ASP: An Interactive APL Circuit Simulation Package

by

Gregory D. Jordan

DRC-01-08-82

April, 1982

ASP ;
AN INTERACTIVE APL CIRCUIT SIMULATION PACKAGE

by Gregory D. Jordan

Submitted in partial fulfillment of the requirements
for the degreee of Master of Science

Department of Electrical Engineering

Carnegie-Mellon University

August 19, 1981

TABLE OF CONTENTS

Acknowledgements

Many thanks to my advisor Steve Director for his cuidance and advíce during the past year.

CHAPTER 1
INTRODUCTION

A circuit designer typically has many hardware and software design aids available for use. When doing hardware work, the designer builds small sub-circuits and tests each one separately before assembling the final project To test these sub-circuits, signals may be entered so the designer can examine the sub-circuit as it is slowly stepped through its functions. At any point, the designer may wish to stop and change portions of the circuit to enhance performance.

Unfortunately, existing circuit simulators [Weeks 73],[Cohen 78] were not designed to be used in this manner. Present simulators can handle large circuits but allow very little interaction. Specifically, existing simulators cannot be halted during execution to make on-line changes to the circuitry. ASP was designed to fill this gap in circuit simulation technology.

This paper will describe the background and implementation of ASP, an APL Simulation Package. Initially, motivation for ASP will be discussed. The advantages and disadvantages of APL will then be mentioned to give the reader a better understanding of the features of ASP. Theory behind the algorithms used in the simulator will then be considered, followed by a review of some implementation considerations. Finally, some of the major design decisions made in ASP will be discussed and justified.

ASP was written to piovide APL users with a highly interactive circuit simulator with good on-line help features. The most obvious use for ASP involves circuit optimization work currently under development at Carnegie-Mellon University. The nature of optimization makes it imperative that a circuit simulator be available. While it is possible to manually derive circuit equations, this becomes very cumbersome for all but the simplest of circuits. ASP allows the developers to make more extensive tests of their circuit optimization algorithms.

APL was chosen for this type of work because of the following features:

- APL is a highly interactive language, that is, user input is very easily incorporated into programs. It is also quite simple to check input for syntax errors or to provide the user with access to global variables.

- It is very easy to develop new algorithms using APL. Once the language has been mastered, the programmer may take advantage of the many powerful operators available in APL to decrease development time.

- Debugging is very simple in APL due to the existence of commands such as TRACE and the availability of global variables.

ASP makes use of these positive aspects by performing interactive checks on user input, making on-line help available, and allowing the user much more control over the simulation than is normally possible.

Unfortunately, ASP is also constrained by the negative aspects of APL. Because APL has no compiler, execution times are relatively slow. To compensate, ASP has some special features which allow the user to exert more control over his/her usage of CPU time.

- ASP allows the user to determine error constraints. To reduce CPU time then, the user may choose to allow more truncation error.

- ASP allows the user to avoid needless dc simulations. Before a transient simulation is done, ASP must somehow determine the operating point of the circuit. If the user wishes, he/she may enter the operating point of the circuit and eliminate the dc simulation. If the operating point is unknown, ASP will solve for it.

- ASP allows the user to specify break points, or halting points during the circuit simulation. At each break point, the user has the option of changing break point data, changing circuit data, or simply continuing with the simulation.

Another negative aspect of APL concerns readability of code. Because of the abundance of operators, APL is a very difficult language to read. To alleviate this problem, ASP code contains many comments. Each function heading describes local and global variables used in the function. The heading also lists any functions which may be called during the execution of the present function. In addition, extensive comments have been written within the body of the function to enhance readability.

CHAPTER 2
THEORETICAL CONSIDERATIONS

Before beginning a specific discussion of the features of ASP, this paper will first examine the theoretical issues which affect circuit simulation. In particular, the concept of modified nodal analysis will be reviewed. With this concept in mind, simple element stamps may be developed for use in the simulator. The idea of stamps can then be applied to more complex elements such as nonlinear elements, capacitors and inductors; however, algorithms to solve nonlinear equations and do numerical integration will be necessary. Finally, it is desirable to solve the resulting matrix equation in an efficient manner. This section will deal with all of the above issues and describe the algorithms used in ASP.

2.1 Modified Nodal Analysis

The modified nodal method was developed as an alternative to the nodal and tableau methods. The major disadvantage of the tableau -approach was that it produced large numbers of equations and variables. The nodal method addressed this problem by making node voltages the basic variables. One disadvantage of the nodal method is that branch currents are not conveniently obtained as output A related problem is that voltage sources and some dependent sources are handled very inefficiently. To remedy these problems, Ho, Ruehli, and Brennan [Ho 75] proposed some modifications to the nodal method. The resulting method was then called the modified nodal analysis (MNA).

The basic premise of modified nodal analysis is that the basic variables are node voltages and the branch currents of voltage sources, inductors, and other elements whose currents are controlling variables. Introducing the extra basic variables allows voltage sources and dependent Sources to be handled much more efficiently than in nodal analysis. It also ^allows inductor currents to be easily accessible as output Because of these characteristics, modified nodal analysis is the method used in ASP.

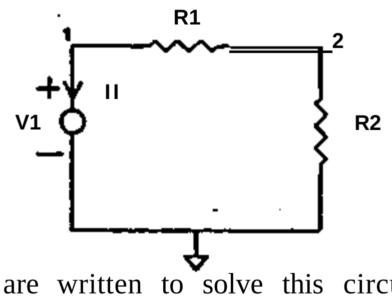
The MNA matrix can in general be expressed in the form:

$$\begin{bmatrix} Y & B \\ C & D \end{bmatrix} \begin{bmatrix} \underline{V} \\ \underline{I} \end{bmatrix} = \begin{bmatrix} \underline{J} \\ \underline{F} \end{bmatrix}$$

where $\underline{V}$ is the vector of node voltages,
$\underline{I}$ is the vector of branch currents associated with voltage sources, inductors, and some dependent sources.
Y contains the partial derivatives of KCL equations with respect to the node voltages,
B contains the partial derivatives of KCL equations with respect to the branch currents,
C contains the partial derivatives of branch relationships with respect to the node voltages, and D contains the partial derivatives of branch relationships with respect to the branch currents.
$\underline{J}$ and $\underline{P}$ contain values of voltage sources and current sources. These two vectors may also contain contributions from nonlinear elements and energy storage elements.

2.2 The Development and Use of Some Simple Stamps

To further illustrate MNA, it will be useful to construct and solve a simple circuit using the modified nodal analysis approach. The simple circuit will consist of a voltage source and two resistors as shown below.



Using MNA. three equations are written to solve this circuit. Two of the equations will add the currents leaving each of the nodes. The third equation will specify the branch relation of the voltage source. The equations resulting from an MNA analysis of the circuit are:

$$1:\quad I_1 + (1/R_1)v_1 - (1/R_1)v_2 = 0$$
$$2:\quad -(1/R_1)v_1 + (1/R_1 + 1/R_2)v_2 = 0$$
$$I:\quad v_1 = V$$

or. in matrix form:

$$\begin{bmatrix} 1 & 1/R_1 & -1/R_1 \\ 0 & -1/R_1 & 1/R_1 + 1/R_2 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ v_1 \\ v_2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ V \end{bmatrix}$$

Looking at the matrix equation, it can be seen that each element in the circuit contributes terms to the Jacobian. It may also be observed that the pattern of Jacobian terms is consistent within element types. For example a resistor between nodes M and N with a conductance of value G will contribute four terms to the Jacobian. (We are assuming, of course, that neither of the affected nodes are ground nodes.) This resistor will add two terms of G in the (M,M) and (N,N) positions of the Jacobian. In addition, two terms of -G will be added in the (M,N) and (N,M) positions. As a group, these terms which are added to the Jacobian for each element are called the element stamp. In the following two paragraphs, a closer look will be taken at resistor and voltage source stamps. For a complete listing of stamps used in ASP, see Appendix A.

Resistor Stamp

Suppose a resistor connected between nodes M and N has a conductance of value G. This resistor will contribute a current to the equations of nodes M and N. The current leaving node M can be expressed as the difference of the node voltages M and N times the conductance G. A similar calculation can be done for the current leaving node N. The resulting effect on the MNA equations due to this resistor can then be expressed as:

	V_M	V_N	RHS
M	G	-G	0
N	-G	G	0

In the stamp notation shown above, the rows are labeled with equation numbers and the columns are labeled with the voltage which the stamp terms in that column are multiplying. RHS indicates the right hand side terms. In addition, BR indicates a branch relationship.

Voltage Source Stamp

Suppose a voltage source connected between nodes M and N has a value of Q volts. This voltage source has a branch current <IV> which contributes current to the equations of nodes M and N. The voltage source value is then determined by its branch relationship equation. If the positive node of the voltage source is at node M, the resulting effect on the MNA equations can be expressed as:

$$
\begin{array}{c|cccc|}
 & V_M & V_N & I_V & RHS \\
\hline
M & 0 & b & 1 & 0 \\
N & 0 & 0 & -1 & 0 \\
BR & \bullet\ 1 & -I & 0 & Q \\
\hline
\end{array}
$$

2.3 Stamps for Nonlinear Elements

The concepts developed in the previous section can now be extended to nonlinear elements such as diodes and transistors. In this section, the development of a diode stamp will be discussed. The transistor stamp or the stamp for any nonlinear element can be derived using a similar approach.

Consider a diode with the branch relation:

$$I = I_s (e^{V/V_T} - 1) \tag{2.1}$$

where V_T is the thermal voltage and Is is the saturation current.

It is obvious that:

$$\frac{dI}{dV} = \frac{I_s}{V_T} e^{V/V_T} \tag{2.2}$$

Now suppose initial guesses of Vo and Io are chosen. The problem *is* then to satisfy equation (2.1) given the constraints of the circuit An easy way to solve this equation would be to use a Taylor series approximation. Linearizing about the initial guess and truncating after the first two terms yields:

$$I \text{ « } Io \bullet \frac{Is}{V_T}\, c^{Vo/V_T} \; (V - Vo) \qquad (2.3)$$

The new I and V found when this equation is solved with the circuit constraints are presumably closer to the real solution than Io and Vo were. Replacing the initial guess with I and V, one can repeat the procedure to get an even more accurate approximation. Using this iterative method with appropriate initial guesses. I and V will eventually converge to the true solution. This is the form of the Newton-Raphson method which is used in ASP.

Given this method, it is now possible to derive the form of a diode stamp. Note that there are three separate terms in the Taylor series equation. Two of the terms are dependent on Io and Vo and are therefore known. Ignoring the third term, it becomes apparent that the contribution to the circuit from the equation is a current source.

That is, in the equation.

$$I * Io - \frac{Is}{V_T}\, Vo\; e^{Vo/V_T} \qquad \text{<2.4»}$$

I is known since all the right hand side terms are known, A known current corresponds to a current source in the circuit.

The other term is dependent on the unknown diode voltage V.

$$I \text{ « } \frac{Is}{V_T}\, V\; e^{Vo/V_T} \qquad (2.5)$$

I is an unknown in this equation and is dependent on V. An unknown current dependent on the voltage across the element corresponds to a conductance in the circuit.

Combining all of the terms in the Taylor series and converting the contribution of each term into a circuit element yields a stamp. The diode stamp can then be expressed as a current source of value equation (2.4) in parallel with the conductance indicated in equation (2.5). This stamp will be updated according to the Taylor series expansion after each new iteration of V and I is found. When the values begin to converge to the real solution, one can check the error criterion and stop iterating at the appropriate time.

2.4 Stamps For Energy Storage Elements

Stamps are still unknown however, for energy storage elements, a class of elements which includes both capacitors and inductors. Since the branch relations for capacitors and inductors involve derivatives, a method for numerical integration must first be found. Numerical integration schemes make an approximation to the integral by taking small time steps and interpolating. The method used in the simulator was proposed by Van Bokhoven [Van Bokhoven 75] in a 1975 paper. Some changes have been made in order to simplify the algorithm and reduce the complexity of calculations.

