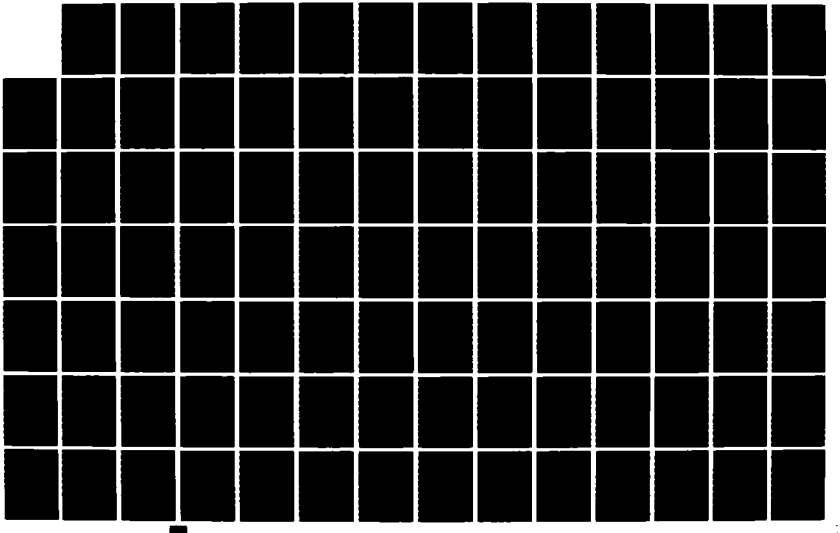
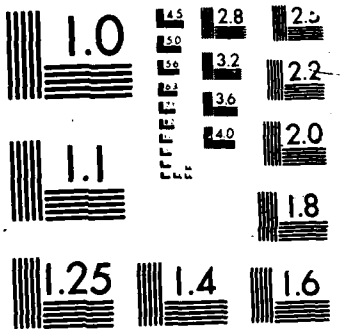
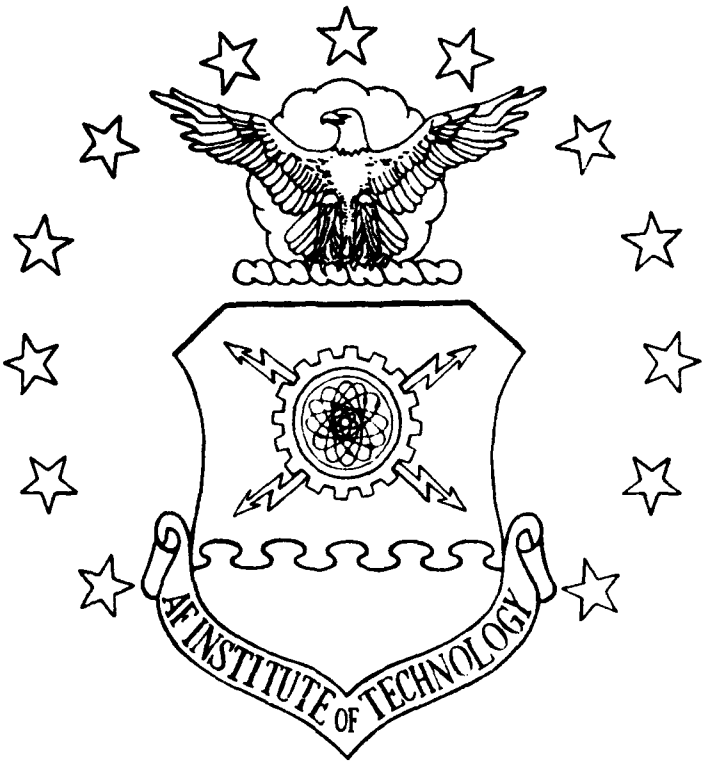
AD-A172 718 DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING D R WUNZ MAR 86 1/2

UNCLASSIFIED AFIT/GCS/ENG/86M-3 F/G 9/5 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.2
2.0
1.8



DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH

THESIS

Donald R. Wunz, Jr.
Captain, USAF

AFIT/GCS/ENG/86M-3

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/86M-3

DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH

THESIS

Donald R. Wunz, Jr.
Captain, USAF

AFIT/GCS/ENG/86M-3

Approved for public release; distribution unlimited

AFIT/GCS/ENG/86M-3

DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Donald R. Wunz, Jr., B.S.

Captain, USAF

March 1986

Preface

This thesis is a study into the use of functional analysis combined with structural diagnosis in analyzing analog electronic circuits. I present the background, an analysis of current research, and propose a design for the incorporation of functional rules and structural subdivision on a circuit. I then implement the design in KEE to demonstrate the proposal and to validate the theory.

While being a continuation of Lt. Ramsey's thesis, which automated the testing process used by Warner Robins Air Logistic Center, this project was also a separate approach to the diagnosis problem. I started my research with an examination of the various approaches suggested in the current literature; then, selecting ideas such as structural representation, functional breakdown, and causal analysis, I laid out a general plan using structure and function to do a preliminary pruning of the search space, and functional/causal analysis to restrict the assignment of components to the list of suspect parts. This approach has been suggested in some of the readings, but appears not to have been implemented as yet.

The development of this project required a few false starts and a lot of analysis of current trends. I would like to thank those who helped me sift through the mass to find what has become a diagnostic system: my thesis advisor, Dr (Maj) Stephen Cross, for his guidance in the research and suggestions of other approaches to consider; Dr Frank Brown for his critical review of this thesis; Capt Thomas Clifford for his reviews and technical assistance; and the technicians and engineers at Warner Robins Air Logistic Center for their help in my understanding

the current system and how things ought to be. I also thank those who provided moral support during this 21 month program.

Special thanks go to my wife, Gloria, and daughter, Melanie, for their love and understanding during my long hours of isolation. I can finally take you both to the beach, or maybe we'll go fishing.

Donald R. Wunz, Jr.

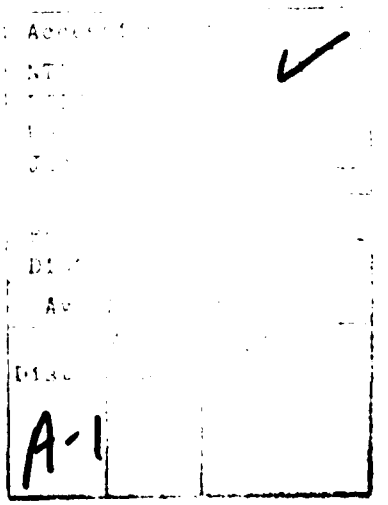


Table of Contents

Table of Contents

List of Figures

Abstract

The purpose of this thesis is to describe and propose an analog electronic circuit diagnostic system in an artificial intelligence environment using an expert system. An introduction to the topic, research into current technology, identification of participants, problems, goals and resources, preliminary design, detailed design and implementation, and conclusions and recommendations are covered.

As part of its continuing study of reliability and maintainability, Rome Air Development Center (RADC) has been tasked to develop an automated system to be used to diagnose electronic circuit boards. The objective is to use an expert system to aid in the diagnosis in order to identify bad components more specifically than is done by the Automated Test Equipment (ATE) now in use at the Air Logistic Centers.

The current system also does not correctly identify some components which are bad, and does not perform some required tests at all. This research project will describe and implement an experimental system in an attempt to correct these problems.

The current knowledge in artificial intelligence has just started to define some of the fundamental requirements for such a system. Discussions have been presented describing different approaches to the problem of circuit analysis. Stallman and Sussman defined a system that uses deep physical knowledge about electronics in an attempt to build a tool that traces through a circuit data base similar to a simulation system in an attempt to isolate the cause of a given fault. De Kleer

proposed a tool using a hierarchical definition based on structure and function. Others have proposed functional tracing, causal reasoning, and combinations of the above.

The problems identified by those involved, RADC and Warner Robins Air Logistic Center, were defined as stated earlier: the ATE system does not isolate the fault to specific components, does not correctly identify faulty components, and does not perform some required tests. The goals of this thesis are to research the background and the current techniques available, develop a definition of necessary data for an expert system knowledge base, propose a method of combining the knowledge base and expert system techniques into a diagnostic tool, and to implement a prototype of the system to demonstrate its usefulness.

The current work by others has proposed some common approaches. Many have suggested the use of functional reasoning about the effect a component has in a circuit; some have suggested the use of a hierarchical structure that can be traced which will improve the search of the knowledge base; and others have recommended the use of specific knowledge of electronic components in the analysis of faulty circuits. The approach taken in this research is to combine these into one diagnostic tool using the capabilities of each technique. A structural representation based on a subdivision of the circuit by functional subunits, combined with specific knowledge of the subunits and their individual components, is used as a preliminary design. This is then implemented with the use of KEE, an expert system building tool chosen for its support of graphics, menu selection, and interface with the underlying Lisp system.

It has been determined, after analyzing the prototype system, that the technique is a feasible approach and deserves further study. From

the prototype, a more user-friendly system could be built taking advantage of other capabilities of the KEE system; in particular, the system can be expanded to include other, more complex circuit designs.

DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS:

A FUNCTIONAL APPROACH

I. Introduction

This chapter presents an overview of the attempts by the Air Force to intelligently diagnose electronic component errors on flight qualified circuit boards, a discussion of the problems encountered, and a description of this research including the scope of the problems covered, the assumptions made, and the approach taken.

Background

The Air Force Logistics Command is responsible for the maintenance of Air Force aircraft. Several Air Logistic Centers exist, each charged with maintaining certain aircraft; Warner Robins is one such center and is responsible for the F-15 aircraft weapon system.

The Warner Robins Air Logistic Center (WRALC) currently uses automatic test equipment (ATE) to test printed circuit boards for the converter-programmer power supply on the F-15 aircraft. The diagnosis system operates 24 hours a day, 7 days a week and consists of a depot of ATE and highly skilled technicians. The ATE isolates problems in the power supply board, creating a list of suspect components; the technicians manually troubleshoot the board to identify the bad components for replacement. When the suspect components are identified, the board is forwarded to another station for replacement of the parts and then returned for testing. If the problem still exists, the process is repeated. This possibly results in several days' time to repair a card. The tests are incomplete and the list of suspect components is usually long

and vague. This is typical of the current method of testing analog circuits.

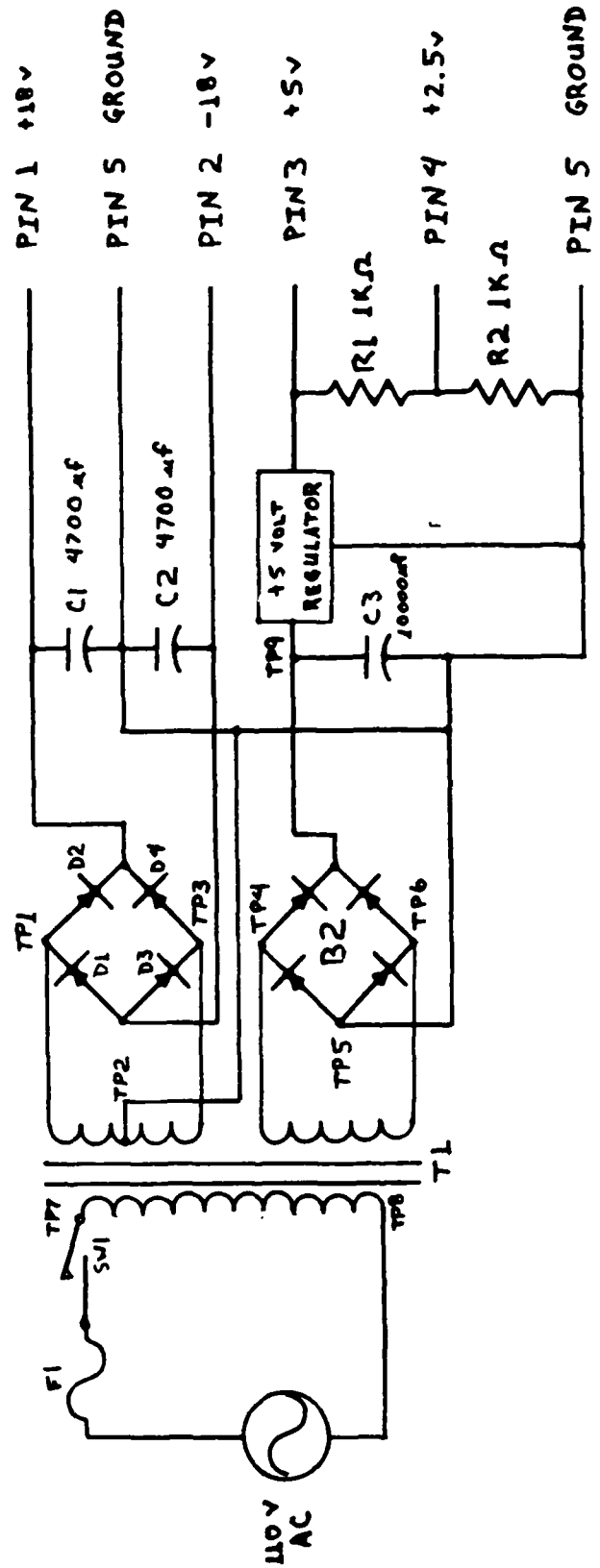
In this thesis, **structural testing** is defined as the testing of a circuit at the component level without regard to the functions of the components within the circuit or sub-circuits. **Functional testing** is defined as the attempt to assign fault at some sub-circuit level and to continue subdividing the circuit into functional units until a specific component can be identified as malfunctioning. Functional testing would more specifically identify suspect components thus being more thorough and requiring fewer test-repair cycles. A more specific identification of suspect components will reduce the maintenance time required for a board.

The analog power supply circuit depicted in Fig. 1 will be used for illustrative purposes -- both to describe problems with current approaches and to demonstrate the functional design advocated in this thesis. This is a simplified version of a typical power supply, specifically that of the Heathkit H-89 computer.

Power supplies in general perform similar functions, that function being the conversion of electrical power from one form to another. In this respect, the H-89 power supply is similar to the power supply found in the Air Force F-15. The F-15 circuit provides power from a different type of source and has provisions for emergency power, but is still just a power supply.

Problems

As part of its ongoing research in reliability and maintainability, the Rome Air Development Center (RADC) has been tasked to automate the



HEATH H-89
POWER SUPPLY

Figure 1: An Analog Power Supply

maintenance diagnostic procedures for AFLC. Towards this end, RADC has sponsored theses at the Air Force Institute of Technology (AFIT); one such thesis was accomplished by Lt James Ramsey (Ramsey, 1984). His thesis researched the use of artificial intelligence in electronic diagnosis and used the F-15 power supply board for a model.

The goals of Ramsey's thesis were, in part, to build an expert system to identify the generalized knowledge required, to implement the system using another technique to get a feel for different representations, and to implement the knowledge base from the ATLAS code currently run on the ATE for the F-15 power supply diagnostics. Other goals given were, to port the system to another machine, to include the physics of the circuit and a graphics interface, to interconnect the system with the ATE, and to implement a circuit simulation system. The earlier goals were accomplished, but the others were left unimplemented or at least incomplete.

This thesis is, in some respects, a continuation of his thesis in that the physics of the circuit (the functional analysis) is studied. The present research effort, however, is more concerned with general knowledge concerning functional analysis and less with specific circuits; hence, Ramsey's implementations of the ATLAS code testing will not be used.

Lt Ramsey's thesis identified numerous problems associated with the hardware and software of the ATE at WRALC. (Ramsey, 1984: I-2 - I-4):

1. Old hardware
2. Inflexible and poorly documented programs
3. Failure to isolate to a single component
4. Failure to isolate to the correct component

5. Failure to use accessible existing circuit connections

6. Failure to use existing computer hardware

7. Difficult interfacing between the unit under test and the ATE

8. Little or no feedback to the operator

9. Failure to check all possible components

Of these, three have been identified to be of primary interest to this project. These have been singled out by the technicians at WRALC as being the most troublesome; these particular problems appear to be something left to an expert and may be solvable using an expert system.

1. Failure to isolate to a single component - Current procedures using ATE result in several components being listed as suspect; sometimes as many as 25 to 30 for the power supply board.

2. Failure to isolate to the correct component - Due to inaccurate testing, the ATE sometimes incorrectly passes bad components and indicates the board is usable.

3. Failure to check all possible components - The current software does not even consider some components of the circuit when running the tests.

Scope

The purpose of this research is to propose and test a methodology and the requirements for performing a functional diagnosis of analog circuits. The final goal of a project of this magnitude would be to provide a viable user package, to be implemented in the field depots, which performs diagnostics on Air Force hardware.

This is a short-term project, however, which is limited in what can be accomplished. Therefore the research goals are restricted to an assessment of the present technology, analysis of basic requirements, and a prototype of the required software. Things not considered are:

graphic display of the circuit under test, automated input of the knowledge base from the schematics, and the external connection of this system to ATE systems for direct machine-machine communication.

Assumptions

The discussion that follows assumes the reader has an elementary knowledge of AC and DC electricity and a basic understanding of discrete (non-integrated chip) electronics (references would include texts such as Schaum's Outline Series books Basic Circuit Analysis, Electric Circuits and Electronic Circuits). Also, familiarity with artificial intelligence and Lisp programming may be helpful (with such references as LISPCraft by Wilensky and Artificial Intelligence by Rich). For implementation of this project, the KEE[1] system on a lisp machine supporting Zetalisp[2] functions was used.

Approach

The remaining chapters will begin with a discussion of the current knowledge, including definitions, a description of expert systems, and a look at current research in circuit diagnosis. Then the components involved in this research will be identified: participants, problems, resources, and goals. Next will be the foundation of the preliminary design and its formalization followed by a detailed design implementation and its testing. Finally, a discussion of the conclusions and recommendations for further work in this area will be presented.

[1]KEE is a trademark of Intellicorp

[2]Zetalisp is a trademark of Symbolics

II. Current Knowledge

Definitions

Obviously, before going into depth concerning this research, some terms must be defined to provide a common basis for the discussion.

Knowledge. First, **knowledge** is defined as the domain-specific information required to: (1) understand the domain's problem statements, and (2) provide the skill necessary to solve some of these problems (Hayes-Roth, 1983: 4). Relevant to this research, (Hayes-Roth, 1983: 4) also distinguishes between public and private knowledge. **Public knowledge** is the published information and data about the domain subject, and **private knowledge** is the heuristic knowledge necessary to make educated guesses, deal with incomplete or wrong data, and to recognize an approach as being on the right path to finding the desired solution.

Expert Systems. **Expert systems** is an area of artificial intelligence that "investigates methods and techniques for constructing man-machine systems with specialized problem-solving expertise." (Hayes-Roth, 1983: 3-4). In other words, an expert system is a system that can diagnose a faulty electronic circuit in a television receiver but cannot watch the TV when finished. An expert system relies heavily on the private (heuristic) knowledge of an expert to fill in the tremendous gaps left by the public knowledge.

Ideal Model of an Expert System

According to (Hayes-Roth, 1983: 16 - 17),

> the ideal expert system contains a language processor for problem-oriented communications between the user and the expert system; a "blackboard" for recording intermediate results; a knowledge base comprising facts as well as heuristic

planning and problem-solving rules; an interpreter that applies these rules; a scheduler to control the order of the rule processing; a consistency enforcer that adjusts previous conclusions when new data (or knowledge) alter their bases of support; and a justifier that rationalizes and explains the system's behavior (see Fig 2).

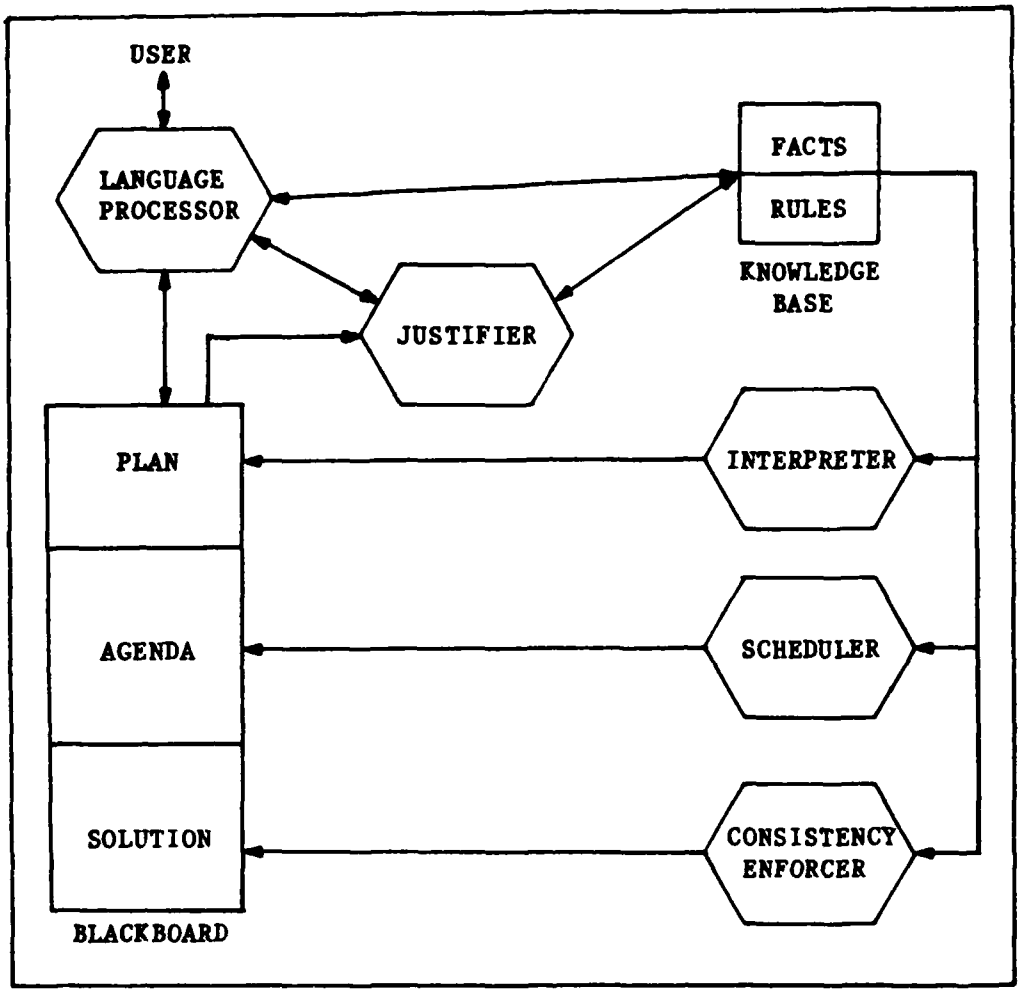


Figure 2: Anatomy of an expert system

Implementing an Expert System

Five basic steps are required to build an expert system: identification, conceptualization, formalization, implementation, and testing (Hayes-Roth, 1983: 23).

