REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED Final Report

4. TITLE AND SUBTITLE
EXPERT SYSTEM FOR CONTROL OF PLASMA, BEAM AND WAVE DYNAMICS IN MICROWAVE TUBES

5. FUNDING NUMBERS
~~F49620-93-C-0019~~
2301-ES
61102F

6. AUTHOR(S)
H. Bacher, R. Begum, T.A. Hargreaves, R. Rogers, J. Siambis, A. Theiss, and R. Vaughan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Litton Industries, Inc.,
Electron Devices Division
960 Industrial Road
San Carlos, CA 94070

8. PERFORMING ORGANIZATION
AFOSR-TR-95
0500

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research -NE
Department of the Air Force
Bolling Air Force Base, DC. 20332

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-93-C-0019

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Three major subsystems were developed for a klystron expert system. These subsystems monitor and control the cathode temperature, the rf input response, and the rf output response. For the temperature-controlled-cathode subsystem, a Fortran code was developed to predict the steady state cathode temperature from transient heater voltage data. For the rf-input subsystem, a circuit having two simultaneously variable geometry adjustments was developed. This circuit symmetrizes the frequencies of and maintains the amplitude of the rf signal at the input cavity. A newly developed expert-system software package successfully demonstrated adjustments of this circuit. A lumped-element model of a two-cavity extended interaction output cavity was developed and used to successfully predict the measurements on a two-cavity cold-test model. This circuit model is ready for use in a large-signal two-gap interaction code. A one-gap large-signal interaction code was written. The code includes relativistic effects, velocity and density modulations, space charge effects, potential energy changes and dynamic beam loading.

14. SUBJECT TERMS Expert system, smart klystron			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Litton

Electron Devices

EXPERT SYSTEM FOR CONTROL OF PLASMA, BEAM AND WAVE DYNAMICS IN MICROWAVE TUBES

Prepared for

DR. ROBERT BARKER

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
DEPARTMENT OF THE AIR FORCE
BOLLING AIR FORCE BASE, DC 20332-6558

July 20, 1995

BY

LITTON SYSTEMS, INC.
ELECTRON DEVICES DIVISION
960 INDUSTRIAL ROAD
SAN CARLOS, CA 94070

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Three major subsystems were developed for a klystron expert system, those for monitoring and controlling the cathode temperature, the rf input response, and the rf output response.

For the temperature-controlled cathode subsystem, a Fortran code was developed successfully to predict the steady-state cathode temperature from a given transient heater voltage vs. time plot. This code modifies a Litton-proprietary Quickbasic program by adding features that can automate the program with an expert system and by adding highly descriptive two-dimensional variables that quickly identify the gun assembly parts whose heat transmission histories are being calculated. Copious program comments made the code easy to read and easy to modify.

The program includes the temperature dependence of thermal emissivities and thermal conductances of the gun-assembly materials as well as geometry factors that simplify complex thermal-resistance calculations. The program is easy to use, very fast to run, and does not require large computer memory.

For the input-cavity subsystem, the design, development and tests of an adjustable transformer resulted in a practical configuration. Tests showed that the input transformer circuit, a combination of a bandpass filter and impedance transformer which is simply referred to as a "transformer," broadens the bandwidth of the input cavity by enhancing the edges of the passband. The transformer has two continuously and simultaneously adjustable controls that are linked to the expert system commands. These controls are useful both for hot test and cold test. The settings of the new transformer are changed by a stepper motor which is driven by an expert system. The new input transformer and its drive fit inside the existing envelope of the klystron.

The expert system software developed in this program (TIPTOE) successfully demonstrated that a software system can perform the function of adjusting a matching transformer in an input cavity circuit. The matching transformer contained two adjustments resulting in a nontrivial system. The formalism was developed, and the software written to easily allow expansion to several variables and several different subsystems. For demonstration purposes, the actual circuit structure was replaced by a lumped-element circuit model simulated with the commercial SUPERSTAR program.

A lumped-element model of a two-cavity extended interaction output cavity (EIOC) was developed with sufficient accuracy to predict measurements on a cold-test model. The EIOC model is ready for use in a large-signal interaction program.

A program which models large-signal klystron interaction at a single gap was developed. With residual discrepancy levels not exceeding 1%, the 1670-line Fortran code includes relativistic effects, velocity and density modulations, space charge effects, potential energy changes and dynamic beam loading, all under large-signal conditions. Other existing klystron programs include some but not all of these effects.

2. INTRODUCTION

2.1 Background on Smart Systems

2.1.1 Other smart systems

Over the past 30 years, computer technology has maintained an ever-increasing influence on the operation of all types of systems both large and small. Among the more interesting developments in this area are "expert" or "smart" computer systems that control complex or time-consuming tasks which normally require highly trained personnel. These expert computer systems are typically based on a computer module equipped with the following:

1) sensors to measure the state of a system
2) a database to determine the expected behavior of the system
3) decision-making algorithms
4) adaptive controls that modify operation of the system.

Appendix 2A describes three operating smart systems that exemplify the following

1) troubleshooting in a system that has many independent coupled variables
2) enhancing performance of a millimeter wave linear amplifier
3) testing TWTs during manufacturing.

2.1.2 Microwave-tube smart systems

Still to be realized is an expert system to monitor and control vacuum-device performance while the device is in end-use operation. Such a system would continuously "watch" and adjust the tube after it is placed in service. An expert system, after detecting conditions that lead to performance degradation or permanent damage, can adjust the operating parameters to prevent or lessen the impact of the undesirable condition. Also, besides monitoring for fault protection (avoiding sudden degradation), and for ongoing performance optimization (avoiding gradual degradation), the system is constantly building a useful database. The system, which records, minute by minute, the tube's condition, identifies unusual happenings that might be correlated with performance difficulties and consequently provides critical data to facilitate failure analyses, to quickly rework the tube, or to adjust the design for performance improvement.

Implementation of a tube smart system is expected to be facilitated by the following:

1) existing technology of practical, mission-oriented expert systems
2) miniaturized sensors
3) miniaturized controls and actuators
4) miniaturized special purpose computers
5) reliable, adaptive and fast software.

2.2 Purpose - A Klystron Expert System

The purpose of this work was to develop three major subsystems of a klystron expert system such as that schematically illustrated in Fig. 2.2-1. In this figure, ten subsystems are linking a klystron to monitor-and-control software. Each subsystem is responsible for sensing the conditions of a particular tube component in order to enable the master software program to control the component's functions. Subsystems, sensors and controls provided by an idealized master system are summarized in Table I.

Since the expert system intended for the klystron is a "model-based" system, the subsystem development includes modeling. In the subsystem, models are used to obtain predictions and expected information which are used in the decision algorithms for making adjustments or for evaluating measurement data. In this system the models can also be adjusted. Resulting discrepancies or errors between the model and measurements can be used to provide adjustments of the model parameters that results in new more accurate predictions. This adjustment process can be repeated until the errors are below specifications.

In this work the three subsystems are those shown in the figure as cathode temperature, rf input response and rf output response.

The cathode-temperature subsystem controls a thermionic electron source which represents a key feature of the device. To minimize malfunctions of (both oxide and dispenser) cathodes, the cathode-temperature subsystem must maintain operating temperatures over a narrow range. This subsystem gives "best guesses" on each of the internal components of the gun assembly to provide continually optimized emission.

The rf-input-response subsystem optimizes the signal entering the klystron by utilizing a circuit of distributed inductances, capacitances and line segments that together symmetrize the bandwidth and match the impedance of the input cavity with the signal generator. The circuit functions as a combination of a bandpass filter and an impedance transformer, the latter part being the more sensitive portion of the circuit. The subsystem contains a variable-geometry circuit, a model to predict expected input responses, and software to control the circuit geometries.

The rf-output-response subsystem optimizes the signal leaving the klystron's output circuit. This output circuit contains a double cavity where the beam converts into rf power a substantial portion of its kinetic energy in a highly nonlinear process. Control of this nonlinear interaction and power extraction requires special computer codes to accompany the adjustment in the output circuit. This subsystem utilizes special codes developed to determine the equivalent lumped-element shunt resistance of the multicavity output system during nonlinear operation.

TABLE I: SENSORS & CONTROLS REQUIRED FOR AN EXPERT SYSTEM

TUBE SUBSYSTEM	SENSOR & CONTROL REQUIRED
GUN ASSEMBLY	CATHODE TEMPERATURE/HEATER POWER AC VOLTAGE, BEAM CURRENT, EMITTANCE, ALIGNMENT.
FOCUSING MAGNET	SENSE MAGNETIC FIELD/MODIFY IN TIME AND SPACE. SEGMENT SOLENOID SO AS TO CHANGE IN CATHODE, COLLECTOR & INTERACTION REGION.
DRIFT TUBE	MONITOR DRIFT TUBE WALL TEMPERATURE & BODY CURRENT/ADJUST BEAM PARAMETERS.
INPUT CAVITY & TRANSFORMER	SENSE AND ADJUST INPUT CAVITY TUNING, BANDWIDTH AND INPUT LOAD MATCH WITH TRANSFORMER IN INPUT LINE.
INTERMEDIATE CAVITIES	SENSE TUNE AND BANDWIDTH/INSERT MECHANICALLY DRIVEN TUNERS FOR CAVITIES.
OUTPUT CAVITY & WAVEGUIDE	SENSE AND ADJUST OUTPUT CAVITY TUNE, BANDWIDTH & LOAD MATCH WITH TRANSFORMER IN OUTPUT WAVEGUIDE.
OUTPUT WINDOW	SENSE TEMPERATURE/ADJUST COOLING, ADJUST OTHER BEAM PARAMETERS. SENSE RESONANCE/SPOIL-RESONANCE.
BEAM COLLECTOR	SENSE TEMPERATURE/ADJUST COOLING, MAGNETIC FIELD. SENSE RESONANCE/SPOIL RESONANCE.
COOLING SYSTEM	TEMPERATURE AT KEY COMPONENTS & LOCATIONS/COOLANT FLOW ADJUSTMENTS.
MECHANICAL SYSTEM	ALIGNMENT, VIBRATION, STRESS/STRAIN.
VACUUM SYSTEM	VACUUM GAGE, VACUUM ION PUMP.

2.3 Scope - Subsystems for Major Components

The scope of this program was limited to the implementation of three subsystems that control the cathode temperature, the response of the rf input, and the response of the rf output. These were zero-order implementations for which refinements and next-order improvements were beyond the scope of the work.

For example, in the cathode-temperature work, there are still temperature gradients in some nodes of the model used for predicting temperatures of the gun assembly components.
Refining the temperature predictions by restructuring the nodes was outside the scope of the work. Other examples of out-of-scope work include (1) writing the code to accommodate a generic gun assembly instead of only a specific assembly, (2) incorporating facilitating features into the code such as inputing data interactively instead of by modifying either the code or the input data file and (3) adding subroutines that allow the code output to interface immediately with voltage and current controls.

Similarly, for the rf-input-response subsystem, some refinements necessary for a working expert system were outside the scope of this work including (1) adding either mechanical or electrical stops to both the axial and azimuthal drives, (2) removing backlash from both drives, (3) reducing rotational compression in the flexible cable, (4) adding locks to both adjustments, (5) adding searching software to find the best of all possible maximum-performance settings instead of finding a local-maximum setting, and (6) adding subroutines to interface with actual hardware.

For the klystron program, interaction beyond that of a single cavity, e.g., interaction with a multi-gap cavity or with a multi-cavity output, was outside the scope of the work.

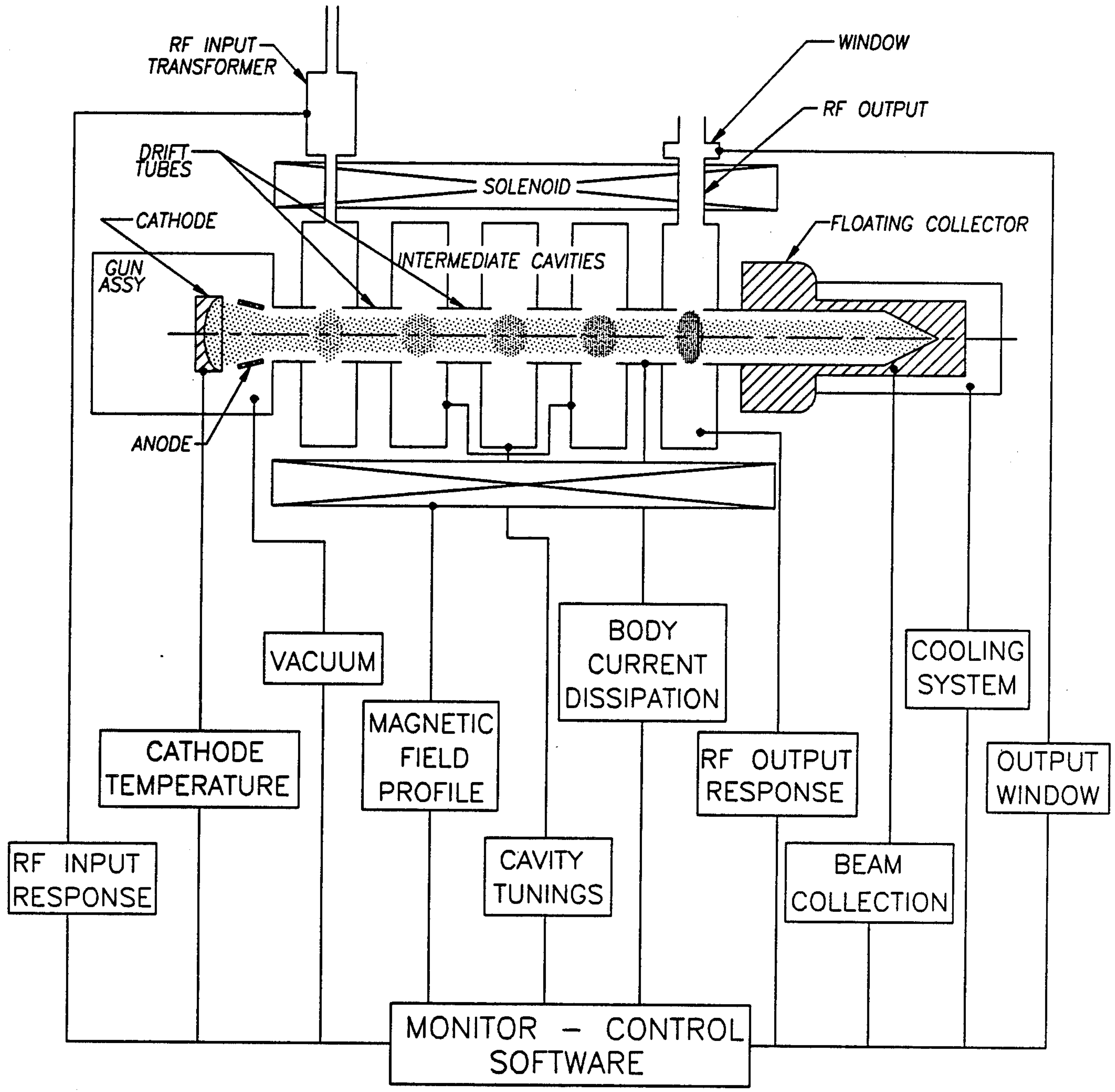


FIGURE 2.2-1
A KLYSTRON EXPERT SYSTEM

APPENDIX 2A
EXAMPLES OF EXPERT SYSTEMS

The following give examples of "expert" or "smart" computer systems that control complex or time-consuming tasks that normally require highly trained personnel.

1) Troubleshooting in systems that have many independent, coupled variables.

The expert system at the Stanford Linear Accelerator Center (SLAC) uses sensors for monitoring the position of charged particle beams that drift inside tunnels. When the beam wanders from a desired position, steering-coil currents that form magnetic fields to guide the particles must be re-set. The expert system achieves this complex task by recording beam location changes that result from combinations of experimental coil-current changes. After comparing the measured position shifts with the shifts predicted by a computer model of the beam-magnet system, the expert system can make final steering-coil adjustments that return the beam to its desired position.

2) Enhancing product performance.

An expert system that extends the linearized bandwidth of a high-power millimeter-wave amplifier exemplifies product performance enhancement. Typically, without an expert system, an amplifier's bandwidth is determined by design features that fix beam voltage, cathode current, rf drive and maximum allowable gain ripple. With the expert system, flat output power is obtained even with less-than-ideal gain ripple by programming each frequency with its own beam voltage, beam current and rf input level. The expert system can change these parameters in less than a microsecond to generate flat frequency sweeps having bandwidths far in excess of the typical amplifier.

3) Testing products during manufacturing.

The expert system that automates TWT tests at Litton EDD exemplifies product testing. After technicians bake out, hipot, and focus the beam of the TWT, an expert system takes over. Measurements, data reduction and plots are made for multiple tests including small signal gain, drive for constant power output, saturated power, transfer functions, Miram curves, standard roll-off curves, cut-off values, and tube aging both with and without rf. Despite appearances, the above tasks represent a small fraction of the expert system's program. The large portion of the program concerns decisions the expert system is expected to make during tests. (For example, while aging the TWT with rf, if the TWT becomes gassy, the system may reduce the rf drive and repeat the previous command. Or, if an arc shuts down the power supply, the expert system will check prescribed conditions then replace voltages in a prescribed order, then check TWT conditions, then resume tests.) Such automated testing is a standard part of TWT manufacturing at Litton EDD.

3. CATHODE TEMPERATURE SUBSYSTEM

The expert system for any thermionic device such as a klystron must necessarily include temperature control of the electron emission surface. Too high temperatures can often be associated with heater or cathode malfunctions and too low temperatures can often be associated with degradation in rf gain and power. Minute-by-minute control of the cathode temperature assures optimal rf performance and maximizes the tube's MTBF.

One goal for this expert system is to design a model-based subsystem to control the cathode temperature. As described below for this case, the subsystem is the cathode-heater assembly. The model-base approach compares predictions of the heater-cathode model with experimental data then adjusts parameters in the model. Comparisons are repeated until any resulting discrepancy or error between the model prediction and the experimental measurement is reduced below required specifications.

After model predictions become satisfactory, the subsystem will be integrated into the overall expert system. As a result, the output from the computer model will control switches which change the heater voltage and current such that the heater power maintains the desired cathode temperature.

3.1 Description of Program

To achieve control of the cathode temperature, a Fortran-language code CATHTEMP was developed to predict the cathode temperature for a given heater power. This code is an automated version of a QuickBasic program (see Litton technical report "A Better Method for Controlling the Cathode Temperature", R.M. Rogers, Litton Systems Inc., Electron Devices Division, 960 Industrial Road, San Carlos, CA 94070) developed for a klystron gun assembly similar to that used in this program. A listing of the program is given in Appendix 3-C and the programming steps are outlined in Appendix 3-A. The program uses an approach that greatly simplifies a complex geometry problem.

In general, for a given heater power, the cathode temperature is a function of (1) the thermal emissivities of the emitting surfaces of the cathode and heater, and (2) the conductivities and the emissivities of various cathode support parts. Although, in general, each part has cylindrical symmetry, each of the 43 gun-assembly parts radiates and conducts to the other parts from portions of conical, cylindrical, or disk-type (end plate) surfaces.

In the model, the geometry is simplified by grouping individual parts into "nodes" that have conduction and radiation linkages to each other (Fig. 3.1-1). The fundamental assumption of this nodal approach is that all parts of a node have the same temperature at a given time.

The inner diameters, outer diameters and lengths of parts are still needed to calculate the radiating and the conducting surface areas of the nodes. The input data to CATHTEMP are managed by an input data file CATHIN which is explained in detail in Section 3.3. For each part of the gun assembly, CATHIN requires the following input parameters:

--- inner diameter
--- outer diameter
--- length
--- density
--- specific heat
--- thermal conductivity
--- emissivity

CATHTEMP also requires the heater voltage and the heater coil electrical resistivity.

The program begins calculations at a time t = 0 which corresponds to the application of power to the heater. The heater current, which when multiplied with the heater voltage provides the power generated during each time increment, is computed by dividing the heater voltage by heater resistance.

Over each of the following time increments, the program calculates the heat transmitted to each node from all of the others. Typically the program runs for 1800 time steps, the first 60 being each 1 second and the remaining being 2 sec. The program stops when a steady state solution is achieved.

The net power flow to the k-th node from all other nodes is given by

$$P_k = P_{1tk} + P_{2tk} + P_{3tk} + \dots\dots + P_{k-1tk} \tag{1}$$

Here P_{1tk} represents the net power flow to k-th node from node 1. The power flow from node 1 to node 2 at a given time t is given by

$$P_{1t2}(t) = P_{1t2}^{conduction}(t) + P_{1t2}^{radiation}(t) \tag{2}$$

where

$$P_{1t2}^{conduction}(t) = \frac{T(1) - T(2)}{R_{1t2}} \tag{3}$$

Here R_{1t2} = thermal resistance between node 1 and node 2 at time t (deg.K/watts)
T(1) = temperature of node 1 at time t (deg.K)

T(2) = temperature of node 2 at time t (deg.K)

The second term in equation (2) represents the power flow due to radiation from node 1 to node 2 at time t and is given by

$$P_{1t2}^{radiation}(t)=\epsilon(T)\sigma F_{ij}A(T(1)^4-T(2)^4) \tag{4}$$

Here

$\varepsilon(T)$ = thermal emissivity of a radiating surface at temperature T
σ = Stefan-Boltzmann constant (watts/inch^2/K^4)
F_{ij} = view factor, fraction of radiation from surface i reaching surface j
A = area of the radiating surface
T(1) = temperature of node 1 at time t (deg. Kelvin)
T(2) = temperature of node 2 at time t (deg. Kelvin)

Once the net power flow to a node at a given time is known, the updated temperature of the node is given by

$$T_n=T_{n-1}+h*\frac{P_k}{C_t} \tag{5}$$

Here

T_n = temperature of node k at time step n (deg.K)
T_{n-1} = temperature of node k at time step n-1 (deg.K)
h = time step (sec)
C_t = thermal capacitance of node k (joules/kg/deg.K)
P_k = net heat flow to node k at previous time step (watts)

The thermal capacitance of node k is given by

$$C_t=\sum m_i c_i \tag{6}$$

Here

i = index of all parts belonging to node k
m_i = mass of i-th part
c_i = specific heat of i-th part.

3.2 Uncertainties

The following parameters are not known exactly:

F_{ij} = View factor
$\varepsilon(T)$ = thermal emissivity of a radiating surface as a function of temperature
R_{ktk+1} = thermal resistance between node k and k+1.

The view factor F_{ij}, the fraction of radiation leaving surface i which is intercepted by surface j, is given by (Fig. 3.2.-1) (see "Fundamentals of Heat and Mass Transfer", F.P. Incropera, D.P. Dewitt, John Wiley & Sons, 1985.)

$$F_{ij}=\frac{1}{A_i}\int\int\frac{\cos\theta_i\cos\theta_j}{\pi R^2}dA_i dA_j \qquad (7)$$

The above factor is unity when (1) two surfaces of equal area are parallel to each other, or (2) the intercepting surface completely surrounds the radiating surface. Otherwise, in general, the above integral is not easy to evaluate.

The temperature dependence of the thermal emissivity $\varepsilon(T)$ is only approximately known for most gun assembly surfaces. For example, the oxide cathode, one of the main radiating surfaces in a gun assembly has a granular BaO-on-Nickel surface that results from several chemical processes. The complex nature of this composite surface makes its thermal emissivity impractical to know. Similarly the radiation between heater and cathode surface is not known exactly due to imprecise knowledge of the thermal emissivity of the heater coil.

The thermal resistance R_{ktk+1} required for conduction calculations between two adjacent constant-temperature nodes is given by thermal resistances of parts located on the node boundary. The thickness of the interface between two nodes is the combined thicknesses of the two parts adjacent to the boundary. Besides various geometry factors, the thermal conduction calculations depend upon (a) the type of joints (spot weld, TIG weld or braze joint) used to join two adjacent parts, (b) the direction of heat conduction in the interface, and (c) the temperature variation of thermal conductivity of the interface. Since the thermal conductivities of different types of joints are unknown, the exact variations of thermal conductivities with temperature of different materials involved are unknown.

3.3 Description of the Input Data File

As shown by the sample file in Appendix 3B, the input data file uses namelist statements to represent a group of variables by one name. Each namelist starts with a '$' sign and ends with

'$end'. Typically, as with the first namelist '$heatcoil,' four variables (id(1,1), od(1,1), length(1,1) and MatType(1,1)) represent the inner diameter (id), outer diameter (od), length and material type for a given part. For variables cast as two dimensional arrays, the value of first dimension gives the node index and the value of the second dimension gives the part index.

The namelist names have both descriptive and numerical portions. For example, 'cath_support_466' stands for cathode support having Litton part number 372466. This report only uses the last three digits since the first three are common to all parts. The namelists $partl, $part2, etc. represent the variable jpart(i) ($1 \le i \le 10$; node index). The variable jpart(i) gives the total number of parts belonging to a given node.

The sixty-six namelist statements of the input file are separated by blank lines to form ten groups corresponding to ten nodes. Fig. 3.3-1A shows the part belonging to node 1 in the gun assembly drawing. Node 1 consists of the heater coil (part # 481) alone. Fig. 3.3-1B shows the heater coil in detail. The heater coil consists of 0.02875 inch tungsten wire which is 55 inches long. Tungsten is numbered as material type 1 in the code (see table below).

The second node includes the heater tab, the heat shields (part # 487, 488, 484), the cathode head (part # 474) and the cathode support (part # 473). Fig. 3.3-2A shows these parts (except the heater tab) in the gun assembly drawing. The heater tab is a portion of the heater coil which joins the coil to the cathode support. The individual part drawings for parts belonging to node 2 are shown in Figs. 3.3-2B through 3.3-2F. Notice that the lengths (compare the dimensions in the input file with those in the drawings) of some parts are obtained by averaging some relevant dimensions due to the bends in the cylindrical surfaces.

The third node consists of the cathode support (part # 466), the heat trap (part # 649), the shadow grid support (part # 425, 426) and a portion of part # 469. These parts are shown in Fig. 3.3-3A in the gun assembly drawing and detailed drawings are shown in Figs. 3.3-3B through 3.3-3F. Notice that part # 469 was divided into four separate sections (A,B,C, and D) for convenience of calculations of cross-sectional areas for conduction and radiation (Fig. 3.3-3C). Note also that the outer and inner diameters of part # 466 were obtained by weighted average of the dimensions shown in the drawing due to bends in the cylindrical surface.

The parts # 469 and 442 belong to node 4 and are shown in Figs. 3.3-4A, 3.3-4B and 3.3-3C. Figs. 3.3-5 shows the locations for nodes 5, 6, and 7. Fig. 3.3-6A shows part # 470 belonging to node 5 and Fig. 3.3-6B shows part # 496 belonging to node 6. The part drawings for the parts belonging to node 7 are shown in Fig. 3.3-7A and 3.3-7B. All parts (# 441, 440, 439, 437) belonging to node 8 are shown in Fig. 3.3-8A in the gun assembly drawing. The individual part drawings are shown in Figs. 3.3-8B through Fig. 3.3-8E. Notice that the part # 439 repeats twice.

Fig. 3.3-9 shows parts (# 435, 432, 431) belonging to node 9 in the assembly drawing and the individual parts are shown in Figs. 3.3-9B through Fig 3.3-9D. Fig. 3.3-10 includes all parts (# 651-654, 452, 448, 450, 446, 455, 454, 445, 457, 456, 029, 406, 409, 407, 414, 413) belonging to node 10. The individual part drawings are shown in Figs. 3.3-10A through Fig. 3.3-10S.

The materials from which the different parts of the gun assembly are made, are numbered in the code as follows:

Name of Material	# for Material Type
Tungsten	1
Molybdenum	2
Nickel	3
Stainless Steel	4
Kovar	5
Alumina	6

3.4 Output

Figure 3.4-1 is a plot of output from a preliminary version of the Fortran code. The two curves show the variation of temperature with time for nodes 1 and 2. Node 1 consists of the heater coil alone (Fig. 3.3-1). Node 2 includes heater tab, part of the cathode support, insulators and cathode head (Fig. 3.3-2). The heater voltage was assumed to be 14 volts for the first 10 minutes of operation then 13 volts thereafter.

The temperature of node 1 (heater) is higher than that of node 2 and the rise time of node 1 is much shorter than that of node 2 because the thermal capacitance of node 1 is much smaller than that of node 2. The small step shown in curve 1 is due to the change in heater voltage from 14 volts to 13 volts 10 minutes after heater turn-on.

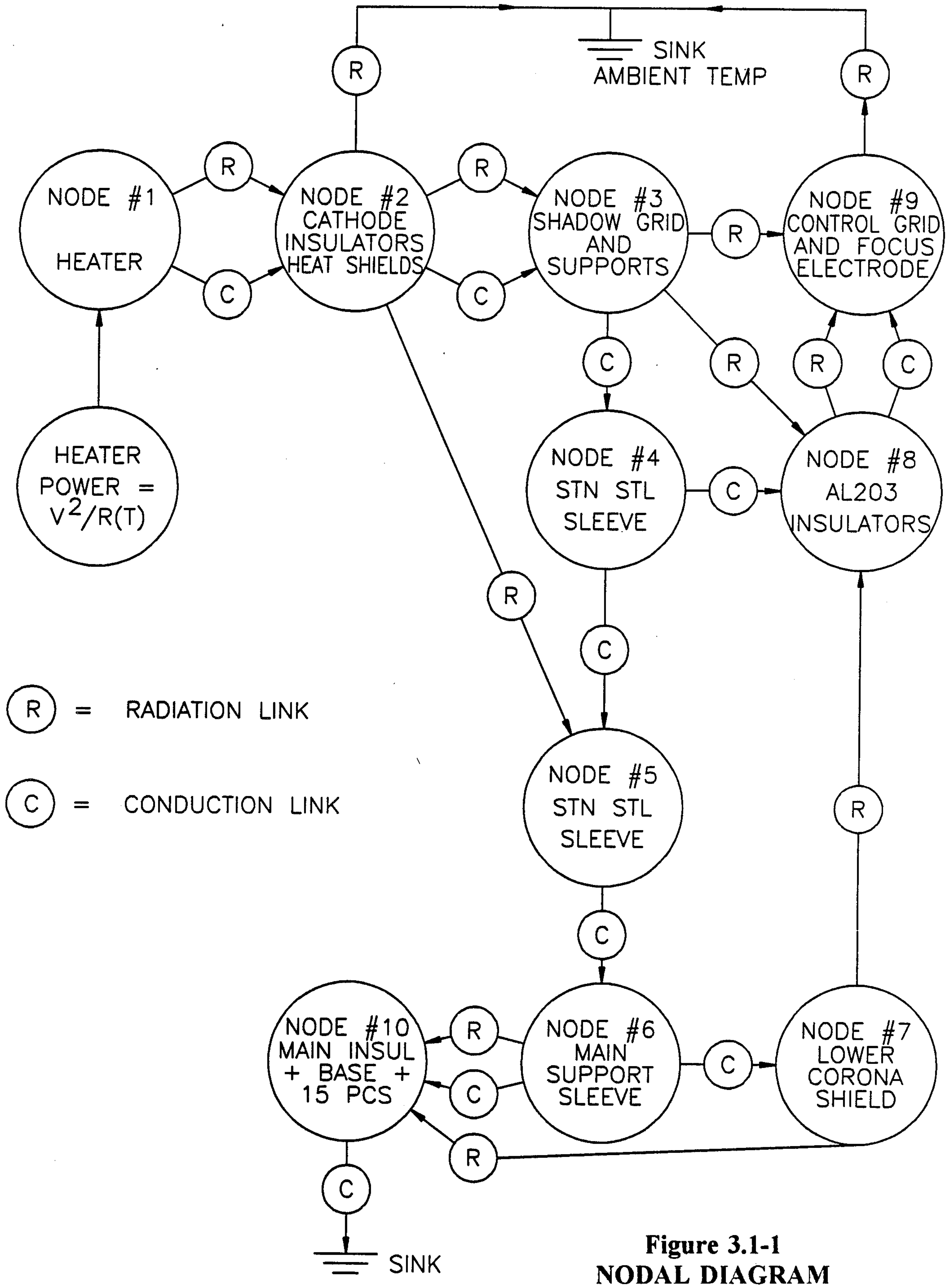
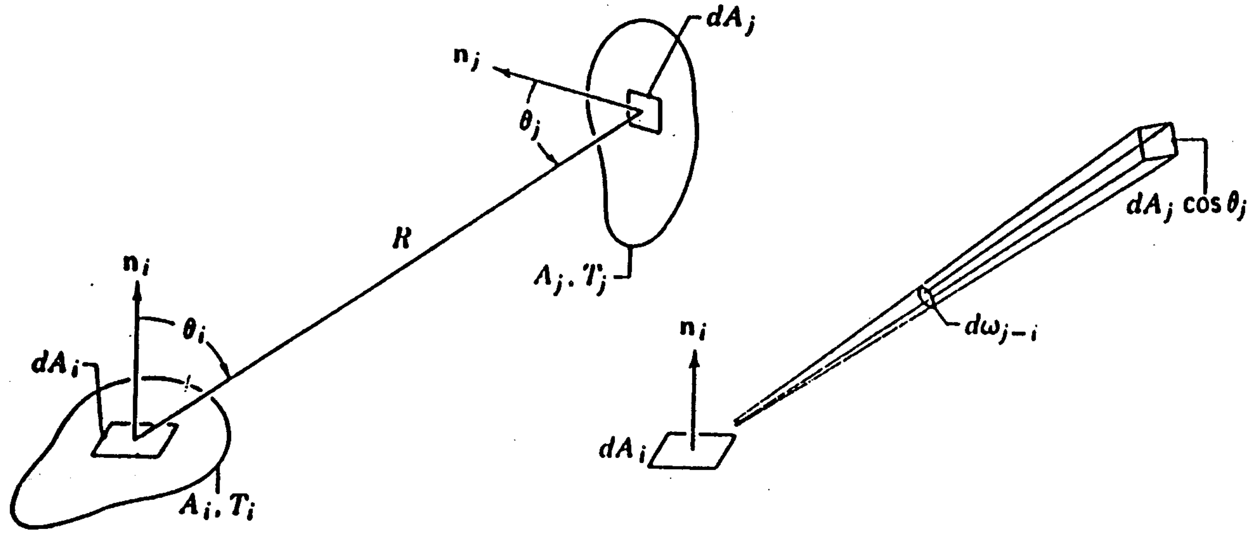


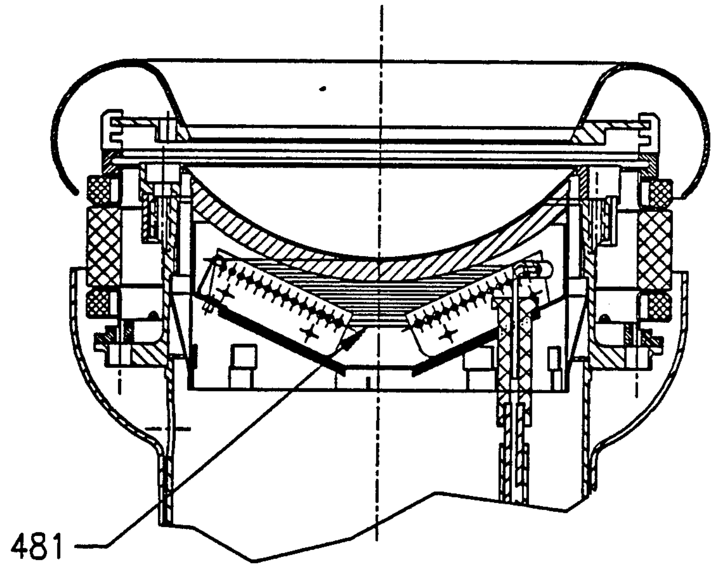
Figure 3.1-1
NODAL DIAGRAM



View factor associated with radiation exchange between elemental surfaces of area dA_i and dA_j.

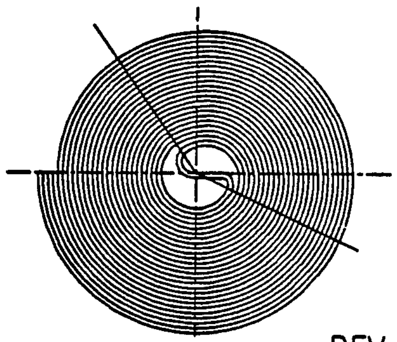
Figure 3.2-1
VIEW FACTOR

A



(NODE 1)

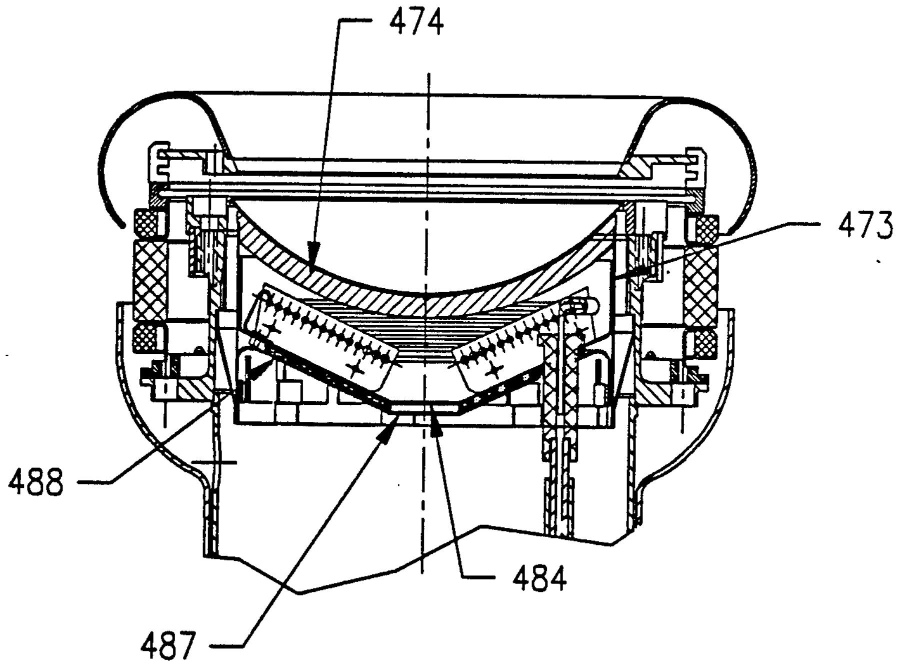
B



HEATER 372481
.02875 TUNGSTEN WIRE

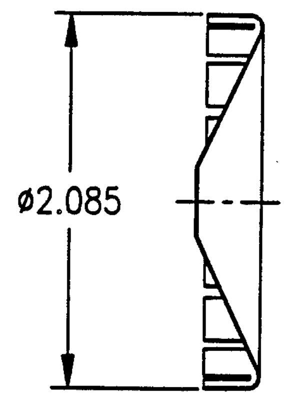
Figure 3.3-1
NODE 1

A

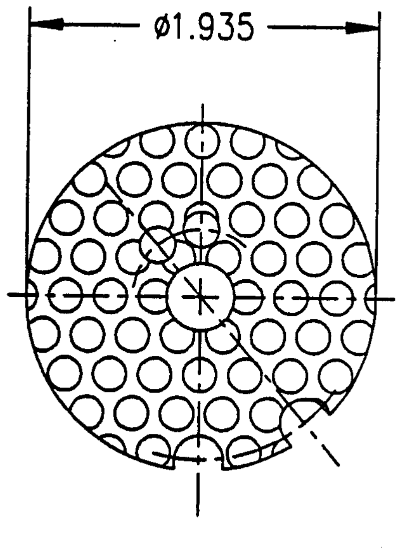


NODE 2

B



C

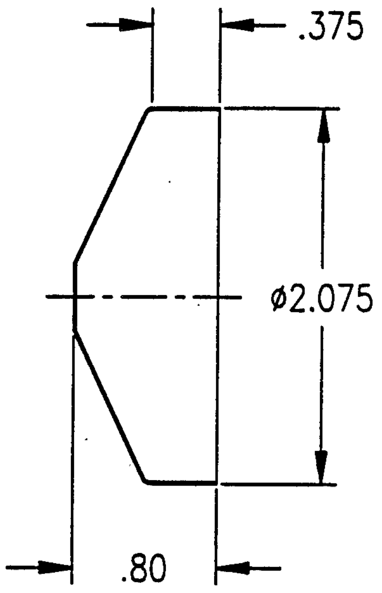


HEAT SHIELD 372487
.010 MOLY

SHIELD, HEATER HEAT 372488
MAKE FROM 436069

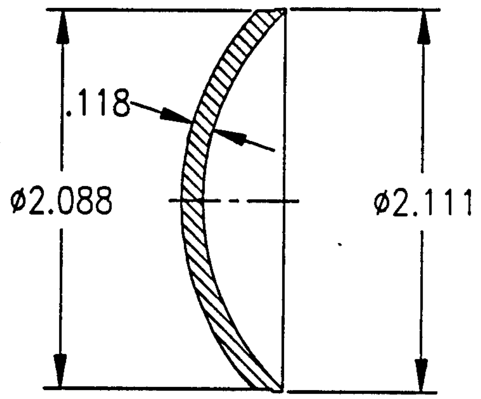
Figure 3.3-2
NODE 2

D



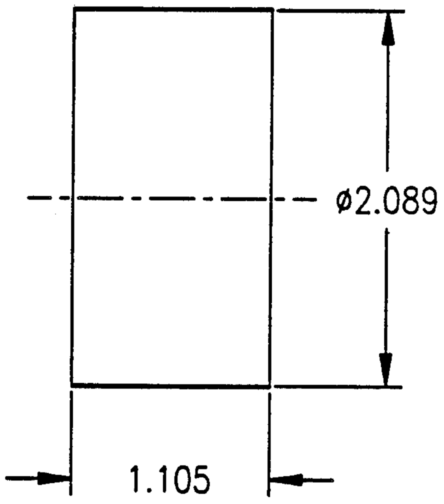
CUP-HEATER ASSY 372484
.010 MOLY

E



CATHODE HEAD 372474

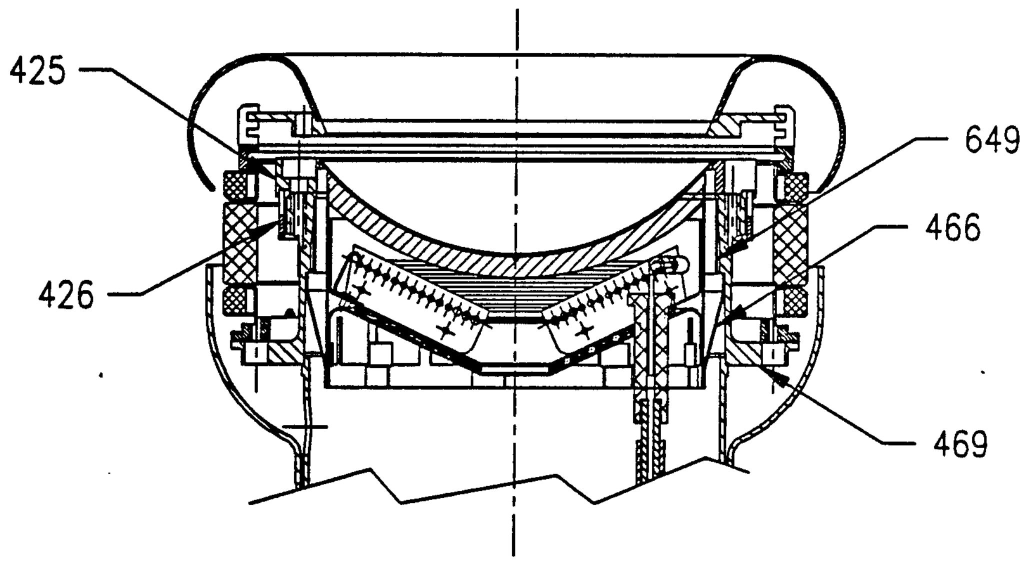
F



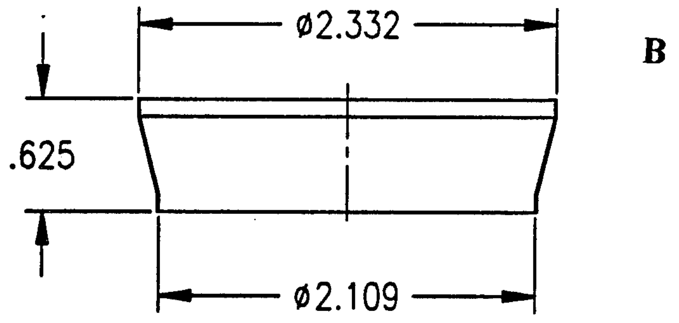
CYLINDER BUTTON 372473
.008 NICKEL

Figure 3.3-2 (CON'T)
NODE 2

A



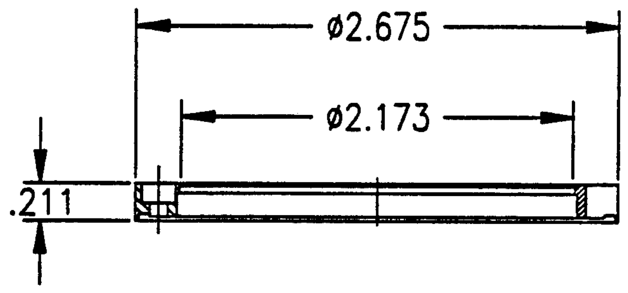
NODE 3



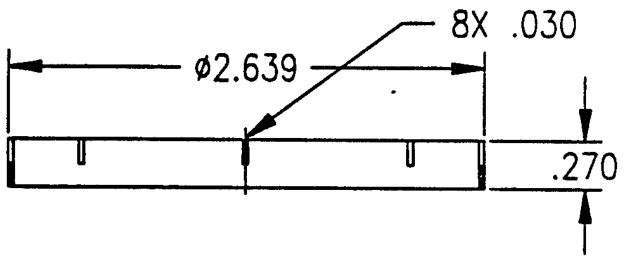
B

SUPPORT, BOTTOM CYL ASSY 372466
220 NICKEL

D



E

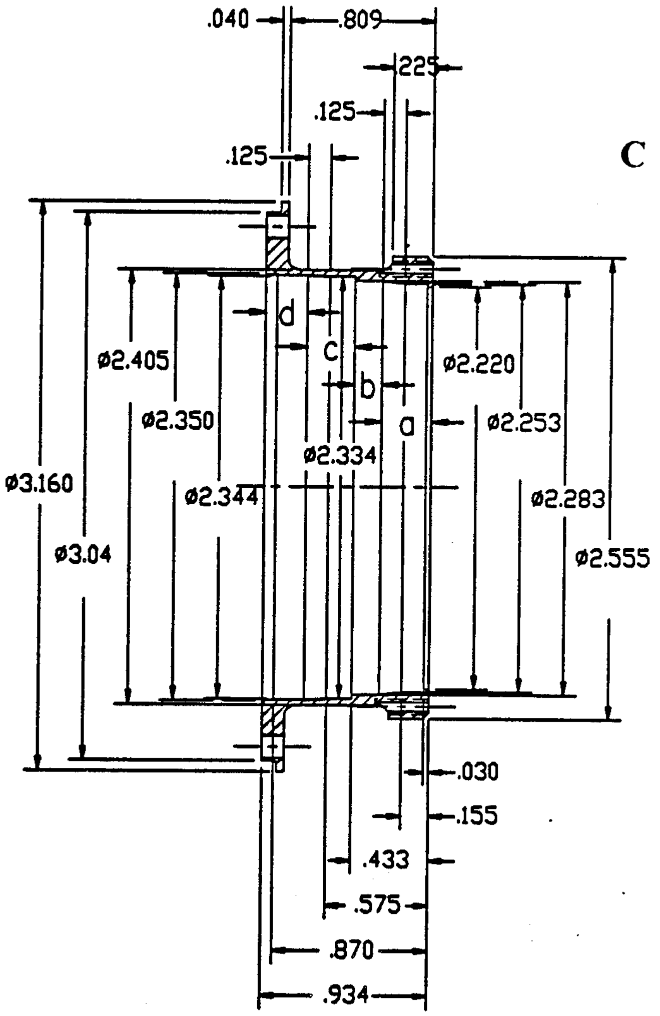


SHADOW GRID SUPPORT 372425
MOLY

SHADOW GRID SUPPORT CYLINDER 372426
KOVAR

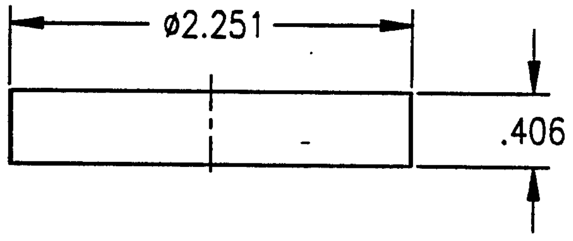
Figure 3.3-3
NODE 3

C



BUSHING 372469
304 SST

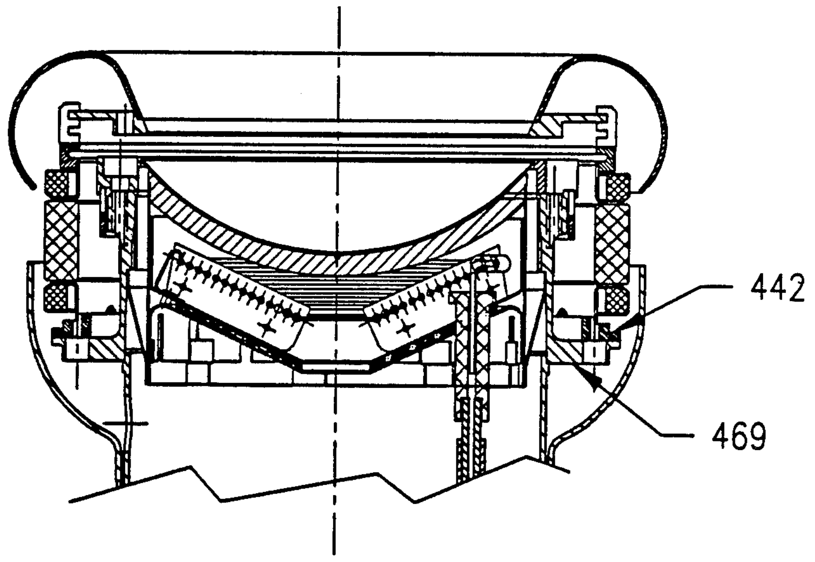
F



HEAT TRAP 372649
200 NICKEL

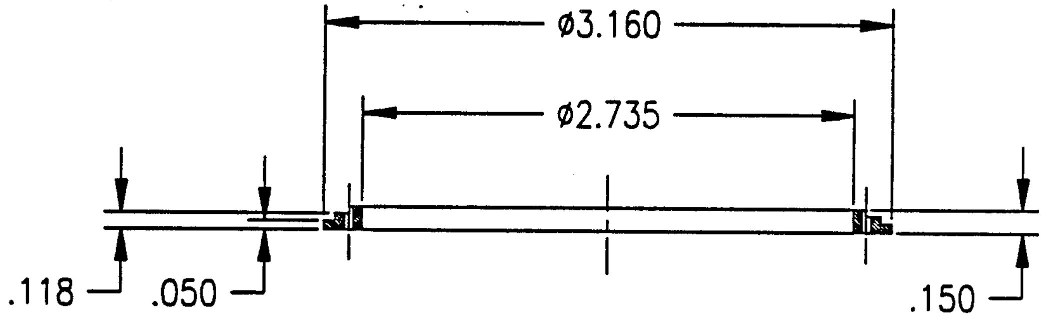
Figure 3.3-3 (CON'T)
NODE 3

A



NODE 4

B



CONTROL GRID SUPPORT 372442
KOVAR

(Node 4 also includes part in Fig. 3.3-3C)