The method is derived using Newton's divided-difference interpolation formula. This method makes predictions to new data using previous, data. Newton's interpolation formula truncated to two terms suggests that a new approximation to X can be made using:

$$X(t_{n*l}) \text{ « } X(t_n > \bullet (t_{n-1} - t_n)X(t_n,t_{n-1} > \qquad (2.6)$$

where t_n indicates the time of the last known value of X,
$t_{n-\wedge}$ is the time previous to t_n,
and $X(t_n,t_{n-1})$ is a divided-difference given by the formula:

$$\frac{X(t_n> - X(l_{n-1})}{t_n - t_{n-1}} \qquad (2.7)$$

To simplify this equation, let us define some new variables.

$$h_I = t_{n+l} - t_n$$

$$h'_1 = t_n - t_{n-1}$$

$$d_i = h_i / h_I'$$

Lei us also define the following conventions:

X

represents the (M-l>st prediction of X at the <n+l)st time step and

X_{n-1}

represents the final calculation of the value of X at the (n+l>st time step.

Equations (2.6) and (2.7> can then be rewritten as:

$$\overset{\_1}{X}_{n-l} \text{ « } X_n \qquad \textbf{(2.8)}$$

$$\overset{\_2}{X}_{n*l} \text{ « } \overset{\_1}{X}_{n*M} + d_1 (X_n - \overset{\_1}{X}_n > \qquad (2.9)$$

Using the above equations, the simulator makes a zerotb **and** first order prediction to the value of X at the new time step. To correct this prediction, Van Bokhoven suggests an approach that is also based on Newton's divided-difference interpolation formula.

Corrections are made by approximating the derivative of X with respect to time. Using divided-differences, the derivative can be approximated by $X(t_{n*"l}, t_{n—1})$. A second order correction can be made using the formula:

$$\left.\frac{dX}{dt}\right/_{n+1} = \frac{X_{n-l} - \overset{\_1}{X}_{n-l}}{h_1} \text{ «• } \frac{X_{n-l} - \overset{\_2}{X}_{n+1}}{h_2} \qquad (2.10)$$

where $h_2 = t_{n*l} - t_{n-l}$

Using these equations then, the stamp for a capacitor or inductor can be calculated. For example, suppose we have a circuit where the only unknown is a capacitor voltage. X. At each time step, zeroth and first order predictions of X would be made. Once these predictions are known, th? capacitor stamp can be temporarily added to the circuit in order to solve for the voltage at the present time step. The capacitor stamp consists of:

A current source of value:

$$(C/h_1)\ \overset{\_1}{X}_{n*l} * (C/h_2)\ \overset{\_2}{X}_{n-l}$$

in parallel with

a conductance of value: .

$$<C/h^1> \bullet (C/hj$$

where C is the value of the capacitor in farads.

The NfNA system of equations can now be solved for the final solution at the in+Dst time step.

Inductors can be handled in a similar fashion. The only major difference is that while the capacitor stamp consists of a current source in parallel with a resistor, the inductor stamp consists of a voltage source in series with a resistor. This is very desirable since inductor currents are available as output because of the presence of a voltage source in the stamp.

The other portion of the Van Bokhoven algorithm is step size control. It is important that step sizes are well chosen for a number of reasons. If the step size is too small, the simulation will take a very long time. If the step size is too large, the numerical integration algorithm may not converge to the desired solution or truncation error may be too large. To minimize CPU time, it is necessary to choose the largest possible step size which satisfies the desired error criterion.

Using the remainder term from Newton's interpolation formula, second order truncation error at the nth step can be calculated using the formula:

$$\overline{E}^{3}_{n} = \frac{(\overline{X}^{3}_{n} - X_{n})}{1 + h_{3}B_{2}} \tag{2.11}$$

where $h_{3} = t_{n+1} - t_{n-2}$

$h_{2}' = t_{n} - t_{n-2}$

$B_{2} = (1/h_{1}) + (1/h_{2})$

$\overline{X}^{3}_{n} = \overline{X}^{2}_{n} + d_{2}\,(X_{n} - \overline{X}^{2}_{n})$

and $d_{2} = d_{1}\,(h_{2}/h_{2}')$

It is now possible to predict the second order truncation error at the (n+1)st step if a step size of h is used. Extrapolating from the above formula and making substitutions yields:

$$\overline{E}^{3}_{n+1} = \frac{h(h_{1} + h)\,((1/h_{1}) + (1/h_{2}))\,\overline{E}^{2}_{n}}{((1/h) + (1/(h+h_{1})))\,h_{1}\,h_{2}} \tag{2.12}$$

If the predicted error is set equal to the maximum allowable error, the resulting nonlinear equation in h can then be solved using the Newton-Raphson technique. This new step size is then used to make the next prediction and correction by updating the stamps of energy storage elements. The circuit equations are then solved and a new error is determined. If the new error is greater than the maximum allowable error or if a nonlinear element stamp did not converge, the simulator rejects the step, cuts the step size in half, and starts the process again. If the new error is within tolerances, the simulator will increment time and solve for a new time step.

2.5 Solving the Circuit Equations

Finally, a technique is necessary to efficiently solve the matrix equations which have been created. Such a technique should take advantage of the special features of the circuit equation matrices:

1. Sparseness - Large circuits especially have a relatively small proportion of nonzero elements. Hence the solution method should avoid storage of the zero-valued coefficients. The method should also avoid needless multiplications by zero.
2. Uniformity - That is, the equations of each circuit have a particular pattern of zeroes and non-zeroes. In spite of updating stamps for energy storage and nonlinear elements, this pattern remains the same. A good method would take full advantage of this fact

One can take advantage of the sparseness and uniformity properties by using a* combination of LU factorization and sparse matrix methods.

LU factorization involves factoring the matrix equation to be solved into an upper triangular(U) and a lower triangular* L) matrix. Once these two matrices are known, a simple forward and backward substitution will solve the problem. Since the matrix is sparse however, many fewer operations need to be done than the total indicated by the substitutions.

The sparse matrix method used in ASP allows for efficient storage of the jacobian matrix. Three vectors are used to index and store members of the jacobian.

- IA is a vector containing column indices of all the elements
- 2A is a vector containing row pointers
- A is a vector containing the nonzero elements of the jacobian

After the three vectors are compiled, pivoting is done using the Markowitz criterion [Markowitz 57] to reduce fill and operation count.

LU factorization and sparse matrix techniques will not be discussed in greater detail in this paper. Further information on sparse matrix techniques can be found in Sparse

Matrices and Their Applications [Gustavson 72]. These algorithms were packaged by Jim Christ and Fred Heaton as a semester project for 18-701. Because of the nature of the interface, the existence of sparse matrix techniques is nearly invisible to the simulator.

CHAPTER 3
IMPLEMENTATION CONSIDERATIONS

This chapter will deal with specific implementation issues concerning ASP. Emphasis will be placed on interesting or new features of ASP. A better understanding of the organization and use of the simulator can be gained by reading this section and referring to the appendices at the appropriate times.

3.1 Help Files

The first unique feature of ASP is the set of twenty seven help files in the ASP workspace. Many of these help files summarize syntax for the different element types. Each time the user is prompted for input, it is possible to type HELP and receive some. In addition to the more specific help files, two general files exist which allow the user to see what type of help is available. The most general of these two files is listed below:

```
A LIST OF ALL HELP FILES AVAILABLE ALONG WITH THEIR
SUBJECT IS:

HELP - THIS FUNCTION!
HELPBKADD - ADDING BREAK POINTS
HELPBKDROP - DROPPING BREAK POINTS
HELPBKUTILS - RESPONDING TO UTILITIES MENU AFTER BREAK
              POINT
HELPBREAK - SPECIFYING BREAK POINTS AT INITIAL INPUT
HELPC - SPECIFYING CAPACITORS
HELPCON - SPECIFYING DESIGN CONSTRAINTS
HELPDEFN - DEFINING NONLINEAR MODELS
HELPDEP - SPECIFYING DEPENDENT SOURCES
HELPDROP - DROPPING ELEMENTS
HELPGYRA - SPECIFYING GYRATORS
HELPI - SPECIFYING CURRENT SOURCES
```

```
HELPINP - LISTING OF ALL CIRCUIT ELEMENT INPUT HELP FILES
HELPINPDEC - RESPONDING TO TYPE OF INPUT QUESTION
HELPL - SPECIFYING INDUCTORS
HELPMACRO - SPECIFYING MACROS
HELPNODEOUT - SPECIFYING NODE VOLTAGES FOR OUTPUT
HELPOUTFILE - WRITING TO EXTERNAL FILE
HELPQ - SPECIFYING NONLINEAR ELEMENTS
HELPR - SPECIFYING RESISTORS
HELPREAD - HAVING FILE READ BY SIMULATOR
HELPRINT - SPECIFYING PRINTING INFORMATION
HELPSIM - CHOOSING TYFE OF SIMULATION
HELPTRAN - SPECIFYING TRANSIENT SOURCES
HELPUTILS - RESPONDING TO GENERAL UTILITIES MENU
HELPV - SPECIFYING VOLTAGE SOURCES
VCURSXPLAIN - SPECIFYING CODED CURRENT AND VOLTAGE
              NUMBERS
```

In case this on-line help is not sufficient, a more extensive source of help is the ASP User's Manual listed in Appendix C.

3.2 Input Overview

Before moving on to discussions of specific elements and features, the input language of ASP will be briefly reviewed. A line of element input is used to specify the following information:

- the element type
- the element number
- the nodes which the element is between
- and the value of the element or information to determine element definition.

Some typical element specifications would be:

```
Rl 1 2 1000
C2 39 50 1E.NG5
VI 987 34 10
```

Other types of input are accepted by ASP as well. The user may choose to specify break points, design constraints or simulation time and printing information. These structures as well as a more comprehensive review of element specifications can be found in the ASP User's Manual which is Appendix C.

3.3 Nonlinear Elements

Another special feature of ASP is the flexibility it has with regard to nonlinear elements. The user may specify nearly any type of nonlinear element for ASP to process. The only restrictions are:

1. The user must be able to write equations to express each element current in terms of node voltages.

2. The user must be able to write equations to express the partial derivatives of each element current with respect to the involved node voltages.

Many well known models lend themselves easily to this format The Ebers-Moll transistor model and the simple diode model are especially useful and easy to specify. It is possible to specify many different models and to save these models in the workspace for future use. It is also possible for the user to specify a nonlinear element with as many nodes as necessary. There is no limit to number of terminals on an ASP nonlinear element.

We now *give* a thorough discussion of how nonlinear elements are handled in ASP. A nonlinear controlled source is specified in the input language by the statement*

QK Nl N2 . . . NN MODEL PI P2 . . . PM

- where Q signifies a nonlinear element
- M is the number of the nonlinear element.
- Nl, N2, . . . NN are the nodes to which the nonlinear element *is* connected,
- MODEL is the name of the function which contains the model definition,
- and PI, P2, . . . PM are parameters to be used by the model function.

Before entering ASP, the user should have created the two defining functions and saved them in the ASP workspace. In our example, the function MODEL is specified so the two functions should have been called MODEL and DMODEL (The derivative function name should always correspond to the current function name preceded by a 'DM. MODEL specifies N-l element currents in terms of the N node voltages. The Nth current can, of course, be determined from the first N-l currents. DMODEL specifies the derivatives of each current with respect to each node voltage. That is MODEL returns an N-l vector of currents and DMODEL returns a N-l x N matrix of derivatives. Both of these two functions may accept and use the P parameters as part of the nonlinear model. Using these two functions and the user input shown above, ASP can accurately model the nonlinear element according to user specifications.

In Chapter 2, it was shown that a diode can be modeled by a parallel combination of a resistor and a current source. It is possible to use this same technique with any nonlinear element. For example, suppose the equation of a noniinear element can be expressed as:

$$i_m = f_m(v_1, v_2, \dots v_N) \tag{3.1}$$

for m=1,2, . . . N-1

If an initial guess of $v_{1,0}, v_{2,0}, \dots v_{N,0}$ is chosen, a Taylor series expansion can be done to linearize the equation.

$$\begin{aligned} i_{m,1} = i_{m,0} &+ (di_{m,0}/dv_1)(v_{1,1} - v_{1,0}) \\ &+ (di_{m,0}/dv_2)(v_{2,1} - v_{2,0}) + \dots \\ &+ (di_{m,0}/dv_N)(v_{N,1} - v_{N,0}) \end{aligned} \tag{3.2}$$

As before, it is possible to notice that there are two types of terms in the above equation. One type consists of all terms which must be calculated at the zeroth iteration. Because these terms are known, they effectively add a current source to the circuit The other type consists of terms which are dependent on the node voltages at the first iteration. Since these terms are unknown and relate the current tc the node voltages, they effectively add a resistance to the circuit. In this way, the stamp for a nonlinear element can be determined. MODEL and DMODEL calculate all the necessary terms to add the stamp to the jacobian. After each iteration, the stamp is updated until the solution converges.