Identification defines the problem and its characteristics without concern, at the moment, with what is available or necessary for its

solution. This most often involves defining the problem in terms of a state space, initial and goal states, knowledge (operators or rules) that transforms one state into another, and choosing a control strategy that systematically applies the operators. The control strategy may be a linear progression or may even resemble how a human "thinks" about the domain; that is, it may follow some heuristic search to apply the next most logical operator.

Next, conceptualization identifies the necessary characteristics and the applicable concepts for the problem solution. The knowledge gained is then structured into an organization to represent a formal model during formalization after which implementation defines the rules required to represent the public and heuristic knowledge. Finally, testing verifies the implementation and evaluates its performance compared to some standard defined by experts in the domain field (Hayes-Roth, 1983: 23-24).

Summary of Current Knowledge

There currently exist several circuit diagnosis systems. These, however, have certain limitations. Some are excellent packages but are restricted to digital applications. Others test analog circuits but are limited to a structural diagnosis, failing to identify specific bad components; they either provide just a list of components or indicate that the failure is within some sub-circuit without giving any further details. To date, little has been done in applying functional diagnosis to a circuit. Following is a review of some of the current literature which, when implemented, could improve analog circuit diagnosis.

Stallman and Sussman. The approach taken by (Stallman, 1979) makes use of the structure of the circuit, the physical properties of the individual components, the physical laws of electricity, and propagation of constraints to analyze the circuit malfunctions. Their **method of assumed states** makes predefined assumptions concerning the circuit's voltage and current levels, and the state of various components within the circuit. These assumptions are then propagated through the circuit assigning 'responsibility' for each assumption to some law or rule, and specific, previously assumed values. When a contradiction is encountered, assumptions leading up to it are un-assumed and another path in the search is taken.

This propagation would be a complete circuit analysis if not for two problems in particular. First, an analog circuit of any complexity would generate a total search space too large to handle; and second, in the case of circuits containing feedback such as oscillators, a propagation schema may not know when to stop. Also, such a structure would be all but incomprehensible to an engineer since everything about the circuit is being considered at the same time; there is no functional breakdown to simplify the user's view of the problem.

De Kleer. (De Kleer, 1984b) looks at six progressive approaches to circuit diagnosis:

1. Modern approach - Now referred to as the conservative approach.
2. Empirical association approach.
3. Knowledge organization along structural and causal lines.
4. Deep knowledge about behavior to make troubleshooting inferences - What De Kleer calls the most powerful.
5. Deep knowledge about fault modes.
6. Causal models.

According to De Kleer, a troubleshooter does two things: he makes

measurements and replaces components. When making measurements, two steps are taken: he computes the results of previous tests, and decides to take more measurements. The troubleshooter's goal is to make the measurement that has a maximum of information gain. With this in mind, the goals of a troubleshooter expert system are: robustness, generality, efficiency, and constructability.

The modern approach is to write a program specifically for the problem at hand. Such a system could contain 100,000 lines of code and be capable of finding only 30% of the possible faults. Knowledge is implicit in the code; influence is not explicit. Thus the program is not robust. Since it is for a specific system, the program is not general; it follows the same path every time, from start to stop in the same order, resulting in poor efficiency. And, finally, it would be difficult to construct such a large program without error.

With the empirical association approach, a list of if-then rules is defined and processed. The system size is down to 10,000 rules from the 100,000 lines of code. Knowledge is explicit and therefore easier to change, but, if another rule is found, a problem arises in finding just where it should be placed. With 10 thousand rules, robustness is still poor and constructability is bad. Such a system is also weak in generality because a new system would require a whole new set of rules. Efficiency, however, would be close to optimal since the only rules fired would be those whose if part were satisfied.

When some sort of knowledge organization is added, the system is easier for the engineer to understand, and the robustness would improve; however, all else would remain the same.

The approaches described above all share one limitation: they ignore behavior of parts or components. The fourth technique uses deep knowledge about behavior of components to make troubleshooting inferences. Knowing the behavior of given parts, and values from measurements, constraints may be propagated until a conflict is found, identifying the cause of the problem. The problem here is one of expected measurements versus actual measurements. Parts have tolerances, the measurements have variances, etc.

Most rules are automatically handled by the coincidence mechanism and there are only about 100 rules. Robustness is dramatically improved and generality is much better. Efficiency is good since the system does most of the work, and constructability is much better.

The fault mode reverses the propagation of fault values and searches for the fault from the output of the system. De Kleer claims such an approach would reduce the rulebase to 25 rules, but admits that more would have to be added to allow for the backward propagation of constraints. The fault mode improves the performance according to the defined goals, but the extra code required for the backward propagation would add to the initial effort.

The last considered is the causal model, which infers about rules to fill in weak associations. This method also adds a mechanistic model which defines the physical layout and connections of the circuit. Here, as in the fault mode approach, much more work is required at the beginning to provide the mechanistic model database and the extra code needed for the inference mechanism.

Davis. While the work by (Davis, 1983 and 1984) is aimed primarily at digital electronic circuit analysis, there are a few fundamental

design ideas that may be applied to circuit analysis in general. They have taken two approaches to circuit diagnosis: functional diagnosis and structural (physical) diagnosis. Functional analysis is performed on a functional hierarchy of the circuit while structural analysis relates the physical components in a structural hierarchy.

First, a look at the functional picture. The device under test is separated into functional components and a hierarchy is built up by successively expanding the components into subcomponents. Each component is treated as a "black box" with given "ports" representing input and output connections to other external components. Diagnosis begins at the top of the functional hierarchy.

The structural hierarchy is developed in relation to the physical hardware. The device consists of a power supply, circuit board, etc., which, in turn consist of ICs, transistors, resistors, wire, etc.; ICs are also broken down to individual gates. A relation is then developed which connects individual components of the functional hierarchy to the structural hierarchy, showing the physical position of each functional component. That is, for example, **gate-1** is physically part of **IC2**, the power supply physically resides on **board-1**, and so on.

This relation is then used during diagnosis to identify other possible fault causes that are "functionally" unrelated but are "physically" related. For example, an unused gate on an IC may be damaged causing one or more gates on the same IC to malfunction; or, as is pointed out in their article, a gate on one IC may be bad, influencing voltage levels on other ICs thus causing the fault to appear in unrelated circuits.

Canton, et al. - IN-ATE/2. (Canton, 1984) describes one of the few analog circuit diagnosis systems available today. IN-ATE/2 uses a hierarchical fault-mode decomposition to provide the search base for test selection using the **gamma miniaverage method**. This search method uses cost of test, cost of component replacement, probability weighting, and resulting proximity to diagnosis as parameters in selecting the best test to perform next. Hierarchical fault-mode decomposition is a functional hierarchy where each successive level provides more detail of each functional unit. This is very similar in concept to the functional representation described by Chandrasekaran and Sembugamoorthy (to be looked at next). Such a representation permits the diagnostic tool to be more selective in its analysis and test selection because it is able to "see" what is directly connected or related to the current test point. The rules in IN-ATE/2 are also predicated by **preconditions** similar to PROSPECTOR; the rules have the form of **preconditions —> if —> then** where the preconditions must be met prior to testing the if clause. The preconditions specify exactly what state the unit under test must be in: previous test results, cabinet opened, voltage/signal applied to test points, etc. The cost associated with making a test is the value of actually performing the test plus the cost of meeting the preconditions not yet satisfied.

Chandrasekaran and Sembugamoorthy. (Chandrasekaran, 1984 and 1985) took a similar approach in their functional representation for diagnostic problem-solving systems. They developed a hierarchical structure that provides more detail deeper in the structure. Five different views are taken at each level: structure, function, behavior, generic knowledge, and assumptions.

The structure defines a module's physical connection to other modules; the function says what the module should do, while the behavior explains how; the generic knowledge provides general information pertinent to the module; and the assumptions are just that, assumptions about the state of the system in relation to the module. When a malfunction is being diagnosed, we start at the top of the hierarchy and descend the structure a layer at a time in an effort to isolate the failed component. The failed component itself may be further subdivided, in which case the search continues in that substructure until fault can be assigned. If, at any level, a module checks out satisfactorily, then no further search is made below it since a fault below it would imply the module itself had failed.

Milne. The technique suggested by (Milne, 1985) carries the diagnostic process a step further than simple functional analysis to include structural testing when functional testing fails. His proposal for functional behavior suggests failure can be assigned to individual components by viewing the expected vs. actual behavior over a specified time frame. He assigns responsibility for specific behavior during a time slice to one component; if the circuit fails in this responsibility, then the identified component is blamed. If, however, the test cannot be performed due to lack of signal, etc., then a set of structural rules are invoked to suggest possible causes such as a shorted or open component.

He describes the output of each component as a waveform, voltage level, etc. with sub-components of the output, based on time slices, described as sub-circuits. Each sub-component of the output is then

assigned to the component of the circuit responsible for that part of the output.

Milne's next step is to apply deep functional reasoning to automatically derive the output from low level definitions of individual electrical components. These "second principles", plus low level rules for waveform addition, permit the system to automatically derive the output for each component and assign it the responsibility for that waveform.

Summary of Approach

With the current knowledge reviewed in this chapter, it is now proper to outline the approach this thesis will take. To develop an expert system, it is necessary to identify the data required and a method of implementing the knowledge base. The characteristics chosen from the current literature for representation of the knowledge base are a hierarchical structure of the circuit such as that suggested by Davis, a functional relationship between modules as suggested by many of the papers read, and some deep understanding of individual components as discussed by Chandrasekaran, Sembugamoorthy and Milne in their papers.

Expert system building tools will be looked at closely along with the frame-based implementation accomplished by Ramsey to decide what type of implementation will be made.

Of course, all this will rely on the identification of the problem and what goals are to be accomplished. These are discussed in the next chapter.

III. Identification

The first step in developing an expert system is to identify who has a problem, what the problem is, what the goals of the system are to be, and what resources will be required.

Participants

In this project, some of the participants are obvious. Since the research is being sponsored by Rome Air Development Center in the interests of Air Force Logistics Command, RADC and AFLC will be declared primary participants. Warner Robins ALC is the specific participant within ALC that requested the study. Other primary participants include the students performing the research and the instructors supervising the thesis research. Secondary participants include all those providing technical assistance. Others involved are those that provide no input to the research but may realize some benefit from the results.

Problems

Lt Ramsey's thesis identified numerous problems associated with the hardware and software of the ATE at WRALC. (Ramsey, 1984: I-2 - I-4):

1. Old hardware
2. Inflexible and poorly documented programs
3. Failure to isolate to a single component
4. Failure to isolate to the correct component
5. Failure to use accessible existing circuit connections
6. Failure to use existing computer hardware
7. Difficult interfacing between the unit under test and the ATE
8. Little or no feedback to the operator

9. Failure to check all possible components

As mentioned earlier, based on interviews with WRALC technicians and on artificial intelligence applications, only three are of primary interest to this project.

1. Failure to isolate to a single component - Current procedures using ATE result in several components being listed as suspect; sometimes as many as 25 to 30 for the power supply board.

2. Failure to isolate to the correct component - Due to inaccurate testing, the ATE sometimes incorrectly passes bad components and indicates the board is usable.

3. Failure to check all possible components - The current software does not even consider some components of the circuit when running the tests.

Ramsey's thesis was an attempt to automate the testing procedures conducted by current ATE. As such, no consideration was given to internal test points not currently used by the ATE or the functional characteristics of the components of the unit under test; thus, all analysis is performed on the final output signals without the aid of internal, intermediate signals, or the specific effects of the submodules on the system's output signals. This forces testing to proceed at the component level in an attempt to assign fault based solely on the final result without regard to module functionality. When a large, complex circuit is involved, little can be done to isolate the problem beyond some sub-circuit which, in all likelihood, still contains far too many components for the resulting suspect-component list to be worthwhile.

First, consider the sample circuit in view of present ATE methods. Such a system would have rules such as:

IF +18v IS LOW
THEN T1, B1, C1 ARE SUSPECT.

IF -18v IS LOW
THEN T1, B1, C2 ARE SUSPECT.

IF +5v IS LOW
THEN T1, B2, +5vReg, C3, R1, R2 ARE SUSPECT.

IF +2.5v IS HIGH OR LOW
THEN T1, B2, +5vReg, C3, R1, R2 ARE SUSPECT.

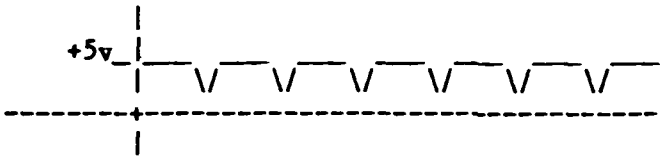
This obviously is of little help if the +2.5v source is wrong; then every component related to that part of the circuit is suspect, leaving the engineer to guess which part is really bad.

With such a large data base of rules, and without some sort of organization, it would be very easy to mislabel the responsible components for some test fault, or to overlook a necessary test. For example, had the test for **+2.5v HIGH** been left out, the board would be malfunctioning but would pass the testing procedure; or, had a test been left out altogether, subfunctions performed by the board would be missed completely.

An understanding of the functional behavior of components is essential to troubleshooting a circuit; without this knowledge, testing performed may be insufficient to properly isolate malfunctioning components. Also, having a functional structure imbedded within the system forces a complete analysis, preventing components from being overlooked in the testing procedures. Such a search would be more specific in assigning responsibility for given faults. Knowing the function expected of components will make testing more specific by isolating a given error to those components responsible for the incorrect behavior. One last benefit is that a structured design permits inserting and deleting

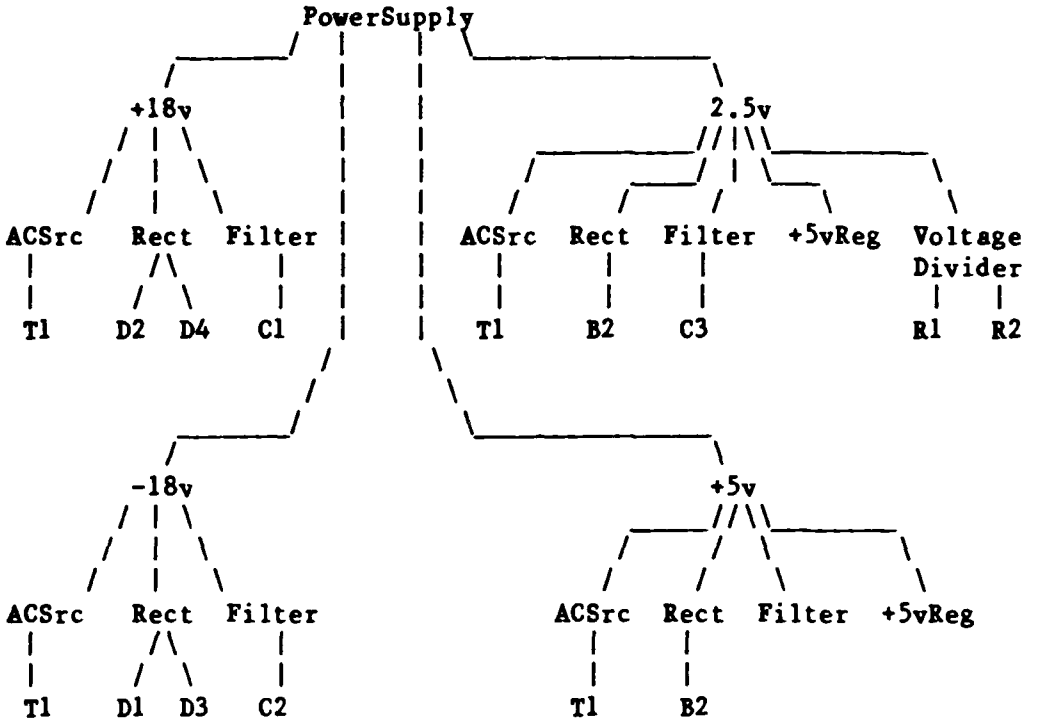
of specific routines more easily than in a system not structurally designed; modification is greatly simplified compared to the analysis required to find the correct place to insert new routines in an unstructured system.

Now, consider the sample circuit using a functional analysis. Suppose the +5v reference is low. According to the rules mentioned earlier, **T1**, **B2**, **+5vReg**, **C3**, **R1** and **R2** are suspect. This is essentially the 5 volt supply sub-circuit. Now look at the waveform between **+5v** and **ground**; if the waveform is like this,



then **B2**, **+5vReg**, **R1** and **R2** are functioning properly and the list of suspect components has been narrowed down to **T1** and **C3**. Measuring the waveform between **TP4** and **TP6** will tell if the voltage level from **T1** is sufficient, and, if so, then **C3** must be bad.

All this can be functionally defined as follows:



where, in the +18v portion, responsibilities of components are:

ACSrc - Supply 13 vrms AC sine wave
Rect - Full-wave rectification of +13v sine wave
Filter - Filter sine wave to provide +18vDC

Goals

The goals of this thesis effort are: to conduct a literature search and describe deficiencies in the present technology concerning electronic diagnosis, to analyze the requirements in relation to the technology available, and to implement a prototype of the diagnostic system to demonstrate the capabilities, limitations and practicality of the design.

The literature search is a sampling of several different styles to identify the strengths and weaknesses of each which will permit the combining of techniques. Such a combination should result in a system

with a collection of good qualities while offsetting some of the bad qualities of each technique by compensation from the others.

The analysis will result in the identification of the characteristics and rules required in the knowledge base necessary for the diagnostic process. The characteristics and rules will be kept separate so changes can be easily made to either as necessary.

The implementation will take place on the chosen hardware/software support system and will provide a basic diagnostic system capable of being examined for its performance. The performance characteristics to be measured include accuracy, ease of use, flexibility, and adaptability to other circuits.

Resources

Finding a system in which to implement this project requires that certain capabilities be considered. Provisions for a structural representation of the knowledge base, graphics capabilities, menu selection facilities, and a "lower level" language interface are among the considerations.

The system will rely heavily on a structural representation to perform the diagnostic procedures defined earlier. The input/output interrelationships of submodules of a circuit require the hierarchical relation which is provided by a structural representation. Also, testing will require either a rule system or interfacing with a lower level such as the Lisp language to perform the diagnostic testing.

Since Ramsey's thesis dealt primarily with the F-15 power supply card and the ATLAS code based on the test requirements document, his OPS5 and Frame KB implementations are not useful in the solutions of the

goals of this thesis. His further efforts with component history added to the knowledge base was also of no use here; as before, the problem exists of identifying the wrong component. The history may show a specific part has caused problems before and is probably at fault again when it is an entirely new problem this time.

To illustrate the insufficiency of the previous approach, consider the following interview with a WRALC technician which provided a helpful insight for the approach to solving the diagnostic problems. The functional description (Fig. 3) and circuit example (Fig. 4) obtained from them provide an example of the type of functional knowledge needed to implement a functional diagnostic system.

> Normally 3Ø rectified DC is regulated through A3Q1 out to pin 27 as 12 VDC. When 3Ø power is lost, a circuit breaker closes and applies 28 VDC to pin 47 from pin 48, which gets it from a battery. The 28 VDC on pin 47 is fed to A1Q1 through A1C4, as well as to the collector of A1Q3. At first, A1C4 acts as a short and the positive voltage turns on A1Q1, which turns off A1Q12, A1Q11, and A3Q6. The 12 VDC output is developed across the series combination of A1VR10, A1CR57, A1R45, and A1Q1. As A1C4 charges, A1Q1 cuts off, A1Q12, A1Q11, and A3Q6 come back on. The signal path for the 28 VDC is now from pin 47, through A3Q6, A1CR34, and A3Q1 which regulates it to 12 VDC, out to pin 27. A3Q6 y provides rough regulation down to 12 VDC and A3Q1 provides fine regulation.

Figure 3: Functional Description

Much of this knowledge is not known to the ATLAS code or Ramsey's implementation of it. Also, with all the interaction within this subcircuit, historical data may be misleading at the least; there may be several causes that result in the same output, and the historical data would just flag all components having a failure history.

