Power Quality Prediction

Based on Determination of Supply Impedance

by

Robert F. Lepard

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 17, 1996

Author_____

Department of Electrical Engineering and Computer Science
May 17, 1996

Certified by____

Professor Steven B. Leeb
Thesis Supervisor

Accepted by____

F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Power Quality Prediction Based on Determination of Supply Impedance

by

Robert F. Lepard

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 1996, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The electric utility service might be very simply modeled, looking back from an electrical connection, as a sinusoidal voltage source in series with an inductor and a resistor. At the building level in the distribution system, these impedances arise predominantly from an upstream transformer, protection circuity, and a length of cable. Harmonic currents generated by some loads flow through these impedances, creating harmonic voltages that result in a distorted voltage waveform in the electrical "neighborhood" of the harmonic source.

The impedances of the electrical service could be identified by briefly closing a capacitor across the electrical service at a precise point in the line voltage waveform. The shape and decay of the transient capacitor current in this RLC circuit can be used to estimate the line impedance. The DESIRE (Determination of Supply Inductance and Resistance) system created in this thesis identifies local utility impedances using this technique. By identifying local impedances, DESIRE assists in the prediction of local voltage waveform distortion created by "power quality offenders". This thesis describe the design, implementation and use of the DESIRE system.

Thesis Supervisor: Steven B. Leeb
Title: Carl Richard Soderberg Assistant Professor of Power Engineering

Acknowledgments

I am grateful to Intel and Tektronix for providing necessary testing equipment for this thesis.

I would like to thank my advisor, Prof. Steve Leeb for his guidance and support during the two years I've worked with him. His work ethic, engineering expertise, positive attitude, and enthusiasm toward engineering has consistantly amazed me.

I am indebted to Steve Shaw for his help in this thesis. His work was essential in the development of the load test stand and in the DESIRE system.

I would also like to thank Tom Respress for his help in the development of the user interface for DESIRE.

I would like to thank Deron Jackson for his help and advice, and for being the resident expert in so many areas.

Thank you Prof. Leeb, Steve, Deron, Tom, Ahmed, and Umair for your friendship and support over the past few years.

Most of all, I'd like to thank my family, Mom, Dad, and Lynette, for their unending love and unwavering support in everything that I have done. I could not have done this without you.

Contents

1 Introduction **10**
- 1.1 Contributions of This Work . . . 11
- 1.2 Thesis Outline . . . 12

2 Load Test Stand **14**
- 2.1 Background . . . 14
- 2.2 Implementation . . . 15
  - 2.2.1 Switch Board . . . 15
  - 2.2.2 Voltage Sensor Board . . . 16
  - 2.2.3 PC Card . . . 16
  - 2.2.4 Software . . . 18
- 2.3 Collection of Data . . . 19
- 2.4 Conclusions . . . 20

3 V-Section Collection **33**

4 Power Transmission Model **38**
- 4.1 Transformer Model . . . 38
  - 4.1.1 Ideal Transformer . . . 38
  - 4.1.2 Real Transformers . . . 41
- 4.2 Simplifying the Transformer Model . . . 42

5 Determining Supply Impedance **46**
- 5.1 Derivation of Formulas . . . 47
- 5.2 Matlab Simulation of RLC Circuit . . . 49
- 5.3 Initial Determination of Supply Impedance . . . 50

6 Documentation of Determination of Supply Inductance and Resistance System **54**
- 6.1 PC Card . . . 55
- 6.2 Software for Capacitor Card . . . 58
- 6.3 Capacitor Box . . . 59

7 Experimental Results **63**
7.1 Calibration . . . 63
7.2 Extraction of Data . . . 65
7.2.1 User Interface . . . 66
7.3 Isolation Transformer Data . . . 69
7.4 Transformer Model Adjustments . . . 70
7.4.1 Winding Capacitance . . . 70
7.4.2 Skin Effect . . . 73
7.5 DESIRE Data . . . 75
7.6 Power Quality Simulation . . . 76
7.6.1 Steady State Distortion . . . 77
7.6.2 Transient Distortion . . . 78
7.7 Analysis of Results . . . 79
7.8 Variable Torque AC Drive . . . 80

8 Conclusions **88**

A Schematic Diagrams of DESIRE **92**

B Code for Load Test Stand **96**
B.1 Interface . . . 96
B.2 Assembly Code . . . 98

C DESIRE **107**
C.1 Using DESIRE . . . 107
C.2 DESIRE Code . . . 113
C.2.1 Desire.c . . . 113
C.2.2 Header Files . . . 123
C.2.3 Rob471.c . . . 168
C.2.4 Matlab Script/Function Files . . . 172
C.2.5 Other Program Files . . . 175

D Code for Prediction of Waveforms **181**
D.1 Rfft2.m . . . 181
D.2 Pred5.m . . . 181
D.3 Predi5.m . . . 182
D.4 Predigen.m . . . 182

E V-Section Code **184**
E.1 Vsmaker.m . . . 184
E.2 Vsfiltb2.m . . . 188
E.3 Vsfiltm2.m . . . 189
E.4 Vspart1.m . . . 189

E.5 Vspart2.m 198
E.6 Vspart3.m 199
E.7 Vspart4b.m 199
E.8 Dicer2.m 201
E.9 Testvs3.m 202
E.10 Medvs.m 204
E.11 Mednum.m 204
E.12 Medfilt.m 205

List of Figures

2-1 Block Diagram of Load Test Stand 21
2-2 Frequency Multiplier Diagram 22
2-3 Edge-Sensitive Lead-lag Phase Detector 22
2-4 Turn on Transients of 25 Watt Incandescent Bulb 23
2-5 Turn on Transients of 100 Watt Incandescent Bulb 24
2-6 Turn on Transients of 200 Watt Incandescent Bulb 25
2-7 Turn on Transients of Laser Printer 26
2-8 Turn on Transients of Rapid Start Compact Florescent Bulb 27
2-9 Turn on Transients of Induction Motor 28
2-10 Turn on Transients of 486 Computer 29
2-11 Turn on Transients of Duct Fan 30
2-12 Turn on Transients of Rapid Start Bulb 31
2-13 Turn on Transients of Instant Start Bulb 32

3-1 Block Diagram of Vsmaker 36
3-2 Plots of Input Data and Vsmaker Data 37

4-1 An Ideal Transformer 39
4-2 Reflecting Impedance Through an Ideal Transformer 40
4-3 Transformer With Perfect Coupling but Finite Magnetizing Inductance 41
4-4 A Practical Transformer 42
4-5 Transformer Model Including Leakage Inductance 43
4-6 Transformer Model Including Winding Resistance 43
4-7 Circuit Model, Reflecting Impedances of Secondary and Ignoring L_m 44
4-8 Simplified Circuit Model of Real Transformer 45

5-1 Simulated Current in Matlab 51
5-2 Current and Voltage Waveforms of Initial Test 53

6-1 Block Diagram of Finite State Machine 57
6-2 Schematic of Capacitor Box 60

7-1 Picture of DESIRE user interface 67
7-2 Unexpected Oscillation 71

7-3 Winding capacitance 72
7-4 Transformer Model with Winding Capacitance 73
7-5 Cross-section of Wire to Show Skin Depth 74
7-6 Model of Source and Current, using Harmonic Components 78
7-7 Total Current Waveform and Estimate 82
7-8 Transient voltage distortion and estimate using DESIRE's R 83
7-9 Transient voltage distortion and estimate using measured R 84
7-10 Steady State VTACD Current 85
7-11 Harmonic Components of VTACD Current 86
7-12 Possible Local Voltage Waveform for VTACD 87

A-1 Switch Board Schematic 93
A-2 Voltage Sensor Board Schematic 94
A-3 PC Card Schematic 95

C-1 Diagram of DESIRE 180

List of Tables

2.1 Programming of 8254 19

6.1 Capacitor Values for Relay Settings 61

7.1 Calibration of Capacitor Box 64
7.2 Calibration of Capacitor Box 64
7.3 Nominal and Calibrated Data from DESIRE 76
7.4 Measured and Estimated Supply Impedances 80

Chapter 1

Introduction

The Nonintrusive Load Monitor (NILM) is a device that monitors voltage and current waveforms at the utility service entrance of a building in order to determine the operating schedule of every load within the building. The NILM offers easier installation at a fraction of the expense of many other load monitoring techniques because of its ability to collect and process information from a single location. The NILM is useful, for example, to electric utilty companies who wish to know the operating schedule of their customers' loads. Knowledge of the operating schedule allows the power company to prepare for peak loads and to predict the quantity and type of future power consumption.

The transient behavior of a typical electrical load is strongly influenced by the physical task that the load performs. The load survey conducted in [1] indicates that intrinsic properties modeled as nonlinearities in the constitutive laws of the elements that comprise a load, or in the state equations that describe a load, or both, create repeatably observable turn-on transient profiles suitable for identifying specific load classes. The turn-on transients associated with a fluorescent lamp and an induction motor, for example, are distinct because the physical tasks of igniting an illuminating arc and accelerating a rotor are fundamentally different. Distinctive transient profiles tend to persist even in loads which employ active waveshaping or power factor correction.

This observation has led to the development of a transient event detector for nonintrusive load monitoring. In [1], [2], and [3], a multiscale transient event detection algorithm was introduced that can identify individual loads operating in a building by examining transient profiles observed in the aggregated current waveforms available at the service entry. This detection algorithm extends the applicability of the NILM to demanding commercial and industrial sites, where substantial efforts, e.g., power factor correction and load balancing, are made to homogenize the steady-state behavior of different loads, and where loads may turn on and off very frequently. With the incorporation of the transient event detector, the NILM is also an important platform for monitoring the performance and power quality of critical loads. For example, the NILM can track down power quality offenders, i.e., loads which draw extremely distorted, non-sinusoidal input current waveforms, by correlating the introduction of undesired harmonics with the operation of specific loads at a target site.

Utilizing the detection algorithm developed for the NILM, a Multiprocessing Load Monitor (MLM) was developed and introduced in [4]. The MLM relies on an array of parallel processors to perform the transient event detection and tree structured decomposition. This method significantly reduces the time necessary to detect transient events, but depends on more complex memory management for combining the results of individual processors.

1.1 Contributions of This Work

The transient event detection algorithm and the parallel version of this algorithm employed in the MLM identify observed transients by comparing them to a library of known transient templates. Searching for complete transients, however, has been shown to be an undesirable approach because it limits the tolerable rate of event generation [1]. No two transient events could overlap significantly if each transient were to be identified correctly. Instead, the transient event detector searches for a time pattern of segments with significant variation, or *v-sections*, rather than searching for a transient shape in its entirety.

The success of this approach depends in part on the assumption that transient shapes for a particular load or load class will be similar to some degree each time a load is activated (assuming no catastrophic changes in the load). An important factor in determining the shape of a load transient, at least for some loads, is the precise instant during the utility voltage line cycle when the load is activated. To study the effects of different activation times on different loads, a power electronic switch, the load test stand, has been developed which permits loads to be activated at a precise instant of the voltage waveform. Transient or v-section variability can be studied with this device.

There are unavoidable impedances in the transmission of power, such as transformers, circuit protection, and lengths of wires. Harmonic currents flowing through these impedances cause distortion in the voltage waveform. A device has been developed that measures local impedances in the distribution network, which, when used in conjuction with the operating schedule determined by the NILM, can be used to predict voltage distortion. This system has been labeled the Determination of Supply Inductance and Resistance system (DESIRE).

1.2 Thesis Outline

In this chapter, we introduced the NILM and its method of identifying loads based solely on transient waveforms. We discussed the dependency of transients on the precise time at which the load is activated and the Load Test Stand that was created to study this behavior. The DESIRE system was also introduced in this chapter.

Chapter 2 discusses the implementation of the load test stand. The hardware and software used by the load test stand are described in detail. The testing of the load test stand and the collection of data using the device are also addressed in this chapter. Chapter 3 details a program that can be used in conjunction with the load test stand to identify activation transients.

Chapter 4 describes the power transmission model that is used throughout this thesis.

Specifically, a circuit model for a transformer is derived and simplified in this chapter. Chapter 5 introduces a method for estimating supply impedance parameters based on the model derived in Chapter 4.

Chapter 6 details the design of the hardware used for DESIRE. Chapter 7 describes the user interface for DESIRE, the method chosen for extracting the supply impedances for recorded data, and experimental data collected using the system. Also in this chapter, some modifications are made to the model discussed in Chapter 4. Several Appendices have been added to augment the body of this thesis.

Chapter 2

Load Test Stand

Different loads exhibit different transient behaviors when activated. The transient patterns associated with a class of devices are functions of the physical nature of the load and the point in the line cycle that the device is activated. To study the relation between the turn on point and the transient, a power electronic switch has been developed that activates loads at a precise point in the line voltage waveform. This switching device assists in the identification of transients for the NILM and also with the identification of supply impedance.

2.1 Background

The NILM separates and identifies the transients of loads as they are turned on. For the NILM to properly identify loads, it must be able to compare the acquired transient with transients of known loads. Since transients can vary with the point in the line cycle at which the load is turned on, the extent of this variation must be determined. This chapter examines the design of a switch that can activate a load at any of 1024 evenly spaced points in a line cycle period. The key components of the switch are three printed circuit boards which will henceforth be referred to as the switch board the voltage sensor board, and the PC card. [See Figure 2-1]. The voltage sensor examines one of the three phase voltage waveforms and generates a square

wave for the personal computer controller. This square wave input waveform is in phase with the input waveform and is used as a reference point for the PC. The user enters a number in the control software that indicates the number of 1/1024's of a line cycle to wait before turning on the switch. At the specified time, the PC activates the switch board, closing the switch and providing power to the load. The next sections describe the design of the load test stand.

2.2 Implementation

Each of the board subsystems in the test stand are described below.

2.2.1 Switch Board

Each switch board has three identical circuits, which individually control the presentation of one of the three voltage phases to the output. To allow the circuit board to use low voltage devices to control the high voltage output, the gate drive for the line voltage switch is provided through optoisolators. A "high side" drive for the line voltage switch and associated components is created using a split-bobbin transformer that is driven using a line-to-line connection to the utility. The low AC output of this transformer is rectified and filtered to create a stable voltage to run power regulators which provide power for the high side control circuitry.

The essential element in the switching device is the alternistor, a high voltage semiconductor thyristor [See Appendix A for schematic]. When a pulse of 125mA flows into the gate of the alternistor, a low impedance path is formed between the input and the output of the alternistor until the input voltage crosses zero with a negative slope, at which point the switch turns off. Similarly, when a pulse of 125mA is drawn from the alternistor, a low impedance path is formed until there is a positive zero crossing in the input voltage. The load test stand ultimately controls the gate current so that the alternistor is in its high impedance state until the point in the line cycle that the user defines, at which point the alternistor becomes a low impedance path. The alternistor remains on until the user turns the switch off.

To drive the alternistors, two npn transistors are connected in series to create a push-pull output, sourced to drive the alternistor gate [See Appendix A]. The transistors are controlled by a pair of optoisolators. The high side input each optoisolator is connected to a signal which is controlled by the PC card. The low side inputs of the optoisolators are connected to the outputs of a high performance comparator, the LT1016 [5]. One of the optoisolators is connected to the Q output of the LT1016, while the other is connected to -Q to ensure that only one of the transistors is turned on at a time [See Appendix A]. When the signal from the PC card is low, both control optoisolators are off and the switch is open. When the signal is high, one of the control optoisolators is forced on, and the switch is in its low impedance state. The inputs of the LT1016 are connected to the floating ground and the output of the alternistor. Because the LT1016 can not support a voltage difference of 170v that could be present when the alternistor is off, protection circuity in the form of a resistor and diodes maximize the voltage difference that the LT1016 sees is 0.6v.

2.2.2 Voltage Sensor Board

Accurate syncronization with respect to the utility voltage waveform is essential for timing accuracy. To achieve syncronization, a circuit was designed that compares the line cycle of one of the three phases of the power source to ground using an LT1016 [See Appendix A]. When the line cycle has a voltage greater than zero, the circuit output is +12v; when the line cycle voltage is less than zero, the circuit outputs -12v. The resulting square wave has a rising edge that corresponds to a positive zero crossing of the line cycle and a falling edge that corresponds to the negative zero crossing. This square wave is used to syncronize the PC Card.

2.2.3 PC Card

The PC card contains the circuitry that controls the communication between the switch board, the voltage sensor board, and the PC. The PC card takes input from the user, controlled by the assembly code software [See Appendix B.2], and from the voltage sensor board to control

the time at which the switch is turned on.

The phased-locked loop in this circuit produces a square wave of 61.44kHz from the square wave output of the voltage sensor. The essential components of the PLL are the phase detector, the low-pass filter, the Voltage Control Oscillator (VCO), and the divide by "n" counter. The phase detector, a 4046, compares the f_{ref} with f_{comp} [See Figure 2-2 and Figure 2-3, both from [6]] The difference, V1, is then passed through a low-pass filter and amplified, appearing as a dc voltage as it reaches the VCO, a 74S124. When $f_{ref} > f_{comp}$, the output of the phase detector is greater than zero and is less than zero for $f_{ref} < f_{comp}$. This output causes the VCO to change frequency so that f_{comp} comes closer to f_{ref}. By dividing the output of the VCO, f_{out}, by 1024 to obtain f_{comp}, the PLL produces the desired output square wave of 61.44kHz [6].

Timing on the PC card is controlled by an 8254, a programmable interval timer/counter containing three independent counters which are designed to be controlled by computer software [7]. The 8254 takes input from the software and a clock, then delays for the programmed number of counts before interrupting the CPU. By providing a delay of any desired length, the 8254 cuts software overhead significantly for any sort of delay. There are 6 different modes that the 8254 can use, including an interrupt on terminal count, hardware retriggerable one-shot, a rate generator, a square wave generator, binary rate multiplier, and a complex wave generator. The mode that will be utilized in this circuit will be the interrupt on terminal count mode. This mode keeps the output low until the programmed number of clock cycles have been counted, at which point the output becomes high [6]. Three command lines are used to tell the chip what parameters are to be used and the number of counts to delay. The first command line sets the mode for the desired counter. For this command line, A1 A0 = 11, $\overline{CS}2 = 0$, $\overline{RD} = 1$, and $\overline{WR} = 0$. The eight bits on the data bus line tell the chip the method of inputting the desired number of counts, the mode of counting, and whether the input is in binary or decimal for the counter that is to be used. The second and third command lines establish the number

of clock cycles that the 8254 will delay with least significant and most significant bits. With one counter, the highest number of cycles that can be delayed is 2^{16} (65536).

To control and isolate the output of the 8254's, an SN74HCT574, an edge-triggered flip-flop, has been connected between the output of the 8254 and the switch board. When the 574 is selected, it acts as a buffer, isolating the output of the 8254 from the transistors that drive the input signal of the switch board. However, when the 574 is not selected, the output of the 574 is in tristate, and does not turn on the optoisolators on the switch board.

2.2.4 Software

A program written in assembly code [See Appendix B.2] takes a value from the C program [See Appendix B.1] to determine the number of line cycles to pause before initiating the counting on the 8254's. This program takes data from a C program which serves as the user interface and creates an interupt service routine that uses nonmaskable interrupts. The program counts line cycles using the square wave from the voltage sensor board, then it initializes the counters in the 8254's when the appropriate number of cycles have elapsed. This program allows for delays in turn on time between phases that could not be produced by using the 8254 alone, which is limited to counting 65536 cycles of the phase locked loop.

The C program takes four inputs for each phase. The first input, I1, initiates a routine in assembly language that creates a delay of I1 line cycles. At the completion of this routine, the 8254 is initialized using the inputs I2, I3, and I4, which correspond to the number of cycles of the PLL (61.44kHz) that the 8254 is to count before it turns on the switch.

The data bus line on the card in the computer is controlled by the Microsoft C command "outp()" [8], which takes two arguments. The first argument, the port, sets the parameters on the card, such as A0, A1, and the chip select, while the second controls the eight data lines. When the port is assigned to set the mode for a counter [See Table 2.1], the data lines are set to one of the following states: 0x30, 0x70, or 0xB0, which specify the counter, keeping the counter in mode 0, interrupt on terminal count, the Read/Write taking in two bytes - least

outp	8254	Function
0x300	0	Sets byte for counter 0
0x301	0	Sets byte for counter 1
0x302	0	Sets byte for counter 2
0x303	0	Sets mode for specific counter
0x304	1	Sets byte for counter 0
0x305	1	Sets byte for counter 1
0x306	1	Sets byte for counter 2
0x307	1	Sets mode for specific counter
0x308	2	Sets byte for counter 0
0x309	2	Sets byte for counter 1
0x30A	2	Sets byte for counter 2
0x30B	2	Sets mode for specific counter

Table 2.1: Programming of 8254

significant first, and the counter in binary. When the port is assigned to set the bytes for the counter, the first data line corresponds to the least significant byte and the second to the most significant. As soon as the second byte is loaded, the 8254 begins counting.

2.3 Collection of Data

To test the switching device, numerous tests were conducted, turning the switch on at different points in the line cycle with different loads. For each test, the voltage waveform that the voltage sensor examines was used as a reference. The time difference between a zero crossing of the reference and the turn on time matched the specified delay. The waveforms collected for these tests also indicate that there is minimal crossover distortion and no noticeable distortion in the steady state output.

Figures 2-4 to 2-13 show transient currents for various types of loads. For each load, a scaled version of the reference voltage is shown on the same graph as the transient current waveform.

2.4 Conclusions

All information collected in the tests indicates that the timing switch is a success. The timing tests verified that we could turn on the line current at any desired point in the cycle. The steady state tests demonstrated that the switch action does not adversely affect the line voltage waveform in steady state when turned on. Therefore, we will be able to use the timing switch to study transient behavior of loads at various points in the line cycle. The results of these studies can then be used to assist in the identification of loads by the NILM.