The function NEWRAP is used to implement the Newton-Raphson iteration in ASP. The first function to be called by NEWRAP is NOQSAVE. This function saves the elements of the jacobian which will be altered by any nonlinear stamp. NOQSAVE is necessary so the jacobian can then be restored after the iteration calculation is done. UPDATQ is then called to update the nonlinear stamp. UPDATQ calls the model function and the derivative function, creates the stamp, and adds it to the jacobian. The matrix equation is then solved and the difference between the new solution and the solution from the previous iteration is measured. The maximum voltage and current errors are found and these are compared with an absolute and relative voltage and current error. If either error is too large. NOQRESTOR is called to restore the jacobian and another iteration is done. If the error is tolerable, NOQRESTOR is called, time is updated, and the simulator can then take another step. A function level description of ASP can be seen in Appendix B.

As the user specifies nonlinear elements, checks are run to ensure that input language syntax has been followed. ASP also checks that the necessary functions have been defined. No checks are done on the model definition function because of the great flexibility which the user has. The user must be careful to define the correct number of currents and derivatives and the user is responsible for the accuracy of the functions. It is anticipated that a user will only need to develop a few nonlinear models and can then store these models in his/her ASP workspace.

3.4 Macros

To augment the extensive nonlinear element capability, ASP also allows the user to define macros or groups of elements. Macros become significant when the user needs to expand models to include energy storage elements. Since no capacitors or inductors can be expressed in a nonlinear model, macros are necessary to allow more sophisticated modeling. It is anticipated that any complex transistor models will be expressed as macros consisting of some nonlinear elements and some capacitors and resistors.

The specification of a macro is done by the statement:

```
MACM N1 N2 . . . NN ELEMENTS P1 P2 . . . PM
```

- where MAC signifies a macro,
- M is the element number of the macro,
- N1, N2, . . . NN are external nodes to which the macro is connected,
- ELEMENTS is the name of the function which defines the macro elements,
- and P1, P2, . . . PT are the parameters of the macro elements.

Previously, the user should have created the function (ELEMENTS in this case) which contains the elements of the macro. The syntax for listing macro elements is only slightly different than the regular element specification syntax. Function definition and macro input are discussed in more detail in the ASP User's Manual.

When specifying a macro, ASP allows the user to specify parameters, and internal and external nodes for the macro. Parameters can be specified as VAL[1], VAL[2], . . . VAL[M] corresponding to P1, P2, . . . PM. Parameter passing allows the same macro to be used for many elements of the same form but with different specs. Internal nodes are specified as (INTERNAL), (INTERNAL+1), etc. ASP uses numbers greater than 7000 to refer to internal nodes. For more information on node and current syntax, see

Appendix I of the User's Manual. Elements of the macro refer to internal nodes in the above manner. After each macro *is* added to the data base, the global variable INTERNAL is updated so it is greater than the highest node specified thus far. This allows the user to specify internal nodes without any knowledge of which numbers have been used. As long as the user specifies nodes greater than INTERNAL, there should be no problem. External nodes are specified as <EXTERNAL[1]), (EXTERNAL[2]>, . . . (EXTERNAL[N]> corresponding to N1, N2, . . . NN. Specifying external nodes in this manner allows the macro to be interchangeable. The user can of course specify a node directly however this seems to defeat the purpose of macros.

Macros are implemented in ASP through the function ADMAC. This function looks at the user-defined ELEMENTS function and assigns numerical values to the variables EXTERNAL, INTERNAL, and VAL. ADMAC then pieces together these values and forms a character string which is of the same format as the regular element specifications. Finally, ADMAC calls the function CHANGE which calls the appropriate function to add the specific element.

• Very few checks are done on macros. The input specification *is* checked for duplicate elements and a minimal number of external nodes. A check is also run to make sure the function containing the macro elements exists. No checks are run on this ELEMENTS function so the user must be careful.

3.5 Design Constraints

Since it is expected that ASP will be used as a design tool, a capability to evaluate design constraints has been included. Specifically, ASP can evaluate any function of the form:

$$I = \int_0^T f(\underline{V},\underline{I})\ dt \qquad (3.3)$$

where f is a function of the node voltages $\underline{V}$ and the branch
currents $\underline{h}$
and T is the ending time of the simulation.

To specify a design constraint, the user types in:

CONK CONFUNC
v

- where CON signifies a design constraint,
- M is the number of the constraint.

• and CONFUNC is the name of the design constraint function.

To define the function CONFUNC, the user must once again be familiar with the node and current syntax found in Appendix I of the User's Manual. CONFUNC should be a function of node voltages and branch currents and should return the value of the design constraint

ASP treats design constraint evaluation as a current source of value CONFUNC charging a capacitor of value IF. Using the numerical integration scheme mentioned in Chapter 2, it is possible to model the capacitor as a current source in parallel with a resistor. Design constraint evaluation then reduces to the problem of analyzing a circuit with two current sources in parallel with a resistor. ASP solves this simple circuit and returns the results of the function integration.

CONFUNC is defined with the aid of the ASP function NUM. This function takes the number of a node voltage or branch current and returns the present value of this voltage or current NUM allows the user to conveniently specify values of voltages and currents as opposed to names. CONFUNC then returns the value of the current source to be used in the circuit explained above.

Function evaluation is done in parallel with solution of the main circuit After a step is accepted, first and second order predictions are made to the value of the design constraint. The correction is then done as explained above. The user need not specify the design constraint as output since it will automatically be shown if constraints have been specified.

Once again, no checks are done on the user-defined function. The user must be careful to specify the proper voltages and currents and to return the proper value of the design constraint function. The user is reminded, however, if the specified constraint function does not exist

3.6 Break Points

Another special feature of ASP is its ability to process break points. Break points are user-specified halting points in the simulation. They are useful when the user can determine the success or failure of his/her test case before the simulation has finished. The user may wish to change the value of one element and then simulate the same circuit again. Break points facilitate manipulations such as this.

Break points are specified by the input line:

```
BREAK T1 T2 . . . TN
```

Break point times need not be specified in ascending order however, they will be serviced in ascending order.

When ASP reaches the initial break point, voltages and currents will be calculated as usual and the break point utilities menu will be called. This menu gives the user seven options.

1. Add break points - The user may add new break points to the present list. This capability is useful if the user would like to continue the unbroken simulation for some time longer than expected.

2. Delete break points - The user may delete break points from the list. This is useful if the user decides that the simulation is running well and no more break points are necessary.

3. Examine break points - The user may examine all break points specified so far.

4. Examine values of constraints, voltages or currents - The user may examine the current status of any constraints, voltages, or currents.

5. Change circuit element data - The user may temporarily return to the main utilities menu to add or drop circuit elements. This capability is useful if the user would like to change one or two circuit elements while keeping the rest of the circuit the same. After these modifications are done, ASP attempts to use the old dc solution for the new circuit. In the case of minor circuit modifications, this would save the user a substantial amount of time. If the attempt fails, ASP solves for the new operating point.

6. Continue with simulation - The user may continue with the simulation as if the break point was never serviced.

7. Exit - Finally, the user may simply choose to exit from ASP.

3.7 Interactive Checks

The final special feature of ASP to be considered is interactive checking. During the input phase, the user has the option of choosing interactive input or file reading input. If the user chooses to input circuit data interactively, each element specification will be examined for syntax errors. Appropriate error messages will be displayed and the user can make immediate corrections and reenter the elements. In the case of file reading input, the user will see a list of errors. The user then has the option of entering the interactive input mode to make corrections.

Element specifications are checked for:

- omission of element numbers
- unacceptable characters (alpha or numeric)
- incorrect number of nodes specified
- shorted element specified
- node numbers or element number greater than 999
- non-integer node numbers
- and the existence of duplicate elements.

In addition, more specific checks are made on some elements. For example, a check is run to detect O-valued resistors. More extensive checks are also run on transient source information.

A more general checking function is also included in ASP. The function HANGELCK checks for hanging nodes, omission of simulation type information, omission of end time and printing increment information, failure to specify any elements, and failure to define a specified design constraint HANGELCK must be successfully completed before a simulation can be run.

CHAPTER 4
DESIGN DECISIONS

One of the most difficult problems encountered while programming ASP involved the resolution of two important design decisions. To discuss these decisions, one must first understand some of the problem constraints. One of the major advantages of ASP was to be its interactive nature and the extent to which the user's input was checked for errors. In conflict with this goal was the necessity to make ASP a flexible and useful tool as well. The second trade-off involved simulation time and truncation error. In order to run a fast simulation, one must relax the error constraints. In order to run an accurate simulation then, one must be willing to sacrifice somewhat in the area of CPU time. In this chapter, these irade-offs will be discussed and the ASP project will be summarized.

4.1 **Flexibility** vs. **Interaction**

Two of the main areas where the flexibility vs. interaction conflict -came into focus were the definition of nonlinear elements and the definition of macros. For nonlinear elements, it would have been possible to define one or two often used nonlinear elements as part of the package. Although the user would be confined to using these predefined models, he/she could be assured of correct syntax. Macros would not be allowed due to the difficulties involved with syntax checking. ASP chooses the alternate option of flexibility. The main reason for this choice is that the user is free to define any type of nonlinear element/macro combination. The user is not at all limited by predefined models or element types. In addition, macros can be used to define blocks of elements which may be redundant in a circuit, thus saving the user input time.

The cost of these advantages was seen as very minimal. The use of parameter passing in macros and nonlinear elements increases the flexibility of each model. Workspaces in APL also allow for easy storage of old models. Definition procedure was also simplified so that a user with some basic familiarity with APL should have very little trouble defining models.. Any initial difficulties the user may have seem insignificant when one views model definition as essentially adding a new, more exact element to the simulator's capability.

Extensive syntax checking is done when any other element is entered by the user. These checks are listed in the User's Manual following the specific element syntax. The only known way for the user to be unexpectedly expelled from the simulation package is when the user incorrectly defines nonlinear models or macros.

4.2 Run Time vs. Error Tolerances

The other important trade-off involves run time and error tolerances. ASP has a step size control mechanism which ties step size to truncation error. If the user is willing to allow large amounts of error, ASP will take relatively large steps. Conversely, if the user will not tolerate much error, ASP will take relatively small steps. Because APL has no compiler, run times can sometimes become excessively long if a tight error criterion is specified.

To illustrate the relationship between truncation error and step size, a dc simulation test case was run using three different error tolerances. The data below shows the effect which error tolerances have on CPU time.

```
Trial 1:
EVABS = .0001
EVREL = .001
EIABS = 1E.NG7
EIREL = .001

CPU time = 14:05 (in minutes:seconds)

Operation count = 1919006

Statement count = 550272
```

```
V(3) = -0.1

V(2) = 9.922115064

V(1) = 10

V(4) = 5.75843106

V(6) = -10

V(7) = -0.4014731114

I(3001) = 1.714295548E-5

I(3002) = -0.009598526889

I(3003) = -0.008638907752
```

```
Trial 2:
EVABS = .001
EVREL = .01
EIABS = 1E.N66
EIREL = .01

CPU time = 6:30 (in minutes:seconds)

Operation count = 969950

Statement count = 278226
```

```
V(3) = -0.1

- V(2) = 9.922115064

V(l) = 10

V(4) = 5.758431061

V(6) = -10

V(7) = -0.4014731123

1(3001) = 1.714295548E-5

1(3002) = -0.009598526888

K30Q3) = -0.008638907751
```

```
Trial 3:
EVABS = .01
EVREL = .1
EIABS = 1E.NG5
EIREL = .1

CPU time = 5:00 (in minutes:seconds)

Operation count = 685241

Statement count = 195558
```

```
V(3) = -0.1

V(2) = 9.922115122

V(l) = 10

V(4) = 5.75843422

V(6) = -10

V(7) = -0.4014802619

1(3001) = 1.714294262E-5

1(3002) = -0.009598519738

1(3003) = -0.008638901316
```

The above data indicates a small correlation between error and correctness of the solution however, this will not always be the case. Obviously, for this circuit and similar circuits, the user should use a relatively lenient set of error parameters. For more complex circuits though, especially those with many energy storage elements, the user will probably want to enforce a tighter error constraint.