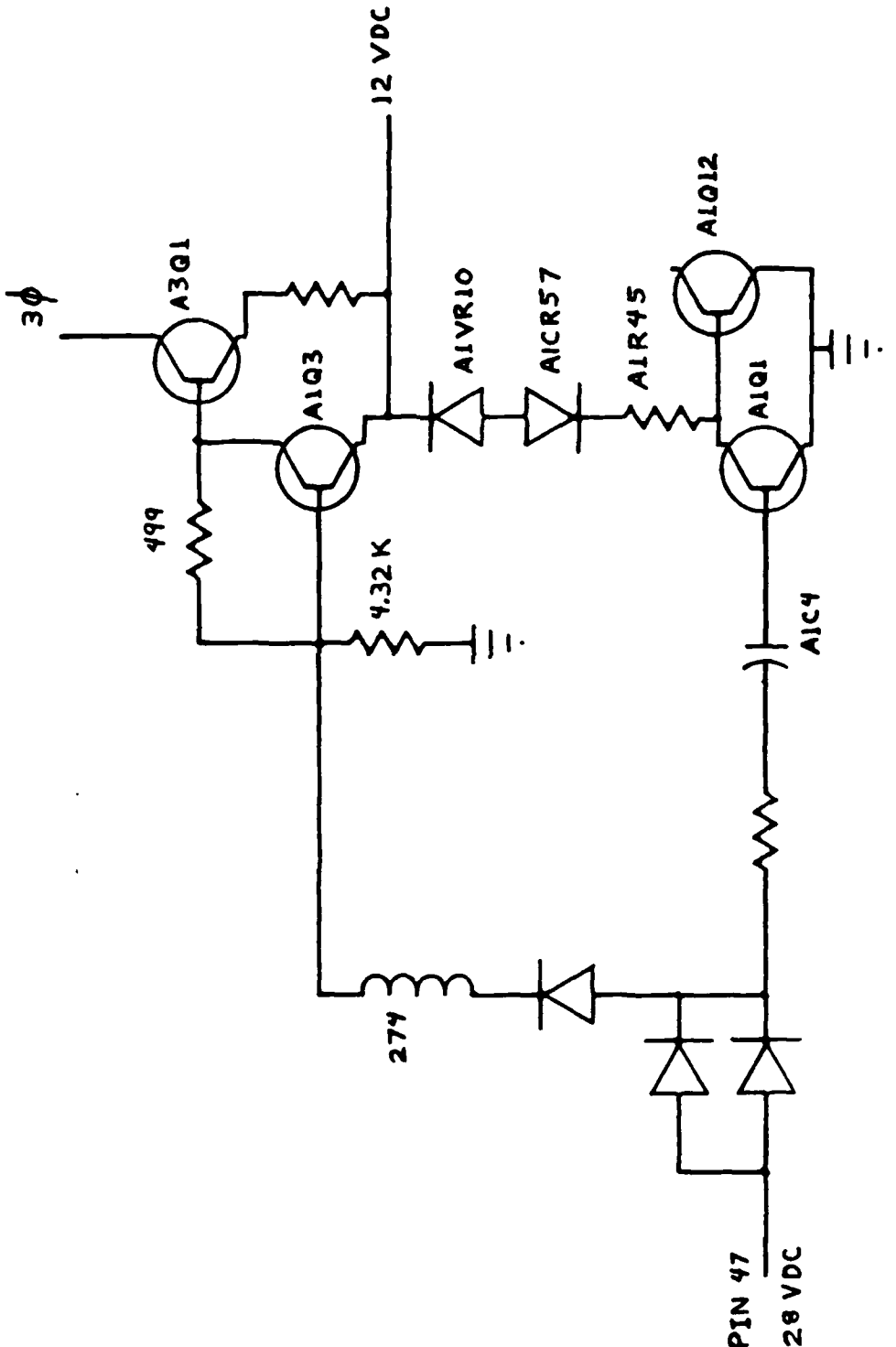


Figure 4: Circuit Example

IV. Preliminary Design

This project will define a system as a functional structure. The system will consist of a functional description, an expected behavior, a list of submodules with an assignment of responsibility for some portion of the behavior, and a set of input and output waveforms for the module indicating good, bad or unknown behavior of the module. An empty list of submodules implies that either the current module has sole responsibility for behavior and must be replaced as a single component, or the subcomponents have no internal test points and a functional rule must be invoked. Since more than one submodule may be responsible for a given behavior, the search must activate the tests for each submodule involved to ensure a complete analysis is made to identify all bad components.

The function is a description of what the module does. The behavior is a step-by-step description of how the module performs its function and includes a list of submodules responsible for each step of the description. The lowest level submodules will also contain a list of input/output waveform pairs which, when matched with the test results, will specify whether the module's condition is good, bad, or unknown; there will be a functional rule which is invoked when the module is classified as "bad" to allow a functional analysis to determine which components within the module are at fault. The general format of the structure definition is:

```
functional_structure
     function
     behavior_1 by sub_module_1, sub_module_2
     behavior_2 by sub_module_2
     in lowest submodules only,
          waveforms
          functional.rule/components
     in all but the lowest submodules,
          parts
```

Each submodule is then described in a similar manner until the complete circuit has been defined.

Looking at a part of the sample circuit's definition:

```
PowerSupply
     Function:  Provide +18vdc, -18vdc, +5vdc, and +2.5vdc
     Submodule functions:  provide +18vdc by +18
                                   -18vdc by -18
                                   +5vdc by +5
                                   +2.5vdc by +2.5
     Parts:  +18, -18, +5, +2.5

+18
     Function:  Provide +18vdc
     Submodule functions:  Receive 36v p-p AC by +18ACSrc
                           Rectify AC by +18Rect
                           Filter +18vdc by +18Filt
     Parts:  +18ACSrc, +18Rect, +18Filt

+18Rect
     Function:  Full-Wave rectify AC input
     Functional.rule/components:
          fullwave.rectifier (D2 D4)
     Waveforms -
          Input              Output             Function
          Flatwave @ 0vdc    Flatwave @ 0vdc      unkn
          Flatwave @ 0vdc    Sinewave              bad
          Halfwave           Sinewave              bad
          Fullwave           Sinewave             good
```

Considerations in the diagnostic process may now be viewed. When determining whether a component is good or bad, its function must be stated, then an answer is needed: "Is the component performing its function?". There are different ways to specify the function being performed by a good component. Mathematics could be used but would be too complex for people to understand (except perhaps the senior engi-

neers having the extensive theoretical background). On the other hand, if the descriptions are simple enough for anyone to understand, then they will probably be too vague to be of any practical use. Therefore, a middle ground must be found that is simple enough for the technician to understand but complex enough to be of value. Thus, the technician's terms, the "standard" language used in the electronics field by amateurs and professionals alike, have been chosen.

In an analog system, frequencies, voltage levels, etc. are constantly changing; in digital circuits, on the other hand, everything runs off a given clocked pulse, and voltages are all typically at just two or three specified levels. Any real discussion about the function of a component in an analog circuit must consider what that component does to the input signal; the selection of waveforms for the functional representation was based on this consideration. In analog electronics, the characteristics of the signal in a circuit provide most of the information needed to determine the condition of the components of the circuit.

Testing will proceed in one of two ways. The first option is to perform a search, such as would be done at a work bench where the unit under test may be opened up and all test points may be used. The search for bad components starts out at the highest level of abstraction, constantly narrowing the fault down to more specific sublevels. The second method for troubleshooting is the approach taken by ATE systems, where a limited number of test points, usually only the output signals, are available. It is necessary in this approach to utilize functional knowledge of the subcomponents in determining which one(s) are causing the

fault seen at the output point. This too is a routine that trims the search tree by dropping down each level of abstraction until specific components can be identified.

The plan of this thesis is to model the actual engineering design process of the equipment to be tested. When an engineer sits down to lay out a circuit, he/she starts out with an overall plan -- design a power supply. To build one, an AC source, a voltage rectifier, a filter, a voltage regulator, etc., are needed. Within each of these a further breakdown defines what is needed to perform its sublevel functions. For example, the voltage regulator may consist of a single regulator component or may comprise several transistors, resistors and capacitors. This same procedure is used whether designing a power supply or a television receiver.

By modeling this procedure to define the system, the definition will be as straightforward as the system design and can be written at the same time. If changes are made to the circuit, the same changes can be accomplished by adding or removing portions of the definition. When the design is complete, the circuit definition will be complete and ready to insert into the diagnosis system.

Such a modeling procedure applies to other areas as well. For example, artificial intelligence approaches to software engineering attempt to ease software maintenance by making the design decisions explicit. In any area, if maintenance is considered during the design, then steps can be taken to use the design decisions in the development of the maintenance procedures.

Testing using the first method described earlier is accomplished by starting at the top and pruning the hierarchical tree by verifying

whether the submodules are functioning properly. If any given submodule is not working, its submodules are examined, and so on, until no further subdivision is defined. Failing modules having no submodules are then replaced and testing is again performed to validate the unit under test.

The general procedure for testing with the first method is as follows:

```
test_module(root)
```

where test_module is defined as:

```
if function not correct
  then
    for each behavior[i] do
      if behavior[i] not correct
        then
          if sub_module_list not empty
            then
              for each sub_module do
                test_module(sub_module)
            else
          else
            replace module
```

Testing by the second method prompts the user for the output waveform of the final stage and works its way backward matching output waves to possible input waves in each subcomponent and determining whether a subcomponent is functioning properly based on the input/output waveforms. The input/output waveform pairs define the function of the subcomponent under different conditions. In any case where a component is defined as bad, it can then be broken down into its own subcomponents where the initial input and final output waveforms are known and a search can be made to further restrict the selection of suspect components.

This is where the "functional reasoning" comes in. The function of a component under differing conditions is represented by the possible

combinations of input and output waveforms. Judging a component based on its effect on the signal is effectively saying a component is good or bad based on its function. A diode's function is to pass current in one direction and to block it in the other direction. If its selected input/output waveform pair shows that a negative pulse is present at the output, given a sine wave at the input, then it can be said that the diode is not functioning properly.

Knowing a module's input and output waveforms, a functional rule can be invoked to further restrict the suspect-component list. If, in the above example, the +18 rectifier has a sine wave input and a half-wave output, then fault can be assigned to a specific diode depending on which half of the sinewave is missing. If the positive half is present and the negative half is missing, then D4 is known to be at fault.

Any component can be analyzed in such a way, whether it is a simple single-part component or a complex subcircuit. Obviously, a more complex component will require a larger, possibly less reliable, functional analysis. A large circuit may not have any internal test points to allow diagnosis according to the earlier input/output waveform test procedures, and be too complex to permit a detailed analysis, resulting in a large list of suspect components. However, such circumstances are still preferable to building a functional reasoner for the entire circuit without benefit of any logical subunits.

The reliability of the suspect component list depends on the functional rule being invoked; the rule may be able to identify specific components or may only be able to suggest that one or more of the list of components is bad. The same is true for multiple-part faults; if

possible, the functional rule may be able to identify several components at fault at one time.

The diagnostic system can identify multiple failures if the waveforms provided allow for failed component input/output; for example, if the output of +5 VDC is a filtered halfwave with a peak of +8 volts, then the output of the rectifier indicates it is bad (a halfwave instead of a fullwave signal) and the regulator output indicates it is bad since the signal is peaking at +8 volts instead of +5 volts.

V. Detailed Design and Implementation

Certain capabilities had to be considered when looking for a system to be used to implement this project. Provisions for a structural representation of the knowledge base, graphics capabilities, menu selection facilities, and a "lower level" language interface were among the considerations.

The system will rely heavily on a structural representation to perform the diagnostic procedures defined earlier. The input/output interrelationship of submodules and the **bench**-type testing require the hierarchical relation which is provided by a structural representation. Also, the **bench**-type testing will require either a rule system or interfacing with a lower level such as the **lisp** language to perform the diagnostic testing.

Graphics are needed to display associated waveforms when running ATE-type testing, and menus are needed to choose the appropriate waveform.

One of the systems available today that provides most of the above requirements is KEE. KEE provides a structural representation of the knowledge base, has window facilities to support menus and other display functions, contains **activeimages** for graphic displays, and permits normal use of **lisp** while KEE itself is active. In addition, KEE provides capabilities for tool prototyping which could be used in a later project to generate ATLAS code for the WRALC analysts. For these reasons, KEE has been selected to implement the diagnostic system.

The system is organized in four parts: circuit representation, waveforms, diagnostic procedures, and functional rules. Of these, only

the circuit representation and functional rules are changed to implement a diagnosis system for a different circuit; the waveforms can be added to as needed, and the diagnostic procedures remain unchanged.

To apply KEE to circuit diagnosis, *all components and subcomponents* will be defined as members of the class **circuits**. Members of this class have slots for a function definition, a subfunction-subcomponent list, a functional rule identifier, waveforms, and a parts list. The subfunction-subcomponent list has the form

```
(subfunction (subcomponent (testpoints))
             (subcomponent (testpoints))
              .
              .
              .
)
```

for each subfunction. Notice that there may be more than one subcomponent responsible for a given subfunction. **(testpoints)** is an optional list of relevant test or connection points for the subcomponent. The function definition is just a narrative statement about the function of the component and is of the form **(functional description)**.

The functional rule identifier slot will have the name of the appropriate functional rule and a list of responsible parts, if the module has no further subdivisions defined in the parts list. If the diagnosis determines that this module is bad, then the functional rule is executed with the input wave, the output wave, and the list of responsible parts as parameters. It is then the responsibility of the rule to determine what parts within the component are at fault.

The **waveforms** slot consists of a list of possible output waveforms with their respective possible input waveforms and a flag indicating whether the input-output transition is a good, bad or unknown transition

for this component. The list has the form

```
((output-waveform characteristics)
        (input-waveform characteristics flag)
        (input-waveform characteristics flag)
          .
          .
          .
)
```

for each output waveform. There may be more than one input that may cause a given output, and the flag, as mentioned above, is one of **g**, **b** or **u** (good, bad or unknown) indicating the performance of the component in translating the given input form into the given output form. The **characteristics** field is a narrative concerning the characteristics of the given waveform such as voltage levels, frequency, etc. More will be said about waveforms in the following discussion of the implementation of waveforms in the system and in the description of the diagnostic procedures.

The parts list of a component is simply a list of the subcomponents of the given component and serves the following purposes. First, it is used to show the hierarchical relationship among the various components. There are two functions taking advantage of this relationship: **graphkb**, and **graph**. **graphkb** has no parameters and will display the entire system graph in the current KEE output window. **graph** receives one parameter, a component, and displays the substructure of the system graph having the given component as its root node.

Waveforms are implemented as members of the class **waves**. Each form is a bitmap associated with a variable name. Each member of **waves** has a variable name assigned to the **form** slot identifying the associated waveform. The waveform names used in the circuit representation are the member names defined here. Methods are defined within **waves** for creat-

ing and displaying waveforms. Each waveform is created using **editbm**, a part of KEE's activeimages package, and displayed with a call to **bitblt***. The methods can be invoked by any function wishing to display or create a waveform.

The **bench**-test procedures are implemented in lisp with a depth-first search performed starting at a given component, expanding any component identified as "not functioning" and pruning the search by not expanding functioning components. At each component, the function is presented to the user, one subfunction at a time, to determine whether it is being performed. The value returned from the call to **bench.diagnose** is a list of bad components.

The **fault**-test procedures (designed from the **ATE**-type tests with functional rules added) are built up from the window and graphics provisions, and with lisp to create a menu of selectable waveforms. Each identified output waveform is matched with corresponding input waveforms for the module; these are then presented as the possible output waveforms from the next module back. At the end of the list of submodules, that is, the first submodule of the given module to be tested, the input waveforms are provided to the user for verification of the functioning of the first submodule.

Illustrative Example

Now consider an example of the two different test methods described above. Assume diode D4 is open and see how the test routines handle the situation.

With the **bench** method, a series of questions is asked of the user and the returned value is a list of suspect components. The following

is the dialog conducted by the system (User responses are underlined):

Does PowerSupply provide +18vDC? n

Does +18 receive 36v p-p AC? y

Does +18 rectify AC? n

Does +18 Rectifier pass +AC sinewave? y

Does +18 Rectifier rectify -AC sinewave? n

Does +18 filter +18vDC? y (Note: It does filter the halfwave that it is receiving.)

Does Powersupply provide -18vDC? y

Does PowerSupply provide +5vDC? y

Does PowerSupply provide +2.5vDC? y

The list of suspect components is: (D4)

As can be seen, the system does identify diode D4 as bad.

Using the fault testing procedure, we ask the system to diagnose the +18 volt power supply. It starts out with a menu selection of the possible output waveforms from the +18 Filter and we select the Filtered Halfwave form. This identifies what corresponding input waveforms are possible of which there is only one, a rectified halfwave. The system then looks at the +18 Rectifier with an output rectified halfwave as a reference. Again, there is only one combination and the +18 ACSrc output, equivalent to the +18 Rectifier input, is a sinewave. This shows that the +18 Rectifier is bad and the +18 ACSrc is good. Since the +18 Rectifier is not further broken down, its functional rule is invoked. The fullwave.rectifier rule then compares the input and output waveforms to identify which diode is bad. Since the positive half of the sinewave is present at the output and the negative half is missing, the rule determines that D4 is bad and returns it as the faulted component.

Extension to De Kleer's Sample Circuit

The next step in the development of this thesis is to implement a more complex circuit to see if the technique still works. The circuit shown in Fig 5 is one used in De Kleer's work and would require hundreds of rules to be implemented on the present ATE system. Based on functional reasoning, De Kleer has speculated that there would only be 25 rules required.

This circuit was broken down functionally and implemented similar to the earlier example with similar results. Looking at a sample run of the **fault** testing will show the similarities.

Entering (diagnose 'voltage-control 'fault) in the lisp window results in a menu of waveforms to choose from. Selecting the filtered halfwave 6 volt peak waveform as the output from **VC-RECT2** causes the diagnostic system to trace backward to **VC-VOLT-CONTROL** which is good according to the waveform pair, then to **VC-FILT2** which may be good or bad depending on its input (the output from **VC-RANGE-SELECTOR**). If the filtered halfwave 12 volt peak output is chosen, then **VC-FILT2** is bad and the trace continues. **VC-REG** is found to be bad since the only possible output is the filtered halfwave 36 volt peak; its input is the filtered halfwave 36 volt peak from **VC-FILT1** which appears to be functioning properly. Its input is from **VC-RECT1** which is found to be good, and last, the output from **VC-ACSRC** shows it to be good.

Looking at the above sample run, by selecting only two waveforms, the system has been able to identify **VC-FILT2** and **VC-REG** as the components responsible for the malfunctioning power supply. The functional rules for these two are then invoked. The rule for **VC-FILT2** identifies **C2** as bad, and the rule for **VC-REG**, being less definite, identifies "One

or both of" D4 and D5 as bad.

Other sample runs using **bench** and **fault** testing are included in Appendix E.

The effort required to implement De Kleer's sample circuit was minimal and consisted of subdividing the circuit and identifying each portion's function, representing the functions in waveform pairs, and adding the functional-cause code for the new subcircuits (such as the pi-rc filter and the halfwave rectifier). A more complex subcircuit such as the current-regulation portion of this circuit (composed of the six transistors and associated capacitors and resistors) requires a more detailed analysis by the design engineer to provide the necessary data for the software engineer to follow when writing the functional rules.

Application to the F-15 Power Supply

Working closely with the engineers at WRALC should provide the necessary information to implement a diagnostic system for the F-15 power supply board. It would be a much larger project than those used here as examples, and may not be able to fill all the gaps since new test points cannct be added; however, this diagnostic package would provide more than the current ATLAS testing does. This system will force the structuring of the problem, provide some of the functional testing, and can fill in the missing portions with causal rules similar to the ATLAS code.

Since the F-15 power supply is a three-phase circuit, much of it can be divided into three parts, one for each phase. Each subsection is then at about the same level of complexity as De Kleer's sample circuit.

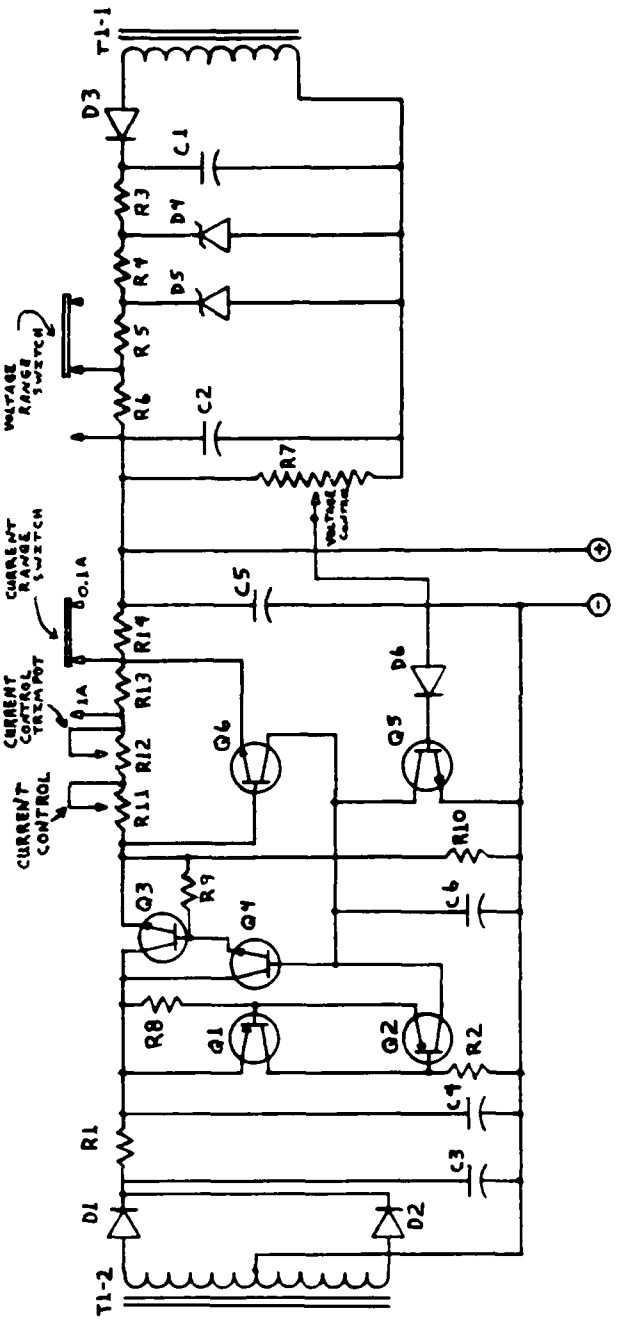


Figure 5: De Kleer's Sample Circuit

The other portions of the circuit are different functional units and can be treated separately.

The circuit is functionally decomposed and tested at the subunit level. The system effectively eliminates many components simply by identifying the function(s) being performed properly. Then the functional rules take over to apply the engineer's knowledge to isolate the bad components.

Analysis

At best, the functional analysis tool developed in this thesis will provide a comprehensive diagnostic aid for the field. At worst, it will still provide the structural format plus the current test procedures included as functional rules. This implies that this approach is better than the current system since it will provide more than just the ATLAS test procedures that are used now. The work involved in implementing a circuit diagnostic tool with this package should be less than that required to write the ATLAS test procedures since the structure is defined and it is less likely that anything would be mislabeled or forgotten. Less time would be spent verifying the tool, allowing more time to be spent analyzing the test requirements. The end result is a more complete, more accurate test package in the same amount of time.

There were three problems identified by WRALC engineers as being of primary interest to this artificial intelligence project: failure to isolate to a single component, failure to isolate to the correct component, and failure to check all possible components.

Assuming the knowledge base for the given circuit is properly designed and stated, this thesis project has solved the three problems for

the analysis of at least some circuits.

The use of waveform analysis for functional isolation of faults, and functional rules to analyze components identified as potentially bad, permits more accurate identification of components at fault, thus solving the first problem.

If the engineer's design specifications are accurate and fully implemented, then all possible components will be tested within a functional unit and good and bad components will be properly identified.

The written analysis of the sample circuit given earlier (Figs 3 and 4) presents the final test of the implementation of this thesis. The analysis proceeds in a step-by-step description of the events that occur when the circuit loses the 3Ø power. The overall function of the circuit can be defined in the terms necessary for this diagnostic system; however, the time-critical events, such as "at first, A1C4 acts as a short (and) turns on A1Q1 ... as A1C4 charges, A1Q1 cuts off", are not definable in the present system. This, other one-time events, and some periodic events need further time-slice analysis which is not currently supported.

