An Intelligent System for the Design of Analog Integrated Circuits

John W. Miller

University Undergraduate Fellows, 1986–1987

Texas A&M University
Department of Electrical Engineering
College Station TX 77843

APPROVED

Fellows Advisor: Karan Watson

Fellows Advisor: M. Stybelinski

Honors Director: Lewis Blackwell

An Intelligent System for the Design of Analog Integrated Circuits

Conventional CAD tools are brought into use late in the design process after much of the functional operation of a circuit has been defined. A new type of design environment must be created to aid the engineer from goal creation all the way to silicon fabrication. Symbolic equation generation is an area of active research which will extend the application of computers earlier into the design process. A tool for equation development in an interactive design environment is described. The system successfully meets the following three constraints for a useful interactive symbolic analysis system. First, it is quicker than an engineer at generating the equations. Like the engineer it is capable of handling large circuits without an exponential growth in calculation time. Secondly, it produces equations simplified in the same form as the design engineer would create by hand. Finally, the entire system is easily modified and expanded by the design engineer personally. The equation developing system is based upon the Norton equivalent circuit approximation. Norton equivalence is used to keep the equations linear, to decouple the circuit into more manageable building blocks, and because it results in equations similar to those developed by engineers in hand analysis. A prototype system was implemented using a rule based development program called CLIPS.[1)] The rule based system environment makes extension to the system by non-programmers a simple task. This paper explains the motivation, theory, and implementation of the symbolic analysis system.

[1)] CLIPS was developed by NASA and is available free of charge.

ACKNOWLEDGMENTS

I would like to thank my advisors Dr. Karan Watson and Dr. M.A. Styblinski for their guidance and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

I. INTRODUCTION

The design of an analog integrated circuit can be divided into two stages. The first stage takes the circuit's requirements and goals and develops a circuit configuration. The second stage starts with the configuration, and is completed with the printing of the circuit masks used for production. Figure 1 shows an outline of the design process. The trend in computer aided design (CAD) technology for integrated circuits has been to automate this second stage of design. A new class of CAD tools is needed to aid circuit designers in the first stage. Together with the CAD software available today, the new class of tools will create a design environment to guide the engineer from goal creation all the way to mask generation. In part, this extension of CAD earlier into the design process will require a revolutionary rather than a gradual change in approach to software. The needed change is to use heuristic as well as algorithmic approaches. Adding heuristic rules (rules of thumb) to control a software system results in less certainty and less mathematical precision, but this cost is inherent to the domain of ideas, symbols, and hypothesis found in the first stage of the design process. Intelligent design is therefore the use of heuristics and algorithms to bring out the power of today's hardware and software design tools.

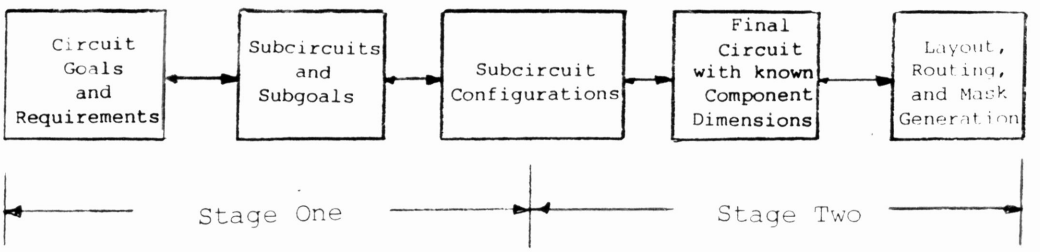


Figure 1. The Design Process

Chapter II of this paper describes this design environment. This envisioned design environment is the motivation for developing the software developed through this research. Chapter III presents a prototype for one of the new tools: an expert system to generate "paper and pencil" equations used by designers. Chapter IV describes how this prototype was implemented using CLIPS. Chapter V tells the lessons learned from the prototype and proposes a direction for development leading to "An Intelligent System for the Design of Analog Integrated Circuits."

II. THE DESIGN ENVIRONMENT

Traditional CAD systems have been limited to helping with the last stage of analog IC design because they have used only algorithmic approaches to solve problems. There is no algorithm to determine what circuit configuration will work best to meet circuit goals. There are, however, heuristics or rules of thumb. Heuristics and other reasoning strategies can serve as the basis for an intelligent design system. A "first generation" intelligent design systems will need to perform the following functions:

(1) It will prepare symbolic equations describing circuit specifications in terms of circuit parameters.

(2) It will maintain a database of design goals. The system will be able to tell if an analyzed design meets these goals.

(3) The system will be an intelligent front end to a set of supporting design tools such as circuit simulators, layout tools, etc. This front end must prepare data and execute these programs automatically when needed.

(4) It will store and use design expertise from a library of circuit descriptions. These descriptions include design information such as the equations derived for the circuit [1].

The envisioned system can be clarified by example. Consider this scenario:

> An engineer starts the system by telling it to begin an operational amplifier design. The computer responds with a list of op-amps and asks if the user would like to review the available designs. The designer chooses an op-amp and is presented a circuit diagram and table of circuit specifications. The engineer points to the specification for maximum offset voltage and lowers the value to 100 microvolts. The computer calculates and then replies that the differential input stage must be redesigned. The engineer asks to see different input stages. The engineer chooses one from a menu and asks to see an equation relating maximum offset voltage to component values...
>
> By the end of the day, the computer sends the final circuit's mask descriptions

to the foundry for production. The new circuit information is stored in a design library for future use.

Significance

The significance of improving computer aided circuit design technology is that it will lower the cost and time of microchip design. Analog circuitry is designed nearly the same way today as 15 years ago. Months are spent writing equations to explain and learn the properties of just one circuit. Automating this step alone will drastically improve productivity, shortening the IC design time from years to weeks or days. The most important benefit for the electronics industry will be a faster pace of technological development. Automated design systems will perform the work we already know well how to do. This will increase the time engineers can spend on things we don't know how to do. Another advantage to industry will be improved management of design projects. The system will require a clear statement of goals and subgoals. The system must track the attainment of these goals. This is a primary function of engineering management.

The design of an intelligent design system is also significant to the field of artificial intelligence. Of interest to the AI community is the application of machine learning to add circuits to the design library. The system could be taught to be creative in its spare time. By this I mean it could make up random circuit "mutations," and see if they can outperform human designed circuits. Improvements would be stored and used as the basis for further mutation. This could lead to an evolution of increasing circuit performance.

This proposed design system also has a significance to other engineering fields. Analog circuitry, as the name implies, can model physical processes in general. Automatic generation of circuit equations is very near to automatic generation of stress and strain equations for a bridge, or reaction equations for a chemical plant for example. For this reason, design advances in the electronics field will be quickly applied in other fields.

Relation to the Current State of Knowledge

The basis for the proposed design system is a circuit analysis program. The purpose of this circuit analysis is to produce design equations and to explain cause and effect within

the circuit. This is not to be confused with a circuit simulator. Simulators, such as SPICE, return numbers rather than equations for the outputs of a circuit [2]. These programs are effective for checking the operation of a well understood circuit. SPICE cannot directly provide the conceptual information required to improve a design. With INTOPT.SPICE it is possible to adjust a parameter interactively while watching output [3]. This is another tool which becomes useful when the designer fully understands the circuit. An interactive SPICE program should be used as one tool within the intelligent design system. Once the design decisions are made, an accurate SPICE simulation can be used to tune the circuit.