The correlation between CPU time and allowable error makes it clear that the user will want to choose the most lenient constraints possible which still yield useful results. It is also clear that even with lenient constraints, CPU times are very substantial. ASP alleviates this problem by giving the user enough flexibility to eliminate all unnecessary simulation time and error constraints. The user can choose his/her own error tolerances simply by setting the global variables EVREL, EVABS, EIREL, and EIABS in the ASP function. These variables correspond to the allowable relative and absolute voltage and current errors. ASP also eliminates any unnecessary dc simulations by prompting the user for the operating point before a transient simulation. The user can either specify the operating point or ask ASP to determine it. If the user can specify the operating point, the CPU time necessary for a dc simulation can be applied to more constructive problems. In addition, ASP allows the user to specify break points to eliminate unnecessary simulation time. The user can then continue or stop the simulation dependent on the previous results. Finally, after a break point has been reached, the user has the option of easily changing circuit data. If only minor changes have been made, ASF will iterate to the new operating point without ramping the independent sources. This will also save a considerable amount of CPU time.

ASP then was created as a relatively efficient simulator in a slow APL environment. Many features have been added to increase the productivity of the simulator and to eliminate unnecessary CPU usage. It is hoped that the combination of these features with the interactive checking and on-line help will make ASP a useful tool.

APPENDIX A
STAMPS

Current Controlled Current Source Stamp

CCCSM N1 N2 N3 N4 X

	N1	N2	N3	N4	100M	RHS
N1					X	
N2					-X	
N3					1	
N4					-1	
100M			1	-1		

In the above stamp syntax, N1...N4 indicate nodes, 100M indicates a CCCS current in the ASP syntax, and RHS indicates the right hand side terms.

Current Controlled Voltage Source Stamp

CCVSM Nl N2 N3 N4 X

	Nl	N2	N3	N4	200M	200M	RHS
Nl					JL		
K2					-1		
N3						·1	
N4						-1	
200K	1	-1				*-X	
200K			1	-1			

Capacitor Stamp in Element Form

CK Nl N2 X

The second order stamp for a capacitor of this form is:

```
        a current source of value
                (X%H[l])#(lst order prediction of capacitor voltage)
             +  (X%H[2])#(2nd order prediction of capacitor voltage)

in parallel with

        a conductance of value
                (X%H[1])+(X%H[2])
```

Inductor Stamp in Element Form

LK Nl N2 X

The second order stamp for an inductor of this form is:

```
        a voltage source of value
                (X%H[l])#(lst order prediction of inductor current)
              • (X%K[2})#(2nd order prediction of inductor current)
```

in series with

```
        a conductance of value
                (X%H[1])+(X%H[2])
```

Gyrator stamp

GYRAK Nl N2 N3 N4 X

	Nl	N2	N3	N4	RHS
Nl			X	-X	
N2			-X	X	
N3	-X	X			
N4	X	-X			

Current Source Stamp

IK N1 N2 X

	N1	N2	RHS
N1			-X
N2			X

Resistor Stamp

RM N1 N2 X

	N1	N2	RHS
N1	1%X	-1%X	
N2.	-1%X	i%>:	

Voltage Source Stamp

VM N1 N2 X

	N1	N2	300M	RHS
N1			1	
N2			-1	
300M	1	-1		X

Voltage Controlled Current Source

VCCSM N1 N2 N3 N4 X

	N1	N2	N3	N4	RHS
N1			X	-X	
N2			-X	X	
N3					
N4					

Voltage Controlled Voltage Source

VCVSM Nl N2 N3 N4 X

	Nl	N2	N3	N4	400H	RHS
Nl					1	
N2					-1	
N3						
N4						
400M	1	-1	-X	X		

APPENDIX B
FUNCTION LEVEL DESCRIPTION OF ASP

In order to facilitate modifications to ASP and to enhance the readability of the APL code, the following function level description has been written. Functions will be grouped according to functionality and each group will be briefly described.

B.1 Mainstream Functions

```
*********************************************************
*                          ASP                          *
*   The introductory function.  ASF initializes variables*
* and asks the user for his input type choice.          *
* ASP then calls one of the INPUT functions.            *
*********************************************************

                           +
                           +
                           +

*********************************************************
*                         INPUT                         *
*   The INPUT functions allow the user to input circuit  *
* data.  INPUT functions are:  INTERINP and FILEREAD.   *
* INPUT functions set MODE to 'add' and call the CHANGE  *
* function.                                             *
*********************************************************

                           +
                           +
                           +

*********************************************************
*                         CHANGE                        *
*   The CHANGE function decides what type of element the *
* user is trying to specify.  CHANGE then calls the pro- *
* per CHECKING function.                                *
*********************************************************

                           +
                           +
                           +

*********************************************************
*                        CHECKING                       *
*   The CHECKING functions check element specifications  *
* for proper syntax.  CHECKING functions are:  TWOCHECK, *
* FOURCHECK, QCHECK, TRANCHECK, CONCHECK, PRNTCHECK,     *
* BREAKCHECK, and MACHECK.  If proper syntax has not     *
* been followed, the user will be given an error message *
* and the INPUT function will attempt to read the next   *
* line.  If proper syntax has been followed, CHANGE will *
* call the appropriate ADD function.                     *
**********************************************************
                           +
                           +
                           +
**********************************************************
*                          ADD                           *
*   The ADD functions add element stamps to the jacobian *
* matrix.  ADD functions are:  ADVCVS, ADVCCS, ADV,      *
* ADCCVS, ADCCCS, ADI, ADRES, ADGYR, ADQ, ADCL, and      *
* ADMAC.  Some of the simple element additions are       *
* handled in the CHECKING functions.  After the ADD      *
* functions are finished, the INPUT functions attempt to *
* read the next line of input.                           *
**********************************************************
                           +
                           +
                           +
```

If there is input on the next line, CHANGE is called and the process repeats until all input has been entered. Once this occurs, the UTILITIES function takes over.

```
                           +
                           +
                           +
**********************************************************
*                       UTILITIES                        *
*   The UTILITIES function allows the user to choose one *
* of the following seven options:                        *
*                                                        *
*       Option 1 - Add more circuit elements.            *
*          UTILITIES calls INTERINP and the user can add *
*          elements as before.                           *
*                                                        *
*       Option 2 - Delete some existing elements.        *
*          UTILITIES calls the function DROP.  This      *
*          function sets MODE to 'drop' and calls the    *
*          CHANGE function.  The CHANGE function then    *
*          calls the proper DELETE function.             *
*                            +                           *
*               DELETE functions drop element stamps     *
*              from the jacobian matrix.  DELETE func-   *
*              tions are:  DPVCVS, DPVCCS, DPV, DPCCVS,  *
*              DPCCCS, DPI, DPRES, DPGYR, DPQ, DPCL,     *
*              DPTRAN, DPMAC, DPCON and DPBREAK.         *
*                                                        *
*       Option 3 - Examine the input to date.            *
*          UTILITIES exhibits the variable INP which     *
*          contains all user input to date.              *
*                                                        *
*       Option 4 - Write the input to an external file.  *
*          UTILITIES calls the function OUTFILE which    *
*          allows the user to write the input to an      *
*          external file.                                *
*                                                         *
*       Option 5 - Continue processing.                   *
*          UTILITIES calls the SETSIM function (see       *
*          below) to set up the simulation.               *
*                                                         *
*   *   Option 6 - Exit                                   *
*          Leave the ASP package.                         *
*                                                         *
*       Option 7 - Return to breakpoint menu if applic-   *
*       able.                                             *
*          The user will return to a previous menu if     *
*          possible.                                      *
***********************************************************
                           +
                           +
                           +
***********************************************************
*                        SETSIM                           *
*   SETSIM does the following to prepare for simulation:  *
*                                                         *
*       - Calls the function HANGELCK to check if all     *
*         necessary information has been specified.       *
*         HANGELCK also checks for hanging nodes in the   *
*         circuit.                                        *
*                                                         *
*       - Calls the SEGREGATE function to distinguish     *
*         between node voltages and branch currents.      *
*         This is done to allow-separate criteria for     *
*         voltage and current error.                      *
*                                                         *
*       - Calls the PIVOT function which is part of the   *
*         sparse matrix package.  PIVOT pivots the        *
*         jacobian to reduce fill.                        *
*                                                         *
*       - In the case of a transient analysis, the user   *
*         is prompted for the operating point.  If it is  *
*         known, a dc analysis can be eliminated.  If     *
*         the operating point is unknown, ASP solves for  *
*         it using the DCSIM function.                    *
***********************************************************
                           +
(Note:  If a dc analysis is being done, DCSIM will be
called.  DCSIM is also called in case of an unknown oper-
ating point.  DCSIM is not called when a transient
analysis is being done and the operating point of the
circuit is known to the user.)
                           +
***********************************************************
*                         DCSIK                           *
*   DCSIM is the function called to provide a dc Simula-  *
* tion.  DCSIM transforms each constant source into a     *
* transient source which reaches its value at time=.ls.   *
* Each transient source is then transformed into a tran-  *
* sient source which reaches its 0 value at time=.ls.     *
* That is each source is ramped up to its 0 value over a  *
* .Is interval.  Each of the 'transient1 sources begins   *
* with a value of 0.  The solution of the circuit is      *
* obvious at- this point.  By taking small time steps and *
* doing an evaluation at each point, the dc solution can *
* be found.  DCSIM sets up this transient simulation     *
* and calls TRANSIK to implement it.  After the dc       *
* solution is found, TRANSIM may be called.              *
**********************************************************
                            +
```

(Note: If only a dc simulation is being done, ASP stops at this point• Continue for more on transient simulations.)
OUTASK is called here to ask the user which voltages or currents should be printed.

```
                           '+
**********************************************************
*                        TRANSIK     -                   *
*   TRANSIK does the following to implement a transient  *
* simulation:                                            *
*      - Initialize variables necessary for simulation.  *
*                                                        *
*      - Call the STARTUP function to start off the      *
*        simulation.  STARTUP essentially does three     *
*        first order corrections while the error         *
*        estimation facility is being initialized.       *
*        STARTUP initially calls the NOCLSAVE function   *
*        which saves the values of positions in the      *
*        jacobian which are affected by capacitors and   *
*        inductors.                                      *
*                                                        *
*      - BEGIN the LOOP                                  *
*              Reset variables.                          *
*                                                        *
*              Call UPDATRAN to update transient source  *
*              stamps.                                   *
*                                                        *
*              Call PRED to predict using the Van  Bok-  *
*              hoven algorithm described in Chapter 2.   *
*                                                        *
*              Call CORRECT to correct using the Van     *
*              Bokhoven algorithm from Chapter 2.        *
*                                                        *
*              Call NEWRAF tc implement the Newton-      *
*              Raphson algorithm.                        *
*              NEWHAP calls NOQSAVE to save values of    *
*              positions in the jacobian which are af-   *
*              fected by nonlinear elements.  A loop is  *
*              then begun:                               *
*                      NOQRESTOR is called tc restore    *
*                      the jacobian to its pre-stamp     *
*                      condition.                        *
*                                                        *
*                      UPDATQ is called to update the    *
*                      nonlinear stamp.                  *
*                                                        *
*                      GAUS is called to do the LU fac-  *
*                      torization.  BACK and FOR are     *
*                      called to do backward and. for-   *
*                      ward, substitution.               *
*                                                        *
*                      Error is then checked to see if   *
*                    the iterations should continue.  *
*                                                     *
*              If not, NOQRESTOR is called again to   *
*              restore the jacobian.                  *
*                                                     *
*              Error is checked to see if the step    *
*              should be accepted or rejected.  If the *
*              step is rejected, cut the step in half, *
               restore the jacobian to its original   *
               condition and return to the beginning of *
Ik             the loop.  If the step is accepted,    *
*              continue with:                         *
.                                                     *
Ik             Restore each transient current source  *
Ik             With TRANIRESTOR.                      *
Ik                                                    *
Ik             Evaluate the design constraint if nec- *
Ik             essary.                                *
Ik                                                    *
Ik             Increment time and restore the jacobian *
Ik             using NOCLRESTOR.                      *
Ik                                                    *
*              Accommodate any printing requests or   *
Ik             break point requests.                  *
*                                                     *
*              Calculate a proposed step size using   *
Ik             HCALC and continue the loop until the  *
*              end time is reached.                   *
*******************************************************
```

B.2 Utility Functions

In addition to the functions already mentioned, there are other Utility functions in ASP. These functions handle small tasks which are necessary for the Mainstream functions to work effectively. A list of these functions and a small discussion on each will now be given.

```
ASSIGN - taxes the user-specified nodes and converts them into
         node numbers which the simulator can user ie.
         consecutive numbers.

BKPNTADD - prompts the user for break points to be added.
           This function is called from BKPNTUTILS.

BKPNTDROP - prompts the user for break points to be dropped.
            This function is also called from BKPNTUTILS.

BKPNTUTILS - gives the user options to change break point data,
             change circuit element data, or continue with
             simulation.

CAPV - determines voltage across two nodes for a given order of
       prediction.
```

COMPACT - drops unused nodes from consideration.