Without internal test points, subcircuits such as the current-control regulator portion of De Kleer's sample circuit contain too many interdependencies for effective analysis; in other words, the system may not be able to say any more than "The current-control regulator is bad". Perhaps, at this point, it may be helpful to return to the statistical data base suggested in Ramsey's thesis. A probability of fault may be better than nothing.

Another minor limitation of this diagnostic system is the limited space available in the selection menu for comments about the waveforms.

A simple solution to this would be to place all expected waveforms in a printed form with full comments and just have an indexed reference in the comment space on the menu.

The use of the product of this thesis is a straight-forward approach when dealing with linear analog circuits. The introduction of feedback loops within the circuit still presents some problems. Other circuits not considered at the waveform analysis level are those containing subcomponents with more than one input, or those containing no inputs such as oscillators.

The problem of multiple inputs should be a feasible addition to the current system; however, feedback presents other questions such as, "How many times should the loop be traversed before making a good/bad/unknown decision about the subcircuit. Feedback loops usually include multiple inputs so this problem will need to be resolved prior to the feedback tracing problem.

Again, a method of time-slicing may provide some answers. Any looping traces conducted will need to consider discrete events which can be represented by a time-slice analysis.

Overall, this thesis accomplished its objectives by defining an approach to solving the problems identified earlier, and implementing a testable model of the approach to show its feasibility.

VI. Conclusions and Recommendations

Conclusions

This thesis has addressed representative problems faced by the Air Force today in testing circuit boards removed from operational systems. It has looked at the current knowledge in the field of circuit diagnosis using artificial intelligence, and defined a functional system of circuit analysis to aid in circuit fault testing.

This project is an attempt to provide more accurate specification of bad components than is available currently in the Air Force using ATE. The procedures outlined above use a functional description to isolate the problem into successively more restricted sub-circuits within the circuit under test instead of applying the structural testing currently performed. The testing defined by this system is best set up by individuals knowledgeable about the circuit to be tested since the engineer's knowledge concerning the circuit is required. Field personnel knowing little or nothing about the circuit can use the system since the diagnostic procedure is menu driven (in the **fault** testing mode), and question/answer driven (in **fault** and **bench** modes), both of which will lead the user from step to step requiring him/her to only make the required measurements, and answer the questions or select from a menu.

There are several restrictions in this preliminary implementation. The circuit to be tested must be decomposed by hand; the subcomponent lists must be hand massaged into the correct order for the testing to progress properly; the functional rules must be written in Lisp. Some of these restrictions may be considered in later theses or in other research. There are people actively investigating other approaches to

designed to relax the restrictions, such as the work by Chandrasekaran. He is pursuing the development of a knowledge base language that supports functional descriptions of circuits at the design stage, from which diagnostic rules may be deduced (Chandrasekaran, 1984). Other, less serious, restrictions may be identified and worked around as experience progresses.

Extending the diagnostic system to another circuit involves defining the component layers of the circuit and identifying the function and input/output waveform matchings related to each. If the circuit is being developed, the definitions may be entered during the process as explained earlier. With a circuit already designed, the process of defining it to the system may be more difficult; analysis may be required if the circuit is an unfamiliar one, and considerable bench testing may be needed to identify the effects of a component on a given input signal.

Any application of this diagnostic system to other circuits (power supplies and other linear circuits) should only require the entry of the circuit data. Other functional rules may need to be added for specialized circuits and for general circuits that have not yet been encountered. The system itself may need other waveforms defined but these would be one-time additions to the system as further waveforms are found to be required.

Recommendations

There is still much to be done. Little consideration has yet been given circuits involving feedback. Automatic interfacing to ATE equipment is a possibility. A more refined interface to make it more appeal-

ing to the users would be helpful. Also, KEE provides more refined capabilities which could be implemented for a more responsive system.

Circuits with feedback control, oscillators, multivibrators, etc. need to be looked at more carefully. Any circuit where the output somehow affects its own input may be difficult to represent as a collection of submodules below the module containing the complete feedback circuit. The development and use of a circuit simulator may prove beneficial. A circuit containing feedback could be passed to the simulator with the initial input and output waves to analyze the performance of the components within the circuit. It would then be the simulator's responsibility to recommend faulted components.

There are two types of interfacing possible to ATE equipment. One would be to have the diagnostic system directly interact with the ATE obtaining the answers needed to identify bad components. The other would be to use the diagnostic system as a tool in guiding the development of the ATE software, selecting what tests need be performed, and when, and using the system to create the test routines. In effect, the system would be used to write the ATLAS code required by the ATE.

Ergonomic considerations could be made to improve "user friendliness". In the AI aspects of user interfacing, research into creating the database for a given circuit from its schematic diagram automatically would be a great plus. Such a system would require considerable knowledge about the function of individual components in any given situation; at present, the required data is input by the human engineer.

KEE has other capabilities that were not used in this thesis due to a lack of in-depth knowledge concerning the implementation of KEE. It

supports multiple knowledge bases which could be used to separate the waveforms from the rest of the circuit description, thus permitting the creation of a new system without copying the basic form every time to include the waveforms. Window scrolling could be implemented instead of the software routines currently used when there are more than six waveforms to be displayed in the diagnostic routine.

BIBLIOGRAPHY

Cantone, Richard R., et al., "IN-ATE/2: Interpretating High-Level Fault Modes", Automated Reasoning Corporation.

-----, IN-ATE/2 User's Guide, Automated Reasoning Corporation, New York, New York, 1984.

-----, "Model-Based Probabilistic Reasoning for Electronics Troubleshooting", Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop, AF Systems Command, AFHRL, June 1984.

Chandrasekaran, B., Rob Milne, "Special Section on Reasoning About Structure, Behavior and Function", SIGART Newsletter, July 1985.

Chandrasekaran, B., V. Sembugamoorthy, "Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems", Cognitive Science, 1984.

-----, "A Representation for the Functioning of Devices that Supports Compilation of Expert Problem Solving Structures", Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop, AF Systems Command, AFHRL, June 1984.

Davis, Randall, "Diagnosis Via Causal Reasoning: Paths of Interaction and the Locality Principle", Artificial Intelligence: Proceedings of the Joint Services Workshop, Denver, Colorado, October 1983.

Davis, Randall, et al., "Diagnosis Based on Description of Structure and Function", Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop, AF Systems Command, AFHRL, June 1984.

De Kleer, Johan, "How Circuits Work", Artificial Intelligence, 24, 1984.

-----, "AI Approaches to Troubleshooting", Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop, AF Systems Command, AFHRL, June 1984.

Fikes, Richard, Tom Kehler, "The Role of Frame-Based Representation in Reasoning", Communications of the ACM, 28, No. 9, September 1985.

Hayes-Roth, Frederick, et al., Building Expert Systems, Addison-Wesley, Reading, MA, 1983.

Milne, Robert, "A Framework for Electronic Diagnosis Expert Systems", Air Force Institute of Technology, Department of Electrical Engineering, 1984.

-----, "Depot Level Problems in the Testing of Printed Circuit Boards", Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop, AF Systems Command, AFHRL, June 1984.

Milne, Robert, "Functional Reasoning for Fault Diagnosis Expert Systems", Applications of Artificial Intelligence II: Proceedings of SPIE - The International Society for Optical Engineering, 548, John F. Gilmore, Editor, Arlington, VA, April 1985.

-----, "Fault Diagnosis Through Responsibility", International Joint Conference on Artificial Intelligence - 1985, August 1985.

Ramsey, James E., Jr., "Diagnosis: Using Automatic Test Equipment and an Artificial Intelligence Expert System", Air Force Institute of Technology, December 1984.

Stallman, Richard M. and Gerald J. Sussman, "Problem Solving About Electrical Circuits", Artificial Intelligence, an MIT Perspective, Volume 1, MIT Press, Cambridge, MA, 1979.

Appendix A: Component Descriptions

Diodes

Diodes perform the basic function of passing current in one direction (while forward biased) and blocking it in the other (while reverse biased). In a DC circuit, the diode will act as a piece of wire if forward-biased and as an open circuit if reverse-biased; in an AC circuit, the diode's performance depends on the input signal. Only that portion of the signal that is negative will not pass through the diode. When considering the 60Hz AC found in typical household electricity, half the sinewave is passed, and half is blocked; this is known as halfwave rectification. In fullwave rectifiers, the sinewave is fed to another diode in the circuit in a way that provides the "negative" half as a second positive halfwave at the output, thus providing a total of two positive halfwave outputs from one positive and one negative halfwave input.

Resistors

Resistors provide a path for the flow of electricity while resisting that flow to some degree depending on the value of the resistor. The waveshape input does not change, however, its voltage level does. If the resistor is in series with another load in the circuit, the waveform's voltage will drop from one side of the resistor to the other providing a reduced voltage to the load. If the resistor is in parallel with another load, however, the same waveshape and voltage level are input to the resistor and the load, and the voltage drop across the resistor will be the total voltage of the waveform.

Capacitors

The performance of a capacitor is a little more difficult to explain. In series with another load, a capacitor passes alternating signals and blocks DC signals, both without regard to voltage levels; also, the lower the frequency of the AC signal, the more resistance a capacitor has in response to it, thus acting something like a resistor the value of which is dependent upon the frequency of the AC signal. A DC signal applied to a capacitor and load in parallel will be totally consumed by the load since DC will not pass through a capacitor. With an AC signal applied, a capacitor has two effects: first, an AC signal will pass through the capacitor; and second, as the positive halfcycle of a sinewave builds, the positive plate of a capacitor charges (causing a negative charge on the negative plate, thus appearing to 'short out' the alternating current). When it starts to fall off, the positive plate appears more positive than the signal and will discharge, trying to stabilize the voltage level applied to the load. The same effect would appear during the fall/rise of the negative halfcycle. If, however, there is no negative halfcycle, such as the output of a half or full wave rectifier, the next rising halfcycle will recharge the capacitor's positive plate and start the cycle over again, thus "filtering" the signal to the load into an almost DC signal. The larger the value of the capacitor, the less ripple and hence the purer the DC signal.

Transformers

DC will not cross a transformer, only a varying signal will pass from the primary winding to the secondary winding of a transformer. The rise of current in the primary winding creates an increasing magnetic

field which induces a voltage in the secondary winding in one direction. As the current in the primary winding falls, the magnetic field collapses which induces a voltage in the secondary winding in the opposite direction. As can be readily seen, an alternating signal whose voltage level never falls negative is converted to one whose waveform is divided by the 0-volt line, half positive and half negative. The voltage at the output is related to the input voltage and the ratio of input windings to output windings.

Switches, Fuses

When a switch is closed, it and a fuse serve the same function, to provide a direct connection from the input point to the output point. When a switch is opened, the direct connection is removed. Fuses only break the circuit when blown and serve to limit the current drawn by the circuit containing them.

Transistors

Transistors provide a variable control for current flow between the emitter and collector. The current flow is controlled by the third pin of a transistor, the base. Current will only flow in one direction between the emitter and collector of a transistor; in an NPN transistor, current goes from emitter to collector; in a PNP transistor, current goes from collector to emitter. Typically, a DC voltage is connected across the emitter and collector of a transistor, and a signal is applied to the base. The rise and fall of the signal on the base is then reflected, either directly or inversely, in the current flow between the emitter and collector. The actual appearance of the input waveform on the EC circuit depends on the bias voltage applied to the base and the

voltage levels of the base's input waveform. Some clipping of the signal may occur if the cutoff voltages are reached and the EC circuit limits or cuts off the current flow.

Inductors

Coils readily pass DC but present a resistance to the passage of AC. The higher the frequency of the signal, the more resistance a coil presents. Connected between some point in a circuit and ground, a coil will act as a direct short for DC and low frequency signals, and a resistance for high frequency signals. A coil in series in a signal path will pass DC and low frequency signals and block high frequency signals. A coil in parallel with a load becomes a high-pass filter (passing high frequencies to the load and shorting out low frequencies), and a coil in series with a load is a low-pass filter (passing low frequencies and DC to the load and blocking high frequencies).

Appendix B: Diagnostics Source

```
;;; -*- Mode:LISP; Package: KEE; Base:10. -*-

;*****************************************************************************
;
;     Filename:  DIAG-LOAD.LISP
;     Version:   1.0
;     Date:      4 Feb 86
;
;     Project:   Functional Diagnostic System
;
;     Author:    Don Wunz
;
;     Description:
;          This file loads the diagnostic routines and the functional
;          rules, then sets the current knowledge base and displays
;          its graph in the current (normally left) KEE diaplay window.
;
;*****************************************************************************

(load 'diagnose.lisp)
(load 'funct-cause.lisp)
(set.kb nil)
(graphkb)
```

```
;;; -*- Mode:LISP; Package: KEE; Base:10. -*-

;*****************************************************************************
;
;     Filename:  DIAGNOSE.LISP
;     Version:   1.0
;     Date:      4 Feb 86
;
;     Project:   Functional Diagnostic System
;
;     Author:    Don Wunz
;
;     Description:
;          This file contains the diagnostic routines responsible for
;          diagnosing through either the bench-diagnosis, the
;          ate-diagnosis, or the functional-fault-diagnosis.
;
;     Contents:
;          set.kb
;          graph
;          graphkb
;          make.wave
;          *wave-list* and associated tv:add-typeout-item-type functions
;          *menu-select*
;          *sensitive-window*
;          diagnose
;          bench.diagnose
;          set.component.list
;          test.subcomponent
;          fail
;          show.list
;          test.ckt
;          get.parts
;          find.components
;          get.input.waveforms
;          get.output.waveforms
;          setup
;          setup.b
;          setup.a
;          show.components
;          func.diagnose
;          column.list
;
;*****************************************************************************
```

```
;*****************************************************************************
;  set.kb is executed during the initial load to set the current
;  working knowledge base, and can be run at any time to change the
;  current knowledge base when changing over to another system to be
;  diagnosed.  set.kb has one parameter, the knowledge base to be
;  selected as the current one.
;
(defun set.kb (kb)
  (setq *kb* (kbreference kb)))


;*****************************************************************************
;  graph will display the subgraph with the given component as the
;  root.  The component must reside in the current knowledge base.
;  The display is to the current KEE output window.
;
(defun graph (component)
  (slot.graph *kb* component 'parts nil nil '(horizontal)))


;*****************************************************************************
;  graphkb will display the current knowledge base in the current
;  KEE output window.
;
(defun graphkb ()
  (slot.graph.kb *kb* 'parts nil nil '(horizontal)))
```

```
;*************************************************************************
;  make.wave is used to create a new waveform to be added to the
;  knowledge base.  The one parameter to it is either nil which starts
;  with a blank bitmap, or the name of a current waveform to be used
;  as a starting form.  Note, the bitmap name given must be the name
;  as found in the FORM slot of the waveform in the knowledge base.
;  To use this function, you must setq a new bitmap name to make.wave
;  such as the following:
;       (setq newwave (make.wave fullwave))
;  then, add it to the knowledge base as instructed, and last, create
;  a member under WAVE with the FORM slot equal to the new bitmap name.
;
(defun make.wave (bm.name2)
  (prog (bm.name1)
        (cond
          (bm.name2
           (setq bm.name1 (bitmapcopy bm.name2)))
          (t
           (setq bm.name1 (make-bitmap 100 100))))
        (editbm bm.name1)
        (terpri)
        (terpri)
        (princ '|Enter |)
        (terpri)
        (princ '|    (add.bitmap.to.kb 'bm.name 'kb.name) |)
        (terpri)
        (princ '|to save the bitmap with your knowledge base.|)
        (terpri)
        (princ '|Then, create a new member of WAVE with bm.name as|)
        (terpri)
        (princ '|    the value of the FORM slot.|)
        (terpri)
        (terpri)
        (return
          bm.name1)))


;*************************************************************************
;  The following is the a-list for the mouseable window to be created
;  during the ate and fault type testing.  The list is for the menu
;  of options for the mouse buttons which can be selected.  the two
;  tv:add functions insert the options "exit" and "re-select" to the
;  popup window.
;
(defvar *wave-list* nil)

(tv:add-typeout-item-type *wave-list* :new-type
                          "exit" (exit) nil "exit")

(tv:add-typeout-item-type *wave-list* :new-type
                          "re-select" nil nil)
```

```
;*****************************************************************************
;  This defines the flavor for the mouseable window used for ate and
;  fault type diagnosis.
;
(defflavor *menu-select* ()
           (tv:centered-label-mixin
            tv:borders-mixin
            tv:top-box-label-mixin
            tv:changeable-name-mixin
            tv:basic-mouse-sensitive-items
            tv:window))


;*****************************************************************************
;  This defines the mouseable window that is created for ate and fault
;  type diagnosis.
;
(defvar *sensitive-window*
        (tv:make-window
          '*menu-select*
          ':borders 2
          ':top 455
          ':bottom 700
          ':right 1088
          ':width 1088
          ':blinker-p nil
          ':label '(:font fonts:bigfnt)
          ':item-type-alist *wave-list*
          ':font-map '(fonts:cptfontcb)))


;*****************************************************************************
;  diagnose is the main function.  When called it verifies the option
;  and calls the appropriate diagnostic function and, when finished,
;  calls the appropriate listing function.  If the option given is
;  invalid, a message is displayed to the user and the function returns.
;
(defun diagnose (component test-type)
  (graph component)
  (cond
    ((equal test-type 'ate)
     (show.components (test.ckt component)))
    ((equal test-type 'fault)
     (func.diagnose (test.ckt component)))
    ((equal test-type 'bench)
     (show.list (bench.diagnose component)))
    (t
     (terpri)
     (princ test-type)
     (princ '| is an invalid option.  <FAULT, ATE or BENCH allowed>|)
     (terpri))))
```

```
;*************************************************************************
;  bench.diagnose is the control function for bench-type diagnosis.
;  The one parameter passed to it is the component to be diagnosed.
;
(defun bench.diagnose (component)
  (prog (component-list functions)
        (setq functions (get.values component 'function))
        (setq component-list
              (set.component.list (list component) functions))
        (cond (functions
               (return
                 (apply 'append
                        (mapcar 'test.subcomponent
                                component-list functions))))
              (t (return (list component))))))


;*************************************************************************
;  set.component.list receives a component name and a list of its
;  functions.  The value returned is a list of occurrances of the
;  component name repeated once for each function in the function list.
;
(defun set.component.list (component functions)
  (prog ()
        (cond (functions
               (return (append component
                               (set.component.list
                                 component
                                 (cdr functions))))))))


;*************************************************************************
;  test.subcomponent recurses by calling bench.diagnose to do a depth-
;  first search through the circuit's hierarchical representation.
;  I will stop at each node to verify the operation of that node and
;  will expand that node's child nodes if it is failing to perform
;  its function properly.  Otherwise, it will 'prune' that part of
;  the search and go on.
;
(defun test.subcomponent (component functions)
  (prog ()
        (cond ((fail component (car functions))
               (cond ((cdr functions)
                      (return
                        (apply 'append
                               (mapcar 'bench.diagnose
                                       (mapcar 'car
                                               (cdr functions))))))
                     (t (return (list component))))))))
```

```
;*************************************************************************
;  fail displays the function of the given component and asks the
;  user if the function is being performed properly.  Returns true
;  if the component is failing, otherwise, returns false (nil).
;
(defun fail (component function)
  (progn ()
         (graph component)
         (terpri)
         (princ '|Does |)
         (princ component)
         (princ '| |)
         (princ function)
         (princ '|? |)
         (cond ((equal 'n (read))))))


;*************************************************************************
;  show.list is called by diagnose when the selected option is 'bench.
;  A list of suspect components is displayed.  All other components
;  are assumed to be good.
;
(defun show.list (line)
  (terpri)
  (cond
    (line
     (princ '|The list of the suspect components is |)
     (column.list line))
    (t
     (princ '|No suspect components found.|)
     (terpri)))
  (terpri))


;*************************************************************************
;  test.ckt is the driver program for ate and fault type testing.  It
;  calls find.components to diagnose the circuit having the root node
;  of component and then formats the returned value into a list of lists
;  of the form (part input-waveform output-waveform condition) which is
;  returned.
;
(defun test.ckt (component)
  (loop with results = (find.components
                         (get.parts component))
        for part in (car results) and
            input-waveform in (cadr results) and
            output-waveform in (caddr results)
        collect (list part
                      (list (car input-waveform)
                            (cadr input-waveform))
                      (list (car output-waveform)
                            (cadr output-waveform))
                      (caddr input-waveform))))
```

```
;*************************************************************************
;  get.parts returns a list of all elemental parts  of the given
;  component.  That is, it returns a list of all the leaves of the
;  hierarchical tree structure having component as its root.
;
(defun get.parts (component)
  (prog (parts)
        (setq parts (get.values component 'parts))
        (return
          (cond
            (parts
             (return
               (apply 'append (mapcar 'get.parts parts))))
            (t
             (return
               (list component)))))))


;*************************************************************************
;  find.components receives a list of parts and determines the condition
;  of each according to the output waveform and input waveform selected
;  from the menu when displayed.
;
(defun find.components (parts)
  (loop with part-list and
          input-waveform-list and
          output-waveform-list and
          intermediate-result and
          final-result
        for part in parts and
            wf first (get.output.waveforms (car parts))
                 then (setq wf final-result)
        collect (car (unitfullreference part)) into part-list
        do (setq intermediate-result
                 (caddr
                   (setup 'Output (car (unitfullreference part)) wf)))
           (setq final-result
                 (get.input.waveforms part intermediate-result))
        if (caddr intermediate-result)
          collect intermediate-result into input-waveform-list
        collect intermediate-result into output-waveform-list
        finally
          (return
            (list part-list
                  (append input-waveform-list
                          (cddr
                            (setup 'Input
                                   (car (unitfullreference part))
                                   final-result)))
                  output-waveform-list))))
```

```
;*************************************************************************
;  get.input.waveforms returns a list of all possible input waveforms
;  given the component and its output waveform.  If the given output
;  waveform is "unk"nown, then all possible input waveforms are
;   returned.
;
(defun get.input.waveforms (part output-waveform)
  (prog (subparts)
        (setq subparts (get.values part 'parts))
        (return
          (cond
            ((atom
               (car output-waveform))
             (return
               (loop with result = nil
                     for wf in (get.values part 'waveforms)
                     if (or (equal (list (car output-waveform)
                                         (cadr output-waveform))
                                   (list (caar wf) (cadar wf)))
                            (equal (car output-waveform) 'unk))
                       do (setq result (append result (cdr wf)))
                     finally
                       (return
                         result))))
            (t
             (return
               (loop with result = nil
                     for wf in (get.values part 'waveforms)
                     do (setq result
                              (append result
                                      (loop with rslt = nil
                                            for wvfrm in output-waveform
                                            if (or
                                                 (equal
                                                   (list (car wvfrm)
                                                         (cadr wvfrm))
                                                   (list (caar wf)
                                                         (cadar wf)))
                                                 (equal (car wvfrm)
                                                        'unk))
                                              do (setq rslt
                                                       (append rslt
                                                               (cdr wf)))
                                            finally
                                              (return
                                                rslt))))
                     finally
                       (return
                         result)))))))
```

```
;*************************************************************************
;  get.output.waveforms returns a list of all output waveforms
;  possible from the given component.
;
(defun get.output.waveforms (component)
  (loop with wf
        for wf in (get.values component 'waveforms)
        collect (car wf)))


;*************************************************************************
;  setup receives the io-flag, module name, and output waveform list
;  and calls either setup.a or setup.b to display the selection menu
;  depending on the total number of waveforms.  If there are more than
;  six waveforms, then we want to allow the user to view all of them
;  before making a selection (we do this by calling setup.b).  If there
;  are six or less, then we can just display them (by calling setup.a).
;
(defun setup (io-flag module wave-form)
  (prog ()
        (return
          (cond
            ((equal wave-form nil)
             (return
               '(unk)))
            ((equal (cdr wave-form) nil)
             (return
               (list nil nil (car wave-form))))
            (t
             (prog (wf)
                   (setq wf (append wave-form '((unk))))
                   (cond
                     ((< (length wf) 7)
                      (return
                        (setup.a io-flag module wf)))
                     (t
                      (return
                        (setup.b io-flag module wf)))))))))))
```

```
;*****************************************************************************
;  setup.b  breaks up the list of waveforms into groups of five forms
;  and displays them with a sixth form of "other" to allow rotating
;  the groups past the window in order to view all possible waveforms.
;  When "other" is selected from the last group, the first group is
;  redisplayed (like wrapping around).
;
(defun setup.b (io-flag module wave-form)
  (loop with pwf = wave-form
        as result = (progn ()
                           (cond
                             ((equal pwf nil)
                              (setq pwf wave-form)))
                           (loop with lwf = nil
                                 for i from 1 to 5
                                 if pwf
                                   collect (car pwf) into lwf
                                 do (setq pwf (cdr pwf))
                                 finally
                                   (setq lwf (append lwf '((oth))))
                                   (return
                                     (setup.a io-flag module lwf))))
        while (equal (caddr result) '(oth))
        finally
          (return
            result)))
```

```
;*****************************************************************************
;  setup.a receives the io-flag, module name, and waveform list and
;  displays the list of waveforms with a title on the menu specifying
;  whether the waveforms are input or output to the named module.
;  The function will then wait until a waveform has been selected and
;  will return the data associated with that form.
;
(defun setup.a (io-flag module wave-form)
  (tv:window-call (*sensitive-window* :deactivate)
    (send *sensitive-window*
          ':clear-window)
    (send *sensitive-window*
          ':set-name
          (string-append
            "Which waveform is "
            (string io-flag)
            (cond
              ((equal io-flag 'Input)
               (string " to "))
              (t
               (string " from ")))
            (string module)))
    (loop for wf in wave-form and x first 0 then (+ x 1)
          do (send *sensitive-window*
                   ':string-out
                   (substring
                     (string-append
                       (car (get.values (car wf) 'form))
                       " "
                       (cond
                         ((cadr wf))
                         (t
                          (string " ")))
                       "                    ")
                     0 20))
             (send *sensitive-window*
                   ':string-out "  ")
             (send *sensitive-window*
                   ':bitblt
                   tv:alu-seta
                   100 100
                   (eval (car (get.values (car wf) 'form)))
                   0
                   0
                   (* x 176)
                   20)
             (send *sensitive-window*
                   ':primitive-item
                   ':new-type
                   wf
                   (* x 176)
                   0
                   (+ (* (+ x 1) 164) (* x 12))
```

```
                    120))
      (loop as blip = (send *sensitive-window* ':any-tyi)
            until (equal (cadr blip) '(exit))
            finally
              (send *sensitive-window* ':clear-window)
              (return
                blip))))


;*****************************************************************************
;  show.components is called by diagnose when the selected option is
;  'ate.  The condition of each component is displayed for the user.
;
(defun show.components (line)
  (terpri)
  (loop for l in line
        do (terpri)
           (princ '|The condition of component |)
           (princ (car l))
           (princ '| with input |)
           (princ (cadr l))
           (terpri)
           (princ '|          and output |)
           (princ (caddr l))
           (princ '| is |)
           (cond
             ((equal (cadddr l) 'g)
              (princ '|good.|))
             ((equal (cadddr l) 'b)
              (princ '|bad.|))
             (t
              (princ '|unknown.|)))
           (terpri))
  (terpri))
```

```
;*****************************************************************************
;  func.diagnose is called by diagnose when the selected option is
;  'fault.  If a component is labeled as bad, its fault function is
;  invoked to isolate the bad components even further.  The function
;  then lists all good components, then all components whose condition
;  is unknown, then all bad components.
;
(defun func.diagnose (results)
  (terpri)
  (loop with good-components = nil and
        unknown-components = nil and
        bad-components = nil
      for rslt in results
      if (equal (cadddr rslt) 'b)
        collect
          (list
            (eval (list (caar (get.values (car rslt) 'funct-cause))
                        `(quote ,(cadr rslt))
                        `(quote ,(caddr rslt))
                        `(quote ,(cadar
                                   (get.values (car rslt)
                                               'funct-cause)))))
            '|in|
            (car rslt))
          into bad-components
      if (equal (cadddr rslt) 'g)
        collect (car rslt) into good-components
      if (equal (cadddr rslt) 'u)
        collect (car rslt) into unknown-components
      finally
        (terpri)
        (princ '|List of good components = |)
        (cond
          (good-components
           (column.list good-components))
          (t
           (column.list '(None))))
        (princ '|List of unknown components = |)
        (cond
          (unknown-components
           (column.list unknown-components))
          (t
           (column.list '(None))))
        (princ '|List of bad components = |)
        (cond
          (bad-components
           (column.list bad-components))
          (t
           (column.list '(None))))
        (terpri)))
```

```
;*****************************************************************************
;  column.list is a utility function called by other functions wishing
;  a columnar display of elements within a list.  Each element within
;  the given list is displayed on a line by itself and indented 10
;  spaces.
;
(defun column.list (list)
  (loop for item in list
        do (terpri)
           (princ "|          |)
           (princ item))
  (terpri))
```