An important development for conceptual circuit analysis was the creation of MACSYMA [4]. MACSYMA is a program which can solve graduate level symbolic mathematical problems symbolically. The algorithms from MACSYMA will need to be used in the proposed system to simplify equations for display to the engineer. Several systems have used MACSYMA to develop design equations from a circuit description. SYN is a system using symbolic manipulation with the theory of propagation of constraints [5]. Propagation of constraints is a theory used to limit the number of equations developed from a circuit. Simple calculations are performed first, and variables are chosen to minimize the length of equations. SYN does not break down the circuits into subcircuits. For this reason, the equation simplification algorithm is slowed to a halt as the circuit size grows.

III. A RULE BASED SYSTEM FOR ANALOG CIRCUIT ANALYSIS

A useful tool for equation development in an interactive design environment must meet the following constraints: First, it must be quicker than an engineer at generating the equations. Secondly it must return the equations in a form understood by the engineer. Finally the system should be flexible enough to accept arbitrary user defined additions to the circuit types available in analysis.

Linear Growth in Analysis Time

Previously reported symbolic equation generating systems have not succeeded in obtaining a linear relationship between device count and analysis time. SYN for instance limits the complexity of equations but still is slowed to a halt as circuit size increases. [6]. In contrast, an engineer can quickly derive the transfer functions of complex circuits by utilizing information about well understood building blocks, and by making intelligent simplifying assumptions.

Equations in a Standard Form

Engineers use the form of an equation to imply information about a circuit beyond the algebraic relationships of the equation. For instance in calculating the gain of a cascaded amplifier, each product in the equation is the gain of the individual stages. Putting the equations in an "illogical form" drastically reduces their usefulness to the engineer. For this reason building block circuit analysis should automatically include information about how to form the equations describing the circuit.

Flexibility and Expandability

The design engineer must be able to add new structures to the known building blocks or else the program would quickly fall out of use. Flexibility can be achieved in two ways. One approach is to make the system so general that it is delivered with the power to analyze any circuit. In contrast, the approach used in this research is to provide a system which can analyze a small number of structures, but the structures can be added by the designer in a simple way. The program can thus become a "note pad" of building block circuits the

engineer creates and stores for future reference. The system must be expandable in that expansions can be made without slowing the analysis down significantly.

Motivation for developing the Building Block Algorithm

The most important feature of this algorithm is it produces simple linear equations. Conventional circuit analysis programs use numerical methods which set up a matrix of simultaneous equations. These programs then triagonalize the matrix, and solve for each parameter. This approach cannot be performed symbolically for large circuits because the computational burden of the matrix manipulations is too great. Another disadvantage of solving simultaneous equations is the resulting formula is *not in the same form* as an engineer would create with hand analysis. The purpose of the system is to explain cause and effect within the circuit to the engineer. For this reason the algorithm was created to return equations in the same form as the engineer would produce with hand analysis. The algorithm accomplishes this without setting up simultaneous equations.

How the System Works

The analysis system is based on the theory of Norton equivalent circuits. The theory states that any linear circuit as seen from the viewpoint of two terminals can be exactly modeled by a current source and a resistance in parallel.

For non-linear circuits the system still uses the equivalence as a linear approximation. A numerical system such as SPICE can later be used for refinement [3]. Simple models are therefore used for understanding. Complicated models are used to confirm the simplifying assumptions and for further refinement. The linear method is used by the system because the purpose is to return symbolic equations. Nonlinear equations would quickly become too complex for useful presentation to an engineer. The linear approach best satisfies the goal of delivering useful explanation to the engineer of how the circuit behaves. This coincides with the linear models used by engineers in "hand analysis" of circuits.

Note that the Norton equivalence is true no matter what load is placed at the terminal. This fact is used to isolate circuit building blocks from one another by modeling them as Norton circuits. Figure 2 shows four building block circuits connected to form a cascaded amplifier.

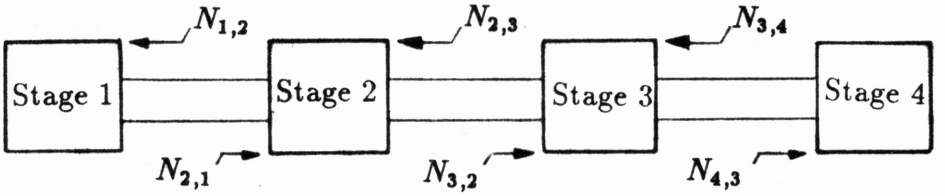


Figure 2. Cascaded Building Blocks

Let $N_{a,b}$ indicate the Norton Equivalent of circuit a as seen by circuit b. This Norton is a current $I_{a,b}$ in parallel with a resistance $R_{a,b}$. The program starts analysis by decoupling the circuits by finding each Norton equivalent. Starting at the input, $N_{1,2}$ is found. Using this value and the parameters for circuit two, $N_{2,3}$ is found. Next $N_{3,4}$ is found. Starting now at the last building block $N_{4,3}$ is found from the parameters of circuit four. Using these values and circuit three $N_{3,2}$ is found. Finally $N_{2,1}$ can be calculated using the parameters of circuit two and its load $N_{3,2}$. Now that the behavior of the circuit interconnections are known the circuit can now be viewed as independent building blocks as shown in Figure 3.

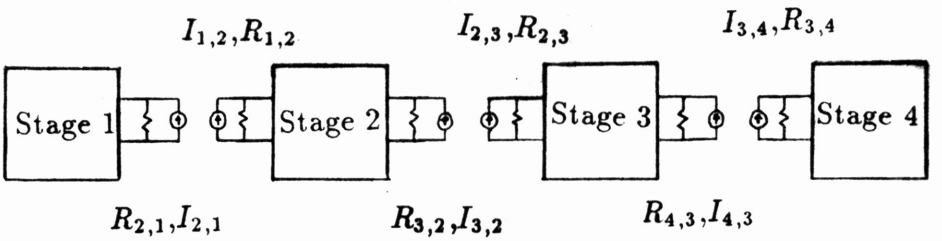


Figure 3. Decoupled Building Blocks

Questions of voltage or current values anywhere in the circuit can be directed to the appropriate building block where simple equations can solve the localized problem.

Example of Circuit Analysis

The simple circuit used to demonstrate the algorithm is shown in Figure 4. A non-trivial example with feedback is shown in Appendix A. Appendix B. shows the SPICE analysis of the same circuit.

