you can use these command files

(almost) like builtin commands.

Nutmeg will use either X or MFB, depending on whether it finds the variable **DISPLAY** in the environment. If you are using X on a workstation, it should already be present, but if you want to display graphics on a different machine than the one you are running **nutmeg** on, **DISPLAY** should be of the form *machine*:0.

If X is being used, the cursor may be positioned at any point on the screen when the window is up and characters typed at the keyboard will be added to the window at that point. The window may then be sent to a printer using the xpr(1) program.

There are a number of pre-defined constants in **nutmeg**. They are:

pi	pi
e	The base of natural logarithms
c	The speed of light
i	The square root of -1
kelvin	Absolute 0 in Centigrade
echarge	The charge on an electron
boltz	Boltzman's constant
planck	Planck's constant (h)

These are all in MKS units. If you have another variable with a name that conflicts with one of these then it takes precedence.

Nutmeg occasionally checks to see if it is getting close to running out of space, and warns the user if this is the case. (This is more likely to be useful with the SPICE front end.)

C-shell type quoting with ' and ", and backquote substitution may be used. Within single quotes, no further substitution (like history substitution) is done, and within double quotes, the words are kept together but further substitution is done. Any text between backquotes is replaced by the result of executing the text as a command to the shell.

Tenex-style ('set filec' in the 4.3 C-shell) command, filename, and keyword completion is possible: If EOF (control-D) is typed after the first character on the line, a list of the commands or possible arguments is printed. (If it is alone on the line it will exit **nutmeg**.) If escape is typed, then **nutmeg** will try to complete what the user has already typed. To get a list of all commands, the user should type <space> ^D.

The values of variables may be used in commands by writing **$varname** where the value of the variable is to appear. The special variables **$$** and **$<** refer to the process ID of the program and a line of input which is read from the terminal when the variable is evaluated, respectively. If a variable has a name of the form **$&word**, then word is considered a vector (see above), and its value is taken to be the value of the variable. If *$foo* is a valid variable, and is of type list, then the expression *$foo[low-high]* represents a range of elements. Either the upper index or the lower may be left out, and the reverse of a list may be obtained with *$foo[len-0]*. Also, the notation *$?foo* evaluates to 1 if the variable *foo* is defined, 0 otherwise, and *$#foo* evaluates to the number of elements in *foo* if it is a list, 1 if it is a number or string, and 0 if it is a boolean variable.

History substitutions, similar to C-shell history substitutions, are also available - see the C-shell manual page for all of the details.

The characters ~, {, and } have the same effects as they do in the C-Shell, i.e., home directory and alternative expansion. It is possible to use the wildcard characters *, ?, [, and] also, but only if you unset noglob first. This makes them rather useless for typing algebraic expressions, so you should set noglob again after you are done with wildcard expansion. Note that the pattern [^abc] will match all characters except a, b, and c.

IO redirection is available - the symbols >, >>, >&, >>&, and < have the same effects as in the C-shell.

You may type multiple commands on one line, seperated by semicolons.

If you want to use a different mfbcap file than the default (usually ~cad/lib/mfbcap), you have to set the environment variable **MFBCAP** before you start nutmeg. The -m option and the **mfbcap** variable no longer work.

VMS NOTES

Nutmeg can be run under VAX/VMS. Some features like command, etc completion, expansion of *, ?, and [], backquote substitution, the shell command, and so forth do not work. (In fact command completion only works on 4.2 or 4.3 BSD.)

Nutmeg will look for start-up commands in the file *spice.rc* in the current directory.

The standard suffix for rawspice files in VMS is ".raw".

You will have to respond to the *-more-* prompt during plot with a carriage return instead of any key as you can do on UNIX.

SEE ALSO

sconvert(1), spice(1), mfb(3), writedata(3)

AUTHOR

Wayne Christopher (faustus@cad.berkeley.edu)

BUGS

The label entry facilities are very primitive - after all, nutmeg isn't a graphics editor (yet). You must be careful to type very slowly when entering labels -- nutmeg checks the X event queue once every second, and can get very confused if characters arrive faster than that.

If you redefine colors after creating a plot window with X, and then cause the window to be redrawn, it will not to the right thing.

When defining aliases like

alias pdb plot db('!:1' - '!:2')

you must be careful to quote the argument list substitutions in this manner. If you quote the whole argument it might not work properly.

In a user-defined function, the arguments cannot be part of a name that uses the *plot.vec* syntax. I.e,

define poke(duck) cos(tran1.duck)

won't do the right thing.

If you type plot all all, or otherwise use a wildcard reference for one plot twice in a command, bad things will happen.

The asciiplot command doesn't deal with log scales or the delta keywords.

There are probably some features that nutmeg doesn't have yet.

CAVEATS

Often the names of terminals recognised by MFB are different from those in /etc/termcap. Thus you may have to reset your terminal type with the command

set term = termname

where termname is the name in the mfbcap file.

The hardcopy command is useless on VMS and other systems without the plot command, unless the user has a program that understands *plot*(5) format.

Appendix L

A short introduction to using NUTMEG with SPICE2G-SB2

The following pages give a reproduction of the introductory document to the NUTMEG program, distributed by U.C. Berkeley,[7] as it has been modified to correspond to use with the SPICE2G6-SB2 program on a PC.

A Short Introduction to Using Nutmeg with SPICE2G6-SB2

Adapted by Kenneth H. Carpenter
from the version by Wayne Christopher of
CAD Group
U. C. Berkeley
for SPICE3 release a7, March 20, 1986

June 22, 1989

Nutmeg is a post-processor for SPICE3: it takes data that SPICE3 generates and plots it on a graphics terminal. Nutmeg can be used in the same way with data from SPICE2, version G6, if the raw data output is first run through the sconvert program to change it from "old binary format" to either "ascii" or "new binary" format. See the manual page for sconvert for details.

[Note for PC users – SPICE and nutmeg were originally written to be run under the BSD Unix operating system. The term "manual page" refers to the Unix user's manual, which is available on-line on such systems. For the PC, a file named, e.g., sconvert.man, contains the information that would be in the Unix manual page for sconvert.]

The contents of the SPICE2G6 raw data file differs from that of the SPICE3 raw data file in that with SPICE2G6 all available data is written to the raw data file regardless of any commands in the SPICE input file. In particular, the data from a transient run does not depend on the TSTART or TSTEP values on the .TRAN control statement. If data would be interpolated to obtain all the values implied by TSTEP then this interpolation must be done within nutmeg to obtain results like those from SPICE3. Also, if TSTART differs from zero, the data from zero to TSTART will be included in the raw data file from SPICE2G6. And, finally, if the numerical method uses steps smaller than TSTEP the raw data file from SPICE2G6 will contain values for each step. If all this data is not to be plotted, then nutmeg commands must be used to select just the points desired.

If you run SPICE2G6-SB2 on an MSDOS PC the raw data file will be output on Fortran unit 15, and the file specification will be taken from the third argument on the command line. [However, if more than one analysis is produced by the run of SPICE, each analysis will have its raw data output to a different file, taking the file specifications from successive arguments on the command line after the third for the additional raw data files.]

The simplest way to use nutmeg on a PC is with the SPDRIVER command. The SPDRIVER batch file automatically names the raw data file, executes sconvert on it, and submits it to the PC version of nutmeg. If it is desired to run nutmeg directly, then, on the PC one gives the command:

```
nutmeg filename
```

to load the raw data in `filename` into nutmeg. If `filename` is not present, nutmeg attempts to load the default filename `rawspice`. Several raw data files can be loaded into nutmeg at the same time. This can be done by placing more than one filename on the command line when invoking the program, or by using the "load" command within it. With several raw data files loaded into nutmeg at once, plots can be made on the same axes of results from different analyses. See the nutmeg manual pages for details of how to do this.

Nutmeg processes sets of data refered to as "vectors". After loading a raw data file into the program, each of the node voltages and source currents are initially available as vectors. The set of vectors coming from a single SPICE analysis is called a "plot". The most recently read raw data file yields the "current plot" until the current plot is changed by a command. [Do not confuse this use of "plot" with the command to actually produce plots on the screen.]

Commands in nutmeg

The most important command in nutmeg is the `plot` command. If you type

```
plot varname
```

where `varname` is the name of one of the outputs in the file (such as `TIME`), nutmeg will plot a graph of the variable on the screen. Nutmeg will print the names of the variables on the bottom line of the screen as they are plotted. When it prints `-more-` it is waiting for you to hit a key before proceeding to something else. You can specify many variables in a `plot` command, like

```
plot TIME v(1) v(2) VWB
```

You can also specify combinations of outputs and functions of them, as in

```
plot v(1) + 2 * v(2)
```

or

```
plot log(v(1)) sin(cos(v(2)))
```

Notice that the variable name `v(1)` is not a function, but rather denotes the voltage at the node named 1. (If there is a variable with a number as a name, like 5, you can name it by writing "5", or if the type is voltage, as `V(5)`.) You can use most algebraic functions, including

trig functions, `log()`, log base 10, `ln()`, log base e, and functions such as `mag()`, the magnitude of the complex number, `phase()`, the phase, `real()`, the real part, and `imag()`, the imaginary part. These all operate on real or complex values. You cannot plot complex values, so should you use these functions to specify what parts of the complex values you want to plot. (The default for complex values is to plot the real part.) A complete list of functions and operations available can be found in the manual page for nutmeg.

The notation

```
plot something vs something_else
```

means to plot something with something_else on the X-axis.

Also, you can modify the `plot` command to plot a subset of the data available. The command

```
plot v(1) v(2) ylimit 1 2
```

will plot the two vectors when the values are between 1 and 2, and

```
plot v(1) v(2) xlimit 1 2
```

will plot them when the scale (time or frequency) is between 1 and 2. The command

```
plot v(1) v(2) xcompress 5
```

plots only every fifth point, and

```
plot v(1) v(2) xindices 20 30
```

plots the values between the 20th time point and the 30th. Any of these keywords may be used together, and they are also available in the `asciiplot` and `hardcopy` commands.

A few other commands are useful – the `display` command prints out the variables that you have available, along with some information about them. You can leave nutmeg with the `quit` command.

You can create new variables with the `let` command, as in

```
let hodedo = v(1) + 2 * v(2)
```

(The `let` command is different from the `set` command, which sets non-vector variables like `width`, `nosort`, and `prompt`.)

You can print variables with the `print` command:

```
print TIME
```

The command


```
print all
```

will print the values of all the data available. Incidentally, you can also use the keyword `all` with any of the other commands that take vector names, like `plot`.

There are also alias and history mechanisms available (see the manual page for details), and a `shell` command, which passes its arguments to the Unix shell (if you are using Unix), or starts a subshell.

Ascii plots are also available, with the `asciiplot` command. The command

```
asciiplot v(1) v(2) log(v(3)) > file
```

creates an ascii plot in the file, `file`. If you want a different size plot, use the `set` command to reset the sizes, like

```
set width=80 height=14
```

The default values are 130 and 60, suitable for a line printer. Normally every `height` lines there is a page break, but if you do

```
set nobreak
```

this will be suppressed.

Hard copies of plots just as shown on the screen can be made by using the `hardcopy` command to produce a graphics metafile in the Unix "plot" format. This metafile can be processed with utility programs to reproduce the screen image on a variety of graphics devices. The format of the command is

```
hardcopy filespec plotargs
```

where `filespec` is the path to and name for the metafile, and `plotargs` are the same as to the `plot` command.

To see what other commands are available, look at the manual page, or type `help` in nutmeg. [Typing "help" will be of no "help" on a PC unless the help tree-structure of files has been loaded onto the PC's disk. This structure is not normally distributed with SPICE2G6-SB2. Refer to the manual pages for nutmeg instead.] This introduction should be enough to get you started. A sample run follows.