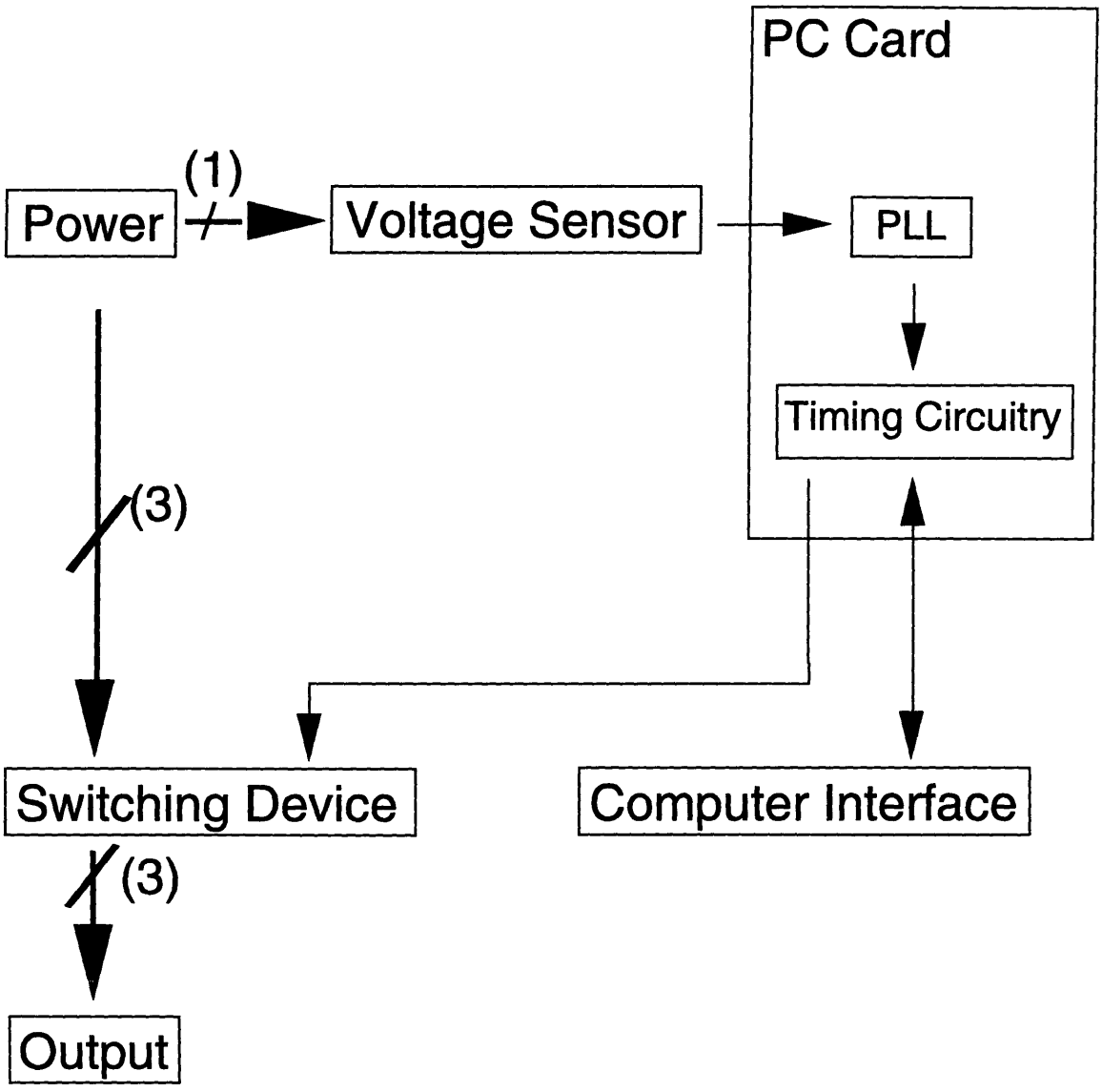


Figure 2-1: Block Diagram of Load Test Stand

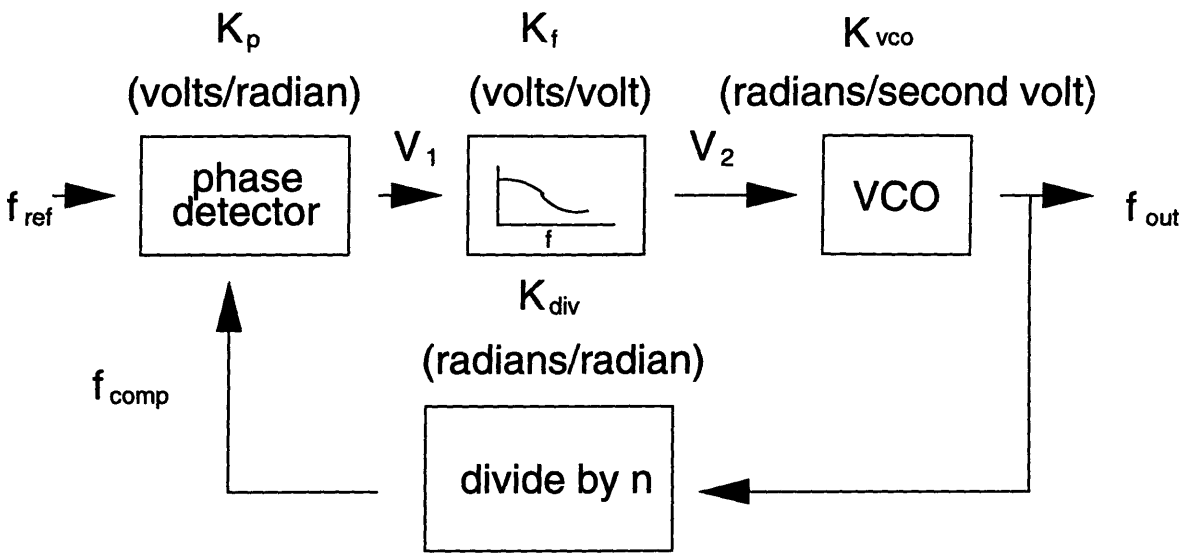


Figure 2-2: Frequency Multiplier Diagram

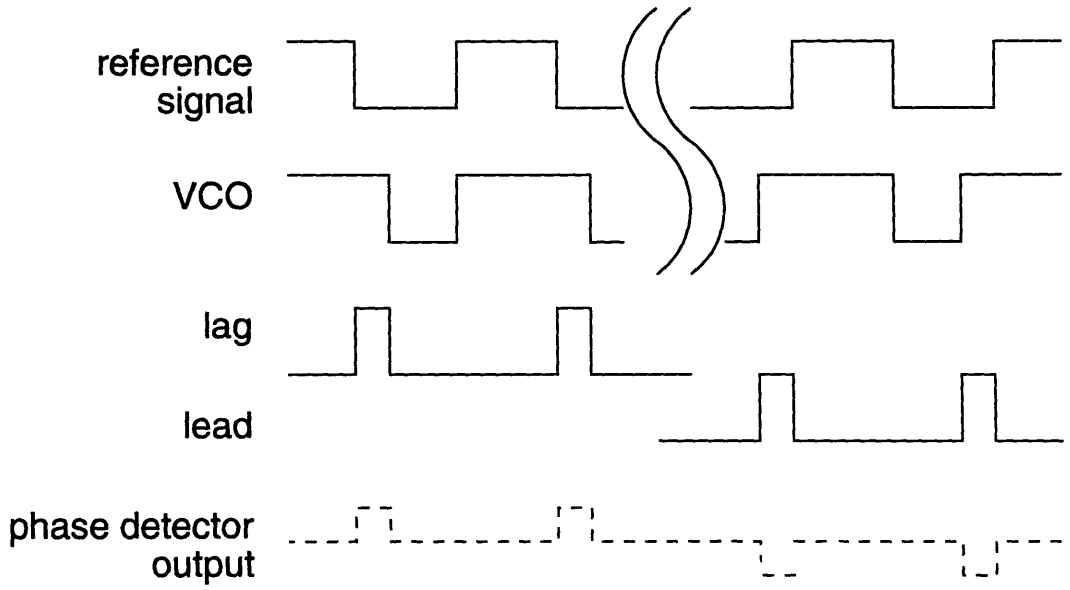


Figure 2-3: Edge-Sensitive Lead-lag Phase Detector

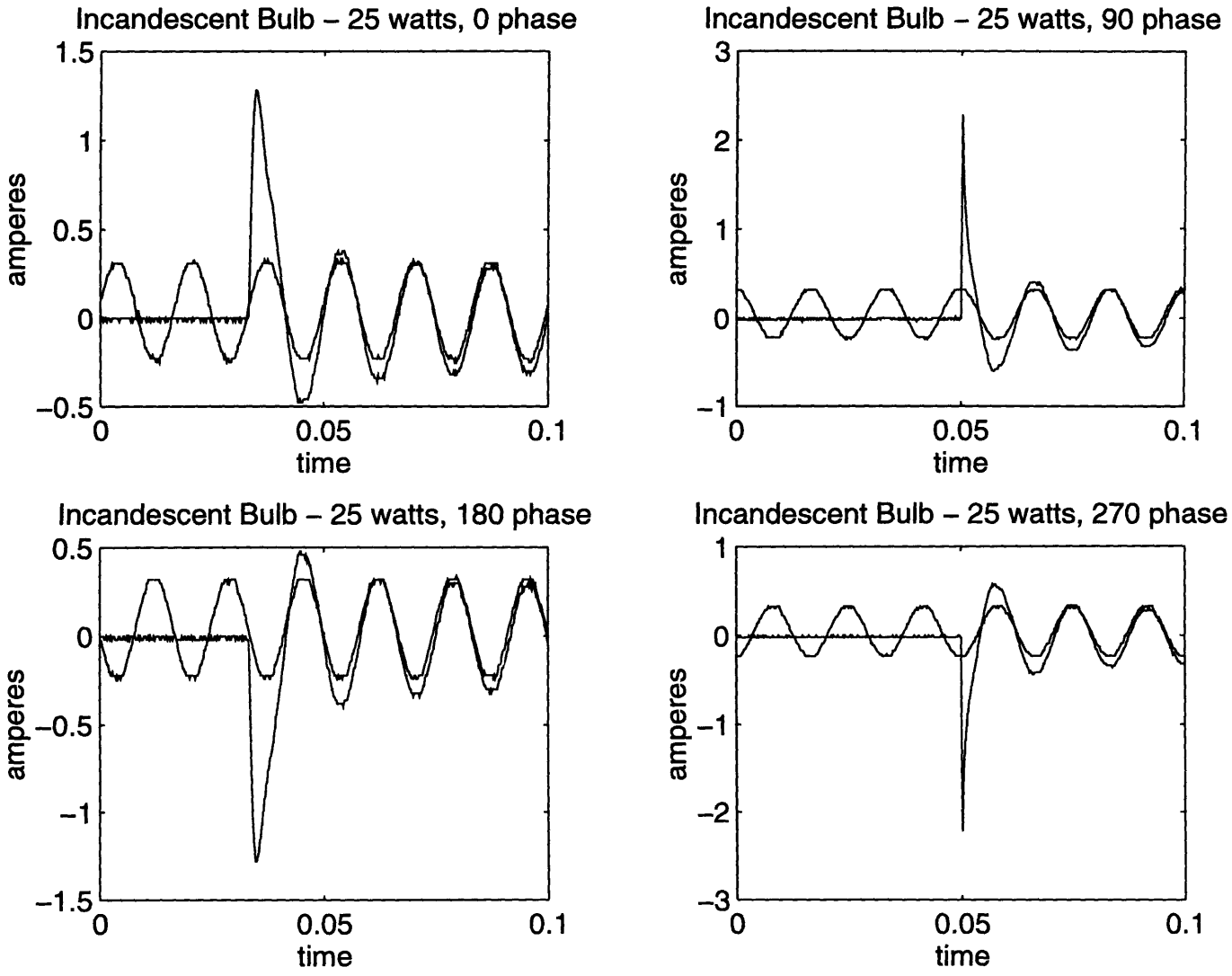


Figure 2-4: Turn on Transients of 25 Watt Incandescent Bulb

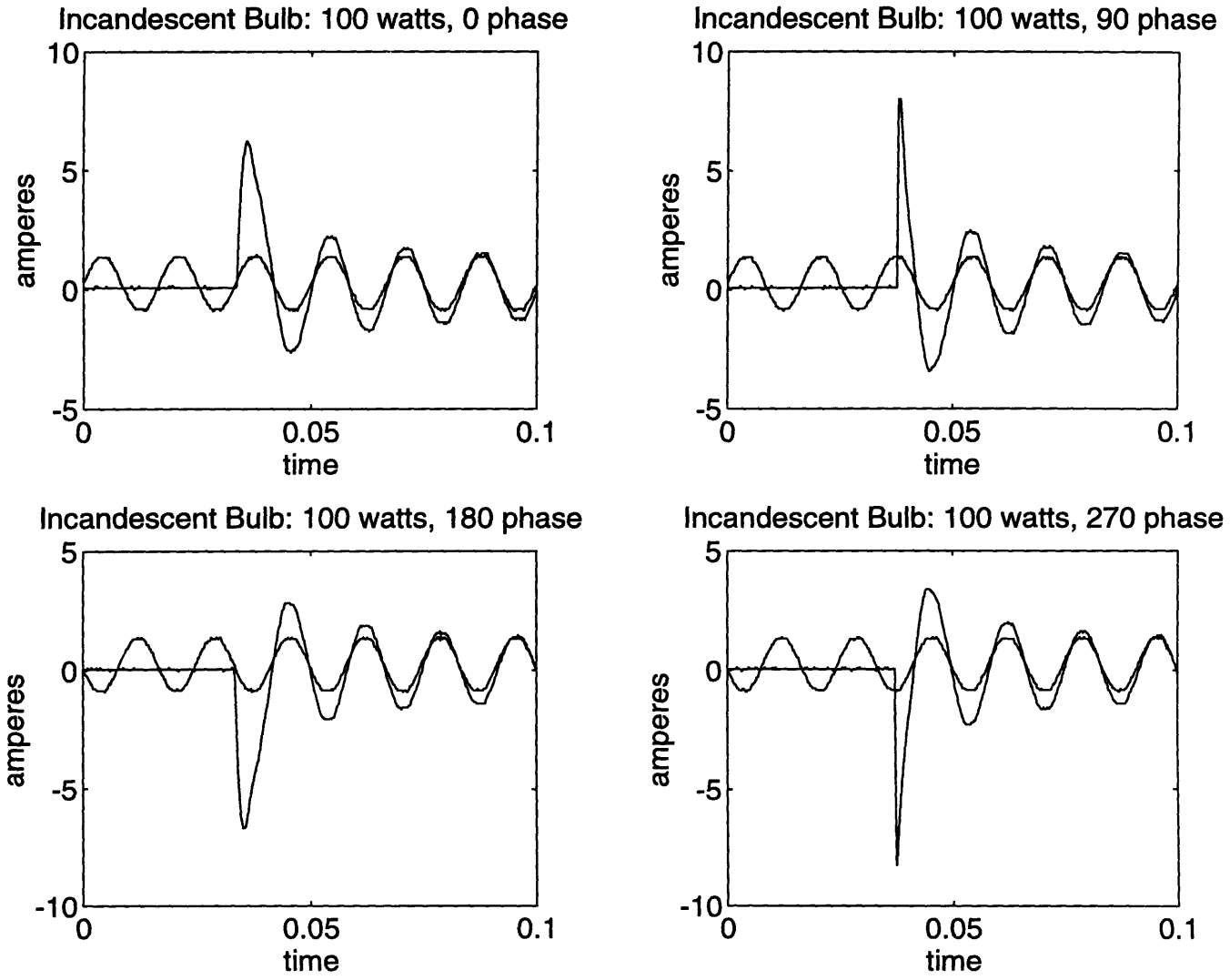


Figure 2-5: Turn on Transients of 100 Watt Incandescent Bulb

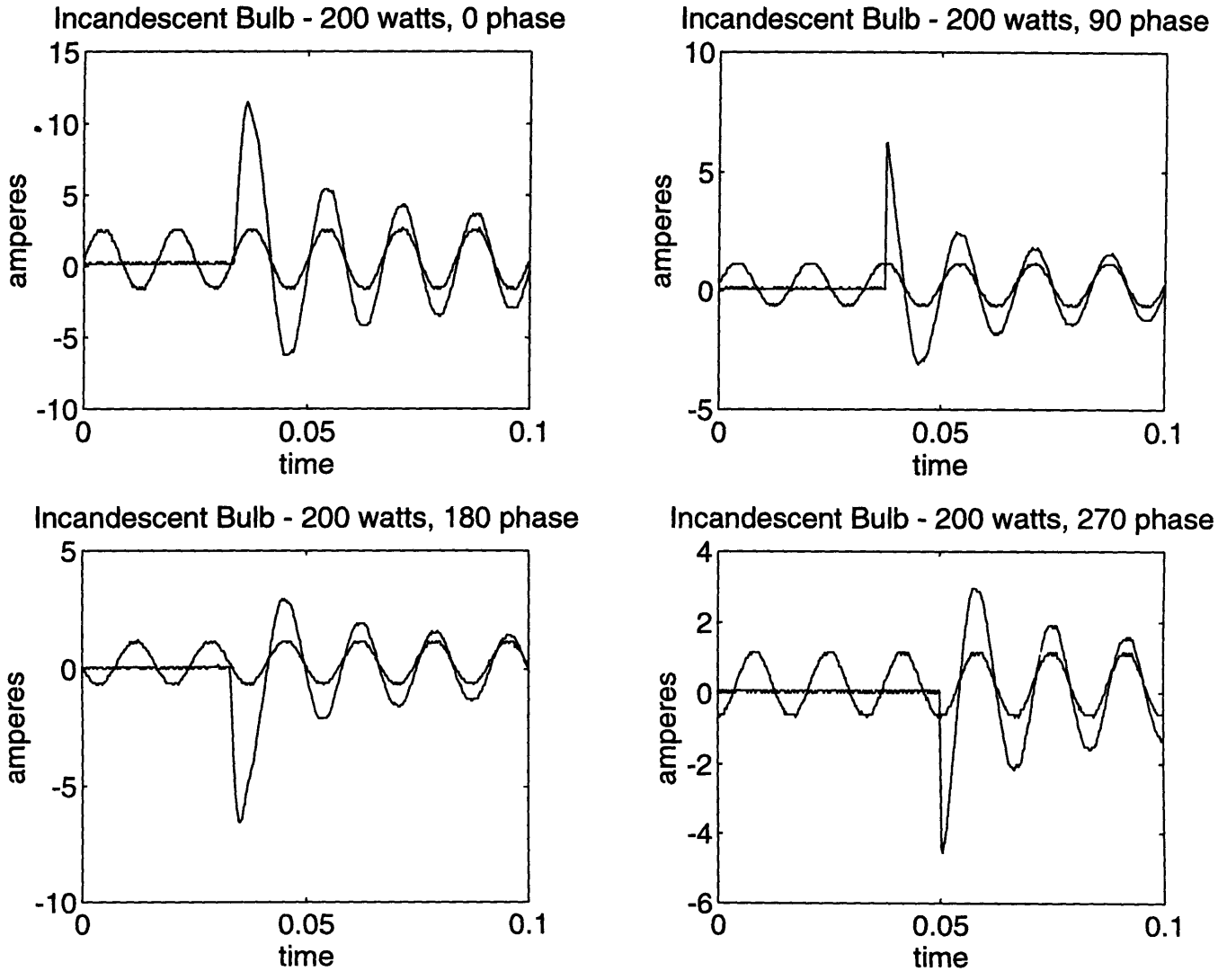


Figure 2-6: Turn on Transients of 200 Watt Incandescent Bulb

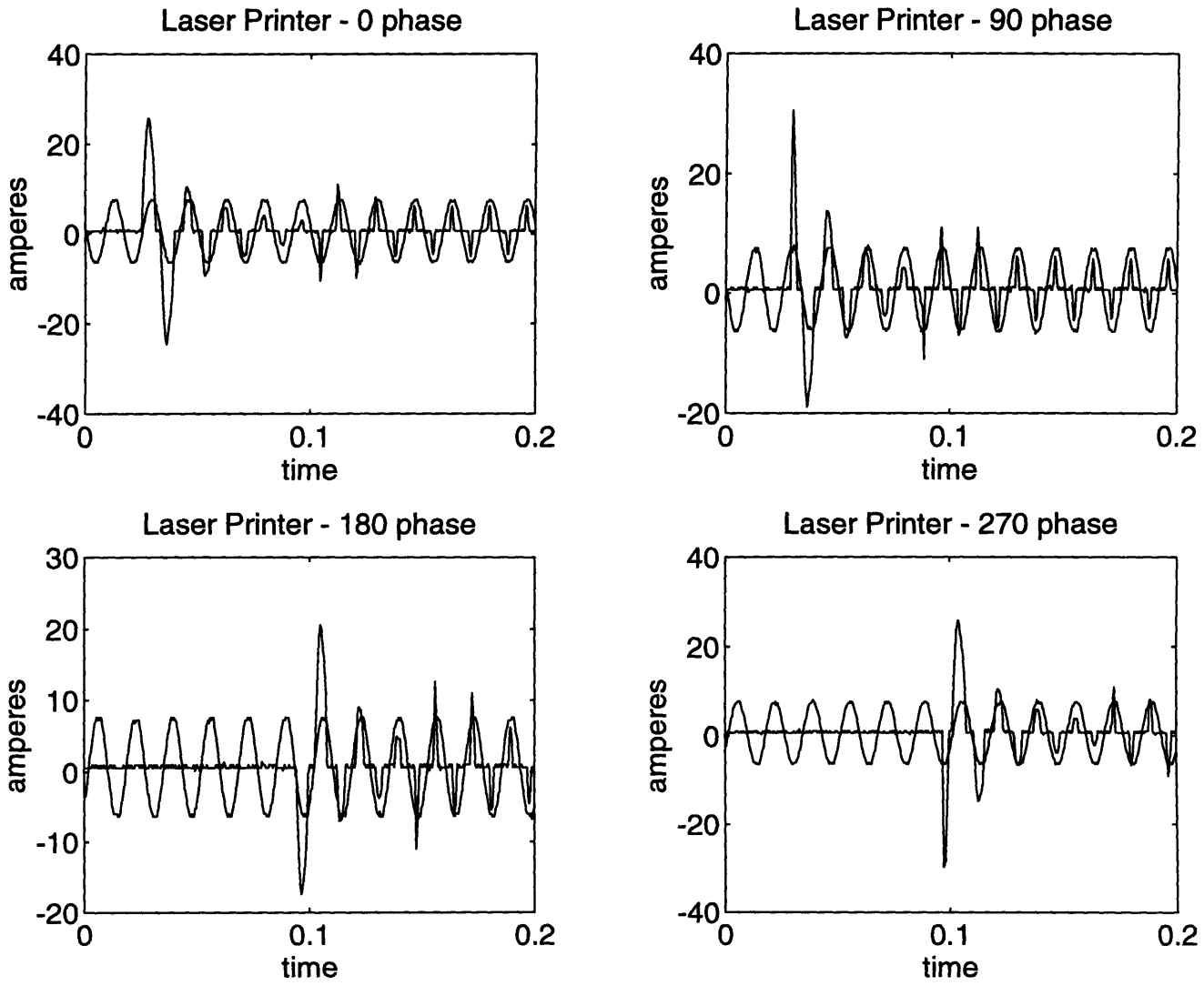


Figure 2-7: Turn on Transients of Laser Printer

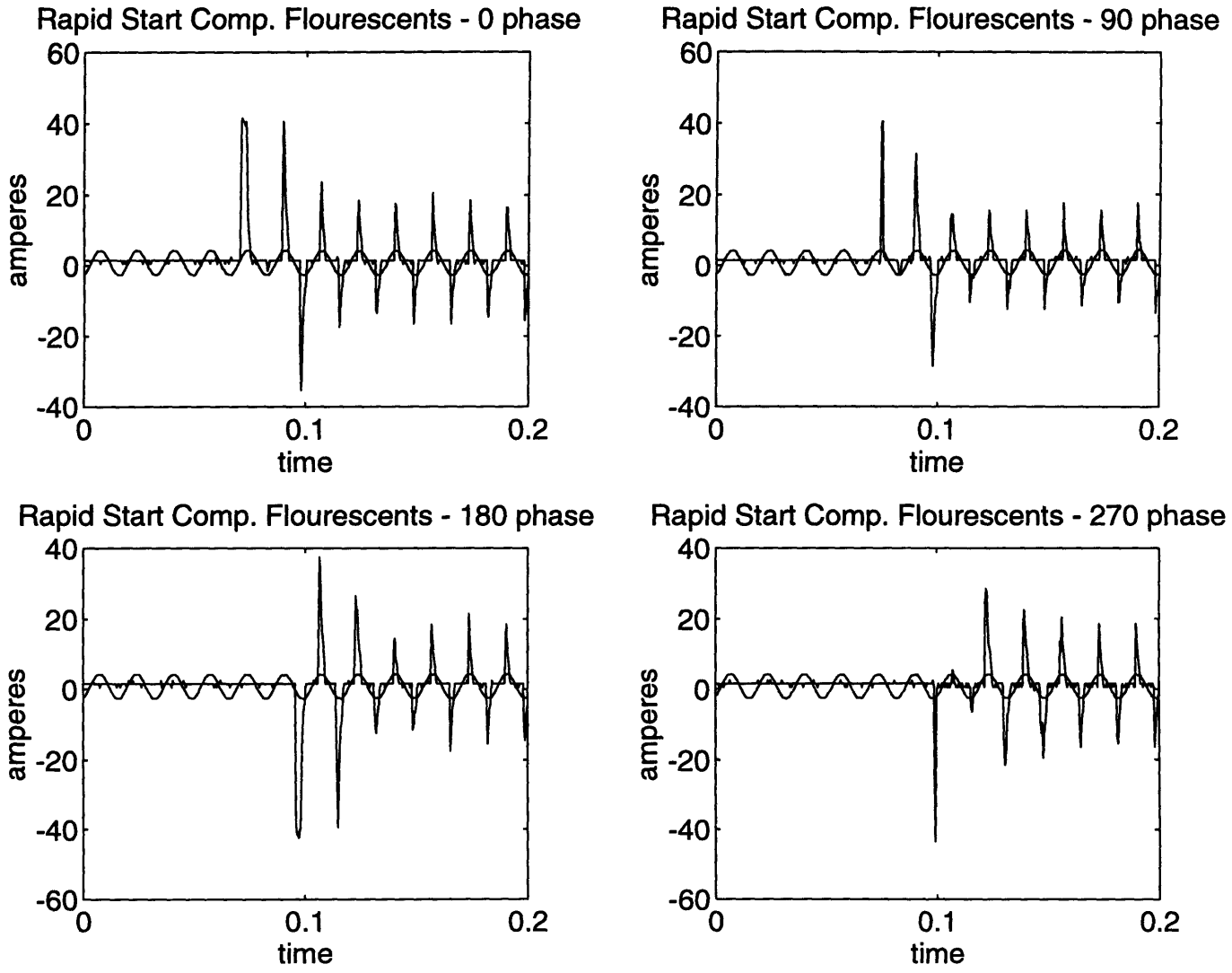


Figure 2-8: Turn on Transients of Rapid Start Compact Florescent Bulb

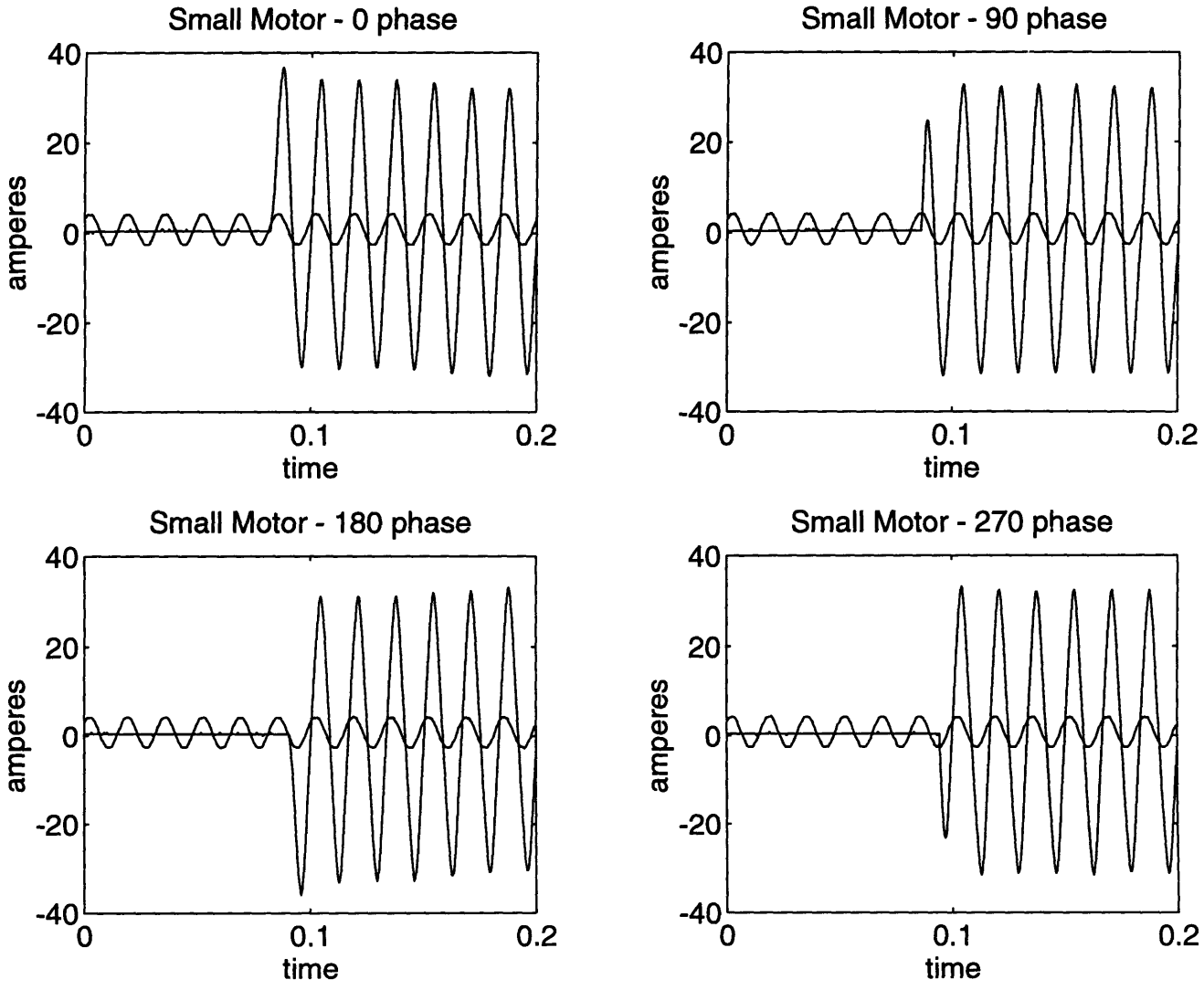


Figure 2-9: Turn on Transients of Induction Motor

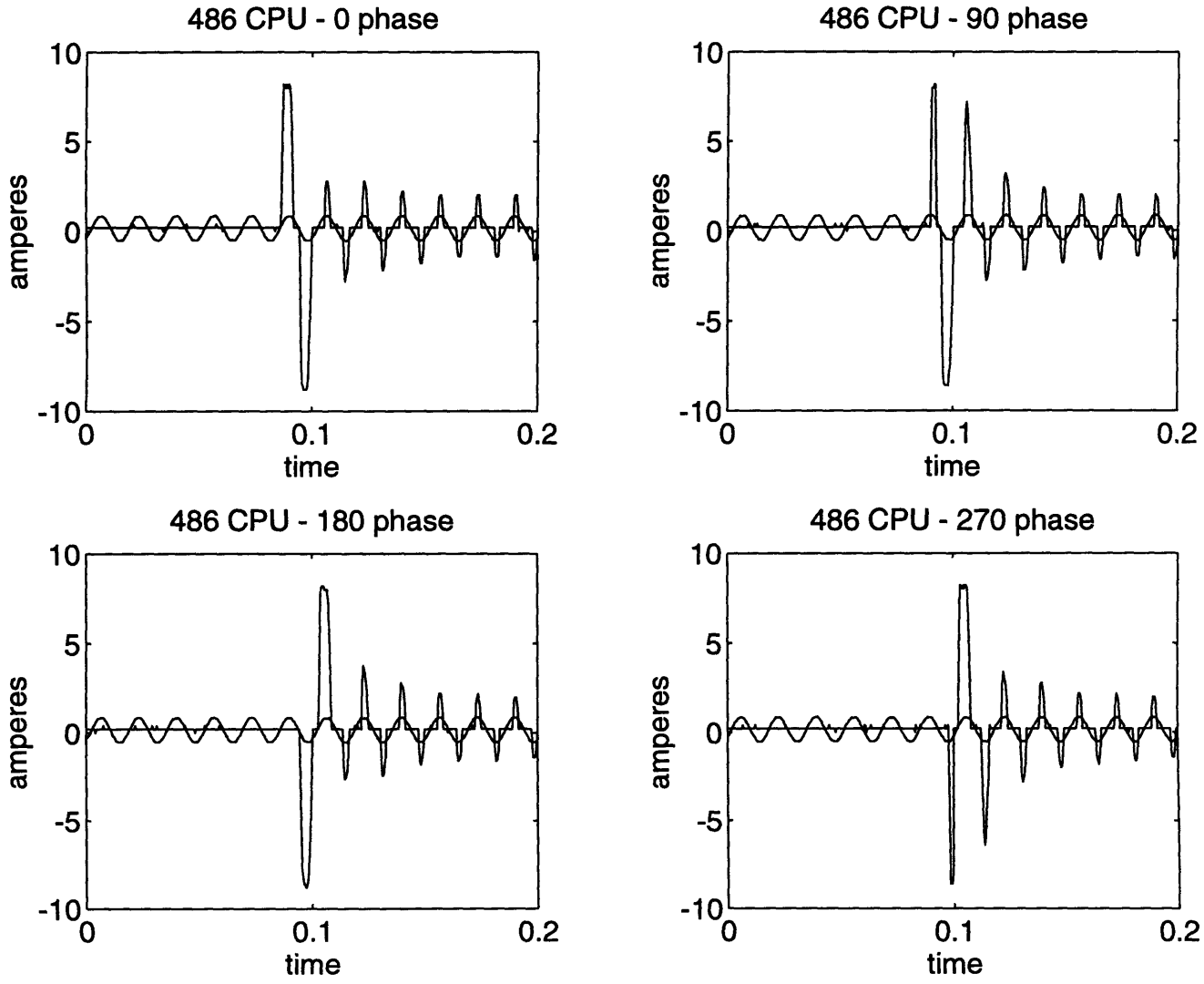


Figure 2-10: Turn on Transients of 486 Computer

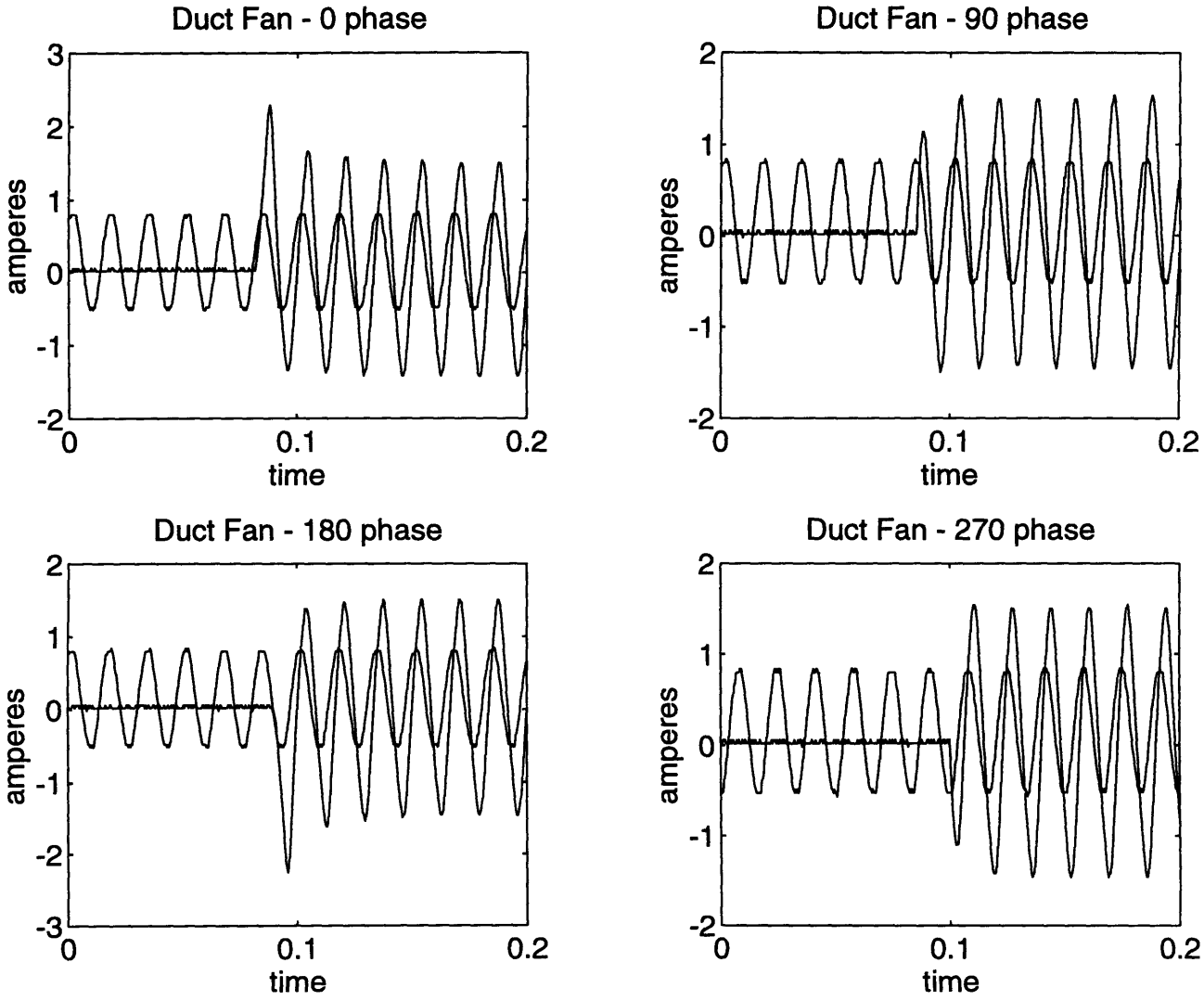


Figure 2-11: Turn on Transients of Duct Fan

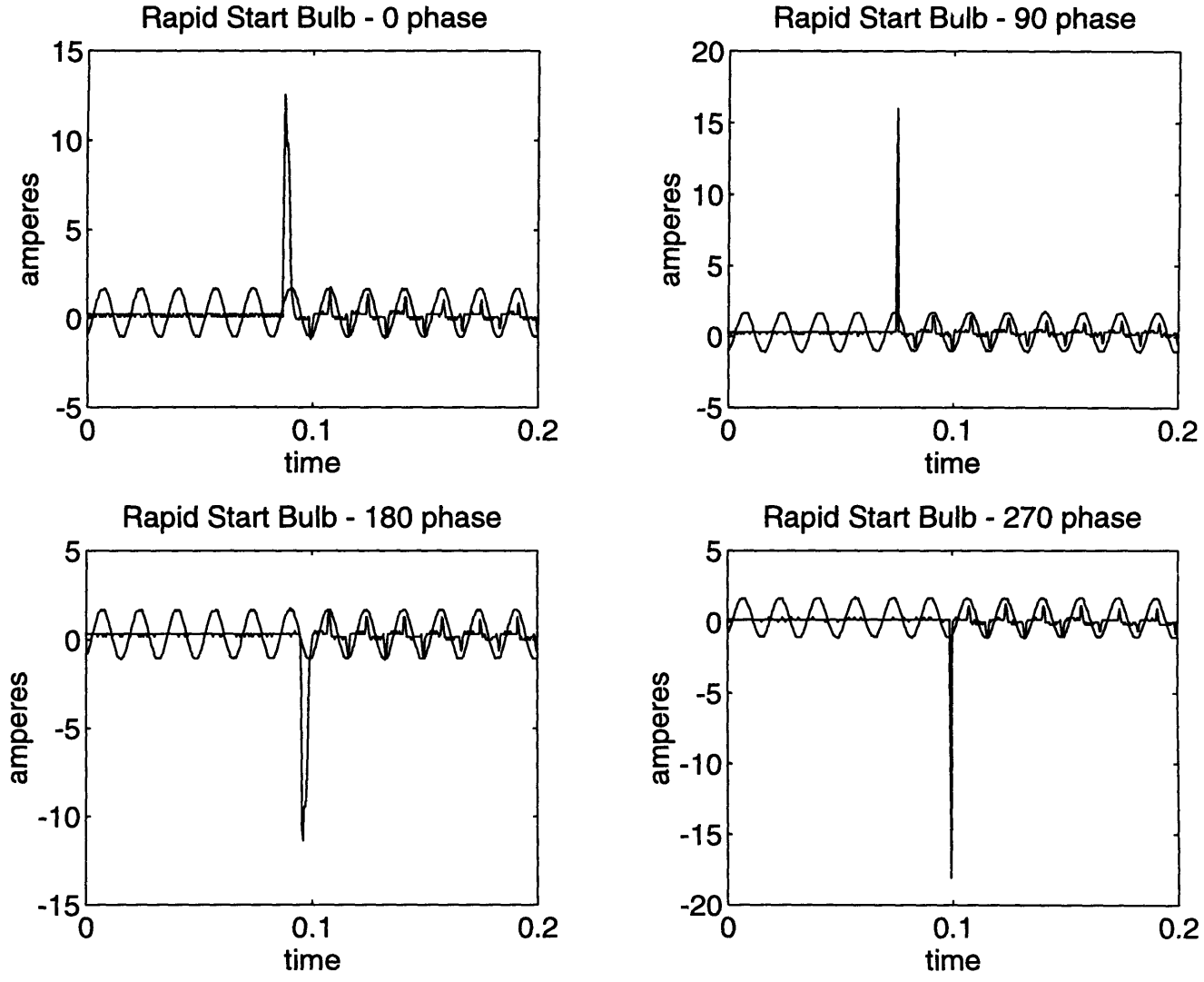


Figure 2-12: Turn on Transients of Rapid Start Bulb

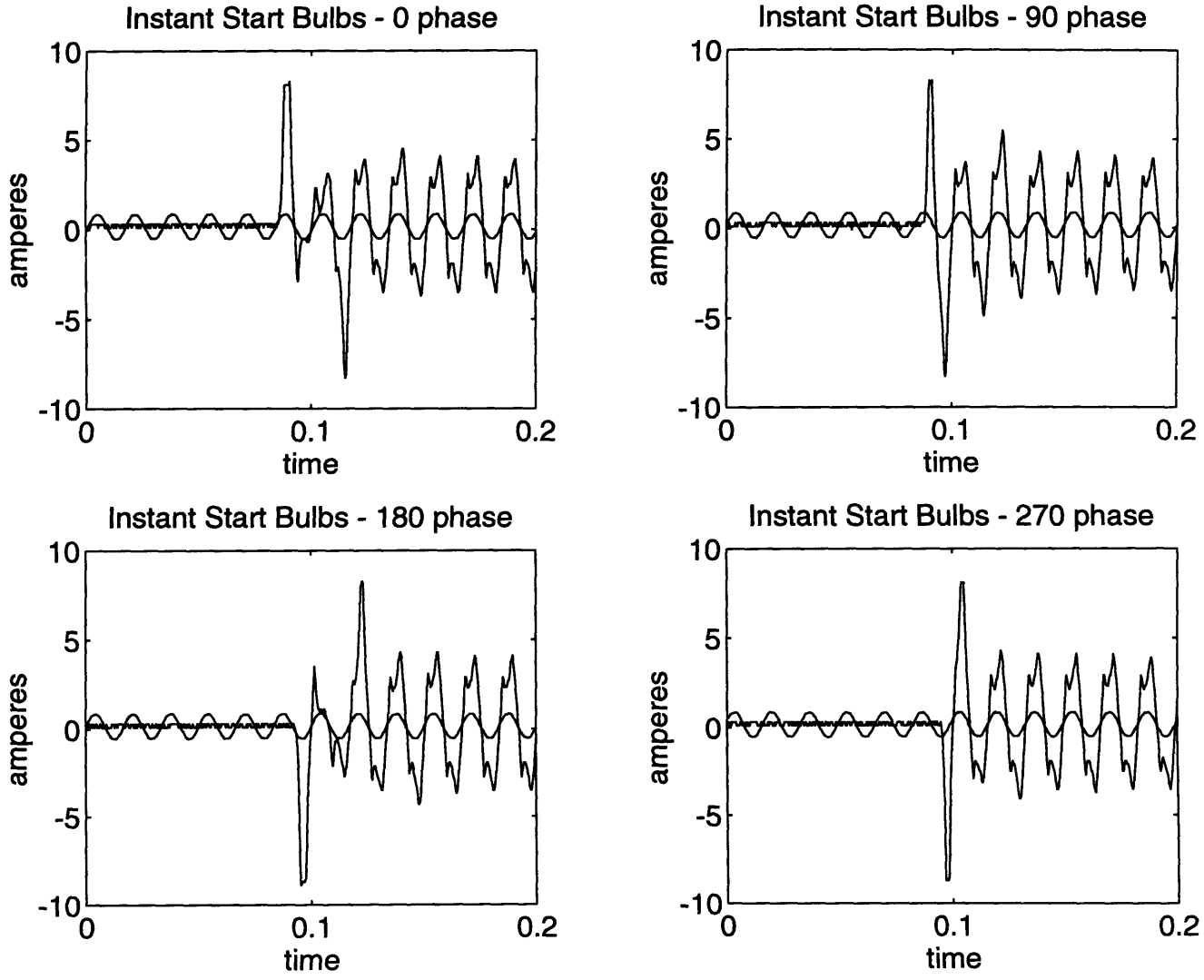


Figure 2-13: Turn on Transients of Instant Start Bulb

Chapter 3

V-Section Collection

In this chapter, "vsmaker.m", a Matlab program designed to automate the selection of *v-sections* for the NILM, is described. When used in conjunction with the load test stand, this program provides the user with average values for the v-section and error tolerance levels so that the v-section is recognized even in the face of minor differences or errors in observations.

"Vsmaker" low pass filters the data recorded by the spectral envelope preprocessor [2] and differentiates it [See Appendix E.1]. Areas of significant variation will have relatively large differentives. By recording the locations in time of the areas of large differentives, a reasonable v-section can be created. The v-section information for each data stream is then used to determine a reliable v-section set [1], and the threshold level needed to insure all v-sections will be recognized.

"Vsmaker" takes four input variables, *filtertype*, *order*, *thresh*, and *vsnum*, and up to ten globally defined data files, $to1, to2, ..., to10$, where each *to* file contains one turn on transient a particular load [See Figure 3-1]. When "vsmaker" is called, an internal matrix, *input* is formed, consisting of each *to* file that present. Each column of *input* corresponds to one file. Since the *to* files can contain noise, an undersireable trait when differentiating a waveform, the *to* files are low pass filtered. The variable, *filtertype*, allows the user to choose between using a butterworth filter or a median filter for the filter process. The *order* variable