Appendix C: Fault Analysis Source

```
;;; -*- Mode:LISP; Package:KEE; Base:10. -*-

;*****************************************************************************
;
;     Filename:  FUNCT-CAUSE.LISP
;     Version:   1.0
;     Date:      4 Feb 86
;
;     Project:   Functional Diagnostic System
;
;     Author:    Don Wunz
;
;     Description:
;          This file contains the functional rules pertinent to the
;          different types of circuit modules.  Each receives the
;          input and output waveforms and a list of components
;          contained within the module.  It is the functional rule's
;          responsibility to diagnose the module and return a list
;          of components it finds to be failed.
;
;     Contents:
;          single.component
;          fullwave.rectifier
;          voltage.divider
;          dual.r.zd.regulator
;          two.way.range.selector
;          pi.rc.filter
;          complex.dekleer.current.regulator
;
;*****************************************************************************


;*****************************************************************************
;  single.component is used for any module comprised of only one
;  component.  If this function is called, that means the module was
;  found to be bad and implies that the one component it contains has
;  failed.  Therefore, this function simply returns the component.
;
(defun single.component (input-wave output-wave component)
  (setq component component))
```

```
;*************************************************************************
;  fullwave.rectifier receives a list of two diodes; the first is
;  the one responsible for the positive halfwave of the input and the
;  second is responsible for the negative halfwave of the input.
;  The value returned is the diode responsible if a halfwave is seen
;  as the output waveform, otherwise, if no waveform is seen output,
;  the list of both diodes is returned.
;
(defun fullwave.rectifier (input-wave output-wave diodes)
  (prog ()
        (return
          (cond
            ((equal (car output-wave) 'nw)
             (return
               diodes))
            ((equal (car output-wave) 'hw)
             (terpri)
             (princ '|Is the positive halfcycle |)
             (princ '|of the input waveform output? |)
             (cond
               ((equal 'n (read))
                (return (car diodes)))
               (t
                (return
                  (cadr diodes)))))))))
```

```
;*****************************************************************************
;  voltage.divider is the diagnostic function for a voltage divider
;  made of two resistors.  The value returned will be one of the
;  resistors (which one depends on the value of the output waveform
;  and the answer to the question asked).
;
(defun voltage.divider (input-wave output-wave resistors)
  (terpri)
  (princ '|Is the resistance measured across |)
  (prog ()
        (return
          (cond
            ((equal output-wave '(nw 0vdc))
             (princ (cadr resistors))
             (princ '| = 0? |)
             (cond
               ((equal 'n (read))
                (return
                  (car resistors)))
               (t
                (return
                  (cadr resistors)))))
            (t
             (princ (car resistors))
             (princ '| = 0? |)
             (cond
               ((equal 'n (read))
                (return
                  (cadr resistors)))
               (t
                (return
                  (car resistors)))))))))
```

```
;*************************************************************************
;  dual.r.zd.regulator analyzes a voltage regulator consisting of a pair
;  of resistor-zenear diode regulators.  If the output is 0VDC, then it
;  is assumed one of the resistors is bad; if the output is 36VDC, then
;  one of the diodes is considered bad.  Note that in this simple
;  implementation, no other considerations are given to partial
;  regulation.

(defun dual.r.zd.regulator (input-wave output-wave components)
  (prog ()
        (return
          (cond ((equal output-wave '(nw 0vdc))
                 (return
                   (list '|One or both of |
                         (list (car components) (caddr components)))))
                ((member output-wave '((hw 36v.peak) (fhw 36v.peak)))
                 (return
                   (list '|One or both of |
                         (list (cadr components)
                               (cadddr components))))))))))


;*************************************************************************
;  two.way.range.selector diagnoses the range selection circuitry
;  composed of a switch and two resistors which form a voltage
;  divider with the switch connected across one of the resistors.

(defun two.way.range.selector (input-wave output-wave components)
  (prog ()
        (return
          (cond ((equal output-wave '(nw 0vdc))
                 (terpri)
                 (princ '|Is output 0VDC when the |)
                 (princ (car components))
                 (princ '| is in the other position? |)
                 (cond ((equal 'n (read))
                        (return
                          (caddr components)))
                       (t
                        (return
                          (cadr components)))))
                (t
                 (return
                   (list '|One or both of | (cdr components))))))))))
```

```
;*************************************************************************
;  pi.rc.filter is a capacitor-resistor-capacitor DC filter diagnostic.
;  The test may be inconclusive in that if the input-wave is a half-
;  wave and the output-wave is a filtered halfwave, we cannot tell if
;  just one or both capacitors are filtering properly.

(defun pi.rc.filter (input-wave output-wave components)
  (prog ()
        (return
          (cond ((equal output-wave '(nw 0vdc))
                 (return
                   (car components)))
                ((member output-wave '((fw 37v.peak) (hw 37v.peak)))
                 (return
                   (cdr components)))
                ((equal output-wave '(ffw 37v.peak))
                 (return
                   (list '|One of | (cdr components))))
                (t
                 (return
                   (list '|One of |
                         (cdr components)
                         '| may be bad|)))))))


;*************************************************************************
;  complex.dekleer.current.regulator is the diagnostic for the current
;  regulator in the IP-28 power supply deKleer uses in his paper
;  referenced in this thesis.  The diagnostics are incomplete here
;  because a more complete analysis of the circuit is needed than is
;  available.  Idealy, the diagnostics would be written by the engineer
;  responsible for the design and converted to the functional rule
;  below by the software engineer.

(defun complex.dekleer.current.regulator
       (input-wave output-wave components)
  (prog ()
        (return 'cc-reg)))
```

Appendix D: DIAGNOSE User's Manual

The contents of this appendix are a combination of a user's guide for the diagnostic system and any "hints and kinks" that can be remembered concerning its operation.

Booting KEE

At the fep> prompt, type boot >kee.boot

Note that resetting the fep is not normally necessary but may be needed in which case, prior to booting, at the fep> prompt, type reset fep and answer y to the question concerning loss of all data; then proceed to boot KEE.

Loading DIAGNOSE and a Knowledge Base

After KEE has completed the boot process, follow these procedures for expanding the lisp and kee windows.

1. Click left on the KEE symbol and select the "show changed elements" option in the pop-up window. An output window will be displayed in the lower left of the screen. Now do the same thing again and a second output window will be displayed. The one we want is the highest, rightmost one. If it is on top, go on to step 2, otherwise move the mouse to a portion of that window (showing above and to the right of the other output window) and click left on it; this will put it on top.
2. Now, click right on the lisp window and select the "inflate" option; then do the same on the right KEE display window.

The KEE windows are now ready.

3. Login -- Log in before proceeding.

4. Click left on the KEE symbol and select the "Load KB" option. When the I/O window requests, enter the path and name of the knowledge base to be loaded.
5. Enter into the lisp window: (load '>diagnose>diag-load) diagnose.lisp and funct-cause.lisp will be loaded, then the user is asked for the current KB. Enter the KB name loaded in step 4.
6. Notice the KB graph in the left KEE display window; move the mouse to the upper left of the right display window and click left on the pen symbol to make the right display window the current one.

The system is now loaded, ready for diagnostic runs.

Creating a Knowledge Base

1. Reset fep and boot KEE.
2. Load the basic-diag KB.
3. Copy it to the desired name by selecting the copy option on the pop-up window (found by clicking left on the KEE symbol) and answer the pathname request in the I/O window.
4. Delete the basic-diag KB.
5. Enter (set.kb 'kb.name) in the lisp window using the kb.name created in step 3.
6. Create a member of class circuits for each module and sub-module.
7. Update the slot values for all the members.
8. Enter (graphkb) in the lisp window and verify the hierarchy graph shown in the KEE display window.

9. Save the new KB.

10. Make the right display window the current one.

Running DIAGNOSE

1. Boot KEE.

2. Load the desired KB and DIAGNOSE software.

3. Type (diagnose 'module 'option) in the lisp window where module is the module/submodule to be diagnosed and option is one of bench, ate, or fault. Bench is used for bench testing such as that done in a service shop where the hardware can be opened up and examined in detail. Ate is used for strict waveform analysis and is here only because it was implemented as an intermediate step in the development of the software during the thesis research. Fault is the same as ate plus inclusion of the functional diagnostic rules.

4. If bench was the selected option, proceed to step 5.
   Respond to the waveform selection by clicking right on the desired waveform. If 'other' is shown, it means there are more waveforms to choose from; when the last group of forms are shown, selecting 'other' will return to the first group of waveforms. If the wrong form is selected, choose 'reselect' in the pop-up window; otherwise, choose the 'exit' option to proceed to the next checkpoint.

5. If ate was the selected option, proceed to step 6.
   Answer each of the questions posed by diagnose. If bench was selected, the questions come from the subfunctions given in the function slot of each module. If fault was selected, the

questions are coming from the functional rules invoked by the system for each module identified as bad.

6. At completion of diagnose, a list of suspected bad components is given, along with a list of the good components and a list of components whose performance is unknown. The list of bad components is given under all three options; this is the only list displayed by the bench option. All three lists are given under the ate and fault options.

Appendix E: Sample Runs

BASIC-DIAG.U Knowledge Base

```
;;;  -*- Mode:LISP; Package:KEE; Base:10. -*-

(BASIC-DIAG
  ("DWUNZ" "18-Nov-85 13:21:12" "DWUNZ" "14-Feb-86 11:54:38")
  NIL
  (KNOWLEDGEBASES)
  NIL
  ()
  ((KBMETHODFILE (BASIC-DIAG))
   (KBSIZE 13)
   (KEE.DEVELOPMENT.VERSION.NUMBER 0)
   (KEE.MAJOR.VERSION.NUMBER 2)
   (KEE.MINOR.VERSION.NUMBER 1)
   (KEE.PATCH.VERSION.NUMBER 3)
   (KEEVERSION KEE2.1)))

(CIRCUITS
  ("DWUNZ" "18-Nov-85 13:21:37" "DWUNZ" "27-Jan-86 13:51:44")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((FUNCT-CAUSE NIL NIL NIL NIL NIL)
   (FUNCTION NIL NIL NIL NIL NIL)
   (PARTS NIL NIL (CIRCUITS) NIL NIL)
   (WAVEFORMS NIL NIL NIL NIL NIL))
  ())

(WAVE
  ("DWUNZ" "13-Dec-85 4:31:10" "DWUNZ" "13-Dec-85 4:55:51")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((DISP.FORM (LAMBDA (SELF X Y)
                  (BITBLT* (EVAL (CAR (GET.VALUES SELF 'FORM)))
                           0
                           0
                           (TV:WINDOW-UNDER-MOUSE)
                           X
                           Y))
              METHOD
              (METHOD))
   (FORM NIL NIL NIL NIL NIL))
  ())
```

```
(SW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:52:17")
  NIL
  (WAVE)
  "Sinewave"
  ()
  ((FORM (SINEWAVE))))

(FW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:29")
  NIL
  (WAVE)
  "Fullwave"
  ()
  ((FORM (FULLWAVE))))

(HW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:38")
  NIL
  (WAVE)
  "Halfwave"
  ()
  ((FORM (HALFWAVE))))

(NW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:52:05")
  NIL
  (WAVE)
  "No (flat) wave"
  ()
  ((FORM (FLATWAVE))))

(FFW
  ("DWUNZ" "14-Feb-86 11:53:46" "DWUNZ" "14-Feb-86 11:54:09")
  NIL
  (WAVE)
  NIL
  ()
  ((FORM (FILTERED-FW))))

(FHW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:18")
  NIL
  (WAVE)
  "Filtered halfwave"
  ()
  ((FORM (FILTERED-HW))))
```

```
(CFW
  (ASSERT "1//02//86 00:59:13" "DWUNZ" "23-Jan-86 15:50:44")
  NIL
  (WAVE)
  "Clipped fullwave"
  ()
  ((FORM (CLIP-FW))))

(CHW
  (ASSERT "1//02//86 01:00:06" "DWUNZ" "23-Jan-86 15:51:00")
  NIL
  (WAVE)
  "clipped halfwave"
  ()
  ((FORM (CLIP-HW))))

(CFHW
  (ASSERT "1//02//86 01:00:09" "DWUNZ" "23-Jan-86 15:50:28")
  NIL
  (WAVE)
  "Clipped filtered halfwave"
  ()
  ((FORM (CLIP-FHW))))

(UNK
  ("" "24-Jan-86 16:58:00" "" "24-Jan-86 16:58:00")
  NIL
  (WAVE)
  NIL
  ()
  ((FORM (UNKNOWN))))

(OTH
  ("" "24-Jan-86 16:57:59" "" "24-Jan-86 16:58:00")
  NIL
  (WAVE)
  NIL
  ()
  ((FORM (OTHER))))

KBEnd
```

H89-DIAG.U Knowledge Base

```
;;;  -*- Mode:LISP; Package:KEE; Base:10. -*-

(H89-DIAG
  ("DWUNZ" "18-Nov-85 13:21:12" "DWUNZ" "10-Oct-85 14:56:41")
  NIL
  (KNOWLEDGEBASES)
  NIL
  ()
  ((KBMETHODFILE (H89-DIAG))
   (KBSIZE 34)
   (KEE.DEVELOPMENT.VERSION.NUMBER 0)
   (KEE.MAJOR.VERSION.NUMBER 2)
   (KEE.MINOR.VERSION.NUMBER 1)
   (KEE.PATCH.VERSION.NUMBER 3)
   (KEEVERSION KEE2.1)))

(CIRCUITS
  ("DWUNZ" "18-Nov-85 13:21:37" "DWUNZ" "27-Jan-86 13:51:44")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((FUNCT-CAUSE NIL NIL NIL NIL NIL)
   (FUNCTION NIL NIL NIL NIL NIL)
   (PARTS NIL NIL (CIRCUITS) NIL NIL)
   (WAVEFORMS NIL NIL NIL NIL NIL))
  ())

(H89
  (ASSERT "11//18//85 13:22:59" "DWUNZ" "18-Nov-85 14:33:33")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((COMPUTE)
               (CPU))
              ((DISPLAY)
               (VIDEO))
              ((PROVIDE POWER)
               (POWER))))
   (PARTS (CPU VIDEO POWER))))

(CPU
  (ASSERT "11//18//85 13:24:53" "DWUNZ" "18-Nov-85 13:24:53")
  NIL
  (CIRCUITS)
  NIL
  ()
  ())
```

```
(VIDEO
  (ASSERT "11//18//85 13:25:20" "DWUNZ" "18-Nov-85 13:25:20")
  NIL
  (CIRCUITS)
  NIL
  ()
  ())

(POWER
  (ASSERT "11//18//85 13:25:25" "DWUNZ" "18-Nov-85 14:36:47")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((PROVIDE 18 VDC)
               (P18 (PIN1 PIN5)))
              ((PROVIDE -18 VDC)
               (N18 (PIN2 PIN5)))
              ((PROVIDE 5 VDC)
               (P5 (PIN3 PIN5)))
              ((PROVIDE 2.5 VDC)
               (P2.5 (PIN4 PIN5)))))
   (PARTS (P18 N18 P5 P2.5))))

(P18
  (ASSERT "11//18//85 13:25:28" "DWUNZ" "25-Dec-85 23:13:05")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((CONVERT 117 VRMS TO 13 VRMS)
               (P18.ACSRC (TP1 TP3)))
              ((FULLWAVE RECTIFY 13 VRMS TO 18 V PEAK)
               (P18.RECT (PIN1 PIN2)))
              ((FILTER 18 V PEAK TO 18 VDC)
               (P18.FILT (PIN1 PIN5)))))
   (PARTS (P18.FILT P18.RECT P18.ACSRC))))

(P18.FILT
  (ASSERT "11//18//85 13:25:54" "DWUNZ" "29-Jan-86 15:02:50")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT C1)))
   (WAVEFORMS (((FW 18V.PEAK)  (FW 18V.PEAK B))
               ((NW 18VDC)     (FW 18V.PEAK G))
               ((NW 0VDC)      (NW 0VDC U)
                               (FW 18V.PEAK B)
                               (HW 18V.PEAK B))
               ((FHW 18V.PEAK) (HW 18V.PEAK G))
               ((HW 18V.PEAK)  (HW 18V.PEAK B))))))
```

```
(P18.RECT
  (ASSERT "11//18//85 13:25:49" "DWUNZ" "27-Jan-86 13:53:51")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((FULLWAVE.RECTIFIER (D2 D4))))
   (WAVEFORMS (((FW 18V.PEAK) (SW 36V.P-P G))
               ((HW 18V.PEAK) (SW 36V.P-P B))
               ((NW 0VDC)     (SW 36V.P-P B)
                              (NW 0VDC U))))))

(P18.ACSRC
  (ASSERT "11//18//85 13:25:44" "DWUNZ" "24-Jan-86 14:58:42")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((RECEIVE 117 VRMS)
               (P18.ACSRC.SWITCH (TP7 TP8))
               (P18.ACSRC.FUSE (TP7 TP8)))
              ((CONVERT 117 VRMS TO 13 VRMS)
               (P18.ACSRC.TRANSFORMER (TP1 TP3)))))
   (PARTS (P18.ACSRC.TRANSFORMER P18.ACSRC.FUSE P18.ACSRC.SWITCH))))

(P18.ACSRC.TRANSFORMER
  (ASSERT "11//18//85 13:26:44" "DWUNZ" "27-Jan-86 13:55:56")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT T1-1)))
   (WAVEFORMS (((SW 36V.P-P) (SW 140V.P-P G))
               ((NW 0VDC)    (SW 140V.P-P B)
                             (NW 0VDC U))))))

(P18.ACSRC.FUSE
  (ASSERT "11//18//85 13:26:35" "DWUNZ" "27-Jan-86 13:56:24")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT F1)))
   (WAVEFORMS (((SW 140V.P-P) (SW 140V.P-P G))
               ((NW 0VDC)     (SW 140V.P-P B)
                              (NW 0VDC U))))))
```

