MAKCAL: AN APPLICATION GENERATOR FOR ADVICE

Michael S. Toth

Michael S. Toth *is a member of technical staff in the Technology CAD Department at AT&T Bell Laboratories in Allentown (Cedar Crest), Pennsylvania. He is responsible for development of AT&T's proprietary analog-circuit simulator, ADVICE. Mr. Toth joined the company in 1986 and has a Bachelor of Music from the Eastman School of Music (Rochester, New York), a Master of Music from the University of Notre Dame (Indiana), and an M.S. in electrical engineering from the University of Houston (Texas).*

With the increased sophistication of analog-circuit simulation tools comes a heavier reliance on the interfaces to these tools. Today, designers can control simulation tools through programs—a technique referred to as ***procedural simulation***—and derive performance measures (e.g., common-mode rejection ratio, bandwidth, etc.) easily from more-basic circuit-simulation variables (i.e., voltages and currents). This technique has proved to be a valuable feature of AT&T's analog-circuit simulator, ADVICE. To enable designers to exploit procedural simulation more fully, a program called `makcal` was created. `Makcal` writes a procedural-simulation program that a designer can use to control ADVICE. `Makcal` includes the capability to select and present performance measures in a variety of graphic and textual formats. Because `makcal` streamlines the programming process, designers can concentrate more fully on designing, which reduces the product's time to market.

Introduction

In the past, circuit designers had to breadboard their circuits (i.e., build throwaway, hardware prototypes of circuits) from discrete components and try the circuits to find out why they did not work. However, changes to the design of an integrated circuit (IC) are not possible once the circuit is fabricated, and design problems are difficult to locate on fabricated ICs.

Because of the evolution toward integrated circuitry, the role of computer-aided design (CAD) has increased. CAD's goal is to ensure that a circuit will work before any attempt is made to manufacture that circuit. Thus, CAD has become indispensable for integrated-circuit design. (Panel 1 defines acronyms and terms used in this paper.)

The circuit simulator is a popular computer-aided-design program. To use such a program, a designer describes a circuit in terms of

Panel 1. Acronyms and Terms

ADVICE	AT&T's analog-circuit simulator
ALA210	AT&T linear array
CAD	computer-aided design
Center	a design-optimization system
CPU	central-processing unit
`HBcurrent`	routine created for ADVICE to compute high-bias leakage current
`IC`	integrated circuit
`ICAPP`	leakage current through a capacitor, computed by the ADVICE routine `HBcurrent`
`LBcurrent`	routine created for ADVICE to compute low-bias leakage current
`lex`	a program generator for lexical processing of character input streams
`makcal`	an application generator for ADVICE
SDTB	specification-driven tool builder
SPICE	computer program to simulate semiconductor circuits
STAGE	tool for building a language-oriented application generator; now called the MetaTool™ specification-driven tool-builder program
`tbl`	preprocessor for formatting and printing tables on the UNIX® system
`yacc`	yet another compiler-compiler; a tool for imposing structure on the input to a computer program

its resistors, capacitors, and similar components, and the simulator then finds the circuit's voltages and currents.

Frequently, the designer wants to know more about the circuit than simple voltages and currents. Several steps usually need be taken to compute useful circuit-performance measures, such as slew rate and common-mode rejection ratio, from the voltage and current information. As circuit simulators became more powerful, such results were more easily derived from the basic voltages and currents.

Simulators also became programmable. That is, we now could describe many manual steps in a suitable language and then use this description to control the simulator. This programmability is referred to as *procedural simulation*.

Procedural simulation is a solution to finding higher-level simulation results, but it can also be a problem. As simulation needs become more sophisticated, programming complexities can pose serious obstacles to getting the job done. Designers typically lack the time to create the programs they need to control the simulator. Also, debugging the program can be tedious, frustrating, and time consuming for someone without experience in programming. (*Debugging* is the task of finding and correcting a program's logic and syntax errors.)