Sample run of nutmeg

In the following sample run of nutmeg user input is in italics, computer output in typewriter style, and comments on what is happening in standard printing.:

[% is the Unix prompt - on a PC it would be replaced by, e.g., C:>]

[Lines not applicable on a PC are preceded by an asterisk.]

% *nutmeg*

```
Title:  SPICE 3-C raw output test heading
Name:       Transient analysis.
Date: 08/19/8403:17:11
```

nutmeg 1 -> *display* Here are the variables currently active:

```
Title: SPICE 3-C raw output test heading
Plotname:       Transient analysis.
Date: Sun Dec 1 11:18:25 PST 1985
TIME                    : time (real, 152 long) [scale]
v(1)                    : voltage (real, 152 long)
v(2)                    : voltage (real, 152 long)
v(3)                    : voltage (real, 152 long)
v(4)                    : voltage (real, 152 long)
v(5)                    : voltage (real, 152 long)
```

nutmeg 2 -> *print line v(1)*

```
 v(1) = (  0  0  0  0  0  0  0  0  0  0.010663  0.031989  0.074641
0.159945  0.330552  0.671767  1.354197  2  2  2  2  2
```

(and so forth)

```
0  0  0  0  0  0  0  0  )
```

nutmeg 3 -> *plot v(1)*

(plot takes place)

nutmeg 4 -> *let zzz = log(v(1))* nutmeg 5 -> *plot zzz xlimit 10n 20n*

(plot takes place)

nutmeg 6 -> *plot v(1) v(2) v(3) + 1 vs TIME * 2*

(plot takes place)

*nutmeg 7 -> *bug* Enter a description of the bug. End with ^D.

```
*This works only if you can mail directly to faustus@ic.berkeley.edu
```

* *Great program!!*

*^D Bug report sent. Thank you.

nutmeg 8 -> *asciiplot v(1) v(2) TIME + 2 > File*

*nutmeg 9 -> *shell lpr File* [on PC, send file to printer]

(Pick up ascii plot ...)

nutmeg 10 -> *help* [Output from "help" is implementation dependent]

nutmeg 11 -> *quit* So long.

%

See the manual pages for nutmeg for a complete list of commands and a more detailed description of how to use the program. [Not all of the commands described in the manual are available or applicable on a PC. These should be evident from the context, or from the error messages produced if one attempts to use "Unix only" commands on a PC.]

Appendix M

Listing of a program to test switch model equations

```
/*switch2.c - KHC - 20FEB90*/
/*Normalized equations for modeling a switch as a constant voltage
 *drop in an RLC circuit.
 *KHC 20FEB90 - from S.Warren program
 */

#include <stdio.h>
#include <math.h>
#include <conio.h>

#define PI 3.1415926536

main()
{
int  choose,j,quit,time;
double alpha,G,i,k,L,one,r_one,r_two,t,two,Vo,Vs,w;
FILE *fp;
char file[20];

time = 1;

while (1)
{
printf("Enter the item of interest (i, G, K):");
choose = getche();
putchar('\n');
switch(choose){
case 'i':
case 'I':
choose = 1;
break;
case 'g':
case 'G':
choose = 2;
break;
case 'k':
```

```
case 'K':
choose = 3;
break;
default:
exit(0);
}
if (time == 1)
{
printf("Enter switch values -  \n");
printf("Vs: ");
scanf("%lf",&Vs);
printf("alpha: ");
scanf("%lf",&alpha);
    time = 2;
        }
w = sqrt(1. - alpha*alpha);
Vo = 1. - Vs;

if (choose == 1)
printf("What is the name of the i output file?\n");
if (choose == 2)
printf("What is the name of the G output file?\n");
if (choose == 3)
printf("What is the name of the K output file?\n");
scanf("%s",file);
fp = fopen(file,"w");
fprintf(fp,"alpha = %f,  Vs = %f\n",alpha,Vs);

for (j=1;j<=200;j++)
{
     t = (double)j*(PI/w)/200;
i = (Vo/w)*exp(-alpha*t)*sin(w*t);
one = (-1.0/alpha) +  (alpha*cos(2*w*t) - w*sin(2*w*t));
two = 0.25*(1./alpha - alpha/4. +exp(-2.0*alpha*t)*(one));
G = Vo/w;
G *= G;
G *=two;
k = Vs*sqrt(G)/i;

if (choose == 1)
fprintf(fp,"%e %e \n",t,i);
     if (choose == 2)
fprintf(fp,"%e %e \n",t,G);
     if (choose == 3)
fprintf(fp,"%e %e \n",t,k);
}
fclose(fp);
}
}
```

Appendix N

An Empirical Study of Magnetiztion in Ferrites Excited by Fast Rise Time Pulses

Abstract

In order to gain a better understanding of the behavior of ferrite cored magnetic couplers when the driving currents contain frequency components above the rating for the ferrite, we have measured the B-H curve for several ferrite materials using fast rise time current pulses to provide the H field. When the current rise time is made sufficiently low, one obtains the DC magnetization curve for the material. As the current rise time is increased the magnetic response of the ferrite appears to undergo a time delay relative to the current. From a sequence of increasing rise time measurements one should be able to obtain a characteristic time for a given ferrite core which can then be incorporated into a model for use in circuit simulation. It is hoped that this approach will provide a model more relevant to magnetic couplers used for high speed transient signals than one that would be based on the hysteresis loops for different steady state sinusoidal excitations.

Data will be presented for measurements of dynamic magnetization curves for Ferroxcube ferrite types 3C8, which is indicated in the manufacturer's data to be usable up to 100KHz; 4C4, which is indicated to be usable up to 20MHz; and 3E2, which is indicated to be usable up to 200MHz.

N.1 Ferrite limitations

Ferrite materials are designed for use over a limited frequency range and for a limited amplitude of magnetic flux density. When either of these limits is exceeded in a transformer core, strange behavior may be expected. When using a ferrite core to enhance coupling for thin film transformers, which were of reasonably high coupling coefficient even when air-cored, the effects of exceeding the design limits on the ferrite are hard to predict. Simulation of circuits containing such transformers is needed to allow optimization of design of systems, and simulation requires adequate modeling of the components. Thus we have undertaken to obtain experimental data, appropriate to this application, for typical ferrites. This data will allow us to evaluate existing models for ferromagnetism and to derive improved models for their behavior when driven into saturation with fast rise time pulses.

N.2 Experiment requirements

The experiment must achieve both saturation of the ferrite and also have a fast enough rise time for magnetic field intensity to exceed the rated frequency response. Rise time and frequency response can

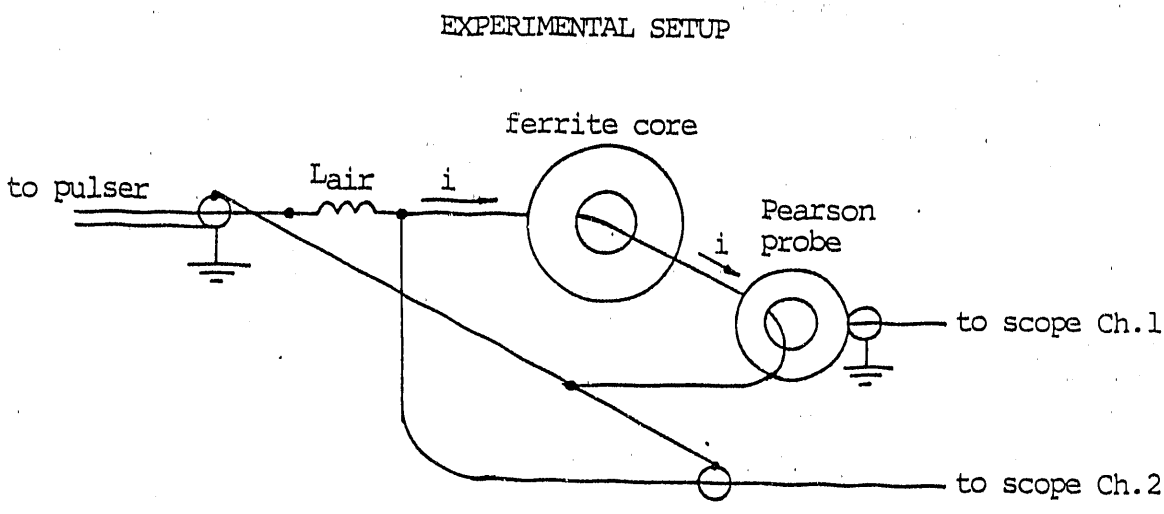


Fig. 1

be related for a sinusoidal field with an amplitude approaching the saturation value. However, we have chosen to use a fast pulse system to achieve the needed excitation of the ferrite.

The experimental set up consists of a mercury pulser having a fast switching ability, along with a charge line of sufficient length to produce a pulse of enough volt-seconds to saturate the ferrite's inductance. The pulser is connected through a delay line to a winding of a few turns (often just a single turn) through a toroidal core. The voltage across the winding is measured with a digitizing oscilloscope, which has a 50 Ohm input that serves to terminate the delay line. The current through the core is determined by means of a current transformer, called a Pearson probe, which is also connected to the digitizing oscilloscope. These are illustrated in Figs. 1 and 2.

N.3 Explanation of pulse application

The action of the pulser on the ferrite is fairly complex, and requires an analysis of transmission lines under transient conditions to be understood. The initial charge on the charge line connected to the switch in the pulser splits into waves of half the charging voltage moving to the right and the left when

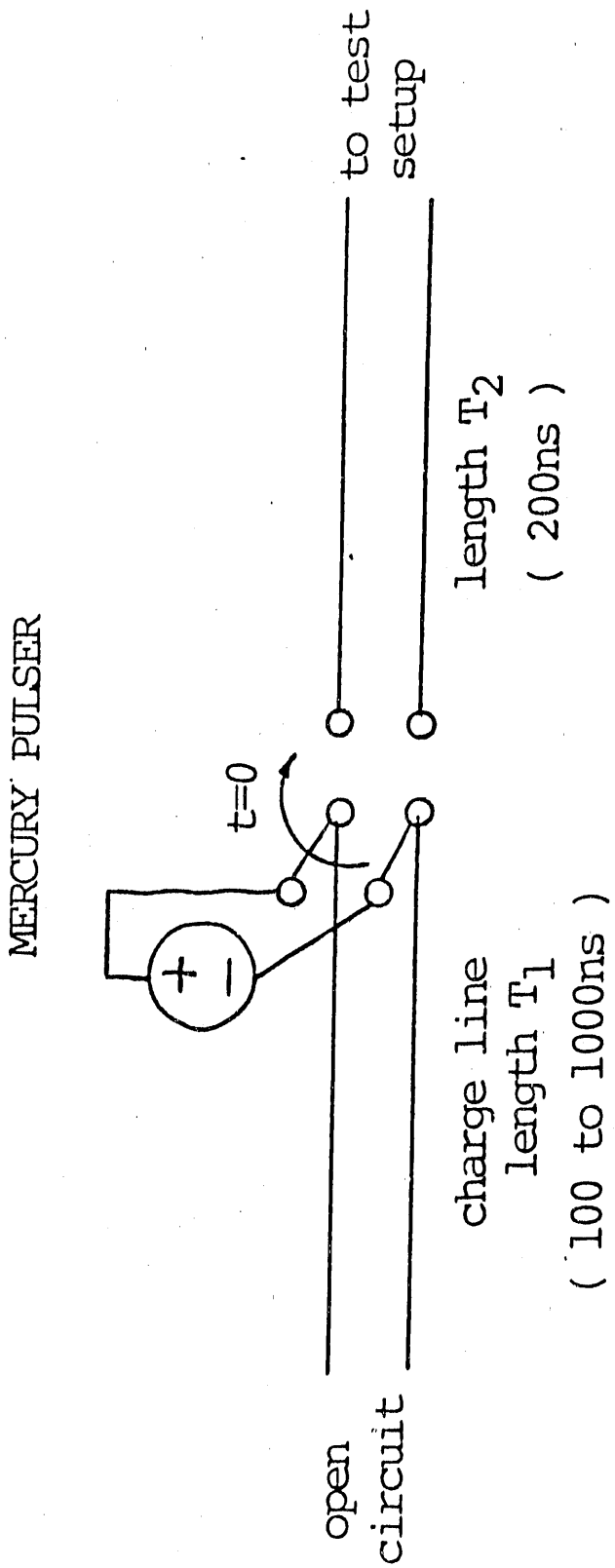

MERCURY PULSER
t=0
to test setup
length T_2
(200ns)
charge line
length T_1
(100 to 1000ns)
open circuit

Fig. 2

the switch is closed. The left moving wave is reflected at the open circuit end of the charge line, so the pulse applied to the delay line has half the amplitude but twice the length of the charge line initial values. The rise time of this pulse is extremely fast, since the inductance and capacitance that would slow the rise time in simple circuits are all included in the transmission line itself. The rise time is essentially limited by the switch closing characteristics, and for the pulser used, was faster than the fastest digitising interval of the oscilloscope.

