This blank page was inserted to preserve pagination.

A Simple Model of Circuit Design

by

Gerald Lafael Roylance

Copyright Massachusetts Institute of Technology 1983

Revised version of a Master of Science Thesis submitted to
the Department of Electrical Engineering and Computer Science
in May 1980.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contracts N00014-80-C-0505 and N00014-80-C-0622.

I'd like to thank Johan de Kleer for his perspective on life and research at MIT. He kept me sane in bedlam. Gerald Sussman, my advisor, for his suggestions -- both good and bad. Johan, again, for telling me which were good and which were bad. Dick Waters and Dave McAllester provided helpful criticism.

I stole the rule interpreter from Jon Doyle and Howard Shrobe.

Several programs helped prepare this report: Richard Stallman's EMACS did the text editing, the program R (written and maintained by Alan Snyder and Elliot Moss) did the typesetting, and SIL and PRESSEDIT (running on a Xerox Alto computer) prepared and merged the illustrations. The result was printed on a Xerox Dover printer. Xerox Corporation graciously donated the Alto computers and the Dover printer to MIT.

And I like to thank Susan Burkhardt for helping me through some difficult times.

A Simple Model of Circuit Design

by

Gerald Lafael Roylance

Abstract

A simple analog circuit designer has been implemented as a rule based system. The system can design voltage followers, Miller integrators, and bootstrap ramp generators from functional descriptions of what these circuits do. While the designer works in a simple domain where all components are ideal, it demonstrates the abilities of skilled designers. While the domain is electronics, the design ideas are useful in many other engineering domains, such as mechanical engineering, chemical engineering, and numerical programming.

Most circuit design systems are given the circuit schematic and use arithmetic constraints to select components values. This circuit designer is different because it designs the schematic. The designer uses a unidirectional CONTROL relation to find the schematic. The circuit designs are built around this relation; it restricts the search space, assigns purposes to components, and finds design bugs.

This empty page was substituted for a blank page in the original document.

CONTENTS

1. Designing Circuits 6

1.1 Introduction 6
1.2 Design Strategy 7
1.3 What's Ahead 9

2. A Ramp Generator 10

2.1 Basic Rules 10
2.2 Design Rules 16
2.3 Design of a Ramp Generator 17
2.4 Loose Ends 24

3. KCL and Feedback 25

3.1 A Design Bug 25
3.2 A Closer Look at KCL 26
3.3 The Feedback Rule 30
3.4 Another Design 34
3.5 Still More Generators 37
3.6 Summary 42

4. Discrete State 43

4.1 Introduction 43
4.2 Design of Independent States 43
4.3 Design of Dependent States 50
4.4 Summary 56

5. Literature 57

6. Conclusions 60

6.1 The Imperfect Past 60
6.2 The Insufficient Present 61
6.3 The Absolutely Perfect Future 62

7. Bibliography 63

1. Designing Circuits

1.1 Introduction

Many people have made mechanical experts that solve difficult problems. Some of these expert programs are mathematicians who discover mathematical ideas <Lenat>, programmers who write code <Manna> <Barstow>, physicists who solve mechanics problems <de Kleer-1>, chemists who synthesize organic molecules <Sridharan>, consultants who diagnose medical problems <Shortliffe>, and electrical engineers who design circuits <McDermott-2>.

This thesis is also about circuit design and describes a simple rule based circuit designer for operational amplifier circuits. It starts with a few circuit design ideas that mimic the behavior of human designers. These ideas are encoded in a computer program, and several classic circuit designs come out. The resulting circuit design system is different from most other circuit design programs because it designs circuits: it chooses the components and how to connect them together. Most other programs start with the connected components and solve for component values using arithmetic constraints.[1] Finding a circuit topology is a more creative task.

Creating an automated circuit designer is beneficial for several reasons: (1) human experts are scare -- we'd like to replace them with more available computer experts <Gorry>. (2) Automated experts are economical. Expert circuit designers are scarce and specialized. Every circuit designer does not know how to design every circuit. Only a few designers, for example, can design high speed analog switches. If an expertise can be incorporated into a widely-distributed automated expert, then human designers could make use of the expertise without learning its details. (3) If we can teach a computer to do a task, then we should be much better at teaching people to do the same task.

Another benefit of an automated circuit designer is due to the disparity between what we can afford to build and what we can afford to design. Today's electronics technology makes it possible to cheaply build many circuits. Microprocessors, for example, cost under $10. Unfortunately there is a barrier to this cheap technology: the cost of designing a circuit far outweighs the cost of its components. Several years and hundreds of thousands of dollars can be spent designing a circuit that costs less than a thousand dollars. This problem is the hardware analog of Winograd's complexity barrier in software engineering <Winograd>.

To demonstrate that it is possible to build an automated designer, some simple rules have been developed that capture some of what an analog circuit designer knows. These rules design circuits from problem descriptions about what the circuits do and not about what parts to use or how to connect them together. Some of these rules have run in an interpreter and have designed voltage followers, current sources, Miller ramp generators, and bootstrap ramp generators. The thesis will describe these rules and discuss some aspects of their implementation. The program fragments and rules given in the text are somewhat different from the actual program because the distinction between causality and constraint was not clear in the original

1. <McDermott-2> is a notable exception.

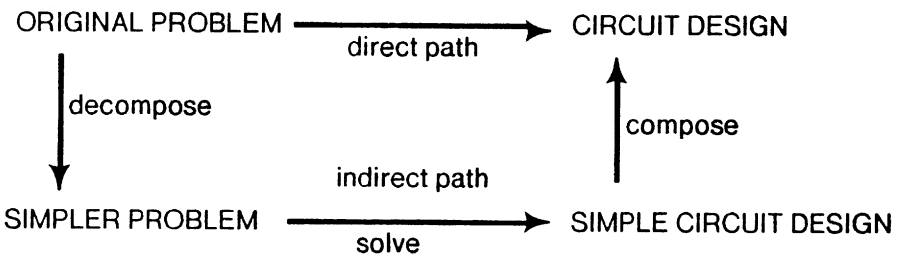
implementation. The section on switches was not completely implemented. More will be said about this later.

This work address only simple design issues and ignores many others. My rules do not attempt to produce a "best" design. When my program must decide among alternatives, it asks the user to choose one. If the program detects a mistake, then the user is told to select one of the other alternatives. This method was chosen because it is simple and allowed me to explore different solutions. The system shows how a machine can generate plausible designs. An expert designer will sit down at his desk and in the course of an hour or so will make several rough schematics. It is the knowledge involved in the development of these schematics, made without lengthy calculation, that these rules are intended to capture.

1.2 Design Strategy

Design problems can be simple or difficult. Simple problems can be solved directly (figure 1). More complicated problems must be decomposed into simpler parts that can be solved individually and then the partial solutions composed into a total solution. This strategy (and its recursive generalization) is not new: it travels under names such as divide and conquer or linear decomposition. The central problem in building an expert design system is understanding how experts break problems up and compose solutions.

Fig. 1. Design Strategy



Some of the simple problems that designers can solve easily are listed below. Some problems are low level -- such as turning a current into a voltage by forcing it through a resistor -- and others are more general strategies -- such as using feedback to make two voltages equal. These design rules are used unidirectionally -- the current through the resistor CONTROLs the voltage across the resistor. The CONTROL relation is transitive and acyclic. The designer uses the rules unidirectionally to create mechanisms where the inputs control the outputs. Some example rules, expressed in English, are:

> **To make one terminal current control another, connect the two terminals together. KCL will insure the currents are equal.**

Two terminal voltages can be made equal by wiring the terminals together.

Negative feedback will copy one node voltage (phase angle, current, time) to another node voltage (phase angle, current, time).

Resistors convert voltages into currents or currents into voltages.

Capacitors take derivatives and integrals.

Switches let two different things CONTROL the same thing at different times.

Circuit goals in different discrete states can be achieved independently except for the continuous state information stored in capacitors.

An initial condition in one state is a final condition of the immediately preceding state.

These design rules and some more detailed analysis rules are organized into a compulsive debugging system. The system is compulsive because it tries to do as much as it can based on local information. The analysis rules keep the system honest, however, by checking up on what has been done. When mistakes in the design are found, they are removed. Debugging is a classic technique in the AI literature <Sussman-1> and has advantages over systems that don't produce bugs. Debugging systems are easier to write because the rules are local -- they do not have to anticipate all the consequences of their actions. Debugging systems, for similar reasons, are also modular. The simplicity of the local rules makes them less likely to have mistakes. Bugs also serve as a focus for design problems: the circuit designer described here expects to see design bugs in several situations. The designer effectively says, "Let's see if I can get away with using a wire here -- if I can't, then the analyzer will point out a design bug and I'll use a more complex method." Some components -- switches -- are only introduced to fix design bugs. Debugging is also a reasonable model of how people solve problems. These advantages do not mean, however, that debugging systems will compensate for poor organization or stupid rules.

1.3 What's Ahead

The following chapters describe some electronic design skills in a rule based system. I assume the reader is familiar with such systems (see <Winston>) and has a modest knowledge of electronics. Chapter 2 designs a simple ramp generator using design rules based upon the simple network theory concepts of device voltage and current constraints (VICs), Kirchoff's voltage law (KVL), and Kirchoff's current law (KCL). Chapter 3 uses a feedback design rule to design a couple variations on the simple ramp generator circuit. Chapter 4 introduces discrete states and switches. The final chapters discuss the literature and conclusions.

2. A Ramp Generator

2.1 Basic Rules

A designer is skilled in the use of various devices and design strategies. My automatic design system encodes these design skills as a collection of facts and rules. A rule interpreter applies this knowledge to the design of particular circuits. This chapter describes some simple design and analysis rules and shows how they result in the design of circuits.

The design rules must not only capture the purely mathematical constraints given by VICs, KVL, and KCL, but also how those constraints can implement mechanism. Mathematical constraints tell us an amplifier's input and output voltages are related. Designers also need a notion of mechanism or control that says the amplifier's input controls its output. The design rules use CONTROL extensively to organize the search for a circuit topology. More will be said about CONTROL later.

The design system uses Shrobe's rule interpreter <SHROBE>. It is derived from the original AMORD <Doyle> <de Kleer-3> but has slightly different conventions for defining rules. In addition, Shrobe has implemented a skeletal task agenda. Both interpreters are built upon belief maintenance systems that provide backtracking and maintain justifications for assertions in the database. These justifications are useful not only for "dependency directed backtracking" <Stallman>, but also for explaining deductions and debugging bad rules.

This chapter uses a ramp generator as a design example. A ramp generator is a circuit with no inputs and one output that produces a rising voltage as shown in figure 2. A ramp generator's output voltage has constant slope. The rules given in this chapter will design the simple circuit that is also shown in the figure. Here is a brief description of how the circuit works: the constant current generator (GEN-1) forces a constant current through the capacitor (CAP-1); the constant current forces the voltage across the capacitor to increase linearly with time, thus producing a constant slope at the output. The VICs (voltage and current constraints) for a capacitor, KVL, and KCL are central elements in the understanding of this circuit, and therefore will be given in detail. The capacitor will be covered first.

The VIC for a capacitor is

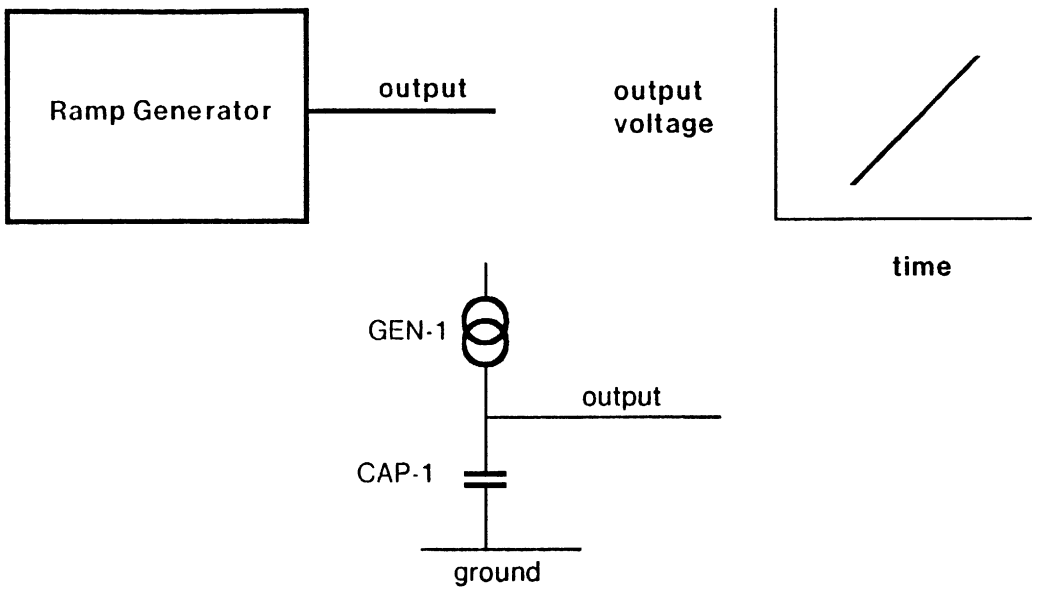
$$\frac{dv}{dt} = \frac{i}{C}$$

where v is the branch voltage across the capacitor, i is the branch current through the capacitor, and C is the capacitance. This equality is expressed in the design system as

```
(= (D-DT (BRV CAP-1)) (// (BRI CAP-1) (CAPACITANCE CAP-1)))  .
```

Though this equation is correct, it does not capture how a designer uses a capacitor because an equation does not capture the notion of mechanism. While the absence of mechanism is a feature of network theory, it is a weakness in design where designers use equalities unilaterally to set values <de Kleer-5>.

Fig. 2. A Ramp Generator



Control is an important distinction: either the current sets the branch voltage or the branch voltage sets the current, but they are not simultaneously constrained. A designer uses control because he wants the inputs to determine the outputs, not vice versa. A unity gain amplifier, a circuit whose output voltage should equal its input voltage, should not enforce that equality by controlling its input voltage to be equal to its output. The direction of control is expressed using the CONTROLS relation:

```
(CONTROLS (D-DT (BRV CAP-1)) (BRI CAP-1))
(CONTROLS (BRI CAP-1) (D-DT (BRV CAP-1)))
```

Only one of these assertions can be active at one time. The first relation says that the derivative of v controls the current i. The second relation says the current i controls the derivative. These are ways a capacitor can be used in a circuit. The arguments of CONTROLS are signals -- voltages and currents. The capacitance C does not appear in the CONTROLS relation because the capacitance is just a parameter; it is not set by any voltage or current in the circuit but is set when the capacitor is manufactured.

The CONTROLS relation is transitive, that is, if A controls B and B controls C, then A controls C. This transitivity is presented in the first rule:

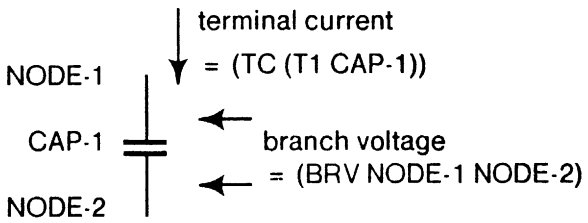
```
(DEFRULE CONTROL-TRANSITIVITY
         ((?F1 (CONTROLS ?X ?Y))
          (?F2 (CONTROLS ?Y ?Z)))
  (ASSERT '(CONTROLS ?X ?Z) '(CONTROL-TRANSITIVITY ?F1 ?F2)))
```

DEFRULE defines a rule. In this case, the rule name is "control-transitivity". The second argument of

DEFRULE is a list of patterns; each pattern has the form (<factname> <fact>). The "?" prefix denotes a pattern variable. This particular rule looks for an assertion of the form (CONTROLS ?X ?Y); when it finds one, the rule binds ?X and ?Y to the corresponding fields of the assertion and ?F1 to the unique factname of that assertion. The rule then looks for an assertion (CONTROLS ?Y ?Z) and if it finds one will bind ?Z to the appropriate value and ?F2 to the unique name for that assertion. If both matches are found, the body of the rule is executed; in this case a new assertion, (CONTROLS ?X ?Z), is made. The second argument to ASSERT records the assertion's support that is used in backtracking. The new assertion depends on facts ?F1 and ?F2. If either fact is not believed at some later time, then the new assertion will be withdrawn. The database maintains this consistency. For more information about the rule interpreter, see <SHROBE>

The capacitor rule (figure 3.) is more complicated than the transitivity rule because some bookkeeping has to be done.

Fig. 3. Capacitor-VIC-1



TEK2112

```
(DEFRULE CAPACITOR-VIC-1
        ((?F1 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F2 (CONTROLS ?X (BRV [(CONNECT (T1 ?CAP) *)]
                                [(CONNECT (T2 ?CAP) *)])))))
  (IF (... non-circularity test ...)
      (ASSERT '(CONTROLS (D-DT (BRV [(CONNECT (T1 ?CAP) *)]
                                    [(CONNECT (T2 ?CAP) *)]))
                         (TC (T1 ?CAP)))
              '(CAPACITOR-VIC ?F1 ?F2))))
```

The first pattern looks for a capacitor ("(CIRCUIT-PART CAPACITOR ?CAP)") and binds the name of the capacitor to ?CAP (all circuit components are given a reference name). The CIRCUIT-PART assertion associates parts such as resistors, capacitors, and amplifiers with reference names. The next pattern finds some ?X that controls the branch voltage ("(BRV <node> <node>)") across the capacitor; the syntax of the bracket expressions ("[]") will be discussed shortly. When the rule finds an ?X, it asserts that the derivative of the branch voltage controls the capacitor's terminal current ("(TC (T1 ?CAP))"). TC refers to terminal current; T1 and T2 refer to the two terminals of a capacitor. The idea behind CAPACITOR-VIC-1

is that if some external thing controls a capacitor's branch voltage, then the causality in the capacitor will be from the branch voltage to the terminal current, not the other way around.

