USTAAD - AN INTERACTIVE PEDAGOGICAL AID FOR ELEMENTARY ELECTRONIC CIRCUIT ANALYSIS

by

Tariq Saaad & Stephen W. Director

DRC-18-4J-82

April. 1902

USTAAD
AN INTERACTIVE PEDAGOGICAL AID FOR ELEMENTARY ELECTRONIC CIRCUIT ANALYSIS

T. SAMAD AND S. W DIRECTOR
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PA

Abstract

This paper describes Ustaad, a GAL program which provides self-paced test and drill facilities in elementary electronics circuit analysis. Circuits, either synthesized or user-specified, are displayed on the user's terminal The user can be quizzed on electrical properties of these circuits* and/or he can interactively analyze them. A sophisticated calculator enables independence of paper-and-pencil calculations, and an adequate editing facility allows interactive circuit design. The integration of Ustaad's varied capabilities and the environmental features of Ustaad are emphasized.

1. INTRODUCTION

During the 1980 International Symposium on Circuits and Systems* the second author participated in a panel discussion on "Educational Trends in Circuit and System Theory**[1]. One of the most interesting suggestions to come out of these discussions was made by Mac Van Valkenburg. He observed the following well known trends:

1. Electrical Engineering undergraduate class sizes are increasing.
2. The size of most Electrical Engineering Department faculties is decreasing.
3. The number of Electrical Engineering graduate students, especially those used as tfashing assistants* is rtfrrf sirg.

These trends have bad a number of adverse effects on the quality of undergraduate education. In particular, one of the most important educational activities, the drilling of students via homework assignments, which require time and effort to generate and grade, is diminishing. Mac suggested that the computer, which is often used as an analysis tool in introductory circuits courses, might be used as a means for drilling students. This paper describes our first attempt at developing a program for such a purpose.

We begin by acknowledging that a great deal of research has been carried out in the field of Computer-Aided Instruction. CAI is concerned with the construction of computer programs capable of *teaching* students, *i.e.* CAI programs are intended to some degree to be a replacement for the human instructor. While this intention may be laudable, present technology, and the present level of understanding of the psychology of learning, falls well short of the sophistication necessary for a college-level CAI program. Natural language understanding, hypotheses generation and testing, and inference and deduction techniques are some of the problem areas that come readily to mind. However, we would like to briefly mention SOPHIE, a CAI system which is indeed impressive in some aspects of its performance (see [2], [3], the first reference lists a terminal session indicative of SOPHIE'S prowess). It is our contention that, in the didactic field, computers and computer programming skills of today are best utilised in tools to aid the */earning* process. There is some recent interest in Computer-Aided Learning, or CAL *(e.g.* [9], [7]), though the only system that seems to have found even limited acceptance is the PLATO system ([1]), the history of which goes back about twenty years. To the author's knowledge, none of the CAL systems have tackled the problem of integrating the analysis, design and tutorial aspects of pedagogy, and given the issue of a comfortable user environment the importance that is its due.

[1]This puiel was chaired by Sholmo Kami. Other members included G.D.Hacbtel. E.S.Kufa. G.S.Moochytz, W.Rhines. G.C.Tomes, and M.E.Van Valkeaburg.

Both PLATO and SOPHIE have addressed important issues in CAI, and their capabilities are indeed impressive. The goals they set for themselves have to a large extent been achieved. In the context of a teaching aid meant especially for extensive drilling, however, they have serious limitations which preclude their being used in the manner and for the purpose we have in mind. SOPHIE'S disadvantage from this point of view is a direct result of its sophistication. The emphasis that was placed on hypothesis generation and testing resulted in parts of the code (the functional simulator) being very circuit-specific* According to [5], the required hand-inputting of the circuit dependent details (for the one circuit SOPHIE understands) required six man-months. We envisage a student sitting down at a terminal for a drill and being drilled on as many different circuits in one session as he has time for and as he feels he needs. This not only rules out using just one circuit for the drilling, it virtually mandates generating circuits in some kind of random fashion. PLATO, on the other hand, is more of a specialised hardware configuration which, though uaeful in a pedagogical context, may be of a utility limited enough otherwise to deter most institutions from the requisite expenditure.

With the above in mind we have developed a program, called Ustaad[2], which is to be used to drill students on the types of exercises normally associated with the first undergraduate course on circuit analysis. We emphasize that Ustaad is meant to be used in conjunction with conventional classroom work, and is not meant to supercede any aspect of the instructor's class presentation. Thus, no claims about Ustaad*s "intelligence" are being made, but a great deal of effort has been expended to ensure that the user will not (easily) get bored or exacerbated with it

In its current implementation, Ustaad's domain is dc electrical circuit analysis. Its main aim is to provide an environment wherein the student can test his knowledge of the subject at a self-determined level and pace. Ustaad generates a circuit, draws it on the student's graphics terminal, and tests the student on the electrical properties of the circuit, grading his answers along the way. The student can also ask his own questions, of the type Ustaad can ask of him *(i.e* Ustaad is a "mixed initiative" system). Furthermore, the student can use Ustaad as an interactive circuit analysis program, specifying his own circuit (though conventional analysis programs cannot be asked directly to determine "the Norton equivalent current source as seen by branch x").