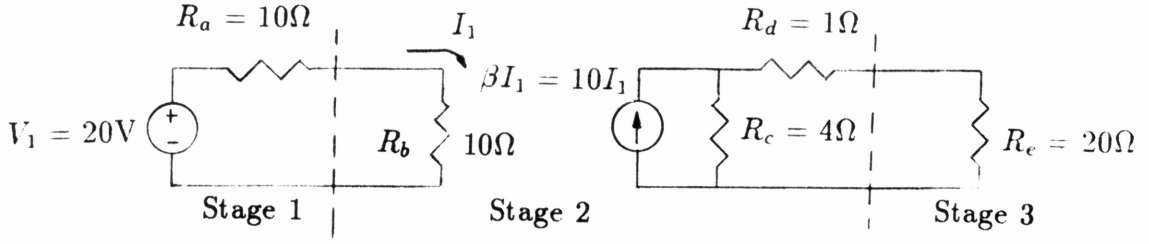


Figure 4. Simple Three Stage Circuit

The Algorithm begins by finding the Norton Output of stage one:

$$N_{1,2} = R_{1,2} \parallel I_{1,2} = R_a \Omega \parallel \left(\frac{V_1}{R_a}\right) \text{Amps}$$

Next the parameter $N_{1,2}$ is stored as a parameter of the second stage. The second stage can now calculate it's Norton Output:

$$N_{2,3} = (R_c + R_d)\,\Omega \parallel \left(\frac{\beta_1 R_{1,2} I_{1,2}}{R_{1,2} + R_b}\right)\left(\frac{R_c}{R_c + R_d}\right) \text{Amps}$$

Now, starting at the output, the Nortons are found looking in from the previous stage:

$$N_{3,2} = R_e \Omega \parallel 0\text{Amps}$$

$$N_{2,1} = R_b \Omega \parallel 0\text{Amps}$$

Once every Norton has been found, voltages can easily be calculated. For example the output voltage:

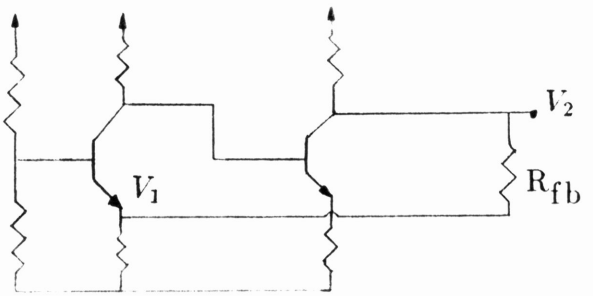
$$V_o = (I_{2,3} + I_{3,2})\left(\frac{R_{2,3} R_{3,2}}{R_{2,3} + R_{3,2}}\right)$$

$$V_o = \left(\left(\frac{\beta R_a (V_1/R_a)}{R_a + R_b}\right)\left(\frac{R_c}{R_c + R_d}\right) + 0\right)\left(\frac{(R_c + R_d)R_e}{(R_c + R_d) + R_e}\right) = 32\text{Volts}$$

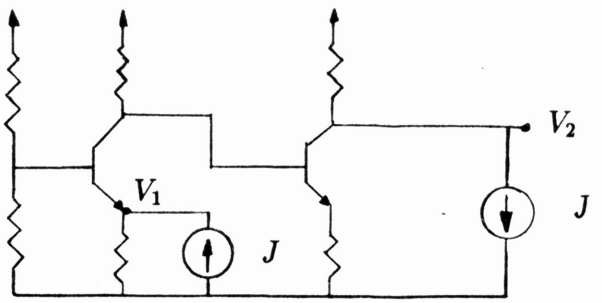
Note the algorithm returns an equation which is close to being simplified. When this procedure is performed on difficult amplifier circuits, the answer is similar to the form an engineer would create while doing hand calculations.

Resolution of Feedback Circuits

The following procedure uses the algorithm for cascaded building blocks to solve circuits with feedback loops. Consider the following simple circuit with a feedback resistor R.



1. Break the feedback resistor R_{fb} and replace it with two current sources with equal current. (The feedback resistance must be identified as the component which can be removed to form a cascaded structure.) This new circuit is identical when the current J in the new circuit is equal to the current through the feedback resistor in the first circuit.



2. Determine the voltages V_{1a}, V_{1b}, V_{2a}, and V_{2b} at the end of the current sources for two different current values J_a, J_b.

3. Calculate the true feedback current J_t as the solution of the three constraint equations:

$$m \cdot J_a + b = V_{2a} - V_{1a}$$

$$m \cdot J_b + b = V_{2b} - V_{1b}$$

$$m \cdot J_t + b = J_t$$

The solution to these equations is equivalent to finding the intersection of the two lines in Figure 5. The first two equations sample the linear relationship between any assumed current and the resultant voltages in the circuit. The third equation is ohms

law appearing as a line with a slope of R_{fb}. The intersection represents the point where the two circuits are identical.

4. Set the current source J equal to the true feedback current. The second non-feedback circuit is now identical to the original feedback circuit. For non-linear circuits, this method gives a linear approximation of the feedback current.

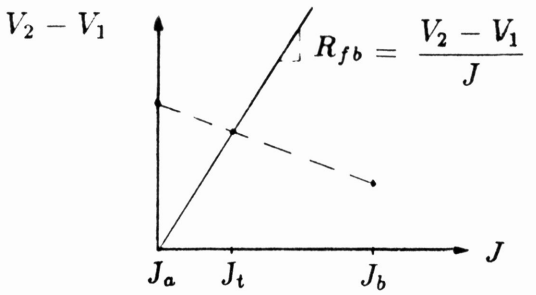


Figure 5. Calculation of True Feedback Current

By choosing $J_a = 0$ and $J_b = (V_{a2} - V_{a1})/R_{fb}$ and defining $J_c \equiv (V_{b2} - V_{b1})/R_{fb}$ the true feedback current can be simplified to:

$$J_t = \frac{J_b^2}{2 \cdot J_b - J_c}$$

IV. IMPLEMENTATION USING CLIPS

The author has implemented a knowledge based system for analog circuit analysis. The first prototype was written with the ART software package running on a Texas Instruments Explorer Computer [7]. The prototype returns approximate D.C. values for currents, voltages, output impedances, and input impedances. This prototype can analyze any number of cascaded amplifier stages. It can reason using Ohm's law, equivalent circuits, infinite values (for resistances), and default circuit values. The second prototype was implemented using CLIPS, an expert system development language which is essentially a subset of ART [8]. CLIPS is written in C and run on VAXes and on MS-DOS based personal computers. The CLIPS prototype is also able to report midband AC gains. The CLIPS prototype allows feedback structures in addition to simple cascaded building blocks. With the addition of software to display equations, this program will return symbolic equations for each value calculated by the system. This is important because the equations were calculated using nearly the same methods an engineer might use in "paper and pencil" symbolic calculations. The engineer can now have the computer take care of the tedious hand calculations required to understand a design. This prototype demonstrates how one of the tools for an intelligent design system can be realized.

The equations describing different building blocks are the task-specific knowledge stored in the brains and notes of human expert circuit designers. This expertise is stored in the expert system in the form of rules. To simplify the initial prototype the building blocks where restricted to three types:

(1) BJT amplifier stages: either Common emitter, Common Collector of Common Base.

(2) Norton Circuits: a current in parallel with a resistance.

(3) Thevinan Circuits: A voltage source in series with a resistance.

The BJT stages are all represented by one Generalized Amplifier stage shown in Figure 6. Figure 7 shows the calculated Norton outputs seen looking into the circuit of Figure 6. Each type of BJT stage can be represented by setting the input and output in the appropriate place.