The IF statement is there to watch out for a circular CONTROLS loop (we don't want to loose the distinction between CONTROLS and =). It prevents the system from thinking both

```
(CONTROLS (BRV NODE-1 NODE-2) (TC (T1 CAP-1)))
(CONTROLS (TC (T1 CAP-1))     (BRV NODE-1 NODE-2))
```

In fact, a good idea would be to look for circular CONTROLS loops because they mean something is wrong (figure 4.).

Fig. 4. Circular CONTROLS Bug

```
(DEFRULE CIRCULAR-CONTROLS-LOOP
        ((?F1 (CONTROLS ?X ?X)))
  (ASSERT '(BUG CIRCULAR-CONTROLS-LOOP) '(CIRCULAR-LOOP-CHECK ?F1)))
```

The expressions in square brackets ("[]") refer to the nodes to which the capacitor is connected. The assertion (CONNECT (T1 CAP-1) NODE-1) means terminal T1 of component CAP-1 is connected to node NODE-1, and the bracket expression [(CONNECT (T1 CAP-1) *)] would then refer to NODE-1. In general, if (P X Y), then [(P X *)] = Y.

If there is no (CONNECT (T1 CAP-1) NODE-1), then the reference expression creates an anonymous node. Later this node may be identified with another node, in which case

```
(ID <node-name1> <node-name2>)
```

is asserted. This assertion causes one of the two node names to be chosen and all assertions that use the other name are reasserted using the selected name. The assertions using the old name will no longer trigger any rules. This naming mechanism is also in Shrobe's interpreter.

A short digression on naming conventions is in order. Instances of generators, resistors, capacitors, and nodes are given names such as GEN-1, RES-1, CAP-1, and NODE-1. Some instances of nodes are given more descriptive names such as INPUT and OUTPUT. Variable names (names inside of rules) refer to these parts as GEN, RES, CAP, and NODE (or sometimes ND). When more than one variable is needed, names such as NODE1 and NODE2 (no hyphen) are used. A name ending with a hyphen followed by a number is an instance, not a variable.

Device terminals are denoted by functions on an instance (which might be represented by a variable): (T1 CAP-1) is one terminal of capacitor CAP-1 and (T2 CAP-1) is the other. T1 and T2 are used for all 2 terminal devices. Amplifiers have terminal designators I+, I-, and OUT. Values of device parameters such as resistance and capacitance are referred to by (RESISTANCE RES-1) or

(CAPACITANCE CAP-1). Currents flowing into terminals are called terminal currents and are designated (TC <terminal>). Voltages are represented by functions on nodes: (BRV NODE-1 NODE-2) is the branch voltage from NODE-1 to NODE-2 and (NDV NODE-1) is the node voltage of NODE-1.

The capacitor-VIC-1 rule only considers the case when an imposed branch voltage sets the capacitor's branch current. Capacitor-VIC-2, shown in figure 5, covers the other case of the branch current controlling the branch voltage.

Fig. 5. Capacitor-VIC-2

```
(DEFRULE CAPACITOR-VIC-2
        ((?F1 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F2 (CONTROLS ?X (TC (T1 ?CAP)))))
  (IF (... non-circularity test ...)
      (ASSERT '(CONTROLS (TC (T1 ?CAP))
                         (D-DT (BRV [(CONNECT (T1 ?CAP) *)]
                                    [(CONNECT (T2 ?CAP) *)])))
              '(CAPACITOR-VIC ?F1 ?F2))))
```

Figure 6 shows some rules that relate terminal currents, VICs, and wiring.

In addition to the CONTROLS constraint, there are also arithmetic constraints (=) that do not have the restrictions of causality that CONTROLS has. These arithmetic constraints not only refer to branch voltages and terminal currents, but also to parameter values such as resistance and capacitance. CONTROLS constraints are included to find mechanism; arithmetic constraints are included to find values for different component parameters. This discussion will focus on CONTROLS and largely ignore the arithmetic constraints except to write down some important constraints from time to time. CONTROLS is important for finding the topology of a circuit; others (<de Kleer-4> <Sussman-4>) have discussed how to use arithmetic constraints. My program does not actually use the form of the constraints shown here; instead it put descriptive markers on the CONTROLS assertions. That approach is limited; the ideas of constraint and control are independent and should be separate deductions.

Fig. 6. Terminal Currents

```
(DEFRULE CAP-VIC-3
        ((?F1 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F2 (CONNECT (T1 ?CAP) ?ND1))
         (?F3 (CONNECT (T2 ?CAP) ?ND2)))
  (ASSERT '(= (TC (T1 ?CAP))
              (* (CAPACITANCE ?CAP) (D-DT (BRV ?ND1 ?ND2))))
          '(CAPACITOR-VIC ?F1 ?F2 ?F3))
  (ASSERT '(= (TC (T1 ?CAP)) (- (TC (T2 ?CAP))))
          '(CAPACITOR-VIC ?F1 ?F2 ?F3)))

(DEFRULE TERMINAL-CURRENT-0
        ((?F1 (CIRCUIT-PART ?TYPE ?COMP)))
  (ASSERT '(= (TC (T1 ?COMP)) (- (TC (T2 ?COMP))))
          '(TERMINAL-CURRENT ?F1)))

(DEFRULE TERMINAL-CURRENTS-1
        ((?F1 (CONTROLS ?X (TC (T1 ?COMP)))))
  (IF (... non-circularity-test ...)
      (ASSERT '(CONTROLS (TC (T1 ?COMP))
                         (TC (T2 ?COMP)))
              '(KCL ?F1))))

(DEFRULE TERMINAL-CURRENTS-2
        ((?F1 (CONTROLS ?X (TC (T2 ?COMP)))))
  (IF (... non-circularity-test ...)
      (ASSERT '(CONTROLS (TC (T2 ?COMP))
                         (TC (T1 ?COMP)))
              '(KCL ?F1))))
```

2.2 Design Rules

All of the rules given so far are analysis rules. They notice the CIRCUIT-PARTs and CONNECTions of a circuit and deduce the controlled currents and voltages. To design, there should be design rules that start with goals for controlling current and voltages and deduce the appropriate CIRCUIT-PARTs and CONNECTions needed to achieve those goals. A design rule is generally the inverse of an analysis rule: the design rule adds the CIRCUIT-PARTs and CONNECTions that will cause the analysis rule to deduce the goal.

Figure 7 shows the first design rule; it is the inverse of an analysis rule given earlier (rules usually come in design-analysis pairs).

Fig. 7. Capacitor-Design-1

```
(DEFRULE CAPACITOR-DESIGN-1
         ((?F1 (GOAL (CONTROLS ?X (D-DT ?Y)))))
  (PROPOSE-METHOD (?F1) ?F2
    (LET ((CAP (GENPREFIX 'CAP)))
      (ASSERT      '(CIRCUIT-PART CAPACITOR ,CAP)
                   '(CAP-DESIGN ?F2))
      (GOAL-ASSERT '(CONTROLS ?X (TC (T1 ,CAP)))
                   '(CAP-DESIGN ?F2))
      (GOAL-ASSERT '(CONTROLS (BRV [(CONNECT (T1 ,CAP) *)]
                                   [(CONNECT (T2 ,CAP) *)])
                              ?Y)
                   '(CAP-DESIGN ?F2)))))
```

This rule looks for the GOAL of something (?X) controlling the derivative of something else (?Y). The body of the LET creates a capacitor (giving it a unique name with the GENPREFIX function) and then asserts two subgoals with GOAL-ASSERT. The method associates ?X with the capacitor's terminal current and ?Y with the capacitor's branch voltage because of the possibility of using the capacitor VIC to establish the required behavior between the terminal current controlling the derivative of the capacitor branch voltage and ?X controlling the derivative of ?Y. The rule uses the capacitor's VICs to take a derivative. The subgoals try to make causal links between ?X and the branch current and between the branch voltage and ?Y. This design rule may not be the only design rule that can achieve the goal; there may be many others.

PROPOSE-METHOD suggests a method to achieve the goal. In general, several methods may be proposed for one goal; the task controller will select one and execute its body. PROPOSE-METHOD has the form

```
(PROPOSE-METHOD <list of factnames justifying proposing the method>
                <factname justifying selection of the method>
  . <body>)
```

The list of factnames labels the support for proposing the method. The task controller, however, decides if the method will be used; the second argument of PROPOSE-METHOD is the reason for proposing the method. The body is a list of expressions to execute when the method is selected.

A design rule proposes one or more methods to achieve a goal. A task controller selects one of the proposed methods and runs it. The method may add circuitry or spawn subgoals. If the desired goal is achieved, then the method was successful and some other goal can be pursued. If the desired goal is not achieved, then the method failed and all the circuitry and deductions it made are retracted (by virtue of the truth maintenance system) and a different method of achieving the goal is tried. If all the methods fail, then the designer fails to achieve the goal; it must discard that goal and work on another. This backtracking is under the control of task agenda portion of Shrobe's interpreter.

2.3 Design of a Ramp Generator

It's time to jump in and design that ramp generator mentioned in the first part of this chapter. The ramp generator is a circuit with an output that has a constant slope; this goal is shown in figure 8. Also shown in the figure are the deductions that the capacitor design rule makes when it is triggered by the goal.

The GOAL-ASSERT and ASSERT specify the design goal. The ramp generator has no inputs, so nothing (NIL) CONTROLs the output. The capacitor design rule notices the goal and, when the task controller lets it, runs. It makes capacitor CAP-1 whose two terminals are connected to NODE-1 and NODE-2 and establishes two subgoals to fill in the design. The first is to set the current through the capacitor and the second is to make the capacitor branch voltage appear between OUTPUT and GROUND.

To achieve the first subgoal, some method of setting the capacitor's terminal current must be found. It shouldn't matter which terminal of the capacitor is controlled but right now the only goal is to control T1 of CAP-1. The rules shown in figure 9 correct this situation. They say if there is a goal to control T1 (T2), then propose achieving that goal by controlling the T2 (T1), the other terminal. These rules are the design versions of the earlier terminal current analysis rules.

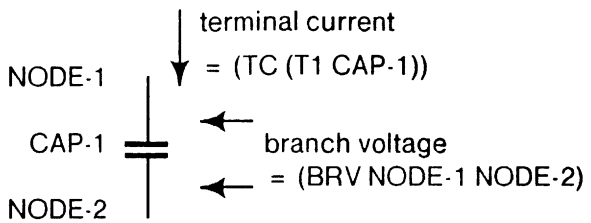
In the case of the ramp generator design, TC-DESIGN-1 proposes solving

```
(GOAL (CONTROLS NIL (TC (T1 CAP-1))))
```

by achieving

```
(GOAL (CONTROLS NIL (TC (T2 CAP-1))))  .
```

Fig. 8. Ramp Generator Deductions




```
(goal-assert '(controls NIL (d-dt (ndv output)))
             '(premise))
(assert      '(= (d-dt (ndv output)) m)
             '(premise))

(GOAL (CONTROLS NIL (D-DT (NDV OUTPUT))))
(= (D-DT (NDV OUTPUT)) M)

(CIRCUIT-PART CAPACITOR CAP-1)
(CONNECT (T1 CAP-1) NODE-1)
(CONNECT (T2 CAP-1) NODE-2)
(GOAL (CONTROLS NIL                (TC (T1 CAP-1))))
(GOAL (CONTROLS (BRV NODE-1 NODE-2) (NDV OUTPUT))
(= (TC (T1 CAP-1)) (* (CAPACITANCE CAP-1) (D-DT (BRV NODE-1 NODE-2))))
```

Fig. 9. Terminal Current Design Rules

```
(DEFRULE TC-DESIGN-1
        ((?F1 (GOAL (CONTROLS ?X (TC (T1 ?COMP))))))
  (IF (... circularity test ...)
      (PROPOSE-METHOD (?F1) ?F2
       (GOAL-ASSERT '(CONTROLS ?X (TC (T2 ?COMP))) '(TC-DESIGN ?F2)))))

(DEFRULE TC-DESIGN-2
        ((?F1 (GOAL (CONTROLS ?X (TC (T2 ?COMP))))))
  (IF (... circularity test ...)
      (PROPOSE-METHOD (?F1) ?F2
       (GOAL-ASSERT '(CONTROLS ?X (TC (T1 ?COMP))) '(TC-DESIGN ?F2)))))
```

This is, however, only a proposal that the task controller would have to accept. Since both the goal and the subgoal are similar, we (not the program) will ignore the subgoal while other methods of setting a terminal current goal are considered. Later, in the next chapter, we will return to this proposal and examine the designs that stem from it.

KCL can be used to set the terminal current of (T1 CAP-1). Simple versions of the KCL design and analysis rules are shown in figure 10; the next chapter discusses these rules more thoroughly. KCL requires the current flowing into a node equal the current flowing out of the node, so if some other terminal forces some current into (T1 CAP-1)'s node, then that current must flow out of the node and into (T1 CAP-1). The KCL design rule proposes to set a terminal current by setting the current flowing into a node. This current flowing into or out of a node is denoted (NC <node>). The KCL rule assumes there are no other terminals connected to the node that might siphon off some current that is supposed to go to (T1 CAP-1). If that assumption is false, then there is a design bug; that bug, how to fix it, and how to use it to advantage, will be taken up in the next chapter.

Fig. 10. KCL Rules

```
(DEFRULE KCL-ANALYSIS-0
        ((?F1 (CONTROLS ?X (TC ?TERM)))
         (?F2 (CONNECT ?TERM ?NODE)))
  (IF (... circularity test ...)
      (ASSERT '(CONTROLS (TC ?TERM) (NC ?NODE))
              '(KCL ?F1))))

(DEFRULE KCL-ANALYSIS-1
        ((?F1 (CONTROLS (TC ?TERM1) (NC ?NODE)))
         (?F2 (CONNECT  ?TERM1 ?NODE))
         (?F3 (CONNECT  ?TERM2 ?NODE)))
  (IF (NOT (EQUAL ?TERM1 ?TERM2))
      (PROGN (ASSERT '(CONTROLS (NC ?NODE) (TC ?TERM2))
                     '(KCL ?F1 ?F2 ?F3))
             (ASSERT '(= (TC ?TERM1) (- (TC ?TERM2)))
                     '(KCL ?F1 ?F2 ?F3)))))

(DEFRULE KCL-DESIGN
        ((?F1 (GOAL (CONTROLS ?X (TC ?TERMINAL))))
         (?F2 (CONNECT ?TERMINAL ?NODE)))
 (PROPOSE-METHOD (?F1 ?F2) ?F3
   (GOAL-ASSERT '(CONTROLS ?X (NC ?NODE))
                '(KCL-DESIGN ?F2 ?F3)) ))
```

The idea of a node current (NC) is based on causality and is independent of assigning arbitrary reference directions for currents and voltages. A node current says that some branch sets the current being

pushed into or pulled out of a node and the remaining branches receive that current. That current is called the node current. The branches that set the node current causally control the branches that receive it.

Applying the KCL-DESIGN rule to

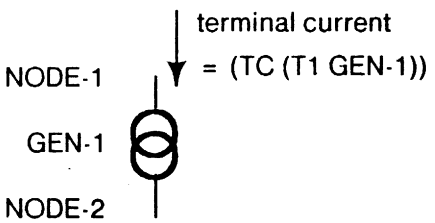
```
(GOAL (CONTROLS NIL (TC (T1 CAP-1))))
```

leaves our last subgoal on this branch of the goal tree,

```
(GOAL (CONTROLS NIL (NC NODE-1)))
```

This goal is achieved by the constant current generator rule shown in figure 11.

Fig. 11. Current Generator




```
(DEFRULE CURRENT-GENERATOR-ANALYSIS
        ((?F1 (CIRCUIT-PART CURRENT-GENERATOR ?GEN)))
  (ASSERT '(CONTROLS NIL      (TC (T1 ?GEN)))
          '(CURRENT-GENERATOR ?F1))
  (ASSERT '(= (STRENGTH ?GEN) (TC (T1 ?GEN)))
          '(CURRENT-GENERATOR ?F1)))

(DEFRULE CURRENT-GENERATOR-DESIGN
        ((?F1 (GOAL (CONTROLS NIL (NC ?NODE)))))
  (PROPOSE-METHOD (?F1) ?F2
    (LET ((GEN (GENPREFIX 'GEN)))
        (ASSERT '(CONNECT (T1 ,GEN) ?NODE)
                '(CURRENT-GENERATOR ?F2))
        (ASSERT '(CONNECT (T2 ,GEN) [(CONNECT (T2 ,GEN) *)])
                '(CURRENT-GENERATOR ?F2))
        (ASSERT '(CIRCUIT-PART CURRENT-GENERATOR ,GEN)
                '(CURRENT-GENERATOR ?F2)))))
```

The current generator satisfies the above goal and allows the goal stack to unwind as analysis rules fire and deduce the achievement of the goals (figure 12).

The only remaining subgoal is making capacitor CAP-1's branch voltage control the ramp

Fig. 12. Current Generator Deductions

```
(GOAL (CONTROLS NIL (TC (T1 CAP-1))))          ;Capacitor Design

(GOAL (CONTROLS NIL (NC NODE-1)))              ;KCL Design
(CONNECT (T1 GEN-1) NODE-1)                    ;Current Generator
(CIRCUIT-PART CURRENT-GENERATOR GEN-1)

(CONTROLS NIL             (TC (T1 GEN-1)))     ;KCL Analysis
(CONTROLS (TC (T1 GEN-1)) (NC NODE-1))
(CONTROLS NIL             (NC NODE-1))         ;satisfies goal
(CONTROLS (NC NODE-1)     (TC (T1 CAP-1)))
(CONTROLS NIL             (TC (T1 CAP-1)))     ;satisfies goal

(= (STRENGTH GEN-1) (TC (T1 GEN-1)))           ;Current generator
(= (TC (T1 GEN-1))  (- (TC (T2 GEN-1))))       ;Terminal Current
(= (TC (T1 GEN-1))  (- (TC (T1 CAP-1))))       ;KCL
(= (STRENGTH GEN-1) (- (TC (T1 CAP-1))))       ;math
```

generator's output node voltage. A node voltage is just a branch voltage between the node and ground, so converting a branch voltage into a node voltage involves referencing one of the nodes of the branch to ground and having the other node of the branch be the node voltage. The conversion rule is shown in figure 13. It creates two subgoals, one to reference one side of the branch voltage to ground and the other to get the other side of the branch voltage controlling the node voltage. Another rule is needed to tell how to achieve these subgoals.