When the fast rising pulse of voltage reaches the end of the delay line, if only the 50 Ohm terminating resistor were present, it would be totally absorbed by the resistor, and the voltage across the resistor would be just a square pulse. With an inductance in parallel with the resistor, having an initial zero current, the current through it must remain zero initially – hence it is just as if only the resistor were present. However, as the voltage remains on the inductor, its integral with respect to time gives a current through the resistor, and this current causes the delay line to behave as if its load were reduced below the characteristic impedance, which in turn causes a negative reflection of part of the voltage. Eventually, the entire current flows through the inductor with the voltage across the resistor dropping to zero — until the next reflection from the other end of the delay and charge line reaches the inductor. This is shown (in Fig. 3) in a SPICE simulation of the set up, using a small linear inductor as the load.

The fast rise time obtained with the set up described above provides one data point for each ferrite material to be tested. To obtain other data points with slower rise times, fixed, air-cored inductors are placed in series between the end of the delay line and the winding through the ferrite. Using inductances between 1 and 40 microhenries enables driving the ferrites' magnetizations at rates from far faster than they can respond to rates where the response is nearly that of the static magnetization.

N.4 Experimental data

The set up described above was used to obtain data for three commercial ferrites – Ferroxcube 3C8, 3E2, and 4C4. The data to be presented here is for just three rise times, one, called "fast", which results with no air-cored inductance in series, one, called "slow", which results with 37μh inductance in series, and one, called "moderate", which results with 10μh inductance in series. The fast data shows the effect of driving the ferrite too fast for its magnetization to be able to follow, the slow data shows essentially the static characteristics, and the moderate data shows a case somewhat in between the other two.

N.4.1 Initial magnetization and hysteresis curves

First note the initial magnetization and hysteresis curves obtained with the slow rise time for the magnetic field intensity. These show the differences in the three ferrites in their saturation levels and permeabilities at below the usual frequency limits. [These are shown in Figs. 4-6.]

Next, note the way these magnetization curves change when the initial current rise is faster. [Cases for 3C8 only are shown in Figs. 7-9.] Only the initial magnetization is affected since on subsequent parts of the curve the rate of change of current is much slower.

The magnetic field intensity versus time is presented for the fast, moderate, and slow cases for 3C8 material in Figs. 10-12. These figures can be related to the magnetization curves to see how fast the field intensity is changing on each part.

N.4.2 Rate of rise of field and flux

Plots have been made (on the same axes) of the initial rise of magnetic field intensity and magnetic flux density for each of the materials for each of the rise times. Only the three curves for the 3C8 material are shown here (in Figs. 13-15). We can see from these curves how the flux rate of increase cannot keep up with the rate of increase of the field as it becomes faster and faster. This is the effect that must be modeled to supplement the modeling of the saturation effect in order to obtain adequate circuit simulations.

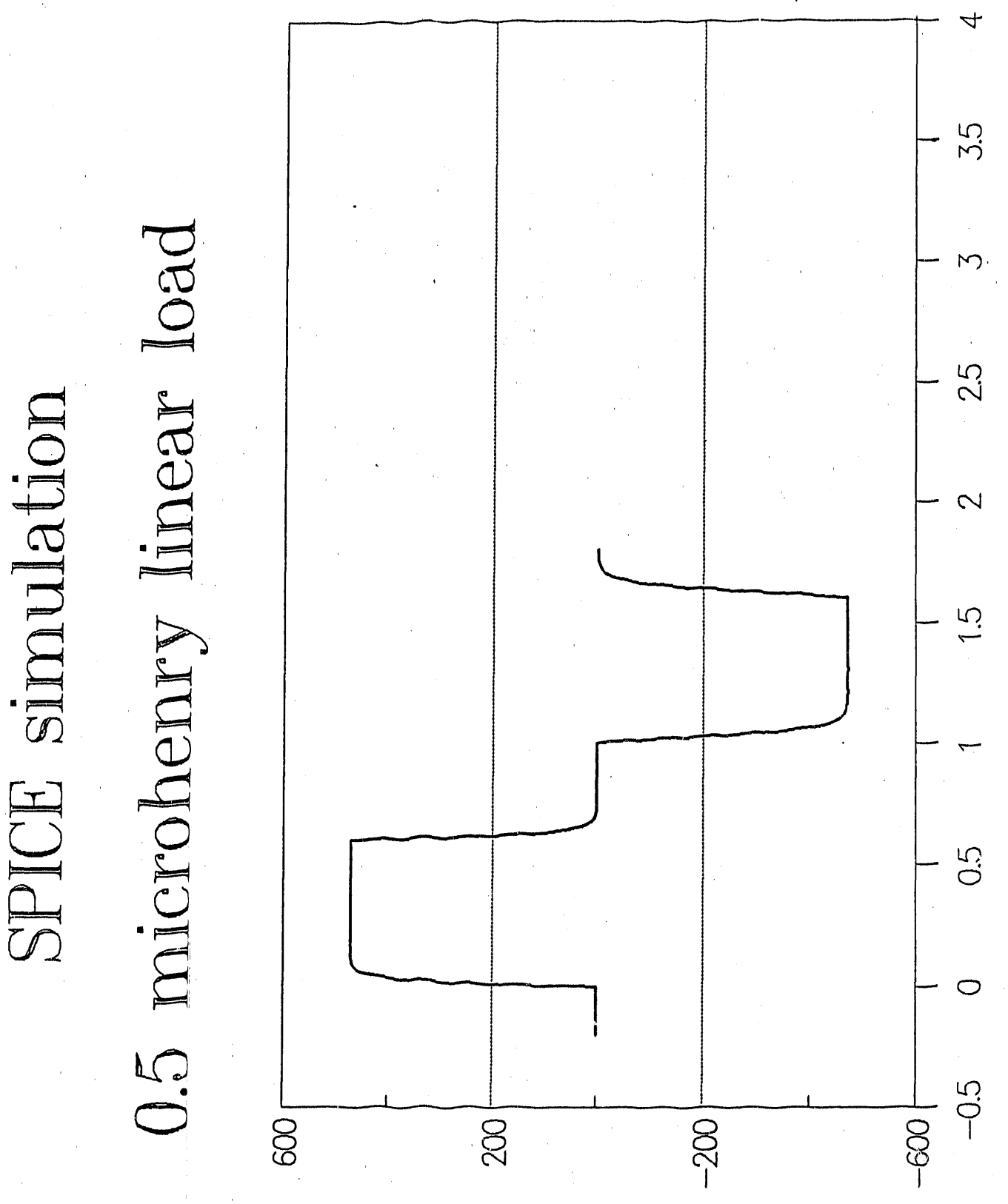

SPICE simulation
0.5 microhenry linear load
H – A/m
600
200
−200
−600
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time – microseconds

Fig. 3

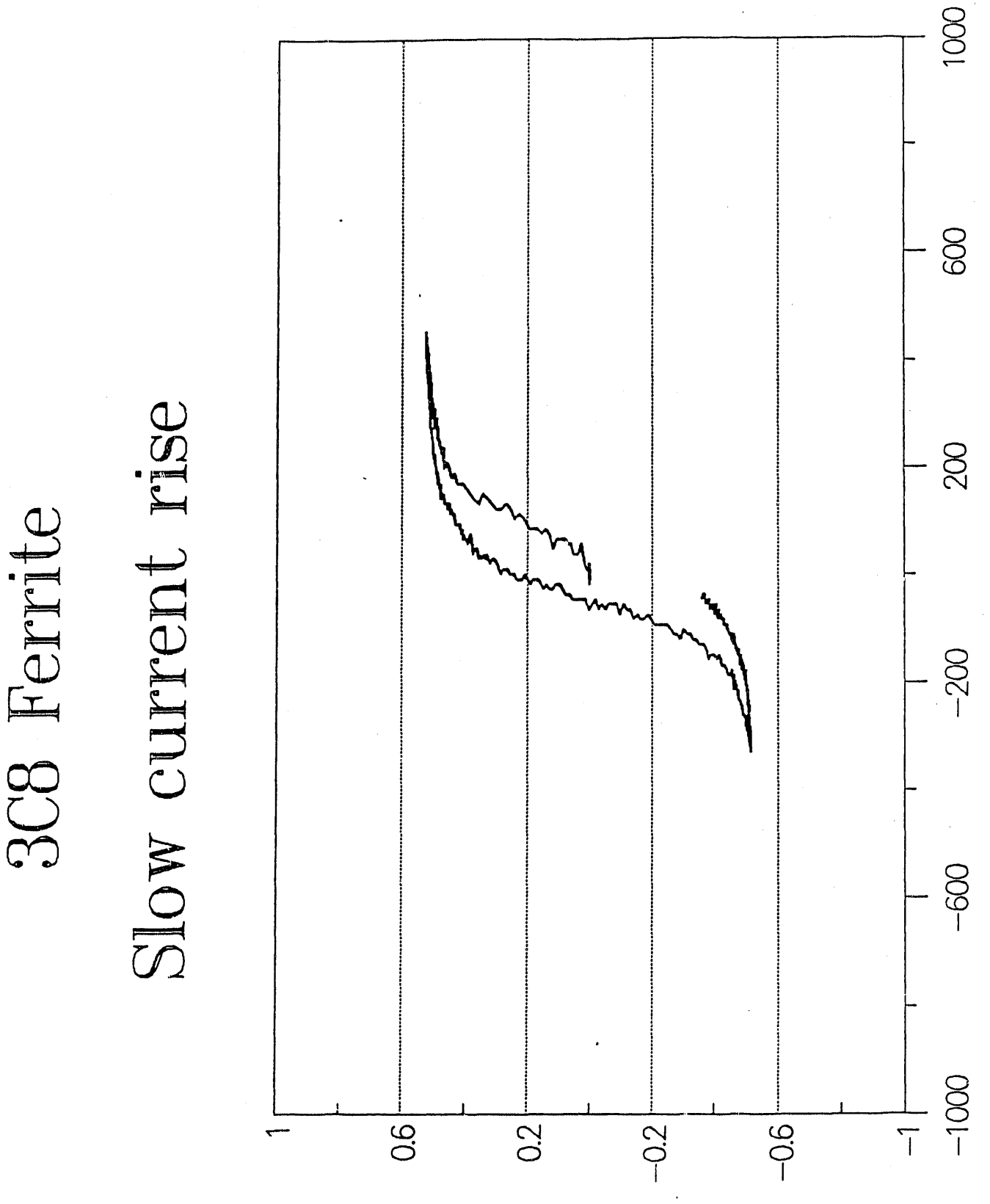

3C8 Ferrite
Slow current rise
1
0.6
0.2
−0.2
−0.6
−1
1000
600
200
−200
−600
−1000
B − Tesla
H − A/m

Fig. 4


3E2 Ferrite
Slow current rise
B – Tesla
1
0.6
0.2
−0.2
−0.6
−1
1000
600
200
−200
−600
−1000
H – A/m