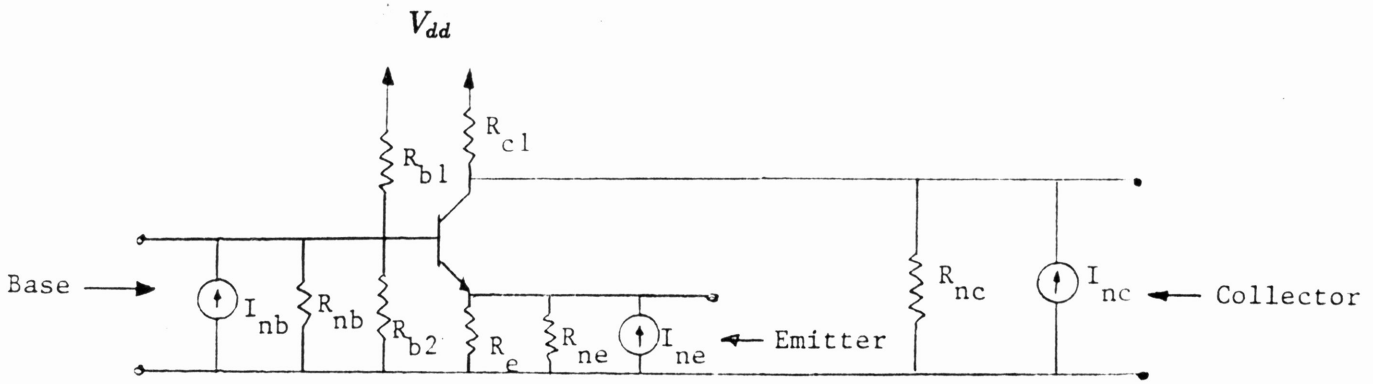
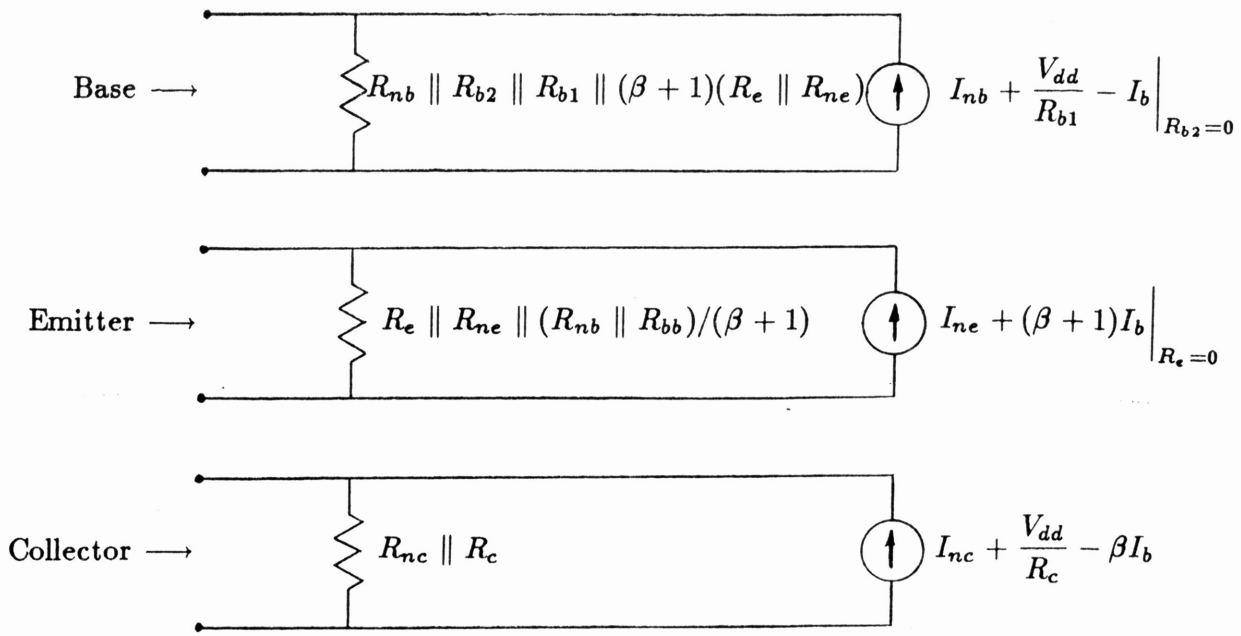


Figure 6. Generalized BJT Amplifier Circuit



Where I_b is calculated from this equation:

$$I_b \equiv \frac{(R_{nb2}(I_{nb} + V_{dd}/R_{b1}) - ((R_e \parallel R_{ne})I_{ne} + V_{be})(1 + R_{nb2}/R_{b1})}{R_{nb2} + (1 + R_{nb2}/R_{b1})(\beta+1)(R_{ne} \parallel R_e)}$$

$$A \parallel B \equiv (A \cdot B)/(A + B) \qquad R_{nb2} \equiv R_{nb} \parallel R_{b2}$$

Figure 7. Calculated Norton Outputs of Generalized Stage

Circuit Representation

A circuit is represented as a set of facts in the expert system database. Facts describing one object are grouped together and given the same value in their second field. This is similar to the groupings called schemata in the ART software package. For example the parameters for the stage ce-1 in Appendix C are represented by the following facts:

```
(deffacts ce-1
   (stages      ce-1 irb irc 2);stage 2 has input irb and output irc
   (instance-of ce-1 common-emitter)
   (parameters  ce-1 rt       rc      10000)
   (parameters  ce-1 dc-0     re      870)
   (parameters  ce-1 ac-0     re      0.000001)
                ;re is bypassed by a capacitor
   (parameters  ce-1 rt       rb1     100000)
   (parameters  ce-1 rt       rb2     17000)
   (parameters  ce-1 ac-0     bjt     100)     ;    AC-Beta
   (parameters  ce-1 dc-0     bjt     0.7 100) ;Vbe DC-Beta
   (voltages    ce-1 dc-0     n2      12)
   (feedback    ce-1 rt irb   r-f 2))
```

The parameter facts have five fields. The first field is the keyword "parameters." The second field gives the name of the stage or circuit building block "ce-1." All facts describing the same building block can be understood as comprising a data-structure. This data-structure is used much like Schemata structures built into the ART environment. The name of the datastructure "ce-1"is placed after the keyword "parameters" to increase the efficiency of pattern matching. The name of a datastructure is a variable within rule patterns while the keyword is a constant. Putting the constant first reduces the number of comparisons required to match a rule pattern with a fact. The third field tells the context

where the parameter is valid "rt, ac-0, dc-0." For instance ac-0 tells the program the parameter is valid for use in AC analysis. The -0 must be included because there can be more than one AC context. RT stands for root. RT tells the system the parameter is valid in any context. These "contexts" are similar to viewpoints used in the ART environment. The fourth field is the name of the component "rb2." Finally the fifth field gives the value of the component "17000." The sixth field is only used for components with more than one value associated with them. For instance, BJT transisors have a value representing the base to emitter voltage and a value for current gain.

Another type of fact which must be included in each building block begins with the keyword "stages." The second field of this fact again gives the name of the building block "ce-1." The third and fourth fields tell where the input and output from the building block are located "irb irc." In this case the input is located at the base and the output at the collector. The last field "2" tells that this building block is connected to stage 1 at the input and stage 3 at the output. This fact is used to relate the Norton outputs of this building block to the Norton loads appearing in adjacent building blocks.