This paper describes a program called `makcal` that is aimed at easing the programming burden that procedural simulation imposes on circuit designers in AT&T. `Makcal` writes a program that can control AT&T's circuit simulator, ADVICE.[1] Because `makcal` reduces the work involved in using ADVICE's procedural-simulation feature, a designer obtains simulation results more quickly and easily, which reduces the time to market for the final product.

ADVICE and the Procedural-Simulation Interface

ADVICE—a proprietary, analog-circuit simulator—has been used within AT&T for several years in the design of integrated circuits. ADVICE has evolved to its present state from our experiences with several earlier programs, including SPICE,[2] and with applications for production-IC designs. A full description of the simulator is beyond the scope of this paper.

The key feature of ADVICE that allows the use of application generators is its procedural-simulation capability. Procedural simulation permits us to automate both the commands to a simulator and the processing of simulation results.

In an interactive ADVICE session, a designer enters commands at the keyboard, views and interprets the results of the simulation, and makes decisions about the design based on this interpretation. The simulation results can be reformatted and processed in a variety of ways. All this work can be done manually, but much time is saved if the process can be automated.

With procedural simulation, designers can describe the steps that the simulator must take to produce the desired results. This permits several commands to be executed automatically, and the results to be post-processed in almost any imaginable way. Several functions have already been made available that obtain bandwidth, slew rate, phase margin, and many other results from the voltage and current information. (A *function* is a set of instructions that perform a specific task. The main or calling program may pass values to the function. When it completes its task, the function returns control, and possibly new values, to the calling program.)

The language used to describe the simulation session could be a custom language or could be taken from a preexisting source. For ADVICE, the choice was the C programming language (which was developed by AT&T Bell Laboratories).

The added power of procedural simulation also means added complexity of programming. For some simple simulation needs, the programs may not be highly complex. But for other cases, the programs become difficult to write and to debug. Syntax errors commonly appear even for simple jobs. Also, realistic time constraints may prevent deadlines from being met if programs become too complex.

Programs can increase in size to the point that it takes weeks to write and debug a single program. For example, this author has seen a program that was about 250 lines of executable code. The program's goal was to obtain the dc (direct current) operating points of circuits for a variety of manufacturing or processing variations. The *dc operating point* is usually the starting point for a simulation, and shows how the circuit operates when the voltages and currents don't vary with time. Much of the code was written to format output and deal with ambiguities (such as the convergence problems of dc operating points) that appear with automated simulation sessions.

A Simulation Problem for Designers

The process of manufacturing ICs involves unavoidable variations in quantities such as oxide thickness and doping density. These variations affect the circuit's performance characteristics, such as gain and delay.

The circuit description that ADVICE uses contains parameters that reflect these process variations and, hence, circuit-performance variations. Several simulations must be done to determine how process variations change circuit performance. Each simulation will use a circuit description that has different parameter values. These parameter changes correspond to the changes in the circuit that result from manufacturing variations. Frequently, the process-dependent parameters are held in separate files called *libraries*.

The libraries that designers use are intended to permit simulations of circuits at the extremes of some processing variations. This way, designers can check circuit performance for the worst cases of processing variations. A circuit that performs within prescribed limits for all these cases will meet its design specifications.

Many steps must be taken to simulate the circuit and compile the results for all cases of interest. Some designers have manually placed simulation results into a spreadsheet and then, with the help of a spreadsheet program, printed a summary of results for the customer. We can automate these manual steps by writing a program.

However, there is a point at which the effort to program equals the manual effort to simulate. Once we reach this point, procedural simulation is no longer useful. This is the problem that designers face today when they need to see the effect of process variations on high-level circuit-performance characteristics.

This paper describes the development and use of a program that helps designers satisfy their needs for procedural simulation. The program, called `makcal`, takes a description of the designer's simulation needs and produces a C-language program. This program is then used with ADVICE to simulate circuits and compile simulation results. The results are collated automatically in the form of tables and graphs.

Application Generators

The class of programs known as application generators are designed to improve software productivity. As input, these programs take specifications that are closely related to a particular subject. For circuit simulation, some of this input specifies the circuit to be simulated and the libraries to be used. An application generator's input is designed to be easier to handle than its output.