Fig. 5

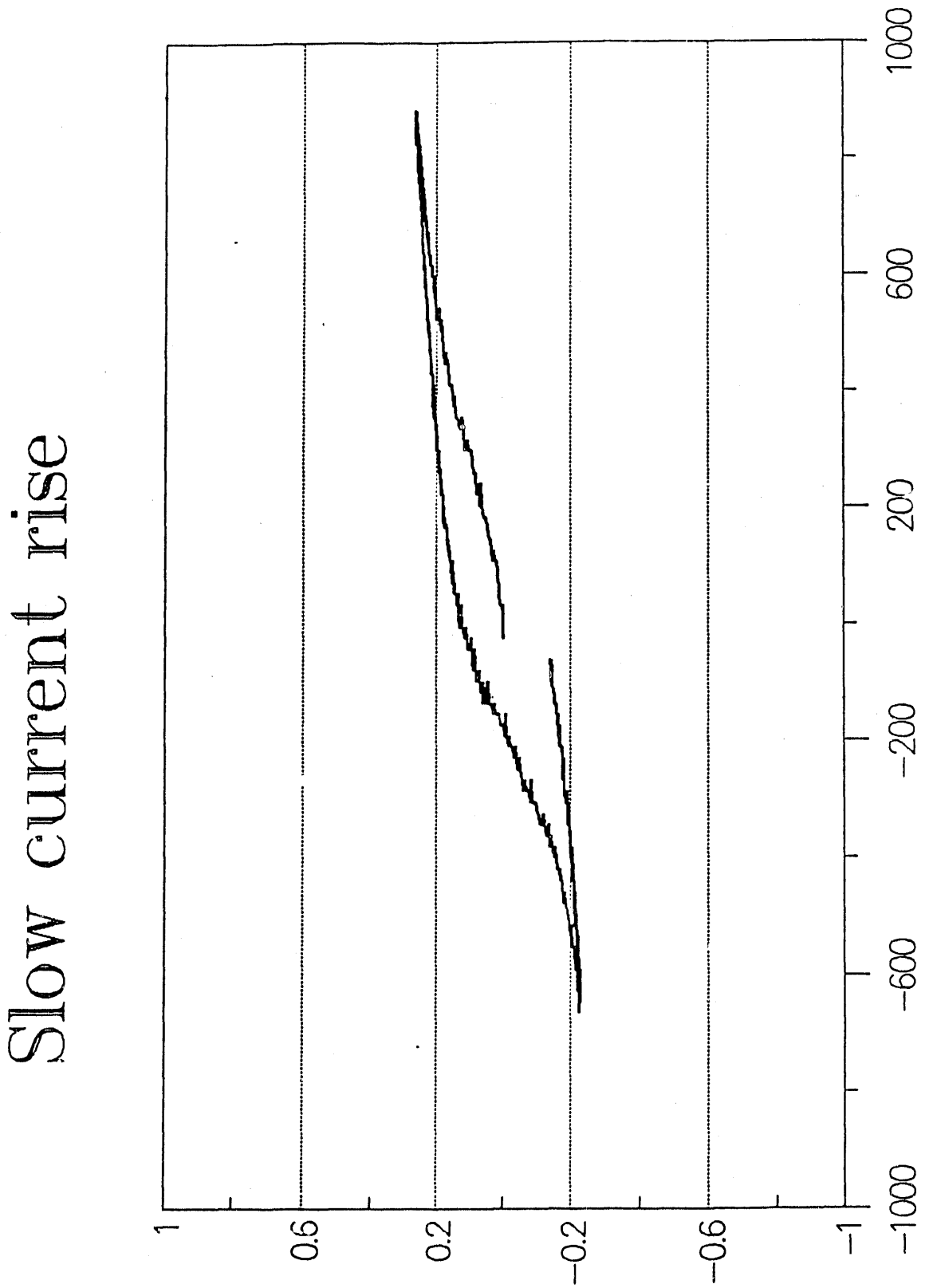

4C4 Ferrite
Slow current rise
B – Tesla
1
0.6
0.2
-0.2
-0.6
-1
-1000
-600
-200
200
600
1000
H – A/m

Fig. 6

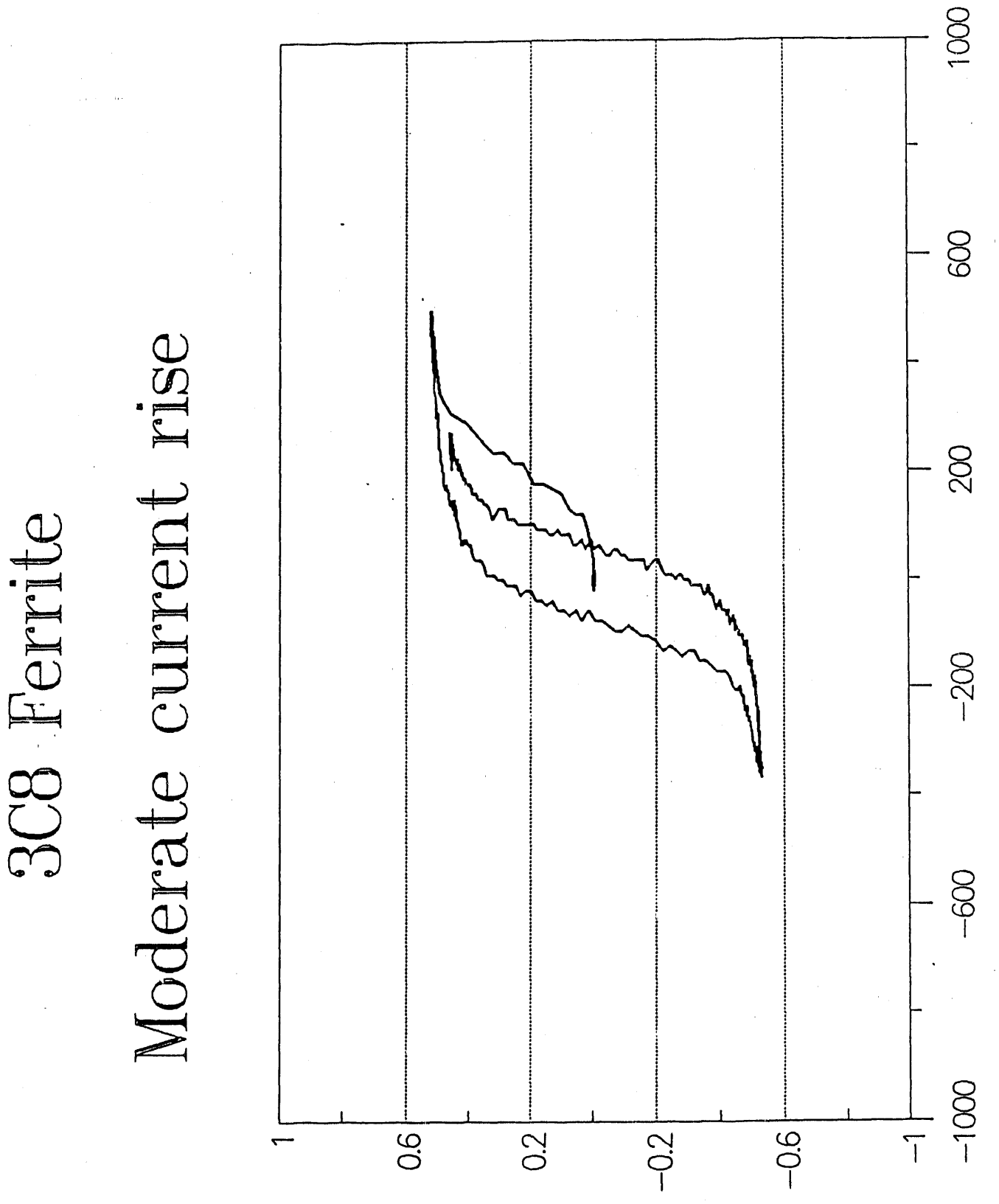

3C8 Ferrite
Moderate current rise
1000
600
200
−200
−600
−1000
H − A/m
1
0.6
0.2
−0.2
−0.6
−1
B − Tesla

Fig. 7

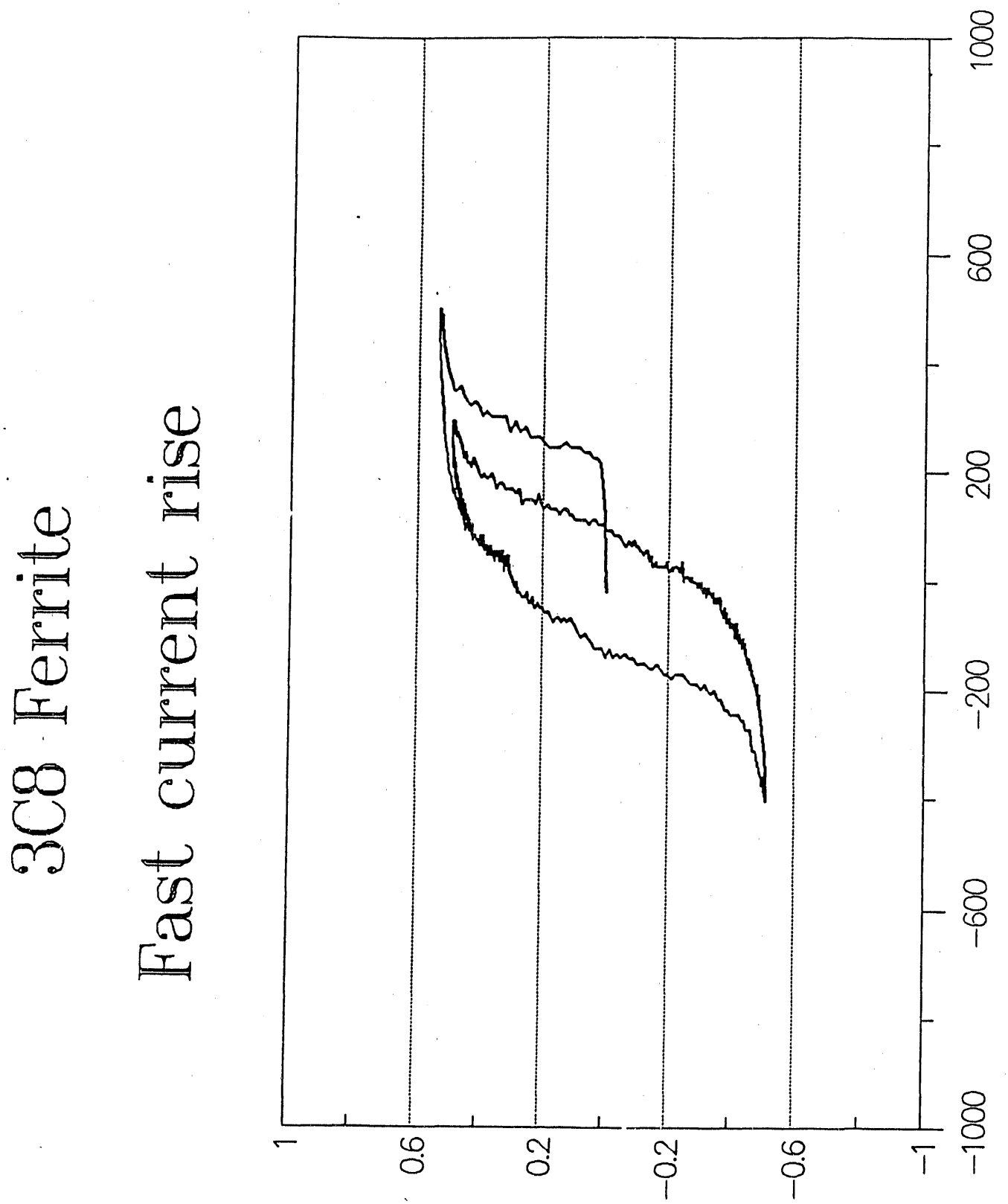

3C8 Ferrite
Fast current rise
B – Tesla
H – A/m
1
0.6
0.2
−0.2
−0.6
−1
1000
600
200
−200
−600
−1000