Two aspects of Ustaad which we believe to be most significant are 1) the integration of its features, and 2) the "comfortableness" of its user environment (Of course, the first factor is to an extent incorporated in the second, but it has other relevance which requires separate treatment). The integration ensures consistency between modules and enables "automatic" sharing of relevant

[2]*UStSBd* is the *Urdu* for "venerated teacher", though in deference to the tradition of acronymous program names, it also stands for l/nusual Sophomoric teaching did for Analysis and design.

information. Fig. 1 shows how stored information is affected by each module, and what global information each module requires. The rectangular boxes are global variables, and the circles represent modules. Arrows from a variable into a module imply that the particular module accesses that variable, and arrows the other way imply writing/updating of the variable by the module. Thus, for example, OCTCEN generates a new circuit which depends on the current difficulty level. Note that the modules are indifferent to the origin of the circuit they access. Thus circuits generated by CKTGEN or manually input (using the EDITOR) are identically treated by all accessing modules.

The importance of a "comfortable" user environment for interactive programs can scarcely be overemphasized, especially for a pedagogical tool. Since we expect Ustaad to be used at a student's own discretion, overcoming his age-old (and now somewhat anachronistic) dependence on paper, pencil, and textbook is largely a matter of ensuring that he is at ease while interacting with the program. Faced with an uncomfortable environment (and this can be reflected in myriad ways) the student is unlikely to be motivated to pursue his scholarship at the "hands" of what would all too obviously be a "dumb** machine. (The loss, of course, would be his, but the altruism of the pedagogical profession demands consideration of such matters).

In the next section we give an "outline** of Ustaad. Subsequent sections discuss individual options available in Ustaad.

2. THE STRUCTURE OF USTAAD

Fig. 2 shows the modules and the control paths which make up Ustaad. At the highest level, the user has the following options:

- **He can cause the generation of a circuit in a quasi-random manner by invoking the CKTGEN module.**
- **He can invoke the EDIT module to enter his own circuit or to edit a stored one.**
- **He can invoke the TESTER to be quizzed on the circuit last generated or specified.**
- **He can employ the QUERY module to perform an interactive analysis of the circuit**
- **He can invoke the CALCULATOR.**
- **He can display the circuit graphically if he is on a graphics terminal, or by listing the branches otherwise, using the DISP module.**
- **He can control the level of difficulty through PROFILE.**
- **He can ask for help.**
- **He can exit Ustaad.**

3. CIRCUIT GENERATION

The circuits which are used by Ustaad to drill the student are synthesized by a three step procedure. These steps are topology selection, element-type assignment, and value assignment, as described below.

1. Topology Selection: The first step in synthesis is to randomly choose a network graph from a predefined set of allowable graphs. A random number generator is used to make this selection. The topologies allowed have been chosen so as to result in a straightforward graphical display. Information about how a topology is to be drawn on the terminal screen is also stored with each topology.

2. Element-Type Assignment: After a topology has been chosen, the random number generator is used to assign particular circuit elements to the branches of the graph according to some predefined probabilities of occurrence. All the element-types— resistors, voltage sources, current sources, and the four kinds of controlled sources- have realistic probabilities of selection associated with them. Different difficulty levels are possible by choosing different probabilities of occurrence.

   Observe that element-type assignment must satisfy the constraints imposed by circuit theory; specifically, we prohibit loops of (controlled and independent) voltage sources, and cutsets of (controlled and independent) current sources.

3. Value Assignment: The last step in the circuit synthesis is the assignment of values to circuit elements. These values are chosen so as to

   a. Avoid a range of values for an element-type which would be unrealistic

   b. Avoid unrealistic values for output quantities, (viz. we do not want voltages of the order of kilo-volts, nor currents in the tens of amperes.)

   c Assign values which are industry standards.

   To achieve satisfactory values we use the random number generator to choose the inantisgy and the exponent of each element value separately. The mantissa is chosen from separate tables of allowable values which are associated with each element-type. The exponent is randomly chosen from an acceptable range of values for each element-type.

4. QUESTIONS AND ANSWERS

The interactive questioning and answering facility is the primary feature of Ustaad and most of the student's time will be spent answering questions put to him, or asking questions himself. It is essential that the syntax and protocol of the student-Ustaad interface be comfortable to the student. Monotonously similar questions, "dry" answers, intolerance of both trifling errors and disastrous misinterpretations, all will undoubtedly force the student back to his paper, pencil, textbook, and calculator. In this section we describe how questions are generated and interpreted.

At present the questions Ustaad can ask or answer have been divided into three "question-types":

1. "simple", in which the quantity asked for is a voltage at a node or across a branch, the current through a branch, or the power dissipated in a branch.

2. "equivalent", in which the quantity asked for is the Thevenin or Norton equivalents between two nodes, or as seen by a branch.