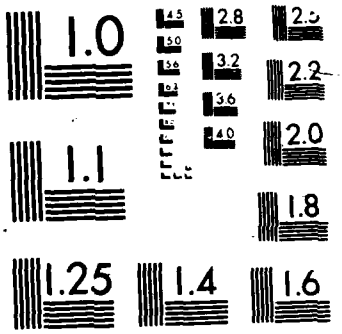
```
(P18.ACSRC.SWITCH
  (ASSERT "11//18//85 13:26:27" "DWUNZ" "27-Jan-86 13:56:49")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT S1)))
   (WAVEFORMS (((SW 140V.P-P) (SW 140V.P-P G))
               ((NW 0VDC)     (SW 140V.P-P B)
                              (NW 0VDC U))))))

(N18
  (ASSERT "11//18//85 13:25:30" "DWUNZ" "23-Jan-86 15:42:32")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((CONVERT 117 VRMS TO 13 VRMS)
               (P18.ACSRC (TP1 TP3)))
              ((FULLWAVE RECTIFY 13 VRMS TO -18 V PEAK)
               (N18.RECT (PIN2 PIN1)))
              ((FILTER -18V PEAK TO -18 VDC)
               (N18.FILT (PIN2 PIN5)))))
   (PARTS (N18.FILT N18.RECT P18.ACSRC))))

(N18.FILT
  (ASSERT "1//01//86 22:12:57" "DWUNZ" "27-Jan-86 13:57:24")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT C2)))
   (WAVEFORMS (((FW -18V.PEAK)  (FW -18V.PEAK B))
               ((NW -18VDC)     (FW -18V.PEAK G))
               ((NW 0VDC)       (NW 0VDC U)
                                (FW -18VDC B)
                                (HW -18V.PEAK B))
               ((FHW -18V.PEAK) (HW -18V.PEAK G))
               ((HW -18V.PEAK)  (HW -18V.PEAK B))))))
```

```
(N18.RECT
  (ASSERT "1//01//86 22:17:11" "DWUNZ" "27-Jan-86 13:58:09")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((FULLWAVE.RECTIFIER (D3 D1))))
   (WAVEFORMS (((FW -18V.PEAK) (SW 36V.P-P G))
               ((HW -18V.PEAK) (SW 36V.P-P B))
               ((NW 0VDC)      (SW 36V.P-P B)
                               (NW 0VDC U))))))

(P5
  (ASSERT "11//18//85 13:25:32" "dwunz" "1-Jan-86 23:19:35")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((CONVERT 117 VRMS TO 6 VRMS)
               (P5.ACSRC (TP4 TP6)))
              ((FULLWAVE RECTIFY 6 VRMS TO 8 V PEAK)
               (P5.RECT (TP5 TP7)))
              ((FILTER 8V PEAK TO 8 VDC)
               (P5.FILT (TP7 PIN5)))
              ((REGULATE 8 VDC TO 5VDC)
               (P5.REG (PIN3 PIN5)))))
   (PARTS (P5.REG P5.FILT P5.RECT P5.ACSRC))))

(P5.REG
  (ASSERT "1//01//86 23:12:49" "DWUNZ" "27-Jan-86 13:58:55")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT 5VREG-1)))
   (WAVEFORMS (((CFW 5V.PEAK)  (FW 8V.PEAK G))
               ((CHW 5V.PEAK)  (HW 8V.PEAK G))
               ((CFHW 5V.PEAK) (FHW 8V.PEAK G))
               ((NW 0VDC)      (FW 8V.PEAK B)
                               (HW 8V.PEAK B)
                               (FHW 8V.PEAK B)
                               (NW 0VDC U)
                               (NW 8VDC B))
               ((NW 5VDC)      (NW 8VDC G))
               ((FW 8V.PEAK)   (FW 8V.PEAK B))
               ((HW 8V.PEAK)   (HW 8V.PEAK B))
               ((FHW 8V.PEAK)  (FHW 8V.PEAK B))
               ((NW 8VDC)      (NW 8VDC B))))))
```

```
(P5.FILT
  (ASSERT "1//01//86 22:21:18" "DWUNZ" "27-Jan-86 13:59:20")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT C3)))
   (WAVEFORMS (((FW 8V.PEAK)  (FW 8V.PEAK B))
               ((NW 8VDC)     (FW 8V.PEAK G))
               ((NW 0VDC)     (NW 0VDC U)
                              (FW 8V.PEAK B)
                              (HW 8V.PEAK B))
               ((FHW 8V.PEAK) (HW 8V.PEAK G))
               ((HW 8V.PEAK)  (HW 8V.PEAK B))))))

(P5.RECT
  (ASSERT "1//01//86 22:21:22" "DWUNZ" "27-Jan-86 13:59:45")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT B2)))
   (WAVEFORMS (((FW 8V.PEAK) (SW 16V.P-P G))
               ((HW 8V.PEAK) (SW 16V.P-P B))
               ((NW 0VDC)    (SW 16VP-P B)
                             (NW 0VDC U))))))

(P5.ACSRC
  (ASSERT "1//01//86 22:21:38" "DWUNZ" "27-Jan-86 14:02:31")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((RECEIVE 117 VRMS)
               (P18.ACSRC.SWITCH (TP7 TP8))
               (P18.ACSRC.FUSE (TP7 TP8)))
              ((CONVERT 117 VRMS TO 6 VRMS)
               (P5.ACSRC.TRANSFORMER (TP4 TP6)))))
   (PARTS (P5.ACSRC.TRANSFORMER P18.ACSRC.FUSE P18.ACSRC.SWITCH))))

(P5.ACSRC.TRANSFORMER
  (ASSERT "1//01//86 23:24:33" "DWUNZ" "27-Jan-86 14:00:18")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT T1-2)))
   (WAVEFORMS (((SW 16V.P-P) (SW 140V.P-P G))
               ((NW 0VDC)    (SW 140V.P-P B)
                             (NW 0VDC U))))))
```

AD-A172 718 DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING D R WUNZ MAR 86 2/2

UNCLASSIFIED AFIT/GCS/ENG/86M-3 F/G 9/5 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
10.0

```
(P2.5
  (ASSERT "11//18//85 13:25:35" "DWUNZ" "23-Jan-86 15:41:16")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((CONVERT 117 VRMS TO 6 VRMS)
               (P5.ACSRC (TP4 TP6)))
              ((FULLWAVE RECTIFY 6 VRMS TO 8 V PEAK)
               (P5.RECT (TP5 TP7)))
              ((FILTER 8V PEAK TO 8VDC)
               (P5.FILT (TP7 PIN5)))
              ((REGULATE 8VDC TO 5VDC)
               (P5.REG (PIN3 PIN5)))
              ((DIVIDE 5 VDC TO 2.5 VDC)
               (P2.5.VOLT.DIV (PIN4 PIN5)))))
   (PARTS (P2.5.VOLT.DIV P5.REG P5.FILT P5.RECT P5.ACSRC))))

(P2.5.VOLT.DIV
  (ASSERT "1//01//86 22:58:51" "DWUNZ" "27-Jan-86 14:03:56")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((VOLTAGE.DIVIDER (R1 R2))))
   (WAVEFORMS (((FW 8V.PEAK)      (FW 8V.PEAK B))
               ((HW 8V.PEAK)      (HW 8V.PEAK B))
               ((FHW 8V.PEAK)     (FHW 8V.PEAK B))
               ((NW 5OR8VDC)      (NW 5OR8VDC B))
               ((CFW 5V.PEAK)     (CFW 5V.PEAK B))
               ((CHW 5V.PEAK)     (CHW 5V.PEAK B))
               ((CFHW 5V.PEAK)    (CFHW 5V.PEAK B))
               ((NW 0VDC)         (NW 0VDC U)
                                  (FW 8V.PEAK B)
                                  (HW 8V.PEAK B)
                                  (FHW 8V.PEAK B)
                                  (NW 5OR8VDC B)
                                  (CFW 5V.PEAK B)
                                  (CHW 5V.PEAK B)
                                  (CFHW 5V.PEAK B))
               ((FW 4V.PEAK)      (FW 8V.PEAK G))
               ((HW 4V.PEAK)      (HW 8V.PEAK G))
               ((FHW 4V.PEAK)     (FHW 8V.PEAK G))
               ((NW 2.5OR4VDC)    (NW 5OR8VDC G))
               ((CFW 2.5V.PEAK)   (CFW 5V.PEAK G))
               ((CHW 2.5V.PEAK)   (CHW 5V.PEAK G))
               ((CFHW 2.5V.PEAK)  (CFHW 5V.PEAK G))))))
```

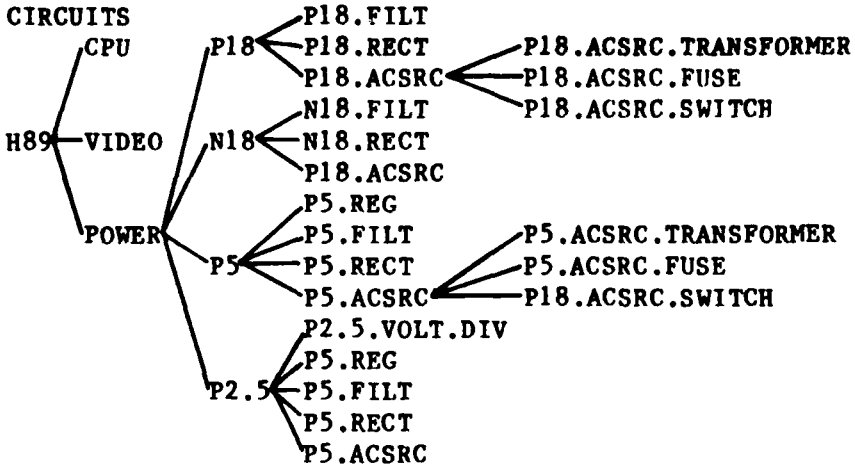
```
(WAVE
  ("DWUNZ" "13-Dec-85 4:31:10" "DWUNZ" "13-Dec-85 4:55:51")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((DISP.FORM (LAMBDA (SELF X Y)
                  (BITBLT* (EVAL (CAR (GET.VALUES SELF 'FORM)))
                           0
                           0
                           (TV:WINDOW-UNDER-MOUSE)
                           X
                           Y))
              METHOD
              (METHOD))
   (FORM NIL NIL NIL NIL NIL))
  ())

(SW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:52:17")
  NIL
  (WAVE)
  "Sinewave"
  ()
  ((FORM (SINEWAVE))))

(FW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:29")
  NIL
  (WAVE)
  "Fullwave"
  ()
  ((FORM (FULLWAVE))))

(HW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:38")
  NIL
  (WAVE)
  "Halfwave"
  ()
  ((FORM (HALFWAVE))))

(NW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:52:05")
  NIL
  (WAVE)
  "No (flat) wave"
  ()
  ((FORM (FLATWAVE))))
```

```
(FHW
  ("DWUNZ" "13-Dec-85 4:33:39" "DWUNZ" "23-Jan-86 15:51:18")
  NIL
  (WAVE)
  "Filtered halfwave"
  ()
  ((FORM (FILTERED-HW))))

(CFW
  (ASSERT "1//02//86 00:59:13" "DWUNZ" "23-Jan-86 15:50:44")
  NIL
  (WAVE)
  "Clipped fullwave"
  ()
  ((FORM (CLIP-FW))))

(CHW
  (ASSERT "1//02//86 01:00:06" "DWUNZ" "23-Jan-86 15:51:00")
  NIL
  (WAVE)
  "clipped halfwave"
  ()
  ((FORM (CLIP-HW))))

(CFHW
  (ASSERT "1//02//86 01:00:09" "DWUNZ" "23-Jan-86 15:50:28")
  NIL
  (WAVE)
  "Clipped filtered halfwave"
  ()
  ((FORM (CLIP-FHW))))

(UNK
  ("" "24-Jan-86 16:58:00" "" "24-Jan-86 16:58:00")
  NIL
  (WAVE)
  "Waveform unknown to system"
  ()
  ((FORM (UNKNOWN))))

(OTH
  ("" "24-Jan-86 16:57:59" "" "24-Jan-86 16:58:00")
  NIL
  (WAVE)
  "Select from other panel"
  ()
  ((FORM (OTHER))))

KBEnd
```

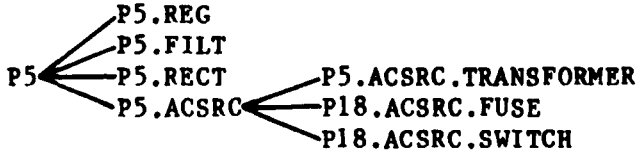
H89-DIAG Sample Run Using BENCH Diagnostics

Figure 6 shows the status of the Symbolics screen at the end of the command **(diagnose 'p5 'bench)**. The H89 Knowledge base is displayed in the left KEE Display window, the right KEE Display window displays the current component being tested, and the Lisp Listener window shows the dialog between the diagnostic system and the user. The right Display window is updated for each question during the dialog to display the component being tested and its substructure.

The PARTS relation in H89-DIAG KB



(Output)The PARTS Relation in H89-DIAG KB



Lisp Listener

```
(diagnose 'p5 'bench)

Does P5 (CONVERT 117 VRMS TO 6 VRMS)? y

Does P5 (FULLWAVE RECTIFY 6 VRMS TO 8 V PEAK)? y

Does P5 (FILTER 8 V PEAK TO 8 VDC)? n

Does P5 (REGULATE 8 VDC TO 5 VDC)? n

The list of the suspect components is
          P5.FILT
          P5.REG
```

Figure 6: H89 Bench Test

Figures 7 through 9 show the steps taken for the command **(diagnose 'p5 'fault)**. Figure 7 is the screen display after entering the command, and shows the "OTHER" option about to be selected. Figure 8 follows with the "FILTERED-HW 8V.PEAK" option to be selected. At the end of the diagnostic run, the Lisp Listener window shows the final output from the run listing the good, unknown, and bad components.