Fig. 8

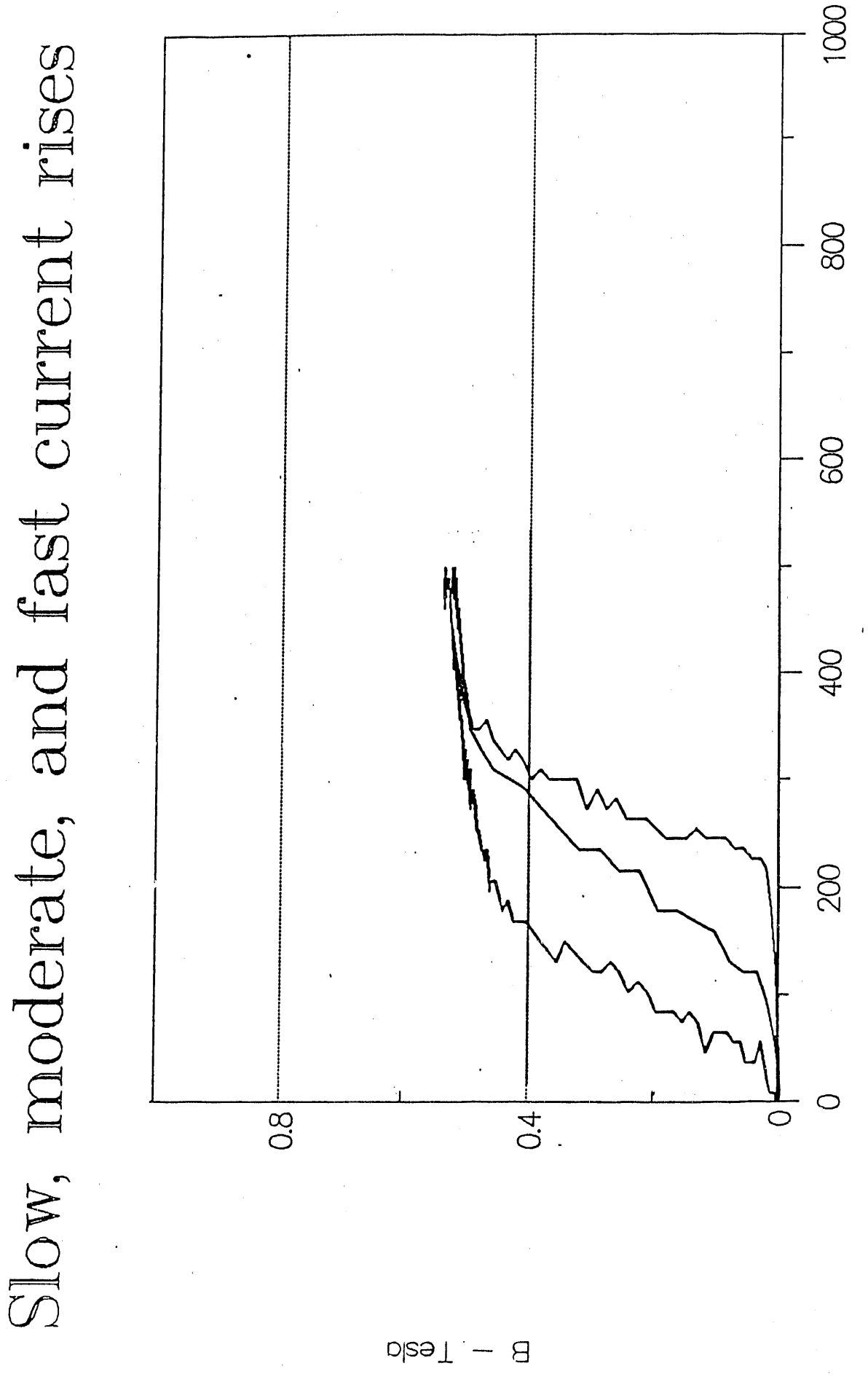

Slow, moderate, and fast current rises
B — Tesla
0.8
0.4
0
0
200
400
600
800
1000
3C8 Ferrite:
H — A/m

Fig. 9

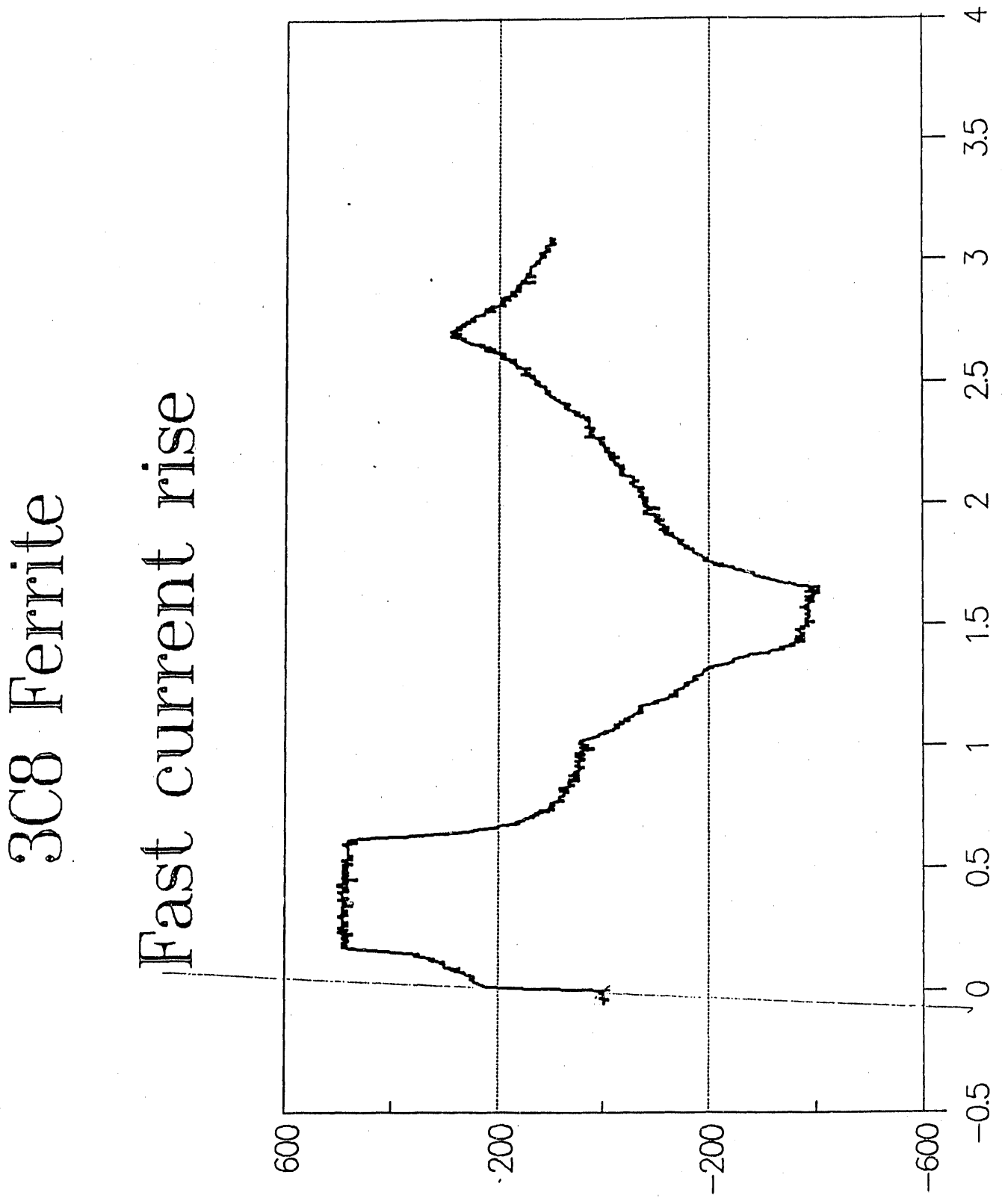

3C8 Ferrite
Fast current rise
600
200
-200
-600
H - A/m
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time - microseconds

Fig. 10

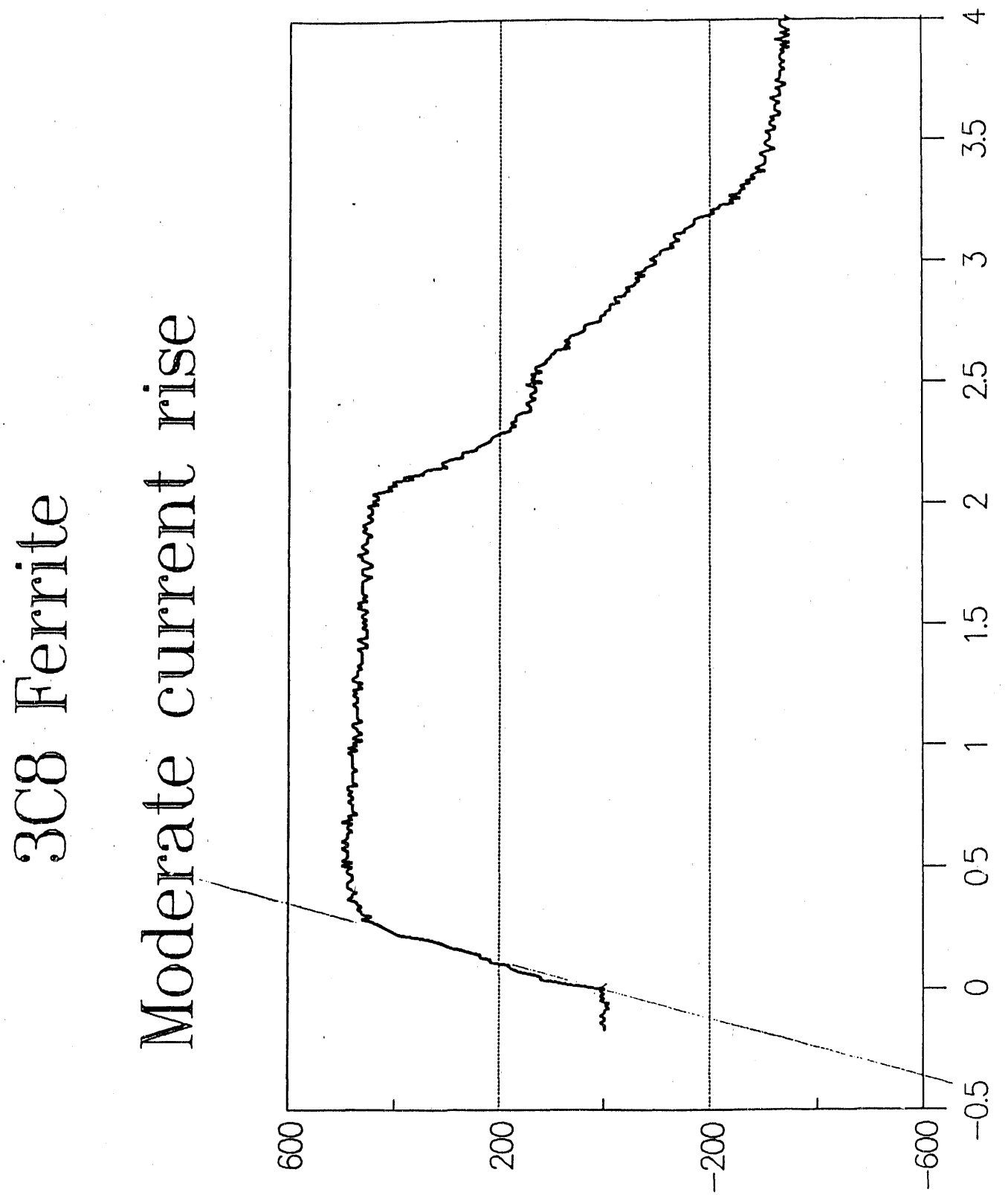

3C8 Ferrite
Moderate current rise
H – A/m
600
200
−200
−600
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time – microseconds