The fact beginning with "instance-of" causes this building block to automatically receive a set of facts associated with common-emitters. These facts include all the connectivity information within the circuit. This feature (called inheritance) keeps the user from having to type in facts common to different building blocks. The inherited facts have default values. For instance if no fact defining the value of Rb1 is found, then Rb1 will be given the value infinity. (In this prototype infinity is just a very large number.)

Rules

The choice of the above circuit description was chosen to make rules easy to write. The rules in CLIPS or ART can be viewed as many IF-THEN statements running at the same time. The IF side of a rule is known as the left hand side (LHS). The THEN side of a rule is the right hand side (RHS). An expert system is made up of rules and facts. The rules LHSs have patterns which can match with facts in the database. If there is a fact matching each pattern in the LHS then the RHS is executed. The commands which can be executed in the RHS include ASSERT, and RETRACT. ASSERT adds a new fact

to the database. RETRACT removes facts from the database. For further explanation see the ART or CLIPS users manuals[7,8]. Why is the CLIPS program used instead of a string of IF-THEN statements in FORTRAN? Because the FORTRAN program would be too slow to run even a very small expert system. CLIPS and ART use the Rete Algorithm to solve the pattern match problem efficiently[9]. (Of course the RETE algorithm could also be programmed in FORTRAN.)

Adding New Building Blocks.

Each building block is supported by three rules in the system. Conceptually the rules have the simple form.

```
1.      IF    the input Norton load is known.
        THEN  calculate the Norton projected to the
                following stage, add this to the facts
                describing the following stage.
2.      IF    the output Norton load is known.
        THEN  calculate the Norton projected to the
                preceding stage, add this to the facts
                describing the preceding stage.
3.      IF    both the input and output Norton loads are known.
        THEN  calculate the Voltages at all nodes in the
                building block.
```

The designer can add to the building blocks by following this template and filling in the appropriate equations in the THEN side of the rules. This step of adding equations can itself be automated by another system written in CLIPS. A system has already been written to guide the user in formulating the rules, automatically adding the syntax details [10]. Another possible approach would be to use a SYN-like propagation of constraint algorithm to develop the equations automatically when a new structure is introduced by the designer.

Control of the Program

The program is divided into four phases. There is always one fact in the database beginning with the word "phase." Every rule has a pattern checking for this fact so that the rule will only fire in the proper phase of the program. In the first phase the input data is preprocessed. In this stage all the inheritance rules are used to create the facts expected in the later phases. Appendix D lists the code for the program phase 1. The second phase begins by solving for the feedback currents if there is a feedback loop. This is done first because every other calculation is dependent on the feedback current. The rest of the second phase holds the rules used to create the menu system. When the user chooses an option from the menu, the rules in the second phase set up a goal and then start the third phase. The third phase performs the Norton equivalent circuit analysis. This phase includes rules holding the transfer function equations for the different building blocks supported in this prototype. Phases four holds all the rules which calculate Voltages and Currents. This phase follows phase three because the Norton equivalent loads on each stage are required to calculate currents and voltages.

V. LESSONS FROM THE PROTOTYPE

Problems with CLIPS

The greatest limitation to CLIPS is there is no direct support for backward chaining rules. Backward chaining is reasoning from a desired conclusion toward premises. The CLIPS version performs all inheritance in the first program phase even for values which will never be needed. The more elegant solution is to have a backward chaining rule which causes a value to be inherited only when it is needed. The absence of support for backward chaining caused the CLIPS version of the prototype to be less expandable than the ART version. It is difficult to build a very large useful system without backward chaining because goal directed control becomes the only way to keep the program from performing unnecessary calculations. The CLIPS prototype uses partial goal direction by stopping calculations when the value sought is found.

The great limitation of CLIPS which is most likely to motivate the purchase of ART is CLIPS does not have a well developed user interface. For example, ART can display the user a justification network which shows how each fact in the database was created. ART can also report which patterns in a rule are being matched and which are not. This debugging tool is missing in CLIPS. This causes the programmer to waste more time trying to determine why a rule is not firing. The problems with debugging a program in CLIPS were minimized because the first prototype was developed with ART, so most of the problems were worked out before the program was run on CLIPS.

Directions for Further Development

Part II of this paper described the design environment targeted for this expert system CAD tool. Several changes to the prototype must be implemented before it can be used within a larger system. The purpose of the prototype is to quickly produce a working system which proves the concepts to be used in the final product. For this reason, compromises are made in the implementation of the prototype. For example, a prototype usually has an incomplete user interface. These compromises must be well understood and documented to prevent them from becoming part of the final delivered system.

One problem with the current prototype is the building block equations are stored as rules rather than facts. This will make it more difficult to add equations for new building blocks dynamically. All the rules with equations should be replaced with a goal directed control algorithm:

IF there is a goal to find an unknown parameter P_i in the database

THEN (IF there is an equation in the database

$P_i = f(P_1, P_2, P_3, ...P_n)$

THEN (IF $(P_1, P_2, P_3, ...P_n)$ are known

THEN Evaluate f and assert result as the value of P_i

Remove the goal to find P_i

ELSE Assert a goal to find each unknown parameter

in P_1 - P_n

ELSE Ask the user for the value of P_i)

ELSE Ask the user for another goal

The primary difficulty preventing implementation of this algorithm is the number of parameters n is not constant. The author has not determined an acceptable method in ART or CLIPS to have one rule bind a variable number of patterns. One solution would to simply write a seperate rule for each possible number of parameters in an equation.

Three approaches for solving problems causes by the limitations of the expert system building tool are:

(1) Add the required features to ART or CLIPS in a level lower language, such as in LISP for ART or in C for CLIPS. This is risky because the additions are not likely to be compatible with future versions of the development systems. Furthermore, the uncompiled ART code in not available.

(2) Write a "wrap-around" Program which preprocesses the data, runs ART or CLIPS, and then analyzes it before reporting the results to the user. For example, a program

written in C could take the equations describing building blocks and automatically create all the rules which use these equations before running CLIPS. One problem with this approach is it might slow down the system.

(3) The most difficult solution is to rewrite the expert system in a lower language. This can result in the most efficient code because features of ART or CLIPS which are not used do not have to be rewritten.

Another change needed in the prototype is to allow symbolic as well as numeric calculation of equations. One way to do this is to replace all the mathematical operators with functions which send the equations to a rational simplification algorithm. The program MACSYMA could be called to perform this simplification. The problem of CLIPS not allowing nested parentheses can be overcome by replacing parentheses by unique letter combinations such as LP and RP instead of "(" and ")". An interface function between the CLIPS operator and MACSYMA would be written in C to add parentheses for MACSYMA and replace them with LP and RP for CLIPS. One difficulty in using an external program such as Macsyma is that some control over the form of the equations might be lost. The Norton equivalent circuit algorithm returns equations in a form similar to the forms created with hand analysis. A simplification algorithm is likely to destroy this form.

Another important addition needed in the prototype is to allow complex number calculations. This change will require each operator to work with imaginary parts for each number. This change should not effect the structure of the expert system.

The prototype currently allows no more than one feedback loop. This can be extended to allow any number of feedback loops. For this to be done, the implementation of contexts or viewpoints must be extended. It must be possible to sprout new viewpoints whenever an unknown feedback current is discovered.