Figure 3.3-4
NODE 4

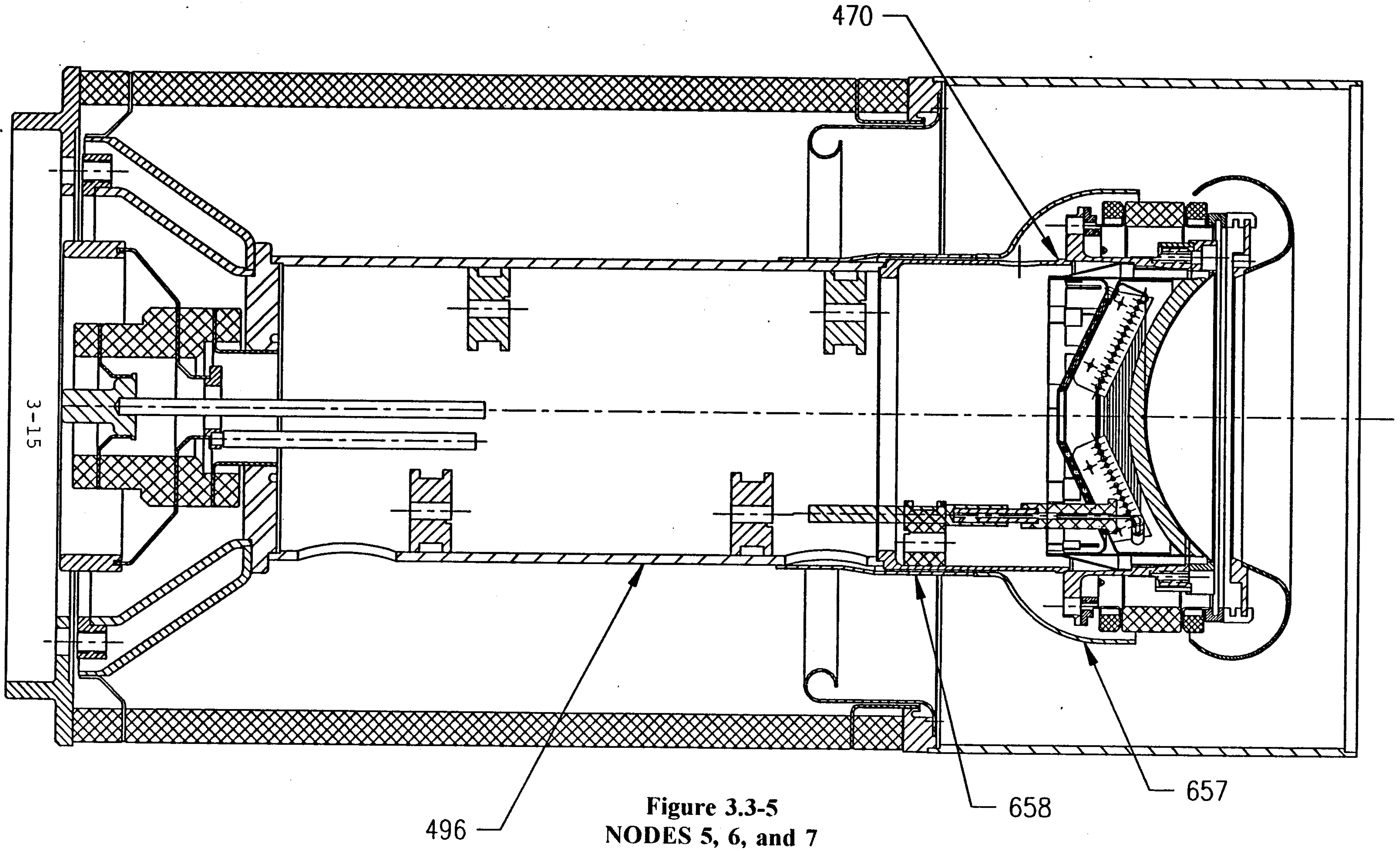
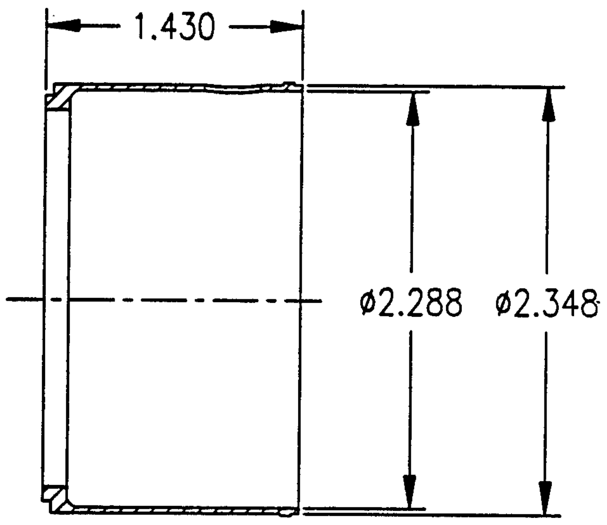


Figure 3.3-5
NODES 5, 6, and 7

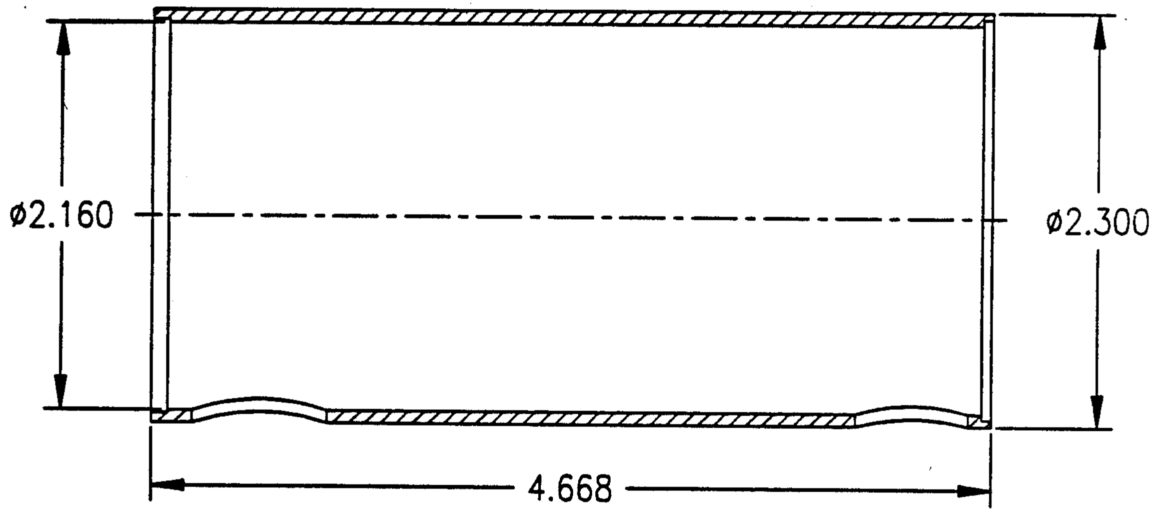
CATHODE LAYOUT
(AWACS)

A



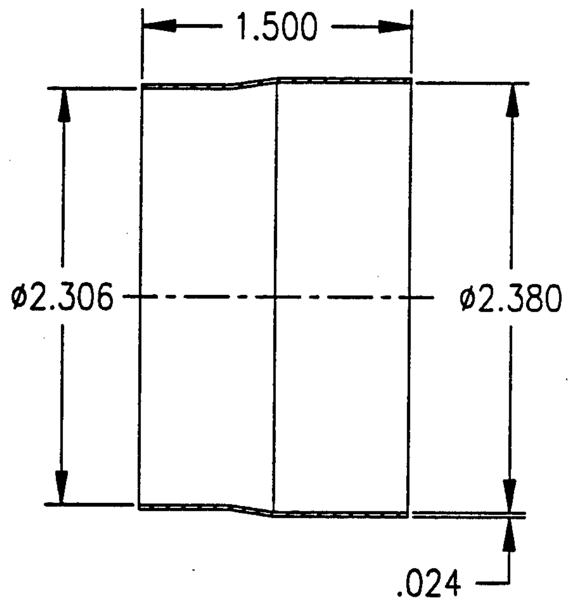
NODE 5
SUPPORT , BUSHING 372470
304L SST

B



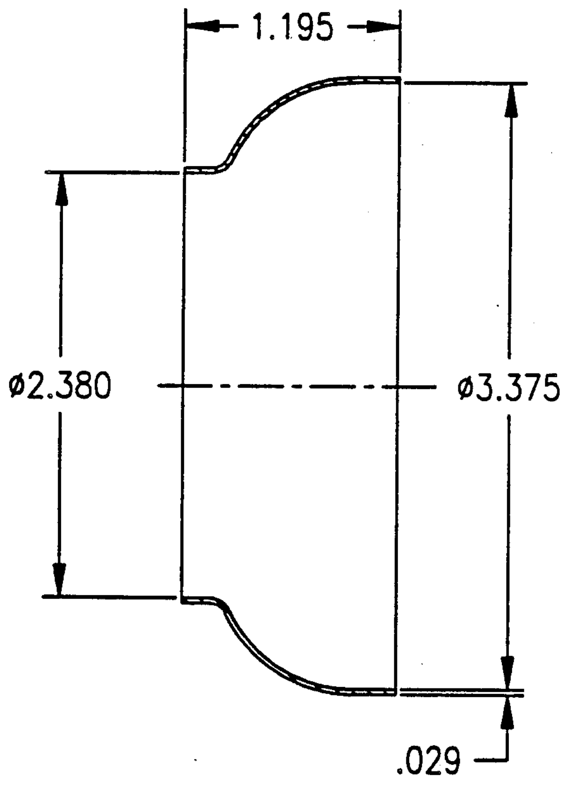
NODE 6
LOWER CATHODE SUPPORT SLEEVE 372496
304L SST

Figure 3.3-6
NODES 5 and 6



A

CYLINDER 372658
304L SST

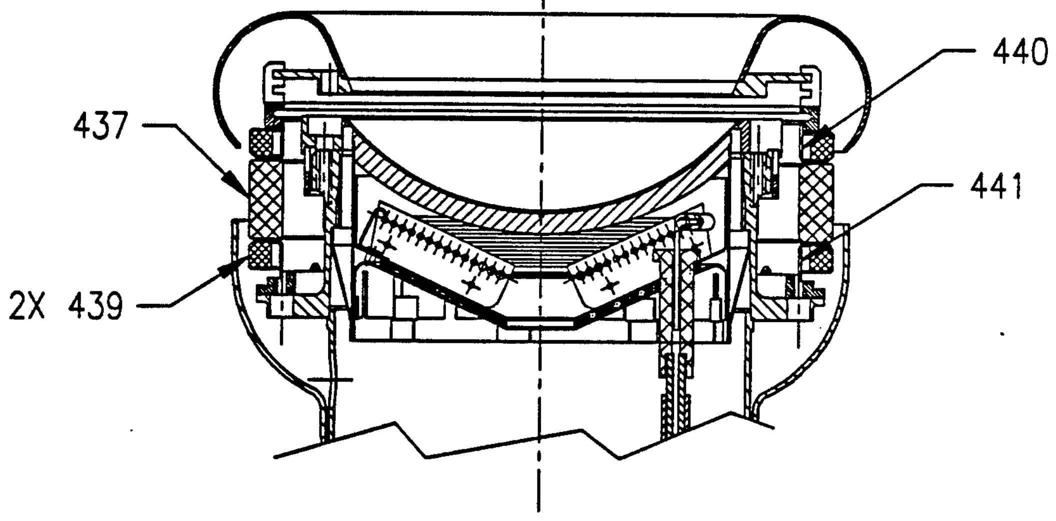


B

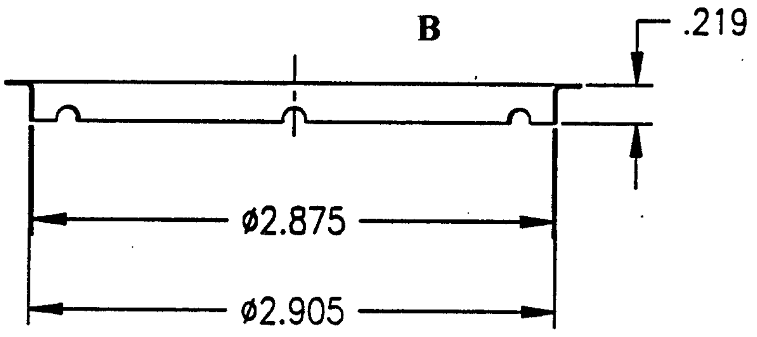
CORONA SHIELD 372657
304L SST

Figure 3.3-7
NODE 7

A

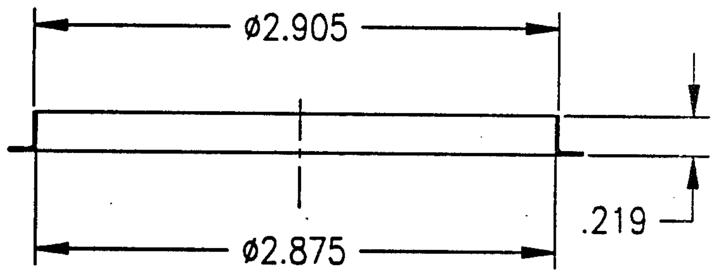


B



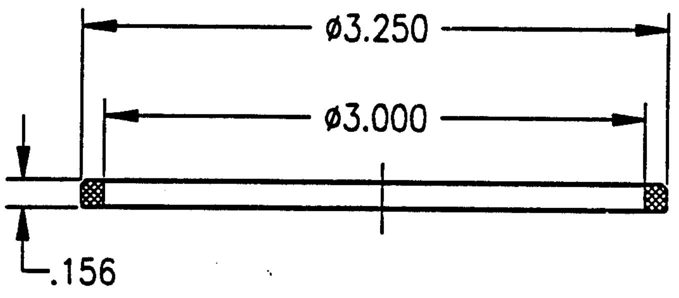
FLANGE CONTROL GRID INSULATOR 372441
KOVAR

C



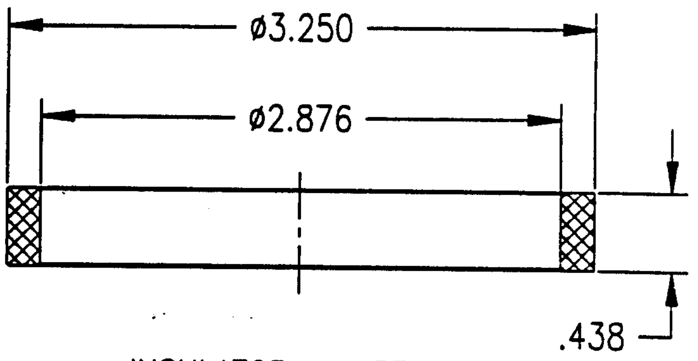
FLANGE UPPER GRID SUPPORT 372440
KOVAR

D



BACK-UP CERAMIC 372439
AL-300

E



INSULATOR 372437
AL-300

Figure 3.3-8
NODE 8

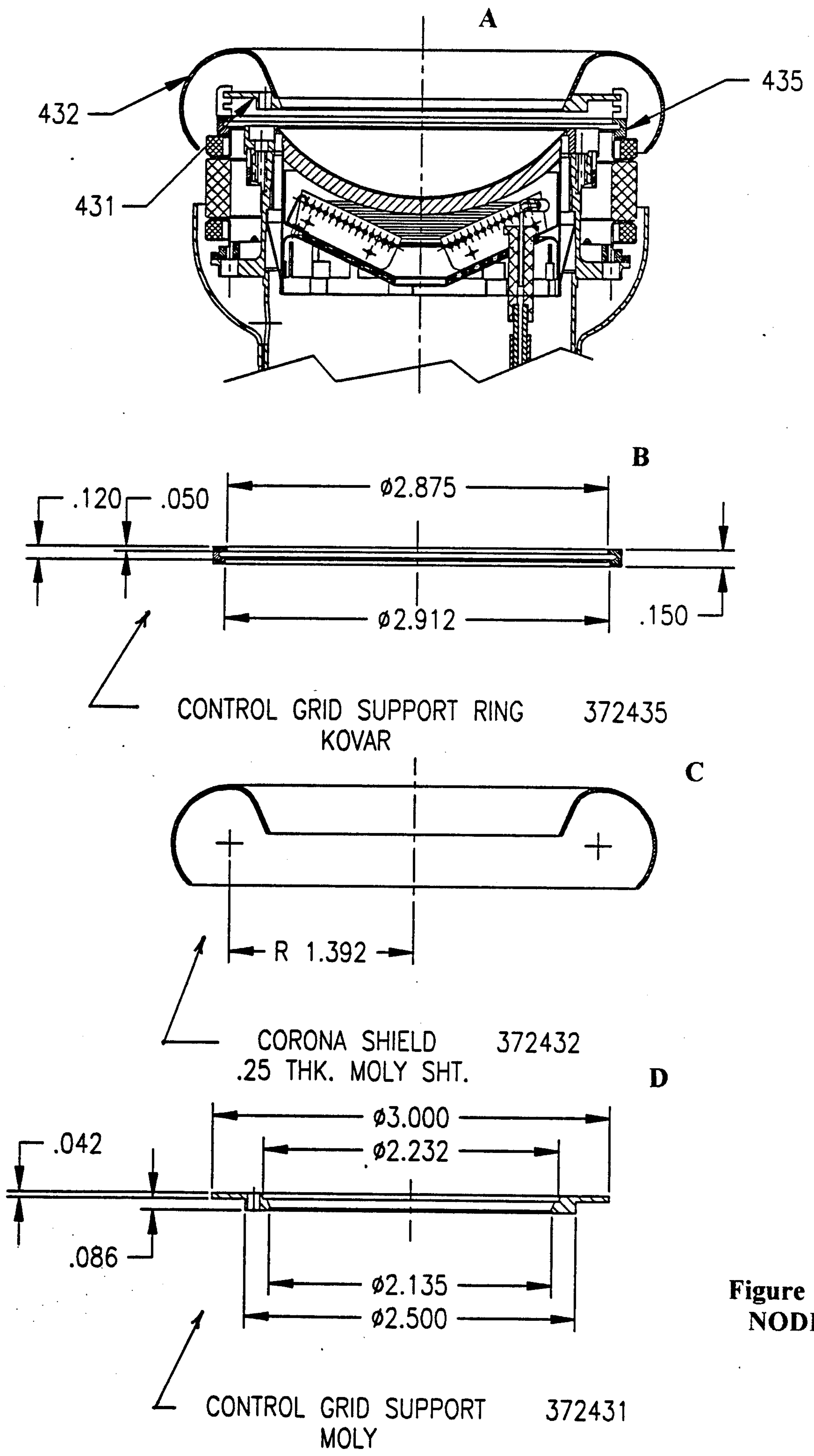


Figure 3.3-9
NODE 9

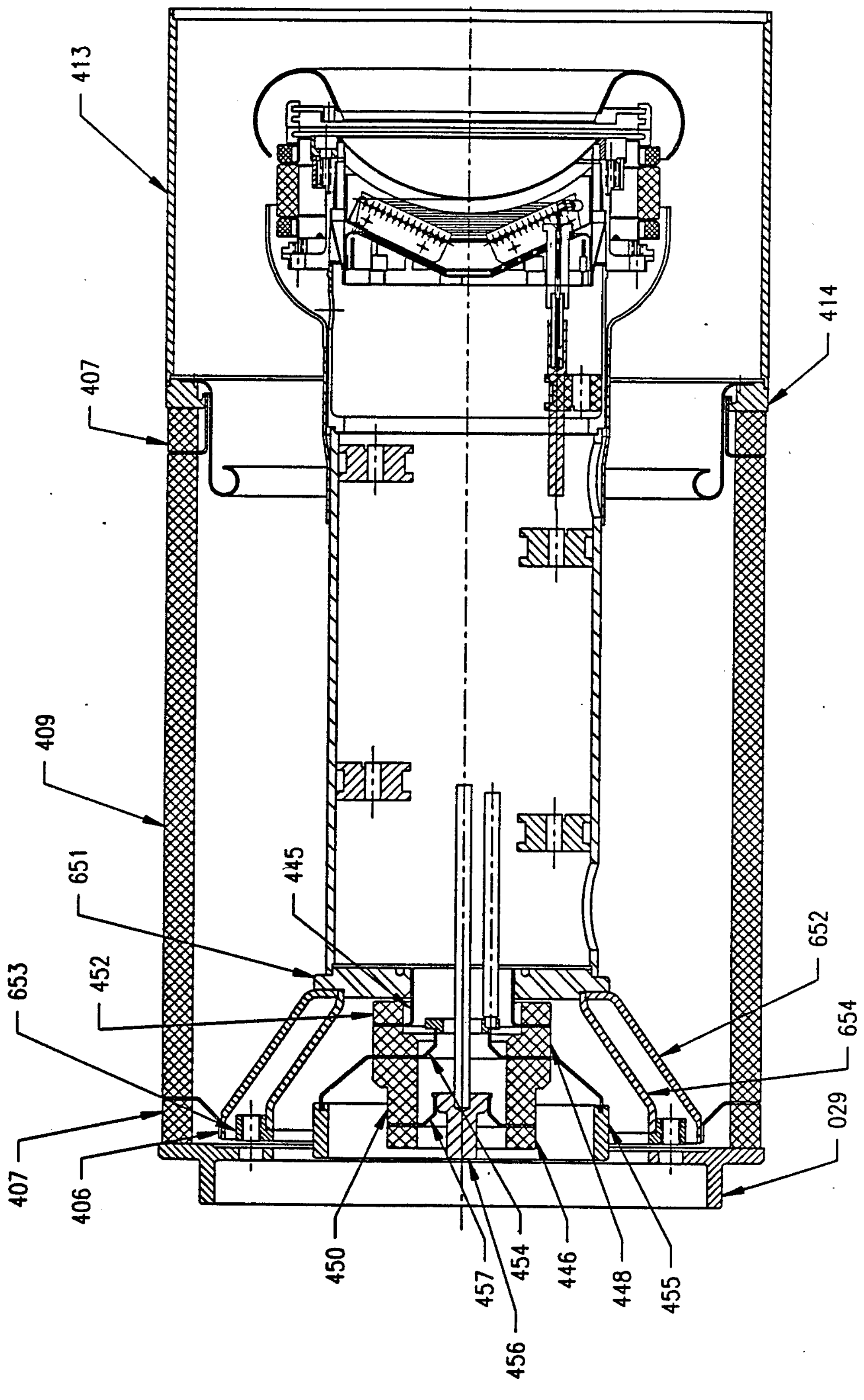


Figure 3.3-10
NODE 10

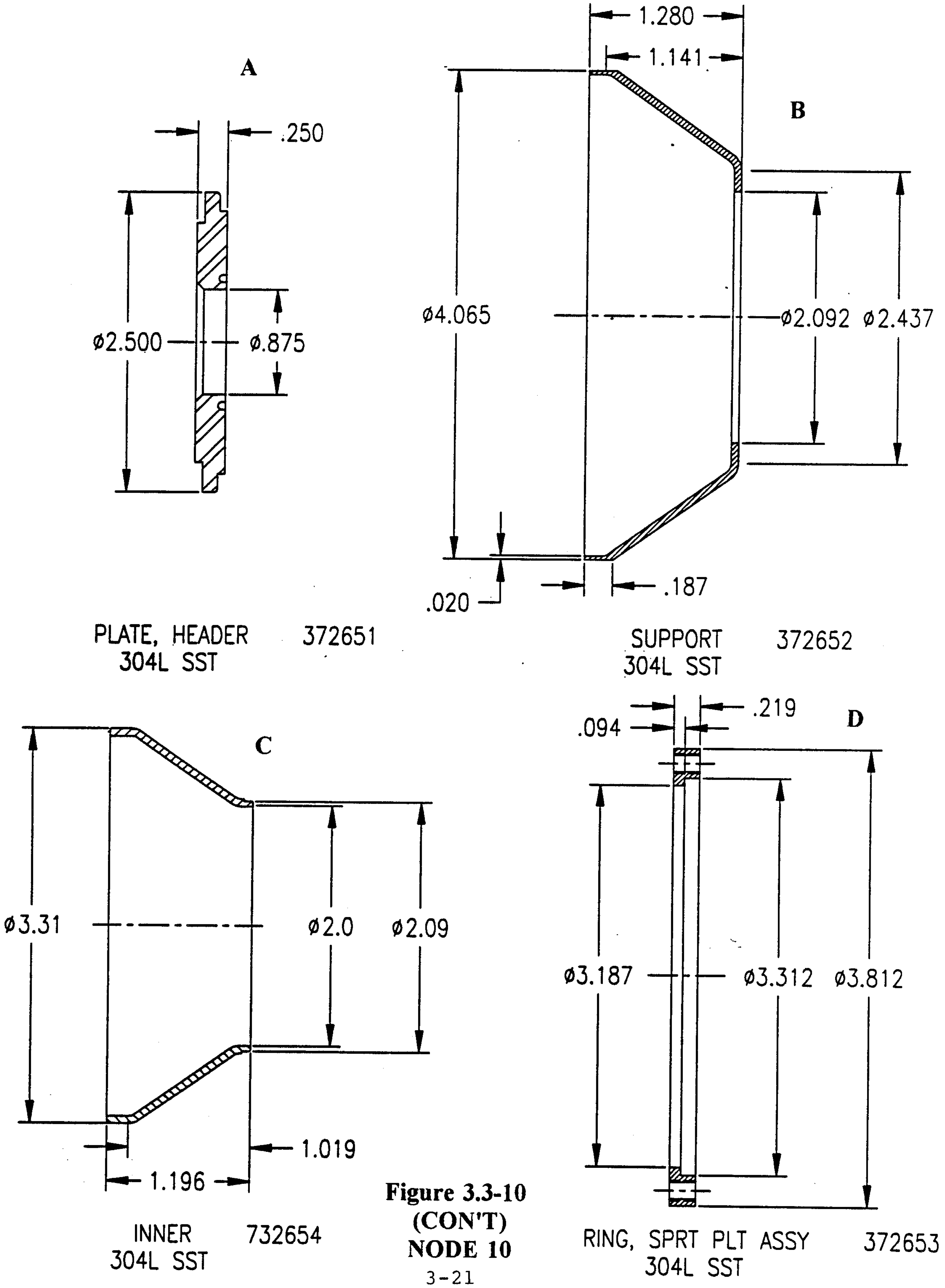
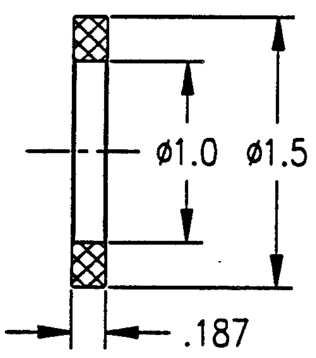


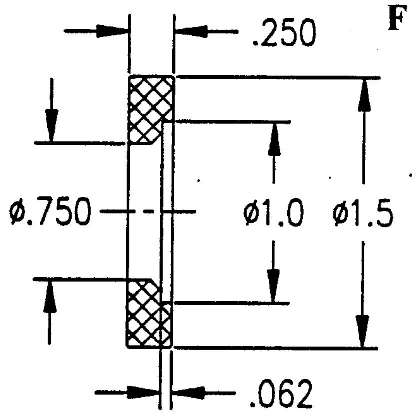
Figure 3.3-10
(CON'T)
NODE 10

E



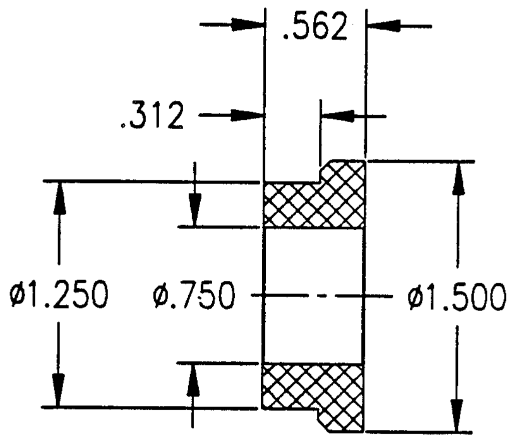
BACK UP INSULATOR
AL 300 372452

F



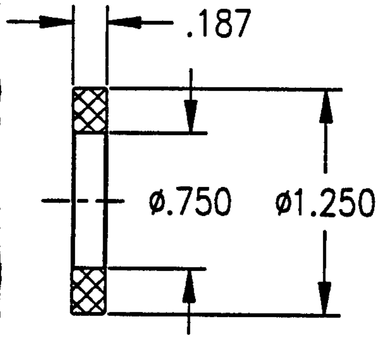
INSULATOR
AL-300 372448

G



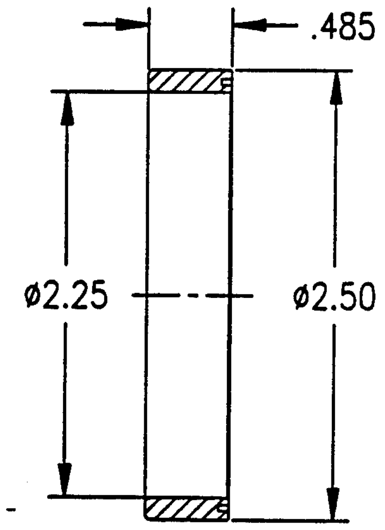
INSULATOR
AL-300 372450

H



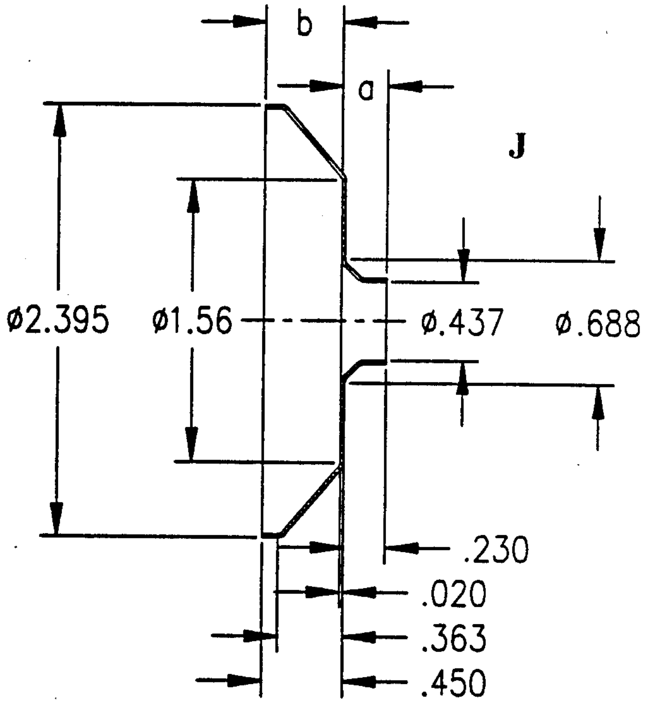
INSULATOR
AL-300 372446

I



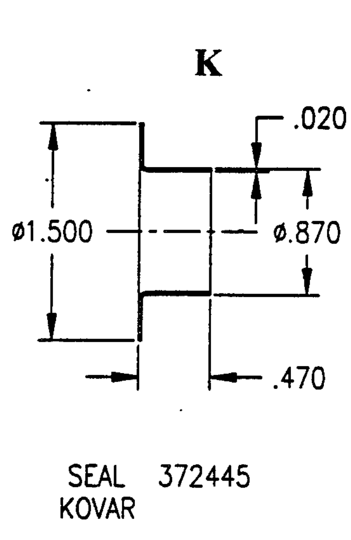
HEATER TERMINAL
304 SST 372455

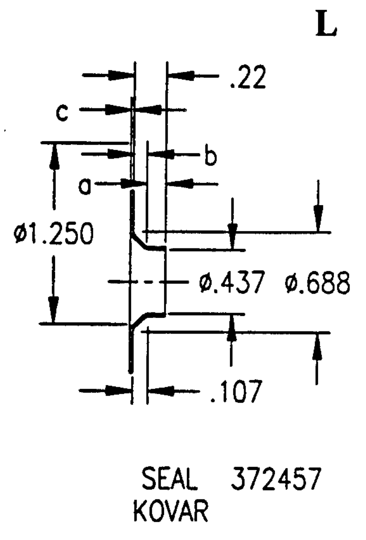
J

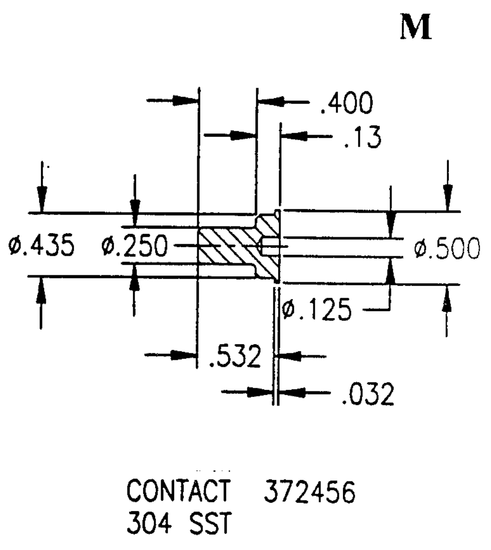


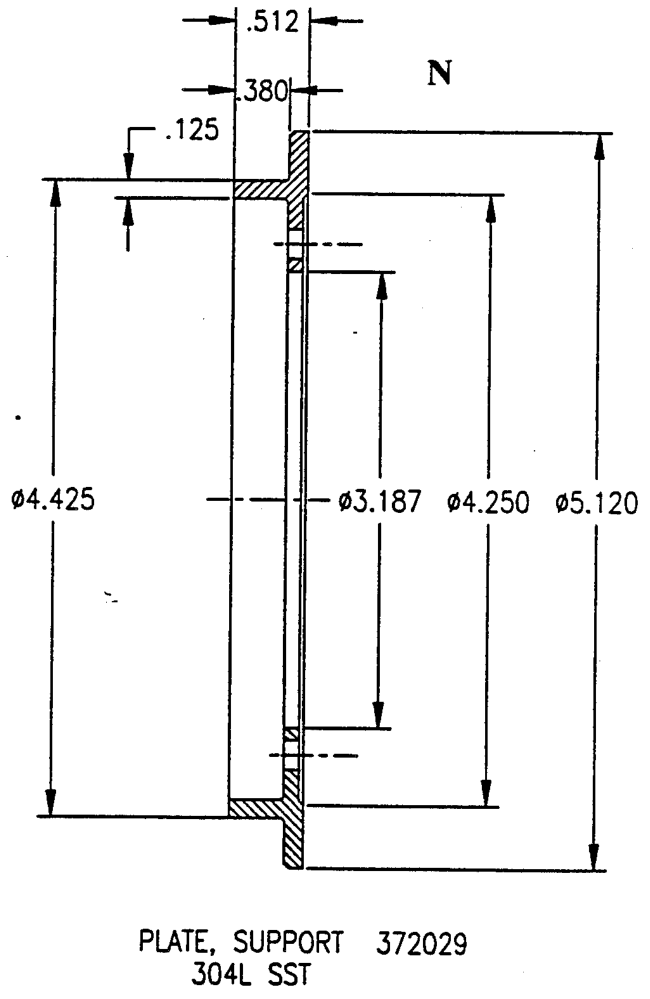
SEAL
KOVAR 372454

Figure 3.3-10 (CON'T)
NODE 10









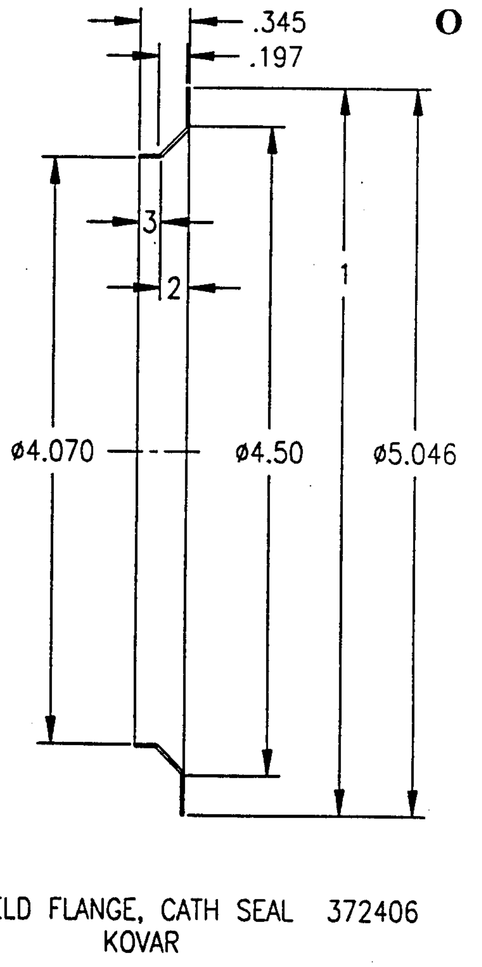
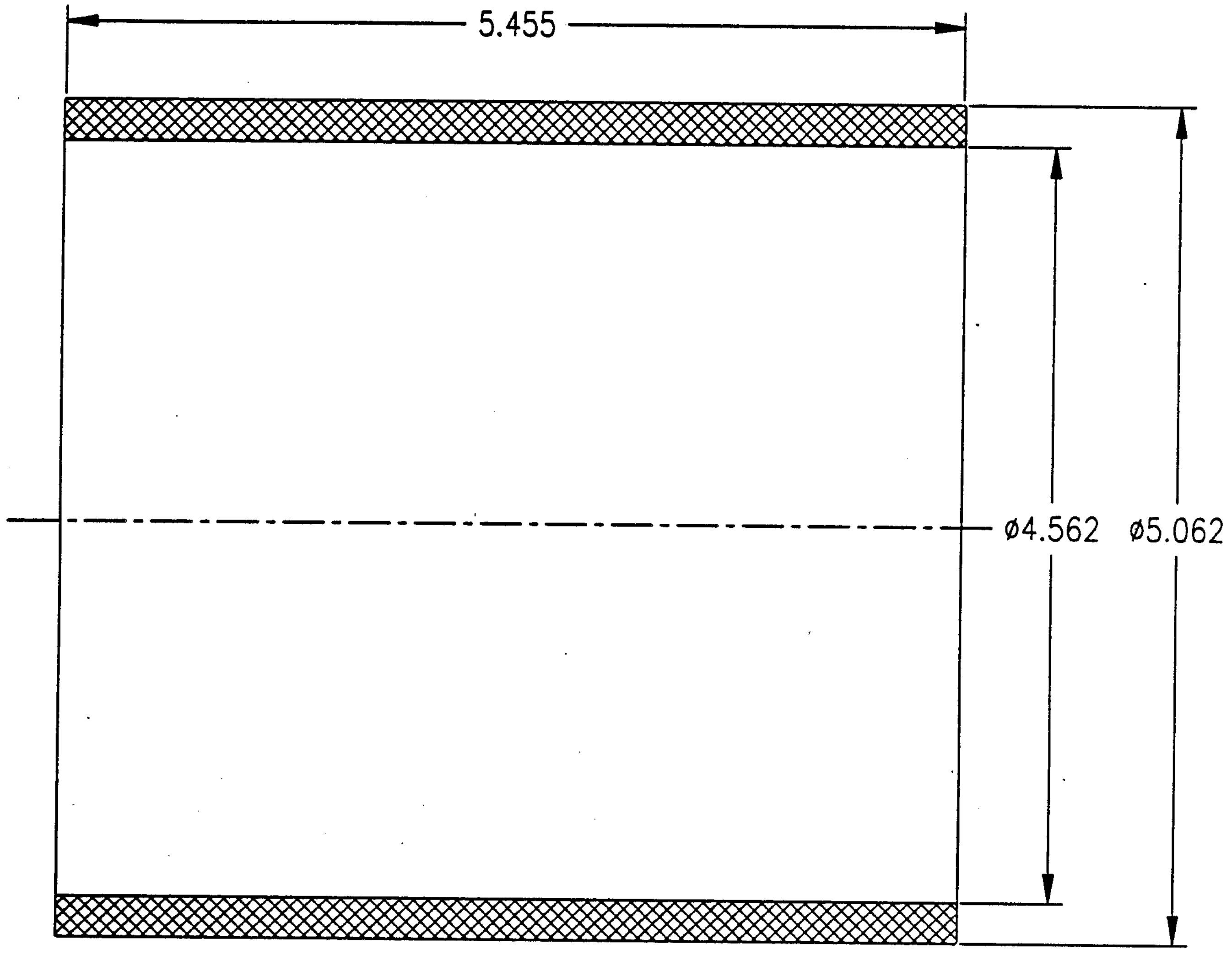


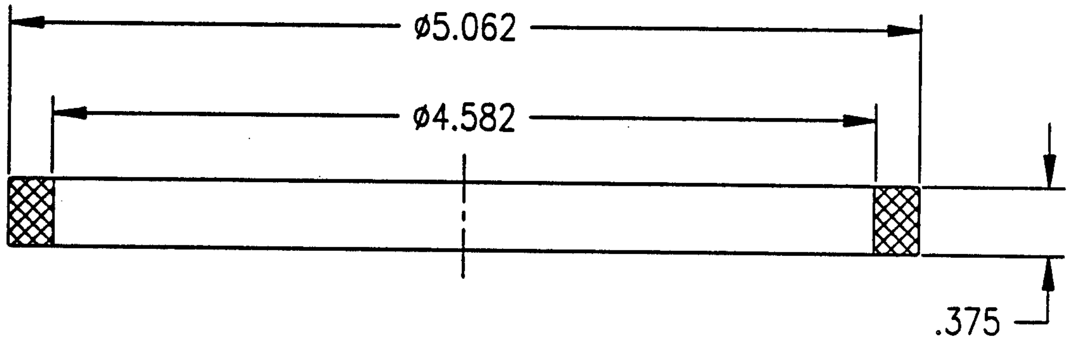
Figure 3.3-10 (CON'T)
NODE 10

P



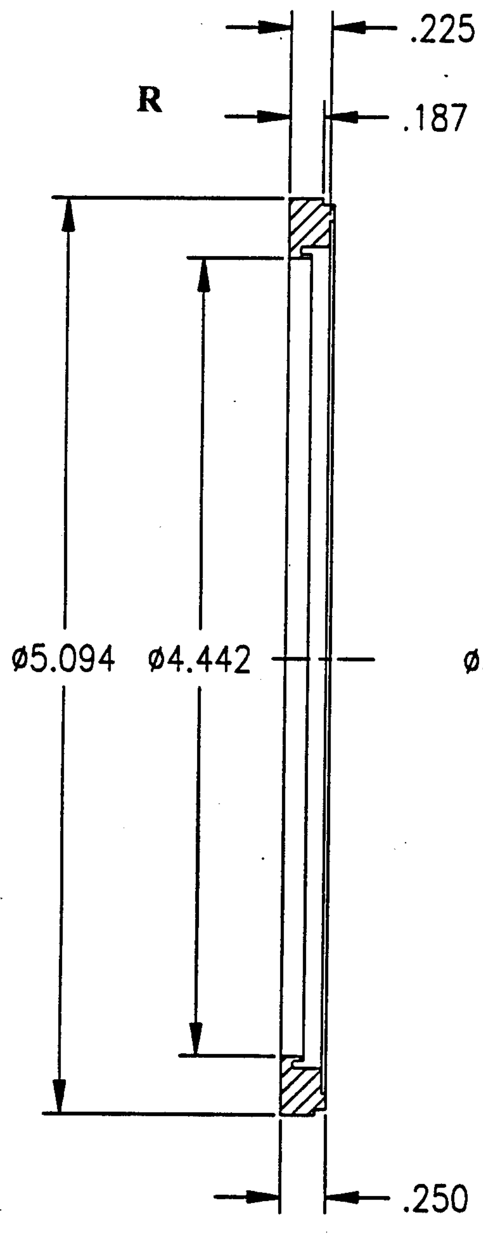
INSULATOR, CATHODE 372409
AL-300

Q

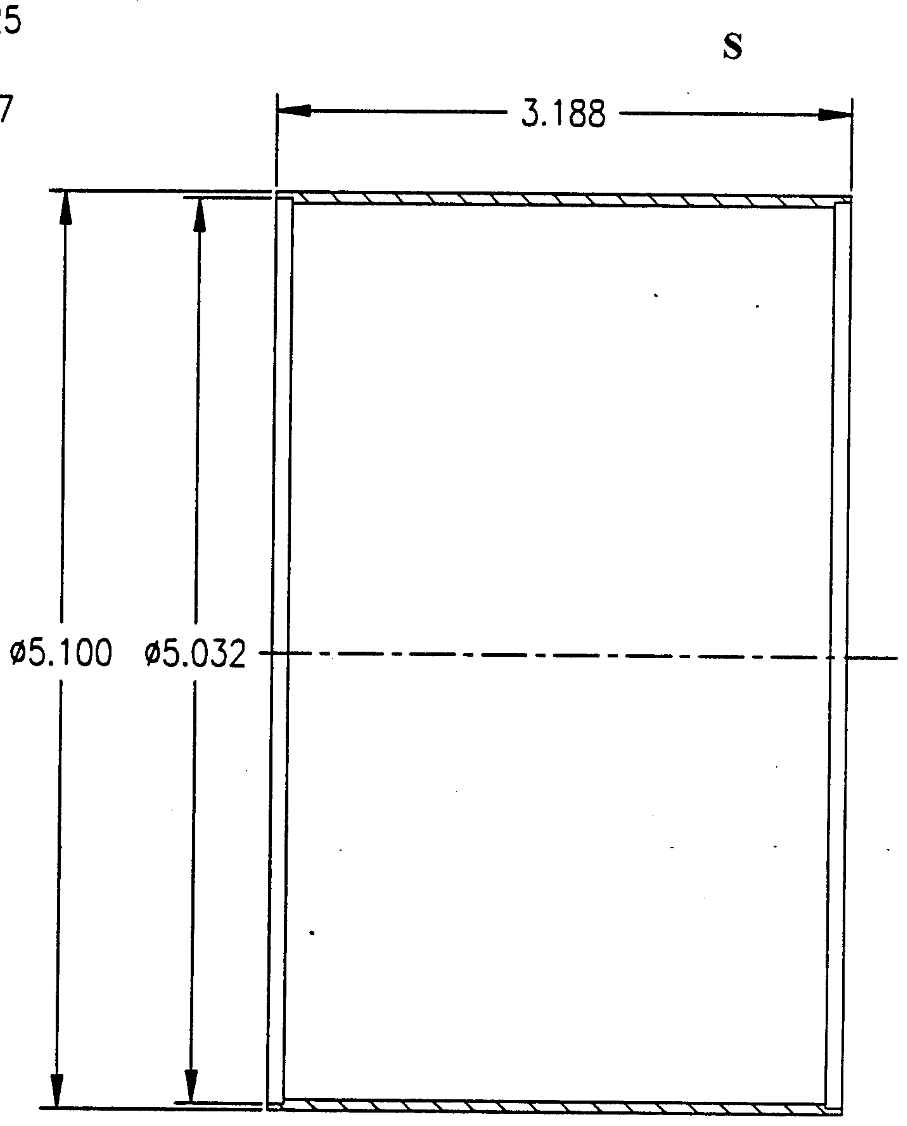


BACK UP CERAMIC 372407
AL-300

Figure 3.3-10 (CON'T)
NODE 10



BOTTOM RING, SEALING ASSY
304L SST 372414



SEALING CYLINDER
304 SST 372413

Figure 3.3-10 (CON'T)
NODE 10

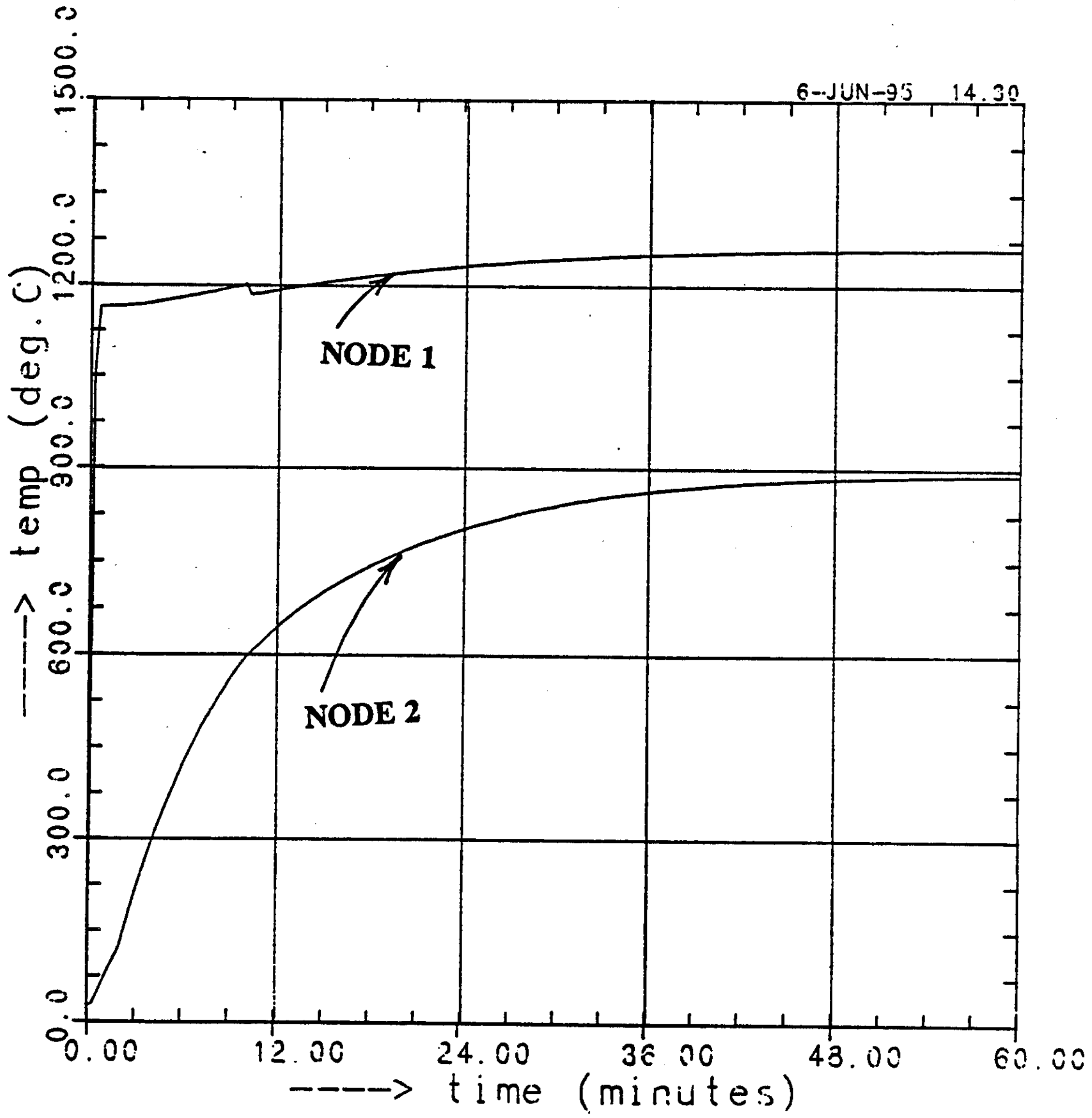


Figure 3.4-1
TEMPERATURE VS. TIME

APPENDIX 3A: Programming Steps

STEP 1:

Enter id, od, length, material type, node number for each part.
Enter specific heat and density for each material type.
Enter initial temperature in degree C.

STEP 2:

Calculate thermal capacitance of each part using following equation:

$$C_t = mc = A*L*\rho*c$$

Here
m = mass (kg)
A = cross-sectional area (inch**2)
L = length (inch)
ρ = density (kg/inch**3)
c = specific heat (Joules/kg/deg.C)

STEP 3:

Compute thermal capacitance of each node by adding thermal capacitances of all parts belonging to that node.

STEP 4:

Calculate the thermal resistance of each part of the gun for the starting temperature using following equation:

$$R_t = L/(k(T)*A)$$

Here k(T) represents the thermal conductivity at temperature T (watts/inch/deg.C).

STEP 5:

Compute thermal time constant of each part of the gun assembly using:

$$\tau_t = C_t R_t$$

STEP 6:

Loop through each time step:

-- Compute time in sec.
-- Enter heater voltage at that time (Volts).
-- Compute electrical resistance of heater element using following formula:

$$R_{heater} = \rho(T)L/A$$

Here $\rho(T)$ is resistivity of heater element at temperature T (ohm-inch).
L is the length of heater coil (inch), A is the cross-sectional area of the heater coil (inch**2).

-- Compute heater power from:

$$P = V*V/R_{heater}$$

-- Calculate the thermal resistance of each part of the gun for the starting temperature using following equation:

$$R_t = L/(k(T)*A)$$

Here k(T) represents the thermal conductivity at temperature T (watts/inch/deg.C).

-- Compute thermal resistances between nodes by adding with a proper weighting factor the thermal resistances of parts located adjacent to the boundary between two nodes.

-- Compute net power flow to a node from all other nodes.

-- Compute the updated temperature of each node at the next time step.

-- Repeat all the above steps (under step 6) for each incremental time step until steady state is reached.