CREATE - is called by PIVOT to create temporary fill elements,
(SPARSE MATRIX FUNCTION)

DESTROY - destroys positions in the jacobian.
(SPARSE MATRIX FUNCTION)

EXAMINE - allows the user to examine present values of design
constraints, voltages, or currents. This function is
called from BKPNTUTILS.

FASTADI - adds a current source for a capacitor stamp.

FASTADRES - adds a resistor for a capacitor or inductor stamp.

FINDINP - finds total element information when given type and
number of element.

FN3EVAL - evaluates the objective function at a given time.

FPWL - returns value of PWL function when given time and
parameters.

FSIN - returns value of SIN function when given time and
parameters.

FUNCK - checks if a given function exists in the workspace.

INCR - increments values of elements of the jacobian.
(SPARSE MATRIX FUNCTION)

IKIT - initializes variables.
(SPARSE MATRIX FUNCTION)

LOOKUP - looks up elements in the jacobian when given row and
column indices.
(SPARSE MATRIX FUNCTION)

*HUK - returns current value of the given voltage or current.

PRINT - prints out the ASP workspace.

PUT - puts a given value into a specified row and column in the
jacobian.
(SPARSE MATRIX FUNCTION)

PWLCK - checks the PWL portion of a PWL transient source.

REDUCE - removes a row and column pair from the jacobian.
(SPARSE MATRIX FUNCTION)

ROW - calculates a given row of the LU factorization.
(SPARSE MATRIX FUNCTION)

*SINCK - checks the SIN portion of a SIN transient source.

TRANSBACK - does the back substitution for the transposed

```
              jacobian problem.
                   (SPARSE MATRIX FUNCTION)

TRANSFOR - does the forward substitution for the transposed
           jacobian problem.
                   (SPARSE MATRIX FUNCTION)

TRANSOL - solves the transpose problem.
                   (SPARSE MATRIX FUNCTION)

TRANSPOSE - obtains column-wise representation of pivoted L and
            U matrices.
                   (SPARSE MATRIX FUNCTION)
```

This list does omit all of the HELP functions which were previously listed. HELP files all begin with the letters 'HELP'.

APPENDIX C
ASP USER S#MANUAL

ASP

APL SIMULATION PACKAGE

A User's Manual

ASP is a simulation package written in APL by

Gregory D. Jordan

TABLE OF CONTENTS

1. INTRODUCTION

This manual *is* intended for use with ASP which is now implemented on the CMU TOPSC system. A large amount of on-line help is available, however this manual is in most cases a more extensive source of information. Some knowledge of APL is valuable if the user would like to explore all of the capabilities of ASP. As a general tool, however, ASP can be used and enjoyed by everyone.

2 GETTING STARTED

Before loading the ASP workspace, be sure to increase the allowable workspace size by typing:

```
)MAXCORE 200
```

Next, to enter the simulator package, type:

```
)L0AD ASP
ASP
```

The user will now be asked to input data. There are two possible input methods. The user may wish to copy data from a previously created external file or the user may wish to input data interactively. The syntax for each element remains the same in both cases. The user will probably want to use interactive input until he/she is familiar with ASP syntax. When naming external files for ASP to read, it is important to keep file names shorter than 8 characters or else an error will result.

3. INPUT SYNTAX

In this section, each type of element will be examined and input syntax will be discussed. Some general points should be made first.

1. Error tolerances can be set by the user by altering the variables EVREL, EVABS, EIREL, and EIABS in the ASP function. These variables are the relative and absolute error allowable for voltages and currents. That is, the simulator checks that the maximum voltage error at each time step is less than:

   EVABS + EVREL×(the voltage which corresponds to the largest voltage error term)

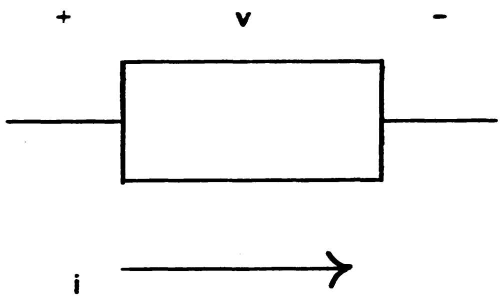
   A similar calculation is done for current error.

2. Scientific notation is acceptable to ASP, however abbreviations are not. That is 1E3 and 1E.NG12 are acceptable, however 1K and 1M are not. When using a TTY terminal, you will not be able to specify any macro element parameters or values less than .0001. In short, this occurs because TTYs convert the (.NG) character to the totally different (-) character. To avoid this problem, either switch to an APL terminal or use individually specified elements rather than macros.

3. Node numbers and element numbers must be positive integers less than 1000. The ground node must be labeled node 0. This is due to an ASP convention for handling internal nodes and currents. No hardship should result since ASP is not designed to handle hundreds of nodes or elements.

4. An acceptable input file consists of circuit elements and a DC or TRANSIENT line to indicate dc or transient simulation. If transient simulation is chosen, the user must also specify a PRINT statement. In addition, the user has the option of specifying break points using a BREAK statement or design constraints using the CON statement.

5. ASP uses the following convention for defining elements:



That is, current always flows from the positive to the negative terminal of an element. In addition, the ASP convention is that the positive terminal is always defined first. Be especially careful of this when defining current sources.

3.1. Resistors

The basic form of resistor input is:

RMN1 N2 VALUE

- where R signifies resistor,
- M is the number of the resistor,
- Nl and N2 are the nodes which the resistor is between,
- and VALUE is the value of the resistor in ohms.

Typical resistor specifications are:

```
Rl 1 2 10000
R5O 67 734 200
```

Checks are run for 0-valued resistors, unacceptable alpha characters, node or element numbers which are not integers or are out of bounds, incorrect number of nodes entered, shorted elements, and duplicate elements, where duplicate means two resistors with the same element number.

3.2. Capacitors and Inductors

The basic form of capacitor or inductor input is:

LM Nl N2 VALUE

CM Nl N2 VALUE

- where L and C signify inductor and capacitor respectively,
- M is the number of the capacitor or inductor,
- Nl and N2 are the nodes which the capacitor or inductor is between,
- and VALUE is the value of the capacitor or inductor in farads or hsnrys.

Typical capacitor and inductor specifications are:

```
Cl 1 5 .00001
L90 35 0 .5798
```

Checks are run for unacceptable alpha characters, incorrect number of node numbers entered, node or element numbers which are not integers or are out of bounds, shorted elements, and duplicate elements, where duplicate means two capacitors or two inductors with the same element number.

3.3. Gyrators

The basic form of gyrator input is:

GYRAM Nl N2 N3 N4 VALUE

- where GYRA signifies 'gyrator.
- M is the number of the cyrator.

- Nl, N2, N3, and N4 are the nodes which the gyrator is connected to,
- and VALUE is the value of the unitless multiplier.

A brief discussion is in order here. Nl and N3 correspond to the positive ends of the voltaees and N2 and N4 correspond to the negative ends. The equation of the gyrator can be written as:

I * -(VALUEHV1-V2)
1^ * (VALUEXV3-V4)

That is, the current on each side of the gyrator is proportional to the voltage on the other side. The proportionality constant is'' «7- VALUE.

Typical gyrator specifications are:

```
GYRA1 10 3 0 5
GYRA43 9 15 34 22 10
```

Checks are run for for unacceptable alpha characters, element numbers or nodes which are not integers or are out of bounds, incorrect number of nodes specified, shorted gyrators, and duplicate elements, where duplicate means two gyrators with the same element number.

3.4. Linear Current Controlled Current Sources

The basic form of linear current controlled current sources is:

CCCSM Nl N2 N3 N4 VALUE

- where CCCS indicates a linear current controlled current source,
- M is the number of the current controlled current source,
- Nl, N2, N3, and N4 are the nodes which the CCCS is connected to,
- and VALUE is the value of the unitless multiplier.

Nodes Nl and N2 are connected to the controlled source. Nodes N3 and N4 are connected to the controlling current Nl and N3 denote <•> nodes and N2 and N4 denote <-> nodes. It is important to note that N3 and N4 are actually shorted nodes. That is, no other element may connect nodes 3 and 4 since the simulator internally puts a 0-valued voltage source across these nodes to determine current The equation of the CCCS may be written:

I = (VALUE)!

That is, the current on one side of the CCCS is proportional to the current on the other side. The proportionality constant is VALUE.

Typical CCCS specifications are:

```
CCCS1 1 2 3 4 50
CCCS67 23 456 32 123 2
```

Checks are run for unacceptable alpha characters, elemenl numbers or nodes which are not integers or are out of bounds, shorted CCCSs (note that nodes 3 and 4 may not be the <u>same</u>

node number), incorrect number of nodes specified, and duplicate elements, where duplicate means two CCCSs with the same element number.

3.5. Linear Current Controlled Voltage Sources

The basic form of linear current controlled voltage source input is:
CCVSM N1 N2 N3 N4 VALUE

- where CCVS signifies current controlled voltage source,
- M is the number of the current controlled voltage source,
- N1, N2, N3, and N4 are the nodes which the CCVS is connected to,
- and VALUE is the value of the multiplier in ohms.

Nodes N1 and N2 are connected to the controlled voltage source. Nodes N3 and N4 are connected to the controlling current. N1 and N3 denote (+) nodes and N2 and N4 denote (-) nodes. Once again, nodes 3 and 4 are shorted because of the internal 0-valued voltage source. The equation of the CCVS may be written:

$$(V1 - V2) = (VALUE)I_{3\to4}$$

That is, the voltage on one side of the CCVS is proportional to the current on the other side. The proportionality constant is VALUE.

Typical CCVS specifications are:

```
CCVS1 1 2 3 4 10
CCVS34 98 333 52 772 2
```

Checks are run for unacceptable alpha characters, element numbers or nodes which are not integers or are out of bounds, incorrect number of nodes specified, shorted CCVSs (nodes 3 and 4 may not be the same number), and duplicate elements, where duplicate means two CCVSs with the same element number.

3.6. Linear Voltage Controlled Current Sources

The basic form of linear voltage controlled current sources is:
VCCSM N1 N2 N3 N4 VALUE

- where VCCS signifies voltage controlled current source,
- M is the number of the voltage controlled current source,
- N1, N2, N3, and N4 are the nodes which the VCCS is connected to,
- and VALUE is the value of the multiplier in mhos.

Nodes N1 and N2 are connected to the controlled current source. Nodes N3 and N4 are connected to the controlling voltage. Nl and N3 denote (+> nodes and N2 and N4 denote (-> nodes. Any element may be placed between nodes 3 and 4. The equation of the VCCS may be written:

I_{1-2} - VALUE(V3 - V4)

That is. the current on one side of the VCCS is proportional to the voltage on the other side. The proportionality constant is VALUE.

Typical VCCS specifications **are:**

```
VCCS1 2 3 4 5 10
VCCS23 34 56 67 87 2
```

Checks are run for unacceptable alpha characters, element numbers or nodes which are not integers or are out of bounds, incorrect number of nodes specified, shorted VCCSs and duplicate elements, where duplicate means two VCCSs with the same element number.

3.7. **Linear Voltage Controlled Voltage Sources**

The basic form of linear voltage controlled voltage source input is:

VCVSM Nl N2 N3 N4 VALUE

- where VCVS signifies voltage controlled voltage source.
- M is the number of the voltage controlled voltage source,
- Nl, N2, N3, and N4 are the nodes which the VCVS is connected to,
- and VALUE is the value of the unitless multiplier.

Nodes Nl and N2 are connected to the controlled voltage source. Nodes N3 and N4 are connected to the controlling voltage. Nl and N3 denote (+) nodes and N2 and N4 denote (-> nodes. Any element may be placed between nodes 3 and 4. The equation of a VCVS may be written as:

(VI - V2) = VALUE(V3 - V4)

That is, the voltage on one side of the VCVS is proportional to the voltage on the other side. The proportionality constant is VALUE.