The final recommended change to the program is to change the input structure. The present structure was chosen to make pattern matching in the rules as easy as possible. In hindsight it appears that the pattern matching could have been accomplished with a simpler circuit description. For this program to be useful it must be able to interface with circuit descriptions more common to CAD systems. One solution to this problem is to use

an extension to the circuit description used by SPICE. A SPICE file puts one element on an input line rather than one node. This SPICE information will have to be supplemented with facts used to partition the circuit into building blocks. A compromise solution would be to write a separate program which takes a SPICE circuit description of building blocks and creates the description used in the prototype.

VI. CONCLUSION

A tool for equation development in an interactive design environment has been described. The system successfully meets the following three constraints for a useful interactive symbolic analysis system. First, it is quicker than an engineer at generating the equations. Like the engineer it is capable of handling large circuits without an exponential growth in calculation time. Secondly, it produces equations simplified in the same form as the design engineer would create by hand. Finally, the entire system is easily modified and expanded by the design engineer personally. The equation developing system is based upon the Norton equivalent circuit approximation.

A prototype system was implemented using a rule based development program called CLIPS. The prototype returns correct AC and DC values for voltages, currents, output impedances, and input impedances. This prototype was demonstrated using BJT amplifier stage building blocks. The rule based system was used because it made extension to the building blocks simple. This implementation in CLIPS successfully demonstrated the algorithm for symbolic equation generation.

REFERENCES

[1] M.A. Styblinski, Interviews concerning the requirements of an intelligent design system, May 1986 - Feb. 1987.

[2] L.W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits", Electronics Research Laboratory, University of California, Berkeley, Memorandum N0. ERL-M520, May 9,1975.

[3] M.A. Styblinski, L.J. Opalski, "INTOPT.SPICE.0", pre-release version, Department of Electrical Engineering, Texas A&M, July 1985.

[4] R. Bogen, et. al., "MACSYMA reference manual", MIT Project MAC, September 1974.

[5] J.D. De Kleer, G.J. Sussman, "Propagation of Constraints Applied to Circuit Synthesis", Circuit Theory and Applications, Vol. 8, pp.127-144, 1980.

[6] Ibid.

[7] ART Reference Manual, Inference Corporation, 1985.

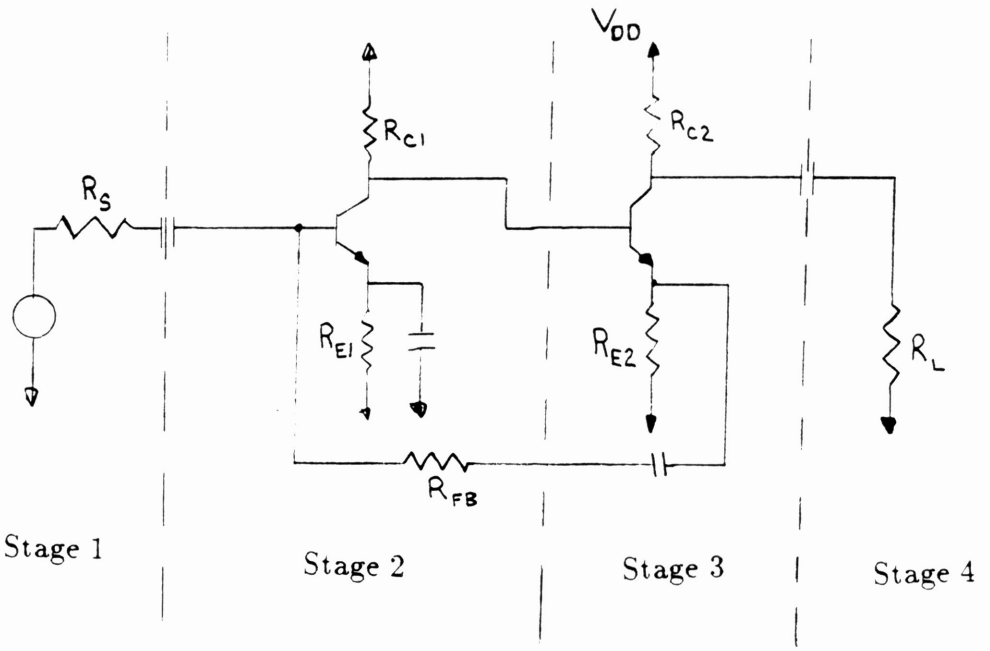
[8] CLIPS Reference Manual Version 3.0, NASA: Mission Planning and Analysis Division's Artificial Intelligence Section, July 1986

[9] Charles L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Artificial Intelligence Vol. 19, pp.17-37, 1982.

[10] Wayne Hasty, "Knowledge Acquisition System for CLIPS (KASC), A Help System to Create CLIPS Rules", Masters of Electrical Engineering Project, Dept. of Electrical Engineering, April 1987.

Appendix A. Analysis of a Feedback Circuit

The circuit used to demonstrate the analysis is a two stage amplifier with feedback of the type shunt-series.[3] In shunt-series feedback, current from the signal at the output is converted to a voltage which is subtracted from the input voltage.



This circuit was analyzed with V_s = 1V, R_s = 10KOhms, R_{b1} = 100KOhms, R_{b2} = 17KOhms, R_{c1} = 10KOhms, R_{e1} = 870Ohms, R_{c2} = 8KOhms, R_{e2} = 1.3KOhms, R_L = 1KOhm, and R_f = 10KOhms.