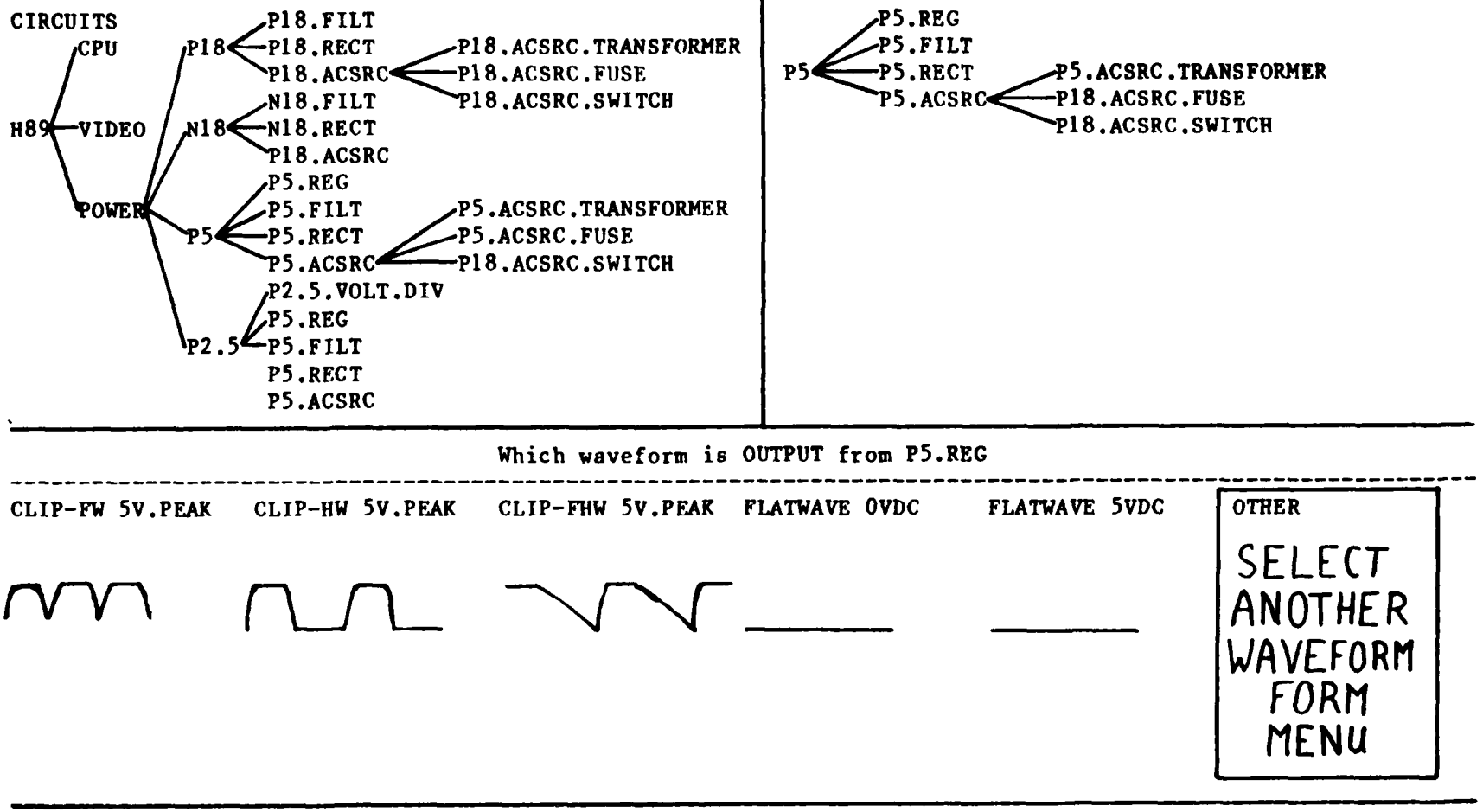


Figure 7: H89 Fault Test - Frame 1

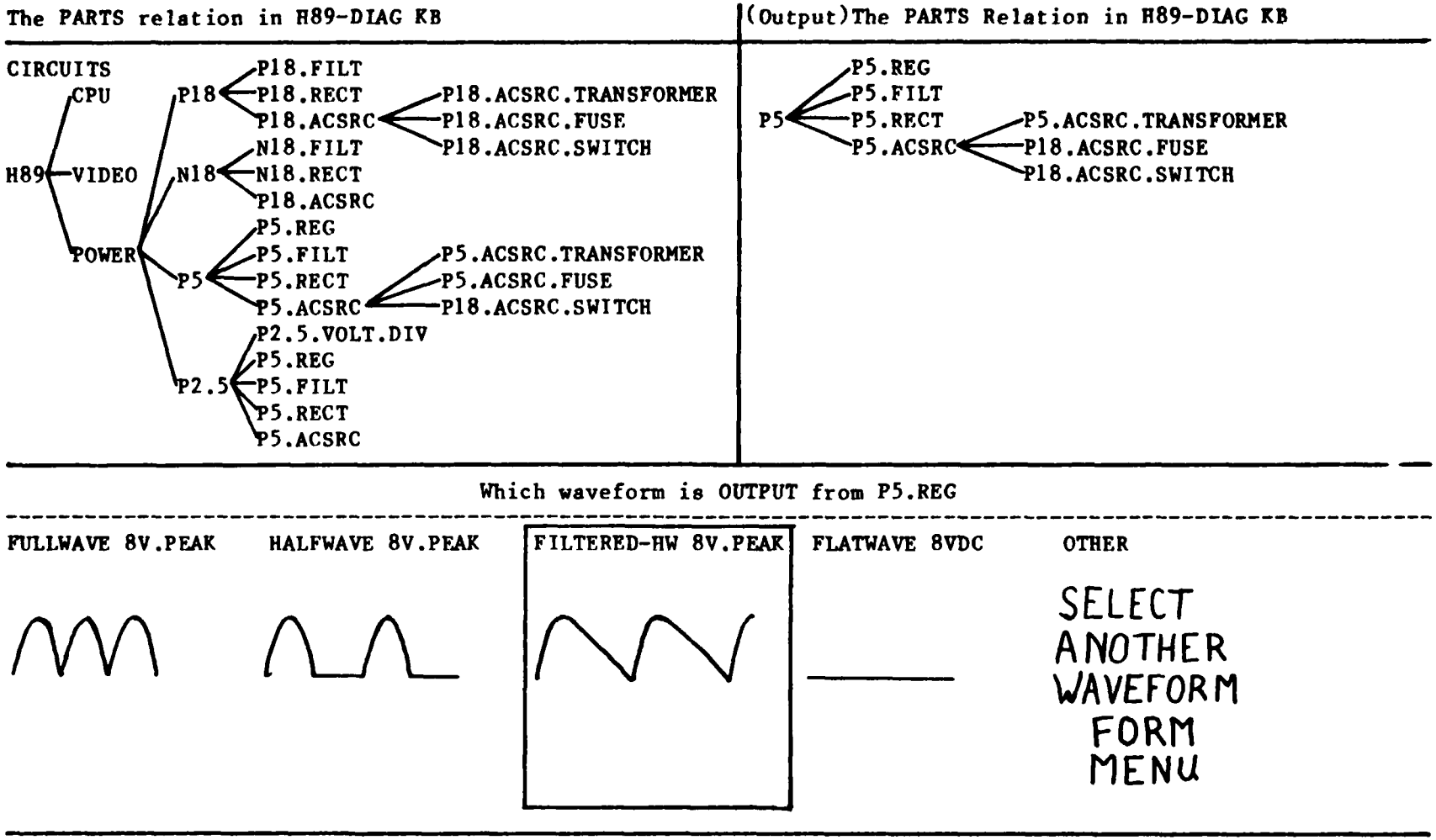


Figure 8: H89 Fault Test - Frame 2

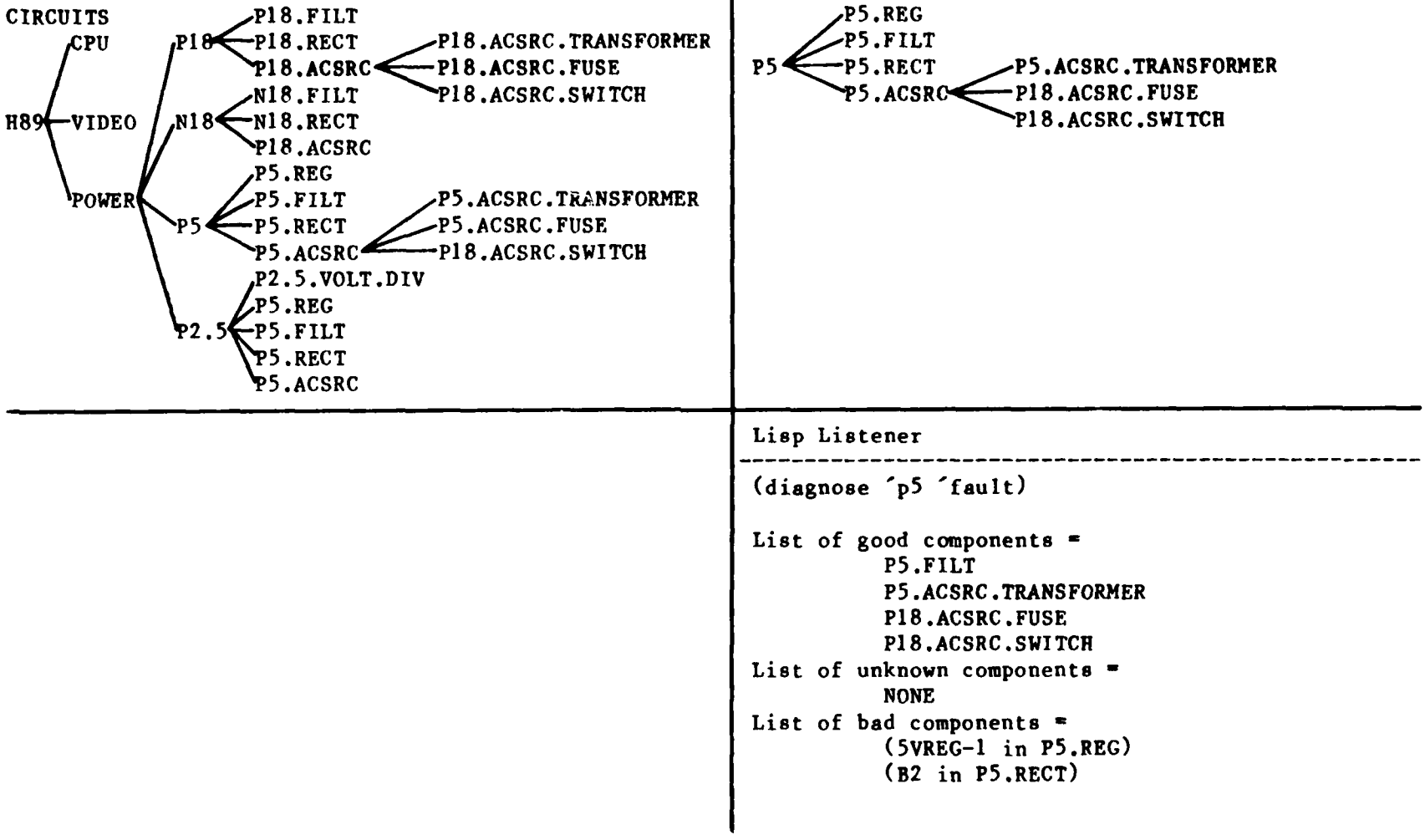


Figure 9: H89 Fault Test - Frame 3

DEKLEER-DIAG.U Knowledge Base

```
;;;  -*- Mode:LISP; Package:KEE; Base:10. -*-

(DEKLEER-DIAG
  ("DWUNZ" "5-Feb-86 21:57:39" "DWUNZ" "14-Feb-86 11:50:56")
  NIL
  (KNOWLEDGEBASES)
  NIL
  ()
  ((KBSIZE 29)
   (KEE.DEVELOPMENT.VERSION.NUMBER 0)
   (KEE.MAJOR.VERSION.NUMBER 2)
   (KEE.MINOR.VERSION.NUMBER 1)
   (KEE.PATCH.VERSION.NUMBER 3)
   (KEEVERSION KEE2.1)))

(CIRCUITS
  ("DWUNZ" "5-Feb-86 21:57:39" "DWUNZ" "5-Feb-86 21:57:40")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((FUNCT-CAUSE NIL NIL NIL NIL NIL)
   (FUNCTION NIL NIL NIL NIL NIL)
   (PARTS NIL NIL (CIRCUITS) NIL NIL)
   (WAVEFORMS NIL NIL NIL NIL NIL))
  ())

(DEKLEER
  ("DWUNZ" "9-Oct-85 21:23:58" "DWUNZ" "9-Oct-85 21:55:15")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((REGULATE VOLTAGE LEVEL)
               (VOLTAGE-CONTROL))
              ((REGULATE CURRENT LEVEL)
               (CURRENT-CONTROL))))
   (PARTS (CURRENT-CONTROL VOLTAGE-CONTROL))))
```

```
(CURRENT-CONTROL
  ("DWUNZ" "9-Oct-85 21:23:58" "DWUNZ" "9-Oct-85 22:00:20")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((FILTER OUTPUT SIGNAL)
               (CC-FILT2))
              ((REGULATE CURRENT LEVEL)
               (CC-REG))
              ((FILTER RECTIFIED AC SOURCE)
                (CC-FILT1))
              ((RECTIFY AC SOURCE)
               (CC-RECT))
               ((PRODUCE AC SOURCE) (CC-ACSRC))))
   (PARTS (CC-FILT2 CC-REG CC-FILT1 CC-RECT CC-ACSRC))))

(CC-FILT2
  ("DWUNZ" "9-Oct-85 21:24:02" "DWUNZ" "14-Feb-86 11:41:52")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (C5))))
   (WAVEFORMS (((NW 0VDC)       (NW 0VDC U)
                                (NW 37VDC B)
                                (NW 30VDC B)
                                (FFW 37V.PEAK B)
                                (FFW 30V.PEAK B)
                                (FW 37V.PEAK B)
                                (FW 30V.PEAK B)
                                (FHW 37V.PEAK B)
                                (FHW 30V.PEAK B)
                                (HW 37V.PEAK B)
                                (HW 30V.PEAK B))
               ((NW 37VDC)      (NW 37VDC G)
                                (FFW 37V.PEAK G))
               ((NW 30VDC)      (NW 30VDC G)
                                (FFW 30V.PEAK G))
               ((FFW 37V.PEAK) (FFW 37V.PEAK B)
                                (FW 37V.PEAK G))
               ((FFW 30V.PEAK) (FFW 30V.PEAK B)
                                (FW 30V.PEAK G))
               ((FHW 37V.PEAK) (FHW 37V.PEAK B)
                                (HW 37V.PEAK G))
               ((FHW 30V.PEAK) (FHW 30V.PEAK B)
                                (HW 30V.PEAK G))
               ((FW 37V.PEAK)  (FW 37V.PEAK B))
               ((FW 30V.PEAK)  (FW 30V.PEAK B))
               ((HW 37V.PEAK)  (HW 37V.PEAK B))
               ((HW 30V.PEAK)  (HW 30V.PEAK B))))))
```

```
(CC-REG
  ("DWUNZ" "9-Oct-85 21:24:02" "DWUNZ" "14-Feb-86 11:27:20")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((COMPLEX.DEKLEER.CURRENT.REGULATOR
        (Q1 Q2 Q3 Q4 Q5 Q6 R2 R8 R9 R10 R11 R12 R13 R14 C6))))
   (WAVEFORMS (((NW 0VDC)       (NW 37VDC B)
                                (FFW 37V.PEAK B)
                                (FW 37V.PEAK B)
                                (NW 0VDC U)
                                (FHW 37V.PEAK B)
                                (HW 37V.PEAK B))
               ((NW 37VDC)      (NW 37VDC B))
               ((NW 30VDC)      (FFW 37V.PEAK G)
                                (NW 37VDC G))
               ((FFW 37V.PEAK) (FFW 37V.PEAK B))
               ((FFW 30V.PEAK) (FFW 37V.PEAK G))
               ((FW 37V.PEAK)  (FW 37V.PEAK B))
               ((FW 30V.PEAK)  (FW 37V.PEAK G))
               ((FHW 37V.PEAK) (FHW 37V.PEAK B))
               ((FHW 30V.PEAK) (FHW 37V.PEAK G))
               ((HW 37V.PEAK)  (HW 37V.PEAK B))
               ((HW 30V.PEAK)  (HW 37V.PEAK G))))))

(CC-FILT1
  ("DWUNZ" "9-Oct-85 21:24:02" "DWUNZ" "14-Feb-86 11:08:58")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((PI.RC.FILTER (R1 C3 C4))))
   (WAVEFORMS (((NW 0VDC)       (NW 0VDC U)
                                (FW 37V.PEAK B))
               ((NW 37VDC)      (FW 37V.PEAK G))
               ((FFW 37V.PEAK) (FW 37V.PEAK B))
               ((HW 37V.PEAK)  (HW 37V.PEAK B))
               ((FHW 37V.PEAK) (HW 37V.PEAK B))
               ((FW 37V.PEAK)  (FW 37V.PEAK B))))))
```

```
(CC-RECT
  ("DWUNZ" "9-Oct-85 21:24:02" "DWUNZ" "14-Feb-86 11:10:57")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((FULLWAVE.RECTIFIER (D1 D2))))
   (WAVEFORMS (((NW 0VDC)      (NW 0VDC U)
                               (SW 74V.P-P B))
               ((FW 37V.PEAK) (SW 74V.P-P G))
               ((HW 37V.PEAK) (SW 74V.P-P B))))))

(CC-ACSRC
  ("DWUNZ" "9-Oct-85 21:24:01" "DWUNZ" "14-Feb-86 11:01:10")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (T1-2))))
   (WAVEFORMS (((NW 0VDC)     (NW 0VDC U)
                              (SW 140V.P-P B))
               ((SW 74V.P-P) (SW 140V.P-P G))))))

(VOLTAGE-CONTROL
  ("DWUNZ" "9-Oct-85 21:23:58" "DWUNZ" "14-Feb-86 10:55:20")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCTION (((RECTIFY VOLTAGE-CONTROLLER OUTPUT)
               (VC-RECT2))
              ((VARY OUTPUT VOLTAGE)
               (VC-VOLT-CONTROL))
              ((FILTER OUTPUT VOLTAGE)
               (VC-FILT2))
              ((SWITCH BETWEEN 10V AND 30V RANGES)
               (VC-RANGE-SELECTOR))
              ((PROVIDE 30V REGULATED DC)
               (VC-REG))
              ((FILTER RECTIFIED AC SOURCE)
               (VC-FILT1))
              ((RECTIFY AC SOURCE)
               (VC-RECT1))
              ((PRODUCE AC SOURCE)
               (VC-ACSRC))))
   (PARTS (VC-RECT2 VC-VOLT-CONTROL VC-FILT2 VC-RANGE-SELECTOR
           VC-REG VC-FILT1 VC-RECT1 VC-ACSRC))))
```

```
(VC-RECT2
  ("DWUNZ" "9-Oct-85 21:24:01" "DWUNZ" "10-Oct-85 14:39:26")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (D6))))
   (WAVEFORMS (((NW 0VDC)          (NW 0VDC U)
                                   (HW 36V.PEAK B)
                                   (FHW 36V.PEAK B)
                                   (CFHW 30V.PEAK B)
                                   (CHW 30V.PEAK B)
                                   (NW 30VDC B)
                                   (HW 12V.PEAK B)
                                   (FHW 12V.PEAK B)
                                   (CFHW 10V.PEAK B)
                                   (CHW 10V.PEAK B)
                                   (NW 10VDC B)
                                   (HW 18V.PEAK B)
                                   (FHW 18V.PEAK B)
                                   (CFHW 15V.PEAK B)
                                   (CHW 15V.PEAK B)
                                   (NW 15VDC B)
                                   (HW 6V.PEAK B)
                                   (FHW 6V.PEAK B)
                                   (CFHW 5V.PEAK B)
                                   (CHW 5V.PEAK B)
                                   (NW 5VDC B))
               ((HW 36V.PEAK)      (HW 36V.PEAK G))
               ((FHW 36V.PEAK)     (FHW 36V.PEAK G))
               ((CFHW 30V.PEAK)    (CFHW 30V.PEAK G))
               ((CHW 30V.PEAK)     (CHW 30V.PEAK G))
               ((NW 30VDC)         (NW 30VDC G))
               ((HW 12V.PEAK)      (HW 12V.PEAK G))
               ((FHW 12V.PEAK)     (FHW 12V.PEAK G))
               ((CFHW 10V.PEAK)    (CFHW 10V.PEAK G))
               ((CHW 10V.PEAK)     (CHW 10V.PEAK G))
               ((NW 10VDC)         (NW 10VDC G))
               ((HW 18V.PEAK)      (HW 18V.PEAK G))
               ((FHW 18V.PEAK)     (FHW 18V.PEAK G))
               ((CFHW 15V.PEAK)    (CFHW 15V.PEAK G))
               ((CHW 15V.PEAK)     (CHW 15V.PEAK G))
               ((NW 15VDC)         (NW 15VDC G))
               ((HW 6V.PEAK)       (HW 6V.PEAK G))
               ((FHW 6V.PEAK)      (FHW 6V.PEAK G))
               ((CFHW 5V.PEAK)     (CFHW 5V.PEAK G))
               ((CHW 5V.PEAK)      (CHW 5V.PEAK G))
               ((NW 5VDC)          (NW 5VDC G))))))
```

```
(VC-VOLT-CONTROL
  ("DWUNZ" "9-Oct-85 21:24:01" "DWUNZ" "10-Oct-85 14:25:00")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (R7))))
   (WAVEFORMS (((NW 0VDC)          (NW 0VDC U)
                                   (HW 36V.PEAK B)
                                   (FHW 36V.PEAK B)
                                   (CFHW 30V.PEAK B)
                                   (CHW 30V.PEAK B)
                                   (NW 30VDC B)
                                   (HW 12V.PEAK B)
                                   (FHW 12V.PEAK B)
                                   (CFHW 10V.PEAK B)
                                   (CHW 10V.PEAK B)
                                   (NW 10VDC B))
               ((HW 36V.PEAK)     (HW 36V.PEAK B))
               ((FHW 36V.PEAK)    (FHW 36V.PEAK B))
               ((CFHW 30V.PEAK)   (CFHW 30V.PEAK B))
               ((CHW 30V.PEAK)    (CHW 30V.PEAK B))
               ((NW 30VDC)        (NW 30VDC B))
               ((HW 12V.PEAK)     (HW 12V.PEAK B))
               ((FHW 12V.PEAK)    (FHW 12V.PEAK B))
               ((CFHW 10V.PEAK)   (CFHW 10V.PEAK B))
               ((CHW 10V.PEAK)    (CHW 10V.PEAK B))
               ((NW 10VDC)        (NW 10VDC B))
               ((HW 18V.PEAK)     (HW 36V.PEAK G))
               ((FHW 18V.PEAK)    (FHW 36V.PEAK G))
               ((CFHW 15V.PEAK)   (CFHW 30V.PEAK G))
               ((CHW 15V.PEAK)    (CHW 30V.PEAK G))
               ((NW 15VDC)        (NW 30VDC G))
               ((HW 6V.PEAK)      (HW 12V.PEAK G))
               ((FHW 6V.PEAK)     (FHW 12V.PEAK G))
               ((CFHW 5V.PEAK)    (CFHW 10V.PEAK G))
               ((CHW 5V.PEAK)     (CHW 10V.PEAK G))
               ((NW 5VDC)         (NW 10VDC G))))))
```

```
(VC-FILT2
  ("DWUNZ" "9-Oct-85 21:24:01" "DWUNZ" "10-Oct-85 14:27:57")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (C2))))
   (WAVEFORMS (((NW 0VDC)          (NW 0VDC U)
                                   (HW 36V.PEAK B)
                                   (FHW 36V.PEAK B)
                                   (CFHW 30V.PEAK B)
                                   (CHW 30V.PEAK B)
                                   (NW 30VDC B)
                                   (HW 12V.PEAK B)
                                   (FHW 12V.PEAK B)
                                   (CFHW 10V.PEAK B)
                                   (CHW 10V.PEAK B)
                                   (NW 10VDC B))
               ((HW 36V.PEAK)      (HW 36V.PEAK B))
               ((FHW 36V.PEAR)     (FHW 36V.PEAK B)
                                   (HW 36V.PEAK G))
               ((CFHW 30V.PEAK)    (CFHW 30V.PEAK B)
                                   (CHW 30V.PEAK G))
               ((CHW 30V.PEAR)     (CHW 30V.PEAK B))
               ((NW 30VDC)         (CFHW 30V.PEAK G)
                                   (CHW 30V.PEAK G))
               ((HW 12V.PEAK)      (HW 12V.PEAK B))
               ((FHW 12V.PEAR)     (FHW 12V.PEAK B)
                                   (HW 12V.PEAK G))
               ((CFHW 10V.PEAK)    (CFHW 10V.PEAK B)
                                   (CHW 10V.PEAK G))
               ((CHW 10V.PEAR)     (CHW 10V.PEAK B))
               ((NW 10VDC)         (NW 10VDC G)
                                   (CFHW 10V.PEAK G)
                                   (CHW 10V.PEAK G))))))
```

```
(VC-RANGE-SELECTOR
  ("DWUNZ" "9-Oct-85 21:24:00" "DWUNZ" "14-Feb-86 10:58:12")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((TWO.WAY.RANGE.SELECTOR ((VOLTAGE RANGE SWITCH) R5 R6))))
   (WAVEFORMS (((NW 0VDC)        (NW 0VDC U)
                                 (HW 36V.PEAK B)
                                 (FHW 36V.PEAK B)
                                 (CFHW 30V.PEAK B)
                                 (CHW 30V.PEAK B)
                                 (NW 30VDC B))
               ((HW 36V.PEAK)   (HW 36V.PEAK B))
               ((FHW 36V.PEAK)  (FHW 36V.PEAK B))
               ((CFHW 30V.PEAK) (CFHW 30V.PEAK B))
               ((CHW 30V.PEAK)  (CHW 30V.PEAK B))
               ((NW 30VDC)      (NW 30VDC B))
               ((HW 12V.PEAK)   (HW 36V.PEAK G))
               ((FHW 12V.PEAK)  (FHW 36V.PEAK G))
               ((CFHW 10V.PEAK) (CFHW 30V.PEAK G))
               ((CHW 10V.PEAK)  (CHW 30V.PEAK G))
               ((NW 10VDC)      (NW 30VDC G))))))

(VC-REG
  ("DWUNZ" "9-Oct-85 21:24:00" "DWUNZ" "10-Oct-85 13:50:03")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((DUAL.R.ZD.REGULATOR (R3 D4 R4 D5))))
   (WAVEFORMS (((NW 0VDC)        (NW 0VDC U)
                                 (HW 36V.PEAK B)
                                 (FHW 36V.PEAK B))
               ((HW 36V.PEAK)   (HW 36V.PEAK B))
               ((FHW 36V.PEAK)  (FHW 36V.PEAK B))
               ((CFHW 30V.PEAK) (FHW 36V.PEAK G))
               ((CHW 30V.PEAK)  (HW 36V.PEAK G))
               ((NW 30VDC)      (FHW 36V.PEAK G))))))
```

```
(VC-FILT1
  ("DWUNZ" "9-Oct-85 21:24:00" "DWUNZ" "10-Oct-85 13:42:21")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (C1))))
   (WAVEFORMS (((NW 0VDC)       (NW 0VDC U)
                                (HW 36V.PEAK B))
               ((HW 36V.PEAK)   (HW 36V.PEAK B))
               ((FHW 36V.PEAK) (HW 36V.PEAK G))))))

(VC-RECT1
  ("DWUNZ" "9-Oct-85 21:24:00" "DWUNZ" "10-Oct-85 13:39:50")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (D3))))
   (WAVEFORMS (((NW 0VDC)      (NW 0VDC U)
                               (SW 72V.P-P B))
               ((HW 36V.PEAK) (SW 72V.P-P G))))))

(VC-ACSRC
  ("DWUNZ" "9-Oct-85 21:23:58" "DWUNZ" "10-Oct-85 13:40:17")
  NIL
  (CIRCUITS)
  NIL
  ()
  ((FUNCT-CAUSE
     ((SINGLE.COMPONENT (T1-1))))
   (WAVEFORMS (((NW 0VDC)     (NW 0VDC U)
                              (SW 140V.P-P B))
               ((SW 72V.P-P) (SW 140V.P-P G))))))

(WAVE
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:40")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
  ((DISP.FORM (LAMBDA (SELF X Y)
                (BITBLT* (EVAL (CAR (GET.VALUES SELF 'FORM)))
                         0
                         0
                         (TV:WINDOW-UNDER-MOUSE)
                         X
                         Y))
              METHOD
              (METHOD))
   (FORM NIL NIL NIL NIL NIL))
  ())
```

```
(SW
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:40")
  NIL
  (WAVE)
  "Sinewave"
  ()
  ((FORM (SINEWAVE))))

(FW
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "Fullwave"
  ()
  ((FORM (FULLWAVE))))

(HW
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "Halfwave"
  ()
  ((FORM (HALFWAVE))))

(NW
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "No (flat) wave"
  ()
  ((FORM (FLATWAVE))))

(FFW
  ("DWUNZ" "14-Feb-86 11:47:30" "DWUNZ" "14-Feb-86 11:50:00")
  NIL
  (WAVE)
  "Filtered fullwave"
  ()
  ((FORM (FILTERED-FW))))

(FHW
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "Filtered halfwave"
  ()
  ((FORM (FILTERED-HW))))

(CFW
  ("DWUNZ" "5-Feb-86 21:57:39" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
```

```
  "Clipped fullwave"
  ()
  ((FORM (CLIP-FW))))

(CHW
  ("DWUNZ" "5-Feb-86 21:57:39" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "clipped halfwave"
  ()
  ((FORM (CLIP-HW))))

(CFHW
  ("DWUNZ" "5-Feb-86 21:57:39" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  "Clipped filtered halfwave"
  ()
  ((FORM (CLIP-FHW))))

(UNK
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:40")
  NIL
  (WAVE)
  NIL
  ()
  ((FORM (UNKNOWN))))

(OTH
  ("DWUNZ" "5-Feb-86 21:57:40" "DWUNZ" "5-Feb-86 21:57:41")
  NIL
  (WAVE)
  NIL
  ()
  ((FORM (OTHER))))

KBEnd
```

DEKLEER-DIAG Sample Run Using the FAULT Option

Figures 10 through 14 show the diagnostic run for the command **(diagnose 'voltage-control 'fault)**. Figures 10 through 12 show the "FLATWAVE 0VDC" option being selected for each of the outputs. Figure 13 shows the "CLIP-FHW 30V.PEAK" option being chosen, and figure 14 displays the final output in the Lisp Listener window listing the good, unknown and bad components found in the diagnostic run.