Fig. 13. BRV to NDV Conversion

```
(DEFRULE BRV-TO-NDV-DESIGN
        ((?F1 (GOAL (CONTROLS (BRV ?NODE1 ?NODE2) (NDV ?NODE3)))))
  (PROPOSE-METHOD (?F1) ?F2
    (GOAL-ASSERT '(CONTROLS (NDV GROUND) (NDV ?NODE2)) '(BRV-TO-NDV ?F2))
    (GOAL-ASSERT '(CONTROLS (NDV ?NODE1) (NDV ?NODE3)) '(BRV-TO-NDV ?F2))))

(DEFRULE BRV-TO-NDV-ANALYSIS
        ((?F1 (CONTROLS (NDV GROUND) (NDV ?NODE2)))
         (?F2 (CONTROLS (NDV ?NODE1) (NDV ?NODE3))))
  (ASSERT '(CONTROLS (BRV ?NODE1 ?NODE2) (NDV ?NODE3))
          '(BRV-TO-NDV ?F1 ?F2)))
```

The wire rule, shown in figure 14, is the simplest method of making one node voltage equal to another. It works by making the nodes identical with an ID assertion. The (SATISFIED ?F1) tells the task controller that the goal was achieved without asserting a troublesome

```
(CONTROLS (NDV ?NODE1) (NDV ?NODE2))
```

which the ID assertion would turn into

```
(CONTROLS (NDV ?NODE1) (NDV ?NODE1))
```

which is a circular CONTROLS bug (something cannot CONTROL itself).

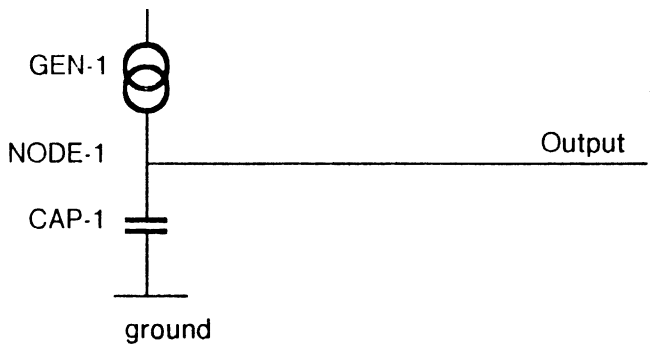
Fig. 14. Wire Rule

```
(DEFRULE WIRE-RULE-ANALYSIS
         ((?F1 (GOAL (CONTROLS (NDV ?NODE1) (NDV ?NODE2))))
          (?F2 (ID ?NODE1 ?NODE2)))
  (ASSERT '(SATISFIED ?F1) '(WIRE ?F2)))

(DEFRULE WIRE-RULE
         ((?F1 (GOAL (CONTROLS (NDV ?NODE1) (NDV ?NODE2)))))
  (PROPOSE-METHOD (?F1) ?F2
    (ASSERT '(ID ?NODE1 ?NODE2)
            '(WIRE-RULE ?F2))))
```

The result of the wire rule is shown in the next figure. This circuit is the one mentioned at the beginning of the chapter. The arithmetic constraints relate the strength of the current generator GEN1, the capacitance of CAP-1, and the desired slope of the ramp M.

Fig. 15. Simple Sweep Circuit



TEK2322

```
(GOAL (CONTROLS (BRV NODE-1 NODE-2) (NDV OUTPUT)))      ;original subgoal

(GOAL (CONTROLS (NDV GROUND) (NDV NODE-2)))
(ID GROUND NODE-2)                                      ;satisfies goal

(GOAL (CONTROLS (NDV NODE-1) (NDV OUTPUT)))
(ID NODE-1 OUTPUT)                                      ;satisfies goal

(CONTROLS (BRV NODE-1 NODE-2) (NDV OUTPUT))             ;satisfies goal
(CONTROLS (BRV OUTPUT GROUND) (NDV OUTPUT))             ;because IDs
(CONTROLS (TC (T1 CAP-1)) (D-DT (BRV OUTPUT GROUND)))

(= (STRENGTH GEN-1) (- (TC (T1 CAP-1))))                ;previous deduction
(= (TC (T1 CAP-1))  (* (CAPACITANCE CAP-1)              ;capacitor
                       (D-DT (BRV OUTPUT GROUND))))
(= (BRV OUTPUT GROUND) (NDV OUTPUT))                    ;definition
(= M (D-DT (NDV OUTPUT)))                               ;original specification

(= (STRENGTH GEN-1) (- (* (CAPACITANCE CAP-1)           ;math
                          M)))
```

2.4 Loose Ends

Designs are not unique because there are several methods to achieve the same goal and those methods produce different circuits. This variety should not be surprising. The system described here does not assume one design is any better than any other. It can be told to produce several designs -- and in the next chapter more designs will be covered. The system does not, however, try to produce a best design.

The ramp generator designed in this chapter is the simplest of many possible circuit designs. The next chapter introduces the feedback rule and uses it to design several other ramp generators. This next chapter also takes up the discussion of the KCL expert that was put off in this chapter.

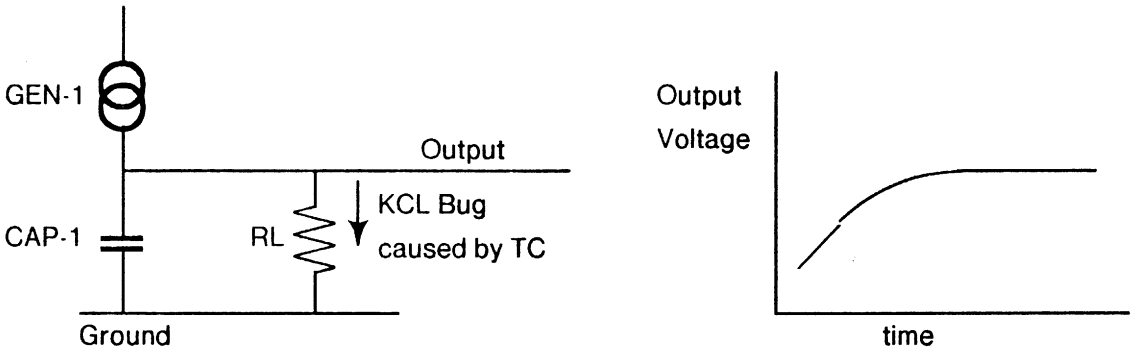
3. KCL and Feedback

3.1 A Design Bug

The ramp generator of the last chapter doesn't work with a resistor (or some other load) connected to its output. This chapter shows that the addition of the resistor violates an assumption of the KCL design rule. The chapter also shows how to design with feedback without violating the KCL design rule assumptions.

Figure 16 shows the simple generator designed in the last chapter with a load resistor connected to its output. The circuit's output voltage is not the desired ramp but a rising exponential as shown in the figure.

Fig. 16. Problems with a Load



The output is different because the load resistor violates an assumption that the designer used to set the current flowing through the capacitor. The designer assumed that all of the current from the current source GEN-1 would flow through capacitor CAP-1, but the load resistor RL steals some of this current. When a component steals another's current, then the design has a KCL bug.

KCL bugs must be fixed or the design won't work. There are four ways to remove KCL bugs: (1) tolerate the connection, (2) cancel the adverse effect, (3) switch out the connection, or (4) abandon the present design and find another. Some connections can be tolerated -- that is they can be left connected because their effect on the design is insignificant. If, for example, the load resistor never stole more than one percent of the constant current generator's current, then the capacitor current would only be in error by one percent. If such an error is acceptable, then the resistor could be left connected.[1] If the error has a significant effect, it may be

1. Some external constraint would have to bound the output voltage of the ramp generator. As specified, the ramp generator's output would grow until the current through the resistor became arbitrarily large. Practically speaking, though, something would be resetting the ramp before the voltage gets too large.

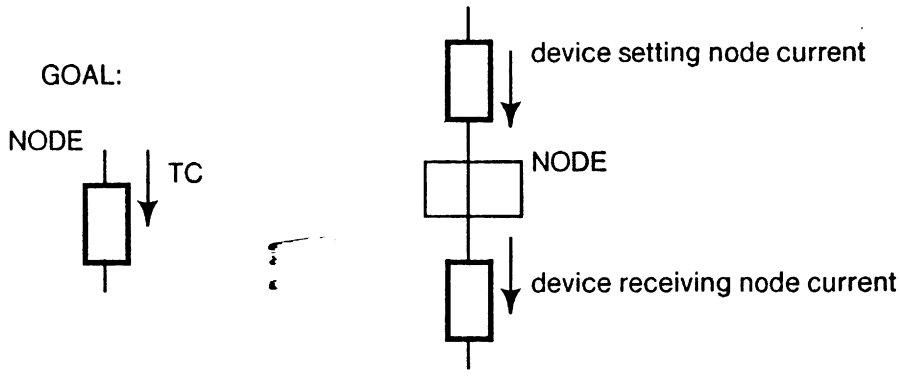
possible to cancel out the error with another connection that supplies the same amount of current that the resistor steals. If a connection cannot be tolerated or canceled, it may be possible to temporarily remove the connection without affecting the operation of the circuit. The chapter on switches will say more about this method of resolving the bug. Finally, the KCL bug may not be fixable and the present design should be abandoned. For example, some resistors cannot be canceled out because their resistance is not known or may change.

3.2 A Closer Look at KCL

What is a KCL bug? How is it found? How is it fixed? A KCL bug occurs when a KCL constraint is used to set a terminal current but another terminal steals some of that current. KCL bugs are found by rules that explicitly look for them. The last section listed four ways that the bug might be resolved; this chapter considers three of them: tolerating connections, canceling connections, and abandoning designs. Another chapter considers switching the connection out. This section will examine KCL more closely.

To answer the question about what a KCL bug is, we first have to examine how KCL is used. KCL is used to set a terminal current of a device. The model that the KCL rule has is that it can set a terminal current by forcing a current into a node; that current will then flow out of the node and into the the desired terminal (figure 17).

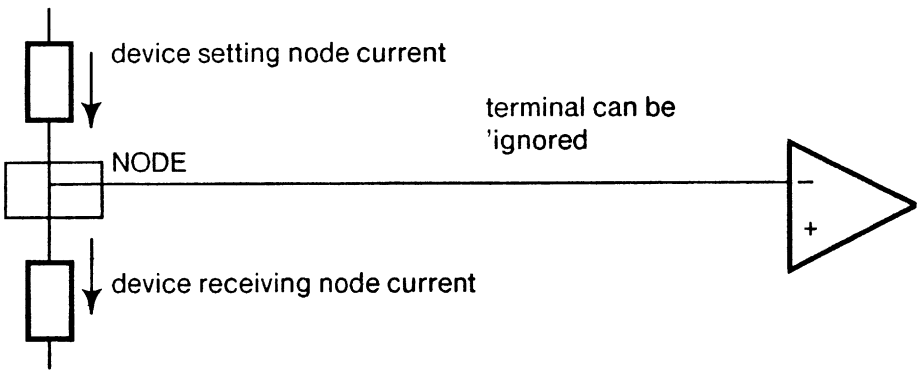
Fig. 17. Setting a Terminal Current
(make nodes line up)



The current flowing out of the node is called the node current or NC. Some other device must supply the node current; in the sweep generator this device was a current generator, but it could have been a resistor or some other component. The design rule assumes that there are only two important terminals

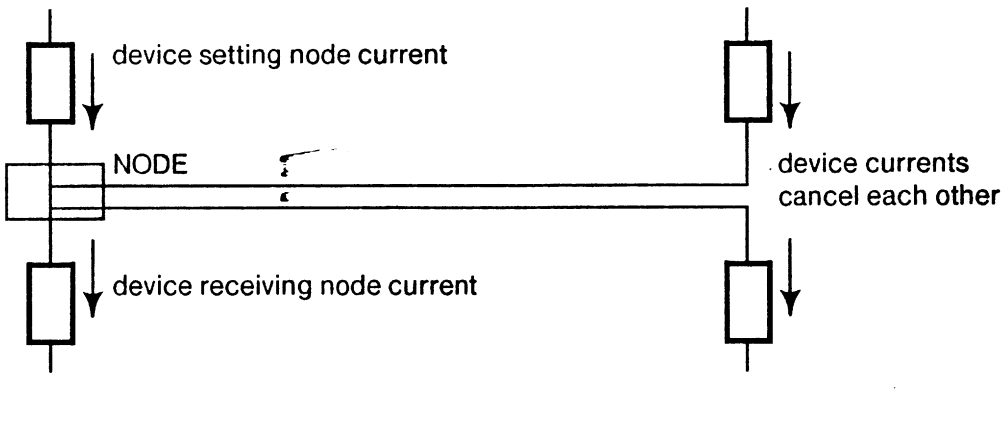
connected to the node where KCL is being used -- the one that supplies the node current and the one that receives it. Other terminals that are connected to the node may cause KCL bugs. Frequently a design will have more than two terminals connected to a node, but the extra terminals can be tolerated because they draw so little current. Operational amplifier inputs, for example, are always tolerated at a node (figure 18).

Fig. 18. A Terminal can be Tolerated



Sometimes there are two extra connections to a node but because one adds the same amount of current that the other takes away, there is no net effect on the node's KCL. The pair of terminals cancel each other out. This circumstance is shown in figure 19.

Fig. 19. Two Terminal Currents Can Cancel Out



The terminal that sets the node current and the terminal that receives the node current are important

terminals to know. The new KCL analysis and design rules are shown in figure 20. These rules are similar to the ones given in the previous chapter except they know about other connections to a node. These other connections are not tolerated until proven otherwise.

Fig. 20. KCL Rules

```
(DEFRULE KCL-ANALYSIS-0
         ((?F1 (CONTROLS ?X (TC ?TERM)))
          (?F2 (CONNECT ?TERM ?NODE)))
   (IF (... circularity test ...)
       (LET ((F3 (ASSUME '(NOT (TOLERATED ?TERM))
                         '(PREMISE))))
            (ASSERT '(CONTROLS (TC ?TERM) (NC ?NODE))
                    '(KCL ?F1 ?F2 ,F3))
            )))

(DEFRULE KCL-ANALYSIS-1
         ((?F1 (CONTROLS (TC ?TERM1) (NC ?NODE)))
          (?F2 (CONNECT ?TERM1 ?NODE))
          (?F3 (CONNECT ?TERM2 ?NODE)))
   (IF (NOT (EQUAL ?TERM1 ?TERM2))
       (LET ((F4 (ASSUME '(NOT (TOLERATED ?TERM2))
                         '(PREMISE))))
            (ASSERT '(CONTROLS (NC ?NODE) (TC ?TERM2))
                    '(KCL ?F1 ?F2 ?F3 ,F4))
            (ASSERT '(= (TC ?TERM1) (- (TC ?TERM2)))
                    '(KCL ?F1 ?F2 ?F3 ,F4)))))

(DEFRULE KCL-DESIGN
         ((?F1 (GOAL (CONTROLS ?X (TC ?TERMINAL))))
          (?F2 (CONNECT ?TERMINAL ?NODE)))
   (PROPOSE-METHOD (?F1 ?F2) ?F3
     (GOAL-ASSERT '(CONTROLS ?X (NC ?NODE))
                  '(KCL-DESIGN ?F2 ?F3))))
```

The previous rules assume that KCL used properly -- one terminal controls the node current and one terminal is controlled by the node current. There are monitors that check whether KCL is used properly and issue a BUG report when it isn't. There are three kinds of KCL bugs: (1) 2 or more terminals CONTROLling a node current; (2) 2 or more terminal currents being CONTROLled by a node current; and (3) a zero impedance terminal connected to a KCL design rule node. The third bug says that KCL cannot be used to control any terminal currents when the node is connected to a power supply. The monitors are shown in figure 21. These KCL bugs don't mean that the circuit violates KCL -- it certainly doesn't. They mean that the designer cannot use KCL to set terminal currents at that node.

Fig. 21. KCL Monitors

```
(DEFRULE KCL-MONITOR-1
         ((?F2 (CONNECT ?TERM1 ?NODE))
          (?F3 (CONTROLS (TC ?TERM1) (NC ?NODE)))
          (?F4 (CONNECT ?TERM2 ?NODE))
          (?F5 (CONTROLS (TC ?TERM2) (NC ?NODE))))
   (IF (NOT (EQUAL ?TERM1 ?TERM2))
       (ASSERT '(BUG KCL ?NODE ?TERM1 ?TERM2)
               '(KCL-MONITOR ?F2 ?F3 ?F4 ?F5))))

(DEFRULE KCL-MONITOR-2
         ((?F2 (CONNECT ?TERM1 ?NODE))
          (?F3 (CONTROLS (NC ?NODE) (TC ?TERM1)))
          (?F4 (CONNECT ?TERM2 ?NODE))
          (?F5 (CONTROLS (NC ?NODE) (TC ?TERM2))))
   (IF (NOT (EQUAL ?TERM1 ?TERM2))
       (ASSERT '(BUG KCL ?NODE ?TERM1 ?TERM2)
               '(KCL-MONITOR ?F2 ?F3 ?F4 ?F5))))

(DEFRULE KCL-MONITOR-3
         ((?F2 (ZERO-IMPEDANCE ?TERM1))
          (?F3 (CONNECT ?TERM1 ?NODE))
          (?F4 (CONNECT ?TERM2 ?NODE))
          (?F5 (CONTROLS (NC ?NODE) (TC ?TERM2))))
   (IF (NOT (EQUAL ?TERM1 ?TERM2))
       (ASSERT '(BUG KCL ?NODE ?TERM1 ?TERM2)
               '(KCL-MONITOR ?F2 ?F3 ?F4 ?F5))))
```

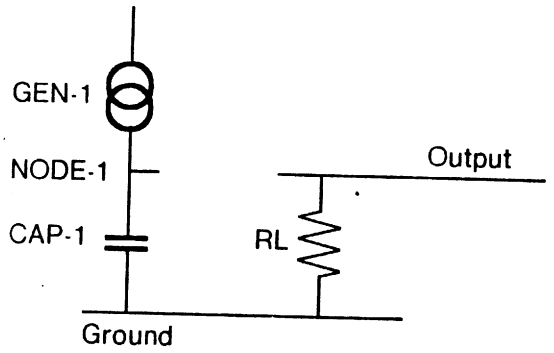
3.3 The Feedback Rule

If the KCL bug cannot be resolved, then the designer must remove the connection and find some other method to achieve the design goal. In the case of the ramp generator with the load resistor, the WIRE rule was trying to achieve the goal:

```
(GOAL (CONTROLS (NDV NODE-1) (NDV OUTPUT)))
```

The WIRE rule fails to achieve this goal because of the KCL bug. Backtracking from the failure of the WIRE rule leaves the design problem as shown in figure 22.