Frequently, an application generator's output is a program. The output of application generators for ADVICE is a program written in C language and used to control the circuit simulation.

With application generators, many programming details can be made invisible to the user. The specification's syntax can be simpler and more intuitive because the domain is restricted to encompass only the user's current interests. As a result, when properly implemented, application generators can save time.

The task of creating an application generator is not trivial. Application generators must read a specification and create a database. To create the database, the input specification must be separated into pieces or *tokens* from which an interpretation can be made that guides the creation of the program.

`Lex` and `yacc` are useful tools for building the database. However, even the use of these tools involves a considerable amount of work. `Lex` creates a program that recognizes regular expressions. With this recognition comes the ability to separate an input specification into tokens. `Yacc` is a compiler compiler. It creates a program to match the pieces of input (or tokens) with some predetermined rules.

An alternative to using `lex` and `yacc` for creating application generators is AT&T's MetaTool™ specification-driven tool builder[3] (SDTB), formerly known as STAGE.[4] An SDTB generates application generators (we often refer to it as an *application-generator generator*) and greatly reduces the effort needed to create them.

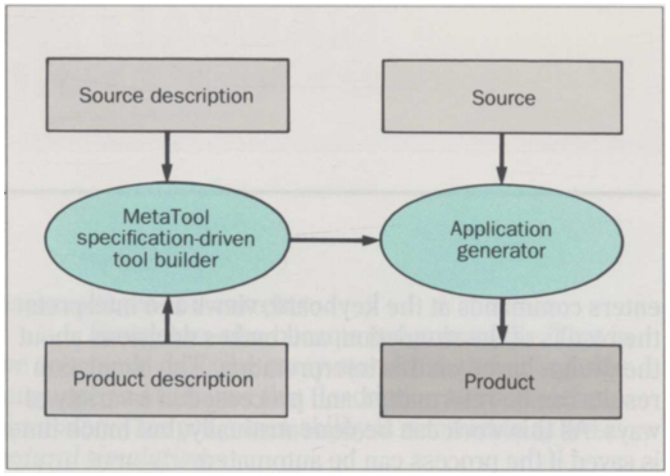

Figure 1. Software-development environment with the MetaTool specification-driven tool builder. The source description (i.e., the language used by the circuit designer) determines the application's language and is used to create a parse tree that can be traversed and used for making decisions. The product description defines the final program to be generated. Both descriptions are used to create the application generator. The source is the input to the application generator and customizes the final program for a specific need. The product is the final program produced by the application generator.

Figure 1 shows the structure of the software-development environment when using the MetaTool specification-driven tool builder. The source description consists of files that determine the nature of the application language, i.e., the language that is used by the circuit designer. Using the source description, the MetaTool specification-driven tool builder creates a parse tree for the programmer. When the application generator is run, it uses the parse tree to make decisions based on the tree's construction.

The product description files define the nature of the final program to be generated. Product description files are similar to a template in that they define the key places to be filled in when the MetaTool software creates the program. However, it is more elaborate than a simple template, because programming constructs such as `if` statements and `for` loops are available. The programming constructs draw on knowledge in the parse tree. (Programming constructs permit execution of pieces of program code based on the existence or lack of certain conditions. For example, an `if` statement identifies what to do because a condition exists or, possibly, does not exist. A `for` loop allows conditional execution and repetition of a group of statements. There is usually a variable that is incremented or decremented on each iteration; the `for` loop's statements are repeated until a logical expression that contains this variable is false.)

The source, shown on the right in Figure 1, is the input to the application generator that the MetaTool software produced. This input consists of one or more files that describe the circuit designer's interests. The circuit designer creates these files before simulating the circuit. The product of the application generator is the C program that is used to control ADVICE.

Both the source description and the product description are used by the SDTB to create the application generator. The amount of work needed to create the ADVICE application generators was reduced by a factor of ten with the SDTB, compared with writing the ADVICE control program by hand. This is a conservative estimate and is based on the difference in the number of lines of code needed by the MetaTool specification-driven tool builder and the number of lines of code in the program the SDTB produces.