sets the order of the butterworth filter or the length of the median filter. "Vsmaker" then calls an external function, "vsfiltb2.m" or "vsfiltm2.m", to perform the filtering process.

The filtered data is placed in an internal function, *filtdata*. Again, each column of *filtdat* corresponds to one *to* file. Each column of *filtdata* is then sent to the function file "vspart1.m" [See Appendix E.4]. "Vspart1" differentiates each column of the input matrix. Every point of the differentiated column that is greater than *thresh* correlates to a point of a v-section. For each point that is identified to be part of a v-section, the corresponding point in *filtdat* is recorded. Any points that are not in v-sections are replaced with zeros at the output. The internal matrix *data* contains the output matrices from "vspart1".

The next step is to determine the number of *to* files that have v-sections at each point in time. Since we are looking for reliable v-sections, the "ideal" v-section would be the common shapes present in each *to* file. "Vspart2.m" is called to record the number of *to* files that each v-section is present in, and returns the matrix *num* [See Appendix E.5]. "Vspart3.m" takes v-sections recorded in *data* and returns the average v-section in the matrix *vs* [See Appendix E.6]. "Vspart4b.m" next receives the data, and returns the matrix *vso* [See Appendix E.7]. *Vso* contains 10 columns, each representing the average v-section that was present in a certain number of *to* files. Column one of *vso* represents any point in time that a v-section was recorded. Column two represents any point in time that was a v-section for two of the *to* files, and so on.

"Vsmaker" now begins a sequence which will selects the v-sections. First, the average of the *to* files at each point in time is stored in *dmean*. The position of the v-section is created using the file "mednum.m", and the most reliable set of v-sections from "vspart4b" [See Appendix E.11]. The number of points in each v-section to be selected is then determined based on *vstype* and *vsnum*. The v-sections are then selected from the *dmean* file. The final v-section is placed in the *out* matrix.

To help in setting the threshold level of each v-section in the NILM, a couple of routines

have been implemented. The files "dicer2.m" and "testvs2.m" take the *out* matrix and finds the minimum threshold level for each v-section, and reports the data in $diffmax$. "Vsmaker" then plots the *to* input files and the vsection for examination by the user. Two examples of these plots are shown in Figure 3-2.