APPENDIX 3B: Input Data File

```
$heatcoil
id(1,1)=0.0,od(1,1)=0.02875,length(1,1)=54.47,MatType(1,1)=1 $end
$part1 jpart(1)=1 $end

$heater_tab
id(2,1)=0.0,od(2,1)=0.02875,length(2,1)=0.65,MatType(2,1)=1 $end
$heat_shield_487
id(2,2)=0.0,od(2,2)=2.085,length(2,2)=0.118,MatType(2,2)=2  $end
$heat_shield_484
id(2,3)=0.0,od(2,3)=2.075,length(2,3)=0.4,MatType(2,3)=2  $end
$heat_shield_488
id(2,4)=0.0,od(2,4)=1.935,length(2,4)=0.028,MatType(2,4)=2  $end
$Cathode
id(2,5)=0.0,od(2,5)=2.2,length(2,5)=0.118,MatType(2,5)=2  $end
$Cath_support_473
id(2,6)=2.089,od(2,6)=2.105,length(2,6)=1.105,MatType(2,6)=3  $end
$part2 jpart(2)=6 $end

$Cath_support_466
id(3,1)=2.2,od(3,1)=2.22,length(3,1)=0.65,MatType(3,1)=3  $end
$ss_469_a
id(3,2)=2.253,od(3,2)=2.555,length(3,2)=0.225,MatType(3,2)=4  $end
$ss_469_b
id(3,3)= 2.283,od(3,3)=2.405,length(3,3)=0.208,MatType(3,3)=4  $end
$ss_469_c
id(3,4)=2.344,od(3,4)=2.405,length(3,4)=0.267,MatType(3,4)=4  $end
$heat_trap_649
id(3,5)=2.231,od(3,5)=2.251,length(3,5)=0.406,MatType(3,5)=2  $end
$Sgrid_support_425
id(3,6)=2.173,od(3,6)=2.675,length(3,6)=0.211,MatType(3,6)=2  $end
$Sgrid_support_426
id(3,7)=2.579,od(3,7)=2.639,length(3,7)=0.27,MatType(3,7)=5  $end
$part3 jpart(3)=7 $end

$ss_469_d
id(4,1)=2.344,od(4,1)=3.04,length(4,1)=0.234,MatType(4,1)=4  $end
$kovar_442
id(4,2)=2.735,od(4,2)=3.16,length(4,2)=0.15,MatType(4,2)=5  $end
$part4 jpart(4)=2 $end

$ss_470
id(5,1)= 2.288,od(5,1)=2.348,length(5,1)=1.43,MatType(5,1)=3  $end
```

```
$part5 jpart(5)=1 $end

$sleeve_496
id(6,1)=2.16,od(6,1)=2.3,length(6,1)=4.668,MatType(6,1)=4  $end
$part6 jpart(6)=1 $end

$cylinder_658
id(7,1)=2.295,od(7,1)=2.343,length(7,1)=1.5,MatType(7,1)=4  $end
$shield_657
id(7,2)=2.818,od(7,2)=2.878,length(7,2)=1.195,MatType(7,2)=4  $end
$part7 jpart(7)=2 $end

$kovar_441
id(8,1)=2.875,od(8,1)=2.905,length(8,1)=0.219,MatType(8,1)=5  $end
$kovar_440
id(8,2)=2.875,od(8,2)=2.905,length(8,2)=0.219,MatType(8,2)=5  $end
$ceramic_439_a
id(8,3)=3.0,od(8,3)=3.25,length(8,3)=0.156,MatType(8,3)=6  $end
$ceramic_437
id(8,4)=2.876,od(8,4)=3.25,length(8,4)=0.438,MatType(8,4)=6  $end
$ceramic_439_b
id(8,5)=3.0,od(8,5)=3.25,length(8,5)=0.156,MatType(8,5)=6  $end
$part8 jpart(8)=5 $end

$kovar_435
id(9,1)=2.875,od(9,1)=2.905,length(9,1)=0.15,MatType(9,1)=5  $end
$corona_432
id(9,2)=1.342,od(9,2)=1.392,length(9,2)=2.039,MatType(9,2)=2  $end
$cgrid_support_a_431
id(9,3)= 2.2320,od(9,3)=3.0,length(9,3)=0.042,MatType(9,3)=2  $end
$cgrid_support_b_431
id(9,4)=2.135,od(9,4)=2.5,length(9,4)=0.086,MatType(9,4)=2  $end
$part9 jpart(9)=4 $end

$plate_651
id(10,1)=0.875,od(10,1)=2.5,length(10,1)=0.25,MatType(10,1)=4  $end
$support_652_a
id(10,2)=4.0,od(10,2)=4.065,length(10,2)=0.187,MatType(10,2)=3  $end
$support_652_b
id(10,3)=3.059,od(10,3)=3.098,length(10,3)=1.093,MatType(10,3)=4  $end
$support_654
id(10,4)=3.059,od(10,4)=3.098,length(10,4)=1.093,MatType(10,4)=4  $end
$ring_653
id(10,5)=3.25,od(10,5)=3.812,length(10,5)=0.219,MatType(10,5)=4  $end
```

```
$insulator_452
id(10,6)=1.0,od(10,6)=1.5,length(10,6)=0.187,MatType(10,6)=6  $end
$insulator_448
id(10,7)=0.8,od(10,7)=1.5,length(10,7)=0.25,MatType(10,7)=6  $end
$insulator_450
id(10,8)=0.75,od(10,8)=1.375,length(10,8)=0.562,MatType(10,8)=6  $end
$insulator_446
id(10,9)=0.75,od(10,9)=1.25,length(10,9)=0.187,MatType(10,9)=6  $end
$heater_terminal_455
id(10,10)=2.25,od(10,10)=2.5,length(10,10)=0.485,MatType(10,10)=4  $end
$seal_454_a
id(10,11)=0.543,od(10,11)=0.583,length(10,11)=0.25,MatType(10,11)=5  $end
$seal_454_b
id(10,12)=1.502,od(10,12)=1.541,length(10,12)=0.45,MatType(10,12)=5  $end
$seal_445
id(10,13)=1.145,od(10,13)=1.185,length(10,13)=0.47,MatType(10,13)=5  $end
$seal_457_a
id(10,14)=0.437,od(10,14)=0.477,length(10,14)=0.117,MatType(10,14)=5  $end
$seal_457_b
id(10,15)=0.543,od(10,15)=0.583,length(10,15)=0.107,MatType(10,15)=5  $end
$seal_457_c
id(10,16)=0.688,od(10,16)=1.25,length(10,16)=0.020,MatType(10,16)=5  $end
$contact_456_a
id(10,17)=0.125,od(10,17)=0.468,length(10,17)=0.13,MatType(10,17)=4  $end
$contact_456_b
id(10,18)=0.0,od(10,18)=0.435,length(10,18)=0.034,MatType(10,18)=4  $end
$contact_456_c
id(10,19)=0.0,od(10,19)=0.250,length(10,19)=0.4,MatType(10,19)=4  $end
$support_029_1
id(10,20)=3.187,od(10,20)=5.12,length(10,20)=0.132,MatType(10,20)=4  $end
$seal_406_1
id(10,21)=4.5,od(10,21)=5.046,length(10,21)=0.02,MatType(10,21)=5  $end
$seal_406_2
id(10,22)=4.265,od(10,22)=4.305,length(10,22)=0.197,MatType(10,22)=5  $end
$seal_406_3
id(10,23)=4.07,od(10,23)=4.11,length(10,23)=0.128,MatType(10,23)=5  $end
$ceramic_407
id(10,24)=4.582,od(10,24)=5.062,length(10,24)=0.375,MatType(10,24)=6  $end
$ring_409
id(10,25)=4.562,od(10,25)=5.062,length(10,25)=5.455,MatType(10,25)=4  $end
$ring_414
id(10,26)=4.442,od(10,26)=5.094,length(10,26)=0.22,MatType(10,26)=4  $end
$cylinder_413
id(10,27)=5.032,od(10,27)=5.1,length(10,27)=3.188,MatType(10,27)=4  $end
```

```
$part10 jpart(10)=27 $end
```

APPENDIX 3C: Source Code for CATHTEMP.FOR

```
      PROGRAM CATHTEMP
C****************************************************************
C File Name: CATHTEMP.FOR
C Descrpt. : This program calculates the temperature of a cathode
c           of a gun vs heater voltage where the different dimensions
c           of different parts of the gun are to be entered as input.
c
c Unit     : MKS except lengths; lengths are in inches.
C
C Input    :
c            File: "cathin.dat"
c                  contains dimensions of different parts of the
c                  gun read by the program.
c            ID = inner dia. of a part (inch)
c            OD = outer dia. of a part (inch)
C             length  = length of a part (inch)
c            spheat = specific heat of a material (Joules/kg/C)
c            dens = density of a material (kg/inch**3)
c            k = thermal conductivity (watts/inch/C)
c            Tamb = ambient temperature in deg.K
c            Temp(i) = temp of i-th node in degree Kelvin.
c            time0 = time in minutes over which heater voltage is
c                    "Vh0" volts
c            jtrans = total # of time steps for transient calculations
c            jtmax = total # of time steps
c            h0 = duration of a time step during transient calculation
c                                                             (sec)
c            h1 = duration of a time step after transient calculation
c                 and until steady state is reached.
c            ampmax = maximum allowable heater current (Amps).
C
c Intermediate
c variables:  mc(i,j) = thermal capacitance of j-th part in
c                       i-th node  (Joules/deg.C)
c            mcn(i)  = thermal capacitance of i-th node
c            Rt(i,j) = thermal resistance of j-th part in
c                       i-th node (deg.C/watt)
c            Rn(i,i+1) = thermal resistance between nodes i and i+1.
c            Pout(i,i+1) = net power flow from node i to node i+1
c            PoutT(i) = net power out from i-th node
c            Pin(i)   = net power in to i-th node
c
```

```
C Output   :  Files = "input.dat" contains all input data written by
c                             the program.
c                    "cathout.dat" contains intermediate results from
c                               the program e.g. disk areas, cylindrical
c                               areas, thermal resistances of different
c                               parts.
c                    "node.dat" contains thermal capacitance of each node.
c                    "temp.dat" contains cathode temperature vs time
c                             and heater temperature vs time.
c               Temp(jtime,inode) = temp. of node  index 'inode'
c                             at time step 'jtime'.
c
C
C Rev. History: Oct. 26, 1993 (SRB)
c               Nov. 16, 1993   Calculation of thermal time constants
c                            for each part added.
c               Dec 29, 1993    Stepping thru time increment added
c                            to update the nodal temperature (do 56)
C*************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      real*8 id(10,50),length(10,50),mc(10,50),mcn(10),L,k
      dimension od(10,50), acyl(10,50), adisk(10,50), MatType(10,50),
     + dens(6),spheat(6),jpart(10),Rt(10,50), Time_const(10,50),
     + Temp(1800,10),Rn(10,10),Pout(10,10),Pin(10),PoutT(10)
c
c Constants:
      pi = 3.141592654
      c = 3.0e08
c     Stefan-Boltzmann constant (watts/inch*2/K**4)
      sigma = 3.658e-11
c Input:
      write(5,*)'Dimensions: length = inch, sp.ht. = Joules/kg/C'
      write(5,*)'density = kg/inch**3, k= watts/inch/C'
c       write(5,*)'Enter total # of nodes'
c       read(5,*)inode
      inode=10
      write(5,*)'enter 1 if diagnostic checks are desired else enter 0'
      read(5,*)idiag
      Tamb = 26.8 + 273.0
      do 10 iT = 1, inode
         Temp(1,iT) = Tamb
10     continue
c     Enter voltage and time steps:
      time0 = 10.0
      Vh0 = 14.0
      Vh1 = 13.0
      jtrans = 60
      jtmax = 1800
      h0 = 1.0
      h1 = 2.0
      ampmax = 15.0
c
      namelist/heatcoil/id,od,length,MatType
      namelist/part1/jpart
      namelist/heater_tab/id,od,length,MatType
      namelist/heat_shield_487/id,od,length,MatType
      namelist/heat_shield_484/id,od,length,MatType
      namelist/heat_shield_488/id,od,length,MatType
      namelist/Cathode/id,od,length,MatType
      namelist/Cath_support_473/id,od,length,MatType
      namelist/part2/jpart
      namelist/Cath_support_466/id,od,length,MatType
      namelist/ss_469_a/id,od,length,MatType
      namelist/ss_469_b/id,od,length,MatType
      namelist/ss_469_c/id,od,length,MatType
      namelist/heat_trap_649/id,od,length,MatType
      namelist/Sgrid_support_425/id,od,length,MatType
      namelist/Sgrid_support_426/id,od,length,MatType
      namelist/part3/jpart
      namelist/ss_469_d/id,od,length,MatType
      namelist/kovar_442/id,od,length,MatType
      namelist/part4/jpart
      namelist/ss_470/id,od,length,MatType
      namelist/part5/jpart
      namelist/sleeve_496/id,od,length,MatType
      namelist/part6/jpart
      namelist/cylinder_658/id,od,length,MatType
      namelist/shield_657/id,od,length,MatType
      namelist/part7/jpart
      namelist/kovar_441/id,od,length,MatType
      namelist/kovar_440/id,od,length,MatType
      namelist/ceramic_439_a/id,od,length,MatType
      namelist/ceramic_437/id,od,length,MatType
      namelist/ceramic_439_b/id,od,length,MatType
      namelist/part8/jpart
      namelist/kovar_435/id,od,length,MatType
      namelist/corona_432/id,od,length,MatType
      namelist/cgrid_support_a_431/id,od,length,MatType
      namelist/cgrid_support_b_431/id,od,length,MatType
      namelist/part9/jpart
      namelist/plate_651/id,od,length,MatType
      namelist/support_652_a/id,od,length,MatType
      namelist/support_652_b/id,od,length,MatType
      namelist/support_654/id,od,length,MatType
      namelist/ring_653/id,od,length,MatType
      namelist/insulator_452/id,od,length,MatType
      namelist/insulator_448/id,od,length,MatType
      namelist/insulator_450/id,od,length,MatType
      namelist/insulator_446/id,od,length,MatType
      namelist/heater_terminal_455/id,od,length,MatType
      namelist/seal_454_a/id,od,length,MatType
      namelist/seal_454_b/id,od,length,MatType
      namelist/seal_445/id,od,length,MatType
      namelist/seal_457_a/id,od,length,MatType
      namelist/seal_457_b/id,od,length,MatType
      namelist/seal_457_c/id,od,length,MatType
      namelist/contact_456_a/id,od,length,MatType
      namelist/contact_456_b/id,od,length,MatType
      namelist/contact_456_c/id,od,length,MatType
      namelist/support_029_1/id,od,length,MatType
      namelist/seal_406_1/id,od,length,MatType
      namelist/seal_406_2/id,od,length,MatType
      namelist/seal_406_3/id,od,length,MatType
      namelist/ceramic_407/id,od,length,MatType
      namelist/ring_409/id,od,length,MatType
      namelist/ring_414/id,od,length,MatType
      namelist/cylinder_413/id,od,length,MatType
      namelist/part10/jpart
c
c
      open (unit=15,file='cathin.dat',status='old')
      open (unit=16,file='input.dat',status='new')
      open (unit=17,file='cathout.dat',status='new')
      open (unit=18,file='node.dat',status='new')
      open (unit=19,file='check.dat',status='new')
      open (unit=20,file='temp.dat',status='new')
c Read data:
      read(15,heatcoil)
      read(15,part1)
      read(15,heater_tab)
      read(15,heat_shield_487)
read(15,heat_shield_484)
read(15,heat_shield_488)
read(15,Cathode)
read(15,Cath_support_473)
read(15,part2)
read(15,Cath_support_466)
read(15,ss_469_a)
read(15,ss_469_b)
read(15,ss_469_c)
read(15,heat_trap_649)
read(15,Sgrid_support_425)
read(15,Sgrid_support_426)
read(15,part3)
read(15,ss_469_d)
read(15,kovar_442)
read(15,part4)
read(15,ss_470)
read(15,part5)
read(15,sleeve_496)
read(15,part6)
read(15, cylinder_658)
read(15,shield_657)
read(15,part7)
read(15,kovar_441)
read(15,kovar_440)
read(15,ceramic_439_a)
read(15,ceramic_437)
read(15,ceramic_439_b)
read(15,part8)
read(15,kovar_435)
read(15,corona_432)
read(15,cgrid_support_a_431)
read(15,cgrid_support_b_431)
read(15,part9)
read(15,plate_651)
read(15,support_652_a)
read(15,support_652_b)
read(15,support_654)
read(15,ring_653)
read(15,insulator_452)
read(15,insulator_448)
read(15,insulator_450)
read(15,insulator_446)
read(15,heater_terminal_455)
```

```
      read(15,seal_454_a)
      read(15,seal_454_b)
      read(15,seal_445)
      read(15,seal_457_a)
      read(15,seal_457_b)
      read(15,seal_457_c)
      read(15,contact_456_a)
      read(15,contact_456_b)
      read(15,contact_456_c)
      read(15,support_029_1)
      read(15,seal_406_1)
      read(15,seal_406_2)
      read(15,seal_406_3)
      read(15,ceramic_407)
      read(15,ring_409)
      read(15,ring_414)
      read(15,cylinder_413)
      read(15,part10)
c
c     Enter density and specific heat for different material:
c     (array dimension indicates 'MatType')
c     MatType=1 => tungsten
c     MatType=2 => moly
c     MatType=3 => Nickel
c     MatType=4 => stainless steel
c     MatType=5 => kovar
c     MatType=6 => alumina
c
      dens(1)= 316.27e-03
      dens(2)= 167.47e-03
      dens(3)= 145.98e-03
      dens(4)= 129.95e-03
      dens(5)= 117.0e-03
      dens(6)= 64.9e-03
      spheat(1) = 142.8
      spheat(2) = 253.96
      spheat(3) = 600.0
      spheat(4) = 490.0
      spheat(5) = 450.0
      spheat(6) = 1273.08
c Write input data to a file called 'input.dat':
      write(16,*)'group-name','    id ','     od','       length',
     + '     Material'
      write(16,*)'-----------------------------------------------'
write(16,*)'heatcoil'
write(16,166)id(1,1),od(1,1),length(1,1),MatType(1,1)
write(16,*)'heater_tab'
write(16,166)id(2,1),od(2,1),length(2,1),MatType(2,1)
write(16,*)'heat_shield_487'
write(16,166)id(2,2),od(2,2),length(2,2),MatType(2,2)
write(16,*)'heat_shield_484'
write(16,166)id(2,3),od(2,3),length(2,3),MatType(2,3)
write(16,*)'heat_shield-488'
write(16,166)id(2,4),od(2,4),length(2,4),MatType(2,4)
write(16,*)'cathode'
write(16,166)id(2,5),od(2,5),length(2,5),MatType(2,5)
write(16,*)'cath_support_473'
write(16,166)id(2,6),od(2,6),length(2,6),MatType(2,6)
write(16,*)'cath_support_466'
write(16,166)id(3,1),od(3,1),length(3,1),MatType(3,1)
write(16,*)'ss_469_a'
write(16,166)id(3,2),od(3,2),length(3,2),MatType(3,2)
write(16,*)'ss_469_b'
write(16,166)id(3,3),od(3,3),length(3,3),MatType(3,3)
write(16,*)'ss_469_c'
write(16,166)id(3,4),od(3,4),length(3,4),MatType(3,4)
write(16,*)'heat_trap_649'
write(16,166)id(3,5),od(3,5),length(3,5),MatType(3,5)
write(16,*)'sgrid_support_425'
write(16,166)id(3,6),od(3,6),length(3,6),MatType(3,6)
write(16,*)'sgrid_support_426'
write(16,166)id(3,7),od(3,7),length(3,7),MatType(3,7)
write(16,166)id(4,1),od(4,1),length(4,1),MatType(4,1)
write(16,*)'kovar_442'
write(16,166)id(4,2),od(4,2),length(4,2),MatType(4,2)
write(16,*)'ss_470'
write(16,166)id(5,1),od(5,1),length(5,1),MatType(5,1)
write(16,*)'sleeve_496'
write(16,166)id(6,1),od(6,1),length(6,1),MatType(6,1)
write(16,*)'cylinder_658'
write(16,166)id(7,1),od(7,1),length(7,1),MatType(7,1)
write(16,*)'shield_657'
write(16,166)id(7,2),od(7,2),length(7,2),MatType(7,2)
write(16,*)'kovar_441'
write(16,166)id(8,1),od(8,1),length(8,1),MatType(8,1)
write(16,*)'kovar_440'
write(16,166)id(8,2),od(8,2),length(8,2),MatType(8,2)
write(16,*)'ceramic_439_a'
write(16,166)id(8,3),od(8,3),length(8,3),MatType(8,3)
write(16,*)'ceramic_437'
write(16,166)id(8,4),od(8,4),length(8,4),MatType(8,4)
write(16,*)'ceramic_439_b'
write(16,166)id(8,5),od(8,5),length(8,5),MatType(8,5)
write(16,*)'kovar_435'
write(16,166)id(9,1),od(9,1),length(9,1),MatType(9,1)
write(16,*)'corona_432'
write(16,166)id(9,2),od(9,2),length(9,2),MatType(9,2)
write(16,*)'cgrid_support_a_431'
write(16,166)id(9,3),od(9,3),length(9,3),MatType(9,3)
write(16,*)'cgrid_support_b_431'
write(16,166)id(9,4),od(9,4),length(9,4),MatType(9,4)
write(16,*)'plate_651'
write(16,166)id(10,1),od(10,1),length(10,1),MatType(10,1)
write(16,*)'support_652_a'
write(16,166)id(10,2),od(10,2),length(10,2),MatType(10,2)
write(16,*)'support_652_b'
write(16,166)id(10,3),od(10,3),length(10,3),MatType(10,3)
write(16,*)'support_654'
write(16,166)id(10,4),od(10,4),length(10,4),MatType(10,4)
write(16,*)'ring_653'
write(16,166)id(10,5),od(10,5),length(10,5),MatType(10,5)
write(16,*)'insulator_452'
write(16,166)id(10,6),od(10,6),length(10,6),MatType(10,6)
write(16,*)'insulator_448'
write(16,166)id(10,7),od(10,7),length(10,7),MatType(10,7)
write(16,*)'insulator_450'
write(16,166)id(10,8),od(10,8),length(10,8),MatType(10,8)
write(16,*)'insulator_446'
write(16,166)id(10,9),od(10,9),length(10,9),MatType(10,9)
write(16,*)'heater_terminal_455'
write(16,166)id(10,10),od(10,10),length(10,10),MatType(10,10)
write(16,*)'seal_454_a'
write(16,166)id(10,11),od(10,11),length(10,11),MatType(10,11)
write(16,*)'seal_454_b'
write(16,166)id(10,12),od(10,12),length(10,12),MatType(10,12)
write(16,*)'seal_445'
write(16,166)id(10,13),od(10,13),length(10,13),MatType(10,13)
write(16,*)'seal_457_a'
write(16,166)id(10,14),od(10,14),length(10,14),MatType(10,14)
write(16,*)'seal_457_b'
write(16,166)id(10,15),od(10,15),length(10,15),MatType(10,15)
write(16,*)'seal_457_c'
```

```
      write(16,166)id(10,16),od(10,16),length(10,16),MatType(10,16)
      write(16,*)'seal_457_d'
      write(16,166)id(10,17),od(10,17),length(10,17),MatType(10,17)
      write(16,*)'contact_456_a'
      write(16,166)id(10,18),od(10,18),length(10,18),MatType(10,18)
      write(16,*)'contact_456_b'
      write(16,166)id(10,19),od(10,19),length(10,19),MatType(10,19)
      write(16,*)'contact_456_c'
      write(16,166)id(10,20),od(10,20),length(10,20),MatType(10,20)
      write(16,*)'supportt_029_1'
      write(16,166)id(10,21),od(10,21),length(10,21),MatType(10,21)
      write(16,*)'seal_406_1'
      write(16,166)id(10,22),od(10,22),length(10,22),MatType(10,22)
      write(16,*)'seal_406_2'
      write(16,166)id(10,23),od(10,23),length(10,23),MatType(10,23)
      write(16,*)'seal_406_3'
      write(16,166)id(10,24),od(10,24),length(10,24),MatType(10,24)
      write(16,*)'ceramic_407'
      write(16,166)id(10,25),od(10,25),length(10,25),MatType(10,25)
      write(16,*)'ring_409'
      write(16,166)id(10,26),od(10,26),length(10,26),MatType(10,26)
      write(16,*)'ring_414'
      write(16,166)id(10,28),od(10,28),length(10,28),MatType(10,28)
      write(16,*)'cylinder_413'
      write(16,166)id(10,29),od(10,29),length(10,29),MatType(10,29)
      write(16,*)'jpart(1) = ',jpart(1)
      write(16,*)'jpart(2) = ',jpart(2)
      write(16,*)'jpart(3) = ',jpart(3)
      write(16,*)'jpart(4) = ',jpart(4)
      write(16,*)'jpart(5) = ',jpart(5)
      write(16,*)'jpart(6) = ',jpart(6)
      write(16,*)'jpart(7) = ',jpart(7)
      write(16,*)'jpart(8) = ',jpart(8)
      write(16,*)'jpart(9) = ',jpart(9)
      write(16,*)'jpart(10) = ',jpart(10)
c
c     ----------------------------------------------------------
c     Compute cylindrical surface area, disk area and mass*sp.ht.
c     for each part:
c     Loop thru different nodes:
      do 55 i = 1, inode
       mcn(i) = 0.0
       write(17,*)'node = ',i
       write(17,*)'part# ','acyl(inch**2) ','  adisk(inch**2) ',
     +  '   mass*sp.ht.(Joules/C)','   Rtherm(C/watt) ',
     +  ' thermal time const'
C        Loop thru different parts in each node:
        do 55 j = 1,jpart(i)
          acyl(i,j) = pi*length(i,j)*od(i,j)
          adisk(i,j) = pi*(od(i,j)**2 - id(i,j)**2) /4.0
c         mass = density*volume
          mc(i,j) = dens(MatType(i,j))*length(i,j)*adisk(i,j)*
     +              spheat(MatType(i,j))
          itype = mattype(i,j)
          call thermres(mattype(i,j),Temp(1,i),length(i,j),
     +                                 Adisk(i,j),Rt(i,j))
          Time_const(i,j) = mc(i,j)*Rt(i,j)
          write(17,167)j,acyl(i,j),adisk(i,j),mc(i,j),Rt(i,j),
     +     Time_const(i,j)
c         Compute total mass*sp.ht. for each node:
          mcn(i) = mcn(i) + mc(i,j)
55      continue
c       -----------------------------------------------------
c       -----------------------------------------------------
c       Loop thru time steps:
        do 56 jtime = 1, jtmax
          write(19,*)'----------------------'
          write(19,*)'Time iteration # = ', jtime
          if (jtime.gt.jtrans) then
             h = h1
          else
             h = h0
          endif
          Time_minutes = (Float(jtime-1))*h/60.0
          if ((Time_minutes.ge.time0).and.(Time_minutes.lt.600.0))
     +                               volt = Vh1
          if (Time_minutes.lt.time0) volt = Vh0
c
c         Compute heater resistance and heater power:
          Rheater = (resis(Temp(jtime,1))*1.0e-06*length(1,1)*2.54)/
     +              (pi*(od(1,1)*2.54)**2)
          amp = volt/Rheater
          if (amp.gt.ampmax) amp = ampmax
          Pinput = volt*amp
          if (idiag.eq.1) then
             write(19,*)'Rheater(ohms) = ',Rheater,'Volts = ',volt,
     +        'amp = ',amp,' Pinput(watts) = ', Pinput
          endif
```

```
c
c       Compute thermal resistance of each part:
        if (idiag.eq.1) then
           write(19,*)'Thermal resistance of each part(deg.K/watt)'
           write(19,*)' Node index (i)',' Part index (j)',' Rt(i,j)'
        endif
        do 561 i = 1, inode
        do 561 j = 1, jpart(i)
          call thermres(mattype(i,j),Temp(jtime,i),length(i,j),
   +                              Adisk(i,j),Rt(i,j))
          if (idiag.eq.1) then
             write(19,171) i,j,Rt(i,j)
          endif
561      continue
c
c       Compute thermal resistance between two successive nodes
c       (where there will be conduction between nodes):
c       Initialization:
        do 562 in = 1, inode
        do 562 jn = 1, inode
          Rn(in,jn) = 0.0
562      continue
c       Here Rt(i,j) = thermal resistance of j-th part in i-th node
c       Rn(i,i+1) = thermal resistance between node i and i+1
        Rn(1,2) = Rt(2,1)
        Rn(2,3) = Rt(3,1) + 0.25*Rt(2,6)
        Rn(3,4) = Rt(4,1)
        Rn(4,5) = 0.5*Rt(5,1)
        Rn(5,6) = 0.5*(Rt(5,1) + Rt(6,1))
        Rn(6,7) = Rt(7,1) + Rt(7,2)
        Rn(6,10) = Rt(6,1)
        Rn(4,8) = 0.5*(Rt(8,1) + Rt(8,2) + Rt(8,4))
        Rn(8,9) = Rn(4,8) + Rt(9,2)
        if (idiag.eq.1) then
           write(19,*)'Thermal resistance between two adjacent nodes
   +                 (deg. K/watt)'
           write(19,*)'Ist node (i) ', '2nd node (j) ',' Rn(i,j)'
           do 560 i = 1, inode
           do 560 j = 1, inode
             write(19,171)i,j, Rn(i,j)
560         continue
        endif
c
c       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
c         Compute thermal power flow from one node to the other:
c         Initialization:
          do 563 in = 1, inode
          do 563 jn = 0, inode
            Pout(in,jn) = 0.0
563         continue
c
c         Calculate radiation from heater to cathode:
          Fac12 = 1.52
          P1t2rad = Fac12*acyl(1,1)*emis(temp(jtime,1))*sigma*
     +              (Temp(jtime,1)**4 - Temp(jtime,2)**4)
          Pout(1,2) = P1t2rad + (Temp(jtime,1) - Temp(jtime,2))/Rn(1,2)
          em = 0.32
          Pout(2,0) = adisk(2,5)*sigma*em*(Temp(jtime,2)**4 - Tamb**4)
c          write(19,*)'emissiv. of heater',' P1t2rad ',' Pout(2,0)'
c          write(19,170)emis(temp(jtime,1)),P1t2rad,Pout(1,2)
c
c         Calculate radiation from cathode support (473) to node 3:
          em = 0.2
          P2t3rad = em*acyl(2,6)*sigma*
     +              (temp(jtime,2)**4 - temp(jtime,3)**4)
          fac23 = 3.0
          Pout(2,3) = P2t3rad+fac23*
     +                (temp(jtime,2)-temp(jtime,3))/Rn(2,3)
c
c         Calculate emissivity of cathode surface:
          em = 0.0522 + 0.000041*temp(jtime,2) +
     +        1.87e-08*temp(jtime,2)**2
          vf = 0.333
          Pout(2,5) = adisk(2,5)*em*vf*sigma*
     +                (temp(jtime,2)**4 - temp(jtime,5)**4)
          Pout(3,4) = (temp(jtime,3) - temp(jtime,4))/Rn(3,4)
c          write(19,*)'emissivity of cathode surface = ',em
c
c         Calculate radiation from shadow grid support to gun ceramic:
          em = 0.07
          Pout(3,8) = (acyl(3,3)+acyl(3,6))*em*sigma*
     +                (temp(jtime,3)**4 - temp(jtime,8)**4)

c
c         Calculate radiation from shadow grid support to control grid support:
          em = 0.0522 + 0.000041*temp(jtime,3) +
     +        1.87e-08*temp(jtime,3)**2
          Pout(3,9) = adisk(3,6)*em*sigma*
     +              (temp(jtime,3)**4 - temp(jtime,9)**4)
c          write(19,*)'emissivity of shadow grid support (to cg) = ',em
c
           Pout(4,5) = (temp(jtime,4)-temp(jtime,5))/Rn(4,5)
           Pout(4,8) = (temp(jtime,4)-temp(jtime,8))/Rn(4,8)
           Pout(5,6) = (temp(jtime,5)-temp(jtime,6))/Rn(5,6)
c
           emvf = 0.07
           P6t10rad = acyl(6,1)*emvf*sigma*
     +             (temp(jtime,6)**4-temp(jtime,10)**4)
           fac610 = 0.3
           Pout(6,10) = P6t10rad + (temp(jtime,6)-temp(jtime,10))/fac610
c
           Pout(6,7) = (temp(jtime,6) - temp(jtime,7))/Rn(6,7)
c
           em = 0.07
           Pout(7,0) = (acyl(7,1)+acyl(7,2))*em*sigma*
     +                          (temp(jtime,7)**4 - tamb**4)
c
           em = 1.0322 + 7.5763e-05*temp(jtime,8) -
     +     1.0828e-06*temp(jtime,8)**2 + 4.514e-10*temp(jtime,8)**3
c          write(19,*)'emissivity of ceramic parallel to SG
c     +    support = ',em
           Pout(8,7) = em*sigma*(acyl(8,3)+adisk(8,3))*
     +              (temp(jtime,8)**4 - temp(jtime,7)**4)
           P8t9rad = em*sigma*(acyl(8,3)+adisk(8,3))*
     +            (temp(jtime,8)**4 - temp(jtime,9)**4)
           Pout(8,9) = P8t9rad + (temp(jtime,8) - temp(jtime,9))/Rn(8,9)
           Pout(8,10) = em*sigma*acyl(8,4)*
     +              (temp(jtime,8)**4 - temp(jtime,10)**4)
c
           em = 0.0522 + 0.000041*temp(jtime,9) +
     +        1.87e-08*temp(jtime,9)**2
c          write(19,*)'emissivity of control grid support = ',em
           Pout(9,0) = em*sigma*acyl(9,2)*(temp(jtime,9)**4 - tamb**4)
c
           Pout(10,0) = (temp(jtime,10) - tamb)/0.5
c
c          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c          Compute net power flow into each node:
           do 564 i = 1, inode
             if (i.eq.1) then
                Pin(i) = Pinput
             else
```

```
              Pin(i) = 0.0
          endif
          jmax = i - 1
          do 564 j = 1, jmax
             Pin(i) = Pout(j,i) + Pin(i)
564     continue
c
c       Compute net power out from each node:
        do 565 i = 1, inode
           PoutT(i) = 0.0
           do 565 j = 0, inode
              PoutT(i) = Pout(i,j) + PoutT(i)
565     continue
c
c       Compute updated temperature of each node:
        if (jtime.lt.jtmax) then
            do 566 i = 1, inode
              temp(jtime+1,i)=temp(jtime,i)+h*(Pin(i)-PoutT(i))/mcn(i)
566         continue
        endif
c
c       writes:
        if (idiag.eq.1) then
            write(19,*)'Node # ',' Net Power input(watts)'
            do 567 i = 1, inode
               write(19,*) i, Pin(i)
567         continue
            write(19,*)'Node # ',' Net Power out(watts)'
            do 568 i = 1, inode
               write(19,*) i, PoutT(i)
568         continue
            write(19,*)'Node # ',' Temperature(deg.K)'
            do 569 i = 1, inode
               write(19,*) i, temp(jtime+1,i)
569         continue
        endif
56    continue
c     ------------------------------------------------------
c     ------------------------------------------------------
c
      write(18,*)'node#  ','  mass*sp.ht.'
      do 57 ii = 1, inode
        write(18,168)ii,mcn(ii)
57    continue
```

```
c      Write temperature (deg.C) of cathode (tempc.dat) and
c      heater (temph.dat) in every 10th time step:
       temp(1,2) = temp(1,2) - 273.0
       write(20,*)0.0,temp(1,2)
       do 58 jtime = 10, jtmax, 10
c         Cathode temp:
          temp(jtime,2)= temp(jtime,2) - 273.0
          Time_minutes = (float(jtime-1))*h/60.0
          write(20,*)Time_minutes,temp(jtime,2)
58      continue
c       write(20,*)9999., 0.
       write(20,200)
200     format(/)
       temp(1,1) = temp(1,1) - 273.0
       write(20,*)0.0,temp(1,1)
       do 59 jtime = 10, jtmax, 10
c         Heater temp:
          temp(jtime,1)= temp(jtime,1) - 273.0
          Time_minutes = (float(jtime-1))*h/60.0
          write(20,*)Time_minutes,temp(jtime,1)
59      continue
c
166     format (1x,3(e13.6,3x),i2)
167     format (1x,i2,3x,3(e13.6,4x),3x,e13.6,3x,e13.6)
168     format (1x,i2,4x,e13.6)
169     format (1x,e13.6)
170     format (1x,3(e13.6,3x))
171     format (1x,2(i2,4x),e13.6)
       close(unit=15)
       close(unit=16)
       close(unit=17)
       close(unit=18)
       close(unit=19)
       close(unit=20)
       stop
       end
```

```
c
c
      subroutine thermres(itype,T,L,Area,Rtherm)
c ******************************************************************
c      This subroutine computes thermal resistance of a part for given
c      temperature (T) and thermal conducitivity (k).
c
c      Input: iType = material type
c             T     = temperature (deg.K)
c             L     = length of a part (inch)
c             Area  = area of a part (inch**2)
c
c      Output:k     = thermal conductivity (watts/inch/deg.C)
c                              or (watts/inch/deg.K)
c             Rtherm = thermal resistance  (deg.C/watt)
c*******************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      real*8 L, k
      if (itype.eq.1) then
         k = (1.34688 - 0.0001629*T + 1.0e-09*T**2)*2.54
      elseif (itype.eq.5) then
         k = (0.148 + 0.0000198*T + 1.35e-07*T**2)*2.54
      elseif (itype.eq.2) then
         k = (1.76 - 0.000575*T)*2.54
      elseif (itype.eq.3) then
           if (T.gt.631.0) then
              k = (0.426 + 0.00025*T)*2.54
           else
              k = (1.19 - 0.00096*T)*2.54
           endif
      elseif (itype.eq.6) then
         k = (0.64 - 0.00116*T + 7.34e-07*T**2)*2.54
      elseif (itype.eq.4) then
         k = (0.0651 + 0.000183*T + 3.69e-08*T**2)*2.54
      endif
      Rtherm = L/(k*Area)
      return
      end
```

```
c
      function resis(T)
c *****************************************************************
c        This subroutine computes electrical resistivity of heater for
c        a given temperature (T).
c
c        Input:
c              T      = temperature (deg.K)
c
c        Output: resis(T) = electrical resistivity
c*****************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      if (T.gt.2600.0) then
         resis = 85.0
      else
         resis = -2.12 + 0.0241*T + 3.11e-06*T*T - 2.36e-10*T**3
      endif
      return
      end
```

```
c
       function emis(T)
c ********************************************************************
c       This subroutine computes thermal emissivity of heater coil at
c       a given temperature (T).
c
c       Input:
c              T      = temperature (deg.K)
c
c       Output: emis(T) = thermal emissivity
c
c       Rev. History: Dec 30, 1993 (SRB)
c*********************************************************************
       implicit integer (i-n)
       implicit real*8 (a-h,o-z)
       if (T.gt.2600.0) then
          emis = 0.37
       else
          emis = -0.0144 + 0.0000734*T + 6.58e-08*T*T - 1.68e-11*T**3
       endif
       return
       end
```

APPENDIX 3D: Output Data File

```
0.0000000E+00   26.79998779296875
0.3000000000000000       29.31898263500668
0.6333333333333333       47.55459741616106
0.9666666666666667       66.75569499779976
 1.300000000000000       85.18700454694306
 1.633333333333333       102.9149830475758
 1.966666666666667       120.0001762787521
 2.300000000000000       150.9224415617416
 2.633333333333333       181.4929230793582
 2.966666666666667       210.3136579309904
 3.300000000000000       237.6163543733053
 3.633333333333333       263.5884191434148
 3.966666666666667       288.3824575757636
 4.300000000000000       312.1233732321649
 4.633333333333333       334.9138184897735
 4.966666666666667       356.8384623799327
 5.300000000000000       377.8474785444233
 5.633333333333333       397.8343341868486
 5.966666666666667       416.8806600483463
 6.300000000000000       435.0588636008560
 6.633333333333333       452.4327029421935
 6.966666666666667       469.0587511079895
 7.300000000000000       484.9874777438392
 7.633333333333333       500.2641021845019
 7.966666666666667       514.9292935537624
 8.300000000000000       529.0197597784422
 8.633333333333333       542.5687503114555
 8.966666666666667       555.6064884986511
 9.300000000000000       568.1605447857408
 9.633333333333333       580.2561592919421
 9.966666666666667       591.9165206536326
 10.30000000000000       600.9169315968364
 10.63333333333333       609.2403230561918
 10.96666666666667       617.4529974241267
 11.30000000000000       625.5355244003309
 11.63333333333333       633.4737127309710
 11.96666666666667       641.2573112675518
 12.30000000000000       648.8777740571761
 12.63333333333333       656.2091796686925
 12.96666666666667       663.2047553480657
 13.30000000000000       669.8959401277021
 13.63333333333333       676.3098946891682
```

13.96666666666667	682.4703280920418
14.30000000000000	688.3980565540503
14.63333333333333	694.1114273196522
14.96666666666667	699.6266549994558
15.30000000000000	704.9580089347375
15.63333333333333	710.1165353027911
15.96666666666667	715.1112565502786
16.30000000000000	719.9511038596462
16.63333333333333	724.6446639914762
16.96666666666667	729.1999810588728
17.30000000000000	733.6244718576074
17.63333333333333	737.9249082020815
17.96666666666667	742.1074371593422
18.30000000000000	746.1776212495630
18.63333333333333	750.1404877428753
18.96666666666667	754.0005806180037
19.30000000000000	757.7620115126342
19.63333333333333	761.4285077000786
19.96666666666667	765.0034561621129
20.30000000000000	768.4899434427406
20.63333333333333	771.8907913203190
20.96666666666667	775.2085885262438
21.30000000000000	778.4457188305466
21.63333333333333	781.6043858481192
21.96666666666667	784.6866349186880
22.30000000000000	787.6943723945372
22.63333333333333	790.6293826417341
22.96666666666667	793.4933430289134
23.30000000000000	796.2878371458356
23.63333333333333	799.0143664637881
23.96666666666667	801.6743606223719
24.30000000000000	804.2691865027042
24.63333333333333	806.8001562256029
24.96666666666667	809.2685341947836
25.30000000000000	811.6755432892382
25.63333333333333	814.0223702955049
25.96666666666667	816.3101706591727
26.30000000000000	818.5400726254103
26.63333333333333	820.7131808302846
26.96666666666667	822.8305793979000
27.30000000000000	824.8933345927202
27.63333333333333	826.9024970716393
27.96666666666667	828.8591037762824
28.30000000000000	830.7641795025028

28.63333333333333	832.6187381809791
28.96666666666667	834.4237839001148
29.30000000000000	836.1803117000247
29.63333333333333	837.8893081641943
29.96666666666667	839.5517518333801
30.30000000000000	841.1686134644368
30.63333333333333	842.7408561549923
30.96666666666667	844.2694353532193
31.30000000000000	845.7552987703667
31.63333333333333	847.1993862121971
31.96666666666667	848.6026293440366
32.30000000000000	849.9659514027637
32.63333333333333	851.2902668677546
32.96666666666667	852.5764811015611
33.30000000000000	853.8254899699179
33.63333333333333	855.0381794495695
33.96666666666667	856.2154252313703
34.30000000000000	857.3580923251381
34.63333333333333	858.4670346718418
34.96666666666667	859.5430947678747
35.30000000000000	860.5871033053964
35.63333333333333	861.5998788320300
35.96666666666667	862.5822274325686
36.30000000000000	863.5349424347673
36.63333333333333	864.4588041407897
36.96666666666667	865.3545795854150
37.30000000000000	866.2230223217079
37.63333333333333	867.0648722345006
37.96666666666667	867.8808553817221
38.30000000000000	868.6716838633472
38.63333333333333	869.4380557175060
38.96666666666667	870.1806548431069
39.30000000000000	870.9001509481614
39.63333333333333	871.5971995228748
39.96666666666667	872.2724418364581
40.30000000000000	872.9265049565398
40.63333333333333	873.5600017899961
40.96666666666667	874.1735311439771
41.30000000000000	874.7676778058823
41.63333333333333	875.3430126410297
41.96666666666667	875.9000927067614
42.30000000000000	876.4394613817436
42.63333333333333	876.9616485092384
42.96666666666667	877.4671705531516

43.30000000000000	877.9565307656960
43.63333333333333	878.4302193655474
43.96666666666667	878.8887137254140
44.30000000000000	879.3324785679823
44.63333333333333	879.7619661692536
44.96666666666667	880.1776165683307
45.30000000000000	880.5798577827643
45.63333333333333	880.9691060286194
45.96666666666667	881.3457659444707
46.30000000000000	881.7102308185826
46.63333333333333	882.0628828185820
46.96666666666667	882.4040932229732
47.30000000000000	882.7342226538941
47.63333333333333	883.0536213105553
47.96666666666667	883.3626292028438
48.30000000000000	883.6615763846184
48.63333333333333	883.9507831862574
48.96666666666667	884.2305604460605
49.30000000000000	884.5012097401386
49.63333333333333	884.7630236104618
49.96666666666667	885.0162857907642
50.30000000000000	885.2612714300358
50.63333333333333	885.4982473133605
50.96666666666667	885.7274720798819
51.30000000000000	885.9491964377078
51.63333333333333	886.1636633755840
51.96666666666667	886.3711083711909
52.30000000000000	886.5717595959357
52.63333333333333	886.7658381161322
52.96666666666667	886.9535580904771
53.30000000000000	887.1351269637483
53.63333333333333	887.3107456566624
53.96666666666667	887.4806087518486
54.30000000000000	887.6449046759003
54.63333333333333	887.8038158774851
54.96666666666667	887.9575190014983
55.30000000000000	888.1061850592582
55.63333333333333	888.2499795947492
55.96666666666667	888.3890628469258
56.30000000000000	888.5235899080985
56.63333333333333	888.6537108784305
56.96666666666667	888.7795710165761
57.30000000000000	888.9013108865012
57.63333333333333	889.0190665005282

```
 57.96666666666667       889.1329694586526
 58.30000000000000       889.2431470841828
 58.63333333333333       889.3497225557559
 58.96666666666667       889.4528150357882
 59.30000000000000       889.5525397954183
 59.63333333333333       889.6490083360041
 59.96666666666667       889.7423285072380

0.0000000E+00   26.79998779296875
0.3000000000000000       1047.878376322195
0.6333333333333333       1164.058287736979
0.9666666666666667       1164.565826284483
 1.300000000000000       1164.850838908240
 1.633333333333333       1165.155140213939
 1.966666666666667       1165.479641357076
 2.300000000000000       1166.154055516540
 2.633333333333333       1166.947060298625
 2.966666666666667       1167.826188377395
 3.300000000000000       1168.792095911000
 3.633333333333333       1169.844926679084
 3.966666666666667       1170.984412098801
 4.300000000000000       1172.209914213049
 4.633333333333333       1173.520436619469
 4.966666666666667       1174.914619629299
 5.300000000000000       1176.383876838241
 5.633333333333333       1177.909307504554
 5.966666666666667       1179.485228915545
 6.300000000000000       1181.106574305963
 6.633333333333333       1182.768468117486
 6.966666666666667       1184.466249685231
 7.300000000000000       1186.195480681939
 7.633333333333333       1187.951944406480
 7.966666666666667       1189.731641024003
 8.300000000000000       1191.530781137601
 8.633333333333333       1193.345778991585
 8.966666666666667       1195.173245942028
 9.300000000000000       1197.009984435099
 9.633333333333333       1198.852982510700
 9.966666666666667       1200.699408734373
 10.30000000000000       1183.546075426721
 10.63333333333333       1184.998867567045
 10.96666666666667       1186.470041758996
 11.30000000000000       1187.954928737595
 11.63333333333333       1189.449499590420
```

11.96666666666667	1190.950228351736
12.30000000000000	1192.454082823521
12.63333333333333	1193.936770272771
12.96666666666667	1195.381069038265
13.30000000000000	1196.789810276023
13.63333333333333	1198.165485588561
13.96666666666667	1199.510326419741
14.30000000000000	1200.826340719542
14.63333333333333	1202.115336894063
14.96666666666667	1203.378941279234
15.30000000000000	1204.618601073382
15.63333333333333	1205.835269923252
15.96666666666667	1207.029523281206
16.30000000000000	1208.202054988908
16.63333333333333	1209.353621400804
16.96666666666667	1210.484978307905
17.30000000000000	1211.596844665470
17.63333333333333	1212.689883402695
17.96666666666667	1213.764692922407
18.30000000000000	1214.821805144378
18.63333333333333	1215.861687421597
18.96666666666667	1216.884746625209
19.30000000000000	1217.891334323983
19.63333333333333	1218.881752393192
19.96666666666667	1219.856258651611
20.30000000000000	1220.815072294282
20.63333333333333	1221.758378995692
20.96666666666667	1222.686335624735
21.30000000000000	1223.599074553587
21.63333333333333	1224.496707566561
21.96666666666667	1225.379329388333
22.30000000000000	1226.247020857455
22.63333333333333	1227.099851773543
22.96666666666667	1227.937883446426
23.30000000000000	1228.761170974113
23.63333333333333	1229.569765274243
23.96666666666667	1230.363714891256
24.30000000000000	1231.143067599055
24.63333333333333	1231.907871816629
24.96666666666667	1232.658177852041
25.30000000000000	1233.394038988350
25.63333333333333	1234.115512423550
25.96666666666667	1234.822660075301
26.30000000000000	1235.515549260218

26.63333333333333	1236.194253256661
26.96666666666667	1236.858851759327
27.30000000000000	1237.509431233422
27.63333333333333	1238.146085175842
27.96666666666667	1238.768914290444
28.30000000000000	1239.378026584282
28.63333333333333	1239.973537391452
28.96666666666667	1240.555569331030
29.30000000000000	1241.124252205391
29.63333333333333	1241.679722845045
29.96666666666667	1242.222124905918
30.30000000000000	1242.751608624830
30.63333333333333	1243.268330538673
30.96666666666667	1243.772453172596
31.30000000000000	1244.264144702202
31.63333333333333	1244.743578594541
31.96666666666667	1245.210933232359
32.30000000000000	1245.666391525786
32.63333333333333	1246.110140515340
32.96666666666667	1246.542370969819
33.30000000000000	1246.963276982329
33.63333333333333	1247.373055567412
33.96666666666667	1247.771906261917
34.30000000000000	1248.160030731976
34.63333333333333	1248.537632388150
34.96666666666667	1248.904916010554
35.30000000000000	1249.262087385493
35.63333333333333	1249.609352954915
35.96666666666667	1249.946919479754
36.30000000000000	1250.274993718000
36.63333333333333	1250.593782118193
36.96666666666667	1250.903490528789
37.30000000000000	1251.204323923753
37.63333333333333	1251.496486144520
37.96666666666667	1251.780179658408
38.30000000000000	1252.055605333370
38.63333333333333	1252.322962228951
38.96666666666667	1252.582447403157
39.30000000000000	1252.834255734917
39.63333333333333	1253.078579761733
39.96666666666667	1253.315609532058
40.30000000000000	1253.545532471903
40.63333333333333	1253.768533265148
40.96666666666667	1253.984793746985

41.30000000000000	1254.194492809913
41.63333333333333	1254.397806321704
41.96666666666667	1254.594907054724
42.30000000000000	1254.785964626012
42.63333333333333	1254.971145447525
42.96666666666667	1255.150612685930
43.30000000000000	1255.324526231389
43.63333333333333	1255.493042674741
43.96666666666667	1255.656315292520
44.30000000000000	1255.814494039292
44.63333333333333	1255.967725546749
44.96666666666667	1256.116153129087
45.30000000000000	1256.259916794160
45.63333333333333	1256.399153259948
45.96666666666667	1256.533995975908
46.30000000000000	1256.664575148759
46.63333333333333	1256.791017772322
46.96666666666667	1256.913447661013
47.30000000000000	1257.031985486649
47.63333333333333	1257.146748818198
47.96666666666667	1257.257852164182
48.30000000000000	1257.365407017404
48.63333333333333	1257.469521901736
48.96666666666667	1257.570302420690
49.30000000000000	1257.667851307535
49.63333333333333	1257.762268476713
49.96666666666667	1257.853651076361
50.30000000000000	1257.942093541721
50.63333333333333	1258.027687649262
50.96666666666667	1258.110522571341
51.30000000000000	1258.190684931246
51.63333333333333	1258.268258858477
51.96666666666667	1258.343326044128
52.30000000000000	1258.415965796252
52.63333333333333	1258.486255095095
52.96666666666667	1258.554268648097
53.30000000000000	1258.620078944574
53.63333333333333	1258.683756309982
53.96666666666667	1258.745368959710
54.30000000000000	1258.804983052314
54.63333333333333	1258.862662742142
54.96666666666667	1258.918470231295
55.30000000000000	1258.972465820870
55.63333333333333	1259.024707961456

55.96666666666667	1259.075253302831
56.30000000000000	1259.124156742842
56.63333333333333	1259.171471475428
56.96666666666667	1259.217249037774
57.30000000000000	1259.261539356576
57.63333333333333	1259.304390793393
57.96666666666667	1259.345850189082
58.30000000000000	1259.385962907308
58.63333333333333	1259.424772877116
58.96666666666667	1259.462322634565
59.30000000000000	1259.498653363429
59.63333333333333	1259.533804934953
59.96666666666667	1259.567815946679

4. INPUT CAVITY SUBSYSTEM

The input cavity subsystem consists of a variable-dimension circuit that symmetrizes the rf signal at the input cavity and a software program to control the circuit. The circuit consists of a bandpass filter to tailor the signal's frequency response and an impedance transformer to maintain the rf amplitude between the 50 Ω seen at the type-N input connector and the 12000Ω beam seen at the input cavity. The enhanced input signal near the half-power points provided by the transformer results in increased bandwidth in both cold test and hot test.

Figure 4.0-1 illustrates the layout of the input cavity circuit on the klystron. Located outside the vacuum envelope but just in front of the input window, the input circuit is inserted within the solenoidal magnet along with the tube body. Control of the circuit by the stepper motor is achieved by threading a cable along the tube body then out of the magnet at the rf output end of the tube. The expert system steps through the circuit configurations and thereby shapes the input cavity response curve.

Figure 4.0-2 illustrates the three input circuit designs of this program. Each of these combines properties of a band-pass filter and an impedance transformer but is designated simply as first, second and third generation "transformers." The first and second generation designs achieved the main objective of getting maximum power to the electron beam over a given frequency band by externally tuning a circuit in front of the input cavity. However the designs were less than ideal in that the first generation had no practical way to vary the circuit parameters and the second had a convenient way of varying only one circuit parameter. By contrast, the third generation design provides a convenient way to continuously vary two parameters simultaneously.

Descriptions of the designs, the tests and the modeling of the earlier generations appear in Section 4.1 and those of the third generation appear in Sections 4.2 and 4.3. The software that interfaces with and controls the circuit is described in Section 4.4.

4.1 First and Second Generation Designs

The first and second generation designs achieved the main objective of getting maximum power to the electron beam over a given frequency band but used an approach that was inconvenient. The following sections describe the external tuning approach and typical test results.

4.1.1 General descriptions

The first generation input transformer utilized in tests is shown in Figure 4.1-1. It consists of a center conductor incorporated into a standard type-N input connector onto which a fixed length slug (choke) of various diameter options is attached with set screws at various axial positions. Changing the transformer's configuration required shutting down the test, removing the klystron from both the socket and focusing solenoid, unscrewing both the outer and inner conductors, loosening the two set screws from the choke, readjusting the axial position of the choke as needed and changing to a slug with different outer diameter. This first concept and procedure was found to be useful but extremely costly and time consuming in both cold test and hot test.

The second generation input "transformer" is shown in Figure 4.1-2. The low-impedance-section slug lies on an axially sliding contact inside the coax outer conductor instead of on the outside of the inner conductor. The slug's sleeves are plated, heat-treated beryllium copper to assure good microwave contact at all positions.

This slug attaches to a right-angled drive mechanism that protrudes through a slit in the coax outer wall. This allows axial repositioning of the slug without having appreciable leakage. The right-angled drive mechanism links to a flexible cable that contains a rotatable center conductor. The cable passes through the focusing solenoid, exists the tube at the collector end, extends out of the x-ray shielded test area and enters a control box that can be manipulated by the test technician. Once adjusted in test, the sleeve can be locked in position.

4.1.2 First generation "transformer" test results

The following results show the effect of three variables (loop coupling, transformer diameter and position of transformer) on the frequency response.