Why Use Application Generators?

Frequently, the programs needed for the design of different circuits are similar. Many designers want to see how well a circuit performs under a variety of manufacturing-process variations.

The programs that perform such tasks will all have functions that run the simulations and send the results to a file. The results will be presented in the same general format, and the procedures will be similar. In the design of operational amplifiers, for instance, such parameters as open-loop gain and common-mode rejection ratio are usually of interest.

Because the designers' simulation requirements have similarities, the programs they need will also have much in common. Application generators are useful when we need to create several programs that share similar characteristics.

We can separate a simulation session into functional segments. For example,

1. Initialize the simulator by reading in the proper circuit and library.
2. Send simulation commands to the simulator.
3. Compile the results and then store them in a suitable format.

Each of these functions can be addressed by a program segment. When combined, these program segments will create a complete program that is suitable for a given procedural-simulation need.

When a program can be split into several separate parts that relate to each other, application generators are valuable.

Application generators lower the complexity of procedural simulation. They not only help eliminate annoyances, such as syntax errors, but also permit graceful and automatic handling of program-execution errors.

Because procedure definitions are kept separate from the final code and are in a simpler form, application generators provide a way for designers to share procedures.

For these reasons, the application generators are particularly well suited for ADVICE.

`Makcal`'s Role

`Makcal` is an application generator for ADVICE and, currently, is available for internal use at AT&T. It has been successful in improving the productivity of designers who rely heavily on ADVICE for their simulation needs.

`Makcal` is particularly well suited for producing comparisons of circuit-performance parameters over a variety of process variations.

As Figure 2a shows, the input to `makcal` consists of two files:

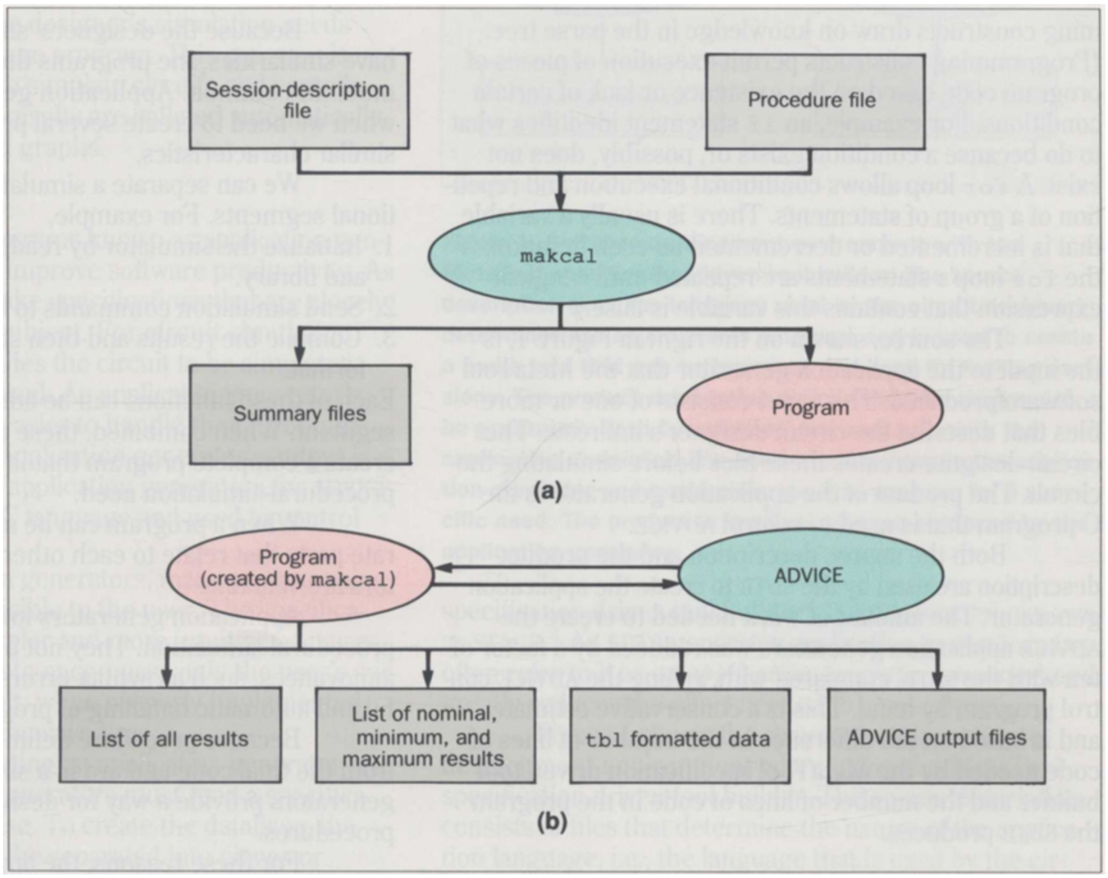

Figure 2. `Makcal` creates a program that controls ADVICE, a circuit simulator. (a) The session-description file presents circuit information, while the procedure file stores procedures similar to the steps a designer takes during an interactive ADVICE session. The output from `makcal` is a program that can control ADVICE, and summary files that can be used to verify the program's intent and log simulation sessions. (b) The program communicates with ADVICE via a two-way, UNIX system pipe. After sending commands to ADVICE, the program produces several output files of simulation results in a variety of forms. Some output files contain simulation results, formatted as tables of data that can be viewed with the ADVICE plotting package. One file contains data formatted for printing with the `tbl` preprocessor.

- The *session-description file* is used to present general circuit information such as the circuit file, library names, and procedure names.
- The *procedure file* contains the procedures that have some similarity to the steps a designer takes when

Panel 2. Procedure for Computing Leakage Current

```
/*
 * High bias leakage current
 */
PROCEDURE HBcurrent()
/* hold mode */
command("vdt -1.8")
command("vdh -0.8")
command("vin -2.5")
command("vcap 2.5")
command("rcap 0.0001")
command("rout 100G")
command(".dc op FILE")
command(".file ofile=dcop8.out")
icapp=dcop_read("FILE=dcop8.out NAME=ihold")
RESULT icapp                    UNITS "A"
```

using ADVICE interactively.

The output of `makcal` consists of a program that is suitable for use with ADVICE, and summary files that can be used for verification and logging purposes. These files can help verify that what `makcal` produces is actually what was intended. They can also be useful for maintaining records of previous activity.

When in use, the program created by `makcal` communicates with ADVICE through a two-way, UNIX® system pipe. (UNIX is a registered trademark of UNIX System Laboratories, Inc. A *pipe* is a channel for communication between programs, and a *two-way pipe* permits communication in both directions between programs.)

Figure 2b shows the result of using this program to control ADVICE. After it sends commands to ADVICE, the program produces several output files that contain simulation results in a variety of forms. Two of the output files contain tables of data. Other files are produced that can be viewed with the ADVICE plotting package, `advplot`. In one file, data is formatted to be suitable for printing with UNIX system's `tbl` preprocessor.[5]

An Application of `makcal`

As an example, we show how a designer at AT&T Microelectronics used `makcal` in an actual application. To be as clear as possible, we show only a portion of this designer's work, a portion deliberately selected to be simple.

Designers at AT&T Microelectronics have implemented a sample-and-hold circuit in an AT&T linear array (the ALA210) and wanted to examine various performance parameters for this circuit. They needed some information about dc operating points, such as the bias and leakage voltages and currents. More involved results were also requested, such as the linearity of the output voltage with respect to the input.

For the designer, a logical first step when using `makcal` is to define the simulation procedure, which is then stored in the procedure file. Because of space limitations, we will not present all the procedures here.