3. "superposition", in which the quantity asked for is the voltage across, current through, or power dissipated in a branch due to some one independent source. Questions of this type are only asked if more than one source is present.

The types of questions permitted is controlled by the student using the *profile* command.

Question Generation: Questions are generated by first selecting (at random) a question-type from those currently allowed, and then selecting (at random) the parameter or parameters for the question (*e.g.* a node number for a node voltage question, or two branch numbers for a superposition question). The question thus generated is then checked for originality, and selection process repeated if the question has already been asked.

After the TESTER prints a question on the user's terminal, it automatically puts the user in calculator mode. The built-in calculator is fairly sophisticated and described in section 5. After performing whatever calculations he needs, the user types his answer, which is then checked with the internally computed answer.

Rather than merely telling the user whether he is right or wrong, and in the latter case giving him the correct answer, Ustaad gives some verbose feedback, the nature of which depends on how close to the correct answer the given answer is. Five degrees of match are checked for:

1. Superlative (error < 0.1%)

2. Good (error < 5%)

3. Wrong sign, but otherwise good (error < 5%)

4. (Just plain) wrong
5. Ridiculous (right/wrong > 100 or wrong/right > 100)

For each degree of correctness, Ustamd outputs a suitable message. If the last degree matches, Ustaad assumes that the question was misinterpreted, and reasks it Of course, if again as poor an answer is given, it gets chalked up against the user and the next question *is* asked.

The User's Questions: Whereas the drill questions the system asks can be limited to a few arbitrary kinds, the user is going to be far less tolerant of such limitations on his questions. Thus SOPHIE uses a semantic grammar ([4]) to define legal questions, which are grammatically correct English sentences. However, the types of questions SOPHIE can answer are far more complex than we have use for (since we are deliberately not using any higher level circuit dependent information in the interests of generality). What we need is a (rigid) syntax that allows expression of all reasonable questions, keeps redundancy to a ipjwtfTumn, is consistent, and yet is simple and straightforward. It bears emphasizing that not only is a pseudo-natural language understanding capability unnecessary, it could be a definite drawback. Given that only a few kinds of questions can be asked, it would be testing the student's patience to expect of him long-winded phrasings. (The only reason it might be justified would be if the above conditions could not be met for a terse input language). Since, at this level, "reasonable" questions are hardly convoluted, the above dictates of good environment practice could be satisfied easily. For consistency, the user can ask of the system the same kinds of questions that are asked of him by the TESTER, albeit in a simpler format Specifically, the query interpreter reads the user's question word by word, and looks at only the first letter of every word (fortunately, no two allowable words in any context start with the same letter). Listed below is a representative sample of the allowable question formats:

```
v n <n>                 ; for the voltage at node <n>
volt nd <n>             ; for the voltage at node <n>
voltage branch <b>      ; for the voltage across branch <b>
E b <b>                 ; for the power dissipated in branch <b>
th vlt n <nl> <n2>      ; for the thevenin voltage between
                              nodes <nl> and <n2>
n r b <b>               ; for the norton resistance seen by branch <b>
s c <bl> <b2>           ; for the current in branch <bl>
                              due to branch <b2>
```

If the user types in a word with an unmatched initial, Ustaad points out his error and lists the available options. This detailed error detection, combined with the simplicity of the formats and their mnemonic nature, should facilitate early adaptation.

While this range of questions is rather limited, we feel that it permits expressions of most inquiries the student will want to make.

It is important to note that some equivalence questions do not have well-defined answers — for example, the Thevenin voltage seen by the resistor in Fig. 3a, the Norton current seen by the resistor in Fig. 3b, and (in some cases) equivalent circuits seen by controlling branches. Such questions are checked for in the user's queries, and an appropriate response made if the check is positive.

5. THE CALCULATOR

The necessity of a calculator for solving problems in circuit electronics is obvious. The calculator in Ustaad is based on a calculator program described in [8]. Added features of Ustaad's calculator include the ability to handle real and negative numbers, built in functions, variables, and array handling. With these additions, the calculator is now perhaps the only feature of Ustaad about which there is no need to be at all apologetic!

Calculations are recursively defined, allowing any number of nested parentheses. Operator precedence is enforced. At present, the maximum variable length is five characters, an array can have a [illegible] of three [illegible] and each dimension c?n have a [illegible] length of twenty five elements. However, changing these constants is a trivial task, involving little else but the redefining of the relevant Pascal *const**.

The student can enter the calculator directly from the top level, and use it as a (superior) substitute for a hand held calculator. The calculator is also called by the TESTER after it puts a question to the user. Whenever the student has a circuit generated, or specifies one himself, circuit element values are automatically assigned as constants in fhf calculator. Conventional notation is used for the constant names. In fact, the names are the same as the labels on the branches drawn by the Graphic Display routine (see the Appendix for an example). Since some questions put to the user have infinity for a right answer, a constant named *INF* is also stored. The TESTER recognizes any answer greater than or equal to 1E10 as infinity. There are other constants automatically stored in the calculator. These provide answers to previous questions. Thus after the user answers the question: `What is the voltage at node 3?` a variable named *VN3* will be created, the value of which will be the computed answer of the question. Variables like these are only assigned for "simple" type questions. The utility of the answers for other question types for future questions is limited. Besides, as will later be described, the user can access these answers as well.