Fig. 22. Old Problem
(GOAL (CONTROLS (NDV NODE-1) (NDV OUTPUT)))

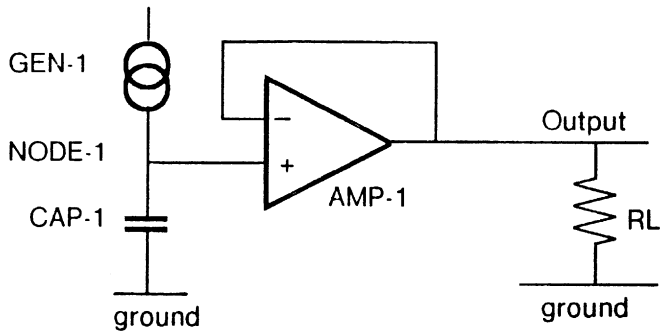


TEK3310

The WIRE rule failed because the load resistor stole current that was supposed to go to the capacitor. Somehow the load resistor must be prevented from stealing the current. A human circuit designer would say that NODE-1 must be isolated from OUTPUT and would suggest using a buffer amplifier (figure 23). While the buffer amplifier also makes a connection to NODE-1 and thus causes a KCL bug, that bug can be resolved easily because an ideal amplifier input draws no input current and therefore does not disturb the KCL constraint at NODE-1. The amplifier also supplies the load resistor with all the current it needs. The design system makes these deductions with 2 rules, one for negative feedback and the other for operational amplifiers. The two steps allow additional flexibility that will be used in later designs. The feedback rule is shown in figure 24.

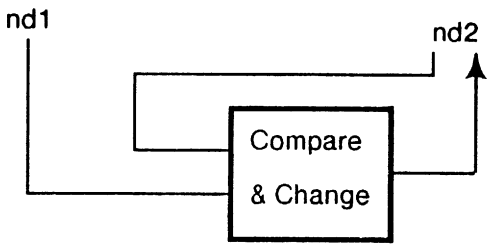
The feedback rule solves the same problem as the WIRE rule, but it isolates the two nodes involved. Feedback looks at the voltage difference between the two nodes (that is, the branch voltage) and then adjusts

Fig. 23. Ramp Generator with Buffer Amplifier



TEK3320

Fig. 24. Feedback Rule



TEK3330

```
(DEFRULE FEEDBACK-DESIGN
        ((?F1 (GOAL (CONTROLS (NDV ?ND1) (NDV ?ND2)))))
   (PROPOSE-METHOD (?F1) ?F2
    (GOAL-ASSERT '(CONTROLS (BRV ?ND1 ?ND2) (NDV ?ND2))
                 '(FEEDBACK ?F2))))

(DEFRULE FEEDBACK-ANALYSIS
         ((?F1 (CONTROLS (BRV ?ND1 ?ND2) (NDV ?ND2))))
  (ASSERT '(CONTROLS (NDV ?ND1) (NDV ?ND2))
          '(FEEDBACK ?F1)))
```

the second node voltage to drive the voltage difference to zero.[1]

The feedback rule produces a goal for a branch voltage to control a node voltage; another

1. The feedback rule assumes an infinite feedback loop gain. An extended FEEDBACK-ANALYSIS rule should calculate the loop gain and check that it is large enough to keep the errors small. Such a check is difficult because the loop gain is frequency dependent. In the interest of simplicity this test has been ignored. A simple but not complete test would check if the CONTROLS relation depended upon an operational amplifier.

component, an operational amplifier, is needed to achieve that goal. The operational amplifier rules (figure 25) provide the match for this CONTROLS goal.

Fig. 25. Operational Amplifier Rules

```
(DEFRULE OPERATIONAL-AMPLIFIER-ANALYSIS
         ((?F1 (CIRCUIT-PART OP-AMP ?AMP))
          (?F2 (CONNECT (I+  ?AMP) ?ND1))
          (?F3 (CONNECT (I-  ?AMP) ?ND2))
          (?F4 (CONNECT (OUT ?AMP) ?ND3)))
  (ASSERT '(CONTROLS (BRV ?ND1 ?ND2) (NDV ?ND3))
          '(OP-AMP-VIC ?F1 ?F2 ?F3 ?F4))
  (ASSERT '(= (NDV ?ND3) (* INF (BRV ?ND1 ?ND2)))
          '(OP-AMP-VIC ?F1 ?F2 ?F3 ?F4))
  (ASSERT '(= (TC (I+ ?AMP)) 0)
          '(OP-AMP-VIC ?F1))
  (ASSERT '(= (TC (I- ?AMP)) 0)
          '(OP-AMP-VIC ?F1))
  (ASSERT '(ZERO-IMPEDANCE (OUT ?AMP))
          '(OP-AMP-VIC ?F1)))

(DEFRULE OPERATIONAL-AMPLIFIER-DESIGN
         ((?F1 (GOAL (CONTROLS (BRV ?ND1 ?ND2) (NDV ?ND3)))))
  (PROPOSE-METHOD (?F1) ?F2
    (LET ((AMP (GENPREFIX 'AMP)))
         (ASSERT '(CIRCUIT-PART OP-AMP ,AMP) '(OP-AMP ?F2))
         (ASSERT '(CONNECT (I+  ,AMP) ?ND1)  '(OP-AMP ?F2))
         (ASSERT '(CONNECT (I-  ,AMP) ?ND2)  '(OP-AMP ?F2))
         (ASSERT '(CONNECT (OUT ,AMP) ?ND3)  '(OP-AMP ?F2)))))
```

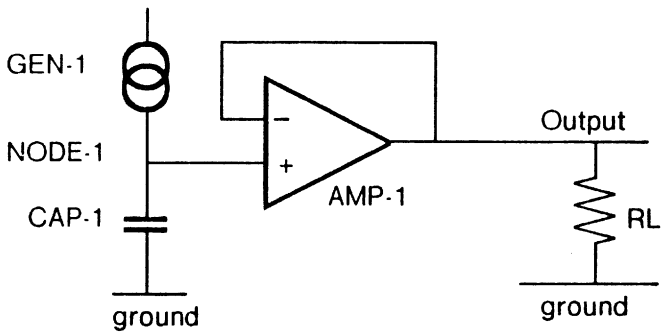
Figure 26 shows what happens when the feedback and operational amplifier rules work on the first goal shown in the figure. The feedback rule first transforms the goal into another one. The operational amplifier design rule then runs; the pattern variables and circuit nodes are paired as (?ND1, NODE-1), (?ND2, OUTPUT), and (?ND3, OUTPUT). In this case the operational amplifier rule is more general than it needs to be because ?ND2 and ?ND3 are paired with the same node. With this special pairing, the rule will make a voltage follower.[1] The design rule also pairs nodes with terminals: (NODE-1, I+), (OUTPUT, I-), and (OUTPUT, OUT). After all the connections are made, the circuit looks as in figure 26. The operational amplifier satisfies the subgoal for feedback, feedback satisfies the goal for controlling a node voltage, and together they satisfy the goal for making the capacitor's branch voltage control the output node voltage. This new design isolates NODE-1 (where KCL sets the capacitor CAP-1's current) from the output node (where

---

1. It is a feature of the representation that several special case circuits fall out general purpose ideas.

the load resistor draws some current) and therefore doesn't violate the protect-KCL monitor.

Fig. 26. Sweep with Follower Amplifier



TEK3335

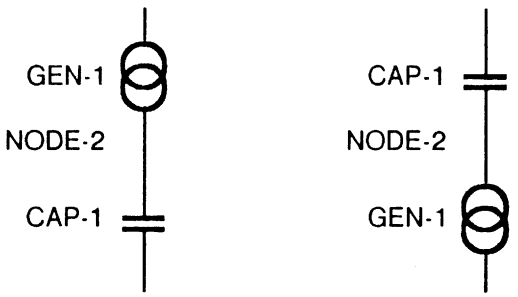
```
(GOAL (CONTROLS (NDV NODE-1)              ;subgoal wire couldn't do
                (NDV OUTPUT)))
(GOAL (CONTROLS (BRV NODE-1 OUTPUT)       ;feedback reformulation
                (NDV OUTPUT)))

(CIRCUIT-PART OP-AMP AMP-1)               ;op-amp design
(CONNECT (I+  AMP-1) NODE-1)
(CONNECT (I-  AMP-1) OUTPUT)
(CONNECT (OUT AMP-1) OUTPUT)
(= (NDV OUTPUT) (* INF (BRV NODE-1 OUTPUT)))
(= (TC (I+ ?AMP)) 0)
(= (TC (I- ?AMP)) 0)
(ZERO-IMPEDANCE (OUT AMP-1))
                                          ;op-amp analyze
(CONTROLS (BRV NODE-1 OUTPUT)             ;satisfies goal
          (NDV OUTPUT))
                                          ;feedback analyze
(CONTROLS (NDV NODE-1) (NDV OUTPUT))      ;satisfies goal
```

3.4 Another Design

Figure 27 shows the two ways that a current generator can be connected to control the capacitor current of a ramp generator. The second configuration is a consequence of an earlier terminal current rule stating that (TC (T1 ?X)) could be controlled by controlling (TC (T2 ?X)). Other circuit designs come from the second configuration; these designs will use the feedback rule.

Fig. 27. Setting Capacitor Current



Setting the current through the capacitor is only one of the design goals of making a ramp generator. The second is turning the capacitor's branch voltage into a node voltage. In the previous ramp generators that was done with the BRV-TO-NDV rule that created two subgoals:

```
(GOAL (CONTROLS (BRV NODE-1 NODE-2) (NDV OUTPUT)))
 leads to ...
    (GOAL (CONTROLS (NDV NODE-1) (NDV OUTPUT)))
    (GOAL (CONTROLS (NDV GROUND) (NDV NODE-2)))
```

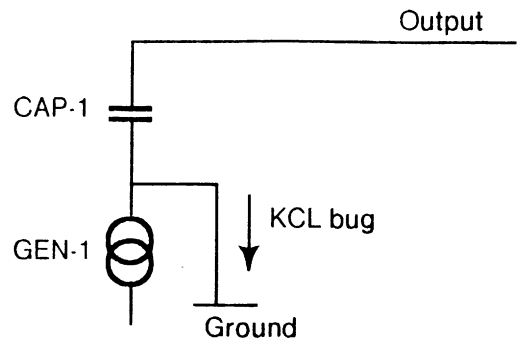
In the first ramp generator (the one without an amplifier) both of these subgoals were accomplished by wiring the nodes together. While this technique works for the subgoal to control the output node voltage when the current generator is connected to (T2 CAP-1), it doesn't work for the node that ground is to control (figure 28). The problem is another instance of the KCL bug -- the connection upsets KCL at the node and prevents the current generator from controlling the capacitor current. There is no way to fix this bug, so the wire rule must be discarded.

The feedback rule can be used instead. A direct application of the FEEDBACK rule creates a new CONTROLS goal:

```
(GOAL (CONTROLS (BRV GROUND NODE-2) (NDV NODE-2)))
```

Unlike the last application of the FEEDBACK rule, an operational amplifier cannot achieve this goal directly because it also suffers from the KCL bug (figure 29). The output of the operational amplifier would steal

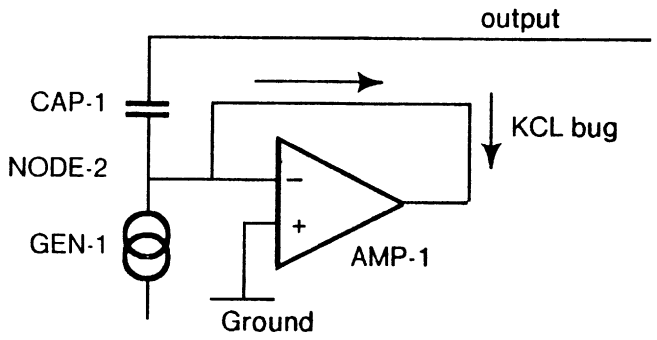
Fig. 28. Wire causes KCL Bug



TEK3437

current from the current generator and prevent it from controlling the capacitor current.

Fig. 29. Using an Amplifier Fails



TEK3438

Though the operational amplifier cannot be used directly, the FEEDBACK rule is not abandoned because there is another method of achieving

```
(GOAL (CONTROLS (BRV GROUND NODE-2) (NDV NODE-2)))
```

This other method uses the transitivity of the CONTROLS relation to turn the present goal into an equivalent goal. The idea is that there might be another way to control (NDV NODE-2) without connecting directly to NODE-2. The rule is shown in figure 30.

This rule needs an intermediate signal (?Y) to control. A human designer knows that changing the voltage on one end of a capacitor will probably change the voltage on the other end. <de Kleer-5> called this property the KVL-Heuristic. If this property is assumed for the moment, then there would be an assertion:

Fig. 30. Transitivity Design

```
(DEFRULE TRANSITIVITY-DESIGN
        ((?F1 (GOAL (CONTROLS ?X ?Z)))
         (?F2 (CONTROLS ?Y ?Z)))
  (IF (... circularity test ...)
      (PROPOSE-METHOD (?F1 ?F2) ?F3
        (GOAL-ASSERT '(CONTROLS ?X ?Y) '(TRANSITIVITY ?F3)))))
```

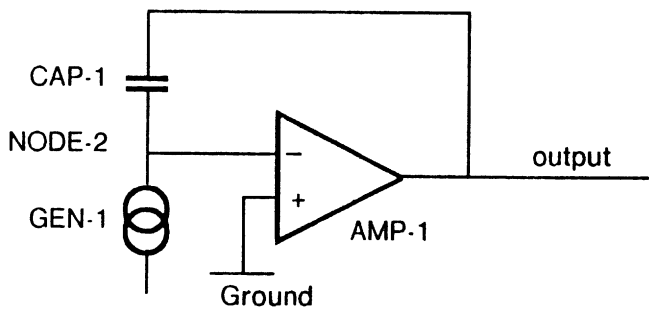
```
(CONTROLS (NDV OUTPUT) (NDV NODE-2))
```

This assertion in conjunction with the transitivity design rule produce the goal in figure 31. An operational amplifier will satisfy this goal; the resulting circuit is also shown in the figure.

A key step in the design of this circuit is noticing that the node voltage of OUTPUT controls that of NODE-2. The rule in figure 32 is responsible for noticing this relation. If there is a goal to control the node

---

Fig. 31. Miller Integrator



TEK3440

```
(GOAL (CONTROLS (NDV GROUND) (NDV NODE-2)))          ;wire fails on this goal
(GOAL (CONTROLS (BRV GROUND NODE-2) (NDV NODE-2)))   ;feedback design
(CONTROLS (NDV OUTPUT) (NDV NODE-2))                 ;brv-to-ndv
(GOAL (CONTROLS (BRV GROUND NODE-2) (NDV OUTPUT)))   ;KVL heuristic

(CIRCUIT-PART OP-AMP AMP-1)                          ;op-amp
(CONNECT (I+  AMP-1) GROUND)
(CONNECT (I-  AMP-1) NODE-2)
(CONNECT (OUT AMP-1) OUTPUT)

(CONTROLS (BRV GROUND NODE-2) (NDV OUTPUT))          ;op-amp
(GOAL (CONTROLS (BRV GROUND NODE-2) (NDV NODE-2)))   ;transitivity
(GOAL (CONTROLS (NDV GROUND) (NDV NODE-2)))          ;feedback succeeds
```

voltage at one end of a capacitor, then this rule suggests controlling the other end.

Fig. 32. Johan's Heuristic

```
(DEFRULE JOHANS-HEURISTIC-2
         ((?F1 (CIRCUIT-PART CAPACITOR ?COMP-NAME))
          (?F2 (CONNECT (T1 ?COMP-NAME) ?NODE1))
          (?F3 (CONNECT (T2 ?COMP-NAME) ?NODE2))
          (?F4 (GOAL (CONTROLS ?X (NDV ?NODE2)))))
  (IF (... circularity test ...)
      (ASSUME '(CONTROLS (NDV ?NODE1) (NDV ?NODE2))
              '(JOHANS-HEURISTIC ?F1 ?F2 ?F3 ?F4))))
```

3.5 Still More Generators

The previous ramp generators used a constant current source to set the current flowing through the capacitor. A resistor can be used instead. The resistor rule, which is similar to the current generator rule, is shown in figure 33.

The resistor rule creates the partial ramp generator design shown in figure 34. To make the resistor's branch current (BRI) constant, the design must impose a constant branch voltage across the resistor. A battery can be used; the following figure show the battery rules.

Using the WIRE rule twice would connect the battery directly across the resistor but also causes a KCL bug at the junction of the resistor and capacitor (figure 36). The fix to this bug is not to use the wire rule but to use feedback instead -- as we've done twice before. The bootstrap ramp generator is the resulting circuit.

An interesting point about the ramp circuit is that now there are three different nodes that satisfy the design goal -- the voltage at the capacitor, the voltage at the operational amplifier, and the voltage at the other end of the battery. While any one of these three voltages meets the goal, there should be some preference placed on the last two because they are low impedance outputs and are immune to the loading problems previously discussed. Nothing has been done to consider this preference.