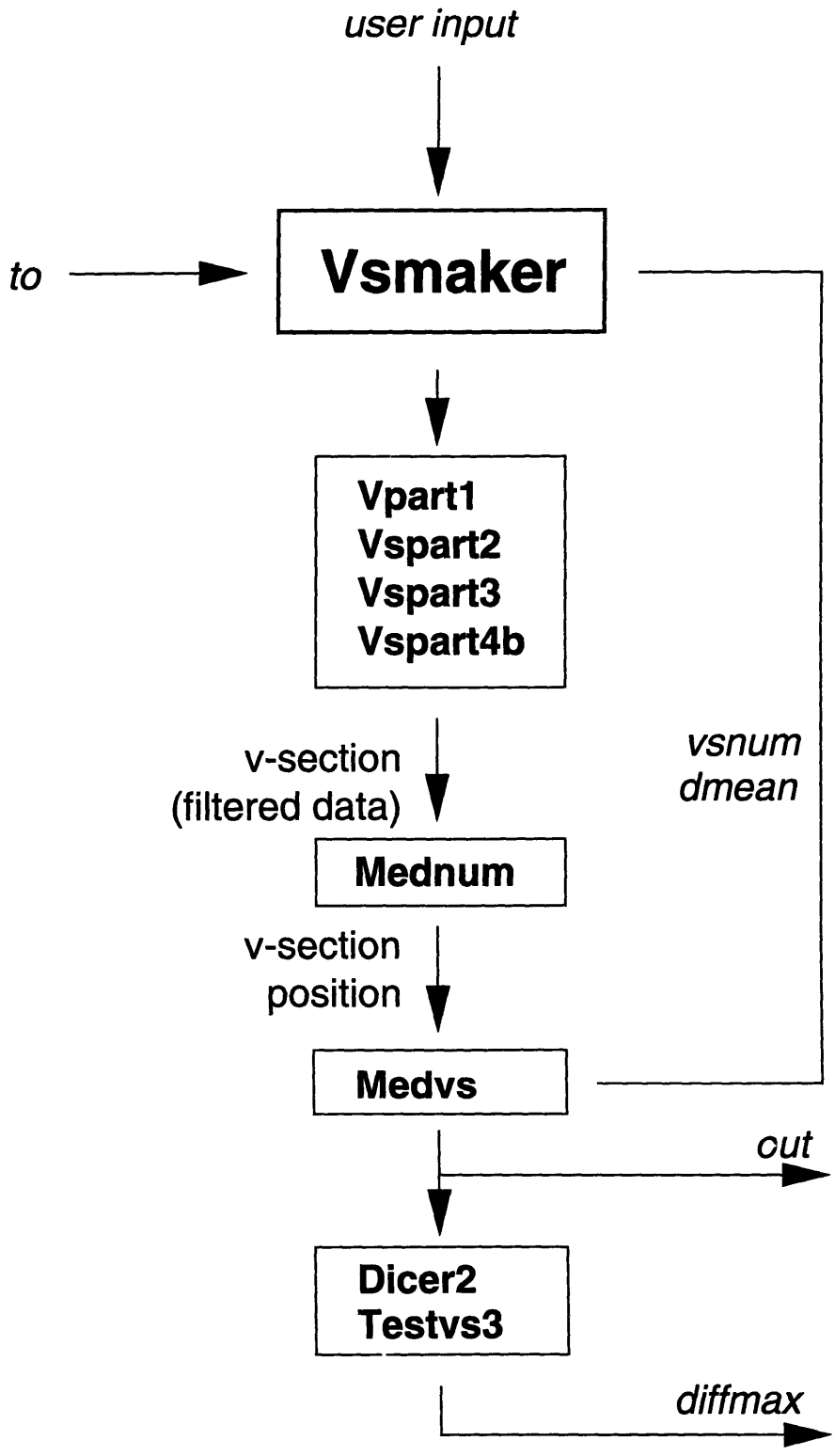


Figure 3-1: Block Diagram of Vsmaker

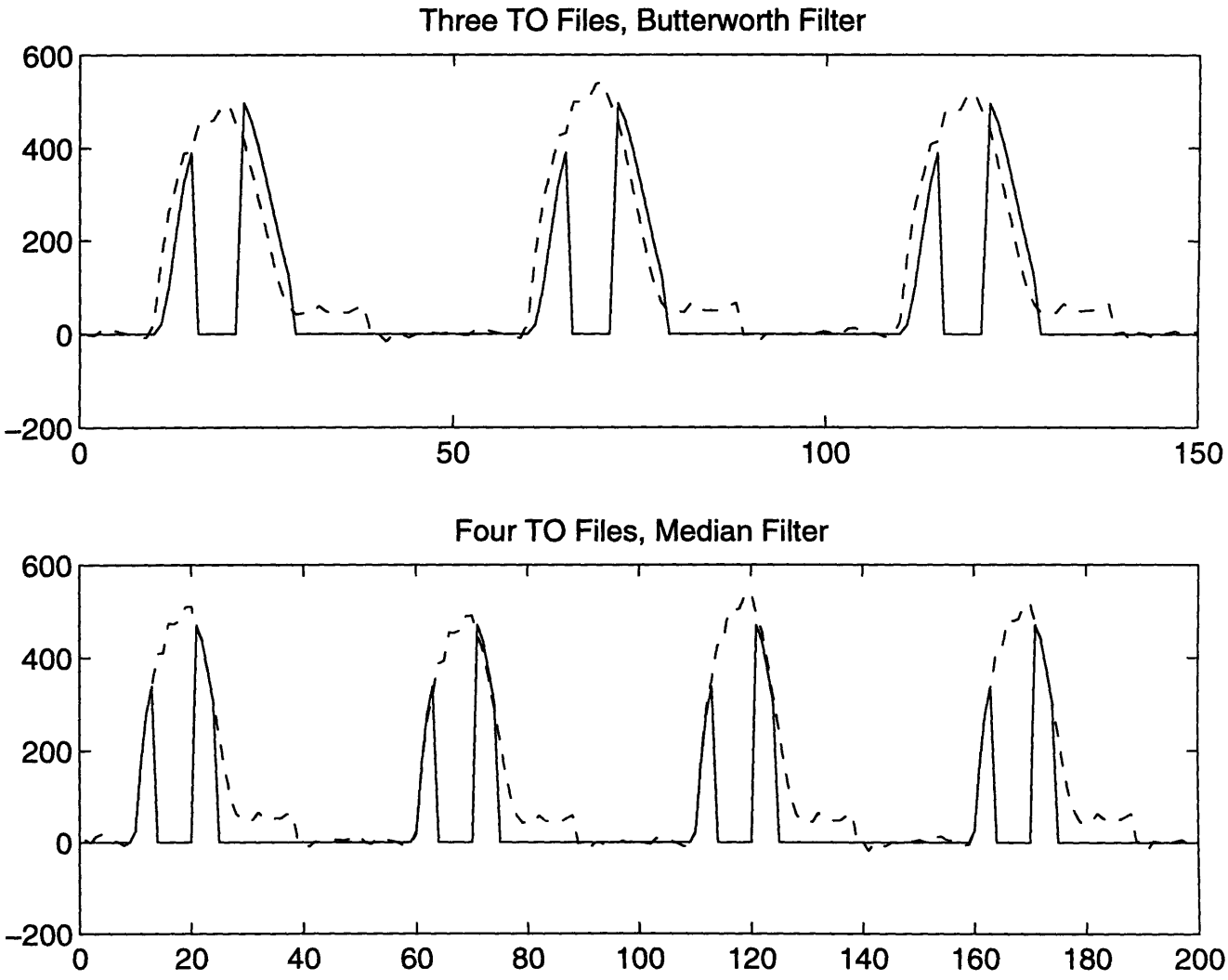


Figure 3-2: Plots of Input Data and Vsmaker Data

Chapter 4

Power Transmission Model

For power quality estimation, we will be examining a line-to-neutral connection to the electric utility. In the line-to-neutral connection, the terminals of a load of interest will be presumed to be connected to the secondary of a grounded single phase transformer driven by an ideal ac source. This simplification of the electric utility will prove adequate for predicting local voltage waveform distortion. This chapter develops a circuit model which will be used in subsequent chapters for power quality prediction.

4.1 Transformer Model

A transformer consists of two or more coils of wire linked by a common alternating magnetic field. One of the coils, the primary, is connected to an ac source. The ac voltage across the primary causes an alternating field in the magnetic core, which induces a voltage across the remaining, secondary coil terminals which can drive a load.

4.1.1 Ideal Transformer

In an ideal transformer, it is assumed that all of the flux that is produced by the primary links the secondary coil, and that the permeabiltiy, μ, in the transformer core is infinite. Figure 4-1

represents a toroidal transformer, and will be used for transformer analysis in this work. The toroidal transformer, though not exibiting the conventional power transformer geometry of the shell transformer [See [9] for a description of shell transformers], will serve to bring out the key physical behavior of a transformer. Faraday's law dictates that $V_1 = \frac{d\Phi_1}{dt} N_1$ and $V_2 = \frac{d\Phi_2}{dt} N_2$,

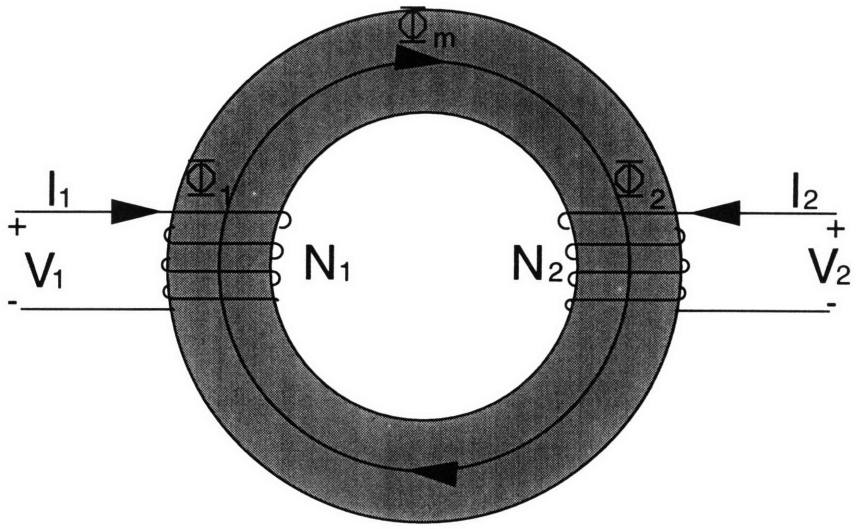


Figure 4-1: An Ideal Transformer

where N is the number of turns in the winding and Φ_1 and Φ_2 are the fluxes through the respective windings. Since all flux links the primary and secondary windings in this model, $\Phi_1 = \Phi_2 = \Phi_m$ and therefore,

$$\frac{V_1}{V_2} = \frac{N_1}{N_2} [9]. \tag{4.1}$$

To find the current in each of the windings, Ampere's Law can be applied to the ideal transformer, using the vector labeled Φ_m as the contour for the integral.

$$N_1 I_1 - N_2 I_2 = \frac{\oint \mathbf{B} \cdot d\mathbf{l}}{\mu \mu_o}, \tag{4.2}$$

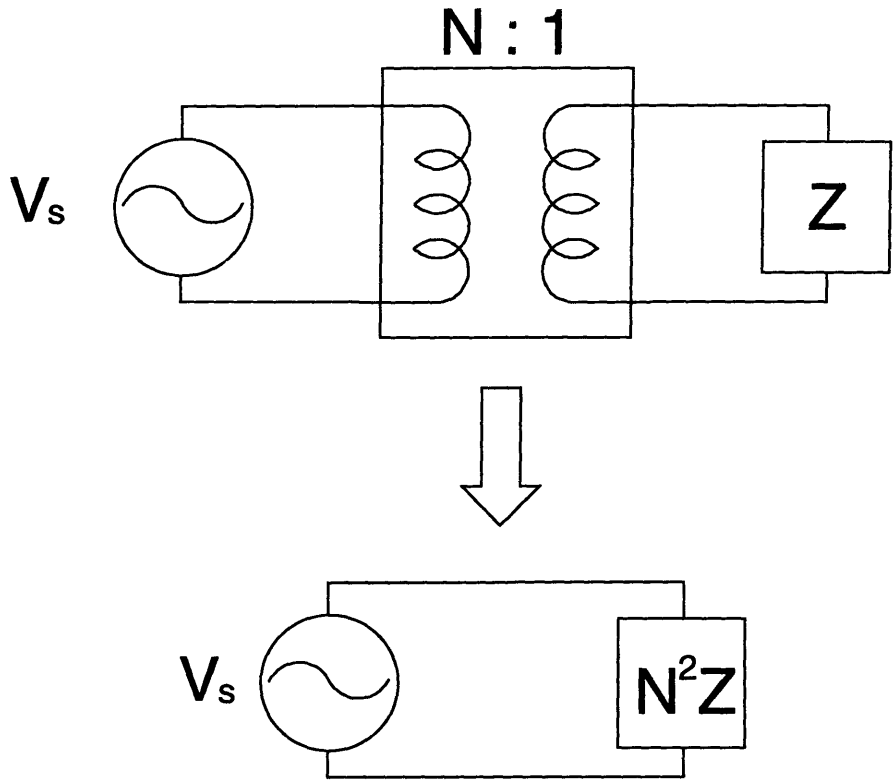


Figure 4-2: Reflecting Impedance Through an Ideal Transformer

where $\mathbf{B}$ is the flux density in the core, $\mathbf{l}$ is the length of the core, and μ is the relative permeability of the core. Evaluating $\oint \mathbf{B} \cdot d\mathbf{l}$ and subsituting into the Equation 4.2,

$$N_1 I_1 - N_2 I_2 = \frac{\Phi}{\mu \mu_o A} \approx 0 \tag{4.3}$$

since μ is infinite in this model. Equation 4.3 then simplifies to

$$N_1 I_1 = N_2 I_2 [10]. \tag{4.4}$$

Assuming that the terminal voltages are sinusoidal, the impedance on secondary can be related to an equivalent impedance on the primary side using Equations 4.1 and 4.4:

$$Z_2 = \frac{V_2}{I_2} = (\frac{N_2}{N_1})^2 \frac{V_1}{I_1} = N^2 Z_1 \tag{4.5}$$

where $N = \frac{N_2}{N_1}$, the turns ratio [11].

4.1.2 Real Transformers

In practical transformers, core permeability is finite. We still assume for the moment that all of the flux links the secondary winding. When μ is finite, Equation 4.4 does not hold. From the primary terminals, a practical transformer with an open circuited secondary winding looks like a finite-valued inductor, referred to as a magnetizing inductance. The magnetizing inductance, L_m, shunts the primary side of an ideal transformer [11]. [See Figure 4-3]

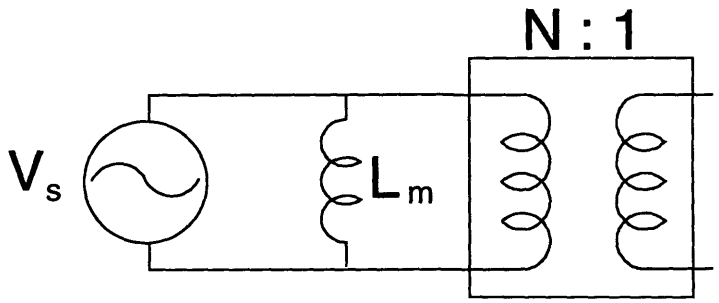


Figure 4-3: Transformer With Perfect Coupling but Finite Magnetizing Inductance

Also in the real system, the coils of the primary and secondary are not perfectly coupled [See Figure 4-4]. Some of the flux that links one coil does not link the second because of losses of flux in the core, in any air gap between the core and the coils, and within the coils themselves. This effect is modeled by placing leakage inductors, L_{lp} and L_{ls} in series with the previous model [See Figure 4-5].

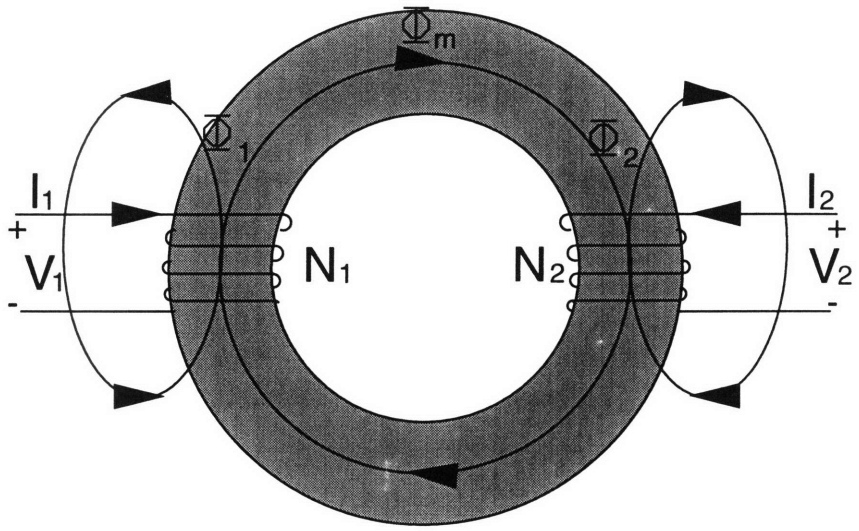


Figure 4-4: A Practical Transformer

Another nonideal characteristic of real transformers is that each of the coils of the transformer has some resistance. These resistances, R_p and R_s, are placed in series in the model in Figure 4-6. This model of a transformer will be used throughout the rest of this thesis.

4.2 Simplifying the Transformer Model

Since the goal of this project is to predict voltage waveforms at the output of the secondary, it would be useful to make any reasonable simplifications of the transformer model. First, we will replace primary side voltage source, winding resistance R_p, and inductances L_{lp} and L_m with a Thevenin equivalent. The Thevenin impedance is

$$Z_p = (R_p + j\omega L_{lp}) \| (j\omega L_m),$$

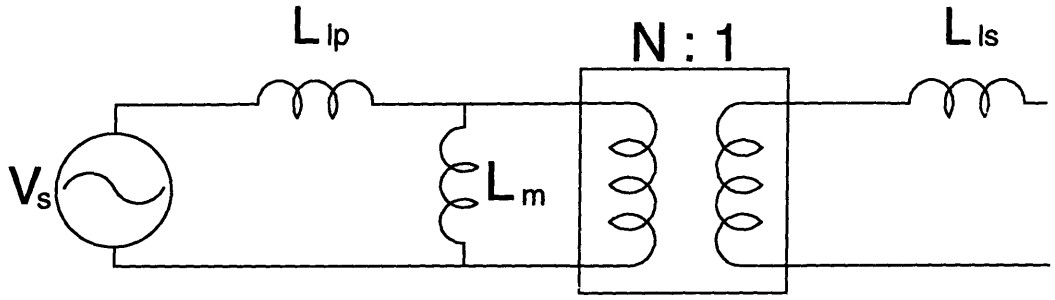


Figure 4-5: Transformer Model Including Leakage Inductance

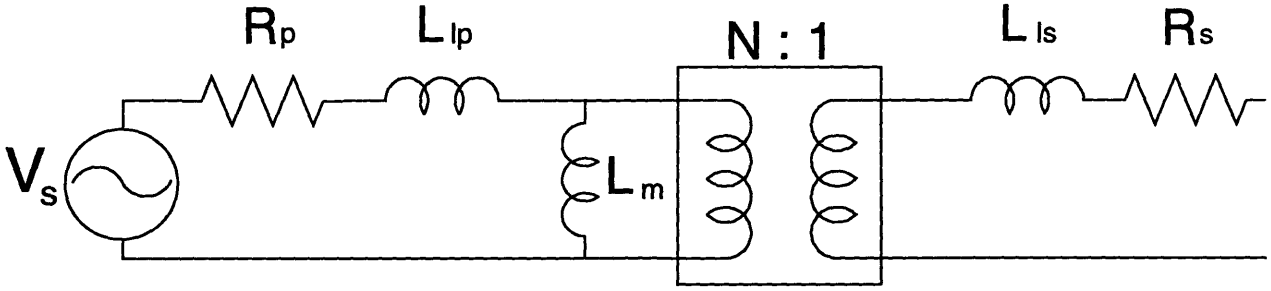


Figure 4-6: Transformer Model Including Winding Resistance

which can be rewritten as

$$Z_p = \frac{j(L_{lp}L_m(L_{lp}+L_m)\omega^3) + R_p^2 L_m(L_{lp}+L_m-L_{lp})}{R_p^2+\omega^2(L_{lp}+L_m)^2}. \tag{4.6}$$

Assuming that $L_m \gg L_{lp}$, Equation 4.6 simplifies to

$$Z_p \approx \frac{R_p L_m^2\omega^2 + j\omega^3 L_{lp}L_m^2}{R_p^2+\omega^2 L_m^2}. \tag{4.7}$$

For a reasonable transformer, $L_m \cdot \omega$ will be much greater than R_p, and $R_p^2+\omega^2 L_m^2 \approx \omega^2 L_m^2$. L_m depends on the characteristics of the materials used, while R_p depends both on the charac-

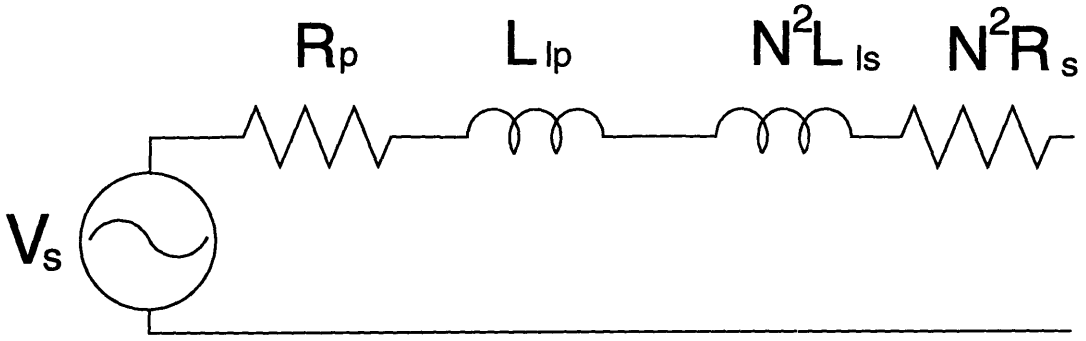


Figure 4-7: Circuit Model, Reflecting Impedances of Secondary and Ignoring L_m

teristics of the materials and the source frequency. Therefore, this may be a bad approximation at high frequencies, but it will be tolerated for low frequencies which are near the transformer rated frequency. Equation 4.7 then reduces to

$$Z_p \approx R_p + j\omega L_{lp} \tag{4.8}$$

The value of the Thevenin voltage source is equal to the open circuit voltage of the impedance divider:

$$V_{thev} = (\frac{j\omega L_m}{R_p + j\omega(L_{lp} + L_m)})V_s$$

Again assuming $L_m \gg L_{lp}$,

$$V_{thev} \approx (\frac{j\omega L_m}{R_p + j\omega L_m})V_s$$

and therefore,

$$V_{thev} \approx V_s. \tag{4.9}$$

Next, the impedances on the secoı dary side of the transformers will be replaced with equivalent impedances on the primary side using Equation 4.5 to eliminate the need for the ideal transformer [See Figure 4-7]. Combining the series impedances, the model simplifies to a

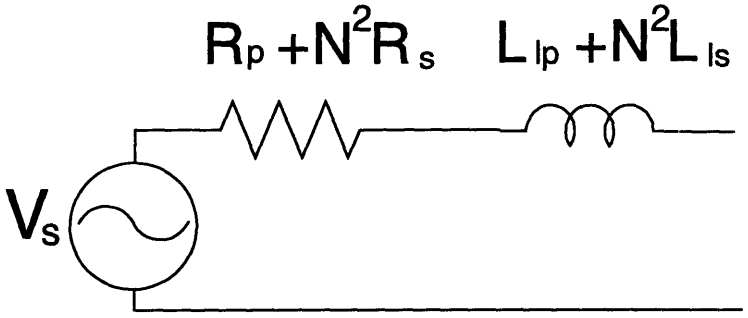


Figure 4-8: Simplified Circuit Model of Real Transformer

voltage source in series with a resistor and an inductor [See Figure 4-8] with the values:

$$V_{thev} \approx V_s$$

$$R_{eq} \approx R_p + N^2 R_s$$

$$L_{eq} \approx L_p + N^2 L_s$$

This is the same model that Beattie examines in [12].

Chapter 5

Determining Supply Impedance

To determine the apparent impedance of a local utility service, we propose to employ the technique suggested in [12]. The supply impedance is determined by closing a capacitor across line voltage at a precise time, and then examining the capacitor current waveform. This transient response is oscillatory, because of the LC combination, and is damped because of the resistor. The angular frequency of the transient will be approximately equal to the undamped frequency of oscillation.

$$\omega = \frac{1}{\sqrt{LC}} \tag{5.1}$$

To measure the supply inductance at various frequencies, different values of capacitors will be used, varying the transient frequency. In order to minimize the number of capacitors necessary to carry out this analysis, a variable capacitor system was created, utilizing switching devices and capacitors placed in parallel. By measuring the impedance at varying frequencies, the user can adjust the capacitance to make $\omega \gg \omega_o$, where ω_o is the angular frequency of the source, to simplify isolating the transient behavior from the base frequency of the driven response. Since ω is determined by L and C, and L is unknown until tests are run, C must be able to vary over a useful range of values.