---

[3] Taken from "Microelectronic Circuits" by Sedra and Smith, page 558

```
        CLIPS (V3.2a 01/09/87)
CLIPS> (load "analyse.")
CLIPS> (load "Cedra-ckt.")
CLIPS> (reset)
CLIPS> (run)
         Starting fb-inheritance
         Starting dc-1 test
         viewpoint- dc-1 v1 v2 are 1.03022242 1.59551072
         viewpoint- dc-2 v1 v2 are 1.03022242 1.59551072
         Starting ac-1 test
         viewpoint- ac-1 v1 v2 are -64.26095581 0.17346773
         viewpoint- ac-2 v1 v2 are 4077.12182617 -11.00382614

     Cascaded Amplifier Circuit Analysis   Root Menu
     ----------------------------------   ---------

            1- Exit to Clips>
            2- D.C. analyze circuit stage
            3- A.C. analyze circuit stage

            2

       Please Choose the stage Number to Analyze
       -----------------------------------------

           1- stage:  source-1
           2- stage:  ce-1
           3- stage:  ce-2
           4- stage:  load-1

           3

      dc Circuit Analysis of stage ce-2
      ---------------------------------

            1- Return to Root Menu
            2- Show Input  Norton
            3- Show Output Norton
            4- Show Currents
            5- Show Voltages
            6- Show Component Values
            7- Show Node Connections
            8- Analyze different stage

            2
The dc Norton Equivalent seen at the input of stage ce-2 is
        0.0053313mAmps  and  1.3130e+05ohms.
```

```
        dc Circuit Analysis of stage ce-2
        ---------------------------------

              1- Return to Root Menu
              2- Show Input  Norton
              3- Show Output Norton
              4- Show Currents
              5- Show Voltages
              6- Show Component Values
              7- Show Node Connections
              8- Analyze different stage

              3

The dc Norton Equivalent seen at the output of stage ce-2 is
        0.71536756mAmps  and  8000ohms.


        dc Circuit Analysis of stage ce-2
        ---------------------------------

              1- Return to Root Menu
              2- Show Input  Norton
              3- Show Output Norton
              4- Show Currents
              5- Show Voltages
              6- Show Component Values
              7- Show Node Connections
              8- Analyze different stage

              4

dc Currents in Circuit stage ce-2

        The dc Current into Terminal rb1 2 is -1.0270e-15 mAmps
        The dc Current into Terminal rb1 1 is 1.0270e-15 mAmps
        The dc Current into Terminal rb2 2 is -1.7302e-16 mAmps
        The dc Current into Terminal rb2 1 is 1.7302e-16 mAmps
        The dc Current into Terminal re 2 is -0.79247880 mAmps
        The dc Current into Terminal re 1 is 0.79247880 mAmps
        The dc Current into Terminal rc 2 is -0.78463244 mAmps
        The dc Current into Terminal rc 1 is 0.78463244 mAmps
        The dc Current into Terminal bjt 3 is -0.79247880 mAmps
        The dc Current into Terminal bjt 1 is 0.78463244 mAmps
        The dc Current into Terminal bjt 2 is 0.00784632 mAmps
        The dc Current into Terminal r-f 1 is -5.6529e-17 mAmps
```

```
        dc Circuit Analysis of stage ce-2
        ---------------------------------

              1- Return to Root Menu
              2- Show Input  Norton
              3- Show Output Norton
              4- Show Currents
              5- Show Voltages
              6- Show Component Values
              7- Show Node Connections
              8- Analyze different stage

              5

dc Voltages in Circuit stage ce-2

           The dc Voltage at Node n2 is 12 Volts.
           The dc Voltage at Node n0 is 0 Volts.
           The dc Voltage at Node n1 is 1.73022223 Volts.
           The dc Voltage at Node n4 is 1.03022242 Volts.
           The dc Voltage at Node n3 is 5.72294044 Volts.

        dc Circuit Analysis of stage ce-2
        ---------------------------------

              1- Return to Root Menu
              2- Show Input  Norton
              3- Show Output Norton
              4- Show Currents
              5- Show Voltages
              6- Show Component Values
              7- Show Node Connections
              8- Analyze different stage

              6

       Circuit stage ce-2 has the following parameter values

       Element bjt has value 0.7 100
       Element r-f has value 1.0000e+19
       Element rc has value 8000
       Element re has value 1300
       Element rb2 has value 1.0000e+19
       Element rb1 has value 1.0000e+19
       Element ire has value 5.6529e-20 1.0000e+19
       Element irc has value 0 1.0000e+19
       Element irb has value 0.00018087 10000
```

```
dc Circuit Analysis of stage ce-2
---------------------------------

      1- Return to Root Menu
      2- Show Input  Norton
      3- Show Output Norton
      4- Show Currents
      5- Show Voltages
      6- Show Component Values
      7- Show Node Connections
      8- Analyze different stage

      7

Circuit stage ce-2 has the following connections

 Terminal rb2 2 is connected at node n0
 Terminal re 2 is connected at node n0
 Terminal ce 2 is connected at node n0
 Terminal rb2 1 is connected at node n1
 Terminal rb1 2 is connected at node n1
 Terminal bjt 2 is connected at node n1
 Terminal rb1 1 is connected at node n2
 Terminal rc 1 is connected at node n2
 Terminal rc 2 is connected at node n3
 Terminal bjt 1 is connected at node n3
 Terminal re 1 is connected at node n4
 Terminal ce 1 is connected at node n4
 Terminal bjt 3 is connected at node n4

dc Circuit Analysis of stage ce-2
---------------------------------

      1- Return to Root Menu
      2- Show Input  Norton
      3- Show Output Norton
      4- Show Currents
      5- Show Voltages
      6- Show Component Values
      7- Show Node Connections
      8- Analyze different stage

      1
```

```
Cascaded Amplifier Circuit Analysis   Root Menu
----------------------------------   ---------

        1- Exit to Clips>
        2- D.C. analyze circuit stage
        3- A.C. analyze circuit stage

        3

  Please Choose the stage Number to Analyze
  -----------------------------------------

      1- stage:  source-1
      2- stage:  ce-1
      3- stage:  ce-2
      4- stage:  load-1

      3
 ac Circuit Analysis of stage ce-2
 ---------------------------------

        1- Return to Root Menu
        2- Show Input  Norton
        3- Show Output Norton
        4- Show Currents
        5- Show Voltages
        6- Show Component Values
        7- Show Node Connections
        8- Analyze different stage

        5

ac Voltages in Circuit stage ce-2

     The ac Voltage at Node n3 is 0.75136518 Volts.
     The ac Voltage at Node n4 is -0.98187023 Volts.
     The ac Voltage at Node n1 is -1.00880289 Volts.
     The ac Voltage at Node n2 is 0 Volts.
     The ac Voltage at Node n0 is 0 Volts.
```

APPENDIX B

SPICE ANALYSIS OF THE FEEDBACK CIRCUIT

DC Analysis using a Linear Model

```
.SUBCKT DCBJT 1 2 3
* 1 2 3 IS COLLECTOR BASE EMITTER
VBE    2    3    DC    0.7
F1     1    3    VBE   100
.ENDS

*TWO STAGE CIRCUIT
VDD    2    0    DC    12
VS     5    0    AC    1
RS     5    6    10K
CS     6    1    100UF
RB11   2    1    100K
RB21   1    0    17K
RE1    4    0    870
CE1    4    0    100UF
RC1    2    3    10K
X1DC   3    1    4    DCBJT
RE2    14   0    1.3K
RC2    2    13   8K
X2DC   13   3    14   DCBJT
CL     13   7    100UF
RL     7    0    1K
RF     14   8    10K
CF     8    1    100UF
.END
```

Results

```
N1   = 1.5955 Volts,  N2 = 12.0000 Volts,  N3   = 1.7302 Volts
N4   = 0.8955 Volts,  N8 = 1.0302  Volts,  N13 = 5.7229 Volts
N14 = 1.0302 Volts
```

SPICE ANALYSIS OF THE FEEDBACK CIRCUIT

AC Analysis

Node	Function
1	base of ce-1
2	vdd
3	collector of ce-1 and base of ce-2
4	emitter of ce-1
13	collector of ce-2
14	emitter of ce-2

```
.SUBCKT ACBJT1 1 2 3
* 1 2 3 IS COLLECTOR BASE EMITTER
VB    2   4   AC   0.0
RPI   4   3   3679.6035
F1    1   3   VB   100
.ENDS

.SUBCKT ACBJT2 1 2 3
* 1 2 3 IS COLLECTOR BASE EMITTER
VB    2   4   AC   0.0
RPI   4   3   4779.30762
F1    1   3   VB   100
.ENDS

*TWO STAGE CIRCUIT
VDD   2   0   DC   12
VS    5   0   AC   1
RS    5   6   10K
CS    6   1   100UF
RB11  2   1   100K
RB21  1   0   17K
RE1   4   0   870
CE1   4   0   100UF
RC1   2   3   10K
X1AC  3   1   4   ACBJT1
RE2   14  0   1.3K
RC2   2   13  8K
X2AC  13  3   14  ACBJT2
CL    13  7   100UF
RL    7   0   1K
RF    14  8   10K
CF    8   1   100UF
```