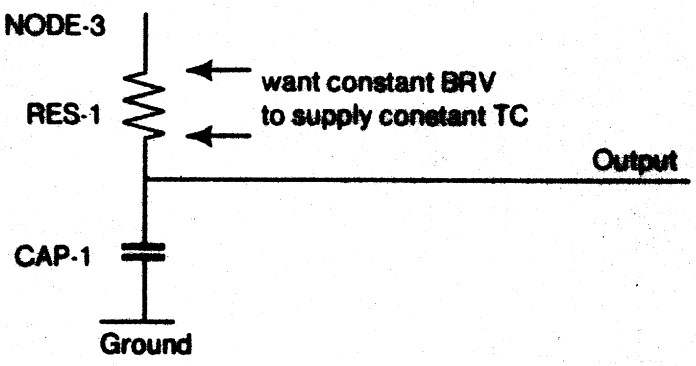
Fig. 33. Resistor Rules

```
(DEFRULE RESISTOR-ANALYSIS-0
         ((?F1 (CIRCUIT-PART RESISTOR ?RES))
          (?F2 (CONNECT (T1 ?RES) ?NODE1))
          (?F3 (CONNECT (T2 ?RES) ?NODE2)))
  (ASSERT '(= (TC (T1 ?RES)) (// (BRV ?NODE1 ?NODE2)
                                 (RESISTANCE ?RES)))
          '(RESISTOR ?F1 ?F2 ?F3)))

(DEFRULE RESISTOR-ANALYSIS-1
         ((?F1 (CIRCUIT-PART RESISTOR ?RES))
          (?F2 (CONNECT (T1 ?RES) ?NODE1))
          (?F3 (CONNECT (T2 ?RES) ?NODE2))
          (?F4 (CONTROLS ?X (BRV ?NODE1 ?NODE2))))
  (IF (... circularity test ...)
      (ASSERT '(CONTROLS (BRV ?NODE1 ?NODE2) (TC (T1 ?RES)))
              '(RESISTOR ?F1 ?F2 ?F3 ?F4))))

(DEFRULE RESISTOR-DESIGN-1
         ((?F1 (GOAL (CONTROLS ?X (NC ?NODE)))))
  (PROPOSE-METHOD (?F1) ?F2
    (LET ((RES (GENPREFIX 'RES)))
         (ASSERT '(CONNECT (T1 ,RES) ?NODE)
                 '(RESISTOR ?F2))
         (ASSERT '(CONNECT (T2 ,RES) [(CONNECT (T2 ,RES) *)])
                 '(RESISTOR ?F2))
         (ASSERT '(CIRCUIT-PART RESISTOR ,RES)
                 '(RESISTOR ?F2))
         (GOAL-ASSERT '(CONTROLS ?X (BRV ?NODE [(CONNECT (T2 ,RES) *)]))
                      '(RESISTOR ?F2)))))
```

Fig. 34. Resistor Current Source

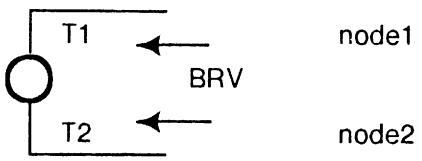


TEK3610

```
(CONNECT (T1 RES-1) NODE-1)
(CONNECT (T2 RES-1) NODE-3)
(CIRCUIT-PART RESISTOR RES-1)
(GOAL (CONTROLS NIL (BRV NODE-1 NODE-3)))

(= (TC (T1 RES-1)) (// (BRV NODE-1 NODE-3) (RESISTANCE RES-1)))
```

Fig. 35. Battery Rules

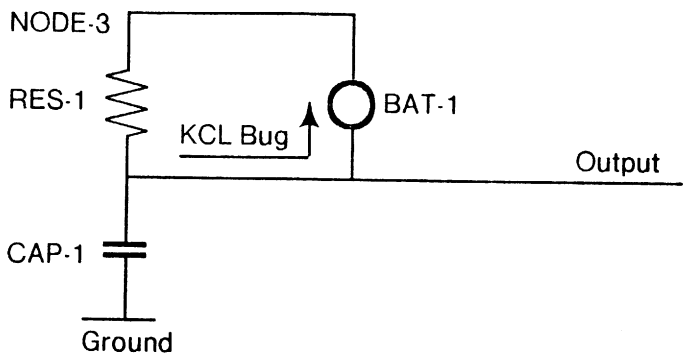


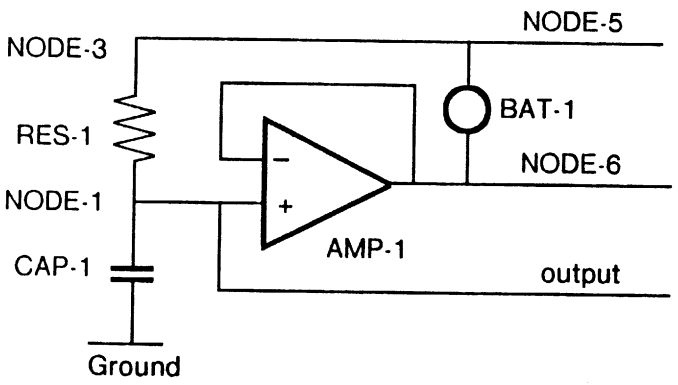
TEK3520

```
(DEFRULE BATTERY-ANALYSIS
         ((?F1 (CIRCUIT-PART BATTERY ?BAT))
          (?F2 (CONNECT (T1 ?BAT) ?NODE1))
          (?F3 (CONNECT (T2 ?BAT) ?NODE2)))
  (ASSERT '(CONTROLS NIL (BRV ?NODE1 ?NODE2))
          '(BATTERY ?F1 ?F2 ?F3))
  (ASSERT '(= (BRV ?NODE1 ?NODE2) (STRENGTH ?BAT))
          '(BATTERY ?F1 ?F2 ?F3)))

(DEFRULE BATTERY-DESIGN
         ((?F1 (GOAL (CONTROLS NIL (BRV ?NODE1 ?NODE2)))))
   (PROPOSE-METHOD (?F1) ?F2
     (LET ((BAT (GENPREFIX 'BAT)))
        (GOAL-ASSERT '(CONTROLS (NDV ?NODE2) (NDV [(CONNECT (T2 ,BAT) *)]))
                     '(BATTERY ?F2))
        (GOAL-ASSERT '(CONTROLS (NDV [(CONNECT (T1 ,BAT) *)]) (NDV ?NODE1))
                     '(BATTERY ?F2)))))
```

Fig. 36. Bootstrap Circuit





TEK3530

```
(GOAL (CONTROLS NIL (BRV NODE-3 NODE-1)))          ;from resistor rule
(CIRCUIT-PART BATTERY BAT-1)                       ;battery design
(GOAL (CONTROLS (NDV NODE-1) (NDV NODE-6)))
(GOAL (CONTROLS (NDV NODE-5) (NDV NODE-3)))
(ID NODE-3 NODE-5)                                 ;wire
(GOAL (CONTROLS (BRV NODE-1 NODE-6) (NDV NODE-6))) ;feedback design
(CIRCUIT-PART OP-AMP AMP-1)                        ;op-amp
(CONTROLS (BRV NODE-1 NODE-6) (NDV NODE-6))
(CONTROLS (NDV NODE-1) (NDV NODE-6))               ;feedback analysis
(CONTROLS NIL (BRV NODE-3 NODE-1))                 ;
```

3.6 Summary

This chapter presented several common ramp generators that are known to human designers by names such as Miller integrators and bootstrap generators. each started with the fundamental idea of forcing a constant current through a capacitor, but used different methods of providing that current and transforming the capacitor's branch voltage into an output node voltage. The current was supplied to t1 or t2 of the capacitor by either a current generator or by a resistor. Wires and feedback set node voltages. The choice of these different methods is often dictated by design bugs that are uncovered as the design progresses. Wires, for example, were acceptable if there was no load, but feedback had to be used if a load was present.

The system generates many designs; it is not selective about these designs -- as long as they conform to the design specifications. The circuits that have been discussed are not designs that have been stored and will be called up when needed, but instead result from the application of fundamental rules about capacitors, resistors, operational amplifiers, and feedback. Using rules rather than specific designs allows the system to design not only these common circuits but also variations on them. These rules and others will design some variations in the next chapter.

4. Discrete State

4.1 Introduction

Some circuits have several distinct modes or behaviors. A practical ramp generator, for example, might have a switch that selects two different output slopes. These different modes are the circuit's discrete states. A circuit may also possess continuous state, for example, in the voltage on a capacitor. The methods of the past couple chapters design circuits with continuous state. This chapter considers some methods that are needed to design circuits with discrete state. In general, discrete state designs need switches and the design rules will discover where and why a switch is needed by finding KCL and KVL bugs.

To keep the design of circuits with discrete state as simple as possible, the different discrete states are assumed to be independent. This assumption lets the designer use the design rules of the previous chapters to design a circuit for each of the discrete states. If the assumption holds, the complete circuit is just the combination of these individual designs plus a few switches to keep the different designs from disturbing each other. Unfortunately, the assumption of state independence is not always valid because capacitors carry continuous state information across discrete state boundaries. This chapter shows examples of both independent and dependent discrete state designs.

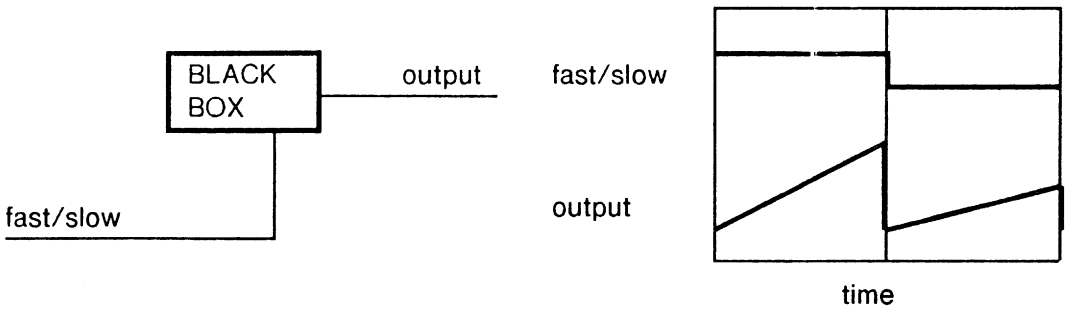
4.2 Design of Independent States

This first section will consider independent discrete states; the next section will consider dependent ones. Independence means that each state does not affect any other state. A dual rate ramp generator will be used as an example. It has two discrete states, (FAST) and (SLOW); the generator's output slope is different during each state. A block diagram and the specifications for this circuit are shown in figure 37. The specifications for the dual rate ramp generator are different from the single rate generator in two respects; both of these differences concern discrete state. The first difference is a COND statement in the specifications to denote conditional values. The second difference is the appearance of situation tags <McCarthy> on assertions to signify when they are valid.

The COND statement is similar to the ordinary LISP COND with the exception of how the condition tests are interpreted. Here they are not evaluated but just refer to the state (or situation). The goal assertion means that during the (FAST) state, NIL controls the slope of the output and during the (SLOW) state, NIL controls the slope of the output. The reason for this strange construction is that the state of the circuit is changed (hence the COND), but in each state no signal controls the output slope (hence the NILs).

The situation tag describes in what situation an assertion holds. A situation is almost the same as a discrete state except that situations sometimes refer to finer divisions of states. In the next section there will be a situation (INITIAL SWEEPING) that refers to the initial instant of the (SWEEPING) state. Rules must be aware of situation tags and only make their deductions if the situation tags of the antecedents are compatible. Situation tags were not shown in the previous rules because only one situation was relevant then.

Fig. 37. Dual Rate Ramp Generator



TEK4210

```
(GOAL ((CONTROLS (COND ((FAST) NIL)
                       ((SLOW) NIL))
                 (D-DT (NDV OUTPUT)))
       NIL))    ;situation tag

((= (COND ((FAST) M1)
          ((SLOW) M2))
    (D-DT (NDV OUTPUT)))
 NIL)   ;situation tag
```

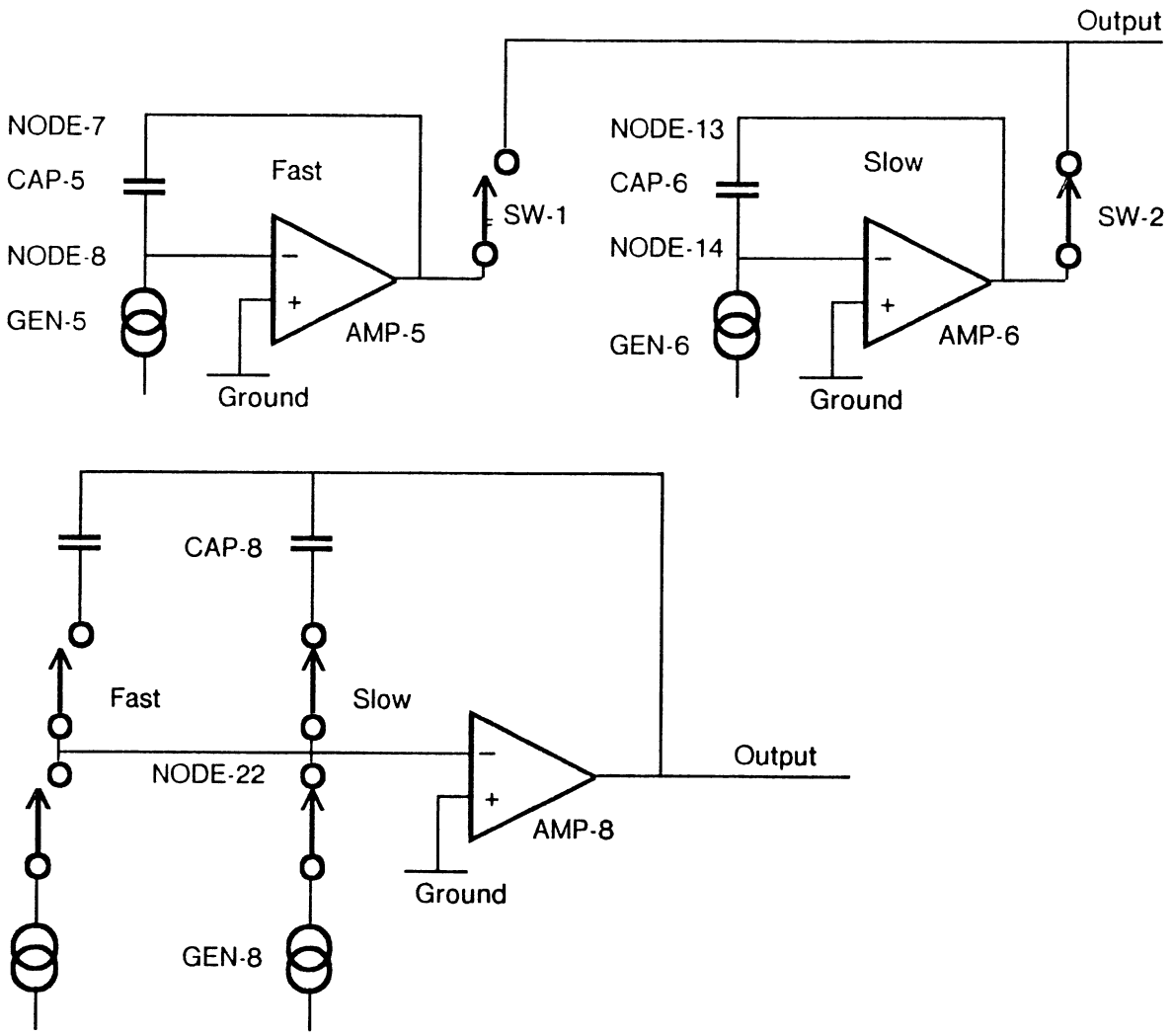
COND statements and situation tags are related. A COND statement can be specialized to a particular situation by replacing the COND with the consequent clause for the particular situation. Alternatively, some assertions true in different situations can be generalized to one COND statement. This property will be used in the SPLIT-STATES design rule to simplify goals that involve more than one situation.

Doing this specialization at different stages in the design of a circuit can give rise to different circuits. Two simple designs of the dual rate ramp generator are shown in figure 38. The first design is developed by immediately specializing the COND and the second by specializing the COND later in the design. Each of these designs will be discussed.

For the first design the SPLIT-STATES rule notices the COND in the design specification and specializes the original goal containing the COND into two subgoals, one for each clause (see figure 39). These subgoals are made by literally substituting the consequent of a clause for the original COND. The situation tags of these new subgoals specify that they only apply during particular situations and not in general. The most general situation tag is NIL (the assertion is true all the time); when a COND is specialized, the situation tag of the component is made by appending the condition to the front of the situation tag for the assertion containing the COND.

The new subgoals that are created by SPLIT-STATES, which are identical to previous goals for designing ramp generators except for the situation tags, are then achieved using the previous methods for

Fig. 38. Two Ramp Generator Designs



TEK4220

Fig. 39. Result of SPLIT-STATES

```
(GOAL ((CONTROLS (COND ((FAST) NIL)
                       ((SLOW) NIL))
                 (D-DT (NDV OUTPUT)))
        NIL))   ;original situation tag

((= (COND ((FAST) M1)
          ((SLOW) M2))
    (D-DT (NDV OUTPUT)))
  NIL)

become

(GOAL ((CONTROLS NIL (D-DT (NDV OUTPUT)))
       (FAST))) ;specialized situation tag
(GOAL ((CONTROLS NIL (D-DT (NDV OUTPUT)))
       (SLOW))) ;specialized situation tag

((= M1 (D-DT (NDV OUTPUT)))
 (FAST))
((= M2 (D-DT (NDV OUTPUT)))
 (SLOW))
```

---

designing ramp generators. Any of the circuits considered so far could be used -- the constant current, Miller integrator, or the bootstrap. Let's use the Miller integrator. The designs for the (FAST) and (SLOW) states and some important assertions about the designs are shown in the next two figures. Situation tags have been included in the figures because they will be important when the two designs are merged. Some assertions, such as CIRCUIT-PART, are not given situation tags because they are not affected by state changes. Capacitor CAP-5 doesn't become something different when the state changes from (FAST) to (SLOW). Though the deductions could be different for the two states (one state could use a Miller circuit and the other a bootstrap circuit), here they are chosen to use the same circuit.