Typical VCVS specifications are:

```
VCVS1 1 2 3 4 10
VCVS34 45 65 789 12 2
```

Checks are run for unacceptable alpha characters, element numbers or nodes which are not integers or are out of bounds, incorrect number of nodes specified, shorted VCVSs, and duplicate elements, where duplicate means two VCVSs with the same element number.

3.8. Constant Voltage Sources

The basic form of constant voltage source input is:

```
VM.N1 N2 VALUE
```

- where V signifies constant voltage source,
- M is the number of the voltage source.
- Nl and N2 are the nodes which the voltage source is between,
- and VALUE is the value of the voltage source in volts.

Nl is the positive node of the voltage source and N2 is the negative node. The voltage source current is defined as flowing from node 1 to node 2. Typical voltage source specifications are:

```
VI 1 0 5
V23 456 34 15
```

Checks are run for unacceptable alpha characters, node or element numbers which are not integers or are out of bounds, shorted elements, incorrect number of nodes entered, duplicate constant voltage sources and duplicate transient voltage sources, where duplicate means two voltage sources with the same element number.

3.9. Constant Current Sources

The basic form of constant current source input is:

```
IM Nl N2 VALUE
```

- where I signifies constant current source.
- M is the number of the current source,
- Nl and N2 are the nodes which the current source is between,
- and VALUE is the value of the current source in amps.

The current from the current source is defined as flowing from node 1 to node 2 through the current source. Typical current source specifications are:

```
II 1 2 5
134 27 0 10
```

Checks are run for unacceptable alpha characters, node or element numbers which are not integers or are out of bounds, incorrect number of nodes entered, shorted elements, duplicate constant current sources and duplicate transient current sources, where duplicate means two current sources with the sama element number.

3.10. Transient Voltage Sources

The basic form of transient voltage source input is:
TRANVM Nl N2 TYPE PARAMETERS

- where TRANV signifies transient voltage source,
- M is the number of the transient voltage source,
- Nl and N2 are the nodes which the transient source is between.
- TYPE indicates the type of transient source, (one of PWL &. SIN)
- and PARAMETERS are parameters required by the specific type.

Nl is the positive node of the voltage source and N2 is the negative node. Current flows through the transient source identically to the constant source. For TYPE specifications, see section 3.11.1 on allowable type information.

Typical transient voltage source specifications are:

```
TRANV1 1 0 PWL 0 C 1 10 2 10 3 20
TRANV34 45 93 SIN 0 1 I 0 0
```

Checks are run for unacceptable alpha characters, element numbers or nodes which are net integers or are out of bounds, incorrect number of nodes specified, shorted transient sources, incorrect numbers of parameters specified, incorrect types specified, duplicate transient voltage sources and duplicate constant voltage sources, where duplicate means two voltage sources with the same element number.

3.11. Transient Current Sources

The basic form of transient current source input is:
TRANIM Nl N2 TYPE PARAMETERS

- where TRANI signifies transient current source,
- M is the number of the transient voltage source,
- Nl and N2 are the nodes which the transient source is between,
- TYPE indicates the type of transient source (one of PWL 4: SIN),
- and PARAMETERS are parameters required by the specific type.

For TYPE specifications, see the following section on allowable types. Nl is the positive node of the current source and N2 is the negative node.

Typical transient current source specifications are:

```
TRANI1 1 0 PWL 0 0 1 10 2 10 3 20
TRANI34 45 93 SIN 0 1 1 0 0
```

Checks are run for unacceptable alpha characters, element numbers or nodes which are not integers or are out of bounds, incorrect number of nodes specified, shorted transient sources, incorrect numbers of parameters specified, incorrect types specified, duplicate transient current sources and duplicate constant current sources, where duplicate means two current sources with the same element number.

3.11.1. **Allowable Types**

PWL - indicates a piece-wise linear source. Some even number of parameters must be specified ie.(Tl VI T2 V2 . . .TN VN). At Tl and before, the value of the source is VI. From Tl to T2, ASP linearly interpolates to find the appropriate voltage or current At T2. the value of the source is V2. ASP continues the linear interpolation until the end of the simulation is reached. If TN is reached before the end of the simulation, the source value remains VN. Ts are specified in seconds and Vs are specified in volts or amps.

SIN - indicates a sinusoidal. source. Five parameters must be specified ie.(offset, amplitude, frequency, delay, damping factor) Parameters are specified in volts or amps, Hz, and seconds. The value of the source can be determined by the equation:

V = offset • (amplitude>$e^{d*mpinrriU}$(sinpart)

where sinpart is:

sinpart = sin(2(pi X frequency >td>

and td is:

(lime - delays

3.12. **Nonlinear Elements**

The basic form of nonlinear element input is:

QM Nl N2 N3 . . . NN MODNAM PI P2 . . . PM -

- where Q signifies nonlinear element,
- M is the number of the nonlinear element,
- Nl, N2, . . . NN are the nodes connected to the nonlinear element,
- MODNAM is the name of the model used for the element,
- and PI, P2. . . . PM are parameters to be used by the model function.

For model definition information, see section 3.12.1. Parameters should only be specified for a model function which expects parameter input. If the function does not expect parameters, none should be specified.

Typical nonlinear element specifications are:

```
Q1 34 21 52 EMOLLA .00000002 56 32
Q43 98 2 DIODE
```

Checks are run for illegal characters in the node number portion, too few nodes specified, element numbers or nodes which are not integers or are out of bounds, use of undefined model functions and duplicate nonlinear elements, where duplicate means two nonlinear elements with the same element number.

3.12.1. Model Definition

Two APL functions must be created in order for the user to define a nonlinear element model. For purposes of illustration, the first function shall be called MODNAM and the second function shall be called DMODNAM. The only actual constraint on these two function names is that they should not contain any numeric characters and DMODNAM should always be the same as MODNAM with a 'D' in front. Some examples of model function names are DIODE,DDIODE and EMOLL,DEMOLL.

The MODNAM function defines the currents which leave each node of the nonlinear element. If an N terminal element is specified, only N-1 currents need to be defined though because of Kirchhoff's Current Law. MODNAM should in this case return an N-1 element vector which defines the current leaving the first N-1 nodes and flowing through the element. Each current is a function of the node voltages which can be accessed as V[1], V[2] . . . V[N]. Parameters can be passed to the functions if they are listed in the nonlinear specification. If no parameters are to be passed, the functions MODNAM and DMODNAM should not expect any.

As an example of a diode specification with one parameter passed:

QM N1 N2 DIODE .000000001

The DIODE function is:

```
.DL FUNC_DIODE VAL
FUNC_1.R̄O 0
FUNC[̄1]_VAL#((.NG1)+*(V[1]-V[2])%.026)
.GO 0
.DL
```

Important features to note include:

- DIODE takes VAL as an argument. In this case, VAL is a scalar. If the user wishes to pass more than one parameter, VAL could be a vector with parameters accessed as VAL[1], VAL[2], etc.

- Since the diode is a two terminal element, DIODE returns a one element vector.
- The voltage at node Nl is accessed as V[l]. The voltage at node N2 is accessed as V[2].
- FUNC[1] refers to the current leaving node Nl and flowing through the diode.

The other function to be specified is the DMODNAM. This function specifies the derivatives of each current with respect to each node voltage. DMODNAM then should return a N-l x N matrix where the [P;Q] element refers to the derivative of the Pth current with respect to the Qth voltage.

Consider again the specification for diodes.
The DDIODE function is:

```
•DL DFUNC_DDIODE VAL
DFUNC 1 2.RO C
DFUNCll;l]  (VAL%.026)#(*(V[l]-V[2])%.026)
DFUNC[l;2]~((0-VAL)%.026)#(*(V[l]-V[2])%.026)
.GO 0
.DL
```

3.13. Macros

The basic form of macro input is:
MACM Nl N2 . . . NN FUNCNAME PI P2 . . . PT

- where MAC signifies a macro,
- M is the number of the macro,
- Nl, N2, . . . NN are external nodes to which the macro is connected,
- FUNCNAME *is* the name of the function which defines the macro (see 3.13.1),
- and PI, P2, . . . PT are the parameters of the macro.

Macros are clusters of pre-defined elements. Macros are useful when defining sophisticated transistor models or when a circuit is very repetitive. Definition of macros is more difficult than normal input however, so the user should be very careful and expect some difficulty. When using the same macro model more than once, the user should NOT drop a macro which contains voltage or current sources. These sources present difficulties because each source current wiil have the same number in the ASP current numbering system (see Appendix I).
Typical macro specifications are:

```
MAC1 1 2 KACRO 1.2 3.87
MAC87 65 80 71 6 MACNAME 1 10 100 .
```

Checks on macro specifications are not very extensive because of the great flexibility the user has. Checks are run for fewer than two nodes, node and element numbers which are not mteeers or are out of bounds, nonexistent FUNCNAME functions, and duplicate macros, where duplicate means two macros with the same element number.

3.13.1. Macro Definition

To define a macro, one must define a function which lists all the elements of the macro. Basically, the syntax is the same as for normal input however there are some exceptions.

- Usually a macro uses internal nodes. To indicate an internal node, use the global variable INTERNAL. That is, allowable internal nodes are (INTERNAL*1), (INTERNAL+45) etc. * Never use an internal node that is numbered less than INTERNAL.

- To specify an external node, use the variable EXTERNAL. This variable is a vector which contains the nodes N1, N2, . . . NN which were named in the macro specification. Allowable external nodes are (EXTERNAL[1]>, <EXTERNAL[2]), . . . <EXTERNAL[N]>.

- All nodes must be enclosed in parentheses. All numbers and numeric expressions must be separated by commas. No parentheses should be used when specifying parameters.

- Parameters which are passed to the macro may be used by referring to them as VAL[1], VAL[2] . . . VALET].

An example of an acceptable two external, two internal node macro is:

```
MAC1 1 2 MACNAME 1.2 3.87

AND

.DL MACRO
Rl,  (EXTERNAL[l3),(INTERNAL),VAL[l]
TRANV1#  (INTERNAL),  (INTERNAL+1) PWL 0,0,l,VAL[2]
R2,  (INTERNAL+1)r(EXTERNAL[2]),1000
.DL
```

The resulting circuit entry from this macro would be:

```
Rl 1 7001 1.2
TRANV1 7001 7002 PWL 0 0 1 3.87
R2 7002 2 1000
```

where 7001 and 7002 refer to internal nodes according to ASP syntax.

4. NECESSARY INPUT

The user must choose between a dc analysis and a transient simulation. The operating point must be found for each type of simulation. If a transient simulation is chosen however, the user will have the option to supply the operating point. This will then save the user a needless dc simulation. If the user does not know the operating point, ASP will solve for it and the transient analysis will follow.

4.1. DC Analysis

To specify dc analysis, the user must input the line:

```
DC
```

A dc analysis will be done and all node voltages and the currents from inductors, voltage sources and some dependent sources will be displayed. For an explanation of the output syntax for currents, see Appendix I.

4.2. Transient analysis

To specify a transient analysis, the user must input the line:

```
TRANSIENT
```

It is also necessary to specify the time of simulation and the printing increment desired. These can be set using the format:

```
PRINT PRINTINCR ENDTIME
```

- where PRINT indicates that the user is setting print parameters,
- PRINTINCR is the time interval between each time ASP prints out results,
- and ENDTIME is the time when the simulation should halt.

ASP will only print out the voltages and currents which the user specifies. The user can specify voltages and currents to output after all circuit elements have been entered and the choice is made to continue simulation. For an explanation of the output syntax for currents, see Appendix I.

5. OPTIONAL INPUT

5.1. Break Point Specification

In the course of the simulation, break points may be specified where ASP halts and the user has options to change previous simulator input Suppose the ENDTIME of the simulation is 1 sec. Furthermore, suppose the user will be able to determine the success or failure of the test before the simulation is complete. Using ASP, the user may specify a break point at say .3 sec. and decide if the simulation should continue.

To specify break points, use the format:
BREAK Tl T2 . . . TN

- where BREAK indicates that the user is setting break points,
- and Tl, T2, . . . TN are times at which the user would like the simulator to halt.