Results

```
N1  = 4.057E-03  Volts, N3 = 1.018E+00 Volts, N13 = 7.486E-01 Volts
N14 = 9.781E-01  Volts
```

Representation of a Feedback Circuit

```
; circuit EXAMPLEA.CLP
; This is a description of a specific common emitter circuit
; This Circuit is from page 558, Sedra and Smith


(deffacts source-1
   (stages      source-1 nil ir2 1)  ;first stage has output at ir2
   (instance-of source-1 nort-eq)
   (parameters  source-1 ac-0  i-thev     0.0001)
   (parameters  source-1 ac-0  rs         10000)
   (feedback    source-1 rt ir2    r-f 2))

(deffacts ce-1
   (stages      ce-1 irb irc 2)  ;second stage has input irb and output irc
   (instance-of ce-1 common-emitter)
   (parameters  ce-1 rt       rc         10000)
   (parameters  ce-1 dc-0     re         870)
   (parameters  ce-1 ac-0     re         0.000001) ;bypassed by capacitor
   (parameters  ce-1 rt       rb1        100000)
   (parameters  ce-1 rt       rb2         17000)
   (parameters  ce-1 ac-0     bjt         100)
   (parameters  ce-1 dc-0     bjt        0.7 100)
   (voltages    ce-1 dc-0     n2         12)
   (feedback    ce-1 rt irb   r-f 2))

(deffacts ce-2
   (stages      ce-2 irb irc 3)
   (instance-of ce-2 common-emitter)
   (parameters  ce-2 rt       rc         8000)
   (parameters  ce-2 rt       re         1300)
   (parameters  ce-2 ac-0     bjt         100)
   (parameters  ce-2 dc-0     bjt        0.7 100)
   (voltages    ce-2 dc-0     n2         12)
   (feedback    ce-2 rt       ire        r-f 1)
   (parameters  ce-2 dc-0     r-f        1e19) ;cap in series
   (parameters  ce-2 ac-0     r-f        10000))

(deffacts load-1
   (stages      load-1  ir2 nil 4)
   (instance-of load-1  nort-eq)
   (feedback    load-1  rt ir2   r-f 1) ;also connected at ce-2 ire
   (parameters  load-1  dc-0  i-thev      0)
   (parameters  load-1  dc-0  rs          1e19) ;cap in series
   (parameters  load-1  ac-0  rs          1000))
```

APPENDIX D

PROGRAM LISTING FOR PHASE-1

```
;;;
;;; phase-1.clp
;;; These rules prepare incoming data for the other rules
;;;    phase-1       creates inherited facts, removes wrong default values
;;;    phase-2       asks user for a goal
;;;    phase-3       calculates value requested in goals
;;;    phase-4       reports the value and returns to phase 3
;;;

(defrule start-phase-1
   ?x <- (initial-fact)
  =>
   (retract ?x)
   (assert (phase 1)))

(defrule inherit-1 "an object with 'is-a' inherits 'is-a' "
   (phase 1)
   (is-a          ?new-object ?old-object)
   (is-a          ?old-object ?older-object)
  =>
   (assert (is-a ?new-object ?older-object)))

(defrule inherit-2 "instances inherit everything except 'is-a' slot-types"
   (phase 1)
   (instance-of   ?new-object ?old-object)
   (?slot-type&:(! (eq ?slot-type is-a))
                       ?old-object ?slot $?value)
  =>
   (assert (?slot-type ?new-object ?slot $?value)))

(defrule inherit-3 "instances get 'instance-of' instead of 'is-a' "
   (phase 1)
   (instance-of   ?new-object ?old-object)
   (is-a          ?old-object ?older-object)
  =>
   (assert (instance-of ?new-object ?older-object)))

(defrule create-dc-ac-vp-parameters "ac&dc inherit from rt (root) vp"
   (declare (salience -20))
   (phase 1)
   (instance-of   ?object ?old-object)  ;only for instance
   (?p ?object rt $?x)
   (test (|| (eq ?p parameters) (eq ?p currents) (eq ?p voltages)
             (eq ?p nort-out) (eq ?p remove-defaults) (eq ?p defaults)
             (eq ?p feedback)))
  =>
   (assert (?p ?object ac-0 $?x))
   (assert (?p ?object dc-0 $?x)))
```

```
(defrule no-more-inheritance "keeps removed defaults from being reinherited"
   (declare (salience -200))
   ?x1 <- (phase 1)
  =>
   (retract ?x1)
   (assert (phase 1 b)))

(defrule remove-defaults-explicit "input file says remove-defaults so do it"
   (declare (salience -30))
   (phase 1 b)
   (instance-of ?new-object ?old-object) ;do only for instances
   ?x1 <- (defaults ?new-object ?vp ?component $?values)
   ?x2 <- (remove-defaults ?new-object ?vp ?component)
  =>
   (retract ?x1 ?x2))

(defrule remove-defaults-implied "parameter already has a value so"
   (declare (salience -30)) ;remove all defaults before use-defaults
   (phase 1 b)
   (instance-of ?new-object ?old-object) ;do only for instances
   ?x1 <- (defaults ?new-object ?vp ?component $?values)
   (parameters  ?new-object ?vp ?component $?x)
  =>
   (retract ?x1))

(defrule use-defaults "if value is unknown then assign it the default"
   (declare (salience -40))
   (phase 1 b)
   ?x <- (defaults ?new-object ?vp ?component $?values)
   (instance-of ?new-object ?old-object) ;do only for instances
  =>
   (retract ?x)
   (assert (parameters ?new-object ?vp ?component $?values)))

(defrule start-phase-2   "phase 1 is over when the agenda is empty"
   (declare (salience -10000))
   ?x <- (phase 1 ?bc)
  =>
   (retract ?x)
   (assert (phase 2)))
```

```
(defrule start-fb-inheritance
   (declare (salience -1000))
   ?x1 <- (phase 1 b)
   (feedback $?)  ;if there is feedback
   =>
   (retract ?x1)
   (printout crlf "         Starting fb-inheritance" crlf)
   (assert (phase 1 c)))

(defrule fb-create-ac-parameters
   (phase 1 c)
   (instance-of   ?object ?old-object)  ;only for instance
   (?p ?object ac-0 $?x)
   (test (|| (eq ?p parameters) (eq ?p currents) (eq ?p voltages)
             (eq ?p nort-out) (eq ?p feedback)))
  =>
   (assert (?p ?object ac-1 $?x))
   (assert (?p ?object ac-2 $?x)))

(defrule fb-create-dc-parameters
   (phase 1 c)
   (instance-of   ?object ?old-object)  ;only for instance
   (?p ?object dc-0 $?x)
   (test (|| (eq ?p parameters) (eq ?p currents) (eq ?p voltages)
             (eq ?p nort-out) (eq ?p feedback)))
  =>
   (assert (?p ?object dc-1 $?x))
   (assert (?p ?object dc-2 $?x)))
```