The second part of calculating the supply impedance is to calculate the resistance. This value can be computed by examining the rate of decay of the oscillatory term of the transient response. Since the rate of decay is exponential, taking the natural log of the decay rate yields a linear term. The slope of this line can be approximated by the relation:

$$Slope = -\frac{R}{2L}. \tag{5.2}$$

5.1 Derivation of Formulas

To derive the formulas used to calculate the supply impedance, one must look at the transfer function of a series RLC circuit. The voltage across the resistor will be examined to eventually find the current flowing through the circuit. The voltage across the resistor can be found by multiplying the input voltage source by the transfer function relating the resistor voltage to the input voltage:

$$V_r = H(s) \cdot V_s. \tag{5.3}$$

where

$$H(s) = \frac{R}{Ls + R + \frac{1}{Cs}}. \tag{5.4}$$

Equation 5.3 can be written as

$$V_r = (\frac{\frac{R}{L}s}{s^2 + \frac{R}{L}s + \frac{1}{LC}})V_s. \tag{5.5}$$

Identification of the unknown component values R and L will be made by examinimg the transient response of the RLC circuit to the transient response after a step in input voltage. To facilitate the extraction of the transient response from the driven part of the total solution, experiments will be configured so that the transient oscillation and decay are much faster than the driven, "60 Hz" component of the solution. The voltage source function used in the test

system is $V_s = V_1 \cos \omega_o t \cdot u(t)$, which has a Laplace Transform of $\frac{V_1 s}{s^2+\omega_o^2}$. For fast transients, e.g., $\omega_o \ll \omega$, the solution is dominated by ω, and the Laplace Transform reduces to $\frac{V_1}{s}$, the equation of a step response to the system. Putting a step in for the voltage source:

$$V_r = (\frac{\frac{R}{L}s}{s^2 + \frac{R}{L}s + \frac{1}{LC}}) \cdot \frac{V_1}{s},$$

which simplifies to

$$V_r = \frac{\frac{R}{L} \cdot V_1}{s^2 + \frac{R}{L}s + \frac{1}{LC}}.$$

Taking the inverse Laplace Transform,

$$V_r = (2V_1 R\sqrt{C})e^{-\frac{R}{2L}t}(\frac{\sinh \sqrt{\frac{R^2C-4L}{4L^2C}}t}{\sqrt{R^2C - 4L}})$$

For $4L \gg R^2C$,

$$V_r = (V_1 R\sqrt{\frac{C}{L}})e^{-\frac{R}{2L}t} \sin \frac{t}{\sqrt{LC}}$$

Solving for the transient current, where $I_r = \frac{V_r}{R}$,

$$I_r = (V_1\sqrt{\frac{C}{L}})e^{-\frac{R}{2L}t} \sin \frac{t}{\sqrt{LC}}. \tag{5.6}$$

Thus, the approximate frequency of the oscillations is given by

$$\omega = \frac{1}{\sqrt{LC}}.$$

By taking the natural log of the peaks of the oscillation, we find

$$\ln I_r = \ln V_1\sqrt{\frac{C}{L}} + \ln e^{-\frac{R}{2L}t}$$

and therefore,

$$\ln I_r = \ln V_1 \sqrt{\frac{C}{L}} - \frac{R}{2L}t.$$

Finally, this gives us the slope of the log of the peaks of the oscillations with respect to time:

$$Slope = -\frac{R}{2L}.$$

5.2 Matlab Simulation of RLC Circuit

To test the equations from [12] and to determine the accuracy to which the values of L and R can be calculated, the analytical solution of the transient was computed using Matlab. To set up the simulation, values of L, R, and C were chosen. To drive the circuit, a simulated cosinusoidal waveform was turned on at a t=0. Next, the voltage across the resistor was calculated so that the current waveform could be determined.

$$V_r = H(s) \cdot V_s \tag{5.7}$$

Taking the Laplace Transform of the source,

$$V_s = V_1 \cos \omega_o t \cdot u(t) \tag{5.8}$$

where V_1 is equal to the amplitude of V_s. Plugging Equation 5.4 and Equation 5.8 into Equation 5.7,

$$V_r = \frac{V_1 \frac{R}{L} s^2}{s^4 + \frac{R}{L}s^3 + (\frac{1}{LC} + \omega_o^2)s^2 + \frac{R}{L}\omega_o^2 s + \frac{\omega_o^2}{LC}}. \tag{5.9}$$

Inserting sample values into Equation 5.9, $R = 0.1$, $C = 300\mu F$, $L = 100\mu H$, $V_1 = 1V$, and $\omega_o = 377$, Equation 5.9 becomes:

$$V_r = \frac{1000s^2}{s^4 + 1000s^3 + 3.33 * 10^8 s^2 + 1.42 * 10^8 s + 4.72 * 10^{12}}. \tag{5.10}$$

Converting this to a time domain expression using a partial fraction expansion and the inverse Laplace transform yields:

$$V_r = 0.175e^{-500t}\sin 5752t - 0.014\sin 377t + 1.29 * 10^{-4}e^{-500t}\cos 5752t + 1.29 * 10^{-4}\cos 377t \tag{5.11}$$

Dividing V_r by R to solve for I_r, and neglecting the cosine terms because they are two orders of magnitude less than the sinusoidal terms,

$$I_r \approx 1.75e^{-500t}\sin 5752t - 0.14\sin 377t. \tag{5.12}$$

After creating a data matrix I_r in Matlab [See Figure 5-1] using Equation 5.12, all information is taken from examining the matrix, to treat the data as if it had been recorded from a physical circuit. By subtracting the estimated steady state response, the transient current was isolated. The angular frequency, ω, for the transient current was measured at 5800 radians per second. From Equation 5.1 and the known value of the capacitor, the inductance was estimated to be 98.5μH, within 2 percent of the actual value. To calculate the resistance in the circuit, the natural log of the peak values of the transient current was taken. The slope of the best fit line was -471. As Equation 5.2 relates this slope to the resistance and inductance of the circuit, R was calculated to be 0.093Ω, within 7 percent of the actual value used to create the simulation.

The errors that were incurred in the simulation were most likely caused by human error in estimating the transient frequency and decay envelope. Note that any error in these times is squared in the calculation of the inductance. The results of the simulation were encouraging, demonstrating that this method for extracting data from current waveforms is plausible.

5.3 Initial Determination of Supply Impedance

To find the approximate values of the supply impedance, an initial test using a digital oscilloscope and a current probe was conducted. The voltage source was turned on at a peak in the

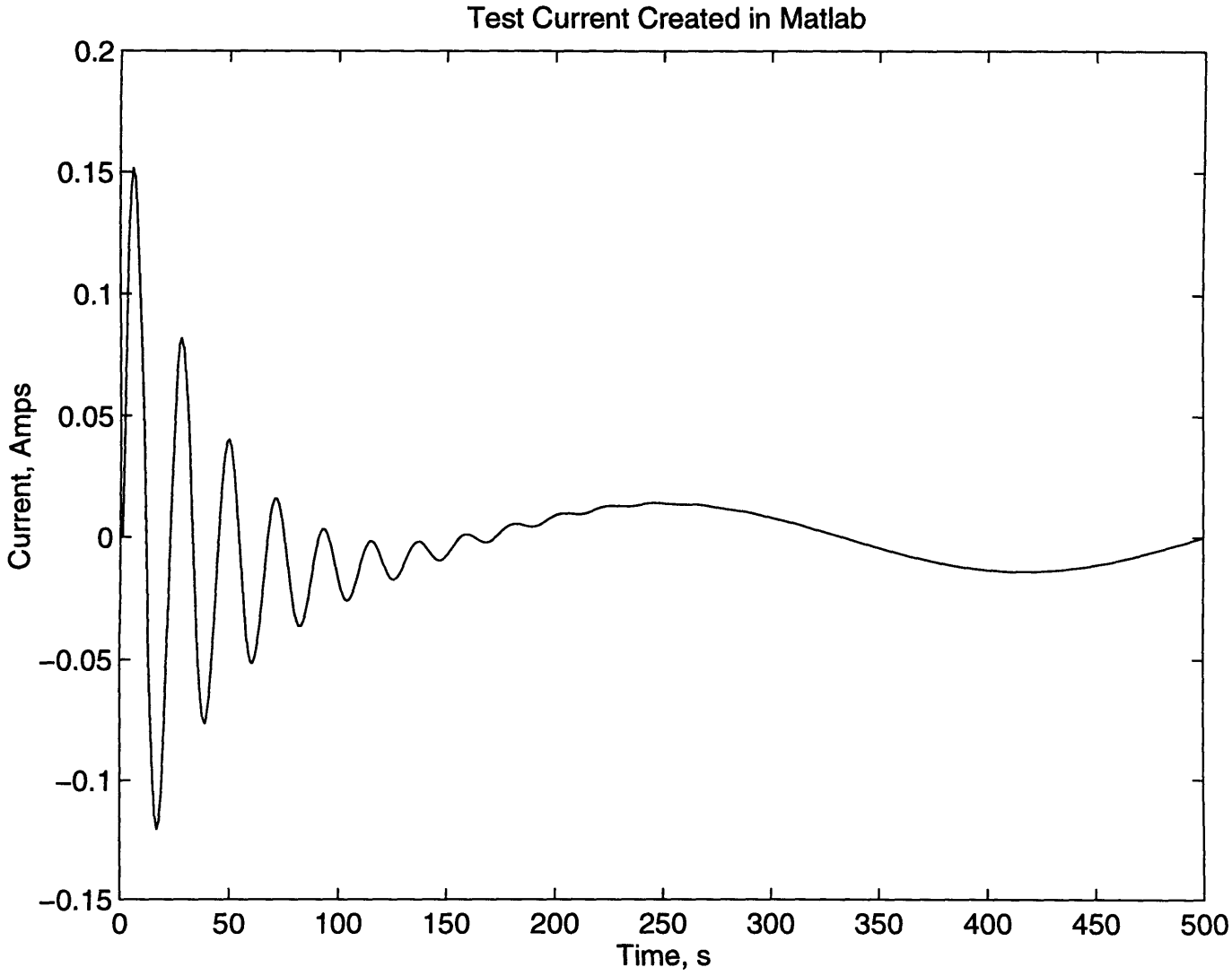


Figure 5-1: Simulated Current in Matlab

voltage by the three phase switch, making it equivalent to a cosine waveform turned on at time t=0. The current waveform was recorded and analyzed [See Figure 5-2]. From Equation 5.1, the supply inductance was found to be 71.5μH, and from Equation 5.2, the supply resistance to be 1.15Ω. Much of this resistance, however, may have been in the leads connecting the capacitor to the load test stand. This value of R would be totally unacceptable because the power dissipated before reaching the wall outlet, I^2R, could reach a kilowatt for currents reaching 30A, i.e., the outlet rating.

Through the simulation in Matlab and the initial determination of parameters, it became apparent that extracting the necessary information in this manner was both time consuming and not precise. An automated method which compensates for parasitic components is introduced in Chapter 6. The automated prototype impedance measurement hardware is called the Determination of Supply Inductance and Resistance (DESIRE) system.

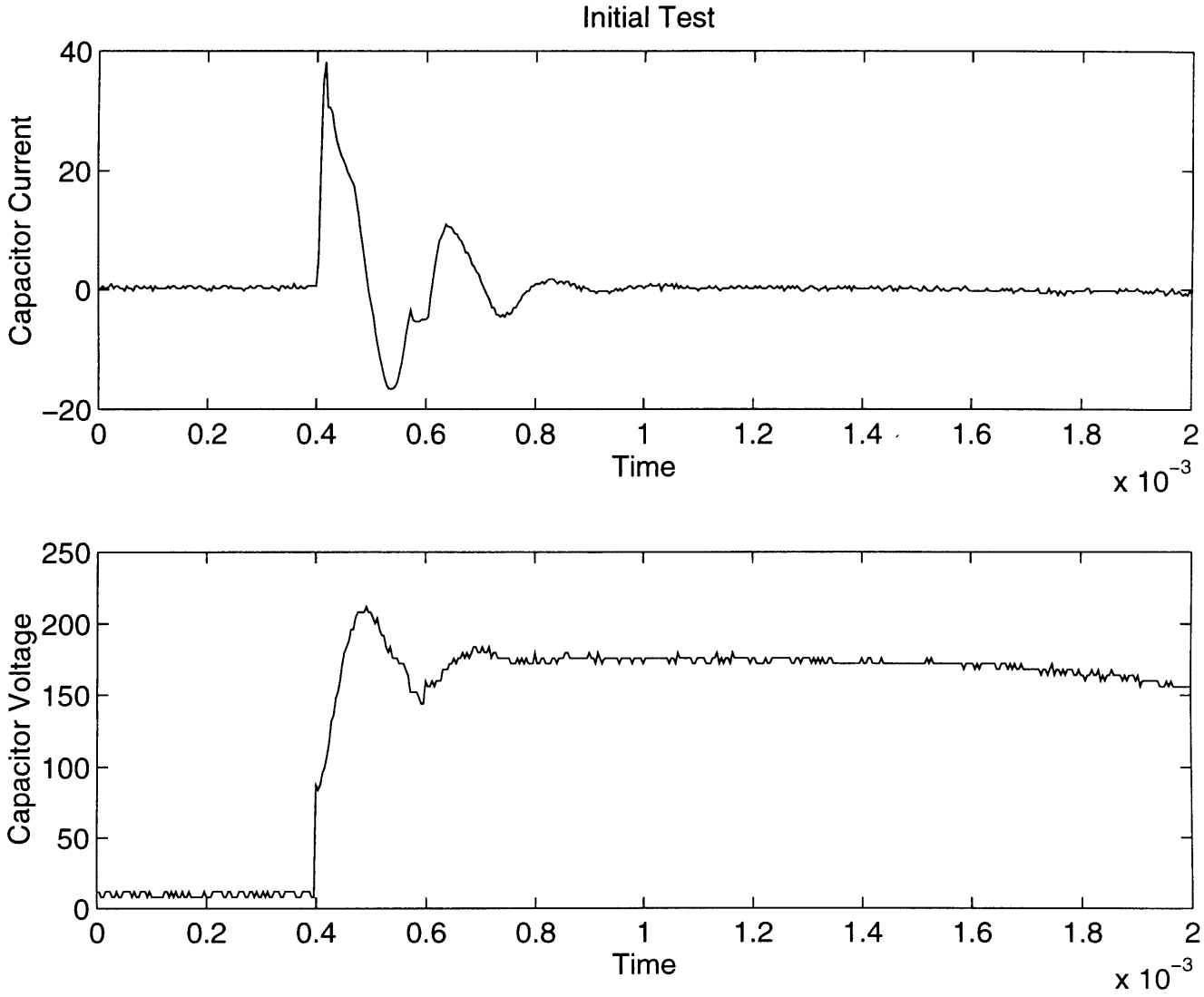


Figure 5-2: Current and Voltage Waveforms of Initial Test

Chapter 6

Documentation of Determination of Supply Inductance and Resistance System

The hardware for the Determination of Supply Inductance and Resistance system (DESIRE) is designed to be mobile and to minimize costs. It would be possible, but not ideal, to take the load test stand, a current probe, a digital oscilloscope and a PC to each location where the utility impedance is to be calculated. To increase the mobility of the equipment, a PC card was designed and constructed to eliminate the need for the current probe and oscilloscope.

The PC card controls the time that the switch based is activated based on inputs from the user and the voltage sensor board. Once the load test stand is activated, current flows through the capacitor and a current sensor. The current sensor measures a scaled version of the current. The output of the sensor is digitized in the computer card and stored in memory. This data is then downloaded from the memory to a file on the PC. The specifics of the load test stand and voltage sensor were described in Chapter 2. This chapter will describe the PC card used in DESIRE, the software that controls the card, and the variable capacitor box.

6.1 PC Card

The Maxim 122, a BiCMOS 12-bit analog-to-digital converter (ADC) [13], serves as the heart of the PC card [See Appendix A for schematic diagram]. There are three possible methods of operation for this ADC, controllable by a mode pin on the chip. Mode 0 (mode = V_{dd}), used for this project, signals the completion of a conversion by a positive to negative transition of the $\overline{INT}$ signal. This mode also requires the user to specifically activate the output of the ADC so that it can be read. In Mode 0, the output data lines on the ADC remain in a high impedance state unless both $\overline{CS}$ and $\overline{RD}$ are selected, so that the data lines do not need to be buffered.

A measurement cycle occurs as follows. The analog input signal is sampled and the conversion of data is initiated on the falling edge of the $\overline{CONSTV}$ signal. The conversion process takes 15 clock cycles, at which point the $\overline{INT}$ pin goes low. At this time, digital data can be read from the ADC by asserting $\overline{CS}$ and $\overline{RD}$. As the data is read, $\overline{INT}$ goes high and the ADC is ready to begin another cycle.

The 8254 [7] is a programmable interval timer/counter that is used both to activate the switch board and syncronize the ADC operation to regular sample instants. The 8254 takes input from the software and from a clock and delays for the programmed number of counts before interrupting the CPU. The 8254 can be configured in any of 6 modes, including an interrupt on terminal count, hardware retriggerable one-shot, a rate generator, a square wave generator, binary rate multiplier, and a complex wave generator. DESIRE uses an 8254 circuit is the interrupt on terminal count mode to activate the switch board. This mode keeps the output low until the programmed number of clock cycles have elapsed, at which point the output becomes high. The second mode used by DESIRE is the "rate generator" to control the rate at which the ADC samples the input.

To control the counting of the 8254, there is a gate line for each of the counters. When the gate is high, the counter is active. When the gate is low, the value of the counter is held,

but does not increment. To control the gates of the counters, a 74HCT574 latch is used. The clock for the latch is the V_{sense} line from the voltage sensor board, a square wave in phase with the reference voltage waveform [See Section 2.2.2 and Appendix A]. The input signal for this latch is a software controlled line from the 573 input buffer. When this line is low, the gate is low. When the input is high, gate is asserted on the next rising edge of the V_{sense} to guarantee that the 8254 starts counting at a positive zero crossing of the reference voltage.

To control and isolate the output of the 8254's, another 574 acts as a buffer. When the latch is selected, it can be considered an ideal buffer, since it is clocked at a high frequency relative to the resolution of the load test stand, isolating the output of the 8254. However, when the latch is not selected, the output of the 574 is in tristate, and does not activate the switch board. The select pin on the latch is controlled by an 573 latch.

The Random Access Memory that is used in this circuit is an Intel 62256, an 8x32K storage device [14]. The FSM initiates writing to memory by activating $\overline{E}$, $\overline{W}$, and $\overline{G}$. The read state of the RAM is controlled by the software program "rob471".

Since the ADC has twelve data lines and the 62256 has only eight, two addresses in RAM must be used to store one data point. One way of doing this would be to write to two consecutive addresses in the RAM. This could be done by buffering the 12 data lines from the ADC. One word could be written to one address and the second would be delayed until the first word is written and the address is incremented. A simpler solution for writing the data is to use two RAM chips in parallel, using the same address for each word. The tradeoff for ease in writing is that it is slightly more difficult to download to the PC, since the data bus has only eight lines. Therefore, the data read from the RAM is latched with 574's. The enables of the latches are controlled by the software interface to avoid contention on the data bus. For the purposes of this project, the second option is used the writing to memory is time constrained, while reading from memory is not.

To control the address for the RAM, two 74HC4040 asynchronous twelve bit counters are

configured to create a 16 bit counter [15]. The first 15 bits are used to increment the address fo the RAM. When the sixteenth bit goes high, each address in memory has been written to, and the stop bit on the FSM is activated to avoid overwriting data. The counters can be reset by the software interface by activating $\overline{CS5}$.

The 74HCT573 is an eight bit synchronous latch with an available tristate feature [15]. Two 573s are used in the circuit. One of the chips is used to control the relays in the capacitor bank. When $\overline{CS1}$ is activated, the data bus values are held in the latch to control the relays. The second 573 is used to control the read/write states of the FSM and the RAM, to initiate sampling data, and to download data to file.

The Finite State Machine controls the timing of the circuit [See Figure 6-1]. This state

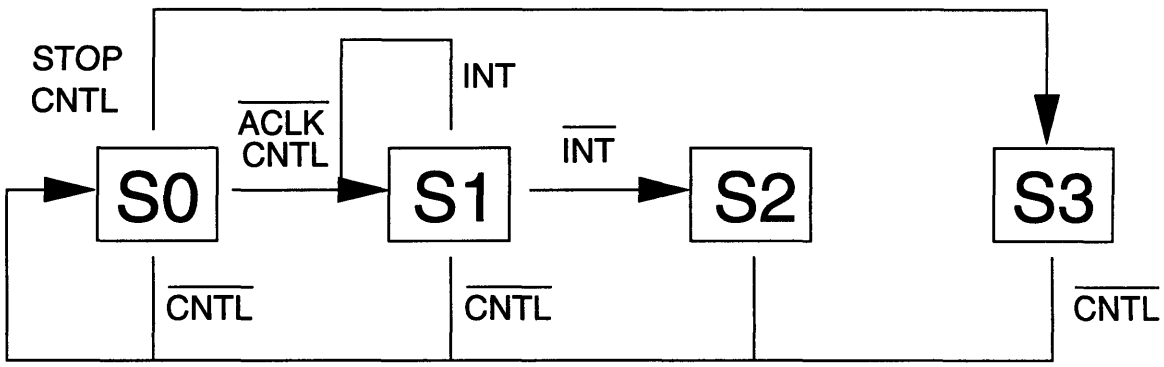


Figure 6-1: Block Diagram of Finite State Machine

machine takes as inputs *aclk* from the PC card clock, *stop* from the counter, and *inc* and *cntl* from the software interface. There are four states in the state machine. The first state. S0, is the wait state. In S0, all of the chip selects and enables are high, so neither the RAM nor the ADC is active. As long as *cntl* is low, the state machine remains in S0. When the software activates *cntl* and *aclk* is low, indicating that the ADC can begin a new conversion, the state machine advances to state S1. On the transition between S0 and S1, the counter is incremented

as the *count* output goes low. Incrementing the counter at this time allows the ripple of the 4040's to settle before writing to the RAM. Once in S1, the state machine will remain there until $\overline{INT}$ goes low, signaling the completion of the a-d conversion, at which time the FSM enters S2. S2 lasts for one clock cycle, during which $WERAM$, $CSADC$, and $RDADC$ are all low, writing the 12 bits of data from the ADC to the RAM. After the data is written to the RAM, the state machine returns to S0. When 32k data points have been written to the RAM, the *stop* is activated. If *cntl* and *stop* are high while in S0, the next state will be S3 to disable further a-d conversions. This state keeps CSADC, RDADC, and WERAM high while enabling the output of the RAM by putting OERAM low, thereby allowing the PC to download data from the memory to file. When *cntl* is asserted low, the state machine returns to S0.

6.2 Software for Capacitor Card

The software used to control the circuitry relies on communication with the computer card. The data bus line on the card in the computer can be controlled by the Microsoft C command "outp()", which takes two arguments. The first argument, the port, sets the parameters on the card, such as A0, A1, and the chip select, while the second controls the eight data lines. To read the data lines on the card, the C command "inp()" is used, which takes only one argument to control the port [8].

The program "rob471" [See Appendix C.2.3], used to communicate with the card, reads information saved in the file "data.cap" that corresponds to relay settings for the capacitor box. The appropiate relays are closed to select the desired capacitor value. After "rob471" initializes the timing circuitry to set the turn on point and rate of conversion, the PC card is signalled to begin the data collection process on the next line cycle. Following a time delay to insure conversion is complete, the data in memory is converted to decimal, scaled, then written to file.

Variables in "rob471" that the user can change for different applications include *work*,

len, *constv*, *pdelay*, and *res*. The variable *work* is used to define the file to which the output of the PC card is written, while *len* defines the number of data points written to *work*. *Constv* is the number of clock signal (from the on board oscillator) per a-d conversion, thereby setting the sample rate. Because of the limitations of the ADC, *constv* must be greater than 15. *Pdelay*, corresponding to a number of clock cycles, sets the time that the switching device is activated. *Res* is the value of the gain resistor in the capacitor box, and is used in scaling the data in memory.

6.3 Capacitor Box

To utilize the PC card, a capacitor needs to be placed across line voltage and the current needs to be measured. Since the supply impedance can vary with location, the frequency of oscillation of the transient is not known for a given capacitor until the test is done. Therefore, the DESIRE capacitor system was designed to permit easy, flexible variation of the capacitor to be closed across the line. [See Figure 6-2].