Fig. 11

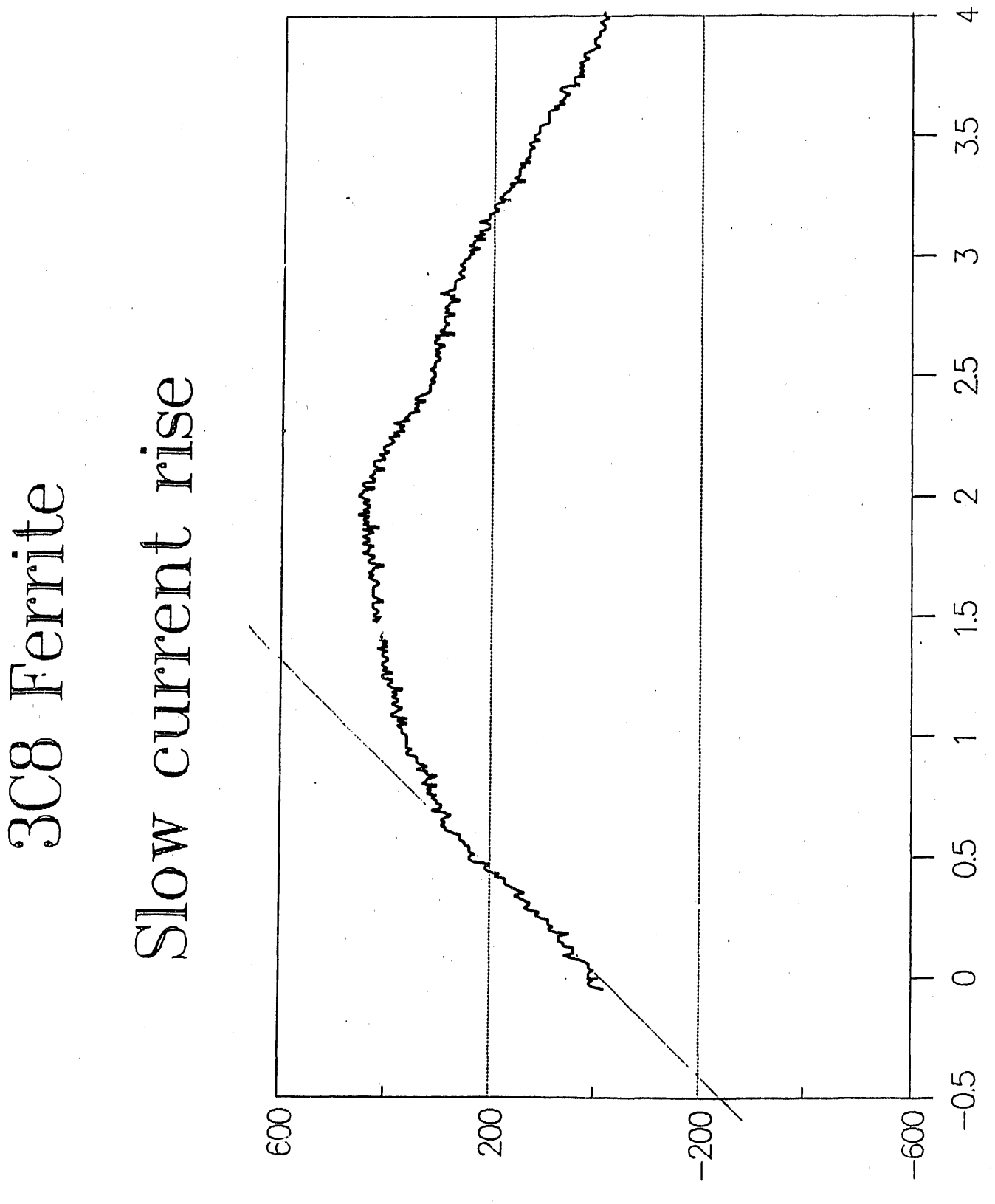

3C8 Ferrite
Slow current rise
H – A/m
600
200
−200
−600
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time – microseconds

Fig. 12

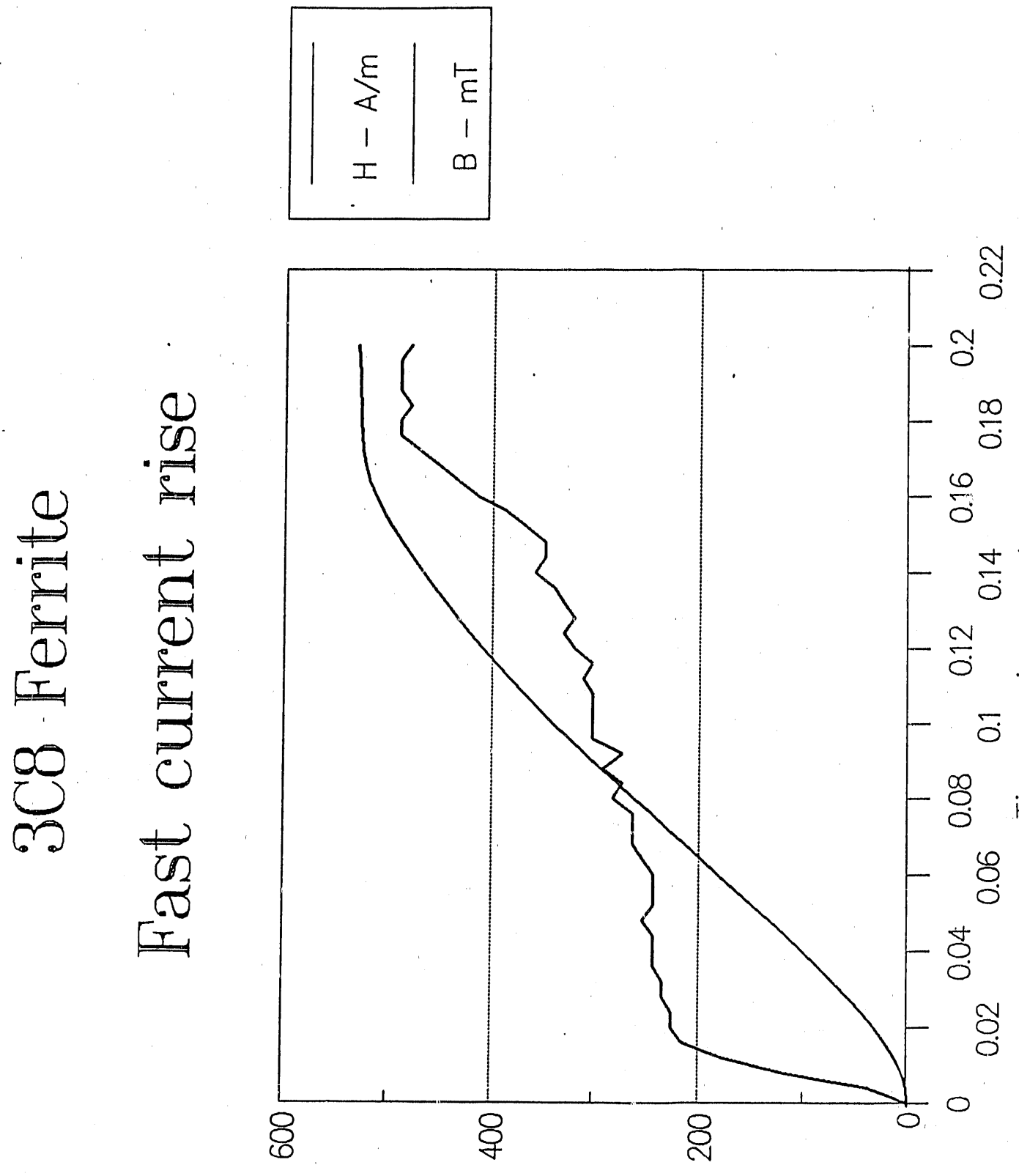

3C8 Ferrite
Fast current rise
H – A/m
B – mT
600
400
200
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22
Time – microseconds

Fig. 13

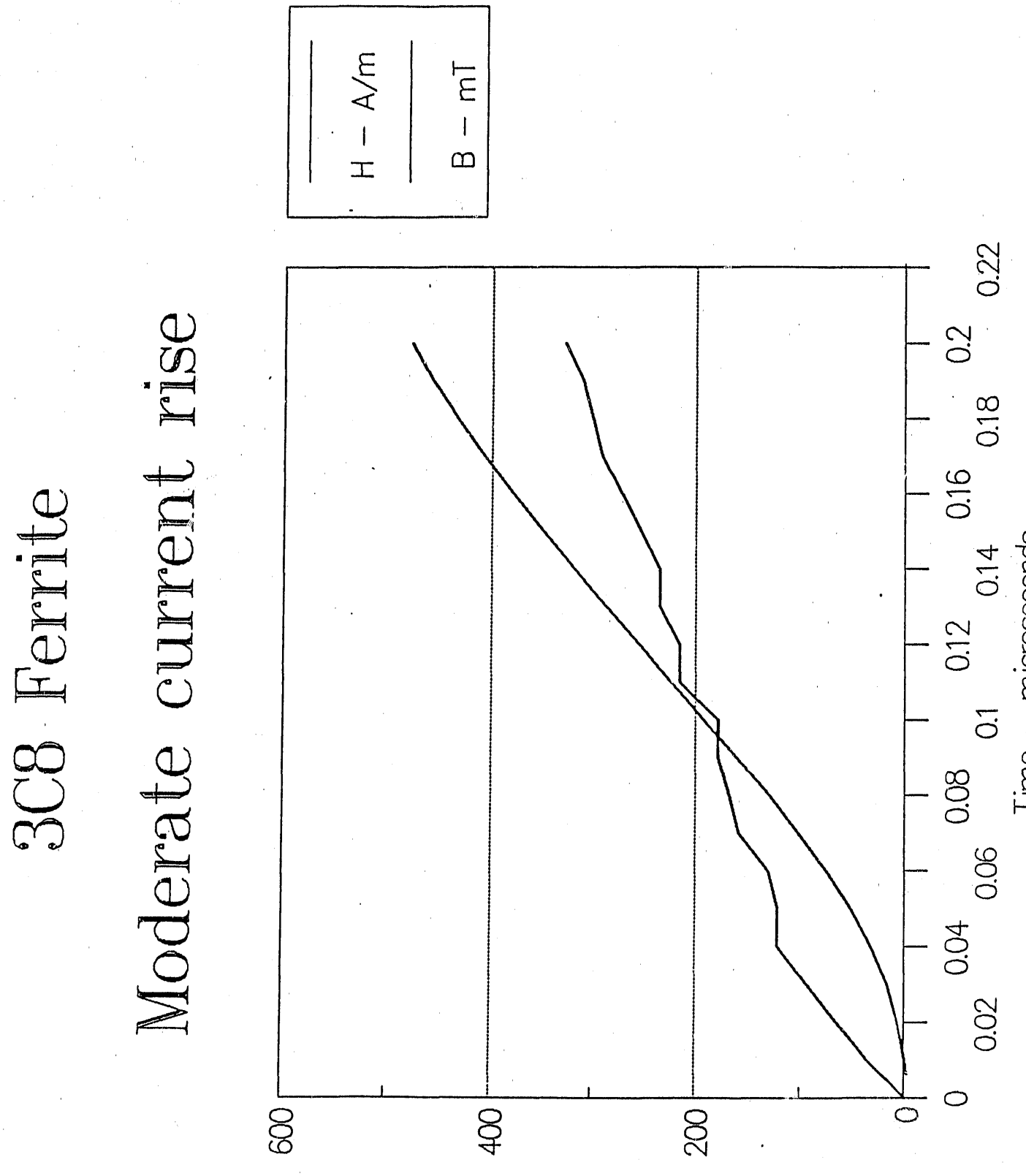

3C8 Ferrite
Moderate current rise
H – A/m
B – mT
600
400
200
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22
Time – microseconds