The user can also define his own variables at any time, though they cannot have the same names as the system defined constants or the built-in functions. The following functions are available to the user SQR,SQRT,SIN,COS,EXP,LN,ATAN,ABS,ARR,SOLVE. All but the last two of these are the obvious mnemonics of the common mathematical functions. In the present system the utility of many of these functions is slight However, these functions have been incorporated in anticipation that Ustaad will be expanded to include time and frequency domain analysis.

An array handling facility has been included since not all questions put to the user are easily solvable using only scalar calculations. Often, computing the answer will entail solving a set of linear equations. The function SOLVE(b,A) calls a Gaussian elimination routine, and returns the solution x of the system of equations Ax«b, provided, of course, that A is nonsingular and A and b are conformable. b and A can be variables representing arrays or they can explicitly be defined with the ARR function. In APL-like fashion, subarrays can be specified with any number of elements in any order along any dimension.

A small number of system functions have also been provided. These are invoked by prefixing them with an *@* (otherwise the calcuator interpreter would not be able to distinguish their names from variable, constant, or function names). Thus after the user has made his calculations, he gives his answer by typing @answer <expression>, where <expression> is any legal combination of variables, constants, indexed arrays, function calls, and numbers. Other system functions allow the listing of variables and constants, the listing of the available functions, reviewing previous questions and answers, going to the next question without answering the present one, the invocation of a help facility, and returning to the top level.

6. OTHER FEATURES

Custom Circuits: The user can enter his own circuit into **Ustaad.** He can then use Ustaad's interactive analysis capability or have Ustaad test his understanding of the circuit Furthermore, all circuits, both those specified by the user and those generated, can be edited. Specifically, branches can be inserted, deleted, and altered. Thus Ustaad can be used as an interactive design tool, where the student can easily switch back and forth between the EDITOR and QUERY Since QUERY only gives quantities explicitly asked for, the user will not be inundated with a flood of irrelevant information.

Graphic Display: The usefulness of a graphic display of the circuit is obvious. In the absence of such a facility, the student would have to draw out the circuit on a sheet of paper. Not only would this be inconvenient, it would negate a professed goal of Ustaad—that of reducing the student's dependence on paper and pencil. A graphic display facility is available at present for the Hewlett-Packard HP2648 graphics terminals.

7. ENVIRONMENTAL MATTERS

Ustaad contains several features which we feel encourage its use by minimising exacerbating circumstances and boredom. A number of these have been mentioned earlier. We now describe, and give our rationale for, such of these which we feel merit fuller accounts, and others which have not been mentioned so far.

- The *profile* command allows some adaptation of Ustaad's resources to better suit the capability and resources of individual students. At present PROFILE can show/change three parameters:

    1. The circuit complexity can be modified by changing the difficulty level. This affects CKTGEN in two ways: First, for higher difficulty levels the network graphs selected are of greater complexity--they have more branches and more loops; second, the weighting for element-type assignment is changed. In the present system, which has two difficulty levels, no controlled sources are assigned for level 0, and the probabilites of voltage and current sources are higher for level 0 than for level 1.

        Though at present the difficulty level affects both graph selection and element-type assignment, it would be a straightforward modification to have independent complexity indicators for these routines of CKTGEN. (This would be needed, for example, when controlled sources have not been covered in the course, but when students are expected to have a knowledge of Modified Nodal Analysis; level 0 circuits can generally be solved without resorting to array calculations.)

    2. The complexity of the questions the TESTER produces can be altered by allowing or disallowing one or more of the three questions-types (simple, equivalent, superposition). Thus Ustaad allows gradual introduction of these concepts.

    3. Lastly, the graphic display can be enabled or disabled, allowing usage with non-graphics terminals. If disabled, cirucits are listed branch by branch. Of course, if display routines were available for a variety of graphics terminals, the appropriate terminal type could be specified.

- There are a number of features associated with the TESTER and in particular with its answer checking:

    1. As mentioned earlier, the student will get a short congratulatory or remonstrative message after inputting his answer to a question. The messages are generated by randomly concatenating appropriate phrases. Fig. 4 shows how this is done for an answer with the worst degree of match. It can be seen that not too many sentences are available, but increasing their number requires little more than a fecund imagination. (A Transformational Grammar [6] would have proved useful here, but would have significantly increased the size of the program).

    2. The degrees of match ensure that students get "partial credit".

    3. Ustaad's tolerance of trifling errors and disastrous misinterpretations is reflected in its checking to see if the user has made a sign mistake but is otherwise close to the right answer, and on its reasking (once) a question to which a ridiculously wrong answer has been given.