Panel 2 is an example of a procedure that obtained the value of high-bias leakage current. Most of this procedure's contents are commands to ADVICE that

Panel 3. Session Description for Sample-and-Hold Circuit

The session-description file specifies the circuit to use, the libraries to include in the simulation, and the procedures to execute. (To conserve space, part of the file has been omitted.)

```
TITLE "Sample and Hold"
MODEL u/models/sample.mod
CIRCUIT sha.tst
CASES  x0 x1 x2 x3 x4 x5 x6
setup()
/*
 * Biases and offsets
 */
biases IIN IAN IIPN IOPN IOPP IIPP IAP IDHL IDTH VOSZ VHCZ ()
/*
 * Offset at positive Input
 */
pos VOSP ()
/*
 * Offset at negative Input
 */
neg VOSN GERR LERR ()
/*
 * Extended Supplies
 */

          :

/*
 * High bias leakage current
 */
HBcurrent ICAPP ()
/*
 * Low bias leakage current
 */
LBcurrent ICAPN ZCAP ()
/*
 * End of session
 */
```

Panel 4. Test Results for HBcurrent

The first output file contains all results from the session in which ADVICE is run and controlled with the program created by makcal. Here, we show a portion of the file:

```
----------------------------------------------------------------------------------
Test results for sha.tst.      Function: HBcurrent       Name: ICAPP
----------------------------------------------------------------------------------
              CASES WHICH DID NOT CONVERGE:  x2
 x0    -4.865440e-08
 x1    -1.477040e-08
Case x2 did not work properly (possibly convergence problems).
 x3    -2.955730e-02
 x4    -8.426589e-09
 x5    -2.503550e-07
 x6    -1.573850e-08
```

set voltages and resistor values. A dc operating point is obtained with the command, `.dc op FILE`. We use the built-in function, `dcop_read`, to read the results from the output file. The keyword, `RESULT`, indicates that this value is a desired output. (A *keyword* is a word or name, that has special meaning to a program.) The units are given so the main routine can display the values with the appropriate symbols.

Panel 3 shows the file that identifies the circuit and libraries. The session description specifies the circuit to use, the libraries to include in the simulation, and the procedures to execute. As Panel 3 shows, a procedure can have either one result, e.g., `ICAPP` for `HBcurrent`, or multiple results, e.g., `ICAPN` and `ZAP` for `LBcurrent`. This example had seven libraries of interest, which are identified with the keyword, `CASES`. The contents of these libraries reflect changes in circuit parameters as a result of manufacturing variations.

After creating both the procedure file and the session-description file, the designer is ready to use `makcal`. Because `makcal` is not interactive, using it only requires that you type the command line. `Makcal` uses default names for files, unless command-line options specify new names. When `makcal` is finished, it produces a program that is called `advpgm.c` by default. The user can also specify a name for this program via a command-line argument.

To use the program created by `makcal`, the designer runs ADVICE and then controls it with the program. To do this, he or she enters the command `.cal advpgm.c` (or `.cal` *program_name*, where *program_name* is the name specified for the program created by `makcal`).

After this program has run, we see the results in several files. The first file contains all the results, and each result is preceded by its name. Panel 4 shows a portion of this file. The result called `ICAPP` is leakage current through a capacitor. The procedure used to compute this current is called `HBcurrent`.

If problems occur during the run, those cases are flagged and messages appear in the output file to identify the problem. As we can see in the panel, there was a problem with convergence for this circuit when using library `x2`.

This example has seven libraries of interest, so there are seven simulations for each procedure. Because

Panel 5. Test Limits for Sample-and-Hold Circuit

The program that controls the ADVICE session creates a listing of nominal, minimum, and maximum results. Here, we show a portion of the file:

```
---------------------------------------------------------------------------
       Sample and Hold     Sat Dec  9  8:57:36 1989
---------------------------------------------------------------------------
Param   Nominal Value     Lower Limit (case)       Upper Limit (case)
---------------------------------------------------------------------------
IUTVP   +45.460000 mA     +34.320000 mA (x5)       +60.610000 mA (x2)
---------------------------------------------------------------------------
ICPT   +146.319626 nA     +25.792023 nA (x5)      +531.901155 nA (x3)
ICBT   +349.194011 nA    +170.189992 nA (x4)      +908.724985 nA (x3)
TFR      +0.721233         +0.484570    (x5)        +1.411538    (x3)
ICAPP  +48.654400 nA     -29.557300 nA (x3)        -8.426589 nA (x4)
---------------------------------------------------------------------------
IAN     +16.326100 mA    +15.629500 mA (x2)       +16.326100 mA (x0)
IIPN    +14.615200 mA    +12.859002 mA (x5)       +15.085898 mA (x2)
IOPN    +5.665481 mA      +5.221371 mA (x6)        +5.665481 mA (x0)
IOPP    -5.734630 mA      -5.734630 mA (x0)        -5.355179 mA (x6)
IIPP   -18.608298 mA     -18.880300 mA (x2)       -16.852900 mA (x5)
                              :
```