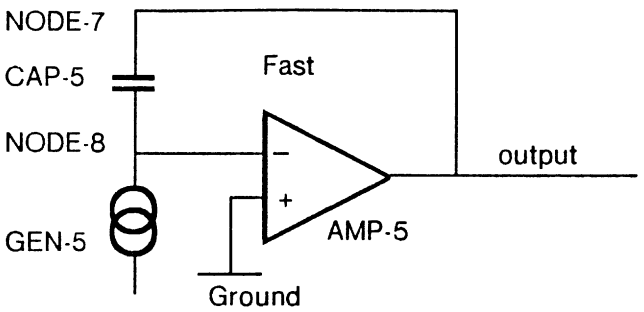
The designs for the (FAST) and (SLOW) states will work in isolation from each other; they must now be merged to achieve the design goal. The simple approach assumes that all the connections are actually present during both states; this brash assumption implies that the complete circuit looks as in figure 42.

This combined circuit is almost right except for a KVL bug caused by two operational amplifier outputs trying to control the same output node voltage. The KVL bug is analogous to the KCL bug but instead of two signals trying to control a node current, there are two signals trying to control a node voltage. The guilty parties are:

```
((CONTROLS (BRV OUTPUT NODE-8)  (NDV OUTPUT)) (FAST))
((CONTROLS (BRV OUTPUT NODE-14) (NDV OUTPUT)) (SLOW))
```

Switches can fix KVL bugs when the CONTROLS constraints causing the bugs don't have to be true during

Fig. 40. Fast Generator



TEK4230

```
(CIRCUIT-PART CAPACITOR          CAP-5)
(CIRCUIT-PART CURRENT-GENERATOR GEN-5)
(CIRCUIT-PART OP-AMP             AMP-5)

((ID NODE-7 OUTPUT)                                  (FAST)) ;wire
((CONTROLS (BRV GROUND NODE-8) (NDV NODE-8))         (FAST)) ;feedback
((CONTROLS (NDV GROUND)        (NDV NODE-8))         (FAST)) ;brv-to-ndv
((CONTROLS (BRV NODE-7 NODE-8) (NDV OUTPUT))         (FAST)) ;capacitor

((CONNECT (T1  CAP-5) OUTPUT) (FAST))
((CONNECT (T2  CAP-5) NODE-8) (FAST))
((CONNECT (I+  AMP-5) GROUND) (FAST))
((CONNECT (I-  AMP-5) NODE-8) (FAST))
((CONNECT (OUT AMP-5) OUTPUT) (FAST))
((CONNECT (T1  GEN-5) NODE-8) (FAST))

((= (TC (T1 GEN-5))  (TC (T1 CAP-5)))            (FAST))
((= (STRENGTH GEN-5) (* (CAPACITANCE CAP-5) M1)) (FAST))
```

---

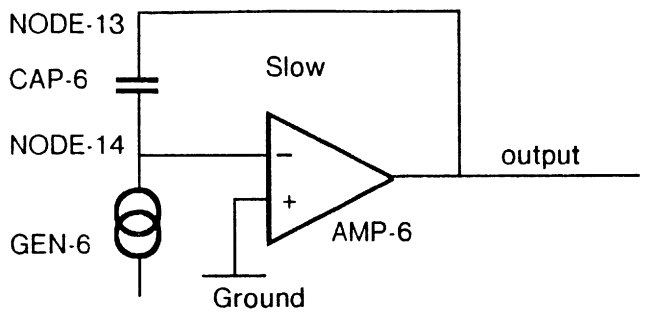
the same state. For the two CONTROLS constraints above, one must be true in the (FAST) state and one must be true in the (SLOW) state, but they don't both have to be true in either state.

A KVL bug signals trouble at a node; the first thing to do is find all the components connected to the node. These connections are:

```
((CONNECT (OUT AMP-5) OUTPUT)   (FAST))
((CONNECT (T1  CAP-5) OUTPUT)   (FAST))
((CONNECT (OUT AMP-6) OUTPUT)   (SLOW))
((CONNECT (T1  CAP-6) OUTPUT)   (SLOW))
```

To remove the bug, each of these connections are considered from the view point of each state. During the (FAST) state, those connection with the (FAST) situation tag must be present. The output of

Fig. 41. Slow Generator



TEK4240

```
(CIRCUIT-PART CAPACITOR          CAP-6)
(CIRCUIT-PART CURRENT-GENERATOR GEN-6)
(CIRCUIT-PART OP-AMP             AMP-6)

((ID NODE-13 OUTPUT)                                    (FAST)) ;wire
((CONTROLS (BRV GROUND NODE-14)  (NDV NODE-14)) (FAST)) ;feedback
((CONTROLS (NDV GROUND)          (NDV NODE-14)) (FAST)) ;brv-to-ndv
((CONTROLS (BRV NODE-13 NODE-14) (NDV OUTPUT))  (FAST)) ;capacitor

((CONNECT (T1  CAP-6) OUTPUT)   (FAST))
((CONNECT (T2  CAP-6) NODE-14)  (FAST))
((CONNECT (I+  AMP-6) GROUND)   (FAST))
((CONNECT (I-  AMP-6) NODE-14)  (FAST))
((CONNECT (OUT AMP-6) OUTPUT)   (FAST))
((CONNECT (T1  GEN-6) NODE-14)  (FAST))

((= (TC (T1 GEN-6))  (TC (T1 CAP-6)))            (FAST))
((= (STRENGTH GEN-6) (* (CAPACITANCE CAP-6) M1)) (FAST))
```

AMP-6 cannot be connected, though, because it causes a KVL bug. A switch will prevent the bug; all those connections that must be made during the (FAST) state are left connected to the OUTPUT. All the connections that are needed only when AMP-6 controls the output are then grouped together and put on the far side of a switch (SW-2 in the figure). Then all the connections are considered from the (SLOW) state. Switch SW-2 isolates those connections to the OUTPUT that only had to be made during the (FAST) state. Switches don't have to be used all the time; shortly there will be an example where a switch is needed for one state, but the connection can be tolerated in the other.

Another design (figure 44) for the dual ramp generator switches components instead of complete sweep circuits. The switches in this design come about because conditional values are calculated for devices that can only have one value. If the SPLIT-STATES rule is not applied immediately to the dual ramp design problem, then the a complete ramp generator could be deduced; the deductions would be similar to those for

Fig. 42. Output KVL Bug

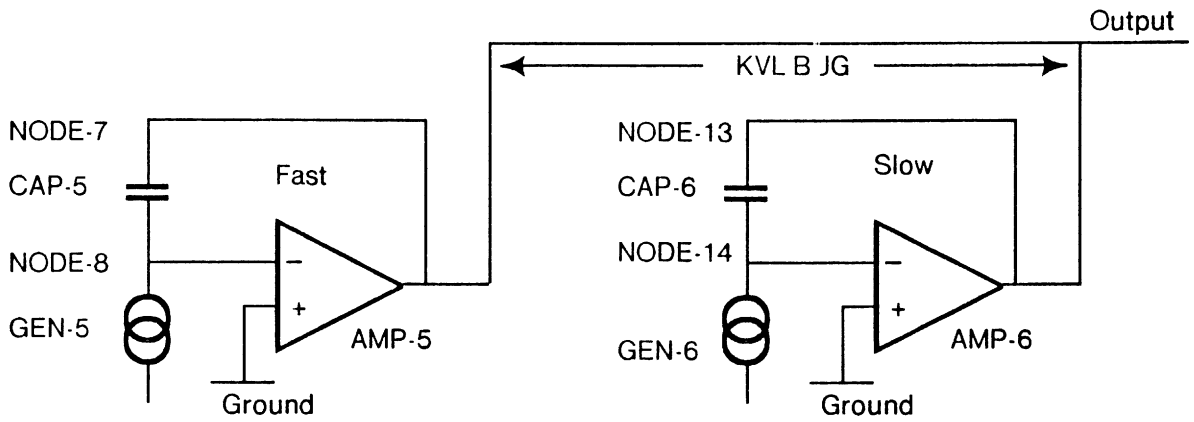
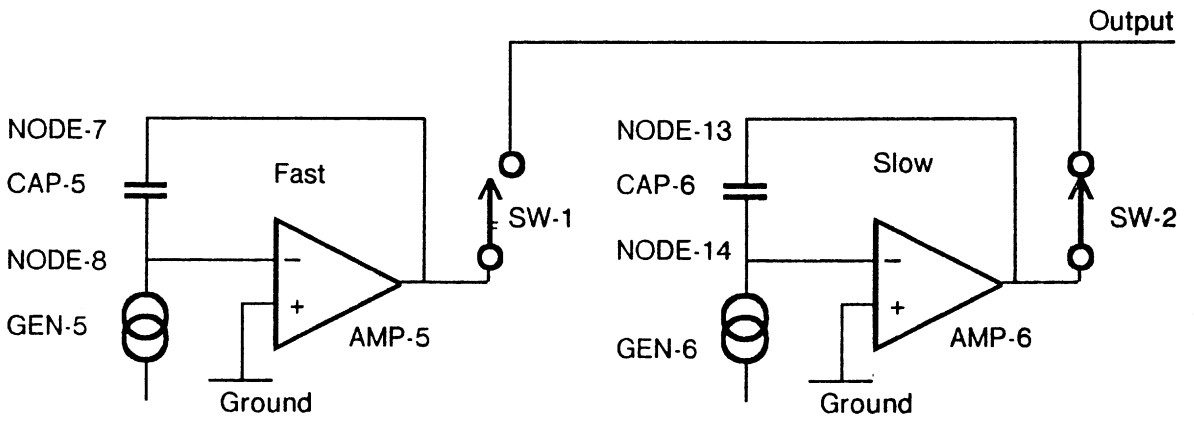


Fig. 43. Fixing a KVL Bug with a Switch



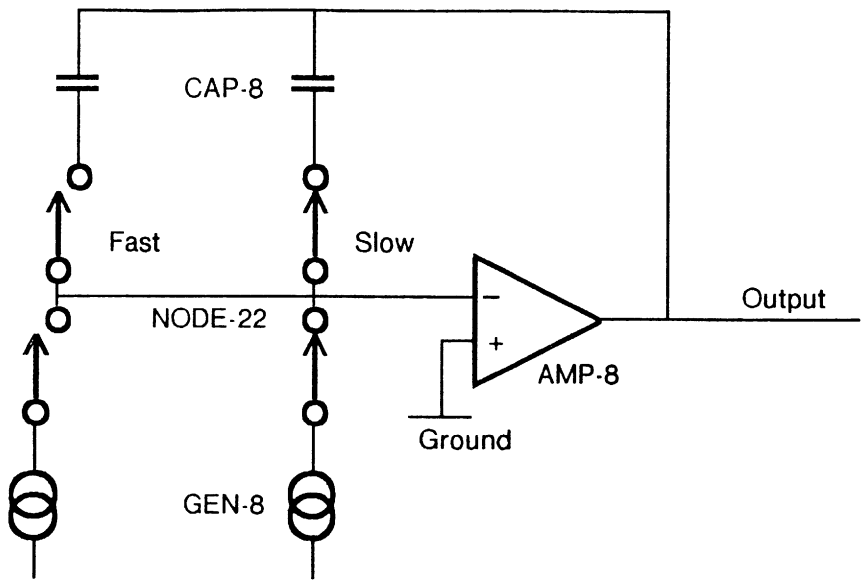
the (SLOW) generator shown previously except that the COND statements are carried around in some of the expressions.

Once the basic Miller circuit has been designed, then SPLIT-STATES is used to simplify the CONDs. Not until values of currents and capacitance are deduced is there any trouble. Once the CONDs are removed, then the following deductions could be made.

```
((= (STRENGTH GEN-8) (* (CAPACITANCE CAP-8) M1))      (FAST))
((= (STRENGTH GEN-8) (* (CAPACITANCE CAP-8) M2))      (SLOW))
```

The trouble here is the STRENGTH and the CAPACITANCE cannot both be constants if M1 is different from M2. Currents and capacitors cannot have conditional values and so trying to assign them conditional

Fig. 44. Switched Dual Ramp Generator



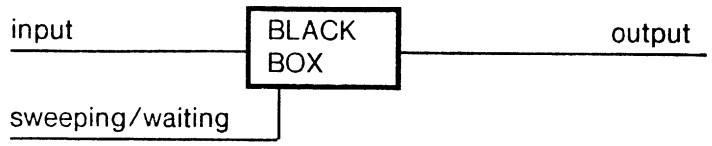
values causes a bug. This bug is easily remedied, however, by using two different current generators and two different capacitors and switching them in during the appropriate state of the conditional, as shown in the figure.

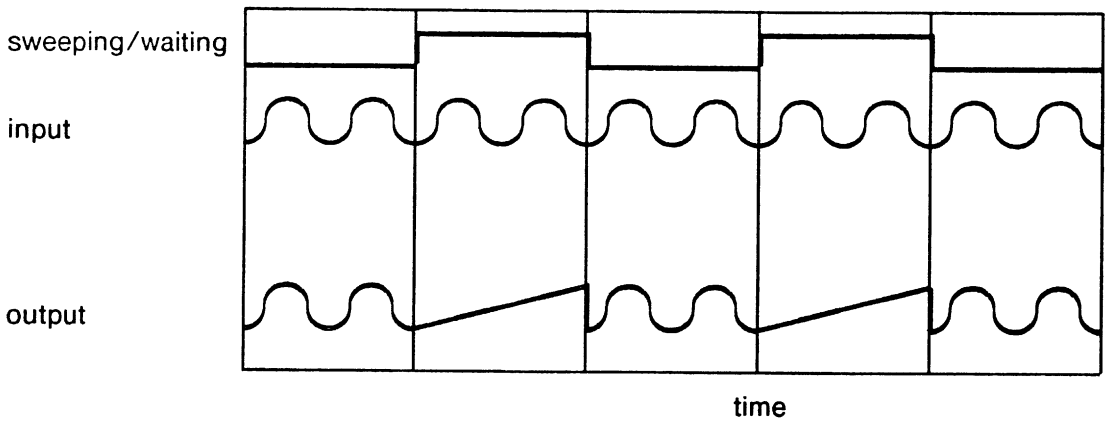
4.3 Design of Dependent States

Capacitors can destroy the idea of independent discrete states because capacitors carry continuous state across discrete state boundaries. A capacitor's voltage does not change instantaneously and so does not change when there is a discrete state change. Consequently constraints on the initial values of a capacitor branch voltages must be achieved in the preceding state. The dependence of one state upon another implies that states cannot be designed in isolation from each other.

This section uses an oscilloscope sweep circuit as a design example. The triggered sweep problem, which was outlined in the first chapter, is to design a circuit with two states, sweeping and waiting, that has two completely different behaviors during those states (see figure 45.). When the circuit is in the sweeping state, it produces a ramp output just as the ramp generators discussed previously do. During the waiting state, the output voltage should follow the input voltage. The crucial additional constraint is that the output voltage should be continuous during the transition between the waiting state and the sweeping state, or, equivalently, the initial output voltage during the sweeping state equals the input voltage. The specifications of the sweep circuit are also shown in figure 45. The situation tag (INITIAL SWEEPING) specifies the initial instant of the (SWEEPING) state.

Fig. 45. Triggered Sweep Problem





TEK4310

```
(GOAL ((CONTROLS NIL (D-DT (NDV OUTPUT)))  (SWEEPING)))
(GOAL ((CONTROLS (NDV INPUT) (NDV OUTPUT)) (INITIAL SWEEPING)))
(GOAL ((CONTROLS (NDV INPUT) (NDV OUTPUT)) (WAITING)))

((= M (D-DT (NDV OUTPUT)))     (SWEEPING))
((= (NDV OUTPUT) (NDV INPUT)) (INITIAL SWEEPING))
((= (NDV OUTPUT) (NDV INPUT)) (WAITING))
```

If the continuity requirement is ignored for the moment, then the goal during the waiting state is a simple one,

```
(GOAL ((CONTROLS (NDV INPUT) (NDV OUTPUT)) (WAITING)))  .
```

The direct way of achieving this goal is to use the WIRE rule, although FEEDBACK could be used instead. The goal during the sweeping state is the familiar ramp generator problem:

```
(GOAL ((CONTROLS NIL (D-DT (NDV OUTPUT))) (SWEEPING)))
```

Figure 46 shows the circuit made from a wire and a Miller ramp generator, one of the possible circuits for a ramp generator. When the wire and Miller circuits are merged, a KVL bug introduces a switch at the output as it does in one of the dual rate ramp generators. The KVL bug occurs because both

```
((CONTROLS (NDV INPUT) (NDV OUTPUT))    (WAITING))
((CONTROLS NIL         (NDV OUTPUT))    (SWEEPING))
```

try to control the output node voltage. Because these assertions have different situation tags, switches can be used to isolate the bugs.[1]

The trouble with this design is that there is no constraint on the initial voltage of the ramp generator. The ramp is always ramping and the switch just samples its output every once in a while. Some changes must be made to the ramp generator to set the initial voltage of the ramp in the (SWEEPING) state. That is the purpose of the goal regarding the (INITIAL SWEEPING) situation. The next rules describe ways to accomplish this goal.

The general strategy for achieving an initial condition is to convert it into a final condition in the immediately preceding state. Achieving the final condition also achieves the initial condition.

The INITIAL-CONDITION analyze and design rules shown in figure 47 carry the initial condition constraint between neighboring states. The assertion

```
(NEXT-STATE ?STATE0 ?STATE1)
```

says that ?STATE1 immediately follows ?STATE0. An initial condition in ?STATE1 is translated to a final condition in ?STATE0. The final instant of ?STATE0 is the same as the initial instant of ?STATE1 (remember capacitor voltages do not change instantaneously). If anything controls the final value of a capacitor's branch voltage in ?STATE0, then it controls the initial branch voltage in ?STATE1.

To set an initial condition in ?STATE1, the capacitor that controls the initial condition must be found because it will carry the continuous state information between neighboring states. In the case of the sweep generator, we want

```
(GOAL ((CONTROLS (NDV INPUT) (NDV OUTPUT)) (INITIAL SWEEPING)))
```

That is, an initial condition is placed on (NDV OUTPUT). The initial condition is really a constraint on the capacitor branch voltage that controls the output node. The deductions based on this initial condition are shown in figure 48.