To accurately sample the data with the ADC, a minimum of ten samples per transient period were desired. As the ADC samples at 200kHz, the maximum frequency that can be accurately sampled is 20kHz. To insure this design specification is met, a parallel array of capacitors was created to allow a variety of capacitances to be selected, specifically, values of .5μF to 100μF. This range of capacitors enables the system to theoretically measure supply inductances as low as 0.03mH. Rearranging Equation 5.1 to solve for L,

$$L = \frac{1}{C\omega^2}.$$

To minimize L, we will insert $2\pi \cdot 20kHz$ for ω that insures the desired number of samples per transient are taken:

$$L \geq \frac{1}{C * (2\pi \cdot 20kHz)^2}. \tag{6.1}$$

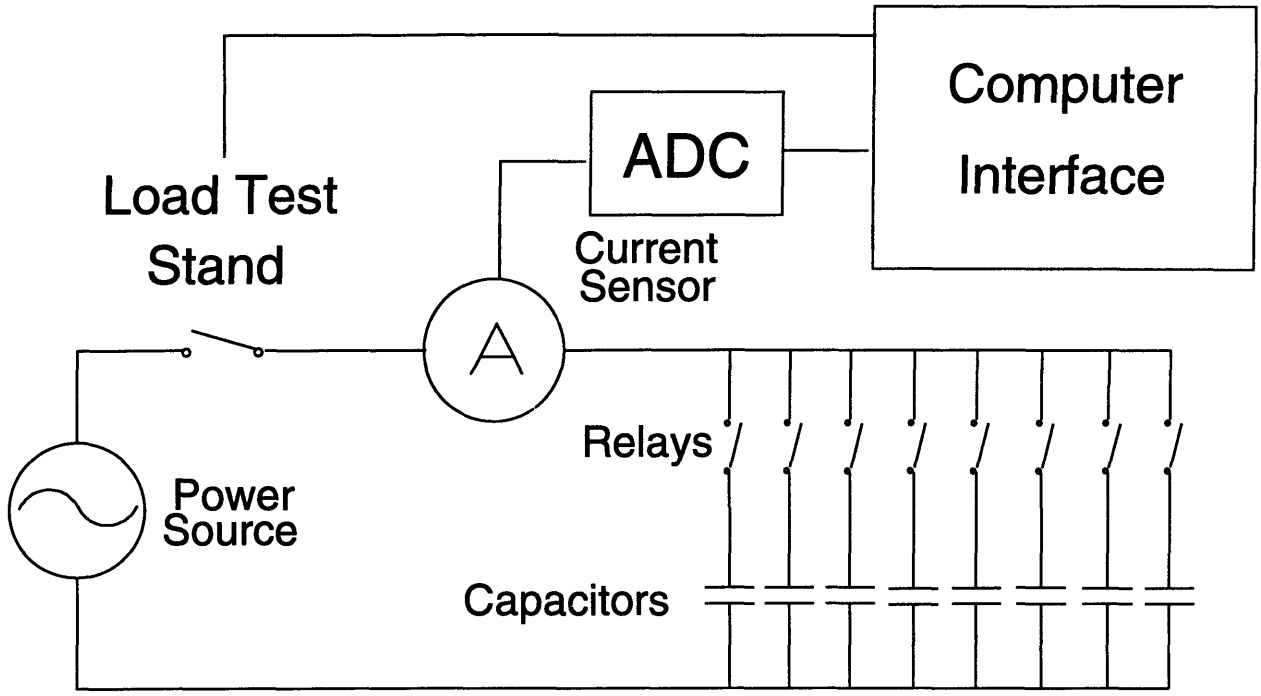


Figure 6-2: Schematic of Capacitor Box

Using larger capacitor values reduces the frequency and increases the amplitude of the current, allowing better resolution of the sampled current. However, the transient current proportional to square root of the capacitor divided by the inductor [See Section 5.1]. Therefore, increasing the capacitor value can lead to extremely high transient currents when it is of the same order of magnitude as the supply inductance. To keep the transient current under 100A when using $120v_{rms}$ ($170v_{p-p}$) line voltage:

$$100A \geq 170V\sqrt{\frac{C}{L}}.$$

Solving for L,

$$L \geq 2.86C. \tag{6.2}$$

Relay Setting	Capacitor Value
1	0.54μF
2	0.99μF
4	2.89μF
8	4.95μF
16	14.95μF
32	25.4μF
64	52.3μF

Table 6.1: Capacitor Values for Relay Settings

Combining Equations 6.1 and 6.2, and minimizing L, we find that the DESIRE system should be able to safely and accurately extract parameters of supply inductances as low as .01mH.

To be able to control the capacitor value used in the system, seven capacitors were placed in parallel. Each capacitor is in series with a relay [See Table 6.1] so that each capacitor can be turned on or off individually [See Section 6.1].

Each relay can is controlled by a 573 latch on the A-D Card. Specifically, one side of the coil of the relay is connected to +15V and the other to a collector of a npn transistor. The base of the transistor is connected to the output the appropriate pin on the 573 through a 4.7kΩ resistor, and the emitter of the transistor is then connected to ground. When the latch output is high, this circuit draws approximately 100mA through the transistor and relay coil, closing the relay. When the latch output is low, no current is drawn through the transistor and coil, opening the relay.

To record the current flowing through the capacitor, an F.W. Bell CL-100 hall effect current sensor is placed around the input to the parallel capacitors. The sensor output, a current equal to 1/1000th of the capacitor current, is shunted to ground by a resistor. The voltage across the resistor is then equivalent in magnitude to the sensor output current multiplied by the value of the resistor. This voltage is then buffered with a follower op-amp, then sampled by the ADC. Since the ADC takes an input signal of -5V to +5V, the value of the resistor must be selected appropriately to insure that the entire input signal is sampled. For a current

of 100A, the maximum desired current, the sensor output is 0.1A. A gain resistor of 47Ω was chosen to utilize most of the full range of the ADC at this large current. Most applications of the system will not use 100A, so a larger gain can be used. For these cases, a switch was included in the system to allow the user to change the gain resistor to 75Ω, 100Ω, or 150Ω.

Chapter 7

Experimental Results

7.1 Calibration

To accurately determine the utility service parameters, the DESIRE system must be calibrated to account for parasitic resistances. The switch box and the capacitor box both introduce undesired, parasitic resistance through wiring interconnections and also capacitor series resistance ("ESR"). These resistances will appear in the impedance that is calculated when extracting R and L from the collected data and must be subtracted off to find the actual utility service values.

Determining the parasitic resistance in the switch involved measuring the resistance with a voltmeter. The value of this parasitic resistance is about 0.1Ω. Determining the resistance in the capacitor box could not be done this way because the impedance is mostly capacitance, and therefor has infinite impedance at DC. A Hewlett-Packard 4192A LF Impedance Analyzer [16] was used, which can calculate the magnitude and phase of an impedance at a wide range of frequencies. An internal function in the impedance analyzer isolates the value of the real part of the impedance, the equivalent series resistance ("ESR"), and returns this value to the user. This calibration data was recorded for several relay settings by driving at between 250Hz and 3kHz [See Tables 7.1 and 7.2].

Freq (Hz)	RS 7 Ω	RS 9 Ω	RS 11 Ω	RS 15 Ω	RS 19 Ω	RS 23 Ω	RS 27 Ω
250	1.00	2.00	1.81	1.11	0.51	0.46	0.54
500	0.61	1.20	0.97	0.65	0.38	0.36	0.39
750	0.54	0.78	0.65	0.49	0.35	0.33	0.34
1k	0.47	0.58	0.50	0.41	0.33	0.31	0.32
1.25k	0.45	0.48	0.43	0.37	0.32	0.31	0.31
1.5k	0.43	0.42	0.39	0.35	0.31	0.30	0.30
1.75k	0.42	0.39	0.36	0.33	0.30	0.30	0.29
2k	0.40	0.36	0.34	0.32	0.30	0.29	0.29
2.25k	0.40	0.34	0.32	0.31	0.30	0.29	0.28
2.5k	0.39	0.33	0.31	0.31	0.29	0.29	0.28
2.75k	0.38	0.32	0.31	0.30	0.29	0.29	0.28
3k	0.38	0.31	0.30	0.30	0.29	0.29	0.28

Table 7.1: Calibration of Capacitor Box

Freq (Hz)	RS 31 Ω	RS 35 Ω	RS 40 Ω	RS 45 Ω	RS 50 Ω	RS 55 Ω	RS 60 Ω	RS 65 Ω
250	0.49	0.44	0.45	0.41	0.33	0.32	0.33	0.37
500	0.37	0.37	0.36	0.33	0.28	0.27	0.27	0.34
750	0.33	0.36	0.34	0.31	0.27	0.26	0.26	0.33
1k	0.31	0.35	0.32	0.30	0.27	0.26	0.25	0.32
1.25k	0.30	0.34	0.31	0.29	0.26	0.25	0.25	0.32
1.5k	0.29	0.33	0.31	0.29	0.26	0.25	0.24	0.32
1.75k	0.29	0.33	0.31	0.29	0.26	0.25	0.25	0.32
2k	0.29	0.33	0.30	0.28	0.25	0.24	0.24	0.31
2.25k	0.28	0.32	0.30	0.28	0.25	0.24	0.24	0.31
2.5k	0.28	0.32	0.30	0.28	0.25	0.24	0.23	0.31
2.75k	0.28	0.32	0.30	0.28	0.25	0.24	0.23	0.31
3k	0.28	0.32	0.29	0.27	0.25	0.24	0.23	0.31

Table 7.2: Calibration of Capacitor Box

7.2 Extraction of Data

Once transient current waveform from the capacitor box has been downloaded to file, the desired information needs to be extracted from this data. Because of its numeric analysis capabilities and flexibility, Matlab was chosen as the computational tool for analyzing the measured transient waveforms. There were two possible methods explored in this thesis for extracting the supply impedance from measured transient waveforms. The first method, suggested in [12], involved measuring the frequency of transient oscillation and the rate of decay of the oscillation, as done in Section 5.3. An alternative method for extracting parameters uses an internal function in Matlab, "fmins", to minimize the difference between a function and a reference waveform by adjusting variables in the function.

Several script files, refered to as M-files, have been written to automate the process of extracting the supply resistance and impedance. The main file, "robir.m", declares several variables that will be used as matrices within the file and by other m-files [See Appendix C.2.4]. A loop is then initiated to process each data file that has been downloaded. The first step in the loop is to load the file from the harddrive into Matlab. Next, the loop extracts the value of the capacitor from the first line of the file, and assigns the time and recorded data to the matrices t and $Data$.

The Matlab command "fmins" [17] is used to optimize the predicted current wavefrom based on Equation 5.6. The command takes as inputs a function, an initial guess of the parameters it is to optimize, and a tolerance level. "Fmins" adjusts each of the parameters of the function to minimize the value of the function. The function that is used to simulate the current waveform, given in the file "decay4.m", is

$$f = offset + k \cdot e^{-\frac{Rt}{2L}} \cdot \cos \frac{t}{\sqrt{LC}},$$

slightly modified from Equation 5.6 to work best with fmins. The offset was added because

there appeared to be a slight DC offset in the system. The other modification to Equation 5.6 was to use cosine wave to instead of a sine wave. "Fmins" proves to be very sensitive to the starting point of the function. Because of noise in the system, the turn on point of the switch could not be determined as precisely as was needed. A much more accurate and reliable method of determining a starting point was to find the peak of the real current waveform, truncate data before this point, and use a cosine wave as an equivalent function:

$$k \cdot e^{-\frac{Rt}{2L}} \cdot \sin \frac{t}{\sqrt{LC}} = k \cdot e^{-\frac{Rt}{2L}} \cdot \cos \frac{t + \frac{\pi}{2}}{\sqrt{LC}}.$$

Substituting in $\tau = t - \frac{p}{2}$ and simplifying,

$$k \cdot e^{-\frac{Rt}{2L}} \cdot \sin \frac{t}{\sqrt{LC}} = k \cdot e^{\frac{R\pi}{4L}} \cdot e^{-\frac{R\tau}{2L}} \cdot \cos \frac{\tau}{\sqrt{LC}}$$

Since k is a constant to be determined, $e^{\frac{R\pi}{2L}}$ can be included in the term k to simplify the calculations. "Fmins" then optimizes the system function by adjusting R, L, $offset$, and k.

The parameters estimated by "fmins" are recorded and used to create the estimated current waveform using the function "dec5." This waveform is then plotted along with the real data to evaluate how well the parameters have been predicted.

7.2.1 User Interface

The user interface for DESIRE was developed by Tom Respress using a Microsoft C compiler, version 6.0 [18]. The interface has been designed to simplify the process of collecting data and estimating impedances while presenting the data in an attractive graphical format. When the "desire.exe" file is run, the graphical user interface is entered. In this interface, the user has the ability to acquire new data, save the data, and load previously saved data. The interface contains area for four plots of data, a text area that displays the estimated R and L, eight capacitor setting buttons, and seven function buttons. At all times in the program, all buttons

and four plots are shown, but the data in the plots and the functionality of the buttons depends on the status of the program. As the program is entered, all capacitor and function buttons appear, as well as four blank plots[See Figure 7-1].

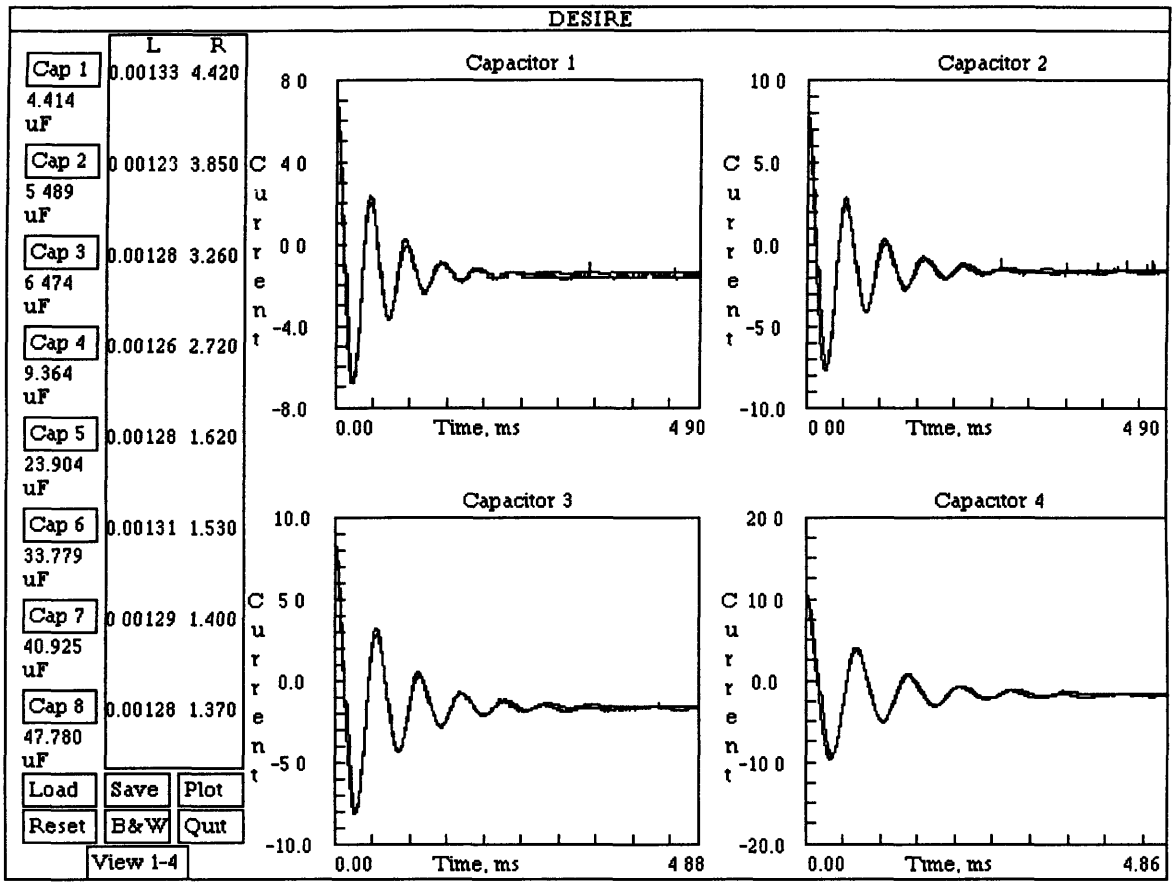


Figure 7-1: Picture of DESIRE user interface

Each of the buttons can be "pressed" by clicking on it with the mouse or by hitting a keyboard shortcut. For each of the capacitor buttons, this shortcut is the number of the capacitor. For the function buttons, the shortcut is the first letter of the button name (ie. the shortcut for **Cap 1** is "1", and the shortcut for **Plot** is "p").

The capacitor buttons control the capacitor values that will be placed across line voltage when data is collected. Clicking on a capacitor with the left mouse button will increase the

value by about $0.5\mu F$, while the right button will increase it by about $5\mu F$. Using the keyboard shortcut will increase it by $0.5\mu F$. The values of the capacitors can be changed only when no plotted data is shown on the screen.

The **plot** button interrupts the graphic interface to collect data using the capacitor values that are currently displayed. The "rob471" program is called, and each capacitor value is individually closed across line voltage, and the transint current waveforms are recorded. At the completion of this process, matlab is run using the "matrob.m" file as the matlab startup program. The m-file "robir.m" is called to estimate the supply resistance and inductance [See Section 6.2]. The graphical interface is reentered and the real and simulated data are plotted in the graphs, and the estimated supply impedance is displayed in a text area. The data plotted is saved found in the files with the root name "data". **Plot** can be run at any time.

The **view** button allows the user to toggle between viewing the plots of the first four sets of data and the last four sets of data. **View** can only be used when data is plotted on the screen.

The **save** button saves data that is currently plotted to files with the root name "svdat". If there are "svdat" files at the time that save is called, these files will be written over. The **save** button is functional only when data is plotted on the screen. The **load** button copies the "svdat" files to "data" files and plots this data on the screen. **Load** can be called at any time, but assumes that there the "svdat" files exist and were created by the DESIRE program. The information contained in the "data" files will be written over when **load** is called.

The **reset** button clears the plots and the text area, then resets the capacitor buttons to the settings in "default.cap". **Reset** can be pressed at any time. Pressing **reset** allows the user to change the capacitor settings to collect new data once plot has been run.

The **B&W** and **Color** buttons toggle between the color setting and the black and white setting. The black and white setting has been implemented for better pictures, and appears better on a black and white monitor than the color setting does. These buttons can be pressed

at any time, and change only whether the screen appears in color or black and white.

The **quit** button returns the user to the dos prompt, and can be called at any time. The data that has last been used by DESIRE still remains in the "data" files.

7.3 Isolation Transformer Data

The method chosen for testing DESIRE was to estimate the impedances of a transformer using DESIRE, then compare these values to impedances measured using an impedance analyzer. To accomplish this task, a Deltec Model DT100R single phase isolation transformer with a 1:1 turns ratio was used. This platform offered access to both the primary and secondary sides of the transformer, simplifying the reference measurements of its internal impedances. Once the supply impedances were calculated, they were used to predict voltage distortion at the secondary of the transformer.

The impedances of the isolation transformer were measured using the HP Impedance Analyzer. By looking at only the primary side of the transformer, the resistance in the primary winding, R_p is determined taking at the real part of the impedance, and similarly for R_s on the secondary side. Tables 7.1 and 7.2 show the resistances on the primary and secondary side for a range of frequencies. The imaginary part of this impedance looking at the primary side corresponds to the sum of the leakage inductance and the magnetizing inductance.

$$\Im(Z) = L_l + L_m \tag{7.1}$$

This inductance, which did not change as the driving frequency was varied, was equal to 124mH. With the secondary side shorted and using Equation 4.5, the transformer model can be reduced, as done in Figure 4-7. Consequently, the magnetizing inductance is in parallel with the series combination of R_s and L_{ls}. Driven at a frequency of 60Hz, the impedance of L_m is much greater than the impedance of R_s and L_{ls}. Consequently, the inductance looking into

the primary side is the total leakage inductace of the transformer, which is equal to 1.3mH. Substituting into Equation 7.1, we see that $L_m = 123mH$.

7.4 Transformer Model Adjustments

When current waveforms drawn through the isolation transformer were recorded, it became clear that our model for the transformer was incomplete. Specifically, an unexpected ringing in the transient wavefrom and a frequency dependent resistance could not be explained in this model. Therefore, the model for the transformer needs to be revised.

7.4.1 Winding Capacitance

Upon close examination of the transient current of the capacitor box, an unexpected, high frequency oscillation was discovered [See Figure 7-2]. Since any additional series capacitance or inductance would not increase the order of the system, the oscillation must result from a parallel LC combination. To this point, the model used does not include such a combination. By looking at the physical design of a transformer, we see that the windings of the transformer are close together and are very numerous. There is some capacitance between each of the windings of the transformer. [See Figure 7-3.] Since there are "capacitors" across consecutive and nonconsecutive winding pairs, there capacitances are in parallel and in series with each other. A single capacitance can be used to represent the combination of capacitances. The capacitance between individual windings is extremely small, but the total capacitance of the system may be enough to cause the observed oscillation. The capacitance added to the model will represent capacitance between windings of the primary with other primary windings (C_p), capacitance between windings of the secondary with other secondary windings (C_s), and capacitance between the primary and secondary windings (C_m). Adding these capacitors to the model results in the model seen in Figure 7.4.1. To try to determine these capacitances, the HP Network Analyzer was used to evalutate the driving point impedance and the transfer function

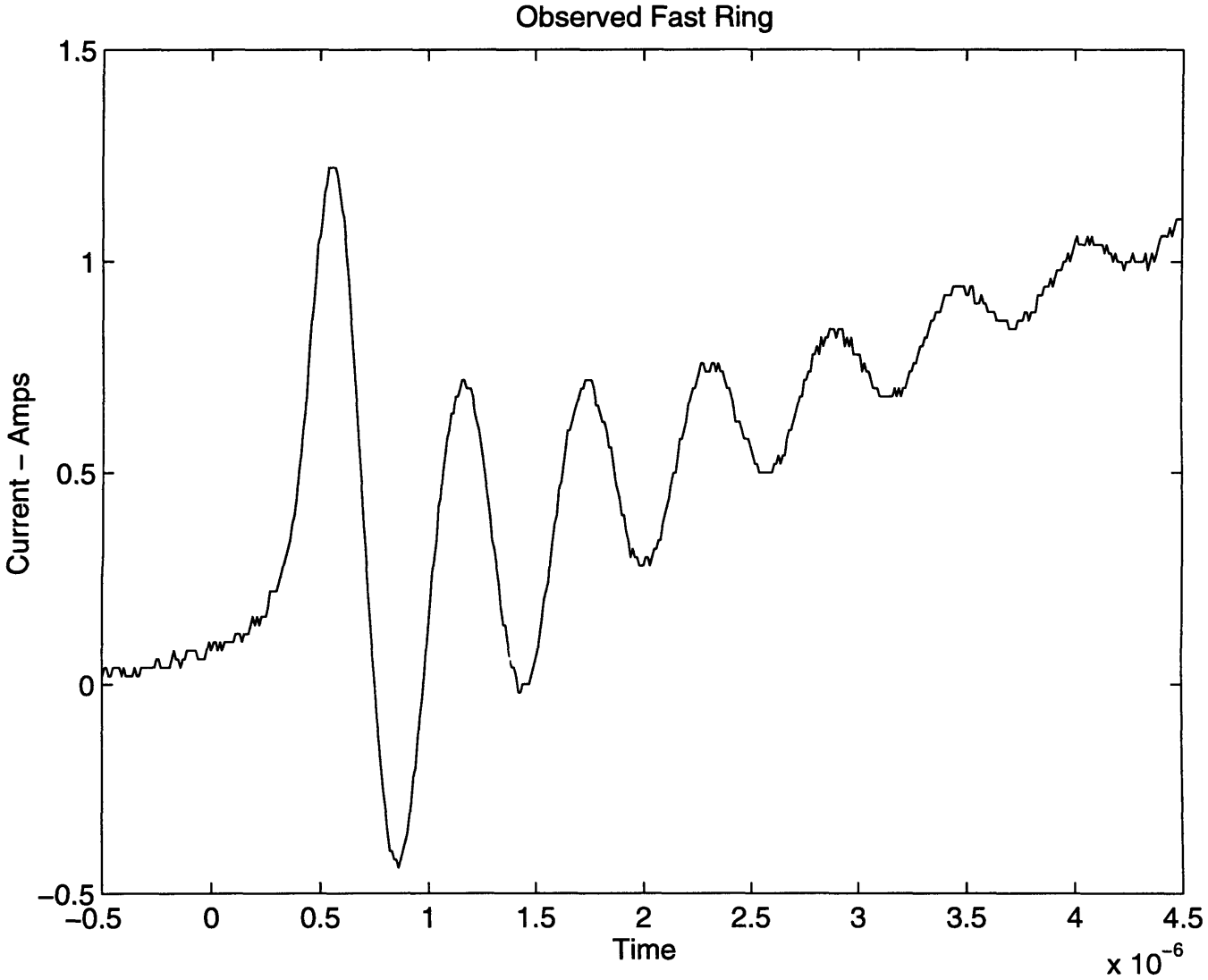


Figure 7-2: Unexpected Oscillation

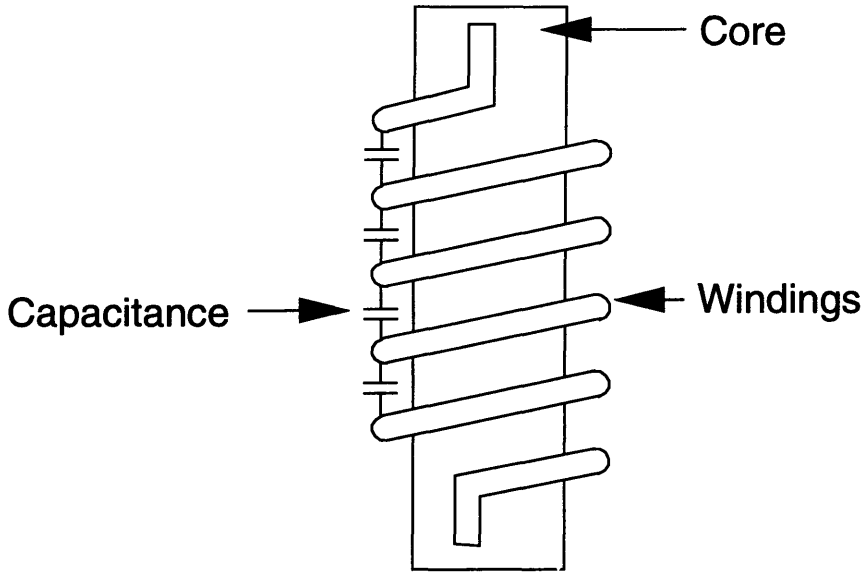


Figure 7-3: Winding capacitance

between the voltages at primary and secondary sides of the transformer. Using the resistor and inductor values previously measured, the capacitor values were adjusted to fit the transfer function and the driving point impedance plots. Since the transformer is 1:1, C_s will be approximately equal to C_P to simplify the problem. The resulting values are $C_s = C_p = 0.01nF$ and $C_m = 22.4nF$. These capacitances appear quite reasonable for the windings of the transformer.