- Ustaad maintains a log file of a student's session. This file lists all the circuits that were generated or specified during the session, all the questions that were asked along with their correct answers and the user's answers, and all the queries the user made. The student therefore has a summary of the session available to him. Furthermore, the log file specifies the random number generator seed for each generated circuit, enabling recreation of a session or some part of it The user could thus give himself a test identical to an earlier one, and discover whether or not he has corrected the deficiencies the initial taking of the test indicated

These then are the main features of Ustaad which have been incorporated in order to increase the quality of the user environment, and to thereby make interacting with Ustaad an experience devoid of exacerbation or boredom. The final judgement on the quality of our effort will only come when Ustaad sees extended use, which it has not seen as yet Undoubtedly, with such usage will come a host of complaints. Even without the benefit of proper evaluation, a few insufficiencies of the system are apparent: in particular more extensive on-line help will probably be necessary, and it has also been suggested that novice and expert modes be added (in the expert mode, for example, Ustaad*s smart-alec responses would be disabled—their purpose is to maintain the student's interest until he learns enough about Ustaad to appreciate it. for its functional merits). However, a good deal of deliberation has gone into both the user-visible parts of the system and its internal structure, and these and other modifications will probably not be more than extensions of already existing parts of Ustaad, and should not require more than straightforward programming changes.

8. CONCLUSION

Let us turn finally to some details of Ustaad's implementation. Ustaad is written in Pascal-20 and runs on a DECSYSTEM-20 at C-MU. The object code occupies about 58K (36 bit) words, of which about 8K are dedicated to the graphics routines. Though the program is fairly large, the response time is excellent. In fact it is only when Ustaad is drawing out the circuit on the screen that a noticeable wait is encountered.

A word about portability; though Pascal-20 extends Standard Pascal in a number of ways, we have in general refrained from using these extensions. The advantage of separate compilation provided by Pascal-20, however, was too great to overlook, and of course we had to conform to the file opening and closing routines.

References

[1] D.L. Bitzer, and R.L. Johnson.
PLATO: A Computer-Based System Used in the Engineering of Education.
In *Proceedings of the IEEE,* pages 960-968. IEEE. June, 1971.

[2] J.S. Brown and R.R. Burton.
Multiple Representations of Knowledge for Tutorial Reasoning.
In D.C. Bobrow and A. Collins (editors), *Representation end Understanding,* pages 311-349. Academic Press, 1975.

[3] J.S, Brown, R.B. Burton, and J. de Kleer.
Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE 1,11 and III.
In D. Sleeman and J.S. Brown (editors). *Intelligent Tutoring Systems,* • Academic Press, 1981.

[4] R.R. Burton and J.S. Brown.
Semantic Grammar: A technique for constructing natural language interfaces to instructional systems.
Technical Report AD-A043503/2, NTIS, May, 1977.

[5] J. Carbonell Jr.
Private Communication.

[6] N. Chomsky.
Syntactic Structures.
Mouton, The Hague, 19S7.

[7] A. de Geus, L.R. Pucacco, and R.A. Rphrer.
Tomorrow's Teaching Today—The Microcomputer in Engineering Education.
In *ASEE Annual Conference Proceedings.* ASEE, 1981.

[8] P. Grogono.
Programming in Pascal.
Addison-Wesley, 1980, pages 147-130.

[9] C McCorkell and R.N. Wilson.
Efficient Computer-Based Teaching through Task Syndication.
International Journal of Electrical Engineering Education 18:197-203, , .

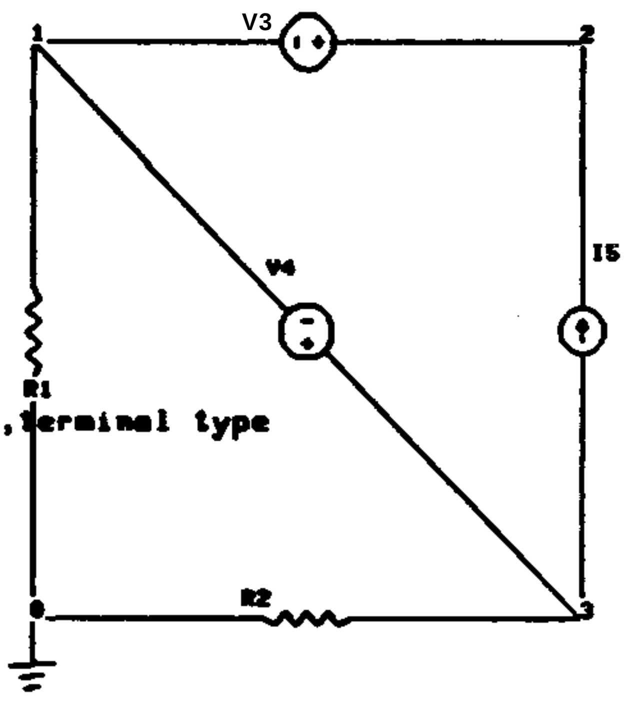
APPENDIX

SAMPLE SESSION

```
HP graphics terminal? [T] :
Do you want a drill? [I] :n
input firstseed, or <cr> :
your level [0-1] :£

Welcome,  'help* will provide succour.

%cktoen
```