Now we have a goal to achieve a final condition of a state. One method to achieve the final condition is to generalize the final condition goal for the entire state. If something is true for the entire state, it is certainly true for the last instant. The rules in figure 49 change a final condition goal into a goal for the entire state.[2] The analyze rule says that if something controls a capacitor branch voltage for an entire state, then it also controls the final value. The design rule does the inverse.

---

1. The bug must be handled differently than before, though, because the assertion (CONTROLS (NDV INPUT) (NDV OUTPUT)) is never made by the wire rule. The problem is that terminals are easily connected to nodes, but connecting nodes together requires making an equivalence class.

2. Another method is to have the state transition occur when the final condition is met. If for example, the state changes when the output of a ramp generator is equal to some input voltage, then the input voltage controls the final value of the state without it being necessary for the input voltage to control the output value of the state. This approach to setting the final condition of a state is frequently used in oscillators.

Fig. 46. Naive Sweep Design

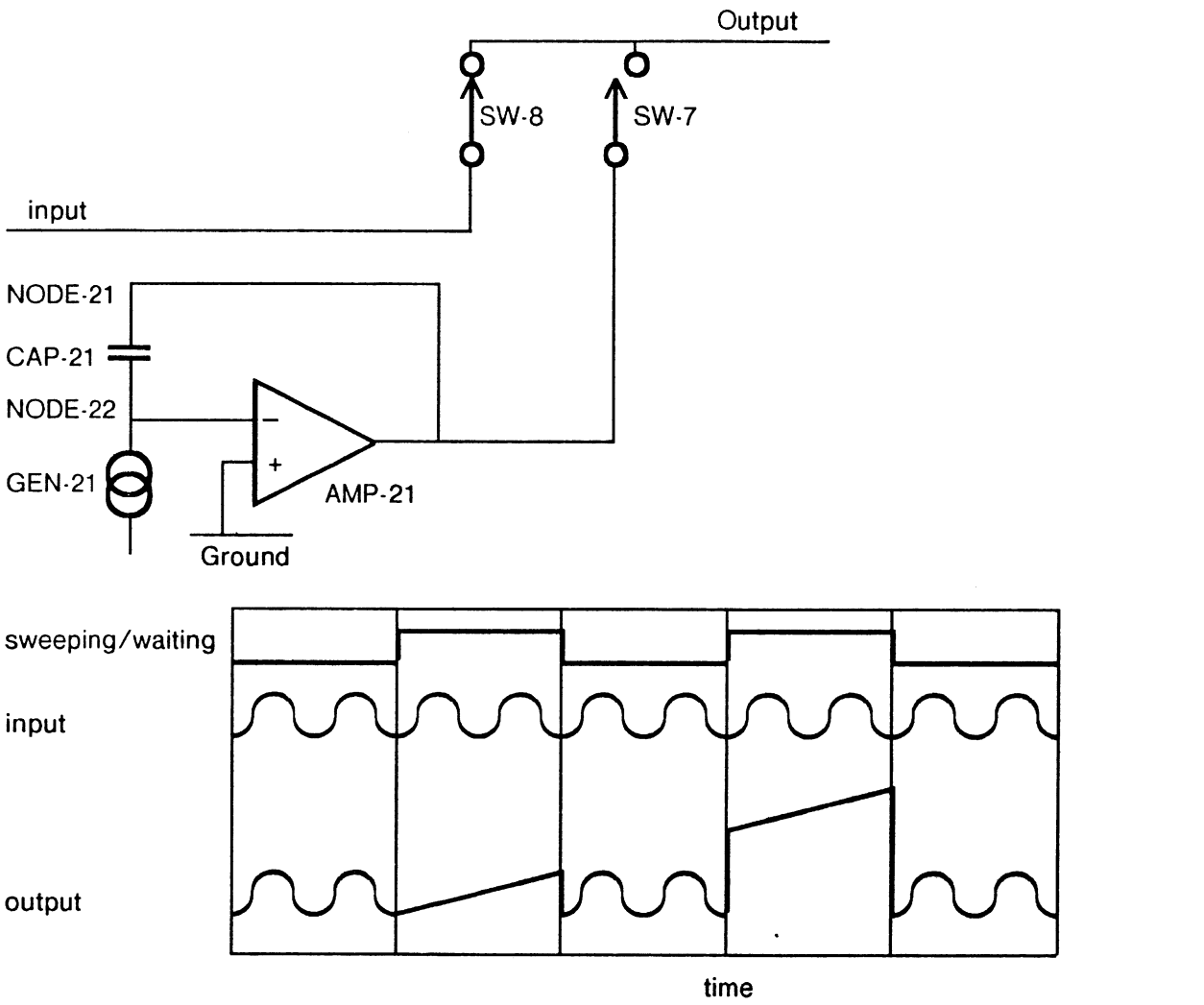


Fig. 47. Initial Conditions

```
(DEFRULE INITIAL-CONDITION-ANALYZE
        ((?F1 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F2 ((CONNECT (T1 ?CAP) ?ND1)        ?STATE0))
         (?F3 ((CONNECT (T2 ?CAP) ?ND2)        ?STATE0))
         (?F4 ((CONTROLS ?X (BRV ?ND1 ?ND2))   (FINAL ?STATE0)))
         (?F5 (NEXT-STATE ?STATE0 ?STATE1))
         (?F6 ((CONNECT (T1 ?CAP) ?ND3)        ?STATE1))
         (?F7 ((CONNECT (T2 ?CAP) ?ND4)        ?STATE1)))
  (ASSERT '((CONTROLS ?X (BRV ?ND3 ?ND4))      (INITIAL ?STATE1))
          '(INITIAL-CONDITION ?F1 ?F2 ?F3 ?F4 ?F5 ?F6 ?F7)))

(DEFRULE INITIAL-CONDITION-DESIGN
        ((?F1 (GOAL ((CONTROLS ?X ?Y)         (INITIAL ?STATE1))))
         (?F2 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F3 ((CONNECT (T1 ?CAP) ?ND1)        ?STATE1))
         (?F4 ((CONNECT (T2 ?CAP) ?ND2)        ?STATE1))
         (?F5 ((CONTROLS (BRV ?ND1 ?ND2) ?Y)   ?STATE1))
         (?F6 (NEXT-STATE ?STATE0 ?STATE1))
         (?F7 ((CONNECT (T1 ?CAP) ?ND3)        ?STATE0))
         (?F8 ((CONNECT (T2 ?CAP) ?ND4)        ?STATE0)))
  (PROPOSE-METHOD (?F1 ?F2 ?F3 ?F4 ?F5 ?F6 ?F7 ?F8) ?F9
    (GOAL-ASSERT '((CONTROLS ?X (BRV ?ND1 ?ND2)) (FINAL ?STATE0))
                 '(INITIAL-CONDITION ?F2 ?F3 ?F4 ?F5 ?F6 ?F7 ?F8 ?F9))))
```

Fig. 48. Initial Conditions of Sweep Circuit

```
(GOAL ((CONTROLS (NDV INPUT) (NDV OUTPUT))      (INITIAL SWEEPING)))
((CONNECT (T1 (CAP-21)) NODE-21)                (SWEEPING))
((CONNECT (T2 (CAP-21)) NODE-22)                (SWEEPING))
((CONTROLS (BRV NODE-21 NODE-22) (NDV OUTPUT))  (SWEEPING))
(NEXT-STATE (WAITING) (SWEEPING))
(GOAL ((CONTROLS ?X (BRV NODE-21 NODE-22))      (FINAL WAITING)))
```

Fig. 49. Final Condition Rules

```
(DEFRULE FINAL-CONDITION-ANALYZE
        ((?F2 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F3 ((CONNECT (T1 ?CAP) ?ND1)        ?STATE))
         (?F4 ((CONNECT (T2 ?CAP) ?ND2)        ?STATE))
         (?F5 ((CONTROLS ?X (BRV ?ND1 ?ND2))   ?STATE)))
  (IF (NOT (MEMQ (CAR ?STATE) '(INITIAL FINAL)))
      (ASSERT '((CONTROLS ?X (BRV ?ND1 ?ND2))   (FINAL ?STATE))
              '(FINAL-CONDITION ?F2 ?F3 ?F4 ?F5))))

(DEFRULE FINAL-CONDITION-DESIGN
        ((?F1 (GOAL ((CONTROLS ?X ?Y)          (FINAL ?STATE))))
         (?F2 (CIRCUIT-PART CAPACITOR ?CAP))
         (?F3 ((CONNECT (T1 ?CAP) ?ND1)        ?STATE))
         (?F4 ((CONNECT (T2 ?CAP) ?ND2)        ?STATE))
         (?F5 ((CONTROLS (BRV ?ND1 ?ND2) ?Y)   ?STATE)))
  (PROPOSE-METHOD (?F1 ?F2 ?F3 ?F4 ?F5) ?F6
    (GOAL-ASSERT '((CONTROLS ?X (BRV ?ND1 ?ND2)) ?STATE)
                 '(FINAL-CONDITION ?F2 ?F3 ?F4 ?F5 ?F6))))
```

The final condition rules transform the goal

```
(GOAL ((CONTROLS (NDV INPUT) (BRV OUTPUT NODE-22)) (FINAL WAITING)))
```

into

```
(GOAL ((CONTROLS (NDV INPUT) (BRV OUTPUT NODE-22)) (WAITING)))
```

This goal is entirely within one state and design ideas of the previous chapters can be used to design the circuit.

The next figure gives some of the deductions in the design of the sweep circuit. The basic goal is to set the branch voltage of the capacitor from the input node voltage. Two subgoals are undertaken to do that. The first pegs one terminal of the capacitor at ground; this goal is already satisfied because the Miller sweep circuit did that in the (SWEEPING) state and, if the connections are carried over into the (WAITING) state, it will still be true.

The second subgoal is to use the input node voltage to set the output node voltage. The wire rule would be a candidate for this job, but it will cause a KVL bug. Feedback is another way to do the job. Connecting the output of an operational amplifier to the output node will also cause a KVL bug, so we have to be clever about controlling the capacitor branch voltage. We are trying to control the node voltage of one terminal of the capacitor; the other terminal of the capacitor is at ground, so the node voltage could be set by setting the capacitor branch voltage. The capacitor branch voltage is controlled by the its terminal current, so the KCL rules propose setting the node current into either T1 or T2 of the capacitor; T1 cannot be used because of an irresolvable KCL bug. T2 can be used. The problem is to find a component whose output

current is controlled by a branch voltage; an operational transconductance amplifier is one possibility; another would be a resistor with a branch voltage imposed across it. The operational transconductance solution is shown in the figure and all of the goals follow from using it.

Fig. 50. Sweep Generator Deductions

```
(GOAL (CONTROLS (NDV INPUT)                          ;goal for initial condition
                (BRV NODE-21 NODE-22))))

(GOAL (CONTROLS (NDV GROUND) (NDV NODE-22)))              ;true by prior work
(GOAL (CONTROLS (NDV INPUT)  (NDV OUTPUT)))
                                                          ;wire rule wont work
(GOAL (CONTROLS (BRV INPUT NODE-21) (NDV NODE-21)))       ; but feedback will
(GOAL (CONTROLS (BRV INPUT NODE-21) (BRV NODE-21 NODE-22)));a KVL law
(GOAL (CONTROLS (BRV INPUT NODE-21) (TC (T2 CAP-21)))     ;capacitor VIC
(GOAL (CONTROLS (BRV INPUT NODE-21) (NC NODE-22)))        ;KCL
(CIRCUIT-PART OTA OTA-21)
```

4.4 Summary

This chapter discussed discrete state design. Sometimes the discrete states of a circuit are independent of one another; in that case the design for each state can be done without worrying about the other states. When the discrete states are not independent, then attention is paid to the transitions between the states. The initial conditions in a state must be present at the state transition and that implies setting them up in the previous state. Whether the states are independent or not, the designs for the individual states must combined to make a complete circuit. Switches are often needed to isolate different parts of a circuit. The designer deduces where these switches belong from KVL and KCL bugs.

5. Literature

The literature on electronic design can be crudely broken into the numeric and the symbolic approaches. Each of these approaches can be further broken down according to its emphasis being either analysis or synthesis. In general, more work has been done on analysis. In almost all work the computer is given a fixed circuit topology; my work is different because the computer develops the circuit topology.

The numerical approach to analysis is a straight forward application of network theory. These programs take a circuit, write the numerical equations of VICs, KCL, and KVL, and solve the equations numerically using matrix methods and numerical integration techniques <Chua>. These programs use sophisticated numerical techniques to produce accurate simulations or frequency analyses of real circuits. A human designer uses these programs to check his design without building it.

Some synthesis programs have been built on top of numerical analysis procedures <Director>. Such programs start with a circuit topology, initial component values for that topology, and an objective function that "grades" the output of the circuit. An analysis program finds the output of the circuit, the objective function grades that output, and then the synthesizer adjusts component values to improve the grade. This design methodology starts with a given topology; it does not create one. During the above optimization procedure, though, some components may "disappear" because their values go to zero (eg, for a capacitance) or to infinity (eg, for a resistor). One could argue that a topology could be found by optimizing a set of nodes that is fully connected with all possible components, but such an argument is not a satisfying explanation of how humans design circuits. An expert circuit designer must create the circuit in a reasonable and intelligent manner.

There are procedures that do some form of topological design. Lin and Chua <Lin> noticed some interesting properties of voltage n-tuplers and used those properties to enumerate a class of n-tupler circuits. Their design algorithm is specific for n-tuplers and could not be used for operational amplifier circuits.

Some procedures do topological design by examining the specified admittance matrix for the desired circuit <Daniels> <Yanagisawa> <Stevenson>. These procedures look for patterns of terms in the matrix and infer particular components based on those terms; the goal is to change the admittance matrix into one that can be made from simple admittances (resistors, capacitors, and inductors). The matrix is changed with some simple row and column operations that imply the use "norators" and "nullators" in the final circuit design. A nullator-norator pair can be made with an ideal operational amplifier, though at the end of the procedure it is not obvious which nullators go with which norators or whether the amplifier gain should be plus infinity or minus infinity.[1] This design procedure is more appealing than the optimization strategy because it introduces new nodes and components only when there is reason to do so. The procedure can design circuits with an arbitrary linear admittance matrices for any number of inputs and outputs, but cannot handle independent sources and switches.

---

1. This ambiguity exists because the procedure deals only with arithmetic constraints and does not have an idea of causality. It assumes the inputs of an operational amplifier are at equal potentials (the nullator constraint) but doesn't know which norator (operational amplifier output) enforces that equality.

The symbolic approach to circuit analysis tries to mimic the problem solving behavior of a person. These systems also have the desirable property that the deductions can be made from local information and therefore the rules are simple to write down. Examples of these programs are in <Stallman> and <de Kleer-4>. Some advantages of these programs are that they solve as much of a problem as they can as information becomes available, they allow parameters to be changed and will remove all the deductions based on the old values, and provide simple explanations of how parameter values were deduced. An offshoot of the development of these analysis programs is the development of truth maintenance systems (TMS) <Stallman> <Doyle> <de Kleer-3> <McAllester> that make the incremental addition and retraction of facts easy to do. My research used Doyle's TMS.

The use of symbolic expressions allows these analysis programs to do synthesis under the guiding hand of a human designer. The systems do not care if they know the current and the resistance and solve for the voltage or know the voltage and current and solve for the resistance. Thus by specifying desired voltages and currents these programs will calculate the component values needed for a particular circuit. (Some care is needed because the system may find ridiculous values in some cases, but if the designer notices a mistake, he can retract the bad values.) This approach is powerful. For example, Sussman's EESYS <Sussman-4> can easily find the transfer function implemented by a particular filter circuit. If the desired transfer function is then specified, EESYS (when augmented with polynomial coefficient matching and root finding routines)[1] will calculate the needed values of resistance and capacitance to make the circuit implement that particular transfer function.

These analysis programs do not view the circuit through a fixed pair of glasses as the classical analysis program does. Instead there are several aspects that the programs consider -- the DC and AC models, for example. These different models interact with each other by passing information between them. The DC model, for example, is used to calculate the transconductance that is used in the AC model. Sometimes equivalent models of a circuit fragment can pool their information to solve an analysis problem that neither could do individually <Sussman-3>. The answers are not necessarily exact, but engineers use the same techniques.

These analysis programs, though they do make use of several modeling aspects, rely heavily on algebraic manipulation to provide their answers. This manipulation, while needed to get particular answers, is not an essential part of design knowledge. De Kleer's NEWTON <de Kleer-1> shows how qualitative knowledge can be used to set up later algebraic computations. His Ph.D. thesis <de Kleer-5> shows that circuit mechanisms can be described with a simple qualitative algebra that does not include numbers. These results are reassuring because it is difficult to believe that people understand circuits by reducing them to algebraic expressions.

Brown's thesis <Brown> on debugging circuits doesn't fit into the classification of analysis or design but it is interesting because it mimics human behavior. His program uses debugging strategies such as signal tracing and a detailed description of a radio to fix receivers that do not work. His system zeros in on a fault by

---

1. I did this.

making hypotheses about the source of trouble and checking those hypotheses with measurements of the faulty radio. The program intelligently proposes only hypotheses consistent with the symptoms of the fault and the measurements that have already been made.

These symbolic programs all deal with a given topology. McDermott <McDermott-2> is one of the few who design circuit topology. McDermott, while working fundamentally on the organization of a general purpose problem solver, undertook the design of electronic circuits as his microworld. His approach to designing filters and transistor amplifiers uses qualitative descriptions to suggest particular circuit fragments and then glues the fragments together. His system is promising and has several good ideas, but unfortunately it works in a difficult design domain without the aid of simplifying abstractions. The system falls into the trap of simultaneously trying to design a transistor amplifier and a bandpass filter. A human designer would divide the tasks into first designing an acceptable filter transfer function, then a filter circuit with ideal amplifiers, and then the transistor versions of the ideal amplifiers. My present work stays in one of these tasks -- using ideal components. Future work should manage all three tasks.