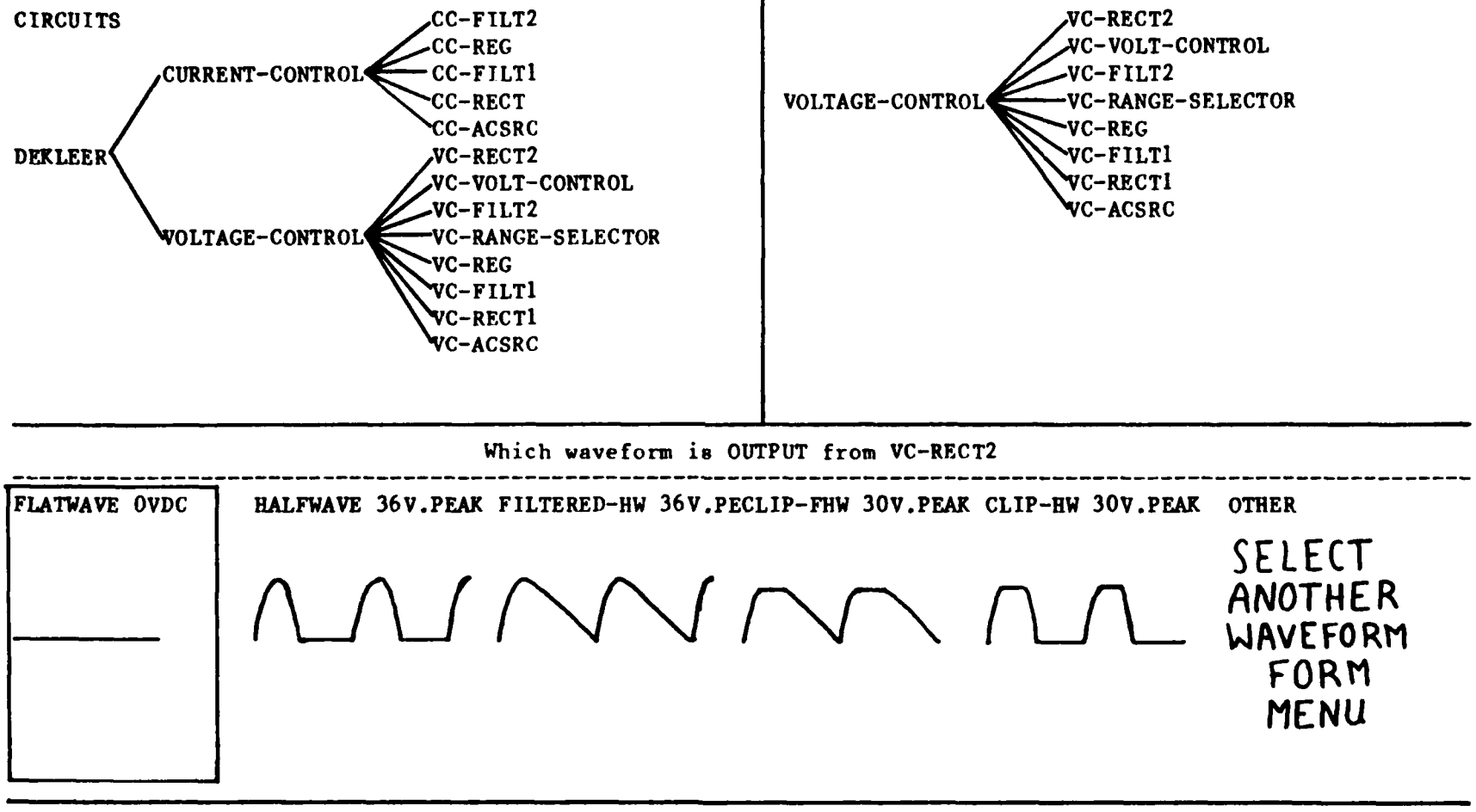


Figure 10: DEKLEER Fault Test - Frame 1

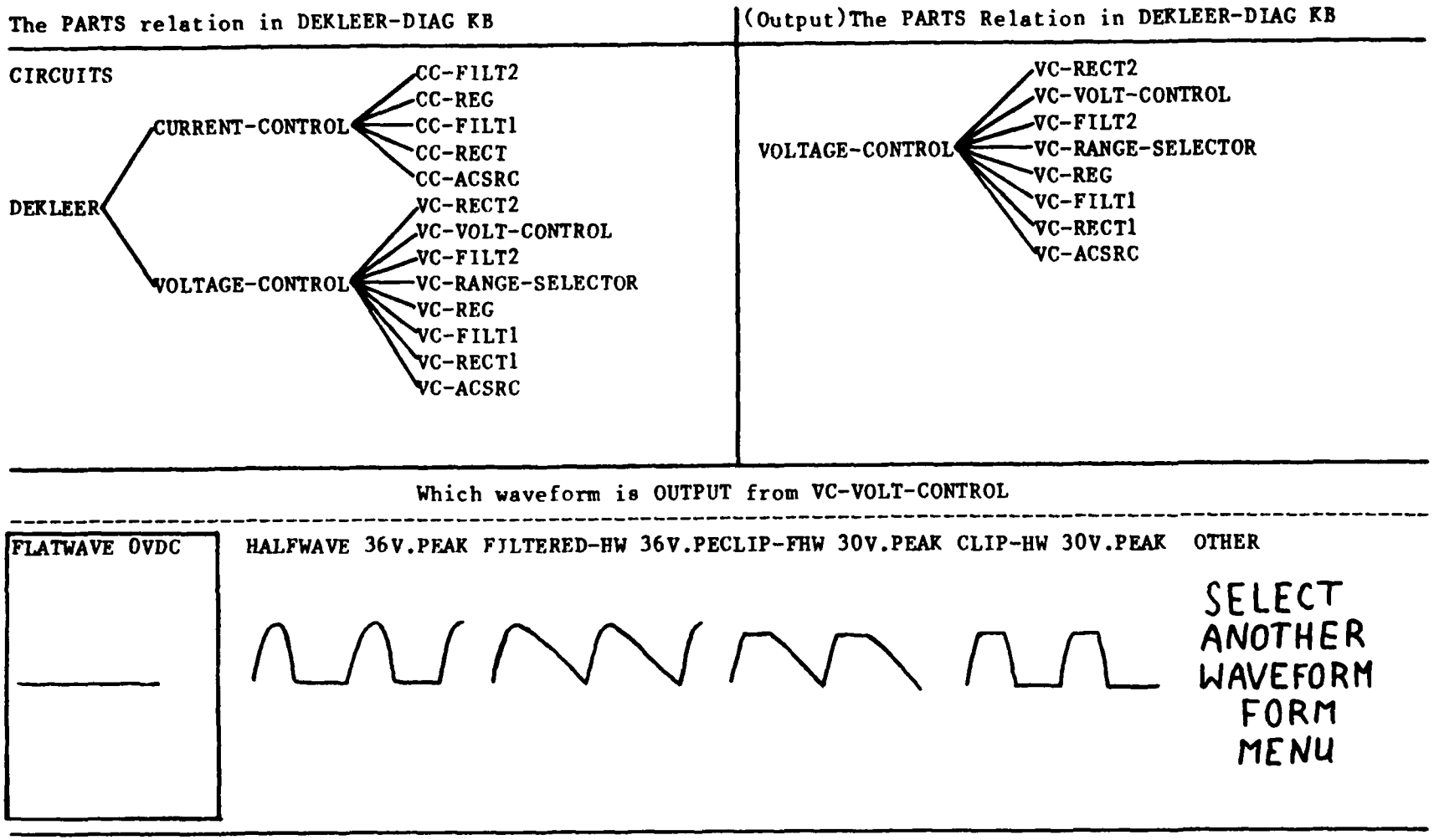


Figure 11: DEKLEER Fault Test - Frame 2

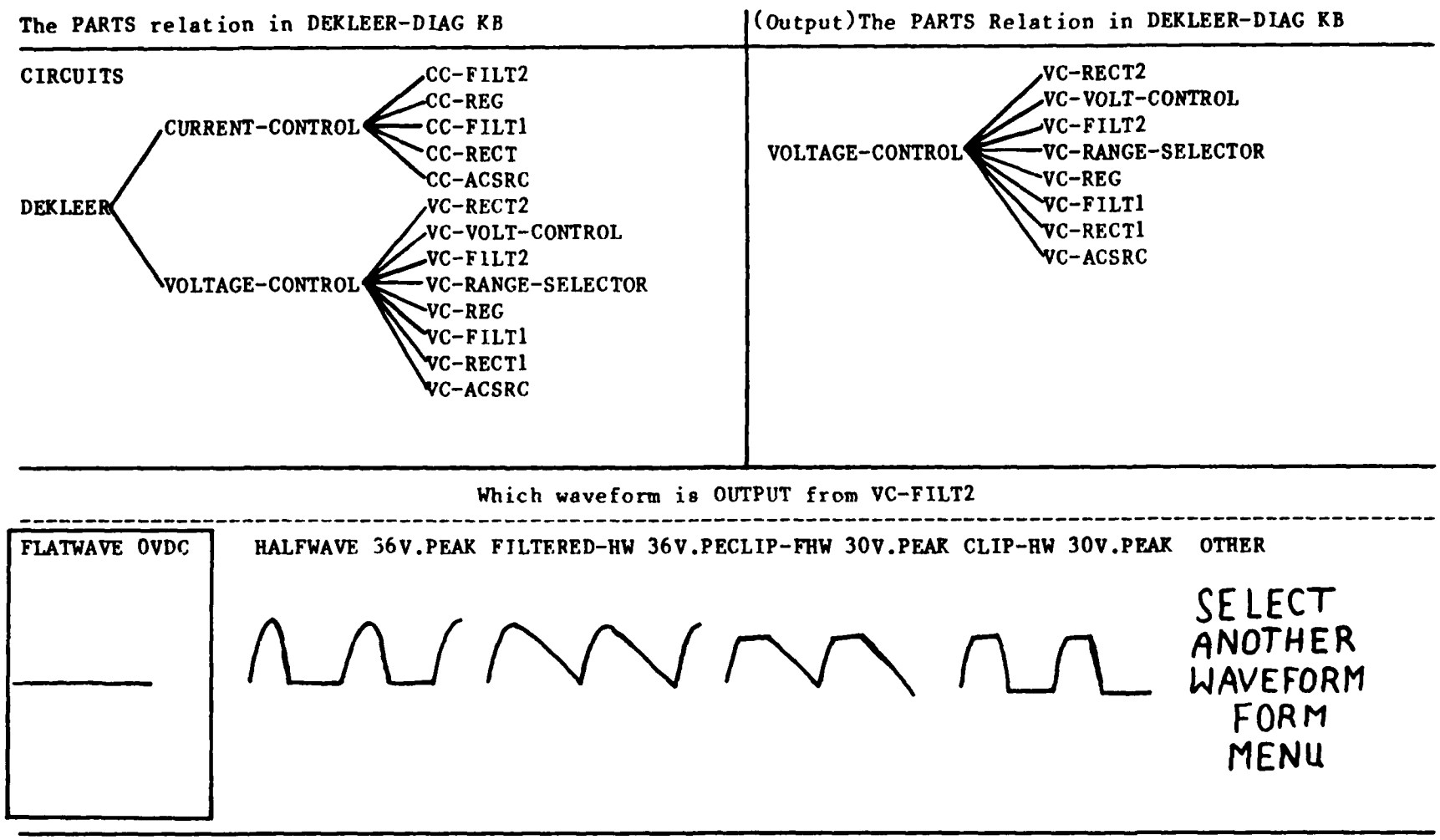


Figure 12: DEKLEER Fault Test - Frame 3

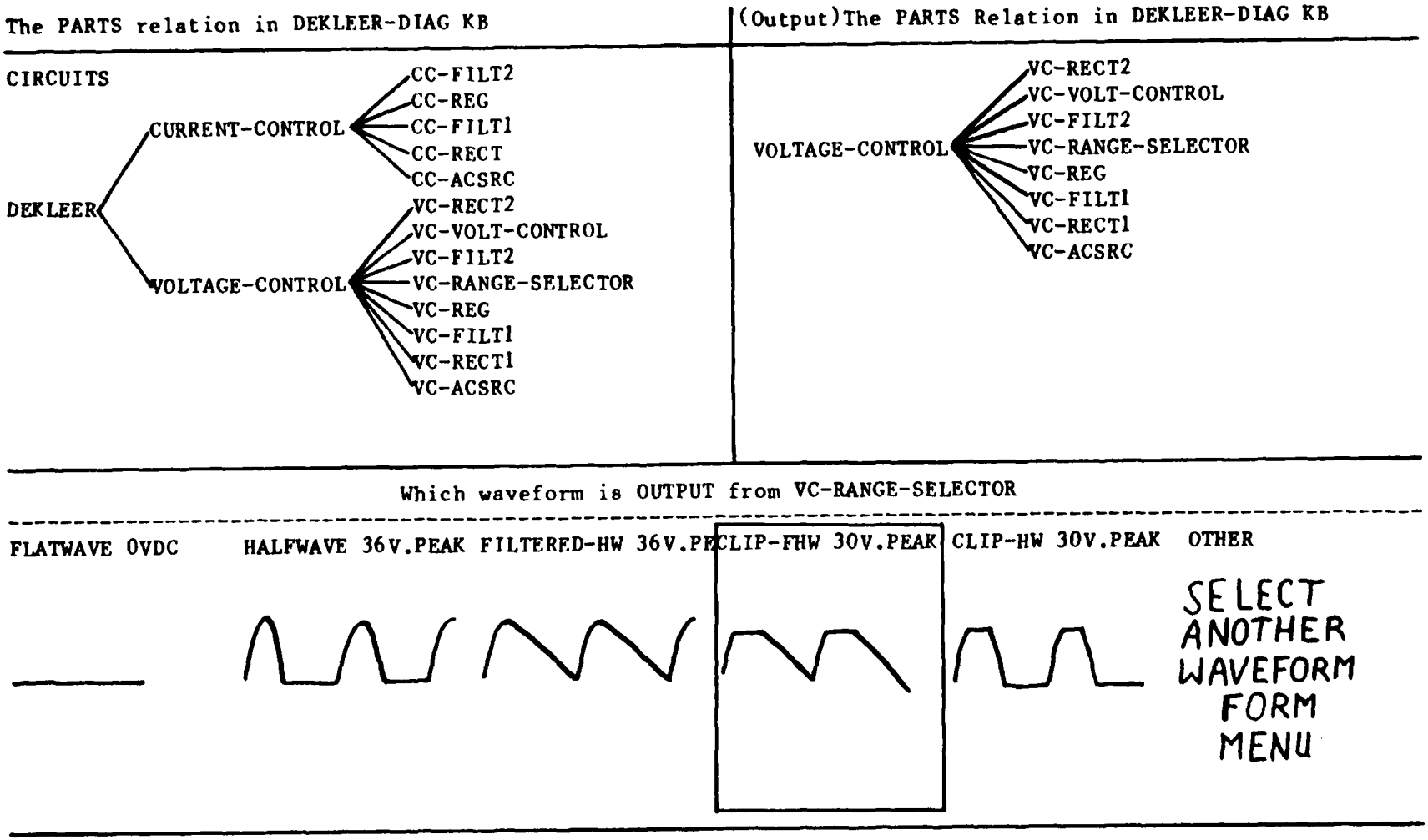
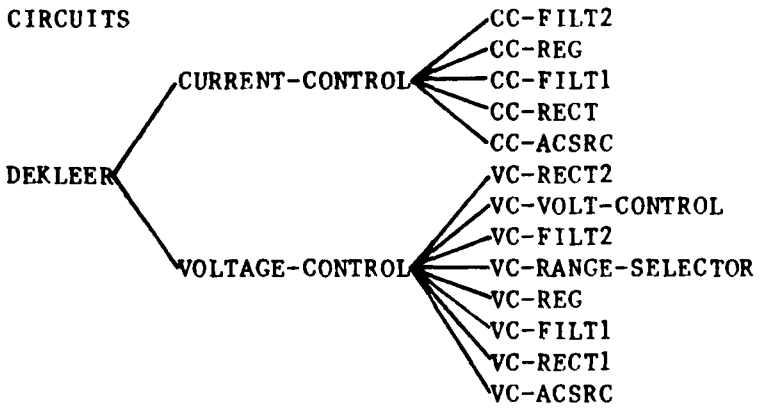
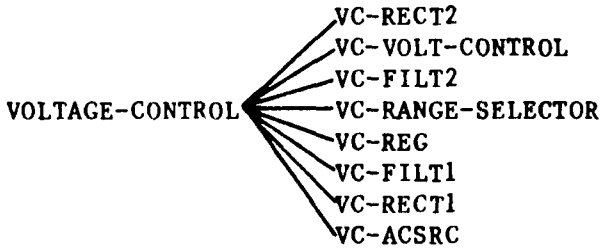


Figure 13: DEKLEER Fault Test - Frame 4

The PARTS relation in DEKLEER-DIAG KB



(Output)The PARTS Relation in DEKLEER-DIAG KB



Lisp Listener

```
(diagnose 'voltage-control 'fault)

List of good components =
    VC-REG
    VC-FILT1
    VC-RECT1
    VC-ACSRC
List of unknown components =
    VC-RECT2
    VC-VOLT-CONTROL
List of bad components =
    ((C2) in VC-FILT2)
    ((One or both of (R5 R6)) in VC-RANGE-SELECTOR)
```

Figure 14: DEKLEER Fault Test - Frame 5

Vita

Donald R. Wunz, Jr. was born on October 22, 1952 in DuBois, Pennsylvania He graduated from high school in Erie, Pennsylvania in 1970 and enlisted in the Air Force in March 1971. He was trained as a base level computer programmer and served nine and one half years in various computer programming jobs. He attended Southern Illinois University at Edwardsville and graduated with a BS in Mathematics / Computer Science Option in 1980. He was commissioned and continued in the Air Force as a systems analyst for the Airborne Warning And Control Division until May 1984 at which time he was selected to attend the School of Engineering, Air Force Institute of Technology.

Permanent address:

General Delivery

Patrick Air Force Base, Florida

32925

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A172 718

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/86M-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Cent.; 8b. OFFICE SYMBOL (If applicable): RADC/RBET	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Rome Air Development Center, Griffiss AFB, NY 13441	10. SOURCE OF FUNDING NOS.
11. TITLE (Include Security Classification): See Box 19	PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO

12. PERSONAL AUTHOR(S): Donald R. Wunz, Jr., B.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1986 March	122

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Artificial Intelligence, Expert System, Electronic Diagnosis
06	04		

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

Title: DIAGNOSIS OF ANALOG ELECTRONIC CIRCUITS: A FUNCTIONAL APPROACH

Thesis Chairman: Stephen A. Cross, Major, USAF
Associate Professor of Electrical Engineering

Approved for public release IAW AFR 190-1.
LYNN E. WOLAVER 9 May 86
Dean for Research and Professional Development
Air Force Institute of Technology (AFSC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Stephen A. Cross, Major, USAF	513-255-3576	AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This describes and proposes an analog electronic circuit diagnostic system using functional knowledge and an artificial intelligence expert system. A prototype of the system is implemented using the KEE expert system building tool to demonstrate its applicability.

Functional knowledge is used to analyze the circuit instead of just the structural information widely used now. This permits a more specific identification of bad components within a circuit. The current system in use at Warner Robins Air Logistic Center returns lists of bad components ranging in length up to 25 and 30 individual parts. Functional reasoning will enable the system to further restrict the specification of bad parts.

Research into the current literature provides the background and basic knowledge needed to determine the required information for the diagnostic system knowledge base.

The system described in this paper proposes the use of a functional rule knowledge base in addition to a structural data base to perform a more complete diagnosis than is now done by the system in use at some Air Force logistic centers.

The data base is searched using the functional knowledge of the circuit components. This knowledge is represented for each component by its effect on the signal passing through it. The input and output waveform pairs (used to represent the effect) that are possible for each component are stored as slot values with a flag indicating the condition of the component given a specific input/output pair.

After the component search is complete, each component identified as bad is then passed with its input/output waveform pair to a functional rulebase for analysis using design-engineer generated rules defining the performance of the component and its comprised parts. These rules then pass back the parts identified as bad as a list of suspect components. The list is then displayed for the technician to use as a replacement guide.

END

11-86

DTIC