'% is the top level prompt.*
The figure below shows the circuit drawn on the screen.

```
   TABLE OF VfILUCS

 • iftcsisioe    ie.iKKOhms
 • 2tCSIST0e     It-.00KOhms
 • 3VS0UKC        l.MVolts
 • 4VS0UKCC       2.00Volts
 • 5CSOURCE       1.00mAmps
Xhelp
top level i:omma nda:
   celc     : calls the calculator
   cktgen   : automatic ckt generation
   disp     : display the ckt
   edit     : specify or change a ckt
   help     : this is the extent of it
   init     : initialize
   profile  : show/change question types*level,terminal type
   query    : interactive analysis
   quit     : <while you#re ahead)
   teat     : quizzes you
X_
```



```
%help
top level oommHids:
    calc     : calls the calculator
    cktgen   : automatic ckt generation
    disp     : display the ckt
    edit     : specify or change a ckt
    help     : this is the extent of it
    init     : initialize
    profile  : show/change question types,level,terminal type
    query    : interactive analysis
    quit     :  (while you*re ahead)
    test     : quizzes you
%test
# 1:  What is the equivalent resistance
 between nodes  1 and  0 ?
 *2
```

'&' is the calculator prompt.

```
preface commands by 9
```

```
  commands are:
    answer <expression> : your answer
    fnclst : builtin functions
    Help : further info
    hint : get a hint (sometimes)
    mon : back to top level
    newqstn : next question
    previous <integtfr> : questions
    profile : show/change question types,level,terminal type
    vardump : all vars
maximum dimension length * 25
maximum number of dimensions * 3
maximum variable length « 5 chars
 &®answer 1/(1/R1+1/R2)
your answer:  5.0000000E+03•  Good.
And the Theyenin voltage?
 &evardump
15    1.0000000E-03   perm
V4    2.0000000       perm
V3    1.0000000       perm
R2    1.0000000E+04   perm
Rl    1.0000000E+04   perm
INF   1.0000000E+10   perm
```

Circuit constants are prestored and can't be overwritten.

```
 6@previous 3
THEV VOLT nodes:  1  0 *  0.0000000
```

The pending question.

```
EQUI\l RES nodes:  1  0 =  5.0000000E+03
 &il=i2=v4/(rl+r2)
  > 1.0000000E-04
```

The user can (multiply) define his own variables.

```
 &8vardump
11    1.0000000E-04   .
12    1.0000000E-04
15    1.0000000E-03   perm
V4    2.0000000       perm
V3    1.0000000       perm
R2    1.0000000E+04   perm
Rl    1.0000000E+04   perm
IMF   1.0000000E+10   perm
```

Which aren't permanent.

```
 &x=rl/rl+r2
  > 1.0001000E+04
 &@aiiswer x
your answer:  1.0001000E+04.  ??????huh
try again: THEV VOLTAGE nodes  1  0 » ?
 &x=rl/(rl+r2)
  > 5.0000000E-01
 &@answer x
your answer:  5.0000000E-01.  ??????huh
Not too good! The correct answer is -  1.00Volts
# 3:  What is the equivalent resistance
 seen by branch  2 ?
 &@answer rl
your answer:  1.0000000E+04.  Good.  No error worth mentioning!
And the Thevenin voltage?
 &@profile
 $?
```

```
        •?•  Is the profile prompt.
  e to exit;  1 for level;
  q for qstn control;  t for te type.
 $3
   currently allowed: SIMPLE    EQUIV     SUPERP
   disallow:equiv
 $fi
#4:  What is the current through branch #  1
        Since the pending question was of the type disallowed,
               a new question is generated.
 &9answer il
your answer:  1.0000000E-04.  Be wary of the sign.  Otherwise OK.
# 5:  What is the power dissipated in branch # 5
 &gnewgstn
Shame!That was a copout.    l.OOmWatts is the right answer.
# 6:  What is the current through branch #  2
 figansweril
your answer:  1.0000000E-04.  Excellent!  No error worth mentioning!
# 7:  What is the Current in branch  1 due to branch  4
 sganswer -il
your answer: -1.0000000E-04.  Good.
# 8:  What is the current through branch #  5
 &ganswer i5
your answer:  1.0000000E-03.  OK  Zero error (or near enough).
  6 out of  8 right so far.
# 9:  What is the current through branch #  4
 figansweril+i2
your answer:  2.0000000E-04.  ???????huh
try again: CURRENT branch  4 = ?
 figansweril-i5
your answer: -9.0000001E-04.  Precisely!  Zero error (or near enough).
#10:  What is the power dissipated in branch # 1
 sgvardmnp
IB4  -1.1000000E-03   perm      answers of 'simple' questions.
IB5   1.0000000E-03   perm
PB4  -2.2000000E-03   perm
IB2   1.0000001E-04   perm
PB5   1.0000000E-03   perm
IB1  -9.9999989E-05   perm
X     5.0000000E-01
11    1.0000000E-04
12    1.0000000E-04
15    1.0000000E-03   perm
V4    2.0000000       perm
V3    1.0000000       perm
R2    1.0000000E+04   perm
Rl    1.0000000E-H04  perm
INF   1.0000000E+10   perm

 figanswersar(ibl)*rl
your answer:  9.9999977E-05.  Good.  Zero error (or near enough).
#11:  What is the current through branch #  3
 &ib3

no such sclr/fnctn
 &@answer 15
your answer:  1.0000000E-03.  Good.
#12:  What is the power dissipated in branch # 2
```

```
 &gmon
        Back to top level.
%query
 •help
        ••' is the query prompt.
voltage, current, power, superposition, thevenin, norton, or quit only
 *v n 4
node number out of range,  nodes numbered 0 to  3
 »vlt Q 3
voltage branch, voltage node only
 »vlt br 4
VOLTAGE across branch  4=  2.00Volts
 •SUP ?
superp volt, superp curr, superp power only
 •sup help
superp volt, superp curr, superp power only
 •superp v 1 3
VOLTAGE across branch  1 due to branch  3 * 0 Volts
 •s v 1 4
VOLTAGE across branch  1 due to branch  4 »-  1.00Volts
 •s c 4 5
CURRENT through branch  4 due to branch  5 *-  l.OOmAmps
 #3
%quit


GoodBye.  Ustaad.log contains a summary of the session
```