When ASP halts at a break point, the user will have the option to add, delete, or examine current breakpoints, examine values of constraints, voltages, or currents, change circuit element data, exit, or continue with the present simulation. The user may only change future break points; he/she may not add or delete break points which have already been passed. If the user decides to change circuit element data, ASP will try to use the old DC solution for the new circuit as well. If this attempt fails, ASP will fmd the new operating point and then begin another transient simulation. If the user chooses the last option, the simulation will continue as if the break had never occurred.

5.2. Design Constraints

The user also has the option of specifying design constraints for the simulator to integrate over time. Design constraints have no effect during the DC analysis however, during the transient analysis, the functions are integrated over time and the results are printed out with the remainder of the output. The user need not ask the simulator to output the design constraint calculations. If a design constraint has been specified, the user will see the value at each printing increment. If no constraint has been specified, no design constraint data will be presented.

To specify a design constraint, use the format;
CONM FUNCNAM

- where CON indicates that a design constraint is being specified,
- M is the number of the design constraint,
- and FUNCNAM is the name of the function containing the design constraint.

For help in defining a design constraint see 5.2.1.

Typical design constraint specifications are:

```
C0N1 TEST
C0N999 FUNC
```

Checks are run for duplicate constraint numbers, undefined constraint functions, and constraint numbers which are not integers or are out of bounds.

5.2.1. Function Definition

To use a design constraint, the user must define a function which returns the value of the constraint at a specific time point To facilitate this, ASP contains the function NUM which serves as an interface between the user defined nodes and currents and the values which the simulator has calculated. NUM accepts numbers which refer to node voltages or currents and returns the value of these voltages and currents at the present time. For help in specifying voltages and currents, see Appendix I.

A typical design constraint definition is:

```
C0N1 FUNC
```

and

```
.DL RESULT_FUNC
RESULT_(((NUK l)-2)*2) + (((NUM 3001)-.001)*2)
.GO 0
.DL
```

APPENDIX I
EXPLANATION OF VOLTAGE AND CURRENT NUMBERING SYSTEM

• Currents and voltages are specified as follows:

• Voltages will appear as their node numbers and can be as high as 999.

• Currents will appear differently depending on the type of element generating the current
CCCS currents - 1000 • element number
CCVS currents - 2000 + element number •
voltage source currents * 3000 + element number
VCVS currents - 4000 + element number
inductor currents - 5000 + element number
inductor voltages - 6000 + element number
Numbers greater than 7000 indicate internal nodes. The only feasible way for the user to determine these is to run the simulation.

Some examples would be:

• 3001 would indicate the current due to voltage source number 1.

• 2048 would indicate the current due to CCVS number 48.

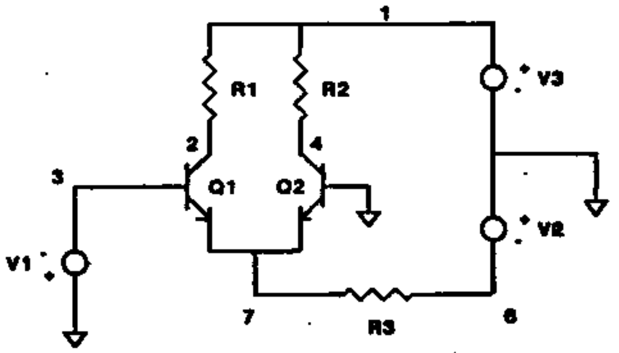
• 7092 would indicate an internal node.

APPENDIX II
TEST CASES

The first test case shows user input and simulator output for a dc simulation. Error parameters are currently:

EVABS s .001
EVREL = .01
EIABS = .000001
EIREL s .01

The circuit is shown below:



<u>TEST CIRCUIT #1</u>

R1: 500 ohms
R2: 500 ohms
R3: 1k ohms

V1: 0.1 volts
V2: 10 volts
V3: 10 volts

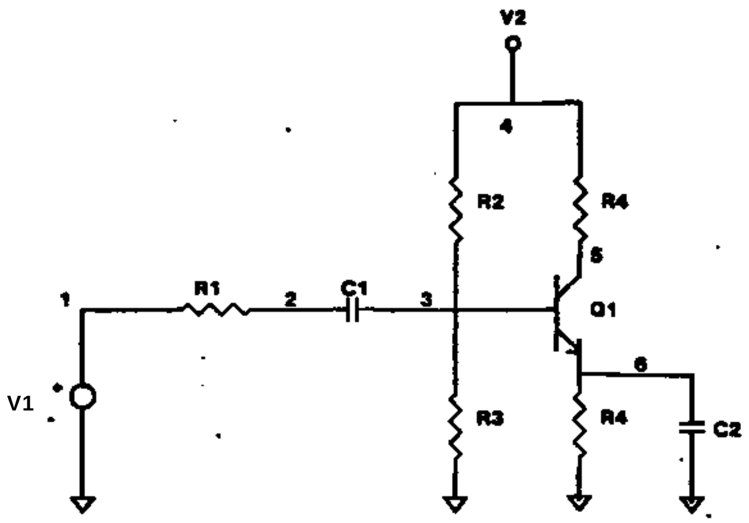
Q1,Q2 model: Emoll

```
@apl
terminal..TTY
APL-20 DECSYSTEM-20 APLSF 2(504)
TTY134)  0:12:41 WEDNESDAY 22-JUL-81 GJOC   [4,452]
CLEAR VS
      )MAXCORE 200
WAS  40P
      )LOAD ASP
SAVED  23:39:22 21-JUL-81 169P
      ASP
WELCOME TO THE APL SIMULATION PACKAGE!  IF YOU ENCOUNTER ANY PROBLEMS
ALONG THE WAY, PLEASE FEEL FREE TO TYPE "HELP-.
WHICH TYPE OF INPUT WOULD YOU PREFER?
           0 - READ FROM AN EXTERNAL FILE
           1 - ENTER DATA INTERACTIVELY


INPUT MODE:  1






YOU WILL NOW BE ASKED TO INPUT THE CIRCUIT DATA.  HELP IS AVAILABLE
IN THIS MODE IF YOU WOULD LIKE.  EACH ENTRY WILL BE CHECKED FOR
ERRORS AND AN APPROPRIATE MESSAGE WILL BE SENT.  IF AN ASTERISK
APPEARS, THE SIMULATOR IS READY TO ACCEPT NEW DATA.  TO
EXIT THE INPUT MODE, TYPE 000.  PLEASE ENTER YOUR DATA NOW.
* VI
THERE IS AN INCORRECT NUMBER OF NODES SPECIFIED.
VI
PLEASE REENTER THIS ELEMENT AS IT HAS NOT BEEN ADDED.
* V.I 1030 1
TEE SECOND CHARACTER MUST BE NUMERIC.
V.I 1030 1
PLEASE RSENTER THIS ELEMENT AS IT HAS NOT BEEN ADDED.
* VI 1050 0 10
NODE NUMBERS GREATER THAN 999 ARE NOT PERMISSABLE.
VI 1050 0 10
PLEASE REENTER THIS ELEMENT WITH APPROPRIATE NODE NUMBERS.
* VI 0 3 .1
* VI 0 6 10
A DUPLICATE ELEMENT EXISTS FOR:
VI 0 5 10
PLEASE REENTER THIS ELEMENT UNDER A NSW NAME.
* V2 0 6 10
* V3 10 10 10
YOU HAVE SPECIFIED THE SAME NODE TWICE.
V3 10 10 10
PLEASE REENTER THIS ELEMENT AS IT HAS NOT BEEN ADDED.
* V3 1 0 10
* Rl 2 1 500
* R2 1 4 500
* R3 6 7 1000
* Ql 7 2 3 EMOLL
* Q2 7 4 0 EMOLL
* DC
* 000
```

```
YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT
OR CHECK THAT THE SIMULATION HAS COMPILED ALL THE DATA.
TO CONTINUE PROCESSING, SIMPLY SELECT OPTION 5.  IF YOU CHOOSE
ANOTHER OPTION, YOU CAN EXIT THAT OPTION AND RETURN TO THIS
MENU BY TYPING 000.

 OPTION 1 - ADD MORE CIRCUIT ELEMENTS

 OPTION 2 - DELETE SOME EXISTING ELEMENTS

 OPTION 3 - EXAMINE THE INPUT TO DATE

 OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

 OPTION 5 - CONTINUE PROCESSING

 OPTION 6 - EXIT

 OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

OPTION => 5
OPERATING POINT NOW BEING DETERMINED
```

```
THE RESULTS OF THE DC SIMULATION ARE:
NODE VOLTAGES:

V(3)     =     -0.1

V(2)     =     9.922115064

V(l)     =     ID

V(4)     =     5.758431061

V(6)     =     -10

V(7)     =     -0.4014731123


BRANCH CURRENTS:

I(3001)     =     1.714295548E-5

I(3002)     =     -0.009598526888

1(3003)     =     -0.008638907751

      )SAVE
  0:42:02 22-JUL-81 166 PGS  ASP <C75> [4,452]
      )OFF HOLD
TTY134)  0:42:15 22-JUL-81
CONNECTED   0:29:34 CPU TIME   0:06:30
278226 STATEMENTS 969950 OPERATIONS

EXIT
@
```

The second test circuit is a simple amplifier. An initial dc analysis is done to determine the operating point. Finally, the transient analysis is done. Error parameters for this test case are the same as for the first test case. The amplifier circuit is shown below:



<u>TEST CIRCUIT #2</u>

R1: 100 ohms
R2: 4K ohms
R3: 1K ohms
R4: 2K ohms
R5: 1K ohms

C1: 100 uF
C2: 1 uF

V1: .Ssin2(pi)t
V2: 15

Q1 model: Emoll

```
@A?L
terminal..TTY
APL-20 DECSYSTEM-20 APLSF 2(504)
TTY134;  5:12:25 WEDNESDAY 22-JUL-81 GJOC   [4,452]
CLEAR WS
      )MAXCORE 200
WAS  40P
      )LOAD ASP
SAVED   4:08:01 22-JUL-81 170P
      ASP
WELCOME TO THE APL SIMULATION PACKAGE!  IF YOU ENCOUNTER ANY PROBLEMS
ALONG THE WAY, PLEASE FEEL FREE TO TYPE -HELP".
WHICH TYPE OF INPUT WOULD YOU PREFER?
           0 - READ FROM AN EXTERNAL FILE
           1 - ENTER DATA INTERACTIVELY

INPUT MODE:  0






ENTER THE INPUT FILENAME:  SIM,DAT
INPUT FILE HAS BEEN READ.
```

(The file sim.dat contains tHe following input;

```
TRANV1 1 0 SIN 0 .5 1 0 0
V2 4 0 15
Rl 1 2 100
Cl 2 3 1E.NG4
R2 4 3 4000
R3 3 0 1000
Ql 6 5 3 EMOLL
R4 4 5 2000
R5 6 0 1E3
C2 6 0 1E.NG6
TRANSIENT
PRINU .01 .15
```
)

```
YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT
OR CHECK THAT THE SIMULATION HAS COMPILED ALL THE DATA.
TO CONTINUE PROCESSING, SIMPLY SELECT OPTION 5.  IF YOU CHOOSE
ANOTHER OPTION, YOU CAN EXIT THAT OPTION AND RETURN TO THIS
MENU BY TYPING 000.

 OPTION 1 - ADD MORE CIRCUIT ELEMENTS

 OPTION 2 - DELETE SOME EXISTING ELEMENTS

 OPTION 3 - EXAMINE THE INPUT TO DATE

 OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

 OPTION 5 - CONTINUE PROCESSING

 OPTION 6 - EXIT

 OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

OPTION => 5
```