In addition to the above work, some basic works on problem solving and on automatic programming have influenced me. The idea of debugging originates in HACKER, Sussman's Blocks World planner <Sussman-1> <Sacerdoti-1-2-3>. The idea of debugging lets problem solving (or design) rules be simpler than they would be if they had to anticipate the interactions with other rules.

Modifying the behavior of electronic circuits with the aid of switches is not discussed in the literature of circuit design. The problem is, however, similar to the problem of putting conditional code in programs and so the literature on automatic programming is relevant. The ideas behind introducing switches in a circuit design have been borrowed from Manna and Waldinger's <Manna> "Conditional Formation" rule that their programming system uses to introduce conditional clauses. Circuit design and computer programming share a lot of common problems. Manna and Waldinger's other rules about recursion formation, well founded orderings, procedure formation, generalization, and simultaneous goals have also affected my views about how circuits should be designed.

6. Conclusions

6.1 The Imperfect Past

Katz Maxim No. 8.[1]

Try to find the real tense of the report you are reading: Was it done, is it being done, or is it something to be done? Reports are now written in four tenses: past tense, present tense, future tense, and pretense. Watch for novel uses of CONGRAM (CONtractor GRAMmar), defined by the imperfect past, the insufficient present, and the absolutely perfect future.

This work tries not only to design circuits but also to mimic the way humans design them. The focus of the circuit design is on circuit topology: finding suitable components and discovering how to connect them together. Design systems that start from a known topology and only compute component values bypass the question of how the circuit originated in the first place. To mimic human designers, the system must do topological design.

There are alternatives to the topological design of circuits. One possibility is an extensive library of practical circuits that can be used for a variety of problems. Certainly a human circuit designer uses such a library: when a designer needs an electrical filter with a particular transfer function, for example, he will look in a catalog of filter circuits and choose an appropriate one. The catalog provides design equations so the engineer's task is little more than selecting a circuit and plugging numbers into equations. The chief drawback of this alternative to topological design is its lack of flexibility: if the problem is not listed in the catalog, then it cannot be solved. Contrast this drawback with the design of the triggered sweep circuit where, through a modification of the circuit topology, a new circuit performs both as a ramp generator and as an amplifier. It is unlikely that such a circuit would exist in a library.

The use of a library or catalog of circuits is not a satisfying model of human circuit designers. While people do know several circuits that perform a particular function -- such as constant current, bootstrap, and Miller ramp generators -- I doubt they depend on these stereotypical circuits in real design. I do not remember the formulas for even simple circuits like the inverting amplifier; instead I deduce the formulas from ideas about virtual grounds, flowing currents, and Ohm's law. The advantage of this mode of thinking is that it not only provides the design formulas for a circuit, but it also clearly outlines how the circuit works; knowing how a circuit works implies that modifying it is just a few steps away. Circuit designers give names to circuits so they can talk about them and the ideas behind their design, but the names are not indices to specific circuit designs. When a human designer stumbles across a new circuit, he does not try to remember that circuit component for component and node for node. Instead he looks for how the circuit works so he can use the idea in other circuits even though the original circuit would be useless. The best example of this kind of idea is feedback: once one has learned how a feedback amplifier works, he can apply the idea to such diverse circuits as ramp generators or phase locked loops.

---

1. Amrom Katz, "A Guide for the Perplexed, or a Minimal/Maxim-al Handbook for Tourists in a Classified Bureaucracy", Air Force/Space Digest, November 1967. Quoted in The Official Rules, Paul Dickson, Dell Publishing Co., New York, 1978.

6.2 The Insufficient Present

The circuit design techniques discussed in this paper are restrictive. The techniques are suited to designing circuits that have only unity gain, have no DC bias, and are direct coupled. Feedback is assumed to always work. Circuits with simultaneous constraints such as gyrators and impedances are not handled. The techniques are generally suitable for circuits such as sweep generators, function generators, and multiplexers. These restrictions are not viewed as difficult to surmount: the ability to achieve simultaneous constraints needs rules that know how to deal with superposition. Such an additional rule would not be a radical change from the methods presented here. It is sometimes surprising to realize how close the rules are to designing another kind of circuit. The triggered sweep circuit is actually a sample hold circuit if the slope of the ramp is set to zero.

How much of the system is coded and works? The design system described here, as mentioned earlier, separates the arithmetic constraints from the control constraints. The actual program kept them together as a control constraint and a qualitative descriptor of the sign and magnitude of the arithmetic constraint. This was done to guarantee large loop gains and the appropriate sign when feedback loops were used. The program designed voltage followers, several ramp generators, and specialized COND clauses, but it did not introduce switches after the KVL and KCL bugs were found. The approach was cumbersome and limited; the separate propagation of control and arithmetic constraints described here is better.

The rules have no information about which rule would be best to apply next because there is no idea of the cost of certain design choices. It appears, though, that the simple idea of doing that which is easiest would work well. The wire rule and the feedback rule, for example, both achieve the same goal. While the system has no bias for either one, the wire rule, because it is cheaper, is almost always the rule to try first; if it fails then feedback can be used. This path of least resistance approach is effective and probably should be included in the interpreter.

For further work a better problem solving language needs to be developed. The current rules used AMORD because AMORD was available. A better language would understand simple mathematical expressions and compute upper and lower bounds; a full algebra system should not be necessary. The required mathematics is similar to problems in linear programming. The rules should also be embedded in some form of hierarchy: circuit design usually occurs at several levels. During one level an amplifier might be an atomic object; later, when a scheme using an amplifier has proved successful, the amplifier can be designed using transistors and resistors. During this second level of design, most of the rules and assertions from the first level are irrelevant and should not be considered in the design. Presently the design is done depth first: one path being completely explored before any other method is considered.

6.3 The Absolutely Perfect Future

Circuit design, which comprises several different skills, might be broken down into 3 broad areas: (1) analog design, (2) finite state machine design, and (3) signal processing. There is a lot that can be done in all three of these areas. Analog design, the subject of this paper, involves building circuits from electronic components such as amplifiers, capacitors, and switches. The rules of this paper are simplistic and incomplete and could be improved and expanded. Devices such as transistors, comparators, and diodes (which have not been considered here) have a wealth of design problems and design procedures that beg to be put into rules. There are constraints on voltage and current limits that must be obeyed. Some simple rules that relate gain-bandwidth and frequency response or gain and collector current could decide when operational amplifiers are inadequate and discrete transistor amplifiers need to be used. Some transconductance calculations can show that one stage provides inadequate gain and thus demand that two or more states be used. There are several possibilities. I've studied the design of the Tektronix 465 oscilloscope sweep generator and found that rules like those described above can design most of its transistor amplifiers and transistor switches.

The second broad area is finite state machine (FSM) design. Little has been done in this area (but see <Grinberg>). The basic task in FSM design is to take a problem description, say the measurement of a time interval, and find an appropriate FSM to solve the problem. The finite state machine has to insure several constraints that are not explicitly stated in the original problem. A time interval, for example, is just a number that the FSM must find a way to compute. The finite state machine will need several states -- perhaps one to clear a counter, another to wait for the start of the interval, another to wait for the end of the interval, and a state to signal that the number is valid. Finding these different states and deciding how they should be implemented is the task of a FSM designer. Such a system would be a simple logic designer. Classic logic design has not focused on specifying an initial machine. Furthermore, classic synthesis techniques don't utilize the variety of parts available and thus produce inferior designs.

The third area is signal processing. There are some fields of circuit design that are so specialized that they need their own models that are independent of component VICs, KVL, and KCL. FSM design is one such field; another is signal processing. Frequency compensation of amplifier and specifying the poles and zeros of filters are tasks that need models in the time and frequency domains. Expert design rules are needed in this area before a practical circuit could be designed. Unfortunately, this area is the most difficult of the three because the the mathematics can be complex.

There are interesting problems in each of these areas and some progress can be made in each. The best area for future work is undoubtably finite state machines because new integrated circuit technology strains a human designer's ability to make quick, error-free, designs. An automated designer is better suited for the tedium of these designs but it must be involved in the design stage if it is to understand the purpose of the different circuits and effectively apply its potential. If it knows that a counter is being used to measure an interval then it can insure that the clear and count commands occur at the proper times; if it does not know the purpose of the counter, then the signals are meaningless and nothing can be checked.

7. Bibliography

Brown Brown, A., Jr., **Qualitative Knowledge, Causal Reasoning, and the Localization of Failure,** MIT AI-TR-362, 1977.

Barstow Barstow, D., **Automatic Construction of Algorithms and Data Structures Using a Knowledge Base of Programming Rules,** Stanford Artificial Intelligence Laboratory Memo AIM-308, Nov. 1977.

Bundy Bundy, A., "Analyzing Mathematical Proofs," **Advance Papers of the 4th International Joint Conference on Artificial Intelligence,** pp 22-28.

Chua Chua, L., and P. Lin, **Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques,** Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

Daniels Daniels, R, "A Nullator-Norator Synthesis Procedure Applied to Gyrators," **Proceedings of the IEEE 12th Midwest Symposium on Circuit Theory,** Texas, April 1969, pp. IX.3.1-IX.3.8.

de Kleer-1 de Kleer, J., **Qualitative and Quantitative Knowledge in Classical Mechanics,** MIT AI-TR-352, 1975.

de Kleer-2 de Kleer, J., "Local Methods for Localizing Faults in Electronic Circuits," MIT AI Memo 394, 1976.

de Kleer-3 de Kleer, J., J. Doyle, C. Rich, G. Steele, G. Sussman, "AMORD = A Deductive Procedure System," MIT AI Memo 435, Jan. 1978.

de Kleer-4 de Kleer, J. and G. Sussman, "Propagation of Constraints Applied to Circuit Synthesis," MIT AI Memo 485, September. 1978.

de Kleer-5 de Kleer, J., **Causal and Teleological Reasoning in Circuit Recognition,** MIT AI-TR-529, 1979.

Director Director, S., "Towards Automatic Design of Integrated Circuits," in **Basic Questions of Design Theory,** W. Spillers, ed., American Elsevier, New York, 1974, p 303.

Doyle Doyle, J., **Truth Maintenance Systems for Problem Solving,** MIT AI-TR-419, January 1978.

Gorry Gorry, G. Anthony, "Research on Expert Systems," Project MAC Technical Memo 56, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, December 1974.

Graeme-1 Graemé, J., et al, **Operational Amplifiers,** McGraw-Hill, New York, 1971.

Graeme-2 Graeme, J., **Applications of Operational Amplifiers,** McGraw-Hill, New York, 1973.

Gray Gray, P., and C. Searle, **Electronic Principles: Physics, Models, and Circuits,** John Wiley & Sons, New York, 1969.

Grinberg Grinberg, M., "Semi-Automatic Digital Designer System," Ph. D. proposal, TR-685, Computer Science Center, University of Maryland, College Park, Maryland, June 1978.

Lenat Lenat, D., AM: an Artificial Intelligence Approach to Discovery in Mathematics as **Heuristic Search,** Stanford University Artificial Intelligence Laboratory Memo AIM-286, 1976.

Lin Lin, P. and L. Chua, "Topological Generation an Analysis of Voltage Multiplier Circuits," **IEEE Transactions on Circuits and Systems,** Vol CAS-24, No 10, October 1977, pp 517-530.

Manna Manna, Z., and R. Waldinger, **Synthesis: Dreams => Programs,** Stanford University Artificial Intelligence Laboratory Memo AIM-302, Stanford, California, November 1977. Also "Synthesis: Dreams => Programs", **IEEE Transactions on Software Engineering,** Volume SE-5 Number 4, July 1979, pp. 294-328.

McAllester McAllester, D., **The Use of Equality in Deduction and Knowledge Representation,** MIT AI-TR-550, January 1980.

McCarthy McCarthy, J., and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence" in **Machine Intelligence,** Volume 4, pp 463-502, ed. by B. Meltzer and D. Michie, American Elsevier, New York, 1969.

McDermott-1 McDermott, D., and G. Sussman, "The Conniver Reference Manual," MIT AI Memo 259a, Jan. 1974.

McDermott-2 McDermott, D, **Flexibility and Efficiency in a Computer Program for Designing Circuits,** MIT AI-TR-402, December 1976.

Refai Refai, S., "Application of Mason-Coates Graph in Linear Active Network Synthesis," **Proceedings of the IEEE 22nd Midwest Symposium on Circuits and Systems,** Philadelphia, June 1979, pp 284-88.

Rich Rich, C., and H. Shrobe, **Initial Report on a LISP Programmer's Apprentice,** MIT AI-TR-354, December 1976.

Roberge Roberge, J. K., **Operation Amplifiers: Theory and Practice,** John Wiley, New York, 1975.

Sacerdoti-1 Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," **Proceedings of the Third International Joint Conference on Artificial Intelligence,** pp 412-422, August, 1973.

Sacerdoti-2 Sacerdoti, E., "The Nonlinear Nature of Plans," **Advance Papers of the Fourth International Joint Conference on Artificial Intelligence,** pp 206-214, Tbilisi, Georgia, USSR, September 1975.

Sacerdoti-3 Sacerdoti, E., **A Structure for Plans and Behavior,** Stanford Research Institute Artificial Intelligence Group Technical Note 109, Menlo Park, California. Also American Elsevier, New York, 1977.

Shrobe Shrobe, H., **Dependency Directed Reasoning for Complex Program Understanding,** MIT AI-TR-503, April 1979.

Shortliffe Shortliffe, E., et al, "A Computer-Based Approach to the Promotion of Rational Clinical Use of Antimicrobials," in **Clinical Pharmacy and Clinical Pharmacology,** Goveia et al, editors, Elsevier/North Holland, 1976. pp 259-273.

Sridharan Sridharan, et al, "A Heuristic Program to Discover Syntheses for Complex Organic Molecules," Stanford AI Memo AIM-205, June 1973.

Stallman Stallman, R., and G. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", MIT AI Memo 380, September 1976.

Steele Steele, G. L., Jr., and G. Sussman, "Constraints", MIT AI Memo 502, Nov. 1978.

Stevenson Stevenson, J., "Network Synthesis by Admittance Matrix Expansion," **Colloquium on Electronic Filters,** Institution of Electrical Engineers Conference Publication Number 167, London, June 1978.

Sussman-1 Sussman, G., **A Computer Model of Skill Acquisition,** American Elsevier, New York, 1975.

Sussman-2 Sussman, G., and R. Stallman, "Heuristic Techniques in Computer Aided Circuit Analysis," MIT AI Memo 328, Mar. 1975.

Sussman-3 Sussman, G., "SLICES: At the Boundary between Analysis and Synthesis," MIT AI Memo 433, July, 1977.

Sussman-4 Sussman, G., EESYS is a collection of programs written for a course at MIT.

Tektronix-1 **465 Oscilloscope Service Instruction Manual,** Tektronix, Inc., Beaverton, Oregon.

Tektronix-2 **475A Oscilloscope Service Instruction Manual,** Tektronix, Inc., Beaverton, Oregon, 1976.

Wakerly Wakerly, J. F. **LOGIC DESIGN PROJECTS Using Standard Integrated Circuits,** John Wiley, New York, 1976.

Winograd Winograd, T., "Breaking the Complexity Barrier (Again)," ACM SIGPLAN Notices, 1975, 10, pp 13-30.

Winston Winston, P., **Artificial Intelligence,** Addison-Wesley, Reading, Massachusetts, 1977.

Yanagisawa Yanagisawa, T., and N. Kanbayashi, "Realization of Arbitrary Conductance Matrix Using Operational Amplifiers," **Proceedings of the IEEE International Symposium on Circuits and Systems,** Munich, West Germany, April 1976, pp 532-35.

CS-TR Scanning Project
Document Control Form

Date : 2/15/96

Report # AI-TR-703

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 66(73-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other:_____

Check each if included with document:
☒ DOD Form(2) ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): FOLLOWS ABSTRACT PAGE

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:
IMAGE MAP! (1-66) UNNUMBERED TITLE PAGE, DEDICATION, ABSTRACT, BLANK, 3-4, 6-65
(67-73) SCANCONTROL, COVER, DOD(2), TRGTS(3)

Scanning Agent Signoff:
Date Received: 2/15/96 Date Scanned: 2/20/96 Date Returned: 2/22/96

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-703	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD-A128631
4. TITLE *(and Subtitle)* A Simple Model of Circuit Design		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald Roylance		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0505 N00014-80-C-0622
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE May 1980 13. NUMBER OF PAGES Pages 65
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Distribution is unlimited.

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Expert Systems
Design Automation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A Simple analog circuit designer has been implemented as a rule based system. The system can design voltage followers, Miller integrators, and bootstrap ramp generators from functional descriptions of what these circuits do. While the designers works in a simple domain where all components are ideal, it demonstrates the abilities of skilled designers. Whilethe domain is electronics, the design ideas are useful in many other engineering domains, such as mechanical engineering chemical engineering, and numerical programming. CON'T

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Most circuit design systems are given the circuit schematic and use arithmetic constraints to select components values. Thisscircuit designer is different because it designs the schematic. The designer uses unidirectional CONTROL relation to find the schematic. The circuit designs are built around this solution it restricts the search space, assigns values to components, and finds design

REPORT DOCUMENTATION PAGE

1. REPORT NUMBER TR-703	2. GOVT ACCESSION NO.
4. TITLE (and Subtitle) A Simple Model of Circuit Design	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald Roylance	8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0505 N0001A-80-C-0622
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209	12. REPORT DATE May 1980 13. NUMBER OF PAGES Pages 65
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217	15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited.	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Distribution is unlimited.	
18. SUPPLEMENTARY NOTES None	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Expert Systems Design Automation	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A simple analog circuit designer has been implemented as a rule based system. The system can design voltage followers, Miller integrators, and bootstrap ramp generators from functional descriptions of what these circuits do. While the designers works in a simple domain where all components are ideal, it demonstrates the abilities of skilled designers. While the domain is electronics, the design ideas are useful in many other engineering domains, such as mechanical engineering chemical engineering, and numerical programming.

CON'T

DD FORM 1473 EDITION OF NOV 65 IS OBSOLETE

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 2/20/1996

M.I.T. Libraries
Document Services