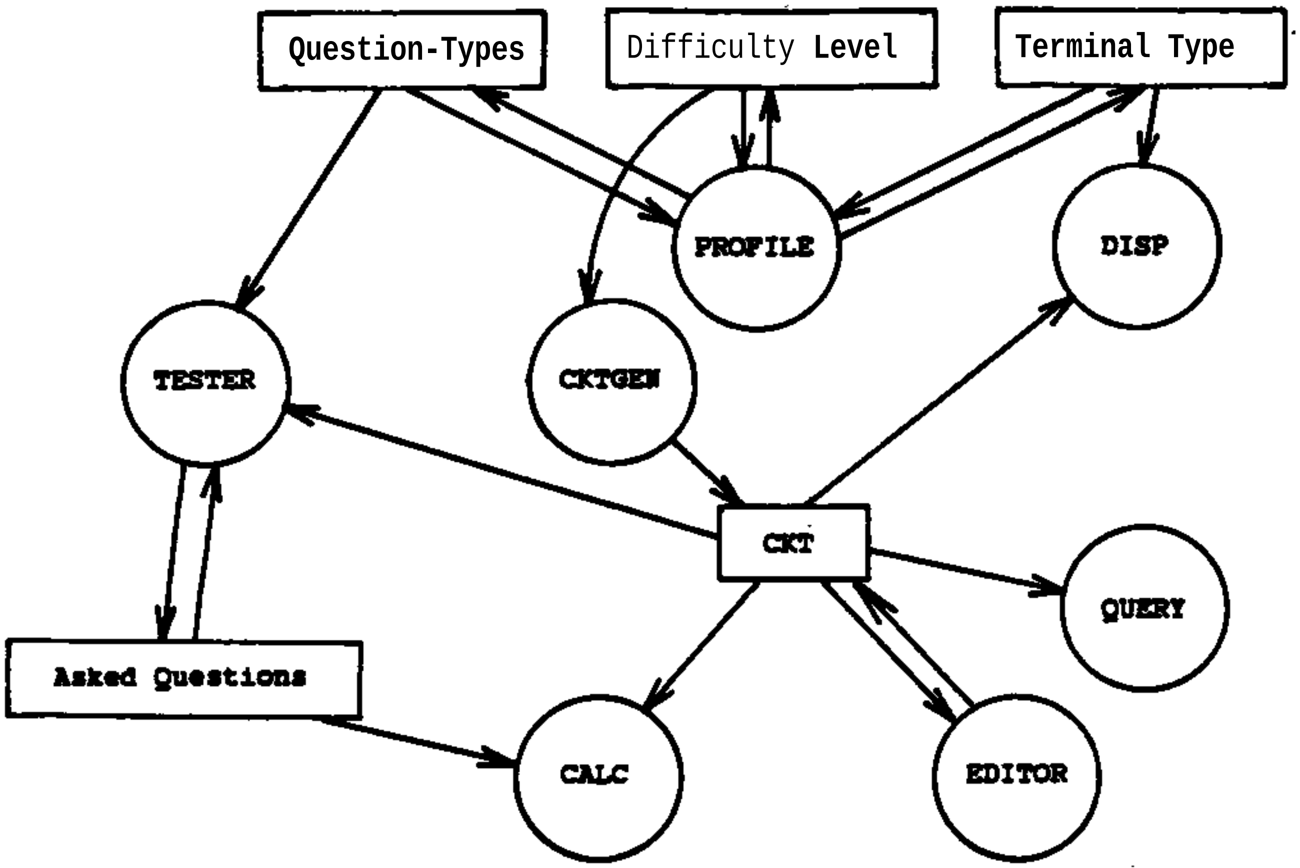


Fig. If shoving modules and globals. CRT is the current circuit.