The measurement results shown in Fig. 4.1-3A were made with a slug outer diameter of 0.470 inches. Curve (a) shows a desired input-cavity response profile when the transformer is set with a nominal inductive loop position, referred to as LP, and a slug axial position of 0.346 inches. This response curve has a 3 dB bandwidth of 86 MHz symmetrically positioned about the center frequency. The power variation between the low end and high end response peaks is approximately 0.5 dB. When the slug's axial position is increased 0.015 inches to 0.361 inches, the low frequency response is enhanced and the high frequency response is lowered as shown by curve (b). Conversely, a decrease of 0.015 in the slug's axial position skews the response towards the high frequencies (curve (c)). In both cases where the slug is moved from the reference position, the cavity bandwidth is decreased.

Figure 4.1-3B shows the effect of varying the coupling loop depth while holding constant the slug axial position. Curve (a) is taken for the same reference settings as was done for curve (a) in Fig. 4.1-3A. Curves (d) and (e) show the result of decreasing the loop depth by 0.015 inches and 0.030 inches respectively. In both cases, the reduced coupling raises the circuit Q thereby decreasing the bandwidths to 78 MHz and 69 MHz respectively.

The sets of measurements in Fig. 4.1-4 were made after the transformer slug outer diameter was increased 0.010 inches to 0.480 inches. Curve (a) of Fig. 4.1-4 again corresponds to a nominal loop position of LP. Comparison of curves (a) of Figures 4.1-3 and 4.1-4 shows the larger slug diameter increases the bandwidth from 86 to 89 MHz and increases the response variation across the band from 0.5 dB to 1.2 dB. The curves in Fig. 4.1-4A show that as the axial position is increased (or decreased) from the reference point, as was done for the curves in Fig. 4.1-3A, the high frequency end (or low frequency end) is enhanced. The bandwidth reduction for the larger diameter slug was slightly less.

Figure 4.1-4B shows that the effect of decreasing the coupling loop depth with the larger diameter slug is similar to that for the smaller diameter slug. Curves (d) and (e) show that decreases of 0.015 inches and 0.030 inches reduced the coupling, raised the circuit Q, and thereby decreased the bandwidths.

4.1.3 Circuit modeling results

The discussion that follows concerns work on the first generation design. However, since the circuit model is the same for the second generation, the results apply equally to the second generation design also. An input cavity and input transformer were modeled on the computer using commercial software named "Superstar." Model details are described in Section 4.3 below.

Figure 4.1-5 shows results of modeling the 0.470 diameter slug set at the reference position. Note that the agreement with curve (a) of Fig. 4.1-3A is excellent. The center frequency is attenuated approximately 0.5 dB below the peaks and both 3 dB frequencies appear the same as with the measured data.

Figure 4.1-6 shows results of modeling the 0.470 diameter slug offset by ± 0.015 inches from the reference position. As with the measurements, the curves become skewed but the amount of skewing is calculated to be less than what is measured.

Figure 4.1-7 shows results of modeling the 0.480" diameter slug set at the reference position. The curves compare well with curve (e) of Fig. 4.1-4B. The calculated response variation is 1.0 dB compared to the measured 1.5 dB.

Figure 4.1-8 shows results of modeling the 0.480 diameter slug offset by ± 0.015 inches from the reference position. As with the measurements, the curves become skewed but the amount of skewing is calculated to be less than what is measured.

Discrepancies between calculated and measured responses likely originate from (1) not including the fringing capacity of the transformer in the model and (2) using theoretical scattering parameters to model the klystron coaxial input window. However, the created model correlates well enough to predict how to modify the transformer geometry to achieve the desired cavity response function.

4.1.4 Second generation transformer test results

A prototype of the design in Fig. 4.1-2A was constructed without a drive mechanism for cold test evaluation. Shown in Fig. 4.1-9 is the measured input cavity response for slug axial position changes of ± 0.015 inch. The response is nearly equivalent to that shown in Fig. 4.1-3 for the center slug transformer. In both figures the labels (a), (b) and (c) correspond to the same displacements. Although the prototype dimensions are not optimized and may still present an impedance discontinuity, the concept validity has been demonstrated.

4.2 Third Generation Transformer

4.2.1 Eccentric coax line approach for impedance changes

For the third generation transformer, the slug-section characteristic impedance was made continuously adjustable by utilizing variable eccentricity. Any eccentricity change resulted in an impedance change. This improved the first and second generation transformers where the impedance was adjusted in steps by replacing inserts in a very time consuming manner. The third generation approach had the following features

a) The new adjustment was continuous.
b) The new and old adjustments were independent of each other.
c) The adjustment was ***non***-erratic.
d) The adjustment had enough range.
e) The design fits the available space.
f) The section was designed for low leakage.
g) The adjustment had reasonable sensitivity.
h) The responses included one within acceptable limits.

Other approaches, such as slide screw tuners, double-stub tuners and double-bead tuners were investigated but were not found suitable for this application.

Figure 4.2-1 illustrates the variable-eccentricity coaxial line. The key element of the eccentric transformer is a hollow eccentric cylindrical slug whose outer radius (R_3 in Fig. 4.2-1A) rotates about an axis that is off center from the center-conductor axis. The offset distance is given by $\frac{1}{2}e_{max}$ in Fig 4.2-1B. When the eccentric slug rotates, the distance 'e' between the center of the slug inner radius (R_2 in Fig. 4.2-1A) and the center-conductor axis varies between zero and e_{max} as shown by the two views in Fig. 4.2-1B and Fig. 4.2-1C. For given values of R_1 and R_2, the distance e is sufficient to define eccentricity. The variation of eccentricity between $0 \leq e \leq e_{max}$ results in variation of the characteristic impedance of the eccentric coax line as given by

$$Z_{ECC} = Z_{CONC} \cosh^{-1}\left(\frac{D^2 - d^2 - 4e^2}{2Dd}\right) \qquad \text{Eq. (1)}$$

Where	Z_{ecc}	=	characteristic impedance of an eccentric line
	Z_{conc}	=	characteristic impedance of a concentric line
	D	=	inner diameter of eccentric slug ($=2R_2$)
	d	=	outer diameter of inner conductor ($=2R_1$)
	e	=	eccentricity of coaxial line

The relationship between eccentricity increase and characteristic impedance decrease is shown in Table 4.2-1. As expected from the $\cosh^{-1}$ dependence in equation (1), the incremental changes in impedance become larger for large eccentricities. For a typical eccentric slug design, the

Eccentricity	**Impedance**	**Equivalent Diameter of First Generation Transformer**
0.000 inches	28.858 Ohms	0.371 inches
0.002	28.838	0.371
0.004	28.776	0.371
0.006	28.673	0.372
0.008	28.528	0.373
0.010	28.340	0.374
0.012	28.108	0.376
0.014	27.832	0.377
0.016	27.510	0.379
0.018	27.139	0.382
0.020	26.719	0.384
0.022	26.246	0.387
0.024	25.717	0.391
0.026	25.129	0.395
0.028	24.477	0.399
0.030	23.756	0.404
0.032	22.959	0.409
0.034	22.077	0.415
0.036	21.099	0.422
0.038	20.012	0.430
0.040	18.795	0.439
0.042	17.421	0.449
0.044	15.849	0.461
0.046	14.011	0.475
0.048	11.784	0.493

Parameters:	D	0.275 inches
	d	0.170 inches

Table 4.2-1
ECCENTRICITY vs.IMPEDANCE

impedance varied between 12 Ω and 30 Ω for an eccentricity e variation between 0.12 cm and 0 cm.

As shown in Fig. 4.2-1 the eccentric slug can simultaneously rotate (azimuthally) and translate (axially) because the slug is joined to the coax-line outer conductor by an axial-motion support structure with springy ends. This support structure, while minimizing internal rf current paths, maintains non-erratic, defined, microwave contact points throughout the adjustments because of air gaps between the springy ends. Fig. 4.2-1A indicates the location of these air gaps.

Also, the eccentric slug has slots that act as a spring to maintain electrical contact with the axial motion support structure and the outer casing of the input transformer (not shown in the figure). The eccentric slug has eight narrow axial slots, four slots originating at one end of the slug, 90 degrees apart, and four slots originating at the other end of the slug also 90 degrees apart but offset from the first group by 45 degrees. Since the EM fields propagate as TEM modes, the axial RF currents are not interrupted by the slots.

4.2.2 Mechanical drive

For the expert system to axially and azimuthally adjust the eccentric slug, the slug was coupled to two gear assemblies that were installed beside the input transformer outer casing. These gear assemblies were rotated by two captured screws which were driven by flexible cables originating from system-controlled stepper motors located outside the tube.

The mechanical drive layout shown in Fig. 4.2-2 illustrates four topics of interest, the coax line, the rotational adjustment, the axial adjustment, and the common drive components. These are described in detail below.

4.2.2.1 Coax line

The coaxial line begins at the coupling loop, bends 90 degrees, passes through the vacuum window, passes through the eccentric slug, then ends forming an N-type connector. The loop, 90° bend, and window sections together comprise the input coax assembly (# 372386) which is an integral part of the tube. The rigid support given by this assembly to the center conductor permits joining new parts with sufficient tolerance control to assure that the resulting dimensions of the N connector satisfy MIL-SPEC. 39012.

Assembly of the coax line after exhaust continues by screwing a threaded end of the center pin (# 444422) into the center conductor of the input coax assembly. The opposite end is centered in the outer part of the coax line (coax sleeve (# 444401)) with center pin support (# 444424). The angular orientation of the coax sleeve is fixed by tightening nut (# 444402) and coax nut retainer (# 444403). The combination of these components represent the transmission line between input connector and coupling loop.

4.2.2.2 Rotational adjustment

The rotational adjustment, which varies the impedance of the eccentric slug region consists of the slug (center transformer (# 444417)), a modified gear (# 444411) that rotates the slug, and a control rod (# 444414) that rotates the gear. A stepper motor drives the rod at the rotational positioning input shown in Fig. 4.2.2. The rotary control rod has to be coupled to the center transformer in such a way that the independence of rotational adjustment and axial adjustment is accomplished.

Fig. 4.2.3 shows the original approach to the rotational adjustment. The rotary control rod (# 444414) was keyed to the pinion modified (# 444410) according to standard mechanical procedures. The pinion modified (# 444410) was allowed to slide on key rotary control (# 444415) and rotary control rod (# 444414) axially. The pinion modified (# 444410) then drives gear modified (# 444411) which is rigidly attached to the center transformer (# 444417). The transformer (# 444416) was slotted to allow a certain range of circular motion of the center transformer (# 444417) for rotary adjustment. However this approach was too complicated and expensive.

Fig. 4.2.2 shows the approach whereby the rotary control road (# 444414), key rotary control (# 444415) and pinion modified (# 444410) were integrated into one piece. This one piece is called gear rod rotary control (# 444411-1). The axial sliding now happens along the teeth of gear rod rotary control (# 444411-1) rather than along the key rotary control (# 444415) illustrated in Fig. 4.2.3.

4.2.2.3 Axial adjustment

The axial adjustment, which moves the above-mentioned rotational adjustment, consists of three components, a control rod (# 444413), a control block (# 444406), and a transformer segment (# 444416). The transformer is rigidly attached to the drive adapter. The combination is then driven by the threaded spindle (called control rod #444413).

The input of the adjustment is shown in Fig. 4.2.2 under "axial positioning." The adaption from the flexible cable to the linear control rod (# 444406) uses the same components as for the rotational adjustment, coupler (# 444418) and nut (# 444412). The rotational motion has to be converted into a linear motion. This conversion was accomplished by a drive which has high resolution and is uni-directional. The rod linear control (# 444413) has a male thread on the outside while the block linear control (# 444406) has the same female thread on the inside. If the rod turns, since the block is prevented from turning, the block will move linearly. In order for the transformer (# 444417) to move along with the block (# 444406) it was rigidly attached to the block. The axial adjustment is now complete.

4.2.2.4 Common drive components

There are two common support blocks for both of the above mentioned adjustments. The support block control (# 444404), shown in both Fig. 4.2.2 and 4.2.3, serves as one set of bearings for rod, linear control (# 444413) and rod, rotary control (# 444414). The support block (# 444405) serves as the second set of bearings on the opposite end. In addition this block also accommodates the components (coupler, # 444418) (nut, # 444412) which mechanically adapt standard flexible cable to the two rods (rotary and linear).

4.3 Modeling and Cold Test

4.3.1 Computer model

The model-based input-cavity subsystem uses an equivalent circuit model of the "transformer" - the combination impedance transformer and bandpass filter - to generate predictions. The filter is comprised of the fixed-geometry input cavity inside the vacuum and the variable-geometry eccentric slug outside the vacuum. The impedance transformer is comprised of the same eccentric slug which forms a quarter-wave transformer and a loop transformer inside the vacuum.

The entire circuit was modeled using the equivalent elements shown in Fig. 4.3-1 and the circuit response was found using the commercial software "Superstar" (now called Eagleware), from which the file RD3.CKT listed in Fig. 4.3-2 was derived. Below is identification of the transmission line elements between the type "N" connector and the electron beam.

aa	75 Ω transmission line between connector and slug
bb	Fringing capacitor of eccentric slug
cc	Transmission line for eccentric slug section
dd	Fringing capacitor of eccentric slug
ee	75 Ω transmission line between slug and window
ff	50 Ω transmission line for window
gg	75 Ω transmission line between window and loop
hh	Loop transformer
ii	Beam impedance and gap capacitor
jj	Coupling capacitor for S_{12} measurements

In the model, to represent the axial movement of the eccentric slug, only the lengths of transmission lines aa and ee can be adjusted and only in a way that keeps the sum of the line lengths constant. To represent rotation of the eccentric slug, the characteristic impedance of the fixed-length transmission line cc can be varied. Both the axial and rotational positions of the slug can be varied independently as in the actual device.

Capacitors bb and dd represent the fringe fields on the sides of the eccentric slug, ee and gg represent the coax lines on the sides of the window, and transmission line ff represents the vacuum window, a window consisting of three lifesaver-shaped pieces of alumina brazed between kovar rings.

The window data in the model result from estimates obtained from having used the same window on a separate program in a matched 50-ohm line. Although preliminary S_{11} measurements confirmed the expected large reflections, additional window-matching work was considered outside the scope of the program.

The capacitor jj represents a small coupling capacitor that is added by the sampling probe when being used for cold test insertion loss (S_{21}) measurements. It is not part of the cavity and has no significant impact on S_{11} measurements.

4.3.2 Cold-test measurements

The cold-test measurement assembly, shown in Fig. 4.3-3, consists of a signal source, a bridge, the input cavity system and the scaler analyzer which measures the reflected power. The input cavity is loaded with lossy Teledeltos paper to simulate the beam loading resistance of 12,000 Ω. The coupling capacitor jj indicates that a probe was used inside the cavity to measure the insertion loss S_{12} defined by

$$\textit{Insertion Loss} = 20 \log_{10}|S_{12}|$$

Such insertion loss measurements, which were presented earlier in Figs. 4.1-4 and 4.1-5, also represent a measure of the transfer function of the input cavity system to the beam. In cold test insertion loss measurements are convenient for setting the circuit positions.

However, in hot test, such insertion loss measurements can be impractical. More convenient are measurements of return loss defined by

$$\textit{Return Loss} = 20 \log_{10}|S_{11}|$$

When there are no lossy elements in the input transformer circuit, the insertion loss is only from mismatches and is related to the return loss by

$$|S_{12}| = \sqrt{(1-|S_{11}|^2)}$$

Clearly, return loss can be used by the expert system in both cold and hot test. The return loss cold tests on the third generation transformer can represent hot test conditions.

Two eccentric input transformers were designed, built and cold tested. Results for one of these is shown in Fig. 4.3-4, where the cavity return loss is given for a series of axial (Z) and azimuthal (θ) positions of the eccentric slug. The optimum position is defined arbitrarily for Z = 0, θ = 0. The range of variations conveniently and quickly obtained from the eccentric slug variations is similar to the range and variations shown in the sections on the first and second generation designs which were inconvenient and time consuming.

4.4 System Tuning of the Transformer

4.4.1 Introduction

Expert system software can be applied to any adjustable portion of a microwave tube. For the input cavity subsystem, software called TIPTOE measures and adjusts the coax eccentricity and the slug axial position in order to achieve a balanced input cavity response.

Software development for input transformer adjustments began before any transformer hardware became available. Consequently, software was first developed to interact, not with an actual microwave tube, but with an equivalent-circuit approximation of the tube. The response of the equivalent circuit was obtained by a separate, commercial, software application SUPERSTAR. (ref: SUPERSTAR, Eagleware Corp., 1750 Mountain Glen, Stone Mountain, GA 30087) The version to interact only with software, TIPTOE1A, was conveniently developed offsite by Dr. Martin Lee, (ref: Dr. Martin Lee, GOAI, 1088 Dartmouth Lane, Los Altos, CA 94024) an expert in the field.

TIPTOE2A, the second generation of the expert system software, incorporated much more stringent requirements of the data for acceptance. Although the expert system software again interacted with the equivalent circuit model analyzed in SUPERSTAR, TIPTOE2A was developed to control hardware. The hardware necessary to connect the computer to the input cavity was either purchased or was already available at Litton, but, since the system was never assembled, the software interface with the hardware drivers was not developed.

TIPTOE1A controlled two adjustable parameters, the equivalent circuit parameters of the slug diameter and the slug axial position, until a response curve was accepted according two criteria. The first criterion was that for curves having two peaks, the ratio of the peaks had to be less than 1 dB. The second criterion was that at the 3 dB points on the side of the response curve exceed the bandwidth specification. TIPTOE2A controlled the eccentricity of the coax plug and the slug axial position. A response curve was accepted according the two criteria above as well as a third, that the ratio of a side peak to the center low point was less than 1.7 dB.

4.4.2 Expert procedure

The purpose of the expert system software is to mimic automatically the actions of a true expert. The true expert takes data, analyzes that data and then decides what changes to make to the system before repeating the sequence. Programming a computer to take data and make the adjustments is a straightforward task. The difficult task is to teach the computer to properly analyze the data and make the proper decisions.

A block diagram of the expert system software to accomplish these tasks is shown in Figure 4.4-1. After setting each of the adjustments to their initial positions, the first step is to take a set of data. This set includes all the data necessary to compare to the acceptance criteria. The raw data is transferred into the computer and any needed data reduction is performed. The reduced data is compared to the acceptance criteria, and if the criteria is met, the expert process is complete.

In the general (and most likely) case, the acceptance criteria is not met with the initial settings. In this case, one of the adjustments is incremented by a small amount and the resultant change in the data is measured. This adjustment is then returned to its initial state and the next adjustment is incremented. The change in the data is again measured and this procedure is repeated until each of the adjustments has been incremented and its response measured.

Once the responses to the incremental adjustments are known, the settings for all of the adjustments to bring the device operation into acceptance may be calculated. One method is to cast the adjustments and responses in the form of a matrix equation:

$$\{ \text{Adjustment Settings} \} \times \{ \text{Response Matrix} \} = \{ \text{Response} \}.$$

To find the desired adjustment settings, one need only invert the response matrix and multiply it by the desired response. However, for code stability reasons, the full adjustments are not made in one step. Instead, the adjustments are made to bring the new response half way to the acceptance criteria. This prevents overshooting the desired response, which may happen if the microwave tube response is nonlinear. By measuring the responses to only small increments, we have essentially linearized the microwave tube response.

At this point, the data is taken again and compared to the original data to ensure that the current adjustment settings are indeed better than the original settings. If they are not, the process is to return to the original settings and reduce the calculated step size by another factor of two. This continues until positive progress is made toward the acceptance criteria. Once this happens, the process of measuring the response of the microwave tube to small incremental adjustments is repeated, then the response matrix is inverted and new settings are calculated, etc. This total process is repeated until the acceptance criteria is met.

4.4.3 TIPTOE1A

The expert system software was developed incrementally. The first implementation, TIPTOE1A, was limited to two simple rules to govern the microwave match into the input cavity of the klystron. The microwave match was altered through two adjustments in the matching transformer. A further simplification made was that the actual klystron input cavity and matching transformer were modelled as lumped circuit elements and analyzed with a separate software package (SUPERSTAR). The two programs interacted by reading and writing files on the computer's hard disk drive.

The TIPTOE1A program is written in fortran and was compiled with Microsoft Fortran 5.0 operating under Microsoft DOS 5.0. A complete listing of the well-commented code may be found in Appendix 4A.

The microwave match into the input cavity may be obtained by measuring either of the S parameters, S_{11} (reflection) or S_{12} (transmission). The S_{12} parameter was chosen to be modelled in the lumped element circuit model (SUPERSTAR) and to be interpreted by the expert system software. An example of the S_{12} data is shown in Figure 4.4-2. There are two goals for the expert system software. The first is that the amplitudes of the two peaks be equal. Therefore, the acceptance criterion for this goal is that the ratio of the amplitudes of the two peaks be sufficiently close to unity. The second goal is that the width of the S_{12} trace be equal to a prespecified width. Again, the acceptance criterion is specified as a ratio -- the ratio of the measured width to the desired width -- must be sufficiently close to unity. This relatively simple set of goals was chosen for the first implementation since it requires a nontrivial solution to the matrix equation described above.

Use of the TIPTOE1A and SUPERSTAR programs may be accomplished by running each in a DOS window under Microsoft Windows on an IBM-compatible personal computer. Version 5.0 of DOS and version 3.1 of Windows were used. It is necessary to have the two programs running simultaneously and continually switch between the two. An example of this operation is given here.

It is convenient to install the SUPERSTAR software into the default \EAGLE subdirectory. Further, it is convenient to place the TIPTOE files and the START.CKT file in the \EAGLE\TIPTOE subdirectory. Next, start the SUPERSTAR program and open the START.CKT file through the Open *.CKT (Text) file... command under the File pulldown menu. This circuit provides a convenient starting point. Figure 4.4-4 shows what should appear on the screen. The TIPTOE1A program utilizes the amplitude of the S_{12} scattering parameter which is the double peaked curve plotted in the graph on the left side of the figure. The phase of S_{12} is plotted in the same graph, and the amplitude and phase of the S_{11} scattering parameter are plotted in the graph on the right side of the figure. This file must now be saved as DEMO.CKT in the \EAGLE\TIPTOE subdirectory by using the Save Circuit As... command under the File pulldown menu. The S parameters calculated by the SUPERSTAR program must also be saved now by using the Write S-Data... command under the File pulldown menu. This data should be saved in the file \EAGLE\TIPTOE\SIGNAL.OUT, which will be read by the TIPTOE1A program.

Now switch to the Windows Program Manager without exiting the SUPERSTAR program. This may be done by holding down the <ALT> key and pressing the <Tab> key until the Program Manager prompt is reached. Next, open a full-screen DOS window and change to the \EAGLE\TIPTOE directory. Typing TIPTOE1A at the command prompt will start the TIPTOE1A program. Figures 4.4-3(A-D) show an example of the output from the TIPTOE1A program. The user is immediately prompted for the desired width. This is the frequency width, in megahertz, of the response curve generated by the SUPERSTAR program and is the only variable input into the program.

After the width is entered, the TIPTOE1A program prints out several parameters. sParameter1 and sParameter2 are the two adjustable circuit parameters read in from the DEMO.CKT file. The TIPTOE1A program modifies these parameters to obtain the desired output signal. Next, several parameters calculated from the SIGNAL.OUT file are printed. The Itr and SI parameters are iteration numbers within the TIPTOE1A program. Amp1 and Amp2 are the amplitudes (in dB) of the two peaks and the first Ratio is their ratio. If TIPTOE1A fails to find two distinct peaks, the ratio of the amplitudes is set to zero. Finally, Width is the frequency width (3 dB down from the highest peak) measured from the SIGNAL.OUT file and the second Ratio is the ratio of Width and the desired frequency width. This is followed by the new circuit parameters S1 (= sParameter1) and S2 (= sParameter2) written into the DEMO.CKT file and finally by the prompt:

Press <return> after new signal has been generated by Eagle Software ... .

Every time this prompt appears, the user must switch to the SUPERSTAR program, open the DEMO.CKT file by using the Open *.CKT (Text) file... command under the File pulldown menu, save the newly generated data in the SIGNAL.OUT file by using the Write S-Data... command under the File pulldown menu, and then switch back to the TIPTOE1A program.

The TIPTOE1A program informs the user of the progress being made by the program. If the new circuit parameters calculated by the TIPTOE1A program result in a signal further from the desired result, the step size is halved and new circuit parameters are calculated. An example of this occurrence is shown in Figure 4.4-3A, beginning with the line: Convergence Criteria Failure:.

At the completion of a full (converging) iteration, the current circuit parameters, iteration numbers, amplitudes, width, and ratios are printed out. An example of this is shown at the top of Figure 4.4-3B.

The Convergence Criteria Failure line will also appear if the TIPTOE1A program fails to find two distinct peaks in the data from the SIGNAL.OUT file. In this case, the TIPTOE1A program again halves the step size and calculates new circuit parameters. An example of this is shown near the bottom of Figure 4.4-3B.

At the bottom of Figure 4.4-3D, five iterations have been completed. As can be seen from the previous iterations, both ratios are approaching unity. Further iterations will continue the progression of both ratios toward unity. The TIPTOE1A program does not have a convergence test built into it, so it will continue the iterations until the built-in maximum of 20 iterations is reached.

4.4.4 TIPTOE2A

The TIPTOE2A program (actually named DEMO2A) utilizes the basic TIPTOE1A formulation and algorithms. There are two goals and two adjustable circuit parameters that are solved for by inverting the response matrix, exactly as was done in TIPTOE1A. Again, DEMO2A interacts with the SUPERSTAR program rather than with actual hardware. The main features added to the TIPTOE1A program include convergence criterion for the ratio of the amplitudes of the two peaks and the ratio of the actual and desired frequency widths, and the addition of several more constraints on the response of the circuit.

The added constraints on the circuit response are shown in Figure 4.4-5. As in TIPTOE1A, the two amplitude peaks must be sufficiently close to each other and the frequency width (measured 3 dB down from the highest peak) must be within certain limits. In addition, all points above the high frequency limit (or below the low frequency limit) must be more than 3 dB below the highest peak, and the circuit response must be above the points shown at those specific frequencies.

A flow chart for the TIPTOE2A (DEMO2A) program is shown in Figure 4.4-6. The goals for the circuit response are hardwired into the code through the use of data and parameter statements. The basic TIPTOE1A algorithm is used to find the circuit parameters that give a response satisfying the amplitude peaks and frequency width goals. Once this solution has been found, the circuit response is checked against the rest of the acceptance criteria. At this point, the program terminates whether the circuit response meets all the goals or not. A simple addition to this code would be to alter the amplitude peaks and frequency width criterion (since they both have ranges of acceptability) and return to the TIPTOE1A algorithm. The source listings for the TIPTOE2A (DEMO2A) program is given in Appendix B along with the command used to create the program using the Microsoft fortran compiler (version 5.0). This program was created under Microsoft DOS version 5.0.

As with the TIPTOE1A program, use of the TIPTOE2A and SUPERSTAR programs may be accomplished by running each in a DOS window under Microsoft Windows on an IBM-compatible personal computer. Version 5.0 of DOS and version 3.1 of Windows were used. It is necessary to have the two programs running simultaneously and continually switch between the two. An example of this operation is given here.

It is convenient to place the TIPTOE2A (DEMO2A) files and a copy of the START.CKT file (from the \EAGLE\TIPTOE subdirectory) in the \EAGLE\TIPTOE2 subdirectory. Next, start the SUPERSTAR program and open the START.CKT file through the Open *.CKT (Text) file... command under the File pulldown menu. This circuit provides a convenient starting point. Figure 4.4-4 shows what should appear on the screen. As with the TIPTOE1A program, the TIPTOE2A (DEMO2A) program utilizes the amplitude of the S_{12} scattering parameter which is the double peaked curve plotted in the graph on the left side of the figure. This file must now be saved as DEMO.CKT in the \EAGLE\TIPTOE2 subdirectory by using the Save Circuit As... command under the File pulldown menu. The S parameters calculated by the SUPERSTAR program must also be saved now by using the Write S-Data... command under the File pulldown menu. This data should be saved in the file \EAGLE\TIPTOE2\SIGNAL.OUT, which will be read by the TIPTOE2A (DEMO2A) program.

Now switch to the Windows Program Manager without exiting the SUPERSTAR program. This may be done by holding down the <ALT> key and pressing the <Tab> key until the Program Manager prompt is reached. Next, open a full-screen DOS window and change to the \EAGLE\TIPTOE2 subdirectory. Typing TIPTOE2A at the command prompt will start the TIPTOE2A (DEMO2A) program. Figures 4.4-7(A-B) show an example of the output from the TIPTOE2A program. Note that there are no parameters entered by the user in this program.

The TIPTOE2A (DEMO2A) program immediately prints out several parameters. sParameter1 and sParameter2 are the two adjustable circuit parameters read in from the DEMO.CKT file. The TIPTOE2A (DEMO2A) program modifies these parameters to obtain the desired output signal. Next, several parameters calculated from the SIGNAL.OUT file are printed. The Itr and SI parameters are iteration numbers within the TIPTOE2A (DEMO2A) program. Amp1 and Amp2 are the amplitudes (in dB) of the two peaks and the first Ratio is their ratio. If TIPTOE2A (DEMO2A) fails to find two distinct peaks, the ratio of the amplitudes is set to zero. Finally, Width is the frequency width (3 dB down from the highest peak) measured from the SIGNAL.OUT file and the second Ratio is the ratio of Width and the desired frequency width. This is followed by the new circuit parameters S1 (= sParameter1) and S2 (= sParameter2) written into the DEMO.CKT file and finally by the prompt: Press <return> after new signal has been generated by Eagle Software ... .

Every time this prompt appears, the user must switch to the SUPERSTAR program, open the DEMO.CKT file by using the Open *.CKT (Text) file... command under the File pulldown menu, save the newly generated data in the SIGNAL.OUT file by using the Write S-Data... command under the File pulldown menu, and then switch back to the TIPTOE2A (DEMO2A) program.

The TIPTOE2A (DEMO2A) program informs the user of the progress being made by the program exactly as was done in the TIPTOE1A program. If the new circuit parameters calculated by the TIPTOE2A (DEMO2A) program result in a signal further from the desired result, the step size is halved and new circuit parameters are calculated.

At the completion of a full (converging) iteration, the current circuit parameters, iteration numbers, amplitudes, width, and ratios are printed out. An example of this is shown in the middle of Figure 4.4-7A. The convergence criteria built into the TIPTOE2A (DEMO2A) program requires that both of the ratios (amplitude and width) be between 0.995 and 1.005. When this occurs (as it does near the bottom of Figure 4.4-7B), the circuit response is checked against each of the criteria described above and shown if Figure 4.4-5. The results of these checks are printed out, and the program stops whether or not the data satisfies all of the checks. A straightforward addition to this code would be to alter the amplitude and width convergence criteria (since there is an acceptable range for each) and return to the TIPTOE algorithm to search for another acceptable set of circuit parameters.

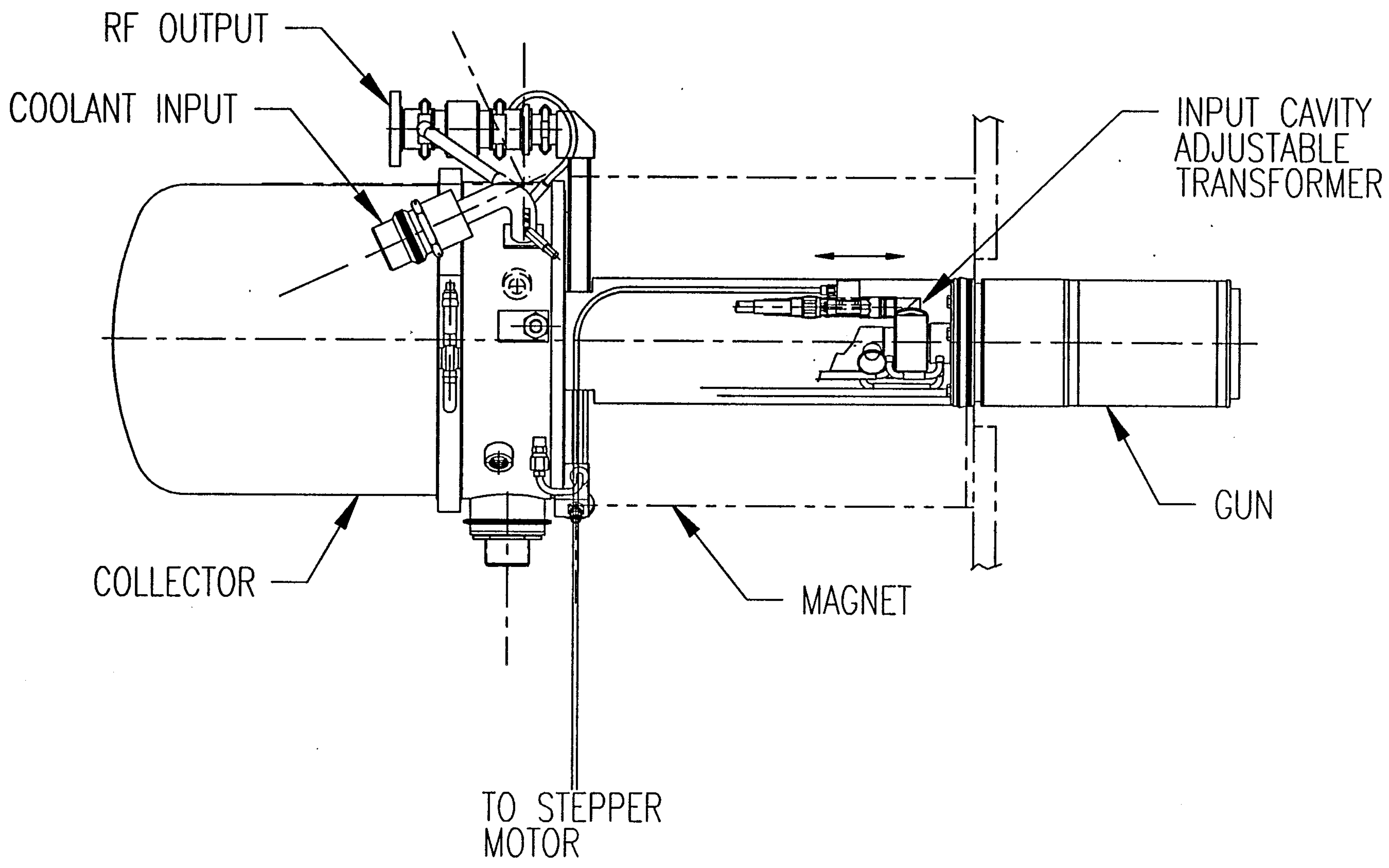
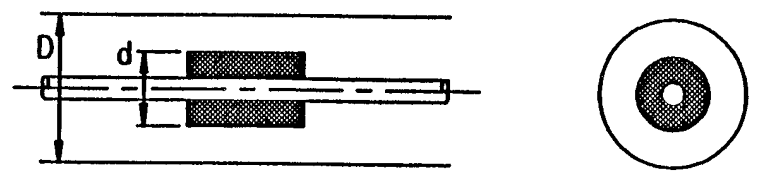
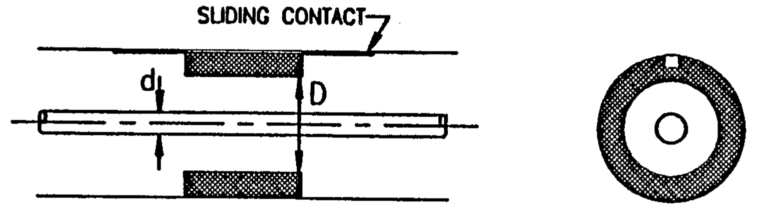


Figure 4.0-1
LOCATION OF ADJUSTABLE CIRCUIT



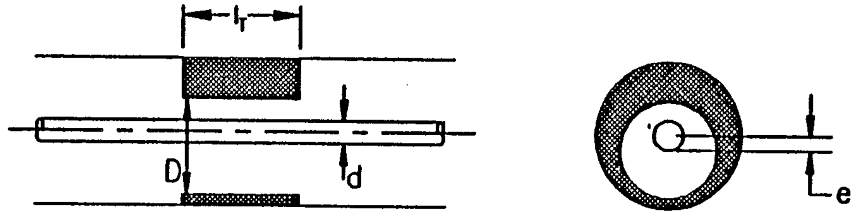
$$Z_T = 60 \ln \frac{D}{d}$$

(a) First Generation Input Transformer Design



$$Z_T = 60 \ln \frac{D}{d}$$

(b) Second Generation Input Transformer Design

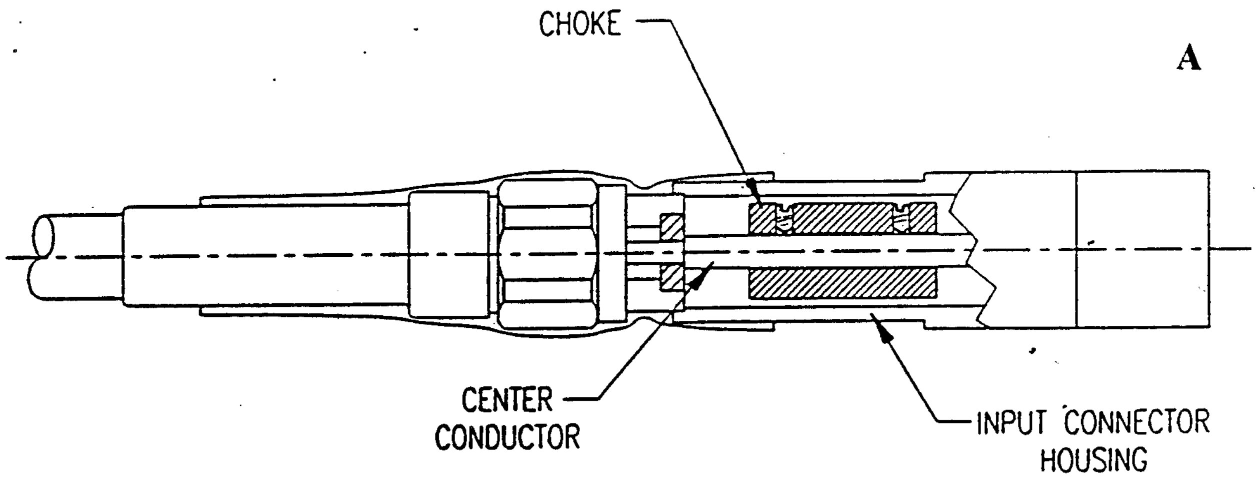


$$Z_T = 60 \cosh^{-1}\left(\frac{D^2 + d^2 - 4e}{2Dd}\right)$$

(c) Third Generation Input Transformer Design

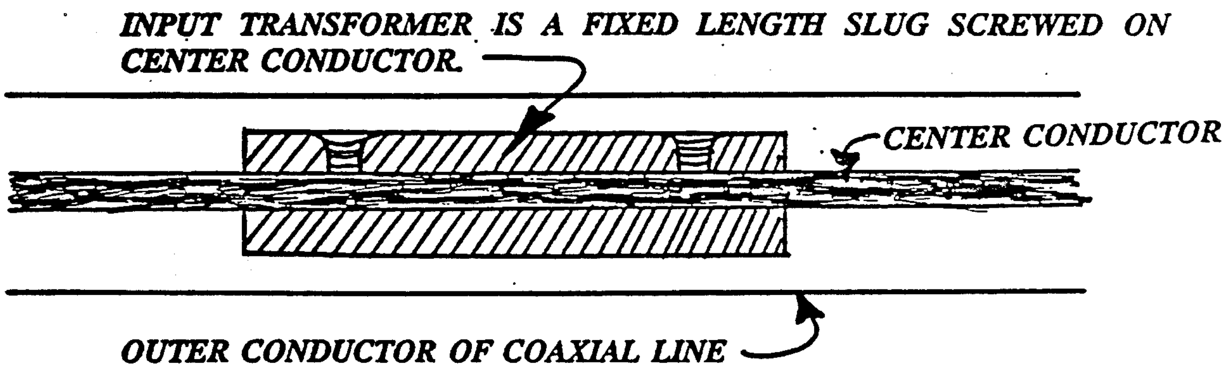
Figure 4.0-2
ADJUSTABLE-CIRCUIT DESIGNS

A



Design Drawing for First Generation Input Transformer

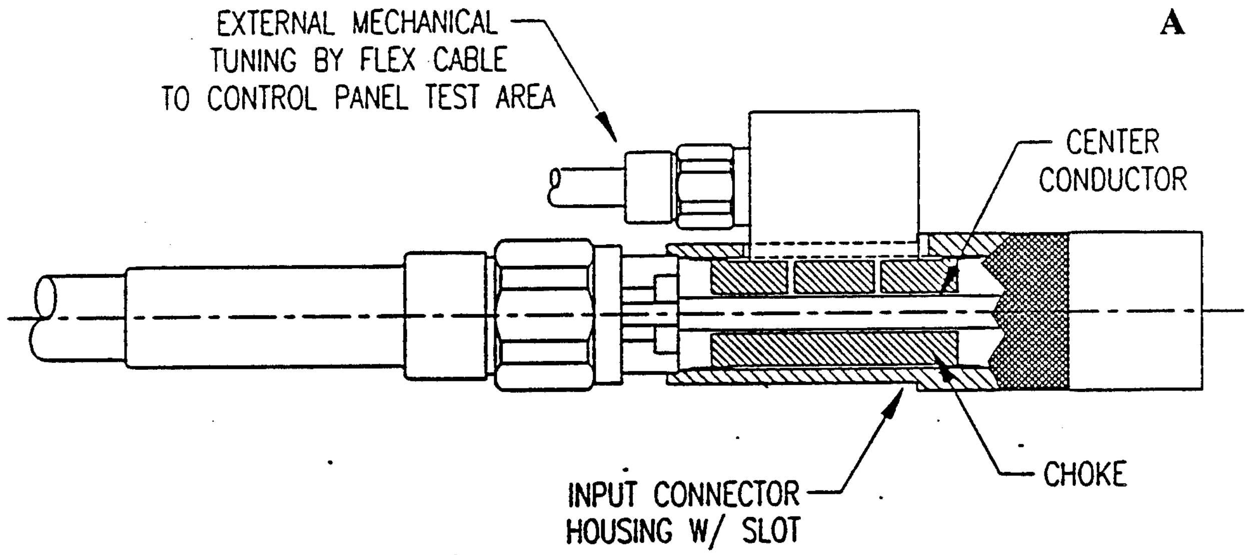
B


INPUT TRANSFORMER IS A FIXED LENGTH SLUG SCREWED ON CENTER CONDUCTOR.
CENTER CONDUCTOR
OUTER CONDUCTOR OF COAXIAL LINE

Detail of Slug for First Generation Input Transformer.

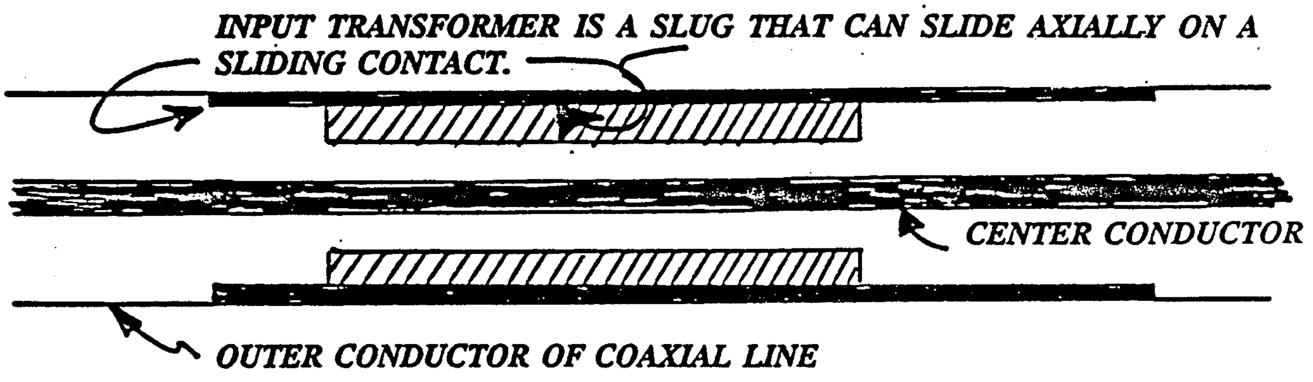
Figure 4.1-1
FIRST-GENERATION DESIGN

A


EXTERNAL MECHANICAL TUNING BY FLEX CABLE TO CONTROL PANEL TEST AREA
CENTER CONDUCTOR
INPUT CONNECTOR HOUSING W/ SLOT
CHOKE

Design Drawing for Second Generation Input Transformer

B


INPUT TRANSFORMER IS A SLUG THAT CAN SLIDE AXIALLY ON A SLIDING CONTACT.
CENTER CONDUCTOR
OUTER CONDUCTOR OF COAXIAL LINE

Detail of Slug for Second Generation Input Transformer.

Figure 4.1-2
SECOND-GENERATION DESIGN

TRANSFORMER DIAMETER...0.470 inches

LOOP POSITION (inches)	0.346 SLUG POS. (inches)	PL -3dB low end (MHz)	PH -3dB high end (MHz)	LOW END PEAK MHz)	HIGH END PEAK (MHz).	FIGURE NO.	CURVE
LP	.346	fo - 45	fo + 41	fo-26.5	fo+16.5	3	(a)
	.361	fo - 41	fo + 40	fo-24.5	-----	3	(b)
	.331	fo - 40	fo + 33.4	-----	fo+13.5	3	(c)
LP -.015	.346	fo - 39.5	fo + 38	fo-20.5	fo+1.5	4	(d)
LP -.030	.346	fo - 34.8	fo + 34.2	fo-11.5	fo+5.5	4	(e)

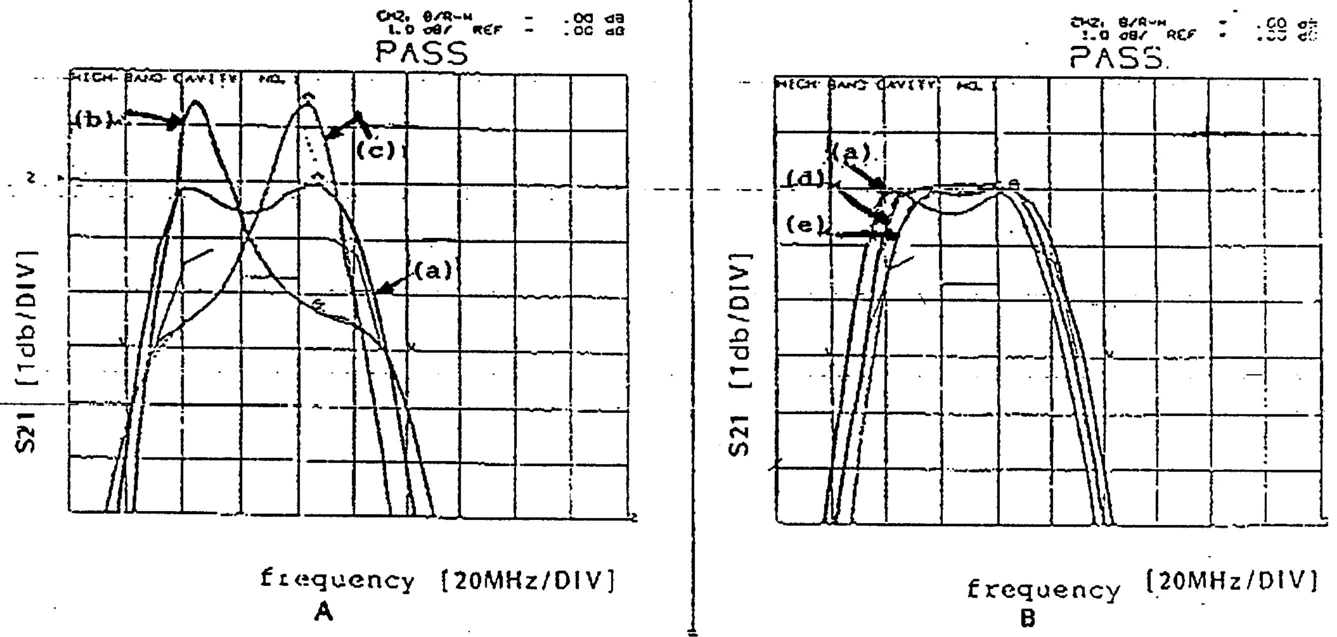
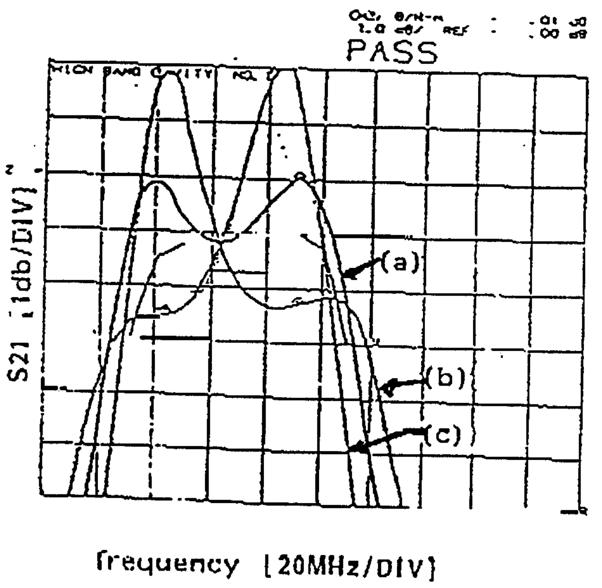


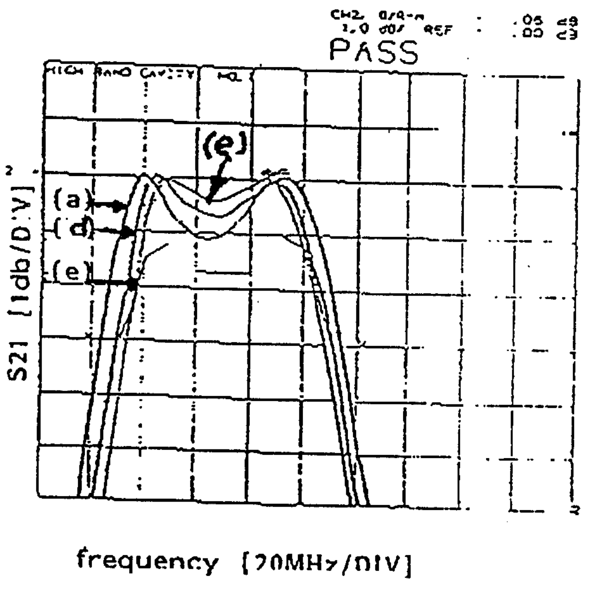
Figure 4.1-3
FIRST-GENERATION TEST RESULTS
Slug O.D. = 0.470 inches

TRANSFORMER DIAMETER...0.480 inches

LOOP POSITION (inches)	0.351 SLUG POS (inches)	PL -3dB low end (MHz)	PH -3dB high end (MHz)	LOW END PEAK (MHz)	HIGH END PEAK (MHz)	FIGURE NO.	CURVE
LP	0.351	fo - 46.5	fo + 42.9	fo-30.5	fo+20.4	5	(a)
	0.366	fo - 41.8	fo + 48	fo-27	-----	5	(b)
	0.336	fo - 45	fo + 34.8	-----	fo+14.4	5	(c)
LP -.015	0.351	fo - 41.2	fo + 40.1	fo-24.5	fo+18.5	6	(d)
LP -.030	0.351	fo - 37	fo + 37.5	fo-21.5	fo+16.5	6	(e)