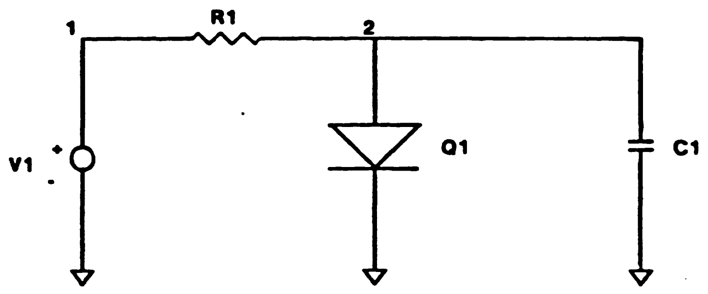
```
IF THE DC SOLUTION IS KNOWN, PLEASE ENTER THE SOLUTION VECTOR
IN THE FOLLOWING ORDER:
3001 1 3002 4 2 3 6 5
OTHERWISE, SIMPLY ENTER A 0.
0
OPERATING POINT NOW BEING DETERMINED.
THE RESULTS OF THE DC SIMULATION ARE:
NODE VOLTAGES:

V(1)       =      0

V(4)       =      15

V(2)       =      6.62032S710E-8

V(3)       =      2.805124689

V(6)       =      2.437090328

V(5)       =      10.61300629


BRANCH CURRENTS:
1(3001)     =     6.620329710E-10

1(3002)     =     -0.005242215684
```

```
TYPE IN DESIRED NODE VOLTAGE OUTPUTS ON SEPARATE LINES.
IF NO MORE ARE DESIRED, ENTER A 0.
*5
*G
```

```
TYPE IN DESIRED CURRENT OUTPUTS ON SEPARATE LIN^S.
IF NO MORE ARE DESIRED, ENTER A 0.
*0
```

```
TIME      V(5)

0.01  10.56251037

0.02  10.52135796

0.03  10.48598946

0-.04 1C-.45456504

0.05  10.42767043

0.06  10.4046288

0.07  10.38531285

0.08  10.36940854

C.09  10.35672836

0.1  1C.34705207

0.11  10.3402017

0.12  10.33600056

0.13  10.33429029

0.14  10.3349194

0.15  10.33774642

0.16  10.34263609
      )SAVE
  5:21:19 22-JUL-81 166 PGS  ASP <075> [4,452]
      )OFF HOLD
TTY134)  5:21:20 22-JUL-81
CONNECTED   0:09:55 CPU TIME   0:08:22
438849 STATEMENTS 1514205 OPERATIONS
```

The final test circuit is a simple circuit with a resistor, diode, and capacitor. This test shows the use of macros, design constraints, and break points. Once again, error parameters are the same and the circuit is shown below:



<u>TEST CIRCUIT #3</u>

R1: 1M ohms / 50K ohms

C1: 100 uF

V1: sin 2(pi)t

Q1 model: Diode

```
@APL
terminal..TTY
APL-20 DECSYSTEM-20 APLSF 2(504)
TTY134) 23:34:29 TUESDAY 21-JUL-81 GJOC   [4,452]
CLEAR VS
      )MAXCORE 200
WAS  40P
      )LOAD ASP
SAVED  23:34:16 21-JUL-81 170P
      ASP
WELCOME TO THE" APL SIMULATION PACKAGE!  IF YOU ENCOUNTER ANY PROBLEMS
ALONG "THE WAY, PLEASE FEEL FREE TO TYPE "HELP".
WHICH TYPE OF INPUT WOULD YOU PREFER?
           0 - READ FROM AN EXTERNAL FILE
           1 - ENTER DATA INTERACTIVELY

INPUT MODE:  0









ENTER THE INPUT FILENAME:  TRTEST,DAT
INPUT FILE HAS BEEN READ.
```

(The file trtest.dat contains the following input:

```
TRANV1 1 0 SIN 0 1 1 0 0
Rl 1 2 1E6
MAC1 2 0 TRANMAC .0001
TRANSIENT
PRINT .01 .15
CON1 TRANOBJ
BREAK .05 .1
```

)

(Another note: The macro and design constraint definitions were previously declared in the workspace. They are now listed below for the purpose of this example.

```
.DL RESULTJTRANOBJ
RESULT_((NUM 2)-.5)*2
.GO 0 "
.DL


.DL TRAKMAC
Ql,(EXTERNAL[1]),(EXTERNAL[2])DIODE
Cl,(EXTERNALCl]),(EXTERNAL[2]),VAL
•DL
```
)

```
YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT
OR CHECK THAT. THE SIMULATION HAS COMPILED ALL THE DATA.
TO CONTINUE PROCESSING, SIMPLY SELECT OPTION 5.  IF YOU CHOOSE
ANOTHER OPTION, YOU CAN EXIT THAT OPTION AND RETURN TO THIS
MENU -BY TYPING 000.

 OPTION 1 - ADD MORE CIRCUIT ELEMENTS

 OPTION 2 - DELETE SOME EXISTING ELEMENTS

 OPTION 3 - EXAMINE THE INPUT TO DATE

 OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

 OPTION 5 - CONTINUE PROCESSING

 OPTION 6 - EXIT

 OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

OPTION => 5
```

```
IF THE DC SOLUTION IS KNOWN, PLEASE ENTER THE SOLUTION VECTOR
IN THE FOLLOWING ORDER:
3001 1 2
OTHERWISE, SIMPLY ENTER A 0.
0 0 0



TYPE IN DESIRED NODE VOLTAGE OUTPUTS ON SEPARATE LINES.
IF NO MORE ARE DESIRED, ENTER A 0.
*2
*1
*0



TYPE IN DESIRED CURRENT OUTPUTS ON SEPARATE LINES.
IF NO MORE ARE DESIRED, ENTER A 0.
*3001
*0
```

```
TIME      V(2)         V(l)         1(3001)      CON1

0.01 5.606510942E-6  0.06279046656 -6.278436005E-8  0.002499950576

0.02 1.515055727E-5  0.1253331283 -1.253179777E-7  0.004999844466

0.03 3.032184939E-5  0.1873811582 -1.873503363E-7  0.007499603524

0.04 5.262128233E-5  0.2486896815 -2.436370603E-7  0.009999172553

0.05 8.048330394E-5  0.309016742 -3.089362587E-7  0.01249849235

YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT OR
BREAKPOINT DATA.
BREAKPOINT MENU:

OPTION 1 - ADD BREAKPOINTS

OPTION 2 - DELETE BREAKPOINTS

OPTION 3 - EXAMINE BREAKPOINTS TO DATE

OPTION 4 - EXAMINE VALUES OF CONSTRAINTS, VOLTAGES OR CURRENTS

OPTION 5 - CHANGE CIRCUIT ELEMENT DATA

OPTION 6 - CONTINUE WITH TRANSIENT SIMULATION

OPTION 7 - EXIT

OPTION => 6


0.05 8.048330394E-5  0.309016742 -3.089362587E-7  0.01249849235

0.06 0.0001156700412 0.3681242566 -3.680085866E-7  0.01499764316

0.07 0.0001554051144 0.4257789554 -4.256235503E-7  0.01749629043

0.08 0.0002007531253 0.481753302 -4.815525489E-7  0.01999450144

0.09 0.0002515734686 0.5358263917 -5.355748182E-7  0.02249222838

0.1 0.0003076779324 0.5377848229 -5.874771450E-7  0.02498942013

0.1 0.0003076779324 0.5877848229 -5.874771450E-7  0.02498942013
```

```
YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT OR
        BREAKPOINT DATA.
        BREAKPOINT MENU:

        OPTION 1 - ADD BREAKPOINTS

        OPTION 2 - DELETE BREAKPOINTS

        OPTION 3 - EXAMINE BREAKPOINTS TO DATS

        OPTION 4 - EXAMINE VALUES OF CONSTRAINTS, VOLTAGES OR CURRENTS

        OPTION 5 - CHANGE CIRCUIT ELEMENT DATA

        OPTION 6 - CONTINUE WITH TRANSIENT SIMULATION

        OPTION 7 - EXIT

        OPTION => 5




        YOU MAY NOW SELECT ANY OF THE OPTIONS BELOW TO MODIFY THE INPUT
        OR CHECK THAT THE SIMULATION HAS COMPILED ALL THE DATA.
        TO CONTINUE PROCESSING, SIMPLY SELECT OPTION 5.  IF YOU CHOOSE
        ANOTHER OPTION, YOU CAN EXIT THAT OPTION AND RETURN TO THIS
        MENU BY TYPING 000.

         OPTION 1 - ADD MORE CIRCUIT ELEMENTS

         OPTION 2 - DELETE SOME EXISTING ELEMENTS

         OPTION 3 - EXAMINE THE INPUT TO DATE

         OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

         OPTION 5 - CONTINUE PROCESSING

         OPTION 6 - EXIT-

         OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

        OPTION => 1
```

```
YOU WILL NOW BE ASKED TO INPUT THE CIRCUIT DATA.  HELP IS AVAILABLE
IN THIS MODE IF YOU WOULD LIKE.  EACH ENTRY WILL BE CHECKED FOR
ERRORS AND AN APPROPRIATE MESSAGE WILL BE SENT.  IF AN ASTERISK
APPEARS, THE SIMULATOR IS READY TO ACCEPT NEW DATA.  TO
EXIT THE INPUT MODE, TYPE 00C.  PLEASE ENTER YOUR DATA NOW.
* R2 1 2 50000

* 000

 OPTION 1 - ADD MORE CIRCUIT ELEMENTS

 OPTION 2 - DELETE SOME EXISTING ELEMENTS

 OPTION 3 - EXAMINE THE INPUT TO DATE

 OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

 OPTION 5 - CONTINUE PROCESSING

 OPTION 6 - EXIT

 OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

OPTION => 2



ON SEPARATE LINES, PLEASE LIST THE ELEMENTS YOU WISH TO DELETE.
TO QUIT, TYPE "000".
* Rl
* BREAK .05 .1
* 000
```

```
 OPTION 1 - ADD MORE CIRCUIT ELEMENTS

 OPTION 2 - DELETE SOME EXISTING ELEMENTS

 OPTION 3 - EXAMINE THE INPUT TO DATE

 OPTION 4 - WRITE THE INPUT TO AN EXTERNAL FILE

 OPTION 5 - CONTINUE PROCESSING

 OPTION 6 - EXIT

 OPTION 7 - RETURN TO BREAKPOINT MENU IF APPLICABLE

OPTION => 7

0.01 1.255809319E-12 0.06279046656 -1.255809331E-6  0.0025

0.02 2.506662553E-12 0.1253331283 -2.506662565E-6  0.005

0.03 3.747623151E-12 0.1873811582 -3.747623164E-6  0.0075

0.04 4.973793619E-12 0.2486896815 -4.973793631E-6  0.01

0.05 6.180334828E-12 0.309016742 -6.180334840E-6  0.0125

0.06 7.362485120E-12 0.3681242566 -7.362485132E-6  0.015

0.07 8.515579097E-12 0.4257789554 -8.515579108E-6  0.0175

0.08 9.635066030E-12 0.481753302 -9.635066041E-6  0.02

0.09 1.071652782E-11 0.5358263917 -1.071652783E-5  0.0225

0.1 1.175569645E-11 0.5877848229 -1.175569646E-5  0.025

0.11 1.274847079E-11 0.6374235399 -1.274847080E-5  0.0275

0.12 1.369093282E-11 0.6845466417 -1.369093283E-5  0.03

0.13 1.457936309E-11 0.7289681551 -1.457936310E-5  0.0325

0.14 1.541025537E-11 0.7705127692 -1.541025538E-5  0.035

0.15 1.618033052E-11 0.8090165265 -1.618033053E-5  0.0375

0.16 1.688654940E-11 0.8443274705 -1.688654941E-5  0.04
      )SAVE
 23:39:22 21-JUL-81 165 PGS  ASP <075> [4,452]
      )OFF HOLD
TTY134) 23:39:31 21-JUL-81
CONNECTED   0:05:02 CPU TIME   0:01:20
50436 STATEMENTS 177797 OPERATIONS
```

References

[Cohen 78] Cohen, E.f Vladmirescu, A., and Pederson, D. 0.,
User's Guide for SPICE
University of California, College of Engineering
Version 2E.0, August 15, 1978.

[Ho 75] Ho, C.# Ruehli, A. E., and Brennan, P. A.
The Modified Nodal Approach to Network Analysis,
IEEE'Transactions on Circuits and Systems
CAS-22, pp. 504-509, June 1975.

[Gustavson 72] Gustavson, F. G.,
Some Basic Techniques for Solving Sparse Systems of
Linear Equations,
in Sparse Katrices and Their Applications
D. J. Rose and R. A. Willoughby, Ed.
New York: Plenum Press, 1972, pp. 41-53.

[Markowitz 57] Markowitz, H. K. ,
The Elimination Form of the Inverse and its Application
to Linear Programming,
Management Science (3):255-269, April 1957.

[Van Bokhoven 75] Van Bokhoven, W. M. G.,
Linear Implicit Differentiation Formulas of Variable
Step and Order,
IEEE Transactions on Circuits and Systems,
Vol. CAS-22, pp. 109-115, February 1975.