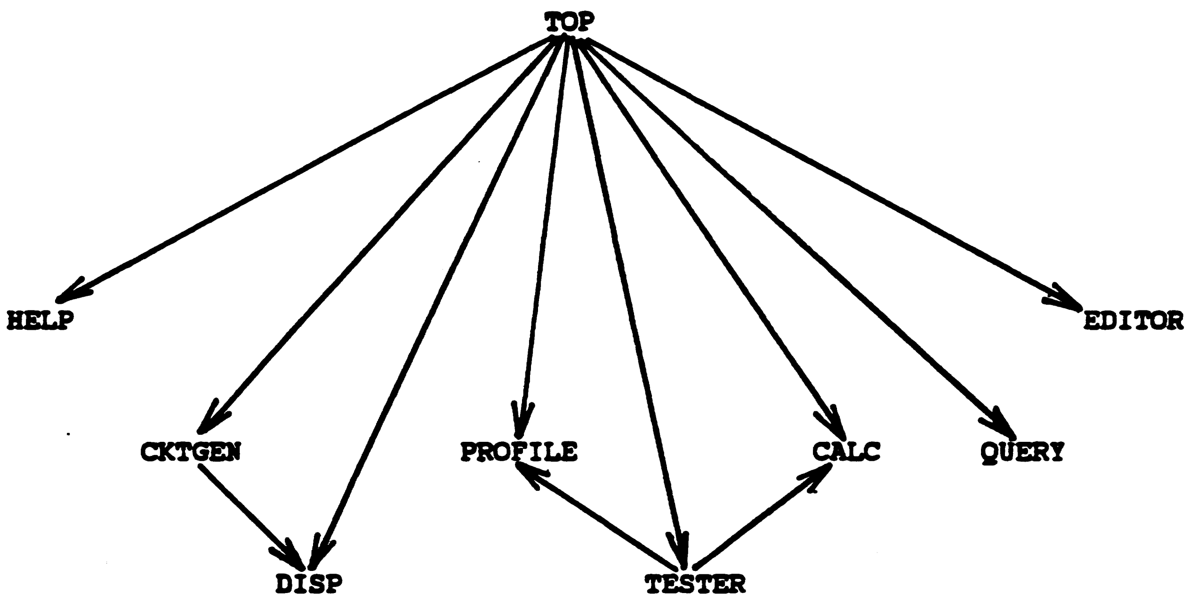


Fig. 2. Modules and control paths.

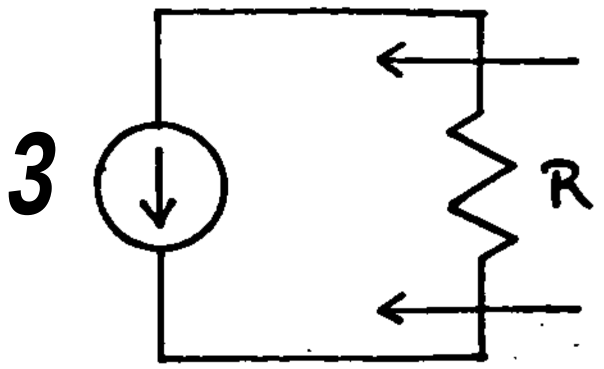
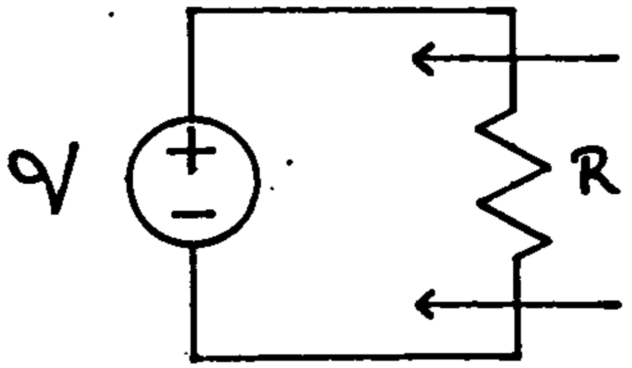


Fig. 3a. Undefined Thevenin Voltage.



F1g. 3b. Infinite Norton Current.

$$\left\{\begin{array}{l}\text{"Teh! Teh! "}\\ \text{"wrong! "}\\ \text{"Not too good. "}\\ \text{""}\end{array}\right\} + \left\{\begin{array}{l}\text{"The right asnwer is " + <value>}\\ \text{<value> + " is the right answer. "}\\ \text{"The correct answer was " + <value>}\\ \text{<value> + " was the answer. "}\end{array}\right\} +$$

$$\left\{\begin{array}{l}\text{"Your error is " } + \left\{\begin{array}{l}\text{"incredible."}\\ \text{"incomprehensible."}\\ \text{"too much."}\\ \text{<numerical value>}\end{array}\right\}\\ \\ \text{""}\end{array}\right\}$$

Fig. 4. Example of response generation.