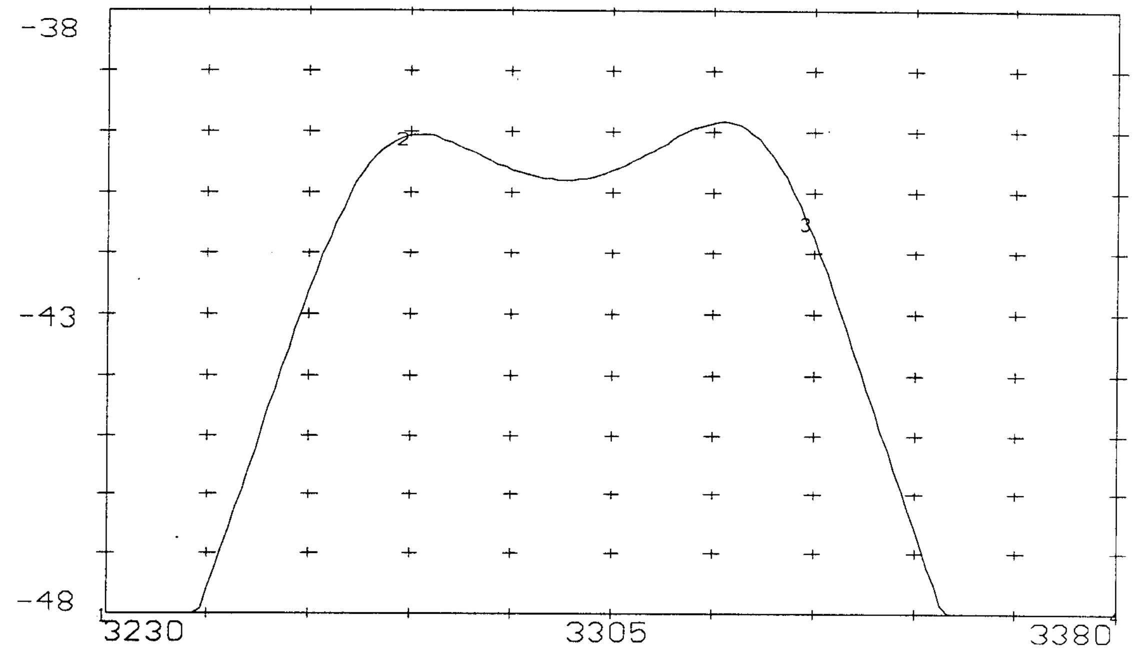
A



B

Figure 4.1-4
FIRST-GENERATION TEST RESULTS
Slug O.D. = 0.480 inches

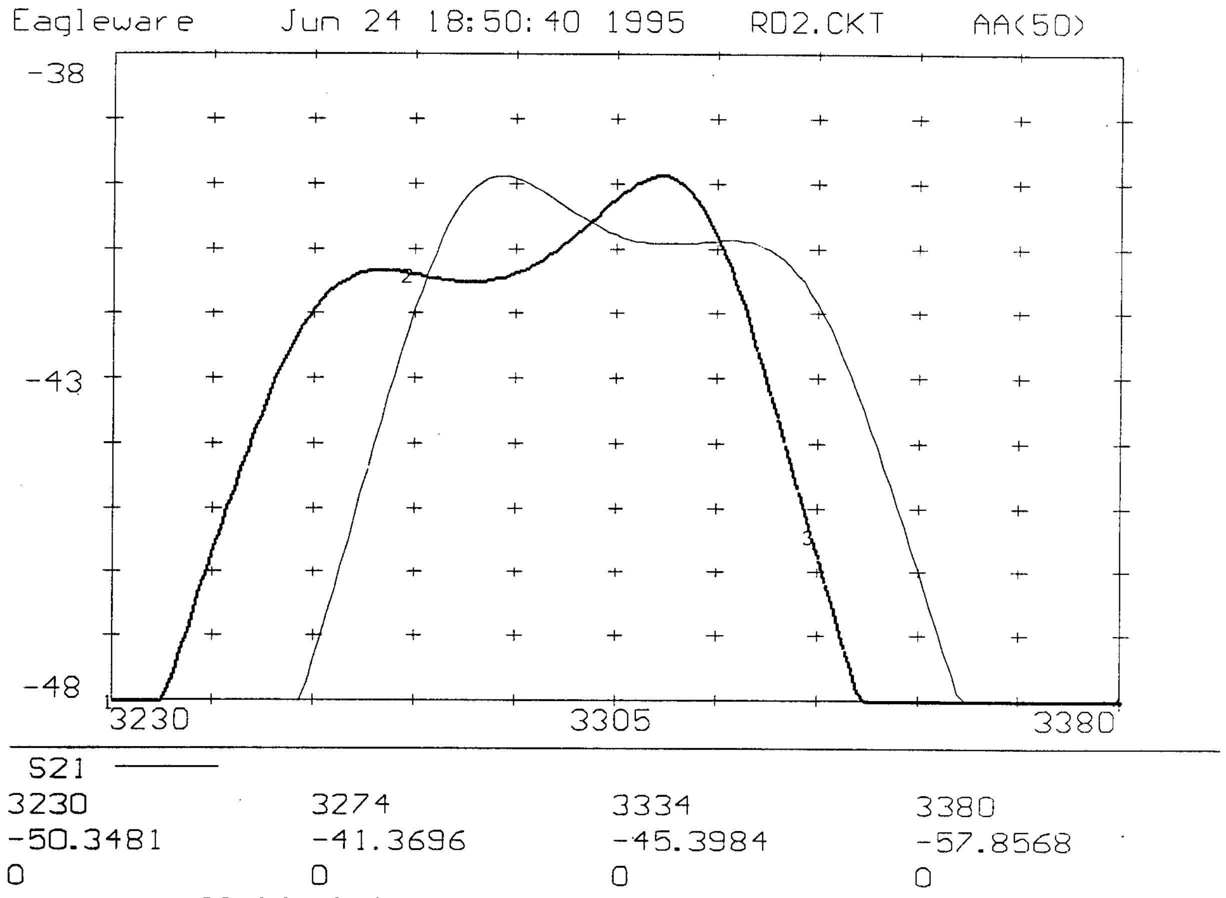
```
Eagleware      Jun 24 18:37:33 1995    RD2.CKT      AA(50)
```



```
  S21 ———
3230            3274            3334            3380
-52.0539        -40.0897        -41.4579        -54.7385
0               0               0               0
```

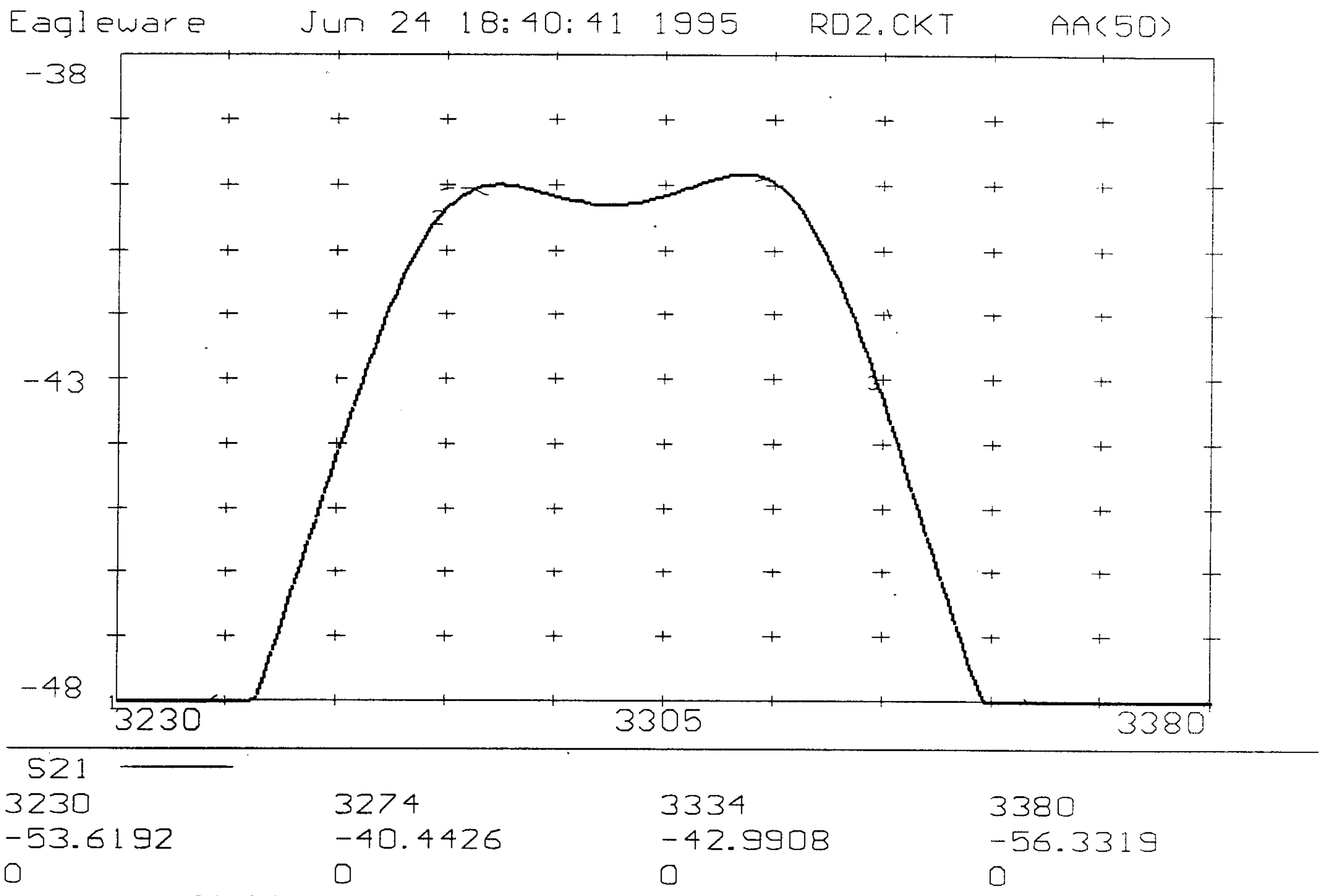
Model calculated response, slug diameter = 0.470 inches

Figure 4.1-5



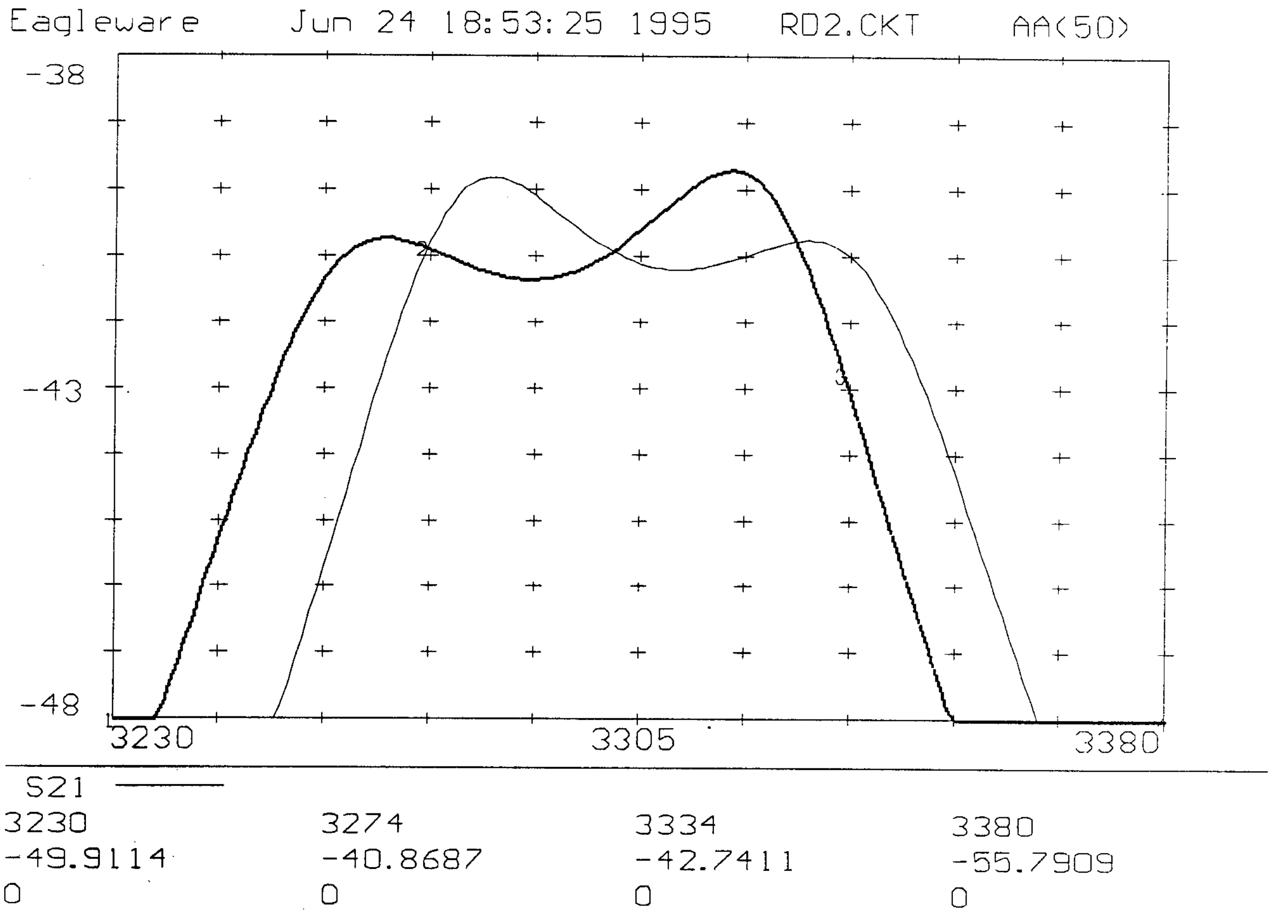
Model calculated response, slug diameter = 0.470 ± 0.015 inches

Figure 4.1-6



Model calculated response, slug diameter = 0.480 inches

Figure 4.1-7



Model calculated response, slug diameter = 0.480 ± 0.015 inches

Figure 4.1-8

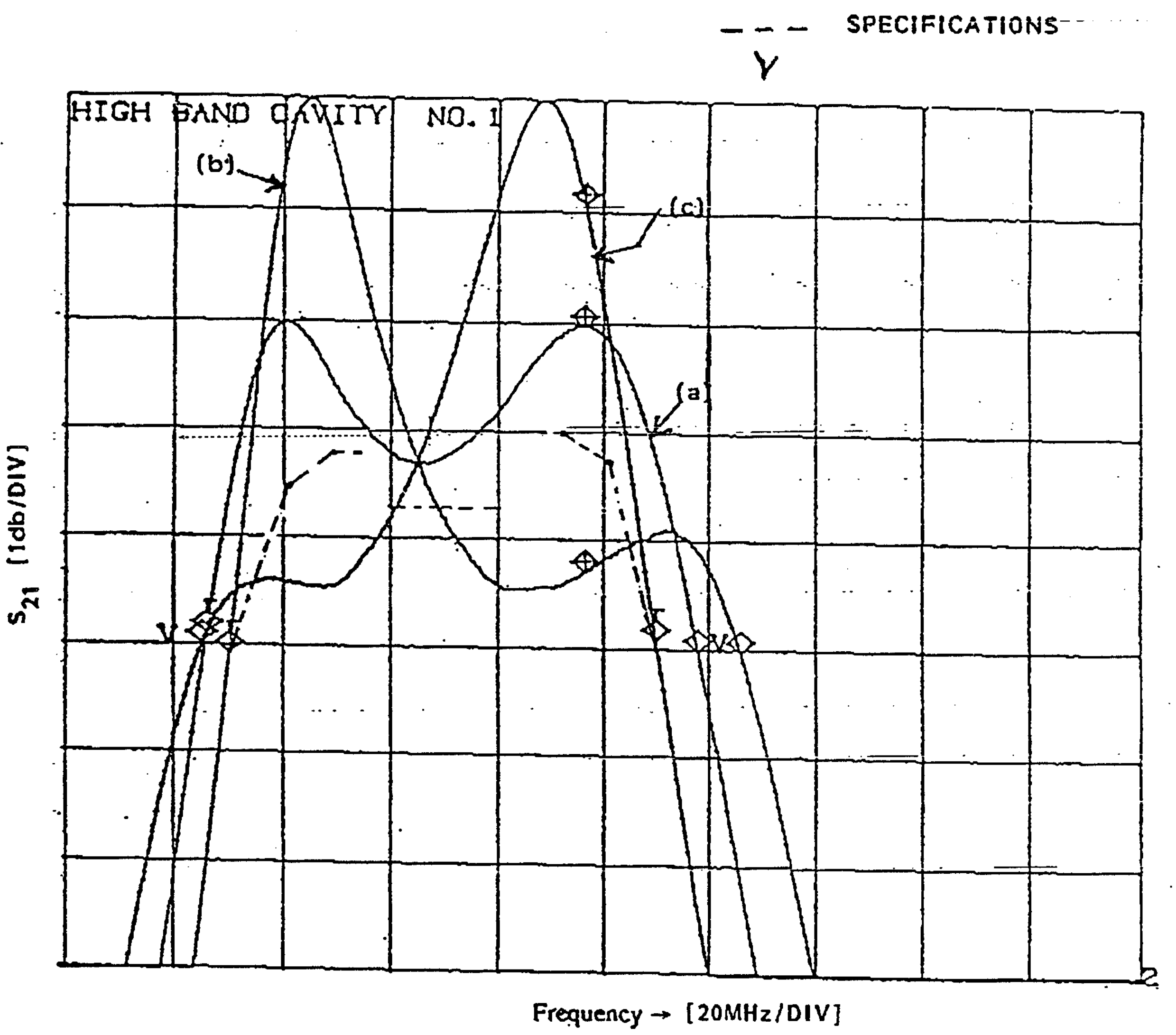
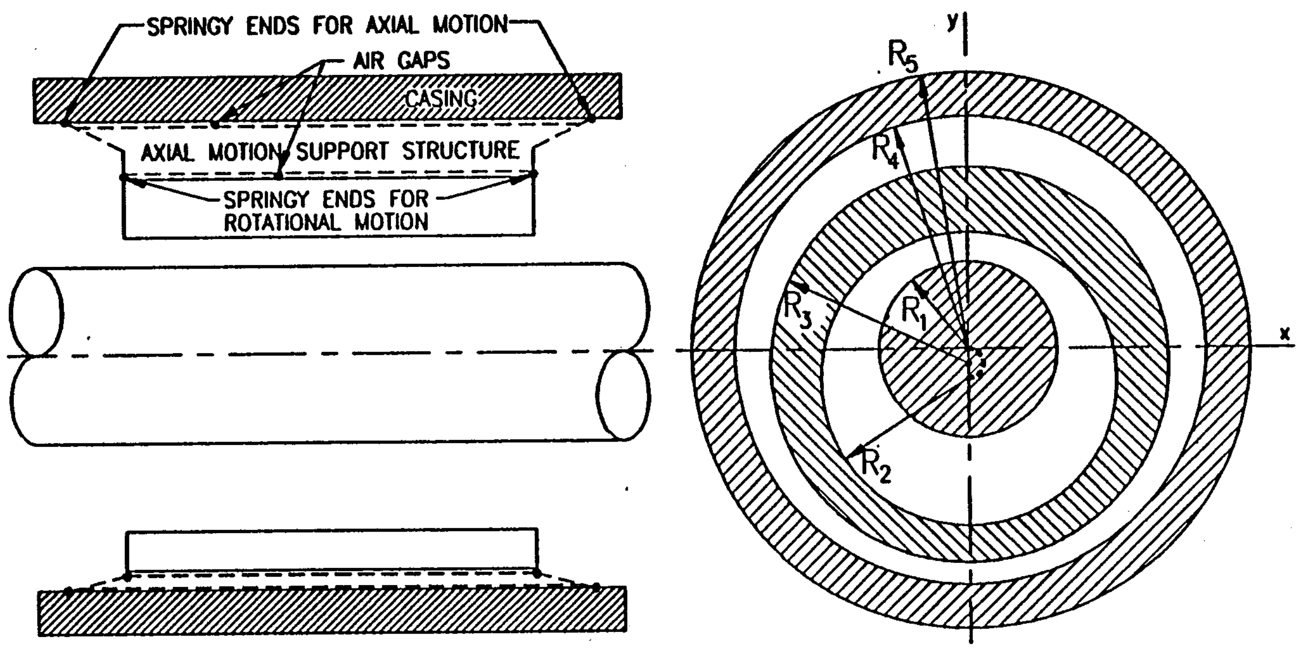


Figure 4.1-9
SECOND-GENERATION TEST RESULTS

A


SPRINGY ENDS FOR AXIAL MOTION
AIR GAPS
CASING
AXIAL MOTION SUPPORT STRUCTURE
SPRINGY ENDS FOR ROTATIONAL MOTION
y
x
R_5
R_4
R_3
R_1
R_2

(a) Constant Θ Crossection (b) Constant Z Crossection

Figure 4.2-1
VARIABLE-ECCENTRICITY COAX LINE

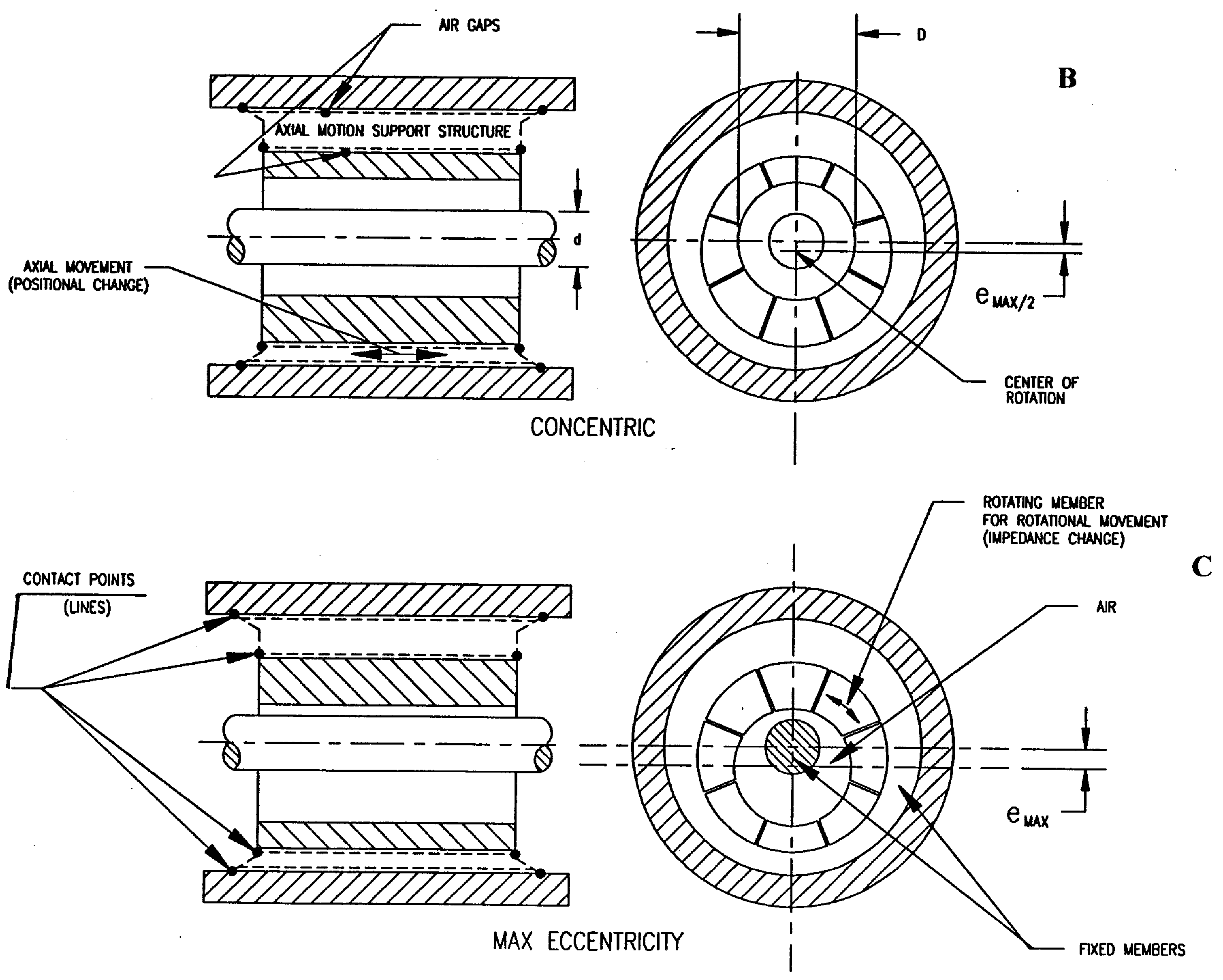

AIR GAPS
AXIAL MOTION SUPPORT STRUCTURE
AXIAL MOVEMENT (POSITIONAL CHANGE)
d
D
B
$e_{MAX/2}$
CENTER OF ROTATION
CONCENTRIC
ROTATING MEMBER FOR ROTATIONAL MOVEMENT (IMPEDANCE CHANGE)
C
AIR
CONTACT POINTS (LINES)
e_{MAX}
MAX ECCENTRICITY
FIXED MEMBERS

Figure 4.2-1 (CON'T)
VARIABLE-ECCENTRICITY COAX LINE

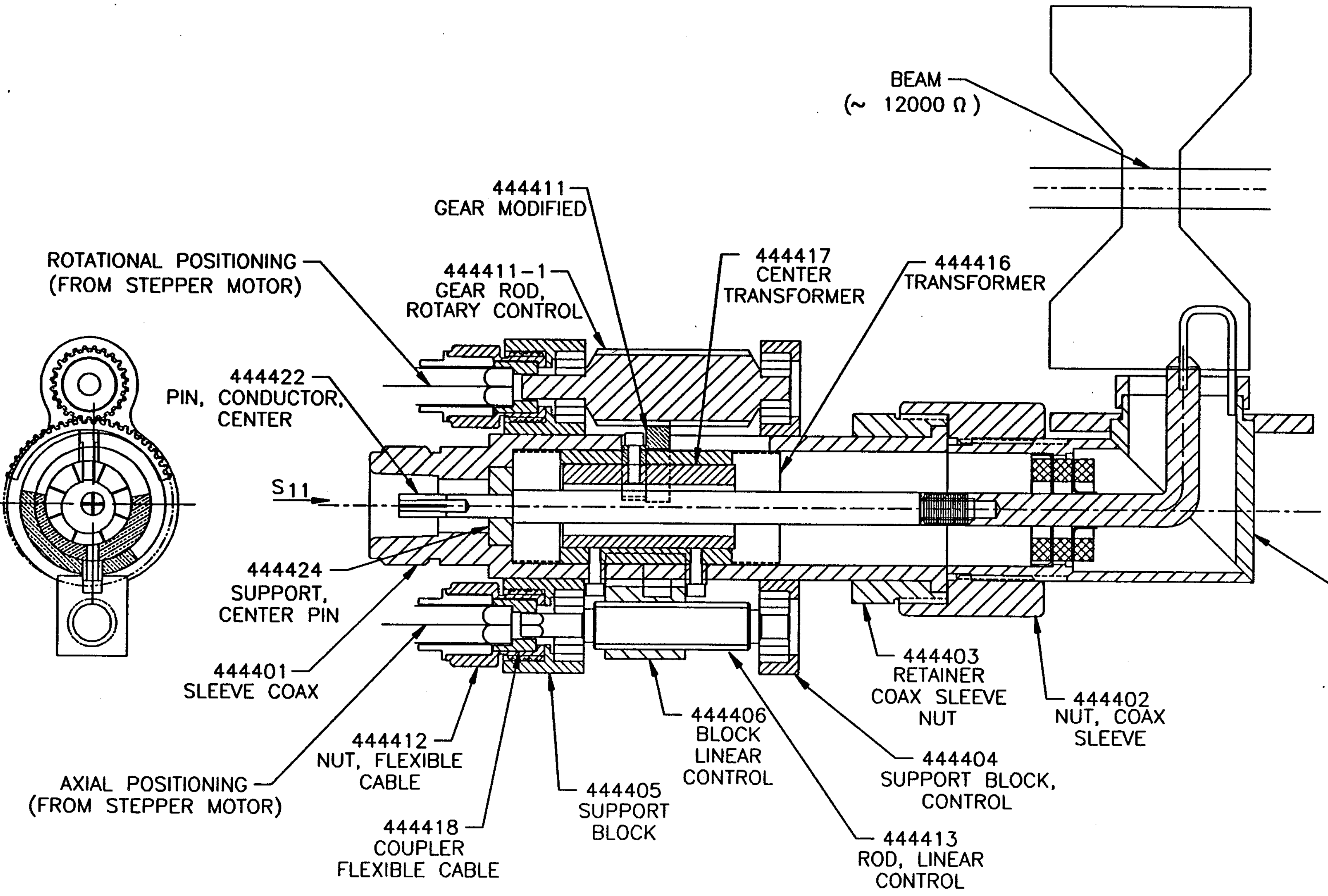


Figure 4.2-2
CROSS-SECTION OF TRANSFORMER AND COUPLING LOOP

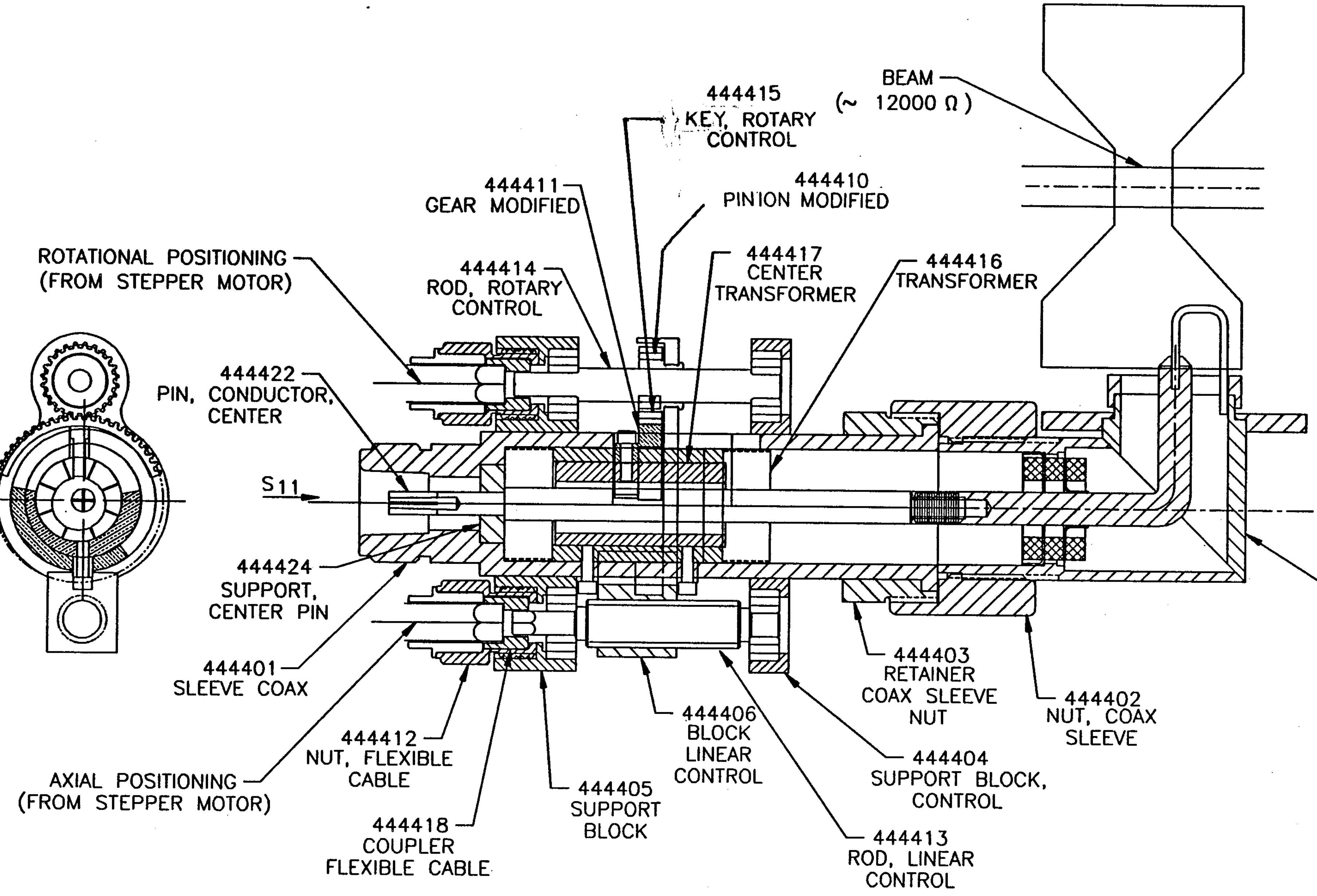

BEAM (~ 12000 Ω)
444415 KEY, ROTARY CONTROL
444411 GEAR MODIFIED
444410 PINION MODIFIED
ROTATIONAL POSITIONING (FROM STEPPER MOTOR)
444414 ROD, ROTARY CONTROL
444417 CENTER TRANSFORMER
444416 TRANSFORMER
444422 PIN, CONDUCTOR, CENTER
S11
444424 SUPPORT, CENTER PIN
444401 SLEEVE COAX
372386 INPUT COAX ASSY
444403 RETAINER COAX SLEEVE NUT
444402 NUT, COAX SLEEVE
444412 NUT, FLEXIBLE CABLE
444406 BLOCK LINEAR CONTROL
444404 SUPPORT BLOCK, CONTROL
AXIAL POSITIONING (FROM STEPPER MOTOR)
444405 SUPPORT BLOCK
444418 COUPLER FLEXIBLE CABLE
444413 ROD, LINEAR CONTROL

Figure 4.2-3
CROSS-SECTION OF TRANSFORMER AND COUPLING LOOP
(Before cost-saving modification)

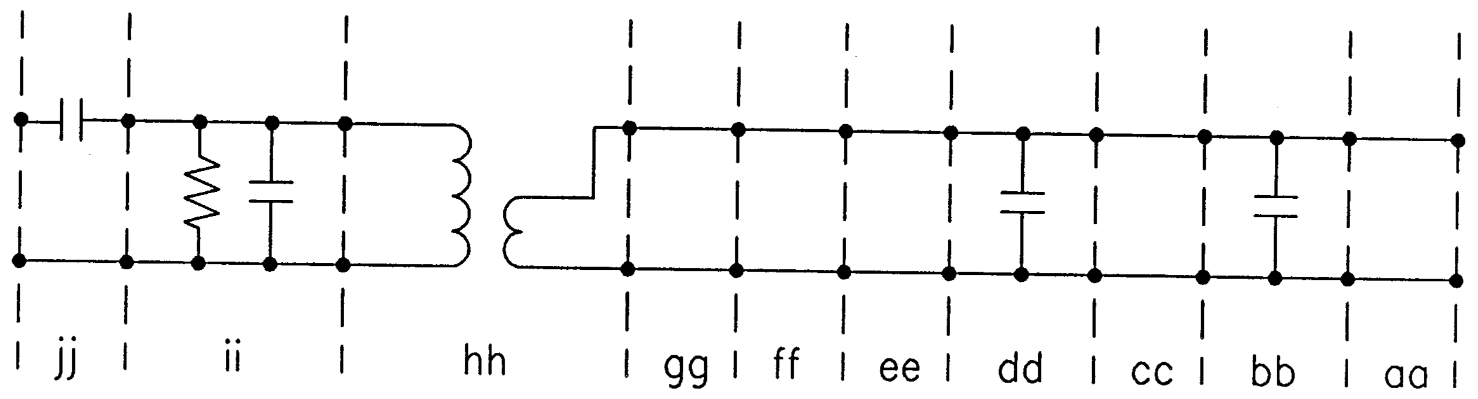
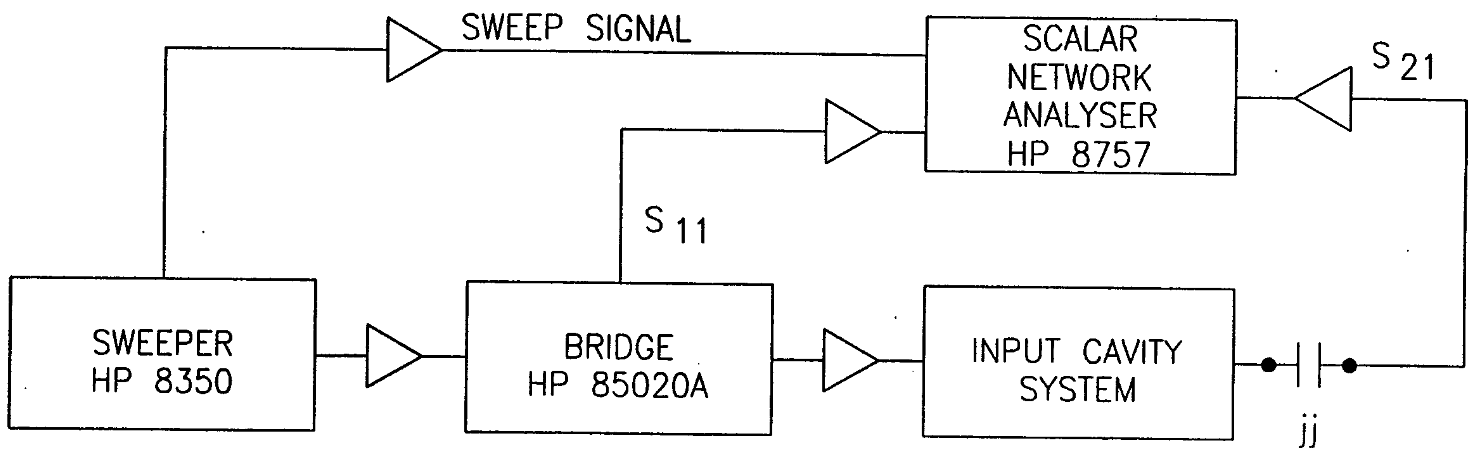


Figure 4.3-1
DIAGRAM OF CIRCUIT IN FILE RD3.CKT

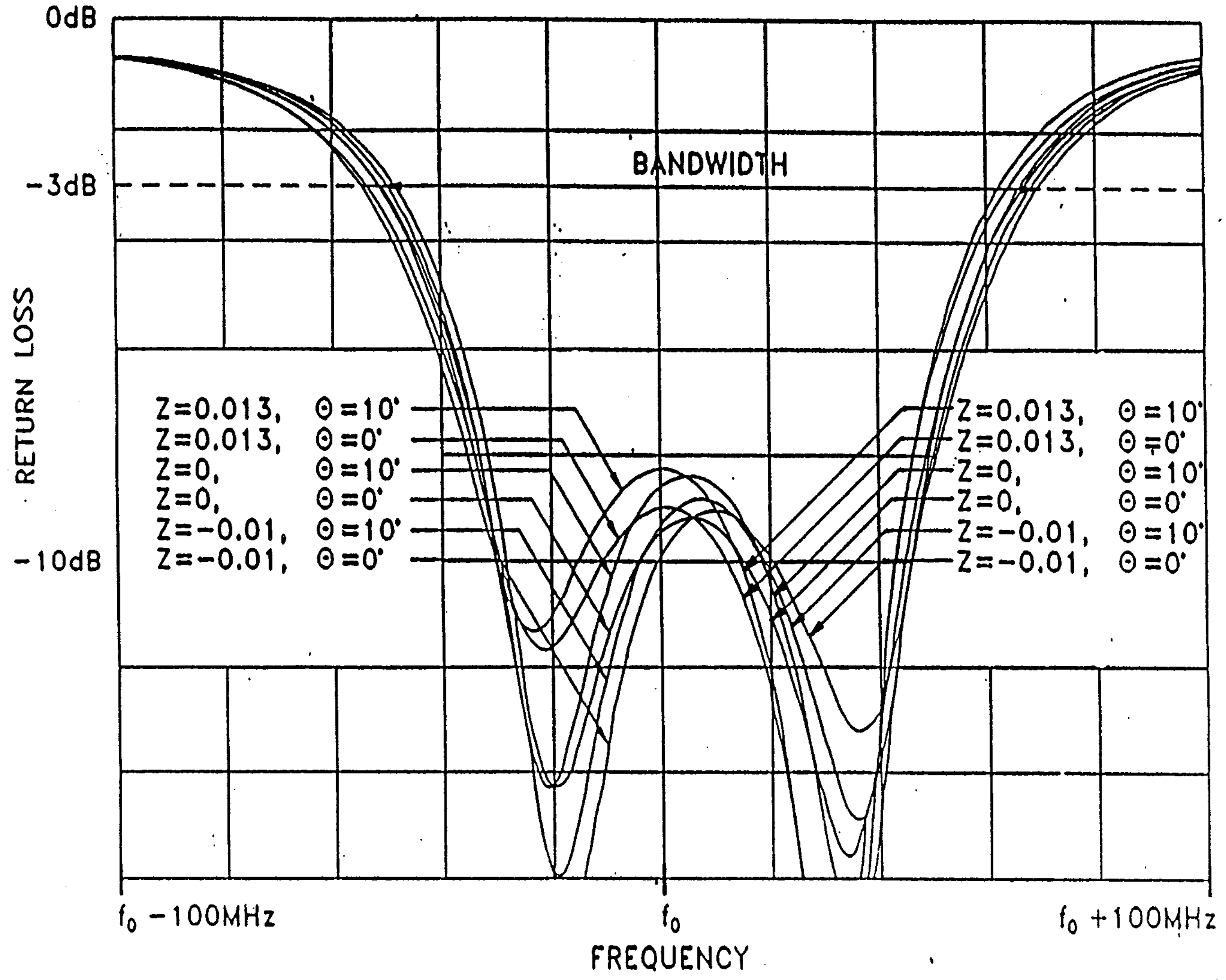
```
CIRCUIT OLD
trl aa ke  75  a  1
cap bb pa ?1
trl cc ke z 22.5 1
equ dd bb
trl ee ke  75.60698 b 1
trl ff ke ?50 ?19 1
trl gg ke  75 20 1
mui hh ** ?2.222553 4.936216 ?0.06302119
prc ii **  18256.68 ?0.4711922
cap jj ** ?0.0006346164
con hh ** 1 0 0 3
con ii ** 0 3
con jj ** 3 2
nod hh **
cax aa hh
WINDOW
aa(50)
gph s21 -48 -38
gph s11 -20 0
freq
swd 3230 3430 1
equation
a=?8.180956
b= 45.93252
l=a+b
l=53/l
a=a*l
b=b*l
d= 0.480
z=60*LN(0.6/d)
opt
3254 3256 s21=-43
3279 3281 s21=-40.2
3324 3326 s21=-40
3344 3346 s21=-43
```

Figure 4.3-2
FILE RD3.CKT



COLD TEST MEASUREMENT ASSEMBLY FOR TUNING THE INPUT TRANSFORMER IN THE INPUT CAVITY SYSTEM.

Figure 4.3-3
COLD-TEST MEASUREMENT



Return Loss Bandwidth Shape as a Function of Frequency and Input Transformer Position Parameters Z and Θ.

Figure 4.3-4
THIRD-GENERATION TEST RESULTS

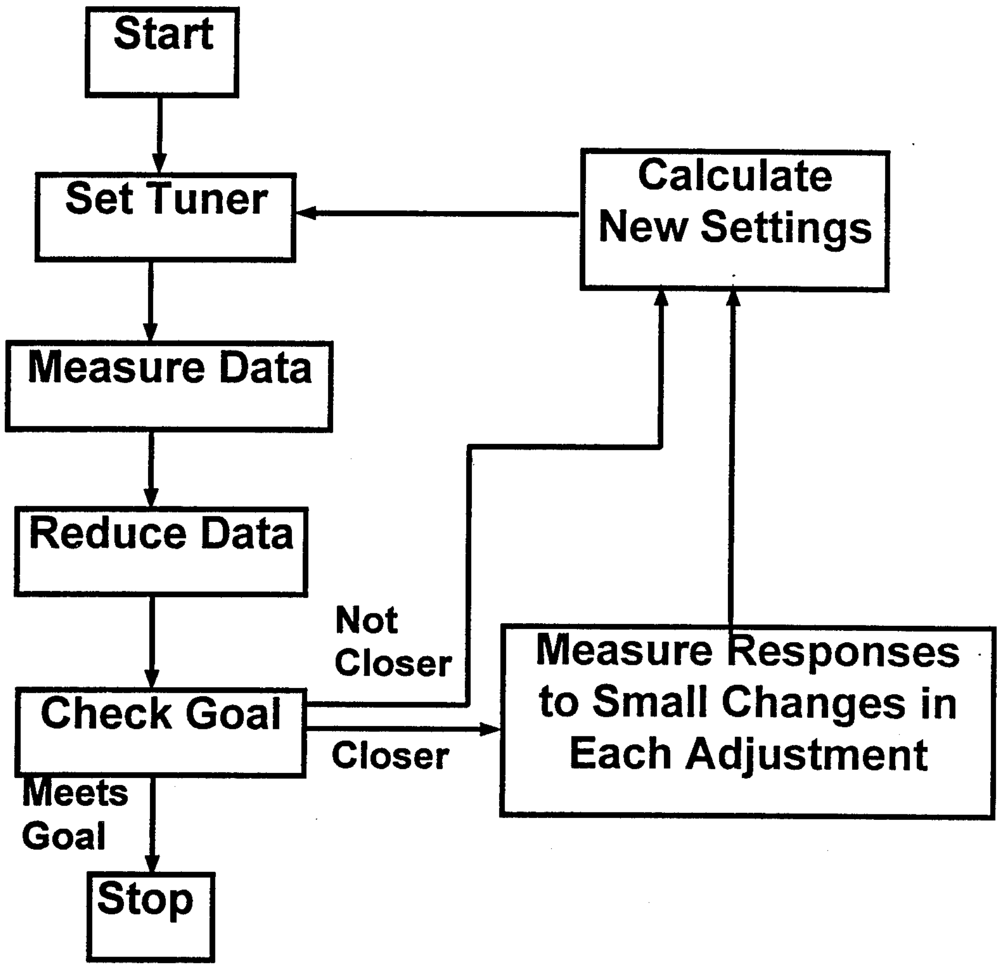


Figure 4.4-1
EXPERT-SYSTEM PROGRAM FLOW CHART

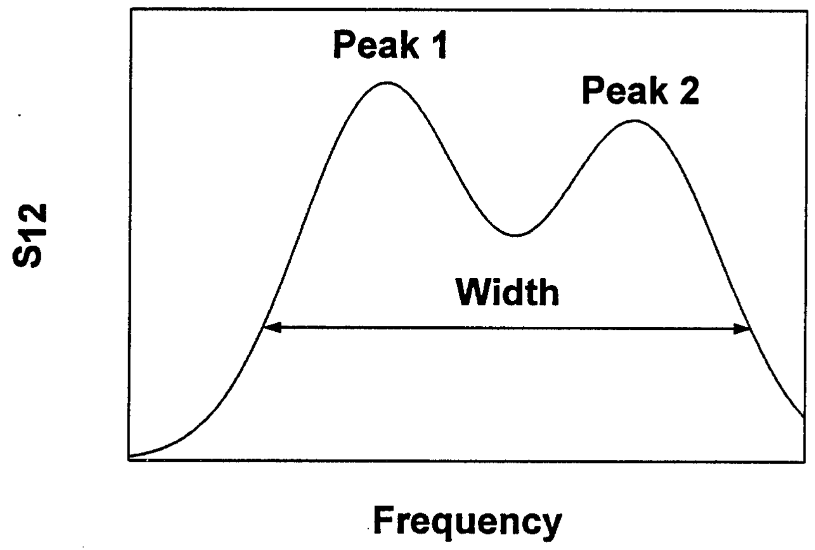


Figure 4.4-2
S_{12} CURVE ANALYZED BY TIPTOE1A

A

```
D:\EAGLE\TIPTOE>tiptoe1a
Please enter Width Criterion:
90

**************************************************
 sParameter1 = 301.02     sParameter2 = 2.1464
**************************************************
 Itr  SI   Amp1    Amp2   Ratio   Width  Ratio
   0   0  -39.33  -39.80   .988  100.88  1.121
**************************************************

Updated .CKT file with:   S1 =>   304.0252   S2 =>    2.1464

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   301.0150   S2 =>    2.1679

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   285.3625   S2 =>    2.1470

Press <return> after new signal has been generated by Eagle Software ...

**************************************************
 sParameter1 = 285.36     sParameter2 = 2.1470
**************************************************
 Itr  SI   Amp1    Amp2   Ratio   Width  Ratio
   1   1  -40.40  -39.04  1.035  102.45  1.138
**************************************************

Convergence Criteria Failure:
 AmpRatio =     1.0348
 Old Distance =     .1215
 New Distance =      .1427

MaxStepSize =>  2

Updated .CKT file with:   S1 =>   293.1888   S2 =>    2.1467

Press <return> after new signal has been generated by Eagle Software ...
```

Figure 4.4-3
TIPTOE1A PRINTOUT

```
************************************************
 sParameter1 = 293.19     sParameter2 = 2.1467
************************************************
 Itr  Sl   Amp1    Amp2   Ratio   Width  Ratio
   1   2  -39.79  -39.37  1.011   99.71  1.108
************************************************

Finished with  1 iterations.
Hit <return> to continue ...


Updated .CKT file with:   S1 =>   296.1206   S2 =>     2.1467

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:   S1 =>   293.1888   S2 =>     2.1682

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:   S1 =>   293.0186   S2 =>     1.7722

Press <return> after new signal has been generated by Eagle Software ...


************************************************
 sParameter1 = 293.02     sParameter2 = 1.7722
************************************************
 Itr  Sl   Amp1    Amp2   Ratio   Width  Ratio
   2   1  -39.88  -39.30   .000  100.60   .000
************************************************

Convergence Criteria Failure:
  AmpRatio =       .0000
  Old Distance =     .1084
  New Distance =    1.4142

MaxStepSize =>   2


Updated .CKT file with:   S1 =>   293.1037   S2 =>     1.9595

Press <return> after new signal has been generated by Eagle Software ...
```

Figure 4.4-3 (CON'T)
TIPTOE1A PRINTOUT

C

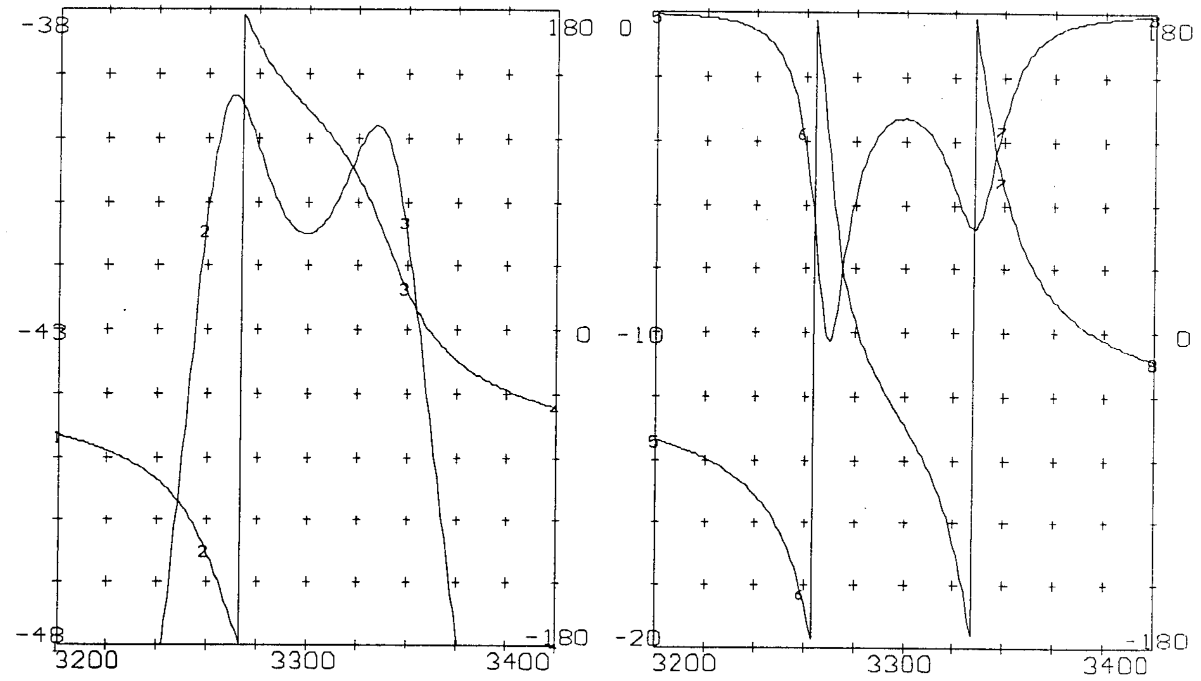
```
************************************************
 sParameter1 = 293.10     sParameter2 = 1.9595
************************************************
 Itr  Sl  Amp1    Amp2   Ratio   Width  Ratio
   2   2 -39.15 -40.12  .976   97.74  1.086
************************************************

Finished with  2 iterations.
Hit <return> to continue ...

Updated .CKT file with:   S1 =>   296.0347   S2 =>     1.9595

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   293.1037   S2 =>     1.9791

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   279.8101   S2 =>     1.8817

Press <return> after new signal has been generated by Eagle Software ...

************************************************
 sParameter1 = 279.81     sParameter2 = 1.8817
************************************************
 Itr  Sl  Amp1    Amp2   Ratio   Width  Ratio
   3   1 -39.65 -39.49 1.004   91.67  1.019
************************************************

Finished with  3 iterations.
Hit <return> to continue ...

Updated .CKT file with:   S1 =>   282.6082   S2 =>     1.8817

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   279.8101   S2 =>     1.9005

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:   S1 =>   279.0865   S2 =>     1.8463

Press <return> after new signal has been generated by Eagle Software ...
```

Figure 4.4-3 (CON'T)
TIPTOE1A PRINTOUT

```
**************************************************
 sParameter1 = 279.09     sParameter2 = 1.8463
**************************************************
 Itr  SI   Amp1    Amp2   Ratio    Width  Ratio
   4   1  -39.56  -39.56  1.000    90.32  1.004
**************************************************

Finished with  4 iterations.
Hit <return> to continue ...


Updated .CKT file with:   S1 =>   281.8774   S2 =>     1.8463

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:   S1 =>   279.0865   S2 =>     1.8647

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:   S1 =>   278.8526   S2 =>     1.8416

Press <return> after new signal has been generated by Eagle Software ...


**************************************************
 sParameter1 = 278.85     sParameter2 = 1.8416
**************************************************
 Itr  SI   Amp1    Amp2   Ratio    Width  Ratio
   5   1  -39.56  -39.56  1.000    90.17  1.002
**************************************************

Finished with  5 iterations.
Hit <return> to continue ...
```

Figure 4.4-3 (CON'T)
TIPTOE1A PRINTOUT

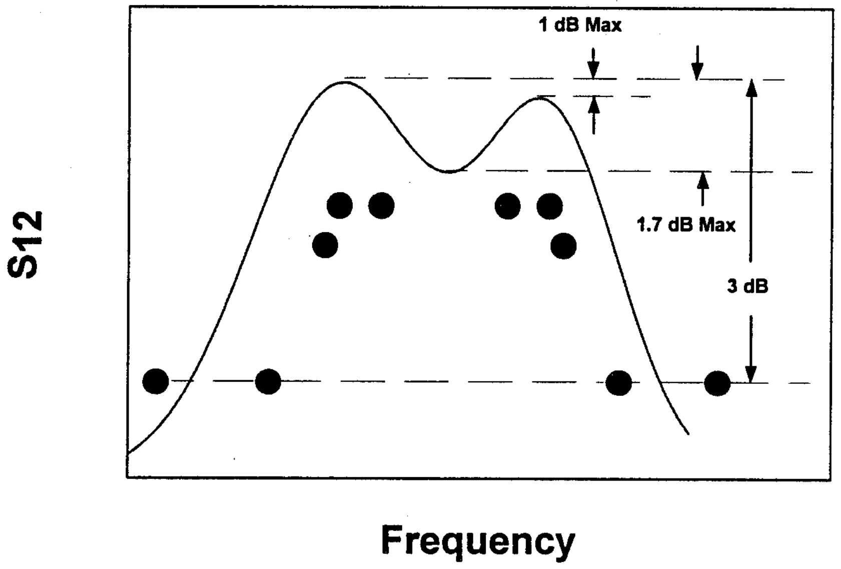
```
Eagleware     Jun 15 13:16:36 1995     START.CKT     INPUT_COUPL(50)
```



```
S21 ———            P21 ———             S11 ———            P11 ———
3200      3259      3339      3400      3200      3259      3339      3400
-59.2189  -41.4372  -41.2947  -56.685   -.057342  -3.67688  -3.60746  -.082784
-59.3195  -124.563  24.7555   -43.4939  -60.552   -147.957  86.2156   -16.2815
```

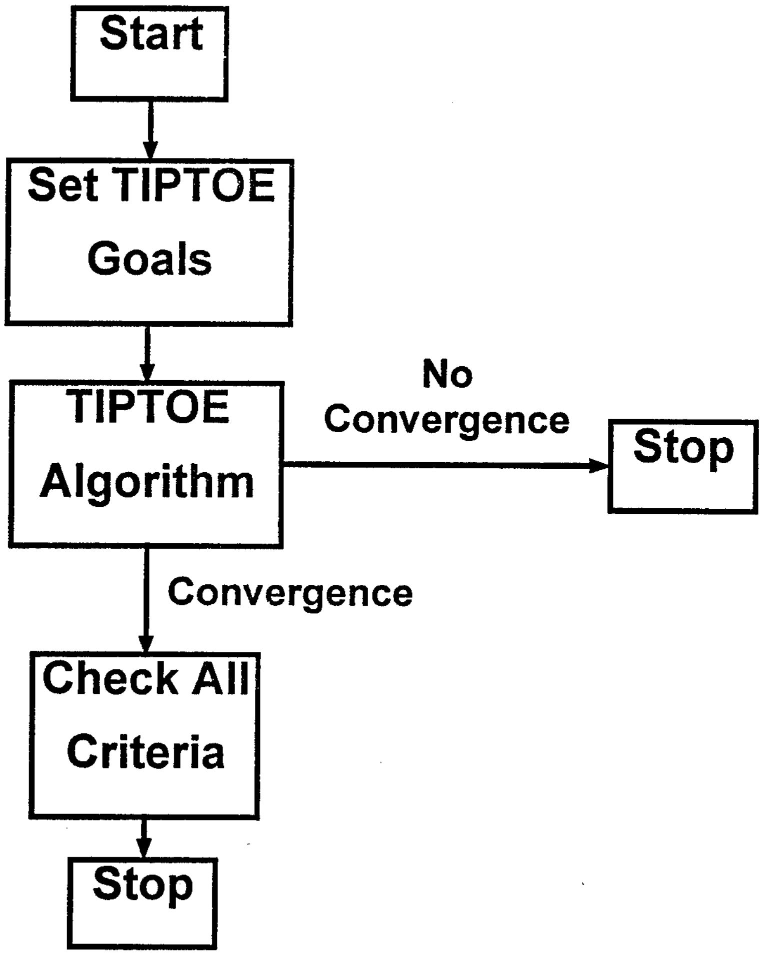
The response of the START.CKT file calculated by the SUPERSTAR program.