Using the circuit simulation program Pspice, the new model was tested to check approximations. Using C_s, C_m and C_p, the current waveform did not closely match the observed current in the real system. By eliminating C_s and C_m from the model, the simulation closely approximates that of the real system, which suggests that the winding capacitances can be lumped into a single capacitance. More research needs to be done in this area to determine the exact cause of this high frequency oscillation. The high frequency oscilliation was of a much smaller magnitude that the entire transient waveform, and had a much faster time constant. Therefore, this high frequency oscillation will not affect the results of DESIRE.

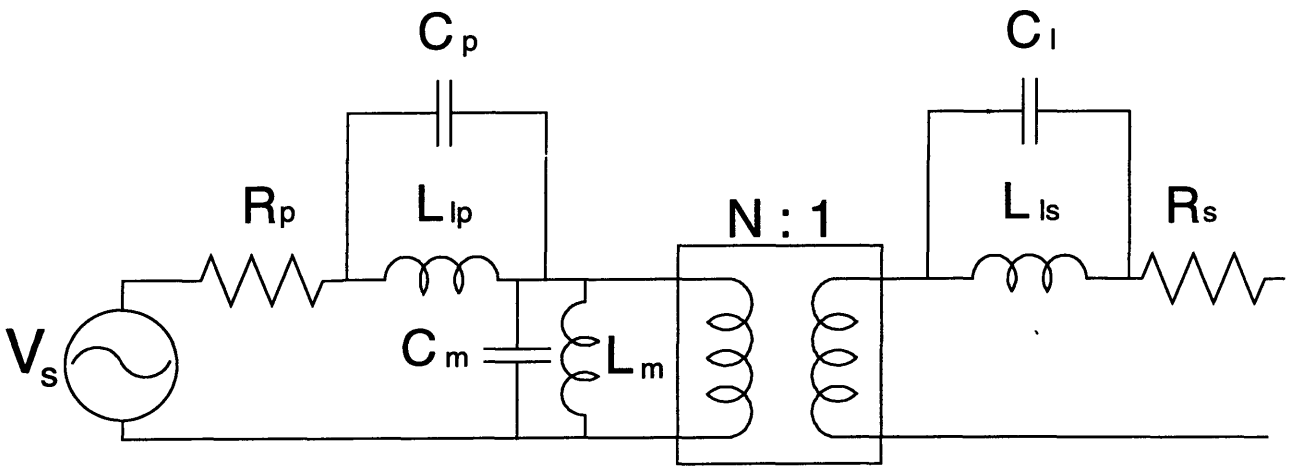


Figure 7-4: Transformer Model with Winding Capacitance

7.4.2 Skin Effect

Another aspect of the transformer that becomes important when utilizing relatively high frequency transients to determine the supply impedance is the skin effect. Current in conductors travels in plane waves. At low frequencies, it is reasonable to approximate that the current is distributed uniformly in the conducting region. However, at higher frequencies, the current density varies significantly with location within the conductor. The skin depth, a measure of how quickly the wave is attenuated within the conductor is useful in calculating resistances at high frequencies. From [10], the skin depth, d, is found to be

$$d = \sqrt{\frac{2}{\mu\mu_o\sigma\omega}}, \tag{7.2}$$

where μ is the relative permeability, σ the electrical conductivity, and ω the angular frequency of the current waveform. This approximation holds only when the skin depth is much less than the radius of the conductor.

In this model, almost all of the current flows in the outer region at high frequencies

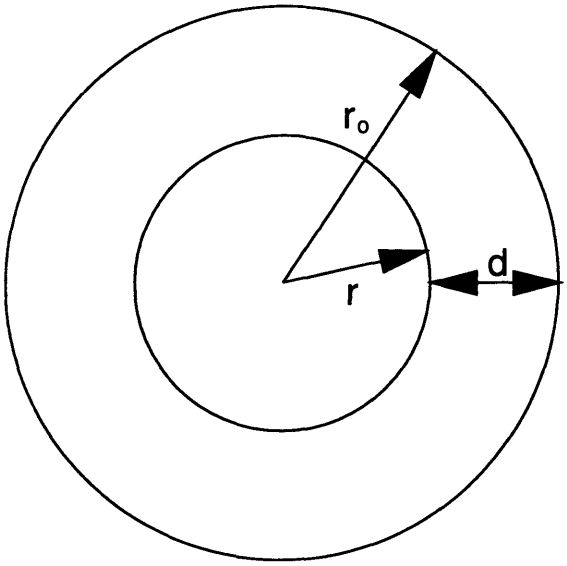


Figure 7-5: Cross-section of Wire to Show Skin Depth

[See Figure 7-5]. For the purpose of estimating the resistance in the wire, the radius of the wire is assumed to be much greater than the skin depth. This allows the approximation that the current density is constant within the surface layer. This information can be used to find the relationship between frequency and resistance for a cylindrical wire. For any material, resistance is equal to the length of the material divided by the conductivity times the area of the conductive matter:

$$R = \frac{l}{A\sigma}. \tag{7.3}$$

For current flowing at low frequencies, A is equal to the entire area of the wire. However, when the skin effect comes into play, A is equal to the area within the skin depth:

$$A = \pi(r_o^2 - r^2)$$

for the cylinder pictured in Figure 7-5. Substituting in $r = r_o - d$, and solving for A in terms

of d,

$$A = \pi(r_o^2 - r_o^2 + 2r_o d - d^2),$$

which reduces to

$$A \approx 2\pi r_o d \tag{7.4}$$

at frequencies high enough that $r_o \gg d$. Combining Equation 7.2, Equation 7.3, and Equation 7.4,

$$R = \frac{l}{2\pi r_o}\sqrt{\frac{\mu\mu_o}{2\sigma}}\sqrt{\omega} \tag{7.5}$$

For the isolation transformer, l, μ, μ_o and σ are all constant, leaving ω as the only variable. Equation 7.5 can be reduced to

$$R = k\sqrt{\omega} \tag{7.6}$$

where

$$k = \frac{l}{2\pi r_o}\sqrt{\frac{\mu\mu_o}{2\sigma}}$$

Since the transformers being studied are designed to be driven at around 60 Hz, the gauge of the wire used in the transformer windings can be relatively large without introducing the skin effect at this frequency. However, frequencies higher than 60Hz will be subjected to the skin effect since manufacturers prefer to use a large gauge to reduce the resistance per unit length.

7.5 DESIRE Data

Using the technique described in Section 7.2, DESIRE calculated the impedances of the transformer, as seen in Table 7.3. The "calc" columns refer to the data calculated by DESIRE and the "meas" columns refer to the data collected using the HP Impedance Analyzer. The resistance that DESIRE calculated was then calibrated using the data recorded in Tables 7.1 and 7.2.

C(μF)	F(Hz)	L(Calc-mH)	L(Meas-mH)	R(Calc-Ω)	R(Calib-Ω)	R(Meas-Ω)
5.49	1930	1.25	1.31	4.1	3.6	4.17
6.47	1750	1.29	1.31	3.93	3.5	3.63
9.36	1460	1.27	1.31	3.09	2.64	2.97
16.1	1120	1.25	1.31	2.35	1.93	2.1
18.95	1020	1.29	1.31	2.33	1.92	1.9
21	980	1.26	1.31	2.15	1.71	1.85
23.9	900	1.3	1.31	2.19	1.74	1.67
26.92	872	1.23	1.31	2.07	1.64	1.63
30.35	794	1.32	1.31	2.2	1.67	1.51
33.8	763	1.29	1.31	1.99	1.54	1.45
40.93	693	1.29	1.31	1.81	1.43	1.34
44.35	664	1.3	1.31	1.8	1.43	1.31
47.78	642	1.29	1.31	1.76	1.39	1.28

Table 7.3: Nominal and Calibrated Data from DESIRE

Notice that the resistance calculated by DESIRE exhibits the skin effect described in Section 7.4.2. With large values of capacitance chosen, the transient frequency remains in the 600Hz range, and is therefore still affected by the skin effect.

7.6 Power Quality Simulation

Once the supply impedances of the isolation transformer are determined, the voltage waveforms at the secondary of the transformer can be estimated based on the current that the load on the transformer draws. To illustrate this concept, this chapter will detail how the current waveforms drawn by a Hewlett-Packard LaserJet III laser printer were recorded and analyzed so that the predicted voltage waveforms could be compared to the real waveforms at the secondary of the transformer. The waveforms recorded were triggered on times that the fuser of the laser printer turned on, corresponding to instances of large current flow through the isolation transformer. With this technique, steady state distortion and transient distortion can be analyzed using one set of data.

7.6.1 Steady State Distortion

The model that is used for the transformer impedances includes both resistive and inductive impedances. Therefore, the voltage distortion depends not only on the instantaneous current but also on the rate of change of the current:

$$V_o = V_s - RI - L\frac{dI}{dt}, \tag{7.7}$$

where V_s is the source voltage and V_o is the voltage across the load. The method chosen to determine the derivative the current was to decompose the current waveform into its frequency components using a Discrete-Time Fourier Transform. The voltage distortion due to each harmonic component can then be calculated independently. The total voltage distortion will be the sum of the distortions due to each component.

The printer draws a current waveform that has significant first, third, fifth and seventh harmonic components in steady state. The relative contribution of each of the components was found in Matlab by taking the DTFT of the recorded current waveform [See Appendix D.3 for script file "predi5"]. The approximate steady state current is

$$I_{ss} = 0.56\sin 2\pi \cdot 60t - 0.39\sin 2\pi \cdot 3 \cdot 60t + 0.34\sin 2\pi \cdot 5 \cdot 60t - 0.12\sin 2\pi \cdot 7 \cdot 60t. \tag{7.8}$$

Figure 7-7 shows the real and the estimated currents drawn by the laser printer. The current drawn for $0 < t < 0.05s$ is considered the steady state current. Each harmonic component of the current is modeled as a current source in parallel with the other current sources. Each modeled source results in voltage distortion, governed by Equation 7.7. For predicting the distorted waveforms, the resistance and inductance at the frequency closest to 60Hz were chosen, $R = 1.40\Omega$ and $L = 1.3mH$. Using the superposition principle, the voltage across the

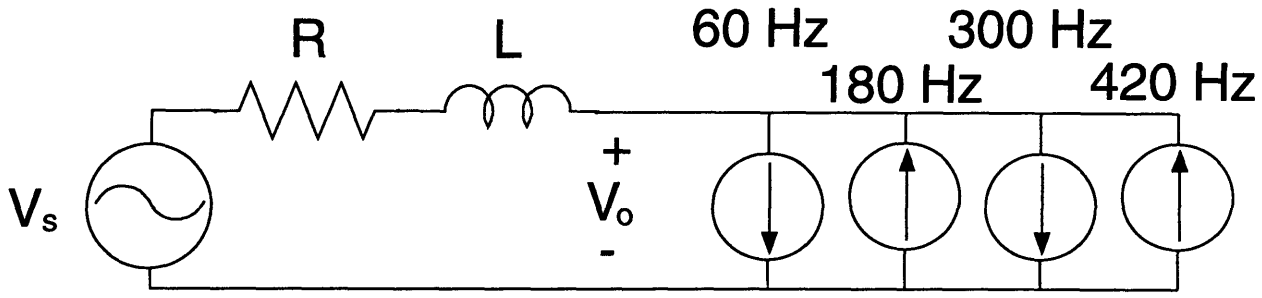


Figure 7-6: Model of Source and Current, using Harmonic Components

load will be equal to the voltages resulting from each source in the model:

$$\begin{aligned} V_o &= 178\sin 2\pi\cdot 60t - 0.78\sin 2\pi\cdot 60t - 0.27\cos 2\pi\cdot 60t \\ &\quad +0.55\sin 2\pi\cdot 3\cdot 60t + 0.58\cos 2\pi\cdot 3\cdot 60t \\ &\quad -0.48\sin 2\pi\cdot 5\cdot 60t - 0.84\cos 2\pi\cdot 5\cdot 60t \\ &\quad +0.16\sin 2\pi\cdot 7\cdot 60t - 0.40\cos 2\pi\cdot 7\cdot 60t \end{aligned} \tag{7.9}$$

7.6.2 Transient Distortion

An interesting distortion occurs when the laser printer periodically activates its fuser. During this phase of printer operation, the load not only draws steady state current, but also a large exponentially decaying sinusoidal current. The voltage distortion during this phase is significant because of the large amount of current that is drawn.

The transient current appears to be a 60 Hz sinusoid that decays away exponentially. Adjusting the amplitude and rate of decay in Matlab to simulate the approach the values of

the real current, we find that the transient current is approximately

$$I_t = 34 \sin 2\pi \cdot 60\tau e^{-10.6\tau} \tag{7.10}$$

for $\tau = t - 0.05$ and $0.05s < t < 0.1s$ [See Figure 7-7]. The resulting voltage distortion is then

$$\begin{aligned} V_{dt} &= -47.0 \sin 2\pi \cdot 60\tau \cdot e^{-10.6\tau} \\ &\quad -16.7 \cos 2\pi \cdot 60\tau \cdot e^{-10.6\tau}. \end{aligned} \tag{7.11}$$

A matlab script file, "pred5.m" has been written to predict the voltage waveforms for this case based on the estimated current waveform and the predicted resistance and inductance of the source [See Appendix D.2]. Using superpositon, "pred5.m" adds I_t to I_{ss} and V_{dt} to V_o to simulate the transient current and voltage distortion waveforms. The results of "pred5" using $R = 1.4\Omega$ and $L = 1.3mH$ are shown in Figure 7-8. In the figure, the measured voltage waveform is the solid plot and the predicted waveform is the dashed line. The predicted waveform is offset in time for ease of analysis in this plot. The graph qualitatively display the voltage distortion. Since the R is calculated based on a transient that is affected by skin depth because it oscillates at over 600Hz, the voltage waveforms as predicted by DESIRE have exaggerated the voltage distortion. Using $R = 0.8\Omega$ and $L = 1.3mH$ in "pred5.m" to predict distortion, the measured values of the isolation transformer, we obtain a graph which predict voltage distortion almost exactly, again offset in time. [See Figure 7-9.]

7.7 Analysis of Results

Table 7.4 shows the supply impedances predicted by DESIRE and those measured using the impedance analyzer for a wide range of frequencies. The information reported by DESIRE accurately reflects the supply impedances for transient oscillation frequencies as low as 600 Hz in the isolation transformer. At this point, the value of resistance reported by DESIRE levels off at around $R \approx 1.4\Omega$, while the measured resistance decreases steadily until it reaches

C(μF)	F(Hz)	L(Calc-mH)	L(Meas-mH)	R(Calib-Ω)	R(Meas-Ω)
5.49	1930	1.25	1.31	3.6	4.17
6.47	1750	1.29	1.31	3.5	3.63
9.36	1460	1.27	1.31	2.64	2.97
16.1	1120	1.25	1.31	1.93	2.1
18.95	1020	1.29	1.31	1.92	1.9
21	980	1.26	1.31	1.71	1.85
23.9	900	1.3	1.31	1.74	1.67
26.92	872	1.23	1.31	1.64	1.63
30.35	794	1.32	1.31	1.67	1.51
33.8	763	1.29	1.31	1.54	1.45
40.93	693	1.29	1.31	1.43	1.34
44.35	664	1.3	1.31	1.43	1.31
47.78	642	1.29	1.31	1.39	1.28
60.79	560	1.34	1.31	1.44	1.20
70.2	510	1.39	1.31	1.47	1.11
96.61	430	1.41	1.31	1.35	0.98

Table 7.4: Measured and Estimated Supply Impedances

approximately 0.8Ω. We attribute this phenomena to the breakdown of assumptions made in the model of the system. As 600 Hz is a factor of 10 greater than 60 Hz, the assumption that the transient frequency is much greater than the driven frequency is not entirely valid. At these relatively low frequencies, the driven response needs to be considered when extracting data from the transient. The method used for predicting voltage waveforms was shown to be a success. By using the measured values of for the isolation transformer, the distorted voltage waveform was predicted with considerable accuracy. This suggest that if improvements are made in the DESIRE system to correctly evaluate the supply impedances, local voltage waveforms could be precisely predicted.

7.8 Variable Torque AC Drive

The transient current drawn by the laser printer significantly distorted the local voltage waveform. This distortion may be tolerable for many applications, since the distortion occurs only

a small fraction of the time that the printer is active. However, distortion of this nature that occurs in steady state would adversely affect the longevity of any load connected to the local voltage waveform. One load was examined, an Allen-Bradley Variable Torque AC Drive (VTACD), that could lead to significant local voltage distortion.

The VTACD examined is used for heating and ventilation on the MIT campus. Access to the drive was limited, but some measurements were taken to record current waveforms of one of its three phases. The current drawn by the VTACD, [see Figure 7-10] was decomposed into its harmonic components using a spectrum analyzer[2]. The data from the spectrum analyzer [See Figure 7-11] indicates significant first harmonic real power consumption (P1), and first (Q1), fifth (Q5), and seventh (Q7) harmonic reactive power consumption. The harmonic components of power consumption directly correspond to harmonic components in the current waveform. Therefore, the harmonic content taken from the spectrum analyzer, can be scaled appropriately and used in conjunction with measurements taken by DESIRE to predict local voltage distortion. Unfortunately, we did not have access to the power source for the VTACD, so measuremnts could not be taken with DESIRE. However, the possible distortion can be illustrated by substituting the isolation transformer for the VTACD's voltage source. Using $R = 0.8\Omega$ and $L = 1.3mH$ for the source impedance, the local voltage waveform for the VTACD was estimated in Matlab using the script file "predigen.m", designed to be used with data from the spectrum analyzer [See Figure 7-12].