many results can be produced from each procedure, it is not difficult to generate large amounts of data. Frequently, what the designer really wants is a listing of nominal, minimum, and maximum results, rather than all the results at once. Therefore, the ADVICE program creates a file that contains this information.

For the current example, this file appears in Panel 5; again, we have omitted some of the file for clarity. This table of results lists the three values of interest, i.e., nominal, minimum, and maximum. The units are displayed, and values are shown with the appropriate engineering scale. The corresponding case (i.e., the library used) is listed with the minimum and maximum values. By definition, the nominal value is given by the first library (here, `x0`) and is *not* computed from the average of the set of simulated results. The first library specified contains model parameters that correspond to nominal device characteristics.

Plots of results are also produced automatically. Figure 3 is the plot of `ICAPP`. The `makcal`-generated program automatically omits the nonconvergent cases.

Benefits of Using `makcal`

As mentioned earlier, the use of ADVICE application generators means designers spend less time programming and the code is less complex. Existing software can be reused, and error checking is built in automatically. These benefits translate directly into increased productivity for circuit designers.

Less Time. Designers are sheltered from many of the difficulties that face C-language programmers and can concentrate on the design instead. Changes in program requirements do not mean C-language code must be altered. Instead, the changes are more intuitively made in the higher level description of the simulation session.

Designers can easily benefit from each other's efforts and code is not rewritten. Much of the tedium involved with C programming is eliminated, because the tedious part is rewritten each time by the application generator.

Reduced Complexity in Coding. One of the major benefits of using `makcal` is the simplified description of the designer's needs. No preprocessor statements are required, and variables do not need to be declared. Code is written to handle nonvalid results in a graceful way.

Because the description of the simulation needs is done at a higher level than C-language code, a designer has less to be concerned about. The designer needs to supply names only for his or her procedures and results. Other variables inside the final C-language program hold intermediate results and information. These variables are invisible to the designer.

Advanced programming techniques are implicitly available for designers whose expertise most likely is not in C-language programming. `Makcal` uses variable argument lists and structures to construct its program. (A *variable argument list* is a C-language feature that permits a particular function to be called with a different number of arguments at different times. A *structure* is a group of variable types. The group is given a name and considered to be a single entity.) Output formatting, such as the conversion of results to engineering units, is handled automatically.

Sharing of Software. Because procedures are simple to create and can be generalized, the design community can easily share them. The same software is created