Figure 4.4-4
START.CKT RESPONSE



Sample circuit response showing acceptance criteria.

Figure 4.4-5
SAMPLE CIRCUIT RESPONSE



Flow chart for the TIPTOE2A (DEMO2A) program.

Figure 4.4-6
TIPTOE2A FLOW CHART

A

```
D:\EAGLE\TIPTOE2>demo2a

**************************************************
 sParameter1 = 293.84     sParameter2 = 2.0995
**************************************************
 Itr  Sl   Amp1    Amp2   Ratio    Width  Ratio
   0   0  -39.33  -39.80  1.012     1.94  1.146
**************************************************

Updated .CKT file with:  S1 =>  296.7774  S2 =>    2.0995

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:  S1 =>  293.8390  S2 =>    2.1205

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:  S1 =>  297.4802  S2 =>    2.0997

Press <return> after new signal has been generated by Eagle Software ...

**************************************************
 sParameter1 = 297.48     sParameter2 = 2.0997
**************************************************
 Itr  Sl   Amp1    Amp2   Ratio    Width  Ratio
   1   1  -39.35  -39.78  1.011     1.77  1.046
**************************************************

Tiptoe finished with  1 iterations.
Hit <return> to continue ...

Updated .CKT file with:  S1 =>  300.4550  S2 =>    2.0997

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:  S1 =>  297.4802  S2 =>    2.1207

Press <return> after new signal has been generated by Eagle Software ...

Updated .CKT file with:  S1 =>  294.2722  S2 =>    2.1078

Press <return> after new signal has been generated by Eagle Software ...
```

Figure 4.4-7
TIPTOE2A (DEMO2A) PRINTOUT

```
***************************************************
 sParameter1 = 294.27     sParameter2 = 2.1078
***************************************************
 Itr  SI   Amp1    Amp2   Ratio    Width  Ratio
   2   1  -39.57  -39.56  1.000    1.72  1.016
***************************************************


Tiptoe finished with  2 iterations.
Hit <return> to continue ...


Updated .CKT file with:  S1 =>  297.2149  S2 =>    2.1078

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:  S1 =>  294.2722  S2 =>    2.1289

Press <return> after new signal has been generated by Eagle Software ...


Updated .CKT file with:  S1 =>  293.8499  S2 =>    2.1000

Press <return> after new signal has been generated by Eagle Software ...


***************************************************
 sParameter1 = 293.85     sParameter2 = 2.1000
***************************************************
 Itr  SI   Amp1    Amp2   Ratio    Width  Ratio
   3   1  -39.56  -39.56  1.000    1.70  1.002
***************************************************


Tiptoe finished with  3 iterations.
Hit <return> to continue ...

Tiptoe converged for wide configuration ...
  SP1=  293.8499 SP2=    2.1000

Signal Passed Amplitude Checks ...
Signal Passed Outer Bounds Checks ...
Signal Passed Interior Bounds Checks ...

Signal passed all checks!

Demo2a ... Normal Completion


D:\EAGLE\TIPTOE2>
```

Figure 4.4-7 (CON'T)
TIPTOE2A (DEMO2A) PRINTOUT

APPENDIX 4A
TIPTOE1A Source Code Listing

```
      program tiptoe1a
c
c*********************************************************************
c
c ... Program to calculate S-parameters utilizing Martin Lee's famous
c     tiptoe algorithm.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Subroutines called:
c
c       ratios
c       sParamIO
c
c ... Comments:
c
c*********************************************************************
c ... Parameters:
c
       parameter       (WIDTH_TOLORANCE=135.)
       parameter       (MAX_ITER=20)
c
c ... Variables:
c
       character*1     adum
       character*128   sName, oName
       logical         rescale
       integer*4       loName
       integer*4       numIter, subIter
       integer*4       maxStepSize
       integer*4       read_mode, write_mode
       real*4          sParm1Old, sParm1New, dSParm1
       real*4          sParm2Old, sParm2New, dSParm2
       real*4          amp1, amp2, width
       real*4          ampRatioOld, ampRatioNew, dAmpRatio
       real*4          widthRatioOld, widthRatioNew, dWidthRatio
       real*4          dAdS1, dAdS2, dWdS1, dWdS2
       real*4          deltaAmp, deltaWidth, deltaSP1, deltaSP2
       real*4          detInv, distOld, distNew
       real*4          width_param
c
c ... Data:
```

```
c
c ...   Define name of file containing signal amplitudes
c
        data sName(1:) /'signal.out'/
c
c ...   Define name of diagnostic output file
c
        data oName(1:) /'tiptoe.out'/
c
c ...   Define read/write modes
c
        data read_mode  /0/
        data write_mode /1/
c
c***********************************************************************
c
c ... Initialize counters
c
      maxStepSize = 1
      numIter = 0
      subIter = 0
      rescale = .false.
c
c ... Ask user for desired signal width
c
      write(*, '("$Please enter Width Criterion: ")')
      read(*, *) width_param
c
c ... Open file for diagnostic output
c
      loName = length(oName)
      open(6, file=oName(1:loName), status='unknown', form='formatted',
     >     err=10)
c
c ... Read in current values for S parameters
c
      call signalGen(read_mode, sParm1New, sParm2New)
c
c ... Calculate Amplitude and Width Ratios
c
      call ratios(sName, width_param,
     >            ampRatioNew, widthRatioNew, amp1, amp2, width)
c
c ... Output amplitude and width ratio values
c
      call print_params(0, sParm1New, sParm2New, numIter, subIter,
    >                   ampl, amp2, ampRatioNew, width, widthRatioNew)
     call print_params(6, sParm1New, sParm2New, numIter, subIter,
    >                   ampl, amp2, ampRatioNew, width, widthRatioNew)
c
c ... Start of iterative loop
c
     do while (numIter .lt. MAX_ITER)
c
c ...   Increment iteration counter
c
       numIter = numIter + 1
       subIter = 0
c
c ...   Store current S parameter values as "previous" values
c
       sParm1Old = sParm1New
       sParm2Old = sParm2New
c
c ...   Store current Amplitude and Width ratios as "previous" values
c
       ampRatioOld   = ampRatioNew
       widthRatioOld = widthRatioNew
c
c ...   Perturb current values of S parameters for next signal
c
       dSParm1   = 0.01*sParm1Old
       dSParm2   = 0.01*sParm2Old
c
       sParm1New = sParm1Old + dSParm1
       sParm2New = sParm2Old + dSParm2
c
c ...   Generate new signal
c
       call signalGen(write_mode, sParm1New, sParm2Old)
c
c ...   Calculate Amplitude and Width Ratios for dS1 signal vector
c
       call ratios(sName, width_param, ampRatioNew, widthRatioNew,
    >              ampl, amp2, width)
       write(6, *)
       write(6, '(" ******************************************")')
       write(6, '(" sParm1Prime = ", f6.2,
    >            " sParameter2 = ", f6.4)') sParm1New, sParm2Old
       write(6, '(" ******************************************")')
       write(6, '(" Iter   Ampl    Amp2   Ratio    Width  Ratio")')
       write(6, '(i4, 1x, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >   numIter, ampl, amp2, ampRatioNew, width, widthRatioNew
      write(6, '(" ********************************************")')
c
c ...  Calculate dS1 partial derivatives
c
      dAmpRatio   = ampRatioNew - ampRatioOld
      dWidthRatio = widthRatioNew - widthRatioOld
c
      dAdS1 = dAmpRatio/dSParm1
      dWdS1 = dWidthRatio/dSParm1
c
c ...  Generate new signal
c
      call signalGen(write_mode, sParm1Old, sParm2New)
c
c ...  Calculate Amplitude and Width Ratios for dS2 signal vector
c
      call ratios(sName, width_param, ampRatioNew, widthRatioNew,
     >            ampl, amp2, width)
      write(6, *)
      write(6, '(" ********************************************")')
      write(6, '(" sParameter1 = ", f6.2,
     >           " sParm2Prime = ", f6.4)') sParm1Old, sParm2New
      write(6, '(" ********************************************")')
      write(6, '(" Iter   Ampl    Amp2   Ratio    Width  Ratio")')
      write(6, '(i4, 1x, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >   numIter, ampl, amp2, ampRatioNew, width, widthRatioNew
      write(6, '(" ********************************************")')
c
c ...  Calculate dS2 partial derivatives
c
      dAmpRatio   = ampRatioNew - ampRatioOld
      dWidthRatio = widthRatioNew - widthRatioOld
c
      dAdS2 = dAmpRatio/dSParm2
      dWdS2 = dWidthRatio/dSParm2
c
      write(6, *)
      write(6, *) ' Iteration #: ', numIter
      write(6, *) '       dAdS1: ', dAdS1
      write(6, *) '       dWdS1: ', dWdS1
      write(6, *) '       dAdS2: ', dAdS2
      write(6, *) '       dWdS2: ', dWdS2
      write(6, *)
c
c ...  Calculate reciprical determinant of partial derivative matrix
```

```
c
        detInv = 1.0/(dAdS1*dWdS2 - dAdS2*dWdS1)
c
  100   subIter = subIter + 1
c
c ...  Estimate needed change in amplitude and width ratios
c
        deltaAmp   = (1.0 -   ampRatioOld)/float(maxStepSize)
        deltaWidth = (1.0 - widthRatioOld)/float(maxStepSize)
c
c ...  Calculate corresponding change in S parameter values
c
        deltaSP1 = detInv*(dWdS2*deltaAmp - dAdS2*deltaWidth)
        deltaSP2 = detInv*(dAdS1*deltaWidth - dWdS1*deltaAmp)
c
c ...  Calculate new S parmeter values
c
        sParm1New = sParm1Old + deltaSP1
        sParm2New = sParm2Old + deltaSP2
c
c ...  Generate new signal
c
        call signalGen(write_mode, sParm1New, sParm2New)
c
c ...  Calculate Amplitude and Width Ratios
c
        call ratios(sName, width_param,
     >              ampRatioNew, widthRatioNew, amp1, amp2, width)
c
c ...  Output amplitude and width ratio values
c
        call print_params(0, sParm1New, sParm2New, numIter, subIter,
     >                amp1, amp2, ampRatioNew, width, widthRatioNew)
        call print_params(6, sParm1New, sParm2New, numIter, subIter,
     >                amp1, amp2, ampRatioNew, width, widthRatioNew)
c
c ...  Check for convergence
c
        distOld = sqrt( (1.-ampRatioOld)**2 + (1.-widthRatioOld)**2 )
        distNew = sqrt( (1.-ampRatioNew)**2 + (1.-widthRatioNew)**2 )

        if (ampRatioNew .eq. 0. .or.  distNew .gt. distOld) then
          maxStepSize = 2*maxStepSize
          rescale = .true.
          write(*, *)
          write(*, '(" Convergence Criteria Failure:")')
          write(*, '("   AmpRatio = ", f10.4)') ampRatioNew
          write(*, '("   Old Distance = ", f10.4)') distOld
          write(*, '("   New Distance = ", f10.4)') distNew
          write(*, *)
          write(*, '(" MaxStepSize => ", i3)') maxStepSize
          goto 100
        else
          if (rescale .and. maxStepSize .gt. 1)
     >       maxStepSize = maxStepSize/2
          rescale = .false.
        endif
c
c ...   Completed with entire iteration
c
        write(*, *)
        write(*, '(" Finished with ", i2, " iterations.")') numIter
        write(*, '(" Hit <return> to continue ...")')
        read(*, '(a)') adum
      enddo
      stop ' Normal program termination'
   10 stop ' Unable to open diagnostic output file'
      end
      subroutine print_params(lU, sParam1, sParam2, numIter, subIter,
     >                        amp1, amp2, ampRatio, width, widthRatio)
c
c***********************************************************************
c
c ... Subroutine to print out S parameters and corresponding signal
c     features
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Input Arguments:
c
        real*4    sParam1, sParam2
        real*4    amp1, amp2, ampRatio
        real*4    width, widthRatio
c
        integer*4 lU, numIter, subIter
c
c ... Return Arguments:
c
c ... Comments:
c
c***********************************************************************
c
c ... Check for screen output
c
      if (lU .eq. 0) then
        write(*, *)
        write(*,
     >       '(" ************************************************")')
        write(*,'("  sParameter1 = ", f6.2, 5x,
     >             " sParameter2 = ", f6.4)')
     >   sParam1, sParam2
        write(*,
     >       '(" ************************************************")')
        write(*,
     >       '(" Itr  SI   Amp1    Amp2   Ratio    Width  Ratio")')
        write(*,'(1x, 2i4, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >   numIter, subIter, amp1, amp2, ampRatio, width, widthRatio
        write(*,
     >       '(" ************************************************")')
      else
        write(lU, *)
```

```
  write(lU,
>       '(" *********************************************")')
  write(lU,
>       '(" sParameter1 = ", f6.2, 5x,
>         " sParameter2 = ", f6.4)')
>   sParam1, sParam2
  write(lU,
>       '(" *********************************************")')
  write(lU,
>       '(" Itr  SI   Amp1    Amp2   Ratio    Width  Ratio")')
  write(lU,'(1x, 2i4, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
>   numIter, subIter, amp1, amp2, ampRatio, width, widthRatio
  write(lU,
>       '(" *********************************************")')
 endif
 return
 end
```

```
      subroutine ratios (fName, width_goal, ampRatio, widRatio,
     >                    amp1, amp2, width)
c
c*******************************************************************
c
c ... Subroutine to calculate amplitude and width ratios based on
c      width requirement and values in current signal vector file.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Input Arguments:
c
        character*(*)   fName
        real*4          width_goal
c
c ... Return Arguments:
c
        real*4          ampRatio, widRatio
        real*4          amp1, amp2, width
c
c ... Subroutines called:
c
c       sRead
c       peakNdcs
c       sigWidth
c
c ... Comments:
c
c*******************************************************************
c
c ... Local Variables:
c
        parameter       (MAXPT=300)
c
        integer*4       nPtSVec
        integer*4       pk1Ndx, pk2Ndx, wd1Ndx, wd2Ndx
        logical         perror
        real*4          sVector(2,MAXPT)
c
c*******************************************************************
c
c ... Read in signal vector
c
```

```
      call sRead (fName, MAXPT, sVector, nPtSVec)
c
c ... Locate positions of peaks
c
      call peakNdcs(sVector, nPtSVec, pk1Ndx, pk2Ndx, perror)
c
c ... Check for error finding peaks
c
      if (perror) then
        ampRatio = 0.
        widRatio = 0.
        return
      endif
c
c ... Locate width indicies and interpolated width
c
      call sigWidth(sVector, nPtSVec, pk1Ndx, pk2Ndx,
     >               wd1Ndx, wd2Ndx, width)
c
c ... Calculate current ratios
c
      amp1  = sVector(2, pk1Ndx)
      amp2  = sVector(2, pk2Ndx)
c
      ampRatio = amp1/amp2
      widRatio = width/width_goal
c
      return
      end
      subroutine sRead (fName, MAXPT, sVector, nPtSVec)
c
c*******************************************************************
c
c ... Subroutine to read in the signal file output by the signal
c      generating program.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Input Arguments:
c
       character*(*)   fName
       integer*4       MAXPT
c
c ... Return Arguments:
c
       real*4          sVector(2,MAXPT)
       integer*4       nPtSVec
c
c ... Comments:
c
c*******************************************************************
c
c ... Local Variables:
c
       integer*4 lfName, i
       real*4    sdum
c
c*******************************************************************
c
      lfName = length(fName)
c
c ... Open signal file
c
      open(2, file=fName(1:lfName), status='old', form='formatted',
     >     err=10)
c
c ... Read in signal values
c
      do i=1,MAXPT+1
        read(2, err=20, end=30, fmt=*)
     >    sVector(1,i), sdum, sdum, sVector(2,i)
      enddo
```

```
c
      write(6, *) ' Signal file: ', fName(1:lfName)
      write(6, *) ' Contains too many records for current array size.'
      stop ' Program terminating ...'
   10 write(6, *) ' Error opening signal file: ', fName(1:lfName)
      stop ' Program terminating ...'
   20 write(6, *) ' Error reading record: ', i
      write(6, *) ' in signal file: ', fName
      stop ' Program terminating ...'
   30 close(2)
      nPtSVec = i-1
c
c ... Convert signal to correct units
c
      do i=1,NptSVec
        sVector(2,i) = 20.*alog10(sVector(2,i))
        write(6, *) i, '   x= ', sVector(1,i), '   y= ', sVector(2,i)
      enddo
      return
      end
      subroutine signalGen (ioMode, sParam1, sParam2)
c
c*********************************************************************
c
c ... Subroutine to generate a new signal given passed values of S
c     parameters.  If i/o mode is set to "read" then the values of the
c     S parameters in the current signal file are passed back to the
c     calling routine.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         12/24/93
c
c ... Input Arguments:
c
        integer*4      ioMode
        real*4         sParam1, sParam2
c
c ... Return Arguments:
c
c       real*4         sParam1, sParam2
c
c ... Subroutines Called:
c
c       sParamIO
c
c ... Comments:
c
c       Arguments sParam1, sParam2 are used as return values when
c       "READ" mode is used.
c
c*********************************************************************
c
c ... Local Variables:
c
        parameter (READ=0)
        parameter (WRITE=1)
c
        character*1   adum
        character*128 pName
c
c ...   Define name of file containing active S parameter values
c
        data pName(1:) /'demo.ckt'/
c
```

```
c***********************************************************************
c
c ... Access .CKT file for read/write of S parameters
c
      call sParamIO (pName, ioMode, sParam1, sParam2)
c
c ... Notify user to generate new signal using Eagle software
c      if in "write" mode.
c
      if (ioMode .eq. WRITE) then
        write(*, '(/, " Updated .CKT file with:   S1 => ", f10.4,
     >                "   S2 => ", f10.4)') sParam1, sParam2
        write(*, *)
        write(*, '(" Press <return> after new signal",
     >             " has been generated by Eagle Software ...")')
        read(*, '(a)') adum
      endif
c
      return
      end
      subroutine sParamIO (pName, ioMode, sParam1, sParam2)
c
c*********************************************************************
c
c ... Subroutine to read in the S parameter file (*.CKT) and
c     read/write the S parameters from/to the file.  The parameter
c     "ioMode" determines whether the S parameters are being read from or
c     written to the file "pName".
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         12/24/93
c
c ... Input Arguments:
c
        character*128   pName
        integer*4       ioMode
        real*4          sParam1, sParam2
c
c ... Return Arguments:
c
c       real*4          sParam1, sParam2
c
c ... Comments:
c
c*********************************************************************
c
c ... Local Variables:
c
        parameter (READ=0)
        parameter (WRITE=1)
        parameter (NUM_REC_MAX=200)
c
        integer*4 lpName, lenRec(NUM_REC_MAX)
c
        character*128 record(NUM_REC_MAX)
c
c*********************************************************************
c
c ... Open S parameter file
c
      lpName = length(pName)
      open(2, file=pName(1:lpName), status='old', form='formatted',
     >     err=10)
c
```

```
c ... Read records from S Parameter file.
c
      numRec = 0
      if (ioMode .eq. READ) then
        do while(numRec .lt. NUM_REC_MAX)
          record(1)(1:) = ' '
          read(2, err=20, end=30, fmt='(a)') record(1)(1:)
          lenRec(1) = length(record(1))
          numRec = numRec + 1
          if (numRec .eq. 2) read(record(1)(17:lenRec(1)), fmt='(f7.5)',
     >        err=25) sParam2
          if (numRec .eq. 46) read(record(1)(4:lenRec(1)), fmt='(f7.3)',
     >        err=25) sParam1
        enddo
      else
        do while(numRec .lt. NUM_REC_MAX)
          numRec = numRec + 1
          record(numRec)(1:) = ' '
          read(2, err=20, end=30, fmt='(a)') record(numRec)(1:)
          lenRec(numRec) = length(record(numRec))
          if (numRec .eq. 2) then
            write(record(numRec)(17:23), fmt='(f7.5)', err=25) sParam2
            if (sParam2 .lt. 1.0) record(numRec)(17:17) = '0'
          elseif (numRec .eq. 46) then
            write(record(numRec)(4:), fmt='(f7.3)', err=25) sParam1
          endif
        enddo
      endif
c
      write(6, *) ' S Parameter file: '//pName(1:lpName)
      write(6, *) ' Contains too many records for current array size.'
      stop ' Program terminating ...'
   10 write(6, *) ' Error opening S Parameter file: '//pName(1:lpName)
      stop ' Program terminating ...'
   20 write(6, *) ' Error reading record: ', numRec
      write(6, *) ' in S Parameter file: '//pName
      stop ' Program terminating ...'
   25 write(6, *) ' Error parsing information from record: ', numRec
      write(6, *) ' in file: ', pName(1:lpName)
      stop ' Program terminating ...'
   30 if (ioMode .eq. WRITE) then
        numRec = numRec - 1
        rewind(2)
        do i = 1, numRec
          write(2, '(a)') record(i)(1:lenRec(i))
        enddo
      endif
      close(2)
c
      return
      end
      function length(string)
c
c***********************************************************************
c
c ... Utility function to determine the length of a character string.
c     The length of the string is determined by finding the last
c     "non-blank" character in the passed string.
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          12/24/93
c
c ... Input Arguments:
c
        character*(*)   string
c
c ... Return Arguments:
c
c       integer*4       length
c
c ... Comments:
c
c***********************************************************************
c
      length = len(string)
      do while(string(length:length) .eq. ' ' .and. length .gt. 0)
        length = length - 1
      enddo
      return
      end
      subroutine peakNdcs(sVector, nPtSVec, pk1Ndx, pk2Ndx, error)
c
c***********************************************************************
c
c ... Subroutine to pick off the indicies of the 2 large peaks in
c     the signal.  An error is returned if 2 unique peaks are not found.
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          11/13/93
c
c ... Input Arguments:
c
        integer*4       nPtSVec
        real*4          sVector(2,nPtSVec)
c
c ... Return Arguments:
c
        integer*4       pk1Ndx
        integer*4       pk2Ndx
        logical         error
c
c ... Comments:
c
c***********************************************************************
c
c ... Local Parameter
c
      parameter         (DOWNTRIG=4)
c
c ... Local Variables:
c
      integer*4         ndxMax, downCnt, minNdx
      logical*2         negSlope, pk1Fnd
      real*4            valMax, valOld
c
c***********************************************************************
c
c ... Initialize values
c
      valMax   = -1.e32
      valOld   = -1.e32
      ndxMax   = -1
      pk1Fnd   = .false.
      negSlope = .false.
      downCnt  = 0
      pk1Ndx   = -1
      pk2Ndx   = -1
      minNdx   = -1
c
c ... Initialize Error Flag
c
      error = .false.
c
c ... Start search for 1st peak
c
      do i=1, nPtSVec
        if (sVector(2,i) .lt. valOld) then
          negSlope = .true.
          if (downCnt .eq. 0) then
            valMax = valOld
            ndxMax = i-1
          endif
          downCnt  = downCnt + 1
          if (downCnt .ge. DOWNTRIG .and. .not. pk1Fnd) then
            pk1Ndx = ndxMax
            pk1Fnd = .true.
          endif
        else
          negSlope = .false.
          downCnt = 0
          if (pk1Fnd) then
            minNdx = i-1
            goto 10
          endif
        endif
c
        valOld = sVector(2,i)
      enddo
   10 continue
c
      if (pk1Fnd) then
c
c ...   Write diagnostics to output screen & output file
c
        write(6, *)
        write(6, *) ' 1st Peak located at    index: ', pk1Ndx
        write(6, *) '                    frequency: ',sVector(1,pk1Ndx)
        write(6, *) '                    amplitude: ',sVector(2,pk1Ndx)
      else
        error = .true.
```

```
          write(6, *)
          write(6, *) ' Could not locate 1st Peak!'
          return
        endif
c
        if (pk1Fnd .and. minNdx .gt. 0) then
c
c ...     Locate 2nd peak
c
          valMax = -1.e32
          ndxMax = -1
          do i=minNdx, nPtSVec
            if (sVector(2,i) .gt. valMax) then
              valMax = sVector(2,i)
              ndxMax = i
            endif
          enddo
c
c ...     Set index for 2nd peak
c
          pk2Ndx = ndxMax
c
          write(6, *) ' '
          write(6, *) ' 2nd Peak located at      index: ', pk2Ndx
          write(6, *) '                      frequency: ',sVector(1,pk2Ndx)
          write(6, *) '                      amplitude: ',sVector(2,pk2Ndx)
c
        else
          error = .true.
          write(6, *) ' '
          write(6, *) ' Error in Peak Search: Couldn''t locate 2nd peak.'
        endif
c
        return
        end
```

```
      subroutine sigWidth(sVector, nPtSVec, pk1Ndx, pk2Ndx,
     >                   wd1Ndx, wd2Ndx, width)
c
c***********************************************************************
c
c ... Subroutine to calculate the width of the signal.
c      Currently the signal width is defined as the FWHM
c      locations spanning over the range of the 2 amplitude peaks.
c
c ... Programmer:        G.L. Kolte
c
c ... Created:           12/01/93
c
c ... Input Arguments:
c
        integer*4       nPtSVec, pk1Ndx, pk2Ndx
        real*4          sVector(2,nPtSVec)
c
c ... Return Arguments:
c
        integer*4       wd1Ndx
        integer*4       wd2Ndx
        real*4          width
c
c ... Comments:
c
c***********************************************************************
c
c ... Local Variables:
c
      integer*4         minNdx
      real*4            valHalfMax, valMin
      real*4            slope, frq1, frq2
c
c***********************************************************************
c
c ... Initialize values
c
      valMin   = 1.e32
      minNdx   = -1
c
c ... Find minimum amplitude
c
      do i=1, nPtSVec
        if (sVector(2,i) .lt. valMin) then
```

```
          valMin = sVector(2,i)
          minNdx = i
        endif
      enddo
c
c ... Output minimum amplitude (in DB scale)
c
      write(6, *) ' '
      write(6, *) ' Minimum amplitude index: ', minNdx
      write(6, *) '                 frequency: ', sVector(1, minNdx)
      write(6, *) '            amplitude (Db): ', sVector(2, minNdx)
      write(6, *) ' '
c
c ... Find maximum amplitude
c
      if (sVector(2, pk1Ndx) .lt. sVector(2, pk2Ndx)) then
        valMax = sVector(2, pk1Ndx)
      else
        valMax = sVector(2, pk2Ndx)
      endif
c
c ... Convert from Db scale
c
      valMin = 10**(valMin/20.)
      valMax = 10**(valMax/20.)
c
c ... Calculate Half maximum
c
      valHalfMax = valMin + (valMax - valMin)/2.
c
c ... Convert back to Db scale
c
      valHalfMax = 20.*alog10(valHalfMax)
c
c ... Output half maximum value
c
      write(6, *) ' '
      write(6, *) ' Half Maximum amplitude in signal: ', valHalfMax
      write(6, *) ' '
c
c ... Determine width indicies
c
      do i=pk1Ndx, 1, -1
        if (sVector(2,i) .lt. valHalfMax) goto 10
      enddo
   10 wd1Ndx = i
```

```
      do i=pk2Ndx, nPtSVec
        if (sVector(2,i) .lt. valHalfMax) goto 20
      enddo
   20 wd2Ndx = i
c
c ... Interpolate (linear) to get half-maximum channel boundaries
c
      slope = (sVector(1, wd1Ndx+1) - sVector(1, wd1Ndx))/
     >        (sVector(2, wd1Ndx+1) - sVector(2, wd1Ndx))
      frq1 = sVector(1, wd1Ndx) + slope*(valHalfMax - sVector(2,wd1Ndx))
      slope = (sVector(1, wd2Ndx-1) - sVector(1, wd2Ndx))/
     >        (sVector(2, wd2Ndx-1) - sVector(2, wd2Ndx))
      frq2 = sVector(1, wd2Ndx) + slope*(valHalfMax - sVector(2,wd2Ndx))
c
c ... Calculate width
c
      width = frq2 - frq1
c
c ... Output width indicies and amplitudes
c
      write(6, *)
      write(6, *) ' Width index 1: ', wd1Ndx
      write(6, *) '     frequency: ', sVector(1,wd1Ndx)
      write(6, *) '     amplitude: ', sVector(2,wd1Ndx)
      write(6, *)
      write(6, *) ' Width index 2: ', wd2Ndx
      write(6, *) '     frequency: ', sVector(1,wd2Ndx)
      write(6, *) '     amplitude: ', sVector(2,wd2Ndx)
      write(6, *)
      write(6, *) ' Interpl Width; ', width
c
      return
      end
```

APPENDIX 4B
TIPTOE2A Source Code Listings

DEMO.BAT

This program creates the DEMO2A.EXE code from several components using the Microsoft fortran (version 5.0) compiler.

```
fl /G2 demo2a.for tiptoe.for siggen.for sigfeat.for check.for
```

DEMO2A.FOR

```
      program demo2a
c
c**********************************************************************
c
c ... Program to demonstrate a tuning method utilizing the Tiptoe
c      algorithm.
c
c ... Programmer:        G.L. Kolte
c
c ... Created:           04/10/94
c
c ... Input Arguments:
c
c ... Return Arguments:
c
c ... Subroutines called:
c
c      tiptoe
c      checkParms
c
c ... Comments:
c
c**********************************************************************
c
c ... Local Variables:
c
       parameter ( DELTA_MAX_THRESH=1.0, DELTA_MIN_THRESH=1.7 )
       parameter ( NUM_OUTER_LIM=2, NUM_INNER_LIM=8 )
c
       parameter ( CNVRG_LIM = 0.005 )
       parameter ( WIDE_AMP_RATIO=1., WIDE_WID_OFF=0.,
     >             WIDE_WID_GOAL=DELTA_MIN_THRESH )
c
       real*4  outer_lim(2, NUM_OUTER_LIM)
       real*4  inner_lim(2,NUM_INNER_LIM)
       logical  tiptoe, checkParms, cwide, cnarr
       integer*4 failMode
c
       data outer_lim / 3240., 3., 3360., 3. /
c
       data inner_lim / 3265., 3., 3270., 1.5, 3275., 1.2, 3280., 1.2,
     >                  3320., 1., 3325., 1.,  3330., 1.3, 3335., 3. /
```

```
c
c**********************************************************************
c
c ... Initialize logicals
c
      cwide = .false.
      cnarr = .false.
c
c ... Look for Widest configuration
c
      if ( tiptoe( WIDE_AMP_RATIO, WIDE_WID_OFF,
     >             (WIDE_WID_GOAL-CNVRG_LIM), CNVRG_LIM,
     >             sP1Wide, sP2Wide ) ) then

        cwide = .true.
        write( *, '( " Tiptoe converged for wide configuration ...",
     >               /, "   SP1=", f10.4, " SP2= ", f10.4, / )' )
     >   sP1Wide, sP2Wide
      else
        write( *, '( " Tiptoe could not converge for wide ",
     >               " configuration ...", / )' )
      endif
c
c ... Check if finished.
c
      if ( checkParms( DELTA_MAX_THRESH, DELTA_MIN_THRESH,
     >                 NUM_OUTER_LIM, outer_lim, NUM_INNER_LIM,
     >                 inner_lim, failMode ) ) then
        write( *, '( /, " Signal passed all checks!", /)' )
        write( 6, '( /, " Signal passed all checks!", /)' )
      else
        write( *, '(" Signal failed check ...", /
     >              "   Mode of failure: ", i2)' ) failMode
        write( 6, '(" Signal failed check ...", /
     >              "   Mode of failure: ", i2)' ) failMode
      endif
c
      stop 'Demo2a ... Normal Completion'
      end
```

TIPTOE.FOR

```
      logical function tiptoe ( ampRatio, widLocOff, widthMag, ratioLim,
     >                          sParam1, sParam2 )
c
c*********************************************************************
c
c ... Function to calculate S-parameters utilizing Martin Lee's famous
c     tiptoe algorithm.  Inputs to this routine are the desired amplitude
c     ratio, the amplitude offset at which the width is to be calculated,
c     the desired magnitude of the width, and the ratio convergence
c     limit.  The funtion returns a boolean indicating whether or not
c     a solution was obtained.  If this function returns true, the valid
c     values of the S Parameters are passed back to the calling routine.
c     If the function returns false the S Parameter values contain the
c     last values attempted prior to failure.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         02/25/93
c
c
c ... Input Arguments:
c
      real*4  ampRatio
      real*4  widLocOff
      real*4  widthMag
      real*4  ratioLim
c
c ... Return Arguments
c
      real*4  sParam1
      real*4  sParam2
c
c ... Subroutines called:
c
c       signalGen
c       signalFeats
c       printParams
c
c ... Comments:
c
c       Tiptoe-1A main routine converted to a function for use
c       in the 2A phase.
c
```

```
c*********************************************************************
c ... Parameters:
c
        parameter      ( MAX_ITER=20, MAX_SUB_ITER=5 )
c
c ... Variables:
c
        character*1     adum
        character*128   oName
        logical         rescale, converged
        integer*4       loName
        integer*4       numIter, subIter
        integer*4       maxStepSize
        integer*4       read_mode, write_mode
        real*4          sParm1Old, sParm1New, dSParm1
        real*4          sParm2Old, sParm2New, dSParm2
        real*4          sAmp1, sAmp2, sAmpMin
        real*4          wChan1, wChan2, sWidth
        real*4          ampRatioOld, ampRatioNew, dAmpRatio
        real*4          widthRatioOld, widthRatioNew, dWidthRatio
        real*4          kickS1, kickS2
        real*4          dAdS1, dAdS2, dWdS1, dWdS2
        real*4          deltaAmp, deltaWidth, deltaSP1, deltaSP2
        real*4          detInv, distOld, distNew
c
c ... Data:
c
c ...   Define name of diagnostic output file
c
        data oName(1:) /'tiptoe.out'/
c
c ...   Define read/write modes
c
        data read_mode  /0/
        data write_mode /1/
c
c*********************************************************************
c
c ... Initialize counters
c
        maxStepSize = 1
        numIter     = 0
        subIter     = 0
        rescale     = .false.
        converged   = .false.
c
```

```
c ... Ask user for desired signal width
c
c     write(*, '("$Please enter Width Criterion: ")')
c     read(*, *) widthMag
c
c ... Open file for diagnostic output
c
      loName = length(oName)
      open(6, file=oName(1:loName), status='unknown', form='formatted',
    >   err=10)
c
c ... Read in current values for S parameters
c
      call signalGen(read_mode, sParm1New, sParm2New)
c
c ... Calculate Amplitude and Width Ratios
c
      call signalFeats( ampRatio, widLocOff, widthMag,
    >                   ampRatioNew, widthRatioNew, sAmp1, sAmp2,
    >                   sAmpMin, wChan1, wChan2, sWidth )
c
c ... Output amplitude and width ratio values
c
      call printParams( 0, sParm1New, sParm2New, numIter, subIter,
    >                   sAmp1, sAmp2, ampRatioNew, sWidth,
    >                   widthRatioNew )
      call printParams( 6, sParm1New, sParm2New, numIter, subIter,
    >                   sAmp1, sAmp2, ampRatioNew, sWidth,
    >                   widthRatioNew )
c
c ... Check that features were located
c
      if ( sWidth .eq. 0 )  then
      tiptoe = .false.
      sParam1 = sParm1New
      sParam2 = sParm2New
      return
      endif
c
c ... Start of iterative loop
c
      do while ( ( .not. converged ) .and. numIter .le. MAX_ITER)
c
c ...  Increment iteration counter
c
      numIter = numIter + 1
      subIter = 0
c
c ... Calculate perturbation for S parameters
c
      kickS1 = 0.01
      kickS2 = 0.01
c
c ... Store current S parameter values as "previous" values
c
      sParm1Old = sParm1New
      sParm2Old = sParm2New
c
c ... Store current Amplitude and Width ratios as "previous" values
c
      ampRatioOld   = ampRatioNew
      widthRatioOld = widthRatioNew
c
c ... Perturb current value of S1 parameter for next signal
c
   25 dSParm1   = kickS1*sParm1Old
c
      sParm1New = sParm1Old + dSParm1
c
c ... Generate new signal
c
      call signalGen(write_mode, sParm1New, sParm2Old)
c
c ... Calculate Amplitude and Width Ratios for dS1 signal vector
c
      call signalFeats( ampRatio, widLocOff, widthMag,
     >                  ampRatioNew, widthRatioNew, sAmp1, sAmp2,
     >                  sAmpMin, wChan1, wChan2, sWidth )
      write(6, *)
      write(6, '(" ******************************************")')
      write(6, '(" sParm1Prime = ", f6.2,
     >          " sParameter2 = ", f6.4)') sParm1New, sParm2Old
      write(6, '(" ******************************************")')
      write(6, '(" Iter   Amp1    Amp2   Ratio    Width  Ratio")')
      write(6, '(i4, 1x, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >  numIter, sAmp1, sAmp2, ampRatioNew, sWidth, widthRatioNew
      write(6, '(" ******************************************")')
c
c ... Check that features were located
c
      if ( sWidth .eq. 0 )  then
        kickS1 = kickS1/2.
```

```
         goto 25
        endif
c
c ...  Calculate dS1 partial derivatives
c
        dAmpRatio   = ampRatioNew - ampRatioOld
        dWidthRatio = widthRatioNew - widthRatioOld
c
        dAdS1 = dAmpRatio/dSParm1
        dWdS1 = dWidthRatio/dSParm1
c
c ...  Perturb current value of S2 parameter for next signal
c
   50   dSParm2   = kickS2*sParm2Old
c
        sParm2New = sParm2Old + dSParm2
c
c ...  Generate new signal
c
        call signalGen(write_mode, sParm1Old, sParm2New)
c
c ...  Calculate Amplitude and Width Ratios for dS2 signal vector
c
        call signalFeats( ampRatio, widLocOff, widthMag,
     >                 ampRatioNew, widthRatioNew, sAmp1, sAmp2,
     >                 sAmpMin, wChan1, wChan2, sWidth )
        write(6, *)
        write(6, '(" ******************************************")')
        write(6, '(" sParameter1 = ", f6.2,
     >         " sParm2Prime = ", f6.4)') sParm1Old, sParm2New
        write(6, '(" ******************************************")')
        write(6, '(" Iter  Amp1   Amp2  Ratio   Width  Ratio")')
        write(6, '(i4, 1x, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >   numIter, sAmp1, sAmp2, ampRatioNew, sWidth, widthRatioNew
        write(6, '(" ******************************************")')
c
c ...  Check that features were located
c
        if ( sWidth .eq. 0 )  then
         kickS2 = kickS2/2.
         goto 50
        endif
c
c ...  Calculate dS2 partial derivatives
c
        dAmpRatio   = ampRatioNew - ampRatioOld
```

```
      dWidthRatio = widthRatioNew - widthRatioOld
c
      dAdS2 = dAmpRatio/dSParm2
      dWdS2 = dWidthRatio/dSParm2
c
      write(6, *)
      write(6, *) ' Iteration #: ', numIter
      write(6, *) '        dAdS1: ', dAdS1
      write(6, *) '        dWdS1: ', dWdS1
      write(6, *) '        dAdS2: ', dAdS2
      write(6, *) '        dWdS2: ', dWdS2
      write(6, *)
c
c ... Calculate reciprical determinant of partial derivative matrix
c
      detInv = 1.0/(dAdS1*dWdS2 - dAdS2*dWdS1)
c
 100  subIter = subIter + 1
c
      if ( subIter .gt. MAX_SUB_ITER ) then
c
c ...   Maximum number of allowed sub-iterations exceeded without
c        convergence.  Return FALSE to the calling program.
c
        write( 6, '( /, " Maximum # of sub-iterations reached in ",
     >              "tiptoe algorithm.", /,
     >              " Retuning to calling program ...", // )' )
        tiptoe  = .FALSE.
        sParam1 = sParm1New
        sParam2 = sParm2New
        return
      endif
c
c ... Estimate needed change in amplitude and width ratios
c
      deltaAmp   = (ampRatio - ampRatioOld)/float(maxStepSize)
      deltaWidth = (1.0 - widthRatioOld)/float(maxStepSize)
c
c ... Calculate corresponding change in S parameter values
c
      deltaSP1 = detInv*(dWdS2*deltaAmp - dAdS2*deltaWidth)
      deltaSP2 = detInv*(dAdS1*deltaWidth - dWdS1*deltaAmp)
c
c ... Calculate new S parmeter values
c
      sParm1New = sParm1Old + deltaSP1
```

```
        sParm2New = sParm2Old + deltaSP2
c
c ...   Generate new signal
c
        call signalGen( write_mode, sParm1New, sParm2New )
c
c ...   Calculate Amplitude and Width Ratios
c
        call signalFeats( ampRatio, widLocOff, widthMag,
     >                    ampRatioNew, widthRatioNew, sAmp1, sAmp2,
     >                    sAmpMin, wChan1, wChan2, sWidth )
c
c ...   Output amplitude and width ratio values
c
        call printParams( 0, sParm1New, sParm2New, numIter, subIter,
     >                    sAmp1, sAmp2, ampRatioNew, sWidth,
     >                    widthRatioNew )
        call printParams( 6, sParm1New, sParm2New, numIter, subIter,
     >                    sAmp1, sAmp2, ampRatioNew, sWidth,
     >                    widthRatioNew )
c
c ...   Check for convergence
c
        distOld = sqrt( (ampRatio - ampRatioOld)**2 +
     >                  (1. - widthRatioOld)**2 )
        distNew = sqrt( (ampRatio - ampRatioNew)**2 +
     >                  (1. - widthRatioNew)**2 )
c
        if (sWidth .eq. 0. .or.  distNew .gt. distOld) then
          maxStepSize = 2*maxStepSize
          rescale = .true.
          write(*, *)
          write(*, '(" Convergence Criteria Failure:")')
          write(*, '("   AmpRatio = ", f10.4)') ampRatioNew
          write(*, '("   Old Distance = ", f10.4)') distOld
          write(*, '("   New Distance = ", f10.4)') distNew
          write(*, *)
          write(*, '(" MaxStepSize => ", i3)') maxStepSize
          goto 100
        else
          if (rescale .and. maxStepSize .gt. 1)
     >      maxStepSize = maxStepSize/2
          rescale = .false.
        endif
c
c ...   Completed with entire tiptoe iteration
```

```
c
      write( *, '( /, " Tiptoe finished with ",
     >              i2, " iterations." )' ) numIter
      write(*, '(" Hit <return> to continue ...")')
      read(*, '(a)') adum
c
c ...  Check for convergence
c
      if ( abs( ampRatio - ampRatioNew ) .le. ratioLim .and.
     >     abs( 1. - widthRatioNew ) .le. ratioLim )
     >   converged = .true.
c
      enddo
c
      if ( converged ) then
c
c ...  Convergence achieved so set return value to TRUE
c
      tiptoe = .true.
c
      else
c
c ...  Maximum number of allowed iterations exceeded without
c       convergence.  Return FALSE to the calling program.
c
      write( 6, '( /, " Maximum # of sub-iterations reached in ",
     >              "tiptoe algorithm.", /,
     >              " Retuning to calling program ...", // )' )
      tiptoe  = .false.
c
      endif
c
c ... Set current S Parameter values
c
      sParam1 = sParm1New
      sParam2 = sParm2New
c
      return
c
   10 stop ' Unable to open diagnostic output file in Tiptoe function.'
      end
      subroutine printParams( lU, sParam1, sParam2, numIter, subIter,
     >                  amp1, amp2, ampRatio, width, widthRatio )
c
c**********************************************************************
c
c ... Subroutine to print out S parameters and corresponding signal
c     features
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Input Arguments:
c
      real*4    sParam1, sParam2
      real*4    amp1, amp2, ampRatio
      real*4    width, widthRatio
c
      integer*4 lU, numIter, subIter
c
c ... Return Arguments:
c
c ... Comments:
c
c**********************************************************************
c
c ... Check for screen output
c
      if (lU .eq. 0) then
      write(*, *)
      write(*,
     >      '(" ************************************************")')
      write(*,'("  sParameter1 = ", f6.2, 5x,
     >           " sParameter2 = ", f6.4)')
     >  sParam1, sParam2
      write(*,
     >      '(" ************************************************")')
      write(*,
     >      '(" Itr  SI   Ampl    Amp2   Ratio    Width  Ratio")')
      write(*,'(1x, 2i4, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
     >  numIter, subIter, amp1, amp2, ampRatio, width, widthRatio
      write(*,
     >      '(" ************************************************")')
      else
      write(lU, *)
      write(lU,
>        '(" ***********************************************")')
   write(lU,
>        '(" sParameter1 = ", f6.2, 5x,
>          " sParameter2 = ", f6.4)')
>    sParam1, sParam2
   write(lU,
>        '(" ***********************************************")')
   write(lU,
>        '(" Itr  SI   Amp1    Amp2   Ratio    Width  Ratio")')
   write(lU,'(1x, 2i4, 2f8.2, 1x, f6.3, 1x, f8.2, 1x, f6.3)')
>    numIter, subIter, amp1, amp2, ampRatio, width, widthRatio
   write(lU,
>        '(" ***********************************************")')
   endif
   return
   end
```

```
      subroutine signalGen( ioMode, sParam1, sParam2 )
c
c**********************************************************************
c
c ... Subroutine to generate a new signal given passed values of S
c     parameters.  If i/o mode is set to "read" then the values of the
c     S parameters in the current signal file are passed back to the
c     calling routine.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         12/24/93
c
c ... Input Arguments:
c
        integer*4      ioMode
        real*4         sParam1, sParam2
c
c ... Return Arguments:
c
c       real*4         sParam1, sParam2
c
c ... Subroutines Called:
c
c       sParamIO
c
c ... Comments:
c
c       Arguments sParam1, sParam2 are used as return values when
c       "READ" mode is used.
c
c**********************************************************************
c
c ... Local Variables:
c
        parameter (READ=0)
        parameter (WRITE=1)
c
        character*1    adum
        character*128  pName
c
c ...   Define name of file containing active S parameter values
c
        data pName(1:) /'demo.ckt'/
```

```
c
c*********************************************************************
c
c ... Access .CKT file for read/write of S parameters
c
      call sParamIO( pName, ioMode, sParam1, sParam2 )
c
c ... Notify user to generate new signal using Eagle software
c     if in "write" mode.
c
      if (ioMode .eq. WRITE) then
        write(*, '(/, " Updated .CKT file with:   S1 => ", f10.4,
     >                "   S2 => ", f10.4)') sParam1, sParam2
        write(*, *)
        write(*, '(" Press <return> after new signal",
     >             " has been generated by Eagle Software ...")')
        read(*, '(a)') adum
      endif
c
      return
      end
      subroutine sParamIO( pName, ioMode, sParam1, sParam2 )
c
c*******************************************************************
c
c ... Subroutine to read in the S parameter file (*.CKT) and
c      read/write the S parameters from/to the file.  The parameter
c      "ioMode" determines whether the S parameters are being read from or
c      written to the file "pName".
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          12/24/93
c
c ... Input Arguments:
c
        character*128   pName
        integer*4       ioMode
        real*4          sParam1, sParam2
c
c ... Return Arguments:
c
c       real*4          sParam1, sParam2
c
c ... Comments:
c
c*******************************************************************
c
c ... Local Variables:
c
        parameter (READ=0)
        parameter (WRITE=1)
        parameter (NUM_REC_MAX=200)
c
        integer*4 lpName, lenRec(NUM_REC_MAX)
c
        character*128 record(NUM_REC_MAX)
c
c*******************************************************************
c
c ... Open S parameter file
c
      lpName = length(pName)
      open(2, file=pName(1:lpName), status='old', form='formatted',
     >     err=10)
c
c ... Read records from S Parameter file.
```

```
c
      numRec = 0
      if (ioMode .eq. READ) then
        do while(numRec .lt. NUM_REC_MAX)
          record(1)(1:) = ' '
          read(2, err=20, end=30, fmt='(a)') record(1)(1:)
          lenRec(1) = length(record(1))
          numRec = numRec + 1
          if (numRec .eq. 2) read(record(1)(17:lenRec(1)), fmt='(f7.5)',
     >        err=25) sParam2
          if (numRec .eq. 46) read(record(1)(4:lenRec(1)), fmt='(f7.3)',
     >        err=25) sParam1
        enddo
      else
        do while(numRec .lt. NUM_REC_MAX)
          numRec = numRec + 1
          record(numRec)(1:) = ' '
          read(2, err=20, end=30, fmt='(a)') record(numRec)(1:)
          lenRec(numRec) = length(record(numRec))
          if (numRec .eq. 2) then
            write(record(numRec)(17:23), fmt='(f7.5)', err=25) sParam2
            if (sParam2 .lt. 1.0) record(numRec)(17:17) = '0'
          elseif (numRec .eq. 46) then
            write(record(numRec)(4:), fmt='(f7.3)', err=25) sParam1
          endif
        enddo
      endif
c
      write(6, *) ' S Parameter file: '//pName(1:lpName)
      write(6, *) ' Contains too many records for current array size.'
      stop ' Program terminating ...'
   10 write(6, *) ' Error opening S Parameter file: '//pName(1:lpName)
      stop ' Program terminating ...'
   20 write(6, *) ' Error reading record: ', numRec
      write(6, *) ' in S Parameter file: '//pName
      stop ' Program terminating ...'
   25 write(6, *) ' Error parsing information from record: ', numRec
      write(6, *) ' in file: ', pName(1:lpName)
      stop ' Program terminating ...'
   30 if (ioMode .eq. WRITE) then
        numRec = numRec - 1
        rewind(2)
        do i = 1, numRec
          write(2, '(a)') record(i)(1:lenRec(i))
        enddo
      endif
      close(2)
c
      return
      end
```