Fig. 14

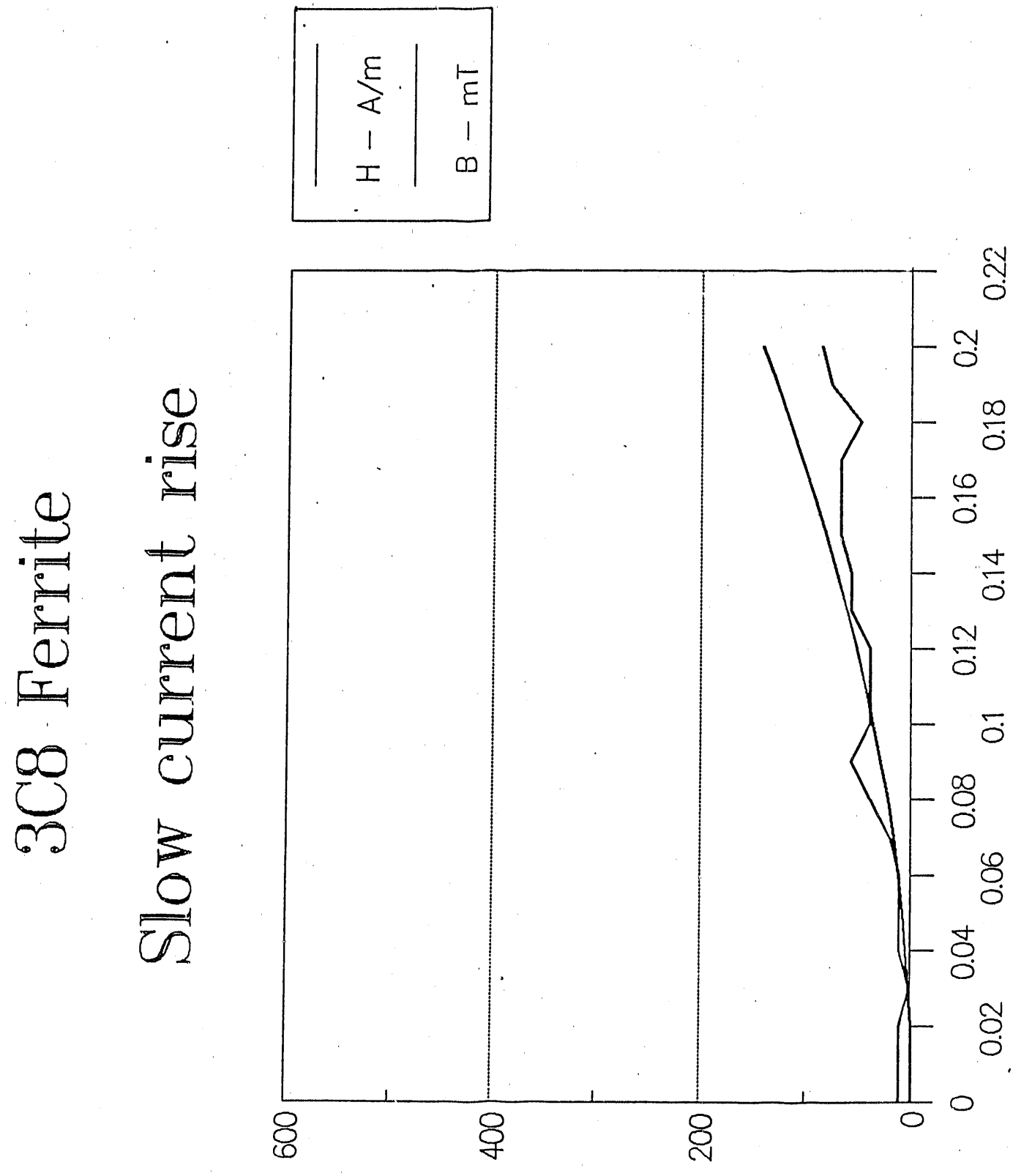

3C8 Ferrite
Slow current rise
H – A/m
B – mT
600
400
200
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22
Time – microseconds

Fig. 15

```
SPICE INPUT FOR 3C8 SIMULATION - FAST CURRENT RISE

Inductor test simulation with saturating inductance
RBIG 1 0 1E+010
T1 1 0 2 0 Z0 50 TD 2E-007 IC 2000 0 2000 0
L0 2 3 1E-009
T2 3 0 5 0 Z0 50 TD 2E-007
VIL 5 6 DC 0
L1 6 0 POLY -1234 3.25E-006 1E-009 18.3 3
RL 5 0 50
GINT 0 10 6 0 1
CINT 10 0 1
RINT 10 0 1000000
.TRAN 1E-007 2E-006 0 5E-009 UIC
.END
```

Table 1

```
SPICE INPUT FOR 3C8 SIMULATION - SLOW CURRENT RISE

Inductor test simulation with saturating inductance
RBIG 1 0 1E+010
T1 1 0 2 0 Z0 50 TD 1E-006 IC 2000 0 2000 0
L0 2 3 1E-009
T2 3 0 4 0 Z0 50 TD 2E-007 IC 0 0 0 0
LA 4 5 37E-06
VIL 5 6 DC 0
L1 6 0 POLY -1234 3.25E-006 1E-009 18.3 3
RL 5 0 50
GINT 0 10 6 0 1
CINT 10 0 1
RINT 10 0 1000000
.TRAN 1E-007 4E-006 0 1E-008 UIC
.END
```

Table 2

N.5 SPICE simulations of saturation without rise time effects

The special version of the SPICE circuit simulation code (a modification of the original version 2G6 by UC Berkeley) that was produced to allow modeling the saturation in ferrites, but as yet does not model the fast rise time effects, has been used to simulate the experimental situation.

The 3C8 ferrite, formed into a (nominal) 1 inch toroid, has a low frequency inductance, for small signals, of about 3 μh. This, along with the saturation current observed, allowed setting the model parameters. (The input for the SPICE runs is shown in Tables 1 and 2.) The result of simulating the "fast" and "slow" rise times is interesting. In the fast case, the H versus time plot is not the same as the experimental one, indicating the need to take the delayed response of the ferrite into account. However, in the slow case the simulation follows the experimental data quite closely. (See the plots of H vs time for both simulations in Figs. 16 and 17 – and superpose them with the experimental cases for comparison.)

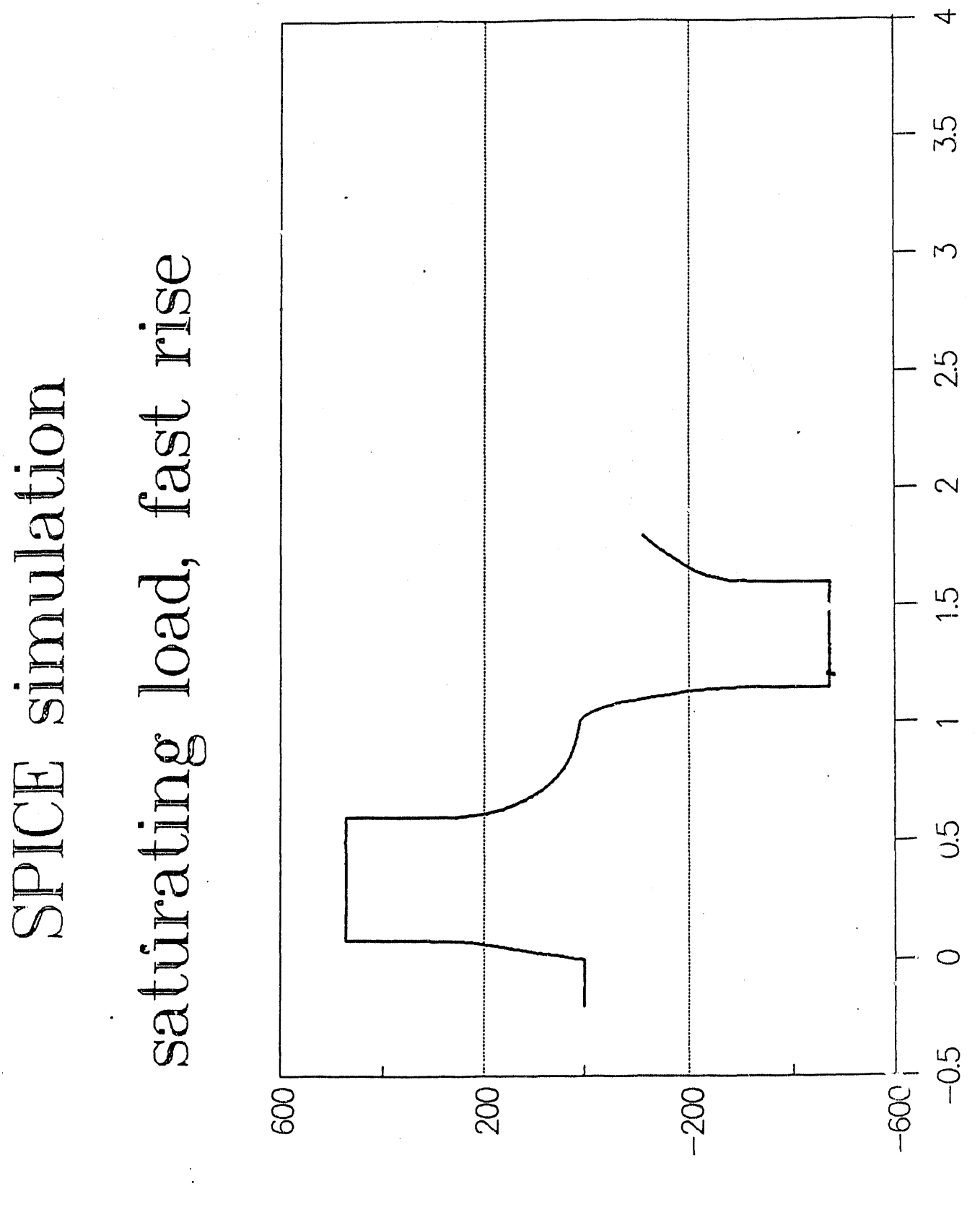

SPICE simulation
saturating load, fast rise
H – A/m
600
200
−200
−600
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time – microseconds