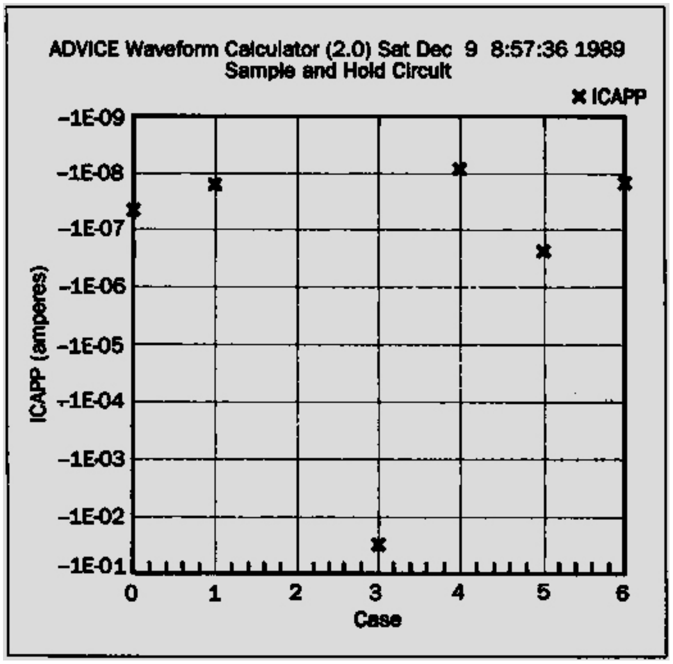

Figure 3. Plotted results of `ICAPP`, leakage current through a capacitor. The y axis is amperes. Each X is the `ICAPP` for the corresponding convergent case; nonconvergent cases (case 2, here) are automatically omitted.

for each user from a session's description. The application generator handles the minor differences, such as circuit names and library names. Thus, a designer does not rewrite the code that is identical.

When changes are needed, the designer does not need to search through and alter the C-language code. Instead, the C code is written (i.e., revised) by the application generator.

Error Checking. A major source of frustration for designers who use procedural simulation is error checking. It is costly when ADVICE expends CPU time but provides few or no results. If problems occur, ADVICE results can often be salvaged. But frequently in procedural simulation, the results from previous commands and post-processing cannot be retrieved. The time spent inserting code to check for errors is also costly.

`Makcal` automatically generates error-checking code. If a problem occurs in obtaining the results from ADVICE, an error message appears in the output file and gives the reason.

The usual reasons for errors are problems with the convergence of dc operating points, or a request for values from ADVICE that were not produced by a simulation. If such a problem occurs, the next procedure or library is invoked depending on where the problem is experienced.

`Makcal` also finds errors in argument counts. This is helpful because procedures can easily be shared and calls to procedures can easily have discrepancies in the number of arguments.

Conclusion

Procedural simulation is a necessity because designers must verify higher level, circuit-performance characteristics. The test limits are set by process variations. To meet this requirement, ADVICE was made programmable through a C-language interface.

As an alternative to programming in C language, circuit specifications can easily be described to `makcal` and automatically translated into C-language code. Because the effort and time to simulate and obtain collated results are reduced, designer productivity is increased.

Customers can benefit from having their circuit specifications presented clearly and can have greater confidence in the reliability of their product.

Designers can reuse their previous files for new circuits, or modify them with little effort. In addition, they can share files with other designers.

Because the effort to simulate is reduced, it is possible to fabricate circuits earlier and reduce the time to market.

Acknowledgments

Many thanks go to Iris Sindelar and all the people in the Advanced Software Products group in Department 54711 for spending the time and helping me understand and use the MetaTool specification-driven tool builder. Andrew Grochowski, Kem Feridun, Edwin Hoch, and other designers at AT&T Microelectronics were very helpful in providing the initial motivation and feedback for this project. I am greatly indebted to Prasad Subramanium and Sally Liu for their encouragement and guidance.

References

1. L. W. Nagel, "ADVICE for Circuit Simulation," *Workshop on Large Scale Networks and Systems*, IEEE International Symposium on Circuits and Systems, Houston, Texas, April 28-30, 1980, IEEE, New York, 1980.
2. L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memorandum ERL-M520, Engineering Research Laboratory, University of California, Berkeley, California, May 9, 1975.
3. AT&T, *MetaTool™ Specification-Driven Tool Builder System Overview*, Release 1, Document No. 193-010-211, June 1990; available from: AT&T, Intellectual Property Division, 10 Independence Boulevard, Warren, New Jersey 07060, 1-800-462-8146.
4. J. C. Cleaveland and C. M. R. Kintala, "Tools for Building Application Generators," *AT&T Technical Journal*, Vol. 67, No. 4, July/August 1988, pp. 46-58.
5. M. E. Lesk, "TBL—A program to Format Tables," Computing Science Technical Report 49, Bell Laboratories, Murray Hill, New Jersey, 1976.

(Manuscript received October 19, 1990)