SIGFEAT.FOR

```
      subroutine signalFeats( ampRatioGoal, widLocOff, refWidth,
     >                  ampRatio, widRatio, sAmp1, sAmp2, sAmpMin,
     >                  wChan1, wChan2, sWidth )
c
c*********************************************************************
c
c ... Subroutine to measure and evaluate the features of the current
c     signal.  Inputs include the desired amplitude ratio,
c     the amplitude offset at which the signal
c     width is to be evaluated and the reference width from which
c     to calculate the width ratio.  The returned feature values include
c     the amplitude of both  principal maxima, the amplitude of the
c     principal minimum, the fractional channel boundaries that
c     determine the signal width, the sinal width, the ratio of the
c     two principal maxima, and the ratio of the measured width to the
c     supplied reference width.
c
c ... Programmer:     G.L. Kolte
c
c ... Created:        03/05/93
c
c
c ... Input Arguments:
c
      real*4        ampRatioGoal
      real*4        widLocOff
      real*4        refWidth
c
c ... Return Arguments:
c
      real*4        ampRatio, widRatio
      real*4        sAmp1, sAmp2, sAmpMin
      real*4        wChan1, wChan2, sWidth
c
c ... Subroutines called:
c
c      sRead
c      peakNdcs
c      sigWidth
c
c ... Comments:
c
c*********************************************************************
c
```

```
c ... Local Variables:
c
         parameter        (MAXPT=300)
c                                                              x
         character*128   sName
         integer*4        nPtSVec
         integer*4        pk1Ndx, pk2Ndx, minNdx
         logical          perror
         real*4           sVector(2,MAXPT), sAmpMax, widthLoc, deltaMax
c
c ... Data:
c
c ...   Define name of file containing signal amplitudes
c
         data sName(1:) /'signal.out'/
c
c*********************************************************************
c
c ... Read in signal vector
c
      call sRead (sName, MAXPT, sVector, nPtSVec)
c
c ... Locate positions of peaks
c
      call peakNdcs(sVector, nPtSVec, pk1Ndx, pk2Ndx, minNdx, perror)
c
c ... Check for error finding peaks
c
      if (perror) then
         ampRatio = 0.
         widRatio = 0.
         return
      endif
c
c ... Set maxima and minimum amplitude values
c
      sAmp1    = sVector(2, pk1Ndx)
      sAmp2    = sVector(2, pk2Ndx)
      sAmpMin = sVector(2, minNdx)
c
c ... Use an average to describe the maximum from which to measure
c      relative offsets.
c
      if ( ampRatioGoal .ge. 1.0 ) then
         deltaMax = ( 1.0 - ampRatioGoal )*sAmp1
      else
```

```
          deltaMax = ( ampRatioGoal - 1.0 )*sAmp2
        endif
        sAmpMax = ( sAmp1 + sAmp2 + deltaMax )/2.0
c
c ... Check "widLocOff" value to determine which width feature to use.
c     Define the width to be the amplitude difference between the maximum
c     and the minimum if the width location offset is zero.  Otherwise
c     define the width to be the distance across the signal at the
c     relative offset distance below the maximum peak.
c
        if ( widLocOff .le. 0. ) then
          wChan1 = sVector(1, minNdx)
          wChan2 = wChan1
          sWidth = sAmpMax - sAmpMin
        else
c
c ...    Set relative width location
c
          widthLoc = sAmpMax - widLocOff
c
c ...    Locate width indicies and interpolated width
c
          call sigWidth( sVector, nPtSVec, pk1Ndx, pk2Ndx, widthLoc,
     >                   wChan1, wChan2, sWidth)
        endif
c
c ... Calculate current ratios
c
        ampRatio = sAmp2/sAmp1
        widRatio = sWidth/refWidth
c
        return
        end
```

```
      subroutine sRead (fName, MAXPT, sVector, nPtSVec)
c
c*********************************************************************
c
c ... Subroutine to read in the signal file output by the signal
c      generating program.
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          11/13/93
c
c ... Input Arguments:
c
        character*(*)  fName
        integer*4      MAXPT
c
c ... Return Arguments:
c
        real*4         sVector(2,MAXPT)
        integer*4      nPtSVec
c
c ... Comments:
c
c*********************************************************************
c
c ... Local Variables:
c
        integer*4 lfName, i
        real*4    sdum
c
c*********************************************************************
c
      lfName = length(fName)
c
c ... Open signal file
c
      open(2, file=fName(1:lfName), status='old', form='formatted',
     >     err=10)
c
c ... Read in signal values
c
      do i=1,MAXPT+1
        read(2, err=20, end=30, fmt=*)
     >    sVector(1,i), sdum, sdum, sVector(2,i)
      enddo
c
```

```
      write(6, *) ' Signal file: ', fName(1:lfName)
      write(6, *) ' Contains too many records for current array size.'
      stop ' Program terminating ...'
   10 write(6, *) ' Error opening signal file: ', fName(1:lfName)
      stop ' Program terminating ...'
   20 write(6, *) ' Error reading record: ', i
      write(6, *) ' in signal file: ', fName(1:lfName)
      stop ' Program terminating ...'
   30 close(2)
      nPtSVec = i-1
c
c ... Convert signal to DB
c
      do i=1,NptSVec
        sVector(2,i) = 20.*alog10(sVector(2,i))
        write(6, *) i, '   x= ', sVector(1,i), '   y= ', sVector(2,i)
      enddo
      return
      end
      subroutine peakNdcs( sVector, nPtSVec, pk1Ndx, pk2Ndx, minNdx,
     >                      error )
c
c***********************************************************************
c
c ... Subroutine to pick off the indicies of the 2 large peaks and the
c      minimum in the signal.  An error is returned if 2 unique peaks are
c      not found.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         11/13/93
c
c ... Updated:         03/03/94
c
c ... Input Arguments:
c
         integer*4      nPtSVec
         real*4         sVector(2,nPtSVec)
c
c ... Return Arguments:
c
         integer*4      pk1Ndx
         integer*4      pk2Ndx
         integer*4      minNdx
         logical        error
c
c ... Comments:
c
c***********************************************************************
c
c ... Local Parameter
c
      parameter         (DOWNTRIG=4)
c
c ... Local Variables:
c
      integer*4         ndxMax, downCnt
      logical*2         negSlope, pk1Fnd
      real*4            valMax, valMin, valOld
c
c***********************************************************************
c
c ... Initialize values
c
      valMax   = -1.e32
      valOld   = -1.e32
      valMin   =  1.e32
      ndxMax   = -1
      pk1Fnd   = .false.
      negSlope = .false.
      downCnt  = 0
      pk1Ndx   = -1
      pk2Ndx   = -1
      minNdx   = -1
c
c ... Initialize Error Flag
c
      error = .false.
c
c ... Start search for 1st peak
c
      do i=1, nPtSVec
        if (sVector(2,i) .lt. valOld) then
          negSlope = .true.
          if (downCnt .eq. 0) then
            valMax = valOld
            ndxMax = i-1
          endif
          downCnt  = downCnt + 1
          if (downCnt .ge. DOWNTRIG .and. .not. pk1Fnd) then
            pk1Ndx = ndxMax
            pk1Fnd = .true.
          endif
        else
          negSlope = .false.
          downCnt = 0
          if (pk1Fnd) then
            minNdx = i-1
            goto 10
          endif
        endif
c
        valOld = sVector(2,i)
      enddo
   10 continue
c
      if (pk1Fnd) then
c
c ...  Write diagnostics to output screen & output file
c
        write(6, *)
```

```
          write(6, *) ' 1st Peak located at    index: ', pk1Ndx
          write(6, *) '                        frequency: ',sVector(1,pk1Ndx)
          write(6, *) '                        amplitude: ',sVector(2,pk1Ndx)
        else
          error = .true.
          write(6, *)
          write(6, *) ' Could not locate 1st Peak!'
          return
        endif
c
        if (pk1Fnd .and. minNdx .gt. 0) then
c
c ...    Locate 2nd peak
c
          valMax = -1.e32
          ndxMax = -1
          do i=minNdx, nPtSVec
            if (sVector(2,i) .gt. valMax) then
              valMax = sVector(2,i)
              ndxMax = i
            endif
          enddo
c
c ...    Set index for 2nd peak
c
          pk2Ndx = ndxMax
c
          write(6, *) ' '
          write(6, *) ' 2nd Peak located at    index: ', pk2Ndx
          write(6, *) '                        frequency: ',sVector(1,pk2Ndx)
          write(6, *) '                        amplitude: ',sVector(2,pk2Ndx)
c
c ...    Locate minimum amplitude between principal maxima
c
          do i=pk1Ndx, pk2Ndx
            if (sVector(2,i) .lt. valMin) then
              valMin = sVector(2,i)
              minNdx = i
            endif
          enddo
c
          write(6, *) ' '
          write(6, *) ' Minimum located at     index: ', minNdx
          write(6, *) '                        frequency: ',sVector(1,minNdx)
          write(6, *) '                        amplitude: ',sVector(2,minNdx)
c
```

```
      else
         error = .true.
         write(6, *) ' '
         write(6, *) ' Error in Peak Search: Couldn''t locate 2nd peak.'
      endif
c
      return
      end
      subroutine sigWidth( sVector, nPtSVec, pk1Ndx, pk2Ndx, widthLoc,
     >                       wChan1, wChan2, width )
c
c*********************************************************************
c
c ... Subroutine to calculate the width of the signal at the DB level
c     specified by the input parameter "widthLoc".  Linear interpolation
c     is performed to find the fractional channel corresponding to the
c     specified amplitude "widthLoc".
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          12/01/93
c
c ... Updated:          03/10/94
c
c ... Input Arguments:
c
        integer*4       nPtSVec, pk1Ndx, pk2Ndx
        real*4          sVector(2,nPtSVec)
c
c ... Return Arguments:
c
        real*4          wChan1
        real*4          wChan2
        real*4          width
c
c ... Comments:
c
c*********************************************************************
c
c ... Local Variables:
c
      integer*4         wd1Ndx, wd2Ndx
      real*4            valMin1, valMin2, valMin
      real*4            slope
c
c*********************************************************************
c
c ... Calcuate validity regions for width
c
c ... Find minimum amplitude on front side of 1st peak
c
      valMin1 = 1.e32
      do i=1, pk1Ndx
        if ( sVector(2,i) .lt. valMin1 ) valMin1 = sVector(2,i)
      enddo
c
c ... Find minimum amplitude on back side of 2nd peak
c
      valMin2 = 1.e32
      do i=pk2Ndx, nPtSVec
        if ( sVector(2,i) .lt. valMin2 ) valMin2 = sVector(2,i)
      enddo
c
c ... Set Minimum acceptable amplitude
c
      if ( valMin1 .lt. valMin2 ) then
        valMin = valMin2
      else
        valMax = valMin1
      endif
c
c ... Set Maximum acceptable amplitude
c
      if (sVector(2, pk1Ndx) .lt. sVector(2, pk2Ndx)) then
        valMax = sVector(2, pk1Ndx)
      else
        valMax = sVector(2, pk2Ndx)
      endif
c
c ... Check if requested amplitude for width measurement is within
c     the acceptable range.
c
      if ( widthLoc .lt. valMin .or. widthLoc .gt. valMax ) then
        write(6, *) ' '
        write(6, *) ' ERROR determining signal width...'
        write(6, *) ' Amplitude for width measurement outside of limit.'
        write(6, *) ' Width requested at amplitude (DB): ', widthLoc
        write(6, *) ' Minimum width amplitude (DB)     : ', valMin
        write(6, *) ' Maximum width amplitude (DB)     : ', valMax
        write(6, *) ' '
c
c ...   Set width to Zero and return
c
        width = 0.
        return
c
      endif
c
c ... Output amplitude at which to measure width
c
```

```
      write(6, *) ' '
      write(6, *) ' Measure width at amplitude (Db): ', widthLoc
      write(6, *) ' '
c
c ... Determine width indicies
c
      do i=pk1Ndx, 1, -1
        if (sVector(2,i) .lt. widthLoc) goto 10
      enddo
   10 wd1Ndx = i
      do i=pk2Ndx, nPtSVec
        if (sVector(2,i) .lt. widthLoc) goto 20
      enddo
   20 wd2Ndx = i
c
c ... Interpolate (linear) to get fractional channel boundaries
c
      slope = (sVector(1, wd1Ndx+1) - sVector(1, wd1Ndx))/
     >        (sVector(2, wd1Ndx+1) - sVector(2, wd1Ndx))
      wChan1 = sVector(1, wd1Ndx) + slope*(widthLoc - sVector(2,wd1Ndx))
      slope = (sVector(1, wd2Ndx-1) - sVector(1, wd2Ndx))/
     >        (sVector(2, wd2Ndx-1) - sVector(2, wd2Ndx))
      wChan2 = sVector(1, wd2Ndx) + slope*(widthLoc - sVector(2,wd2Ndx))
c
c ... Calculate width
c
      width = wChan2 - wChan1
c
c ... Output fractional channels and width
c
      write(6, *)
      write(6, *) ' Signal Width-Channel 1: ', wChan1
      write(6, *) ' Signal Width-Channel 2: ', wChan2
      write(6, *) '      Interpolated Width: ', width
c
      return
      end
      function length( string )
c
c*********************************************************************
c
c ... Utility function to determine the length of a character string.
c      The length of the string is determined by finding the last
c      "non-blank" character in the passed string.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         12/24/93
c
c ... Input Arguments:
c
        character*(*)   string
c
c ... Return Arguments:
c
c       integer*4       length
c
c ... Comments:
c
c*********************************************************************
c
      length = len( string )
      do while ( string(length:length) .eq. ' ' .and. length .gt. 0 )
        length = length - 1
      enddo
      return
      end
      logical function
     > checkParms( deltaMaxThresh, deltaMinThresh, NOL, outer_lim,
     >             NIL, inner_lim, failMode )
c
c*********************************************************************
c
c ... Function to check if current signal meets the spec requirements.
c     Returns TRUE if requirements are met, otherwise returns FALSE.
c     The parameter "failMode" is used to signal the cause of failure.
c
c ... Programmer:      G.L. Kolte
c
c ... Created:         04/24/94
c
c ... Input Arguments:
c
      integer*4       NOL, NIL, failMode
      real*4          deltaMaxThresh, deltaMinThresh
      real*4          outer_lim(2,NOL), inner_lim(2,NOL)
c
c ... Return Arguments:
c
c      boolean value of true or false
c
c ... Subroutines called
c
c               sRead
c               peakNdcs
      real*4    relSigVal
c
c ... Comments:
c
c*********************************************************************
c
c ... Local Variables:
c
      parameter       (MAXPT=300)
      parameter       ( CHN=1, AMP=2 )
c
      real*4          sAmp1, sAmp2, sAmpMin
      real*4          sVector(2,MAXPT), sAmpMax, relVal
      character*128   sName
      integer*4       nPtSVec
```

```
        integer*4       pk1Ndx, pk2Ndx, minNdx
        logical         perror
c
c ... Data:
c
c ...  Define name of file containing signal amplitudes
c
        data sName(1:) /'signal.out'/
c
c**********************************************************************
c
c ... Initialize return value.
c
      checkParms = .false.
c
c ... Read in signal vector
c
      call sRead (sName, MAXPT, sVector, nPtSVec)
c
c ... Locate positions of peaks
c
      call peakNdcs( sVector, nPtSVec, pk1Ndx, pk2Ndx, minNdx, perror )
c
c ... Check for error finding peaks
c
      if ( perror ) return
c
c ... Set maxima and minimum amplitude values
c
      sAmp1 = sVector(AMP, pk1Ndx)
      sAmp2 = sVector(AMP, pk2Ndx)
      sAmpMax = amax0( sAmp1, sAmp2 )
      sAmpMin = sVector(AMP, minNdx)
c
c ... Check feature limits
c
      if ( ( abs( sAmp1 - sAmp2 ) .gt. deltaMaxThresh ) .or.
     >     ( abs( sAmpMax - sAmpMin ) ) .gt. deltaMinThresh ) then
        failMode = 1
        return
      endif
c
      write( *, '(" Signal Passed Amplitude Checks ... ")' )
      write( 6, '(" Signal Passed Amplitude Checks ... ")' )
c
c ... Check outer limits
```

```
c
      do i = 1, NOL
        relVal =
     >    relSigVal( sAmpMax, outer_lim(CHN, i), sVector, nPtSVec )
        if ( relVal .lt. outer_lim(AMP, i) ) then
          failMode = 2
          return
        endif
      enddo
c
      write( *, '(" Signal Passed Outer Bounds Checks ... ")' )
      write( 6, '(" Signal Passed Outer Bounds Checks ... ")' )
c
c ... Check inner limits
c
      do i = 1, NIL
        relVal =
     >    relSigVal( sAmpMax, inner_lim(CHN, i), sVector, nPtSVec )
        if ( relVal .gt. inner_lim(AMP, i) ) then
          failMode = 3
          return
        endif
      enddo
c
      write( *, '(" Signal Passed Interior Bounds Checks ... ")' )
      write( 6, '(" Signal Passed Interior Bounds Checks ... ")' )
c
      checkParms = .true.
c
      return
      end
      real*4 function
     > relSigVal( refAmp, chanNum, sVector, nPtSVec )
c
c*********************************************************************
c
c ... Function to calculate and return the signal amplitude corresponding
c     to the supplied channel frequency.  Linear interpolation between
c     signal values is used.  Returned value is defined as the abosulute
c     value of the distance between the supplied reference amplitude
c     and the amplitude of the signal at the supplied frequency.
c
c ... Programmer:       G.L. Kolte
c
c ... Created:          04/24/94
c
c ... Input Arguments:
c
        integer*4 nPtSVec
        real*4    refAmp, chanNum, sVector(2, nPtSVec)
c
c ... Return Arguments:
c
c       real*4      relSigVal
c
c ... Comments:
c
c*********************************************************************
c
c ... Local Variables:
c
        parameter       ( CHN=1, AMP=2 )
c
        real*4          ampHi, ampLo, chanHi, chanLo, slope, sigVal
        integer*4       i
c
c*********************************************************************
c
      relSigVal = 0.
c
c ... Do not allow extrapolation.  Return boundary values if frequency
c     is outside of table limits.
c
      if ( chanNum .le. sVector(CHN, 1) ) then
        relSigVal = sVector(CHN, 1)
        return
      endif
```

```
c
      if ( chanNum .ge. sVector(CHN, nPtSVec) ) then
        relSigVal = sVector(CHN, nPtSVec)
        return
      endif
c
      i = 1
      do while ( i .le. nPtSVec .and. sVector(CHN, i) .le. chanNum )
        i = i + 1
      enddo
c
      chanLo = sVector(CHN, i-1)
      chanHi = sVector(CHN, i)
      ampLo  = sVector(AMP, i-1)
      ampHi  = sVector(AMP, i)
c
      slope = (ampHi - ampLo)/(chanHi - chanLo)
      sigVal = ampLo + slope*(chanNum - chanLo)
      relSigVal = abs( refAmp - sigVal )
c
      return
      end
```

5.0 OUTPUT CAVITY SUBSYSTEM

The klystron's rf output signal can be expert controlled by having adjustable-dimension output cavities and accurate large-signal predictions, i.e., predictions of interactions between output-cavity electromagnetic fields and non-linearly modulated electron beams. Clearly, modeling large-signal interactions introduces new difficulties for the expert subsystem, but much of this difficulty results from using costly and time-consuming field solvers.

In this work, expected subsystem difficulties were reduced by avoiding field-solvers and by using the rf voltages and rf currents of the cavity to model the interaction. The reduced accuracy resulting from loss of field information is not significant and is compensated for by greater speed and ease of implementation in the expert system.

Voltages and currents are easily obtained using a lumped-element circuit model that predicts a complex impedance at each gap. For a lossless circuit, these complex impedances determine the magnitudes of and phases between the voltages and currents across the gaps.

Because the final klystron smart tube was to have a two-cavity extended-interaction output circuit (EIOC), a lumped-element model of such a circuit was developed. Model predictions of the signal phase at a convenient reference plane were compared to measurements on a cold-test structure. Section 5.1 describes the model and compares predictions with measurements. The EIOC model was ready for use in an interaction model.

In the interaction model, assumed rf currents at the gaps are used with complex impedances from the EIOC model to calculate gap voltages and power at the load. The new voltages correspond to wavelets that remodulate the beam and yield new gap currents. Using new currents, the procedure can be iterated until voltages and currents are self-consistent. Development of an interaction model was completed for a single output cavity. This work is described in Section 5.2.

5.1 Two-Cavity EIOC Equivalent Circuit

The lumped element network model of a two-interaction-gap output circuit for a high-power klystron is shown in Fig. 5.1-1A. The resistances R1 and R2 are included to account for the source impedance of the driving currents, resistance paper, or shorting bars. Capacitance C3 is included to represent radial electric fields in the coupling iris between the two cavities while C6 represents the evanescent modes at the output iris. The leakage reactance of the output iris is not shown as a separate element but is accounted for by the choice of the reference plane in the output waveguide.

The model provides for the inclusion of an inductive post in the output waveguide, but since there is none in this cold test vehicle we set the Post-To-Reference distance equal to zero as shown by the data in any of the figures that follow Fig.5.1-1. Note that, in these data, the Iris-To-Post offset is 6.331 inches which is a little less than the actual 6.88 inches from the EIOC physical iris to the physical reference. One of the universal problems encountered when modeling distributed networks with lumped elements is the location of terminal pairs. The random searching technique shows us that the electrical iris is of the order of 0.5 inch toward the load from the physical iris. The electric and magnetic fields associated with the evanescent modes in the vicinity of the iris are accounted for reasonably well by L6, C6, and this small offset distance.

Figure 5.1-1B shows features of a brass cold test model of a two-gap EIOC. It was constructed with provisions to deform the cavity walls in the vicinity of the coupling iris between the two cavities. From experience with an experimental klystron using a similar EIOC, it was felt that considerable benefit could be derived from having the coupling between the cavities externally adjustable. In previous designs, the deformable cavity walls were placed on the cavity walls far from the coupling iris. We had long since come to the realization that changes in the geometry anywhere in the EIOC had implications for the fields throughout the device, but just what the effect might be of having the movable walls adjacent to the coupling iris was not clear.

We have developed a method of reducing certain complex microwave networks to lumped element models (1) which are far more amenable to analysis once the driving functions are somehow in hand. We took the required data, which consists of the angle of the reflection coefficient taken at some convenient reference plane in the output waveguide where there is only one propagating mode in the band of interest, with (a) both gaps shorted, (b) only the first gap shorted, and (c) the unperturbed EIOC. This was repeated for three configurations of the tuning plugs. In all of these cases, the magnitude of the reflection coefficient is near unity and the model is based on that assumption. After the data for each of these three configurations was processed to find the elements of the lumped element model, further confirming data was taken with the output iris de-tuned by a shorting plug in the output waveguide. The results of this sequence of experiments is presented in Table 5.1-1.

Typical comparisons between model predictions and cold-test measurements are shown for configuration A, the configuration where the cold-test plugs are pushed as far as possible into the cavities. Figure 5.1-2 shows that when both gaps are open there is no measurable difference between measured and predicted phase at the reference plane. Calculations based on the model agree with data taken across a 15% bandwidth on the unperturbed EIOC within approximately 2 degrees rms, which is the reproducibility limit of the raw data. The discrepancies are somewhat larger for the cases where one or more gaps are shorted as shown in Figures 5.1-3 and 5.1-4. In the model, a shorted gap is a 1 ohm resistor across a capacitor which kills nearly all of the electric field in the immediate vicinity and almost none of the magnetic field. Contrary to the model, shorting the gaps in the EIOC removes most, but not all, of the electric field in a cavity while disturbing a small percentage, not zero, of the magnetic field.

Fig. 5.1-5 shows the magnitude and phase of the impedance calculated at the second gap for the case when the first gap is shorted. Values are plotted between 3.1 GHz and 3.6 GHz. When these values and an equivalent set of impedances for the first gap are seen by 30 Amp currents (assumed), the voltages shown in Fig. 5.1-6 are obtained at gaps one and two. These rf voltages

CONFIGURATION 'A' BOTH PLUGS MAXIMUM IN

RESONANT FREQUENCIES, GHz	MODEL	DATA	ERROR
F_pi Both Gaps Open	3.28	3.30	-0.61%
F_2pi Both Gaps Open	3.50	3.56	-1.70%
F_Gap1 ¦ Gap2 Shorted	3.38	3.38	0.00%
F_Gap2 ¦ Gap1 Shorted	3.42	3.47	-1.45%

CONFIGURATION 'B' BOTH PLUGS AT +0.25 INCHES

RESONANT FREQUENCIES, GHz	MODEL	DATA	
F_pi Both Gaps Open	3.18	3.20	-0.63%
F_2pi Both Gaps Open	3.42	3.48	-1.74%
F_Gap1 ¦ Gap2 Shorted	3.26	3.27	-0.31%
F_Gap2 ¦ Gap1 Shorted	3.34	3.40	-1.78%

CONFIGURATION 'C' BOTH PLUGS AT +0.50 INCHES

RESONANT FREQUENCIES, GHz	MODEL	DATA	
F_pi Both Gaps Open	3.19	3.20	-0.31%
F_2pi Both Gaps Open	3.45	3.47	-0.58%
F_Gap1 ¦ Gap2 Shorted	3.25	3.26	-0.31%
F_Gap2 ¦ Gap1 Shorted	3.39	3.40	-0.29%

TABLE 5.1-1
COMPARISON OF MEASURED AND PREDICTED TWO-CAVITY EIOC FREQUENCIES

have levels comparable to the beam voltage and therefore represent conditions where saturation and other nonlinear phenomena are likely to dominate. If nonlinearities are ignored, the EIOC model alone can provide an upper bound on the available output power by assuming the interaction remains linear. Fig. 5.1-7 shows the output power estimates from using the voltages in the EIOC model and the assumed 30 Amp gap currents.

5.2 Klystron Large-signal Program

5.2.1 Beam model

The beam model used in RELMOD9 is a highly stylized model in which any degree of density modulation and any degree of velocity modulation, with any phase shift between them, can be independently specified. This allows the interaction algorithm to be tested under the widest range of input modulations without having to spend time finding prior cavity combinations that would produce these modulations. The input beam is effectively defined by specifying the injection time and injection velocity for each charged particle over one rf period. The charged particles in the model are visualized as discs, each disc being a cloud of actual electrons. Since they are not represented as being hard discs, but as clouds, they can pass through each other without giving rise to spurious infinities in the computations. The algorithm for the space charge forces correctly models the mutual force between two discs as increasing as the discs approach each other, up to the point where they just touch, and then decreasing as they interpenetrate, becoming zero when the discs are coincident. Hard particle models have problems with infinities when two charges coincide, and these problems are purely artifacts of the computation: in the real world of electron beams, the distance between electrons is so enormous relative to their size that individual electron-electron collisions essentially never occur. It is only when we try to model the 10^8 or so real electrons in one beam wave length by a mere 100 or so charges in the model (to keep the problem within the capability of a computer) that the problem appears, and then has to be dealt with in the manner described above.

The number of charges per wavelength (N) is an important parameter: it must be large enough to provide a realistic model of the interaction, but no larger, since the running time of a computer program will be nearly proportional to N. Prior programs have generally used N = 24 or 32. Tests made with a predecessor of RELMOD9 allowed N to vary from 24 to 1024. The results (Figure 5.2-1) showed that there were significant changes in going from 24 to 96, a slight further change at 128, and no significant further changes on out to N = 1024. As a result of this, we have provisionally settled on N = 96 as the most cost-effective value. Professor Onodera, working at Stanford earlier in 1993, came independently to the same conclusion.

The injection times for the charged particles are derived from the equation

$$x^n + y^n = 1 \tag{1}$$

which, for n > 2, represents a "squared circle" -- i.e., for large n it appears like a round-cornered square. By inverting one quadrant and attaching it to the next, we obtain a curve such a Figure 5.2-2, which allows us to pick equal intervals along the x axis and get corresponding y values which are bunched, but still have some outliers. These outliers are useful because we want to have most of the charges in a bunch of controlled tightness, with a few in the skirts of the distribution so that we can see what happens to the unfavorably-phased electrons, of which there will always be some in any real klystron. The distribution based on (1) always gives us one charge or particle in exactly the wrong phase, so that we get an estimate of the maximum velocity of any exiting electron, which is useful information for collector design and x-ray studies. (In general, the ***exactly*** antiphase electron will not be the most-accelerated electron, but it is a good approximation to it.)

The relation between the exponent n and the equivalent density modulation index dmi is given by the empirical equation

$$n = 1 + 0.5\, dmi + 0.125\, dmi^4 \tag{2}$$

and the corresponding y value is then given by

$$\begin{aligned} y &= 0.5 - (0.5^n - x^n)^{1/n} \qquad (0 \le x \le 0.5) \\ &OR \\ y &= 0.5 + (0.5^n - (1-x)^n)^{(1/n)} \qquad (0.5 \le x \le 1) \end{aligned} \tag{3}$$

for the two halves of the curve.

For dmi = 2, this gives a very tight bunch, as shown in Figure 5.2-3. The outliers appear as the triangular blips along the baseline. The number of them, for N = 96, is rather minimal for giving information on out-of-phase electrons, but this is not an argument for increasing N: no real klystron is going to produce as nearly-perfect a bunch as is shown in Figure 5.2-3, so there will be far more outliers when we get to computation of an actual tube design.

5.2.2 Induced current and voltage

If the bunch shown in Figure 5.2-3 were passed through a short-circuited gap (i.e. no rf voltage) with no space charge forces, it would emerge unchanged; the induced current in the gap would be of the form shown in Figure 5.2-4: the induced current pulse is much wider then the bunch because it is being induced as long as the bunch is in the gap. The current pulse shoulders are rounded because we have intentionally used a beam model with outliers in it. The effective width of the current pulse corresponds to the gap transit angle (in this case approximately 150°).

If we now remove the short circuit, allowing the gap to form part of a cavity with appropriate Q and other parameters (to be discussed later), an rf voltage will be developed as the product of the induced current and the gap impedance. By Lenz's law this voltage will be in such a phase as to retard the beam. At the high dmi we have considered this voltage will be near or at saturation, comparable to or somewhat larger than the beam dc voltage. There will therefore be a major reduction of beam velocity, and a corresponding reduction of induced current (since this is proportional to the charged particle velocity). The calculation therefore has to be repeated with the new current, and iterated until the voltage and current converge to mutually consistent values. When this has been done, we obtain the highly distorted current waveform shown in Figure 5.2-5. To extract useable information from this, we need to Fourier-analyze it and select the fundamental-frequency component for multiplication into the cavity impedance. Clearly, there is a lot of harmonic content in Figure 5.2-5, and the Fourier-analysis routine does provide data on the first 5 harmonics, but at present we cannot make any use of the information for higher than the fundamental because we have no information on the cavity impedance at these harmonic frequencies. This is a problem for future study.

Next we can short-circuit the gap again, but allow the space-charge force to be calculated and included. The result is shown in Figure 5.2-6: the "step" in the middle of the pulse reflects the fact that the bunch of Figure 5.2-3 is now blowing itself apart. The front half of the bunch is accelerated, thus inducing more current, and the rear half is retarded, inducing less current.

Finally, we unshort the gap, still retaining the space-charge forces, and obtain Figure 5.2-7. We note that this has much more similarity to Figure 5.2-5 than to Figure 5.2-6, reflecting the fact that the rf forces dominate the space charge forces even when the bunch is very tight. Finally, Figure 5.2-8 shows the waveform of Figure 5.2-7 broken down into its first five harmonic components, together with the resulting rf voltage. The latter appears as a pure sine wave because it is derived only from the fundamental component of the induced current.

Knowing the value of the rf voltage, we can then determine the power dissipated in the cavity, and in any load connected to the cavity, by using the normal circuit-theory formulas. Note that we do ***not*** treat loading due to the beam by Feenberg's or any other formulas. This loading has already been taken into account in the process of converging on a self-consistent voltage and induced current.

5.2.3 The space-charge model

As we have seen, the space-charge effect is relatively small compared to the induced voltage, but it tends to take up a ***major*** part of the computation time if not done by an efficient method. The Green's function approach is an example of a method that is impeccably correct from an academic standpoint, but is computationally terribly inefficient: its running time is proportional to N^2.

Instead, we use an approximate method which introduced a small error (in what we have shown is already a minor component of the forces involved), but has a running time proportional to N log N. We further reduce the time by using a 24-nodes-per-wavelength model for the space charge while retaining 96 nodes per wavelength for the induced current. At each time step, the charge of each disc is apportioned to the two nodes on either side of the actual disc position; the proportion is the inverse of the distance of the disc from each node. This converts the space charge distribution from one of equal charges at unequal positions into an almost-equivalent one of unequal charges at uniformly-spaced positions. We can then use formulas given by Hechtel (see J. R. Hechtel, "The Effect of Potential Beam Energy on the Performance of Linear Beam

Devices," IEEE Trans. Electron Devices, ED-17 #11, Nov. 1970, pp. 999-1009) or others to obtain the voltages at these nodes. Finally, the field strength at the actual position of a disc is found by identifying the three nearest nodes, at which the potentials are P_1, P_2, and P_3; the field is then

$$E_{sc} = \{0.5(P_3 - P_1) + x\ (P_1 - 2P_2 + P_3)\}\ /\ (\lambda_e/24) \tag{4}$$

where x is the distance (normalized to the node spacing $\lambda_e/24$) of the disc from node P_2 ($-0.5 < x \leq 0.5$).

This algorithm has been in use for many years (see J. R. M. Vaughan, "Calculation of Coupled-Cavity TWT Performance," IEEE Trans. Electron Devices, ED-22 #10, October 1975, pp. 880-890) and is well tested. The description was included here for completeness. It is an example of using the gradient of a potential, rather than an inverse square law, to obtain the force on a particle; the potential approach is not only a more elegant method for many physics problems, but is commonly much more efficient computationally.

5.2.4 Relativistic motion of the disc

The foregoing sections have shown (in somewhat general terms) how the rf field and space-charge fields are evaluated. The sum of these is the total electric field acting on the disc. Calculation of the resulting disc motion has traditionally been by one of two methods:

i) Integrate the relativistic equations-of-motion by numerical methods (Runge-Kutte, etc.) using a considerable number of sub-steps.

ii) Using the Newtonian equations-of-motion with a relativistic mass $\gamma^3 m_o$.

Of these two, the first is valid, but abominably slow, the second is fast but inaccurate, since the value of γ changes during the step (unless the step is very short).

The following formulation, derived recently at Litton, takes account of the variation of γ during the step, and is only slightly slower than (ii):

take a time step Δt satisfying

$$\Delta t < 0.5\times10^{12}\ v_1/E \tag{5}$$

where v_1 is the velocity at the start of a step, and E is the electric field ($E_{rf} + E_{sc}$) evaluated at the mid-time and projected mid-point of the step. Let

$$f = \frac{e}{m} E\ \Delta t\ . \tag{6}$$

Then the velocity v_2 at the end of the step is
where γ_1 is the γ corresponding to the known v_1. The distance traveled is then

$$v_2 = v_1 + f/\{\gamma_1^2 (\gamma_1 + 1.5\ v_1 f/c^2)\} \tag{7}$$

$$\Delta z = 0.5\ (v_1 + v_2)\ \Delta t \tag{8}$$

These formulas are not analytically exact, but they are accurate to better than 0.01% as long as (5) is satisfied. In the program a fixed Δt equal to 1/96 of the rf period is used. This satisfies (5) by nearly an order of magnitude, except for a disc which is almost stopped. If this happens, the formulas do not become invalid, but the errors might creep up to the 0.1% level for a few steps. The limitation (5) is not at all severe: it allows steps in which the velocity change is as much as 2.5%. A numerical integration method would require *very* much shorter steps to achieve the same accuracy.

The formulas are valid up to beam voltages of a few MeV, which more than covers all foreseeable klystron designs. They require rethinking for the many-MeV accelerator region.

5.2.6 The interaction

To perform the interaction calculation, we begin with a gap voltage obtained from analytical klystron theory. The beam motion is calculated by the formulas of section (4) under this voltage. The induced currents are summed and Fourier analyzed to find the fundamental component. The induced voltage is found by multiplying this current into the cavity impedance at the rf drive frequency. The process is iterated with the new voltage until convergence is achieved (typically about 10 iterations, taking less than two seconds each on a MicroVax computer, with a 96-disc model).

When convergence has been reached, we know

a) the power dissipated in the cavity and load

b) the change in K.E. of the beam, from the differences of entry and exit velocities

c) the change in P.E. of the beam, from the potential depressions at the nodes of the space charge algorithm.

The last of these has proven to be considerably smaller than we had expected: the decrease in P.E. due to space-charge debunching of the beam seems to be about compensated by the increase in P.E. that results from the beam slowing down. The net change, in cases run so far, is about 2 orders of magnitude less than (a) or (b), which may justify ignoring it in most cases. However, the mechanism for calculating it is in place in the program.

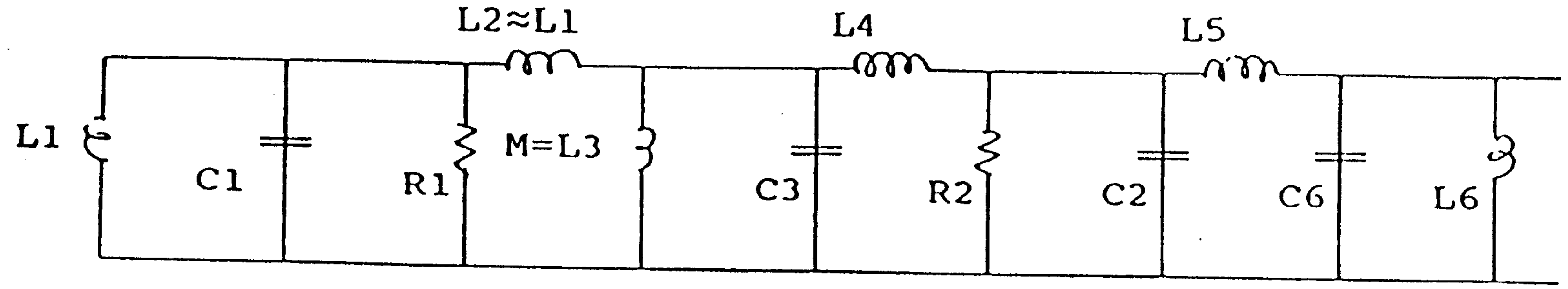
Figure 5.2-9 shows the comparison of (a) and (b) across a 10% band for a case corresponding approximately to a single-gap output cavity for the L-5792 klystron. The agreement is considerably closer than we have been able to obtain with any prior program. Inclusion of (c) does

not reduce the remaining discrepancies: it merely shifts them around. Figure 5.2-10 shows part of the printout of a typical test case, showing how the rf voltage amplitude and phase, the induced current amplitude and phase, and the P.E. change, all converge in 10 iterations to almost-constant values. The following lines show the degree of agreement between the generated power and the ΔK.E. of the beam. Figure 5.2-9 was derived from multiple runs of this kind.

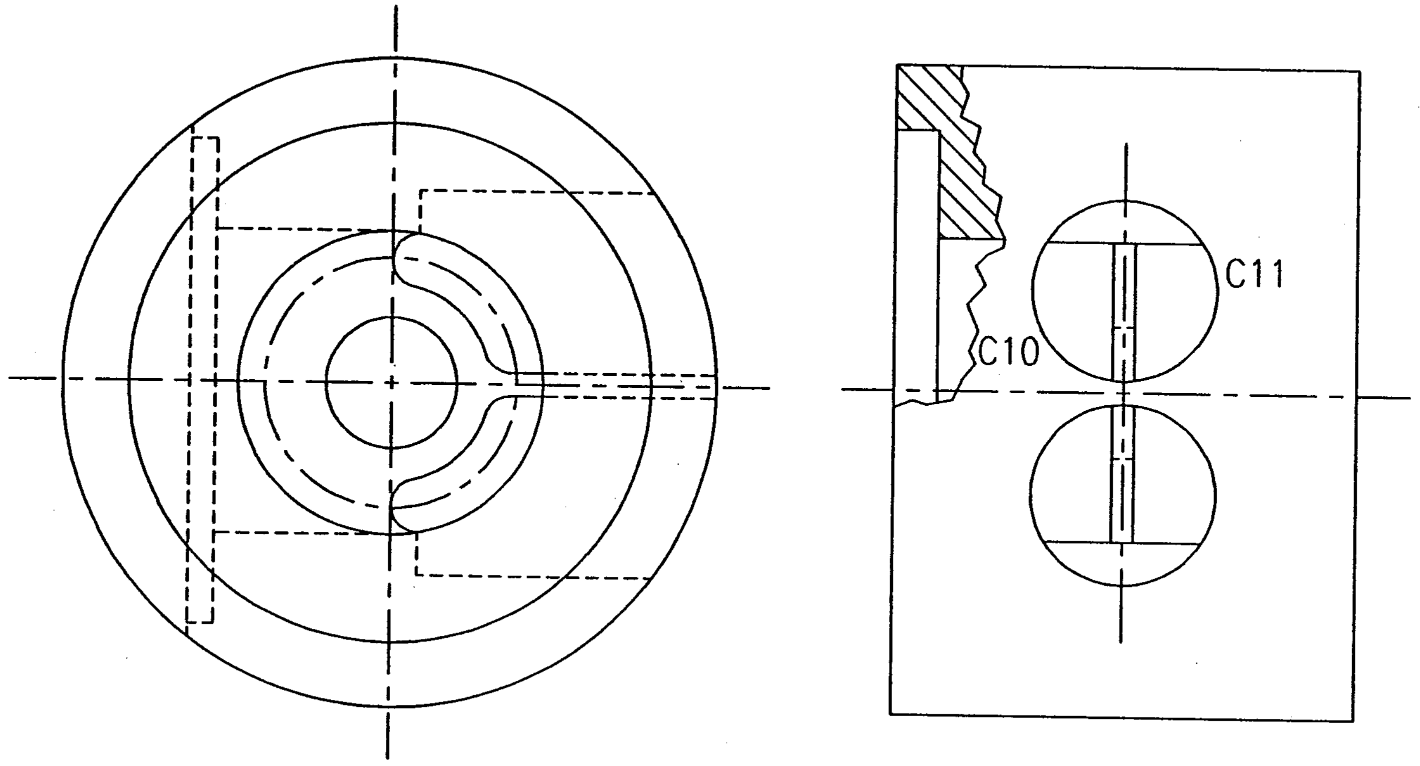
As noted earlier, the beam loading is being calculated dynamically, not from Feenberg's formulas. We find that, at saturation, the beam loading can be as much as 10 times higher than the Feenberg value, corresponding to a sharply reduced beam-loading Q. The Q_{BL} is no longer constant, but depends on the degree of modulation. At full intensity, Q_{BL} is about half the Feenberg Q_{BL}. This results in a response curve that passes through a maximum and decreases if the Q_E (external loading) value is too high. This is in agreement with observation, but was not found in prior programs. From these observations we have derived a formula for the optimum value of Q_E:

$$Q_E = 1/\sqrt{\left\{\frac{2\Delta f}{fo}\right\}^2 + \left\{\frac{(dmi)\ (R/Q)\ (1 - vmi \sin \psi)}{(V_{Idc})\ (1.42\ \theta - 0.112\ \theta^2)}\right\}^2}$$

where 2Δf is the bandwidth.



(A) Equivalent Circuit



(B) Cold-test Model

Figure 5.1-1
TWO-CAVITY EIOC

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0   I2 = 30.0 Amp
TransitTime  = 227.0 picoSec (Fixed)
Phase @ F1   =  2.815 * PI/2
Phase @ F2   =  3.269 * PI/2
Fslot        =  8.010 GHz
Firis        =  3.121 GHz
Iris-Post    =  6.331 Inch
Post-Ref     =  0.000 Inch
Displacemnt  =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.47
R1 = 20000  R2 = 20000 Ohm
THIS ERROR INDEX = 1.9700 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GB2051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 07:56:06
```

B

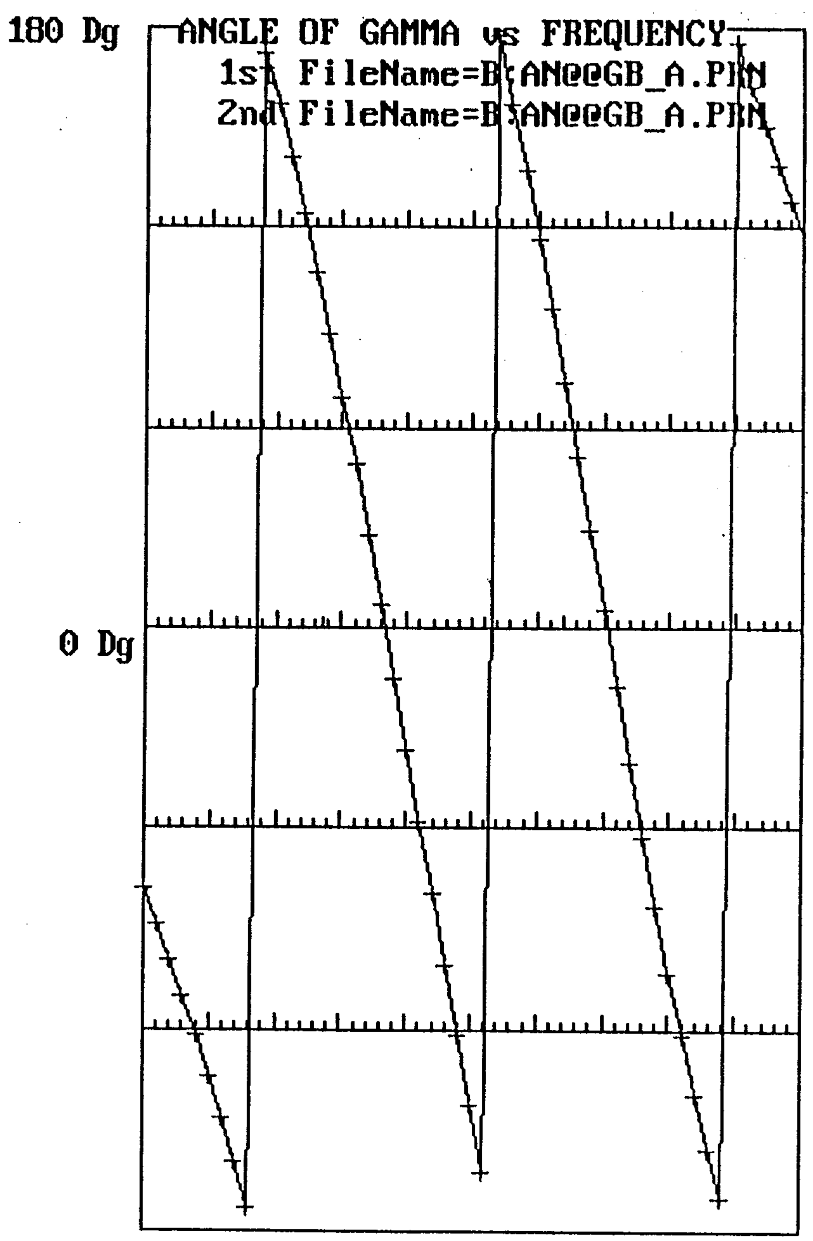


Figure 5.1-2
EIOC - CONFIGURATION 'A'
TWO GAPS OPEN
(A) Equivalent circuit data
(B) Phase vs. frequency for model predictions (plain line)
and cold-test measurements (hashed line)

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0    I2 = 30.0 Amp
TransitTime   = 227.0 picoSec (Fixed)
Phase @ F1    =  2.815 * PI/2
Phase @ F2    =  3.269 * PI/2
Fslot         =  8.010 GHz
Firis         =  3.121 GHz
Iris-Post     =  6.331 Inch
Post-Ref      =  0.000 Inch
Displacemnt   =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.47
R1 =     1  R2 = 20000 Ohm
THIS ERROR INDEX =13.4856 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GB2051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 07:59:46
```

B

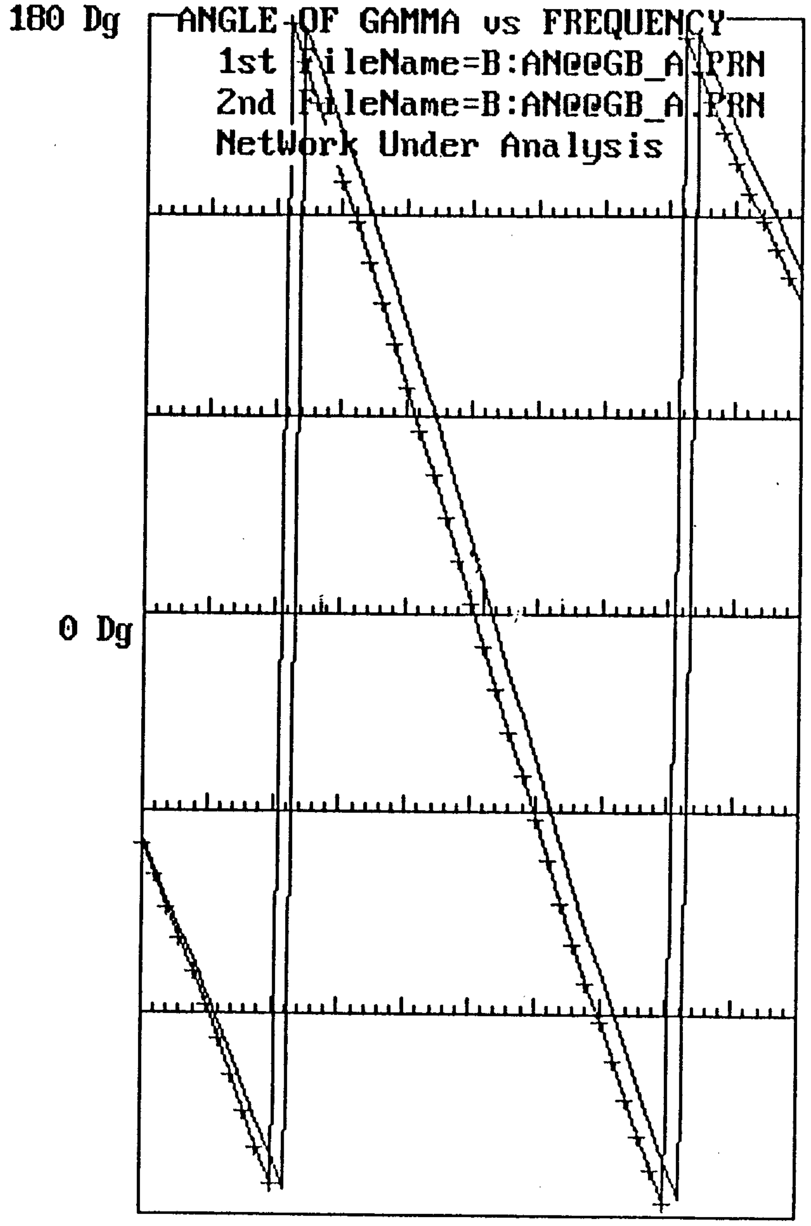


Figure 5.1-3
EIOC - CONFIGURATION 'A'
FIRST GAP SHORTED, SECOND GAP OPEN
(A) Equivalent circuit data
(B) Phase vs. frequency for model predictions (plain line)
and cold-test measurements (hashed line)

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0   I2 = 30.0 Amp
TransitTime  = 227.0 picoSec (Fixed)
Phase @ F1   =  2.815 * PI/2
Phase @ F2   =  3.269 * PI/2
Fslot        =  8.010 GHz
Firis        =  3.121 GHz
Iris-Post    =  6.331 Inch
Post-Ref     =  0.000 Inch
Displacemnt  =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.47
R1 =     1  R2 =     1 Ohm
THIS ERROR INDEX =11.8255 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GB2051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 08:02:20
```

B

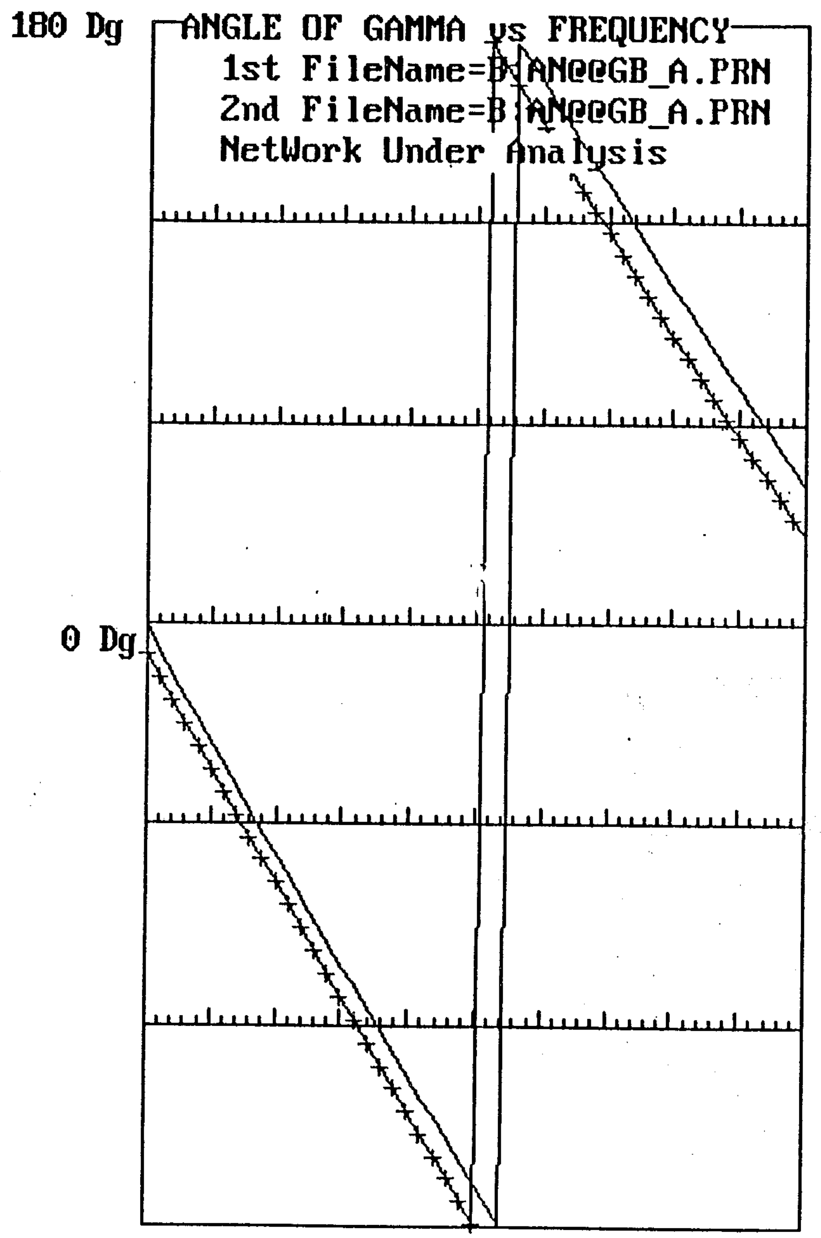


Figure 5.1-4
EIOC - CONFIGURATION 'A'
BOTH GAPS SHORTED
(A) Equivalent circuit data
(B) Phase vs. frequency for model predictions (plain line)
and cold-test measurements (hashed line)

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0   I2 = 30.0 Amp
TransitTime  = 227.0 picoSec (Fixed)
Phase @ F1   =  2.815 * PI/2
Phase @ F2   =  3.269 * PI/2
Fslot        =  8.010 GHz
Firis        =  3.124 GHz
Iris-Post    =  6.331 Inch
Post-Ref     =  0.000 Inch
Displacemnt  =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.49
R1 =     1  R2 = 20000 Ohm
THIS ERROR INDEX =13.5687 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GB2051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 08:27:51
```