Fig. 16

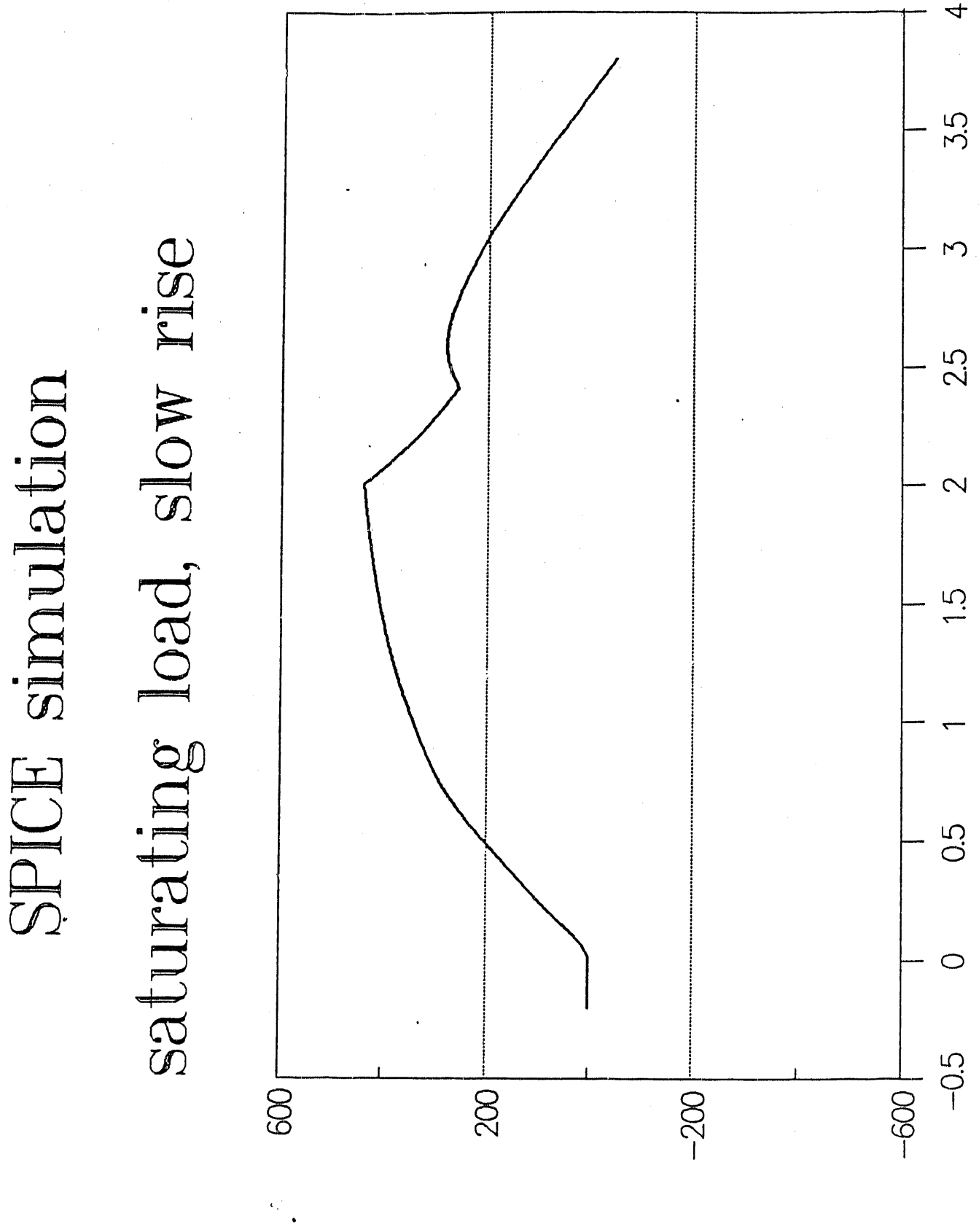

SPICE simulation
saturating load, slow rise
H – A/m
600
200
-200
-600
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time – microseconds

Fig. 17

N.6 Summary – and work to be done

We have obtained much data on ferrites in addition to that presented here. [Some of this data still needs to be analyzed.] There are still other experimental configurations to use to give additional data relevant to the intended use of the ferrites. These configurations will involve different shapes of pulses and different sizes and shapes of the ferrites themselves; but the results are expected to remain qualitatively the same as those given above.

The 4C4 ferrite has much lower permeability than the 3C8 and 3E2 varieties, and so does not show such a marked change in behavior on fast rise time (i.e., the behavior was not so good to begin with). The 3C8 and 3E2 suffer a time delay on the order of a hundred nanoseconds between the application of a magnetic field intensity and the production of magnetization. This delay needs to be modeled into circuit simulations in order to determine if it is a significant factor in the use of these materials.

Appendix O

Source listing of the ADIFNUT program

The program ADIFNUT was written to convert data files transferred from a Tektronix digital oscilloscope to an MSDOS PC into a form that could be plotted or further manipulated by NUTMEG or similar programs. A listing of the source code for ADIFNUT follows.

```
/*adifnut3.c - K.H.Carpenter - 12OCT89
 *Program to read binary files produced by a Tektronix 2430 digital
 *oscilloscope and convert the trace data to an output file of 1024
 *data points for input to the NUTMEG program.
 *
 *This version is for use with measurments of ferrite magnetization,
 *and so assumes two input files, the first for current, which has
 *been converted to voltage by an external device, and the second for
 *voltage, which must be integrated with respect to time to get flux.
 *(Experimental measurements performed by Steve Warren during summer,
 *1989, at LLNL - see his notes for details.)
 *
 *The program reads the scope vertical and horizontal scale settings
 *from the file headers, but ignores the offset information in the
 *headers, as it varies as to definition between versions of the
 *Tek software.  Instead, the program uses knowledge of the experiment
 *to automatically derive the offsets from the data.
 *
 *The program prompts for entry of external attenuation factors, and
 *for the geometry of the ferrite device under test.  The output file
 *is then constructed to contain magnetic flux density and magnetic
 *field intensity variables versus time.
 */
#include <stdio.h>
#include <string.h>

double atof(char *s);
void usage(char *me);
char brev(char in);
char curin[1024], volin[1024];
char curhead[1024], volhead[1024];
char seps[] = " |=()\n\r";
double flux[1024];
main(int argc, char **argv)
{

    int i, j, turns = 0;
    char pname[65], day[65];
    float dxx, fluxoff, dx, ox, scale, vscale, offset, voffset,
         atten = 0., vatten = 0., area = 0., length = 0.;
    FILE *incur, *invol, *out;
    char *token, *dim;
    if(argc != 4){
        usage(argv[0]);
        exit(1);
    }
    if((incur = fopen(argv[1],"rb")) == NULL){
        fprintf(stderr,"Can't open current file: %s\n",argv[1]);
        exit(1);
    }
    if((invol = fopen(argv[2],"rb")) == NULL){
        fprintf(stderr,"Can't open voltage file: %s\n",argv[1]);
        exit(1);
    }
    if((out = fopen(argv[3],"w")) == NULL){
        fprintf(stderr,"Can't open output file: %s\n",argv[2]);
        exit(1);
    }
    printf("Current file: %s\n",argv[1]);
    /*Now enter header data into array while printing it to screen.
     *No provision for checking for overflow of array space, since
     *for this experiment the headers are always less than 1024 chars.
     */
    j = 0;
    while((i=getc(incur))!='#'){putchar(i); curhead[j++] = i;}
    curhead[j] = 0;
    putchar(i);
    for(i=0;i<5;i++){
        putchar(getc(incur));
    }
    fread(curin, 1, 1024, incur);
    while((i=getc(incur))!= EOF)putchar(i);

    printf("\nVoltage file: %s\n",argv[2]);
    j = 0;
    while((i=getc(invol))!='#'){putchar(i); volhead[j++] = i;}
    volhead[j] = 0;
    putchar(i);
    for(i=0;i<5;i++){
        putchar(getc(invol));
    }
    fread(volin, 1, 1024, invol);
    while((i=getc(invol))!= EOF)putchar(i);

    /*Data has been read; find offsets.  Since there are 256 pretrigger
     *samples for all these experimental files, and since the trigger
     *occurs soon after the voltage begins to rise from zero, use the
     *first 200 samples to establish the offsets.
     */
    for(j = 0, voffset = 0., offset = 0.; j < 200; j++){
        voffset += volin[j];
        offset  += curin[j];
    }
    voffset /= 200;
    offset  /= 200;

    /*Find time zero as first sample where maximum of first 200 readings
     *is doubled (after offset is subtracted).
     */
    for(j = i = 0; j < 200; j++)if(volin[j] > i)i = volin[j];
    i = 2*(i - voffset);
    for(j = 0; j < 1024; j++)if((volin[j] - voffset) > i)break;
    ox = --j;

    /*Now read headers to find scale factors for current, voltage, and time.
     *Also pick up date information for output and check for consistency
     *between current and voltage files.
     */
    token = strtok(curhead,seps);
    if(strcmp(token,"ADIF")){puts("Bad current file - aborting"); exit(1);}
    while(token = strtok(NULL,seps)){
        if(!strncmp(token,"IDE",3)){
            strtok(NULL,seps);
            strcpy(day,strtok(NULL,seps));
```

```
            strcat(day," ");
            strtok(NULL,seps);
            strcat(day,strtok(NULL,seps));
            strcat(day,"\n");
        }
        if(!strncmp(token,"DIM",3)){
            dim = strtok(NULL,seps);    /*stores type of dim field*/
            while(strncmp((token = strtok(NULL,seps)),"SCA",3));
            token = strtok(NULL,seps);
            if(strncmp(dim,"X",1))scale = atof(token);
            else dx = atof(token);
        }
    }
    token = strtok(volhead,seps);
    if(strcmp(token,"ADIF")){puts("Bad voltage file - aborting"); exit(1);}
    while(token = strtok(NULL,seps)){
        if(!strncmp(token,"DIM",3)){
            dim = strtok(NULL,seps);    /*stores type of dim field*/
            while(strncmp((token = strtok(NULL,seps)),"SCA",3));
            token = strtok(NULL,seps);
            if(strncmp(dim,"X",1))vscale = atof(token);
            else dxx = atof(token);
        }
    }
    if(dxx != dx){
        puts("Time scale of current file different \
from voltage file - aborting");
        exit(1);
    }
    /*Now apply scale factors to offsets.*/
    voffse  *= vscale;
    offset *= scale;
    ox *= dx;

    /*Now print out calculated values ar  ask for values from
     *external data.
     */
    printf("\
          current scale = %g      current offset = %g\n\
          voltage scale = %g      voltage offset = %g\n\
          time scale    = %g      time offset    = %g\n",
        scale,offset,vscale,voffset,dx,ox);

    while(atten <= 0.){
        fputs("Enter external attenuation factor for current,\n\
including current to voltage conversion factor (from notes):",stdout);
        scanf("%g",&atten);
    }
    while(vatten <= 0.){
        fputs("Enter external attenuation factor for voltage (from notes):"
            ,stdout);
        scanf("%g",&vatten);
    }
    while(turns <= 0){
        fputs("Enter number of turns on torus:",stdout);
        scanf("%d",&turns);
    }
    while(area <= 0.){
        fputs("Enter cross sectional area of torus (square cm):",stdout);
        scanf("%g",&area);
    }
    while(length <= 0.){
        fputs("Enter effective path length of torus (cm):",stdout);
        scanf("%g",&length);
    }
/*Now change area and length for output factors:*/
    area = 10000./(turns*area);
    length = 100*turns/length;
/*Now do numerical integration of voltage by trapezoidal rule:*/
    flux[0] = 0.;
    dxx = 0.5 * dx * vscale * vatten;
    fluxoff = dx * voffset * vatten;
    for(i = 1; i < 1024; i++){
        flux[i] = flux[i-1] + dxx*(volin[i-1] + volin[i]) - fluxoff;
    }
    fputs("\nEnter \"Plot Name\" for NUTMEG:",stdout);
    fflush(stdin);
    pname[63] = 0;
    fgets(pname,65,stdin);
    if(pname[63]!=0)pname[63] = '\n';
    fflush(stdin);
/*We now have all the data, so write out in NUTMEG input format:*/
    fprintf(out,"Title: Current file: %s,  Voltage file: %s\n",argv[1],argv[2]);
    fprintf(out,"Date: %s",da,);
    fprintf(out,"Plotname: %s",pname);
    fputs("Flags: real\n",out);
    fputs("No. Variables: 5\n",out);
    fputs("No. Points: 1024\n",out);
    fputs("Command: version SPICE 3b1\n",out);
    fputs("Variables:\t0\tTIME\ttime\n\t\t1\t!CUR\tcurrent\n\
\t\t2\tVIND\tvoltage\n\t\t3\tB\tTesla\n\t\t4\tH\tA/m\nValues:\n",out);

    for(i = 0; i < 1024; i++){
        fprintf(out," %d\t%g\t%g\t%g\t%g\t%g\n",i,i*dx-ox,
            ((scale*curin[i])-offset)*atten,
            ((vscale*volin[i])-voffset)*vatten,
            flux[i]*area,
            ((scale*curin[i])-offset)*atten*length);
    }
}
void usage(char *me){printf("Usage: %s curfile volfile nutfile\n",me);}
```

END

DATE FILMED

02/21/91