B

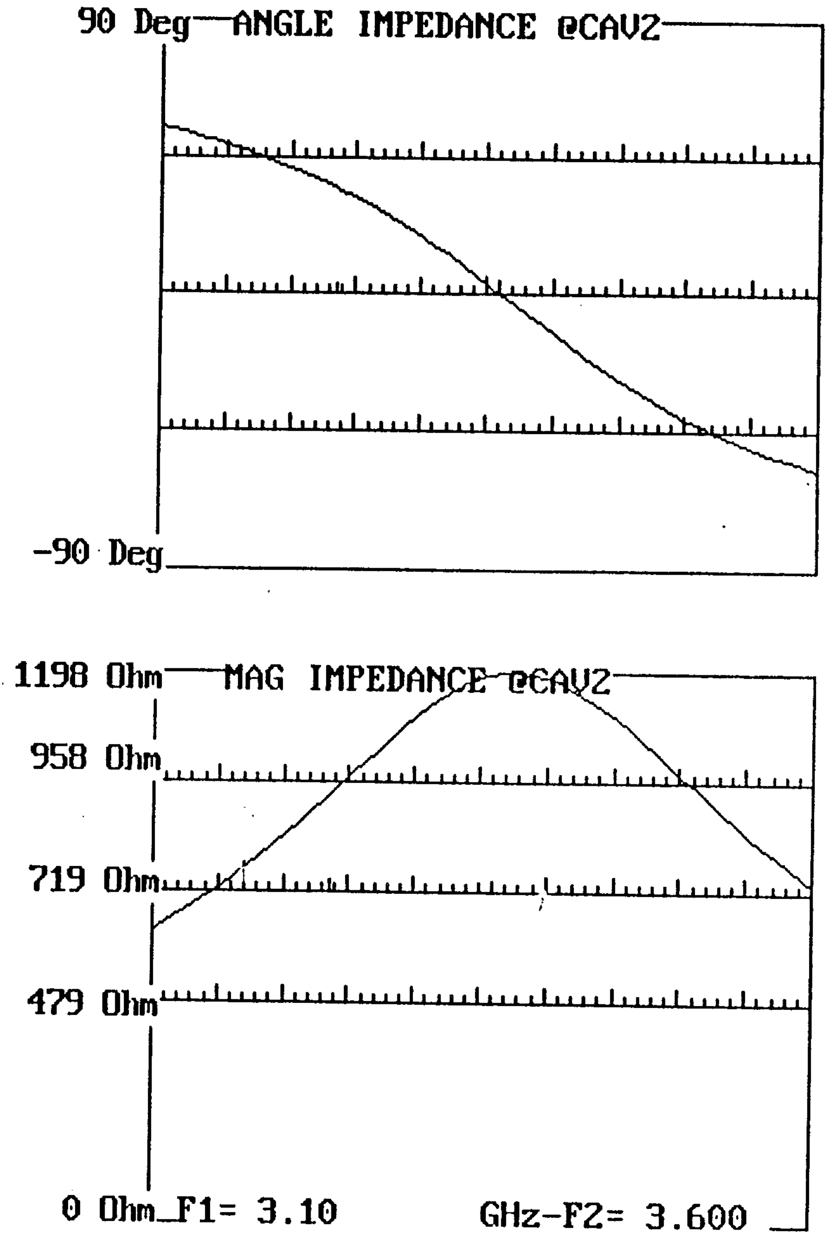


Figure 5.1-5
EIOC - CONFIGURATION 'A'
FIRST GAP SHORTED, SECOND GAP OPEN
(A) Equivalent circuit data
(B) Model-predicted impedance vs. frequency

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0   I2 = 30.0 Amp
TransitTime  = 227.0 picoSec (Fixed)
Phase @ F1   =  2.815 * PI/2
Phase @ F2   =  3.269 * PI/2
Fslot        =  8.010 GHz
Firis        =  3.121 GHz
Iris-Post    =  6.331 Inch
Post-Ref     =  0.000 Inch
Displacemnt  =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.47
R1 = 20000  R2 = 20000 Ohm
THIS ERROR INDEX = 1.9700 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GBZ051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 08:11:59
```

B

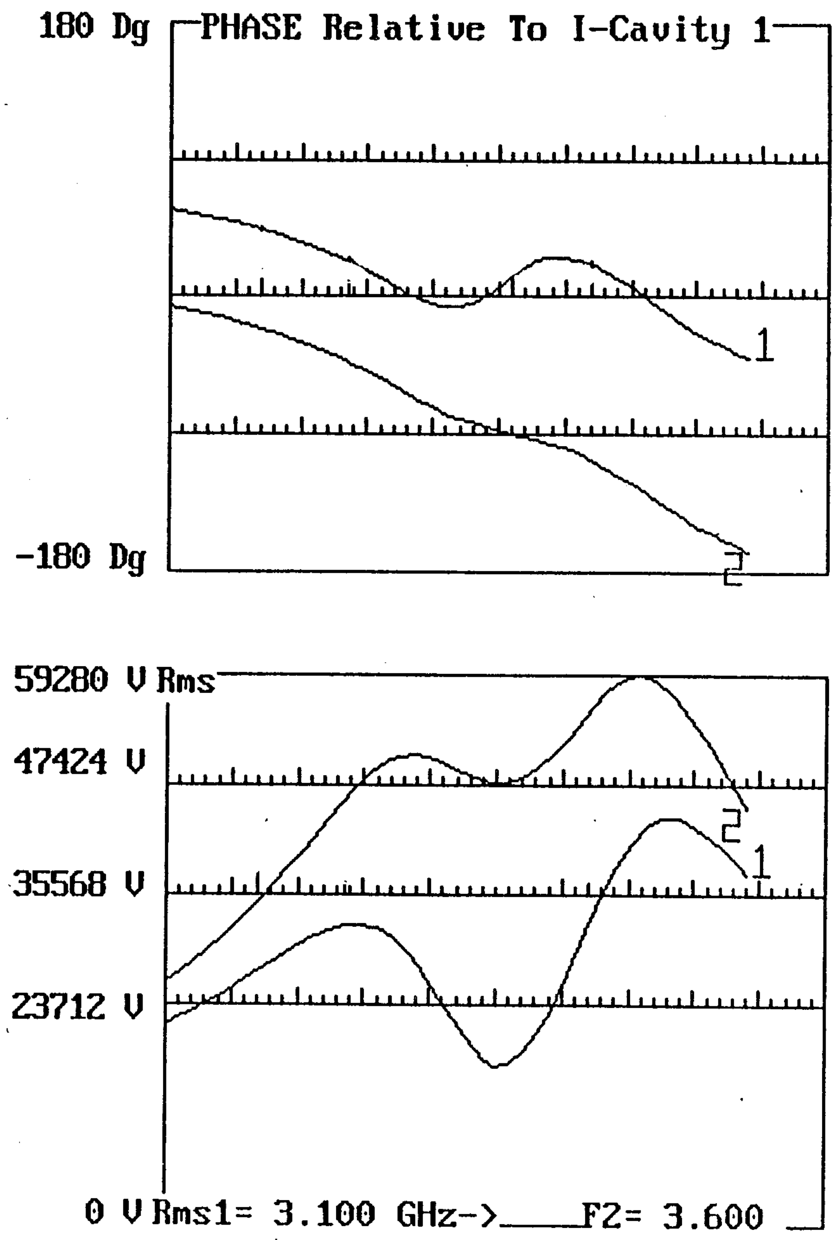


Figure 5.1-6
EIOC - CONFIGURATION 'A'
BOTH GAPS OPEN
(A) Equivalent circuit data
(B) Model-predicted voltage vs. frequency for both gaps

A

```
L1 = 10.413, L2 =  9.526 nHy
L3 = 0.706, L4 =  5.662 nHy
L5 = 23.713, L6 = 10.343 nHy
C1 = 0.432, C2 = 0.432 pFd
C3 = 0.559, C6 = 0.251 pFd
I1 = 30.0   I2 = 30.0 Amp
TransitTime  = 227.0 picoSec (Fixed)
Phase @ F1   =  2.815 * PI/2
Phase @ F2   =  3.269 * PI/2
Fslot        =  8.010 GHz
Firis        =  3.121 GHz
Iris-Post    =  6.331 Inch
Post-Ref     =  0.000 Inch
Displacemnt  =  0.000 Inch
Rs1/Q = 110  Rs2/Q = 110 Ohm
Zwgm = 213 Ohm, Qext = 11.47
R1 = 20000  R2 = 20000 Ohm
THIS ERROR INDEX = 1.9700 Dg RMS
1stAngleFile = B:AN@@GB_A.PRN
2ndAngleFile = B:AN@@GB_A.PRN
G+iB FILE    = B:GBZ051P.PRN
PROGRAM FILE = OC_FEB18.BAS
DATE> 03-01-1995. TIME> 08:08:58
```

B

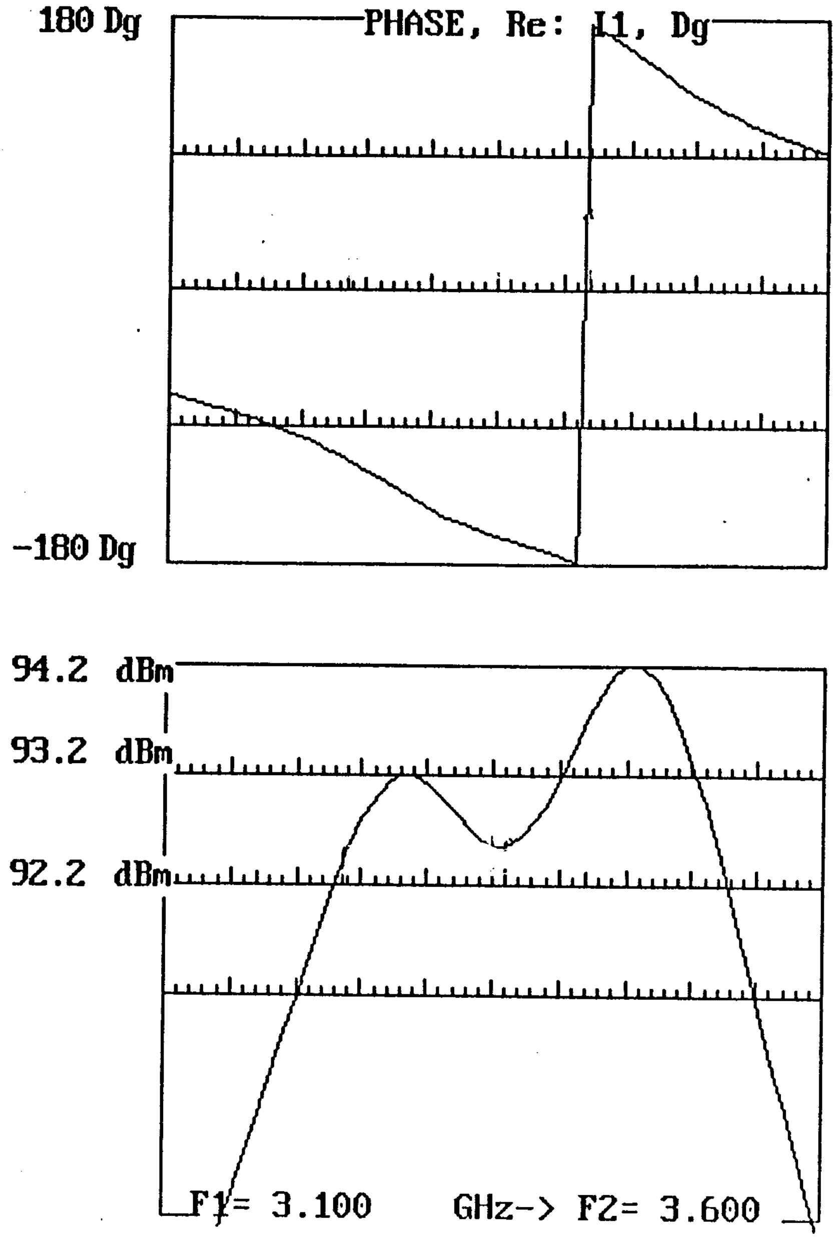


Figure 5.1-7
EIOC - CONFIGURATION 'A'
BOTH GAPS OPEN
(A) Equivalent circuit data
(B) Model-predicted linearized output power vs. frequency

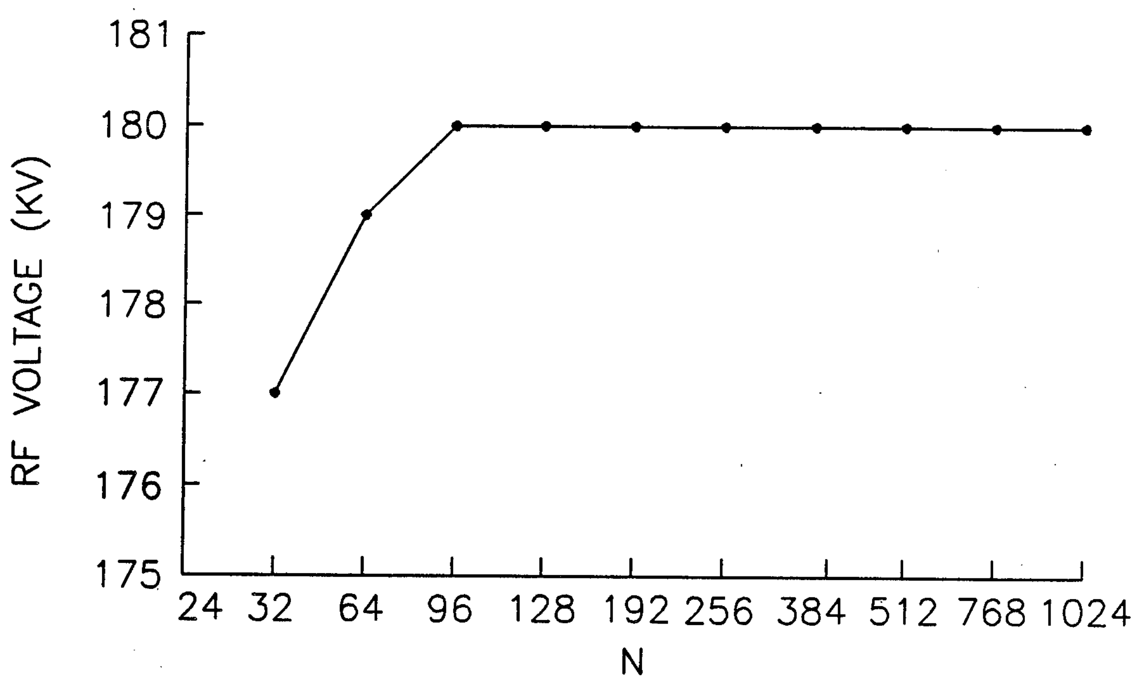


Figure 5.2-1
COMPUTED RF VOLTAGE VS. NUMBER OF DISCS

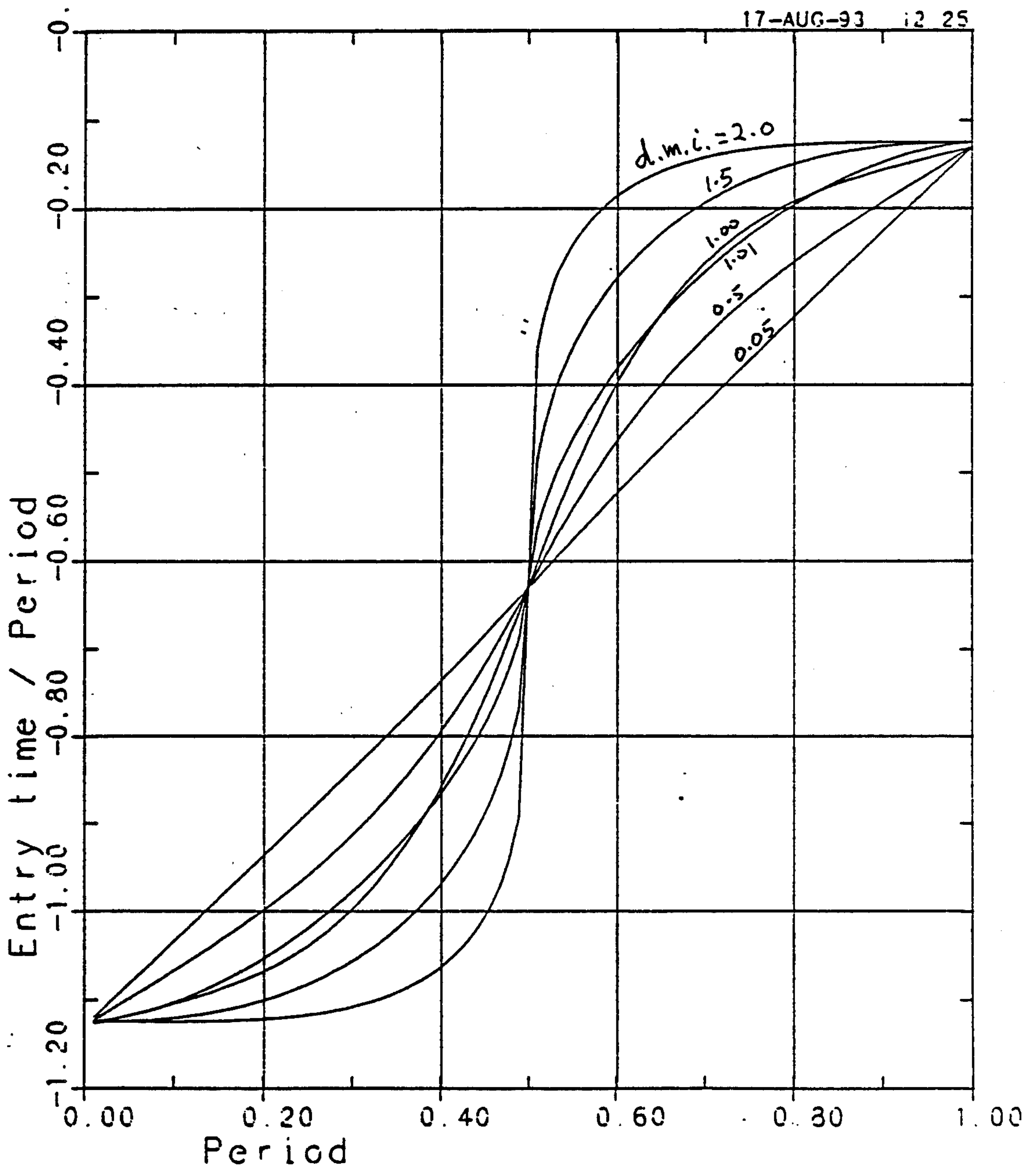


Figure 5.2-2
SQUARED CIRCLE MODEL FOR INITIAL BUNCHING

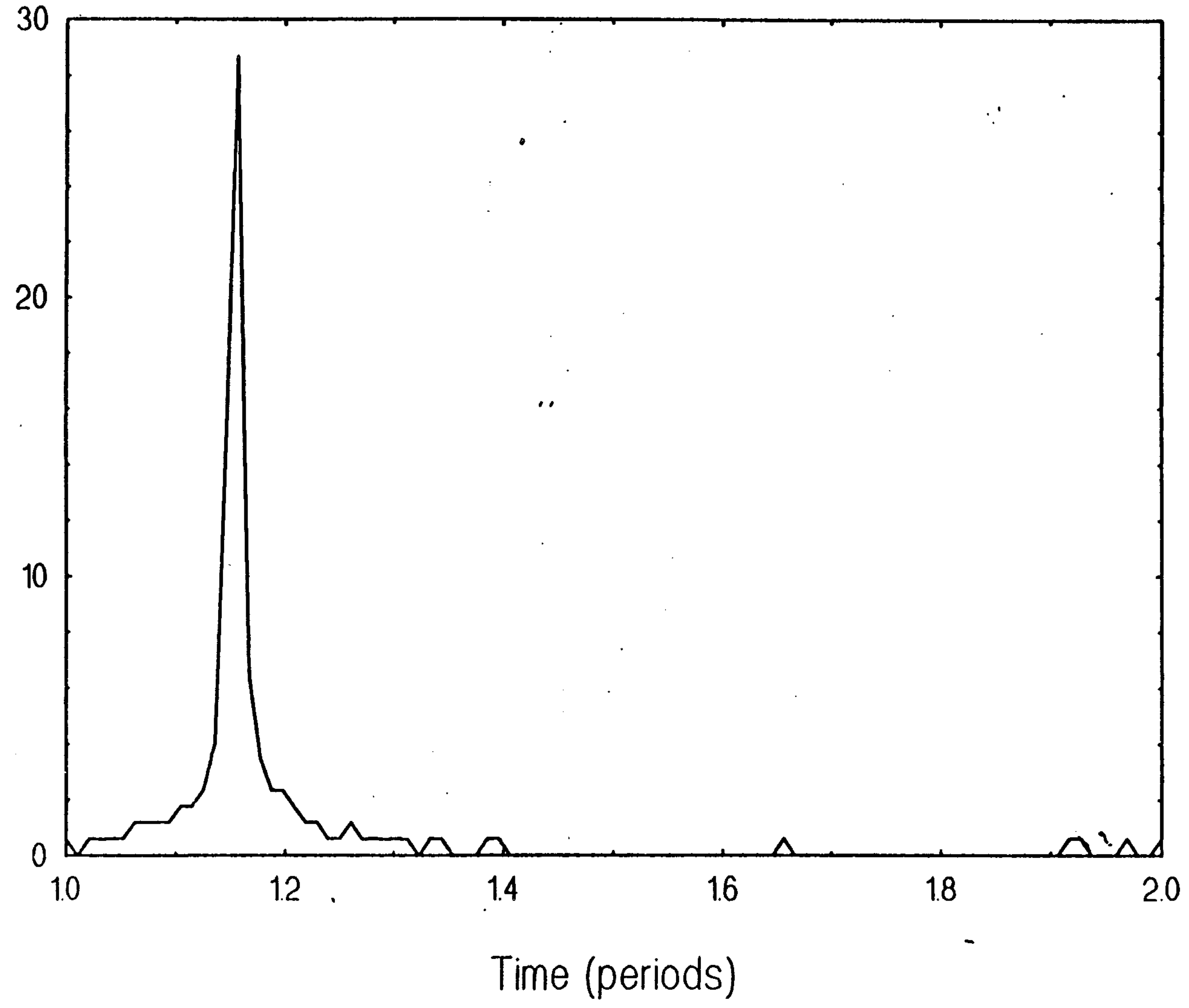


Figure 5.2-3
INJECTED BUNCH

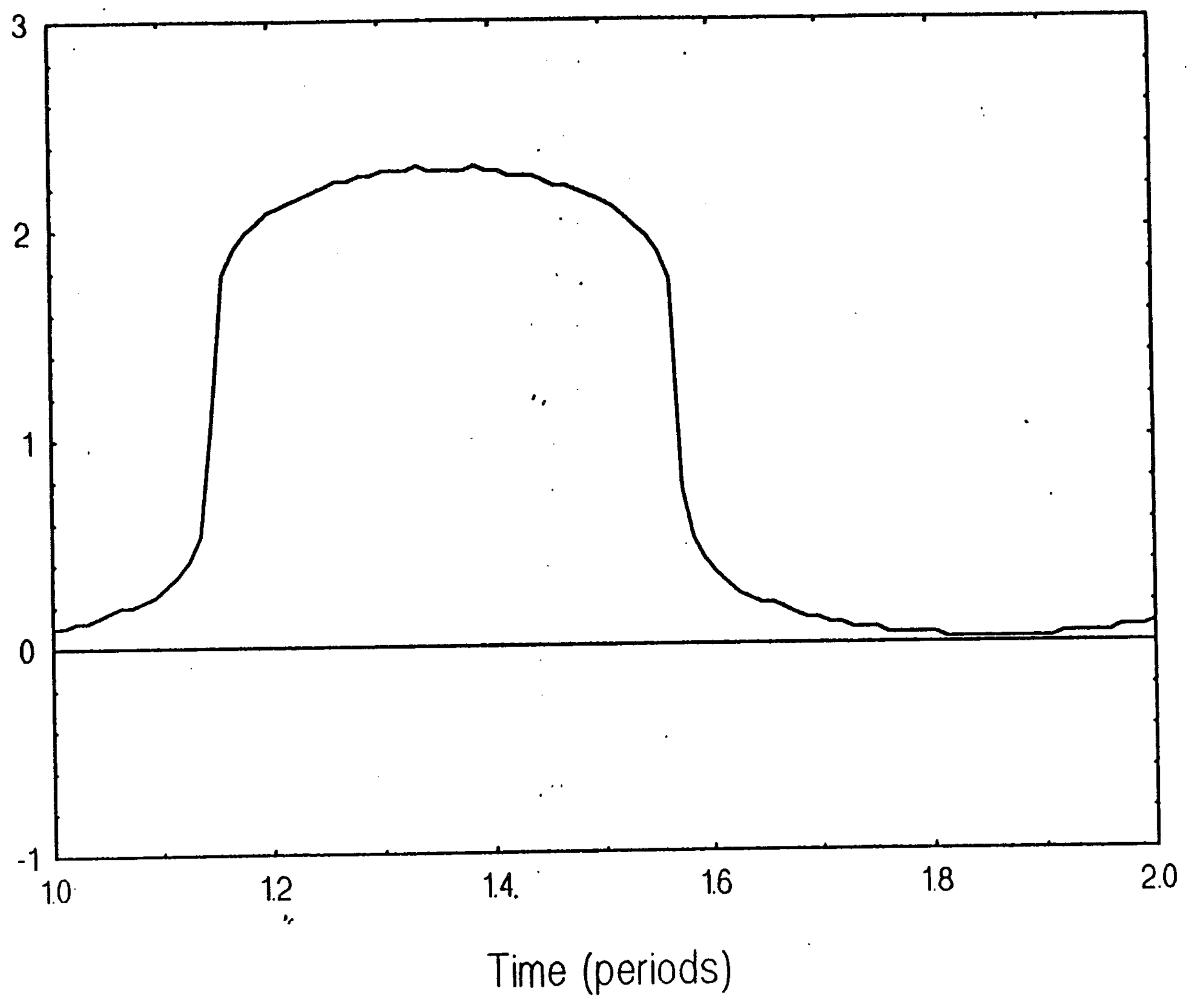


Figure 5.2-4
INDUCED CURRENT - NO RF, NO SPACE CHARGE

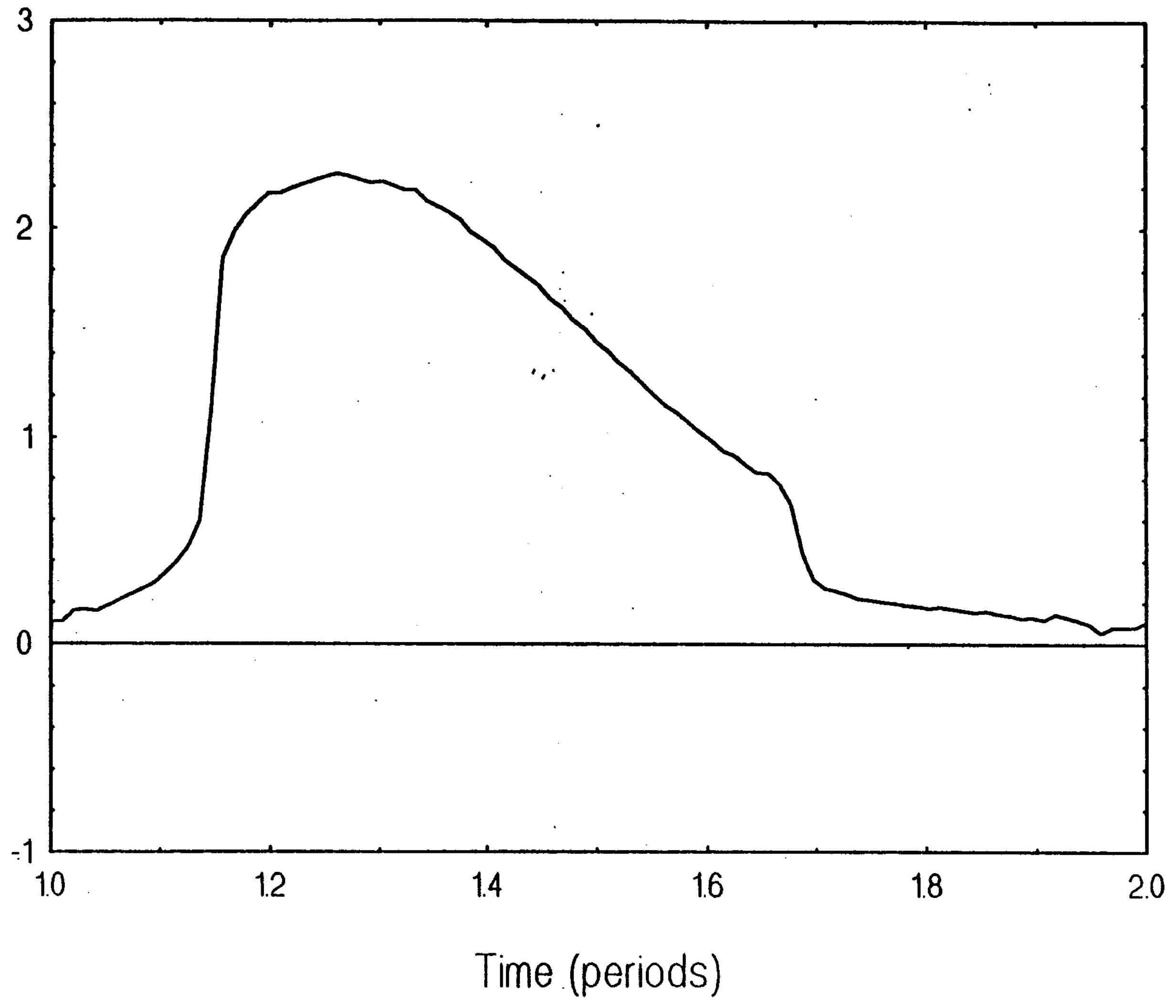


Figure 5.2-5
INDUCED CURRENT - RF, NO SPACE CHARGE

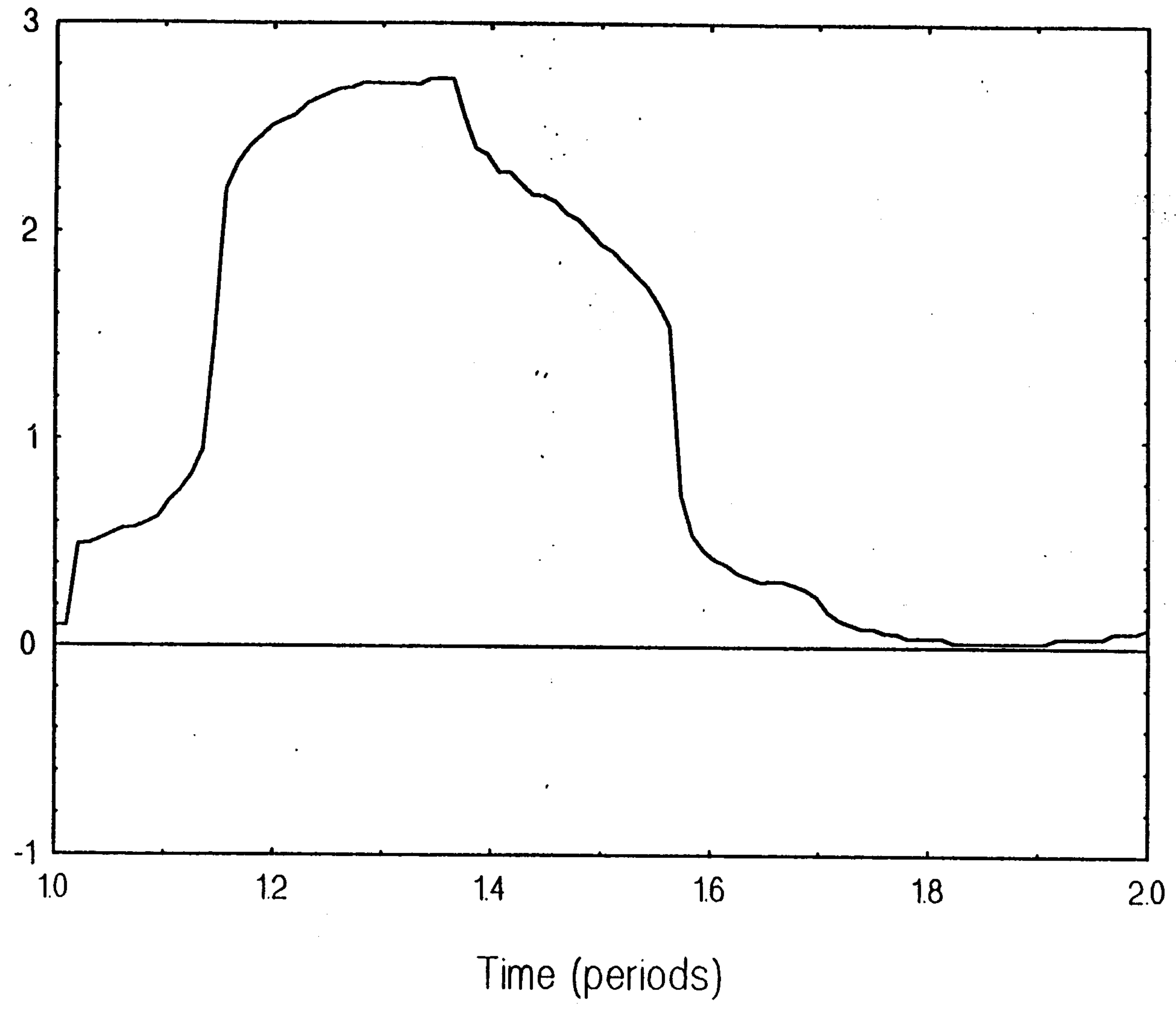


Figure 5.2-6
INDUCED CURRENT - SPACE CHARGE, NO RF

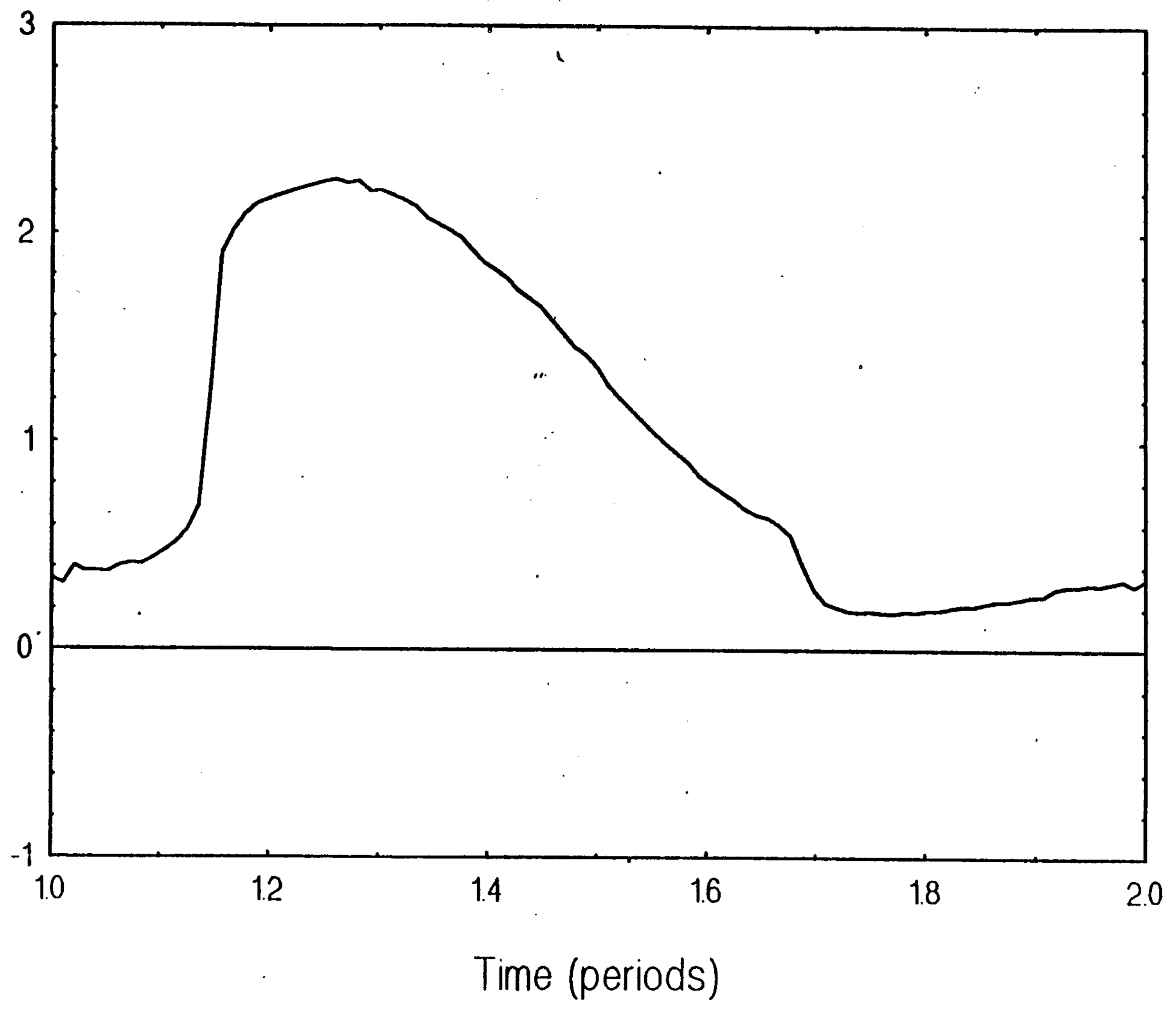


Figure 5.2-7
INDUCED CURRENT - SPACE CHARGE AND RF

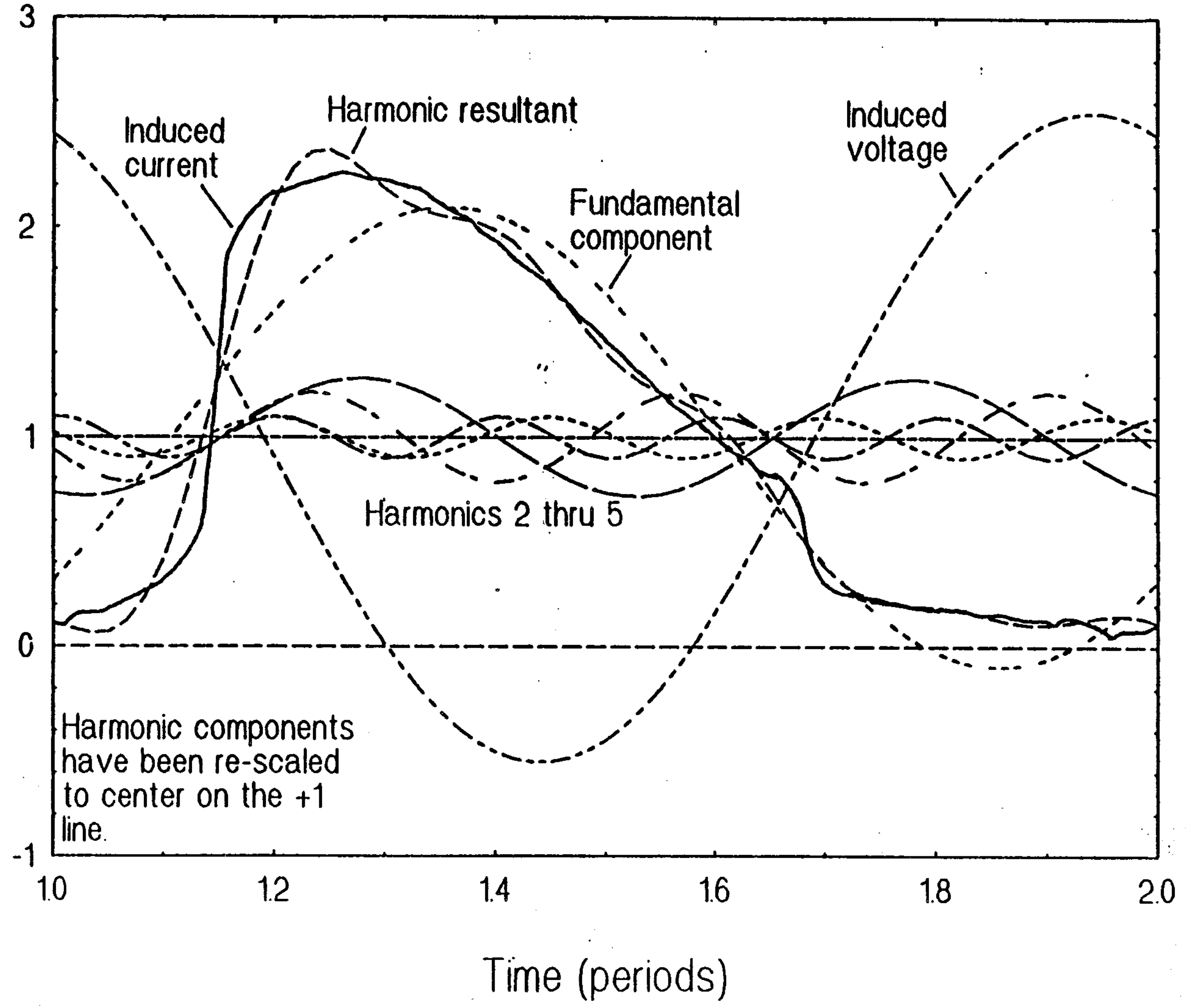


Figure 5.2-8
HARMONIC ANALYSIS OF FIG. 5.2-7

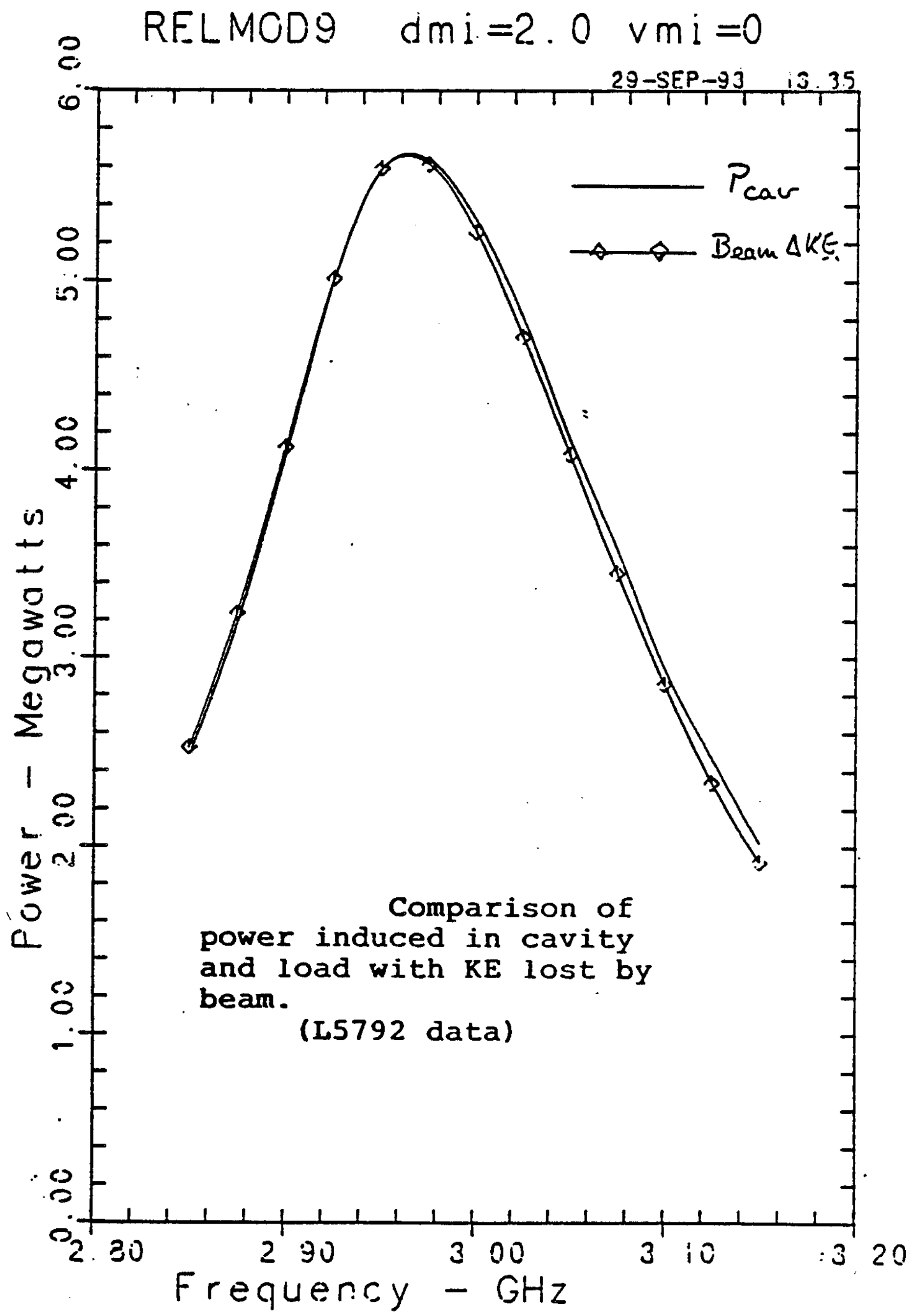


Figure 5.2-9
POWER BALANCE

```
dc transit time:  1.44154E-10 sec;  transit angle:  2.69913 r,  154.6 deg.

Next estimate of induced gap voltage:     243476.3 @ 131.68 deg.
Repeat 10 times (T), Use once (U), Enter new values (E),
          or Continue with prior value  112528.8 @  86.32 deg. (C) ? T
Vrf= 243476.3 @ 131.68,  Irf(fund)=  100.84 @ 123.39,   delta PE=  5.3523E+04
Vrf= 198287.5 @ 148.77,  Irf(fund)=   63.41 @  89.22,   delta PE=  5.5519E+04
Vrf= 158289.8 @ 143.26,  Irf(fund)=   48.99 @ 117.32,   delta PE=  4.4735E+04
Vrf= 159787.5 @ 135.62,  Irf(fund)=   66.80 @ 127.09,   delta PE=  5.4006E+04
Vrf= 163829.4 @ 133.01,  Irf(fund)=   69.53 @ 124.66,   delta PE=  5.2758E+04
Vrf= 166017.5 @ 132.36,  Irf(fund)=   69.66 @ 123.36,   delta PE=  5.5127E+04
Vrf= 166587.1 @ 132.33,  Irf(fund)=   69.23 @ 122.76,   delta PE=  4.7644E+04
Vrf= 166693.5 @ 132.39,  Irf(fund)=   69.08 @ 122.63,   delta PE=  4.7903E+04
Vrf= 166657.4 @ 132.43,  Irf(fund)=   69.01 @ 122.59,   delta PE=  4.7702E+04
Vrf= 166591.6 @ 132.44,  Irf(fund)=   68.97 @ 122.62,   delta PE=  4.6506E+04
Beam power at entry: 9.06700E+06 Watts;    delta: -5.58308E+06 Watts
         at  exit: 3.48393E+06      Cavity loss:  5.55055E+06

Next estimate of induced gap voltage:     166602.4 @ 132.46 deg.
Repeat 10 times (T), Use once (U), Enter new values (E),
          or Continue with prior value  166591.6 @ 132.44 deg. (C) ? C

96 point Fourier analysis (-99.9 is code for indeterminate):
                         dc        fund.      2nd        3rd        4th        5th
 cosine terms:  79.71798 -40.91079 -24.41533  -6.89699   4.96995   7.96750
 sine terms:                55.56426 -13.06102 -14.18394  -8.23575   0.27801
 amplitudes:                69.00057  27.68932  15.77190   9.61915   7.97235
 phases (deg):                122.61   -159.36   -127.18    -73.89    -16.75
 dB:                           -1.25     -9.18    -14.07    -18.37    -20.00

M =  0.722920  N =  0.829323   Feenberg g/g0 =  0.181991  b/g0 =  0.040932
Max induction =  190.02         Fourier g/g0 = -0.596329  b/g0 = -0.118022
Av. delta K.E.:   -58157.        equiv. g/g0 = -0.5884     g/gFe =  -3.233
Vel. range in gap: 0.1484087 to 1.2885508 * U0,   # of stopped discs =   0
Exit vel. range: 0.5029183 to 1.2885508 * U0,   gFourier/gF'berg =  -3.27669
```

Convergence of rf voltage and current to consistent values. When converged, their ratio is equal to the (complex) cavity impedance. The near-equality of the delta KE and the Cavity loss is an independent check on the accuracy (i.e., it was not used in obtaining the convergence). The Fourier analysis of the final current waveform follows; the dc term 79.71798 should be 80.0 Amp, the beam current, if there were no approximations. The last two lines show that a disc was slowed to under 15% of its initial velocity at some point in the gap, but was then reaccelerated to just over 50% at the exit. The maximum velocity, on the other hand, occurs at the exit plane.

Figure 5.2-10
CONVERGENCE

6. CONCLUSION

A generic expert system has been developed and applied to a specific klystron. The heart of the expert system, the software, was developed in a general way and applied to an input cavity transformer that has two adjustments. It is a straightforward extrapolation to extend the software to monitor and control additional subsystems with numerous adjustments. In this work, three major subsystems were developed for a klystron expert system, those for monitoring and controlling the cathode temperature, the rf input response, and the rf output response.

The expert system software developed in this program (TIPTOE) was written in Fortran so that it would be easily transportable between different computing systems. It successfully demonstrated that a software system can perform the function of an expert by adjusting the hardware settings, obtaining the resultant data, and converging on the optimum hardware settings.

The software was applied to the matching transformer in the input cavity circuit of the klystron. A new matching transformer was designed for the input cavity having both variable impedance and axial position along the coaxial transmission system. The new transformer incorporated an axially sliding, eccentric design. Tests showed that the input transformer circuit, a combination of a bandpass filter and impedance transformer (simply referred to as a "transformer") broadens the bandwidth of the input cavity by enhancing the edges of the passband. The transformer has two continuously and simultaneously adjustable controls that are linked to the expert system commands. These controls are useful both for hot and cold tests. The settings of the new transformer are changed by a stepper motor which is driven by an expert system. The new input transformer and its drive fit inside the existing envelope of the klystron. The transformer contained two independent adjustments, resulting in a nontrivial system controlled by the expert system software.

The expert system formalism was developed, and the software written to easily allow expansion to several variables and several different subsystems. For developmental and initial demonstration purposes, the actual circuit hardware was replaced by a lumped-element circuit model simulated with the commercial SUPERSTAR program. The lumped-element circuit model used was demonstrated to give results equivalent to the actual hardware.

The temperature of an operating cathode in a microwave tube has a strong effect on the tubes operation. If the temperature is too low, the emitted current will be low; if the temperature is too high, the lifetime of the cathode will be reduced. Unfortunately, the cathode temperature cannot normally be directly measured in an operating klystron. Therefore, a Fortran code was developed to predict the steady-state cathode temperature from a given transient heater voltage vs. time plot.

This code modified a Litton-proprietary Quickbasic program by adding features that can automate the program with an expert system and by adding highly descriptive two-dimensional variables that quickly identify the gun assembly parts whose heat transmission histories are being calculated. Copious program comments made the code easy to read and easy to modify. Before the program can be used with a given electron gun, it should be calibrated by measuring the actual cathode temperature, as a function of time, in response to various changes in the heater voltage. The different thermal emissivities and thermal conductivities of the elements in the model are then adjusted to obtain agreement between the measurements and the code results.

The code simplifies the actual electron gun geometry into a set of nodes, made up of one or more parts, that is represented by a single temperature. Since the number of nodes is relatively small, this code can run very quickly on a personal computer. The program includes the temperature dependence of thermal emissivities and thermal conductivities of the gun-assembly materials as well as geometry factors that simplify complex thermal-resistance calculations. The program is easy to use, very fast to run, and does not require large computer memory.

A lumped-element model of a two-cavity extended interaction output cavity (EIOC) was developed with sufficient accuracy to predict measurements on a cold-test model. The EIOC model is ready for use in a large-signal interaction program. This type of cavity model is often used in large-signal programs to greatly simplify the calculations and produce a code that executes rapidly enough to be used as a tube design tool. In addition, such a large-signal code may be linked to an expert system to predict the effect of adjusting tuners in the output cavity, allowing the optimum settings to be found before moving the tuners in the actual cavity.

Another program, which models large-signal klystron interaction at a single cavity gap was developed as well. With residual discrepancy levels not exceeding 1%, the 1670-line Fortran code includes relativistic effects, velocity and density modulations, space charge effects, potential energy changes and dynamic beam loading, all under large-signal conditions. Other existing klystron programs include some but not all of these effects.