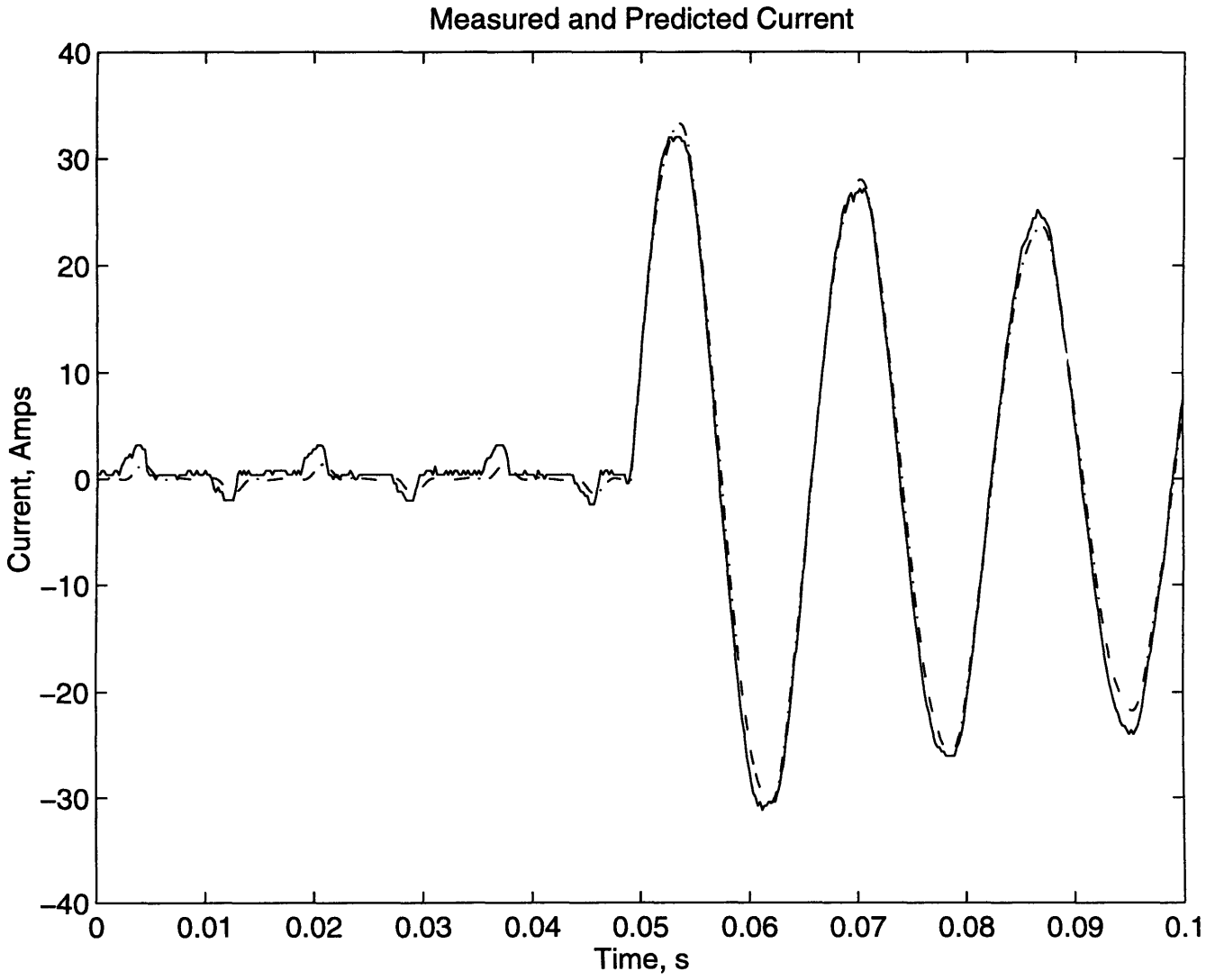


Figure 7-7: Total Current Waveform and Estimate

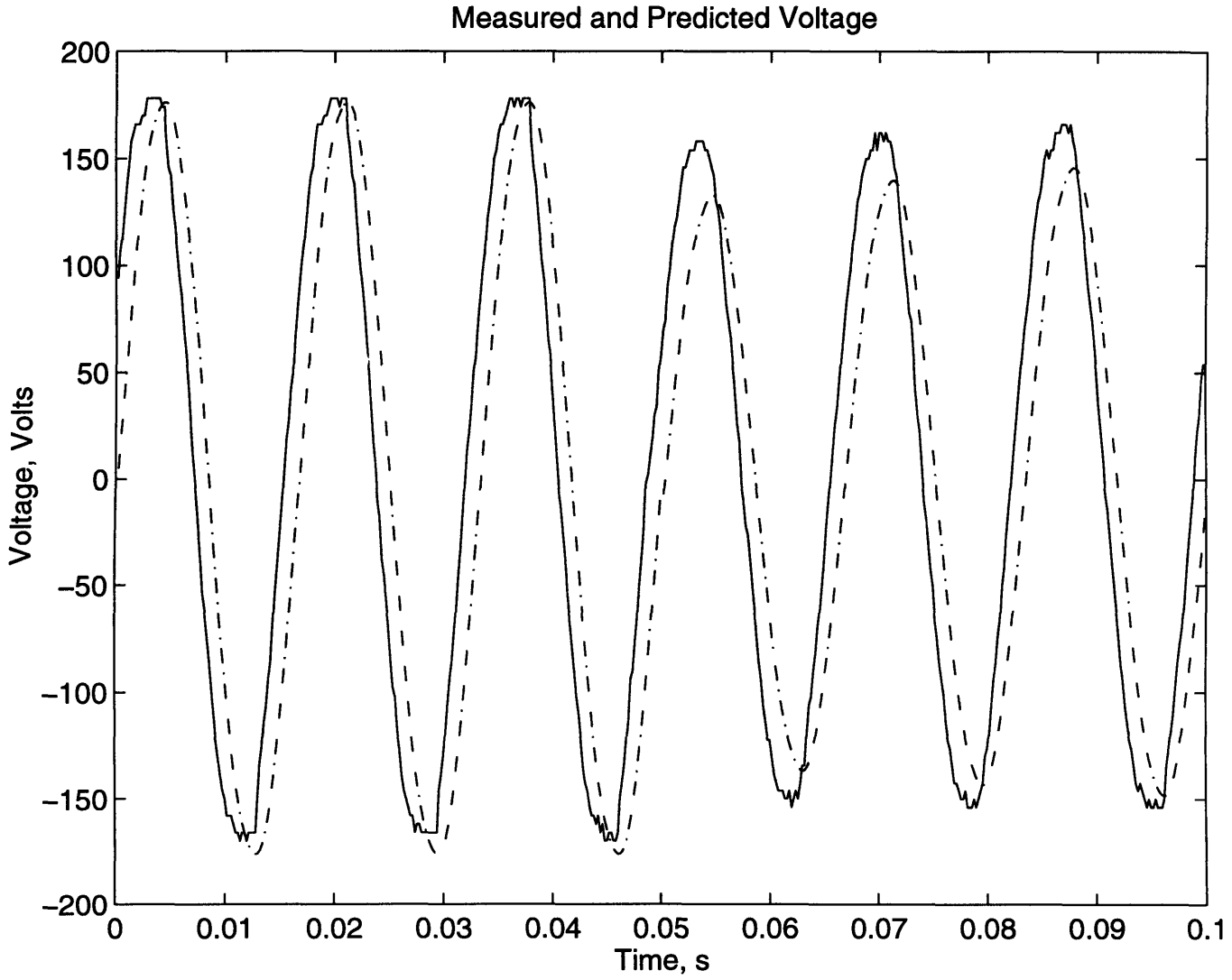


Figure 7-8: Transient voltage distortion and estimate using DESIRE's R

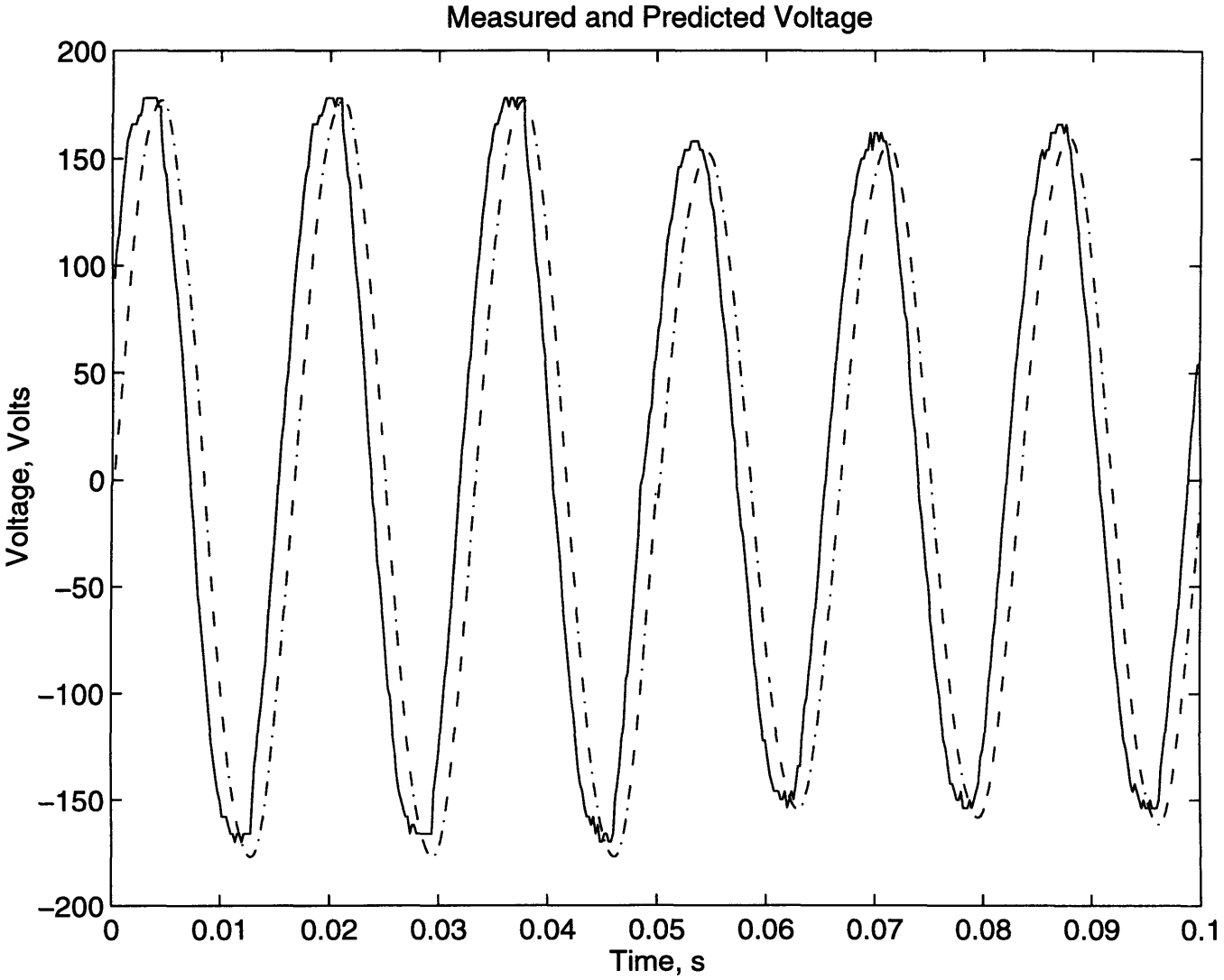


Figure 7-9: Transient voltage distortion and estimate using measured R

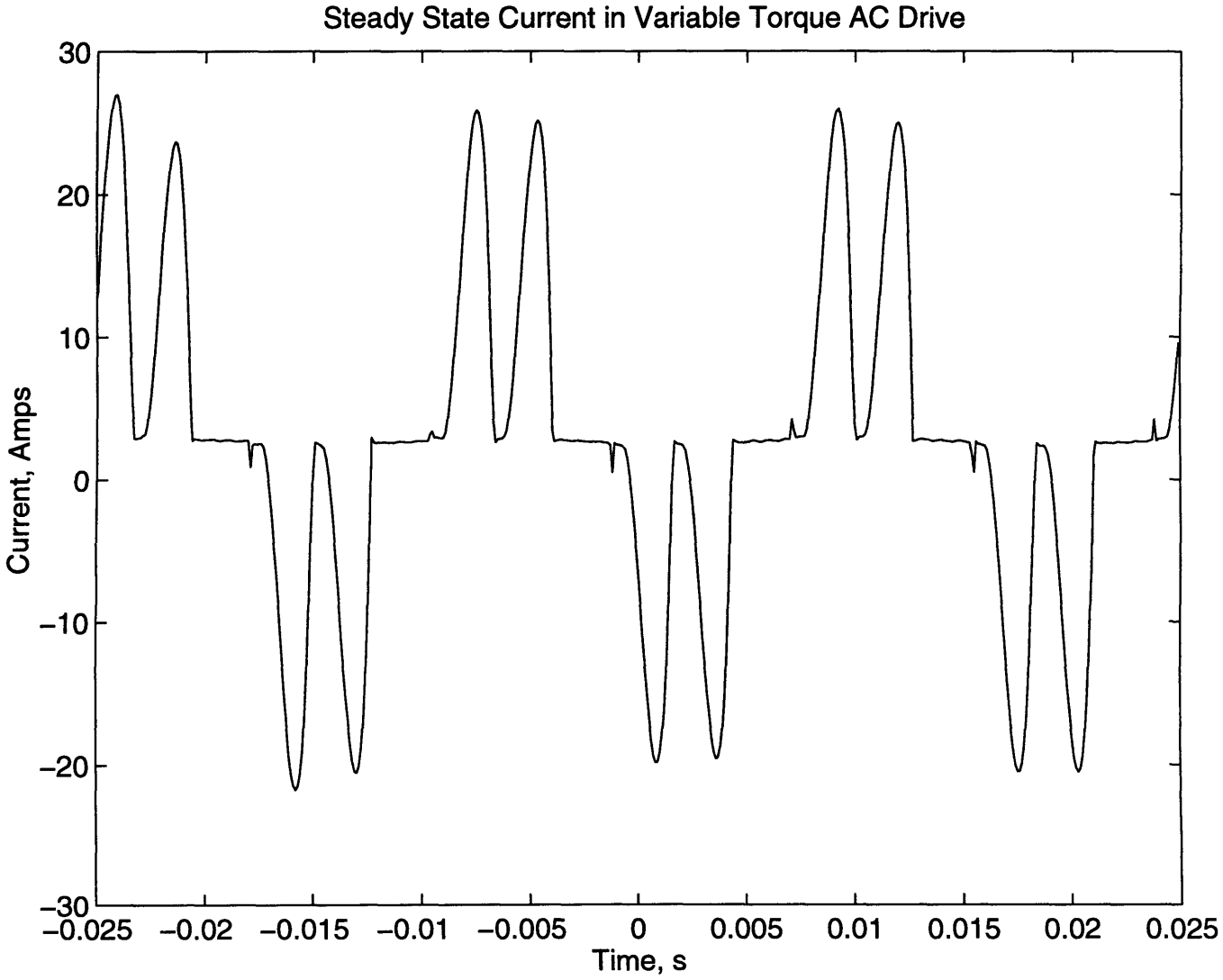


Figure 7-10: Steady State VTACD Current

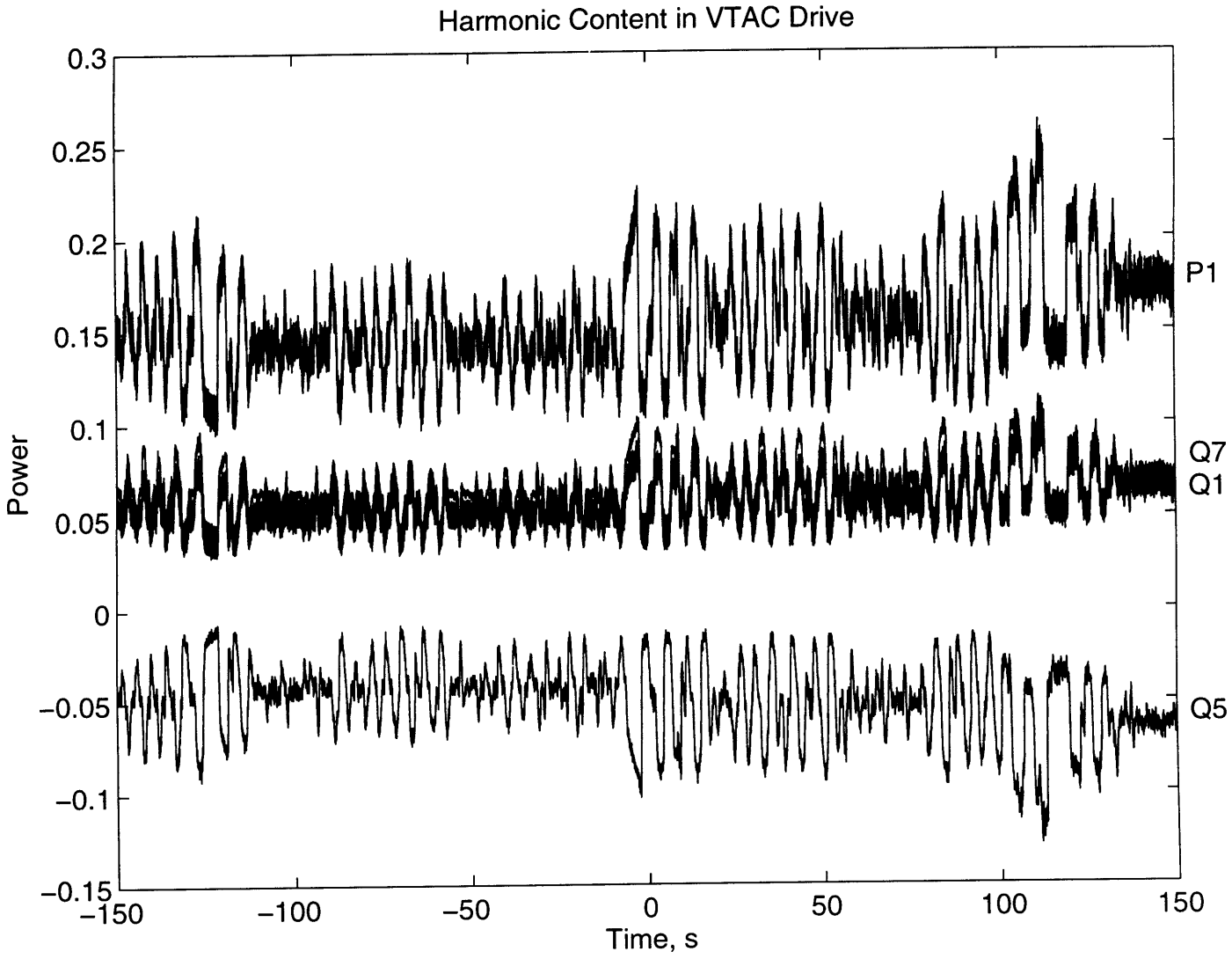


Figure 7-11: Harmonic Components of VTACD Current

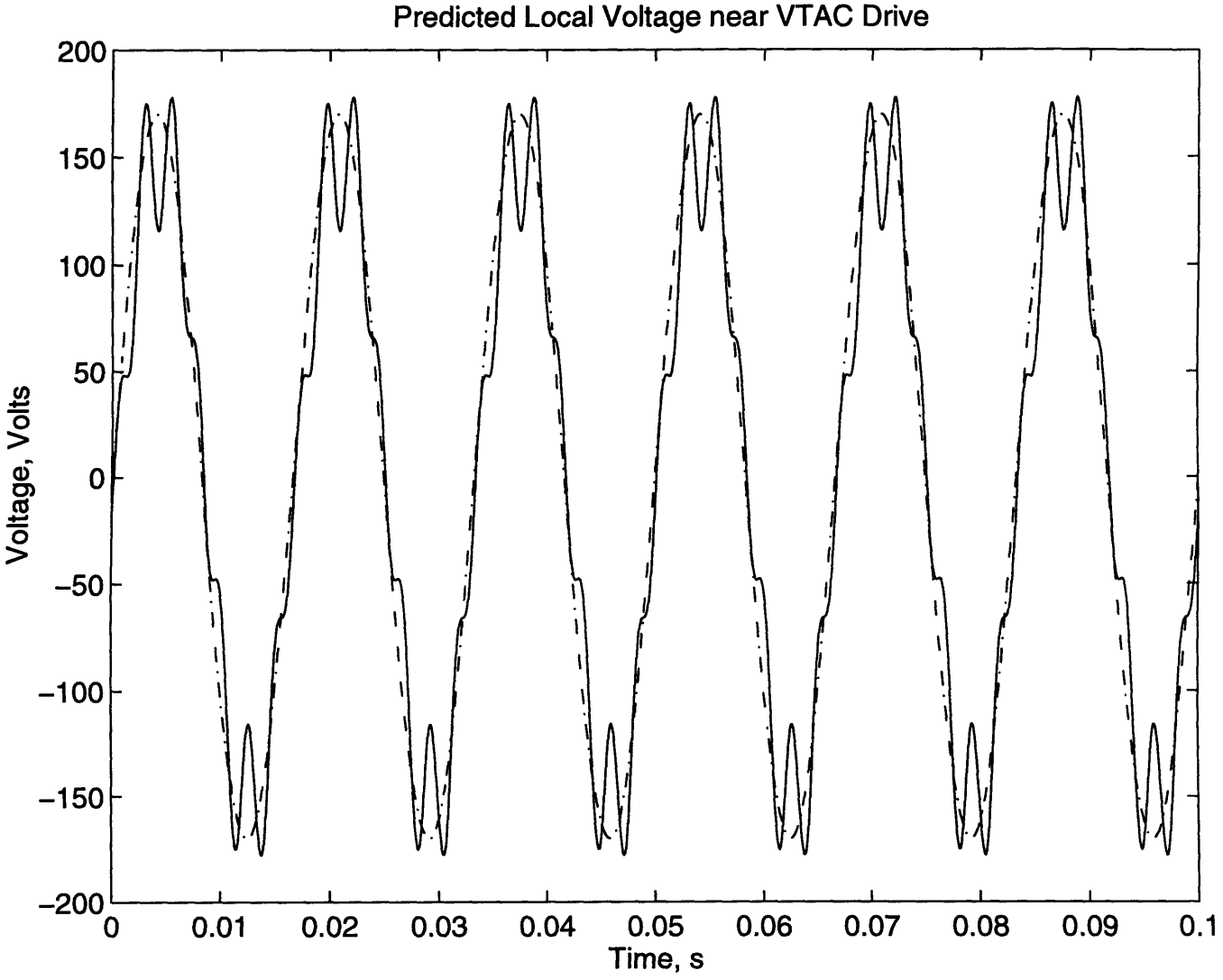


Figure 7-12: Possible Local Voltage Waveform for VTACD

Chapter 8

Conclusions

Summary of Results

The premise of this thesis was to increase the capabilities of the Nonintrusive Load Monitor. Specifically, a precise timing switch was developed in Chapter 2 to study how the point a load is activated in a line cycle affects the transient power that it draws. When used in conjunction with the automated v-section collection program, described in Chapter 3, reliable v-sections can be created for the NILM. These v-sections are more precise than v-sections that are created by hand, and therefore can have lower tolerance levels when identifying v-sections. This improvement in v-section idententification allows the NILM to distinguish between similar v-sections since the error tolerance levels are lower, reducing the number of false recognitions and the number of missed recognitions.

Next, we described a model for power transmission in Chapter 4. The DESIRE system was developed in Chapters 5, 6, and 7 to attempt to measure the parasitic impedances in this transmission model. When a capacitor switched across a line-to-neutral connection, a transient current waveform is observed. Because we have a reasonable model for the transmission of power, the parasitic impedances in the transmission path can be determined based on the transient current waveform. The DESIRE system was tested by comparing predicted values to values measured using an impedance analyzer. When all assumptions were valid, the

measurements taken by DESIRE proved accurate.

Using the impedances calculated by DESIRE, we can approximate the local voltage waveform based on the current waveforms drawn in the local area. The resulting voltage waveforms exhibited the qualities of the measured voltage waveforms, but limitations in the DESIRE system resulted in exaggerated voltage distortion. We also demonstrated that this technique for predicting voltage waveforms is accurate when the correct supply impedances are used.

Directions for Future Work

Further research could be done to integrate measurements taken from DESIRE into the NILM to constanly predict local voltage waveforms. This could be done in either of two ways. The spectrum analyzer, used to collect data in Section 7.8, could constantly update the current waveforms that the NILM observes. These currents would then need to be continually processed to obtain predicted voltage waveforms. A second method could utilize the ability of the NILM to identify loads that are active. Each of the loads to be identified by the NILM could have its current waveforms recorded. The data from DESIRE could then be used to identify the amount of voltage distortion caused by each load. When the NILM recognizes the activation of a load, the voltage distortion cause by the load in question could be added to the previously predicted voltage waveform to model the total voltage distortion.

DESIRE could also be used in the design of power transmission in buildings. By using measured the impedances in transmission paths from DESIRE with the Radial Panel Installation Designer (RAPID) [19], a power network can be modeled and analyzed before it is constructed. The designer will be able to identify areas which could have potential poor power quality based on the location and types of the loads in the network.

Bibliography

[1] S.B. Leeb, S.R. Shaw, and J.L. Kirtley Jr. Transient event detection in spectral estimates for nonintrusive load monitoring. *IEEE PESS Winter Meeting*, January 1994. Paper 95 WM 042-2-PWRD New York, NY.

[2] S.B. Leeb and S.R. Shaw. Harmonic estimates for transient event detection. *Universities Power Engineering Conference (UPEC)*, 1994. Galway, Ireland.

[3] S.B. Leeb and Jr. J.L. Kirtley. A transient event detector for nonintrusive load monitoring. U.S. Patent Application. U.S. Patent No. 5483153.

[4] Umair A. Khan. A multiprocessing platform for transient event detection. Master's thesis, Massachusetts Institute of Technology, May 1995.

[5] Linear Technology. *1990 Linear Data Book*, 1990.

[6] P. Horowitz and W. Hill. *The Art of Electronics.* Cambridge University Press, Cambridge, 1980.

[7] Intel. *Intel Peripheral Components*, 1990.

[8] Mitchell Waite and Stephen Prata. *C: Step-by-Step.* Carmel Indiana, 1989.

[9] R.E. Steven. *Electrical Machines and Power Electronics.* Van Nostrand Reinhold (UK) Co., Berkshire, 1983.

[10] I.S. Grant and W.R. Phillips. *Electromagnetism.* John Wiley & Sons Ltd., London, 1975.

[11] J.G. M.F. Schlecht Kassakian and G.C. Verghese. *Principles of Power Electronics.* Addison-Wesley Publishing Company, Reading, MA, 1991.

[12] W.C. Beattie and S.R. Matthews. Impedance measurement on distribution networks. *UPEC*, 1994.

[13] Maxim. *Maxim 1995 New Releases Data Book*, 1995.

[14] Intel. *Intel Memory Components Handbook*, 1986.

[15] Texas Instruments. *Texas Instruments High Speed CMOS Data Book*, 1990.

[16] Hewlett-Packard. *Operation and Service Manual: 4192A LF Impedance Analyzer*, 1984.

[17] The Math Works Inc. *Matlab for 80386-based MS-DOS Personal Computers: User's Guide*, 1990.

[18] T.J. Respress. Desire: Final report. Technical report, Massachusetts Institute of Technology, 1996.

[19] Steven B. Leeb. *A Conjoint Pattern Recognition Approach to Nonintrusive Load Monitoring.* PhD thesis, Massachusetts Institute of Technology, January 1993.

[20] S.B. Leeb and J.L. Kirtley. A multiscale transient event detector for nonintrusive load monitoring. *Proceedings of the IEEE*, November 1993.

Appendix A

Schematic Diagrams of DESIRE

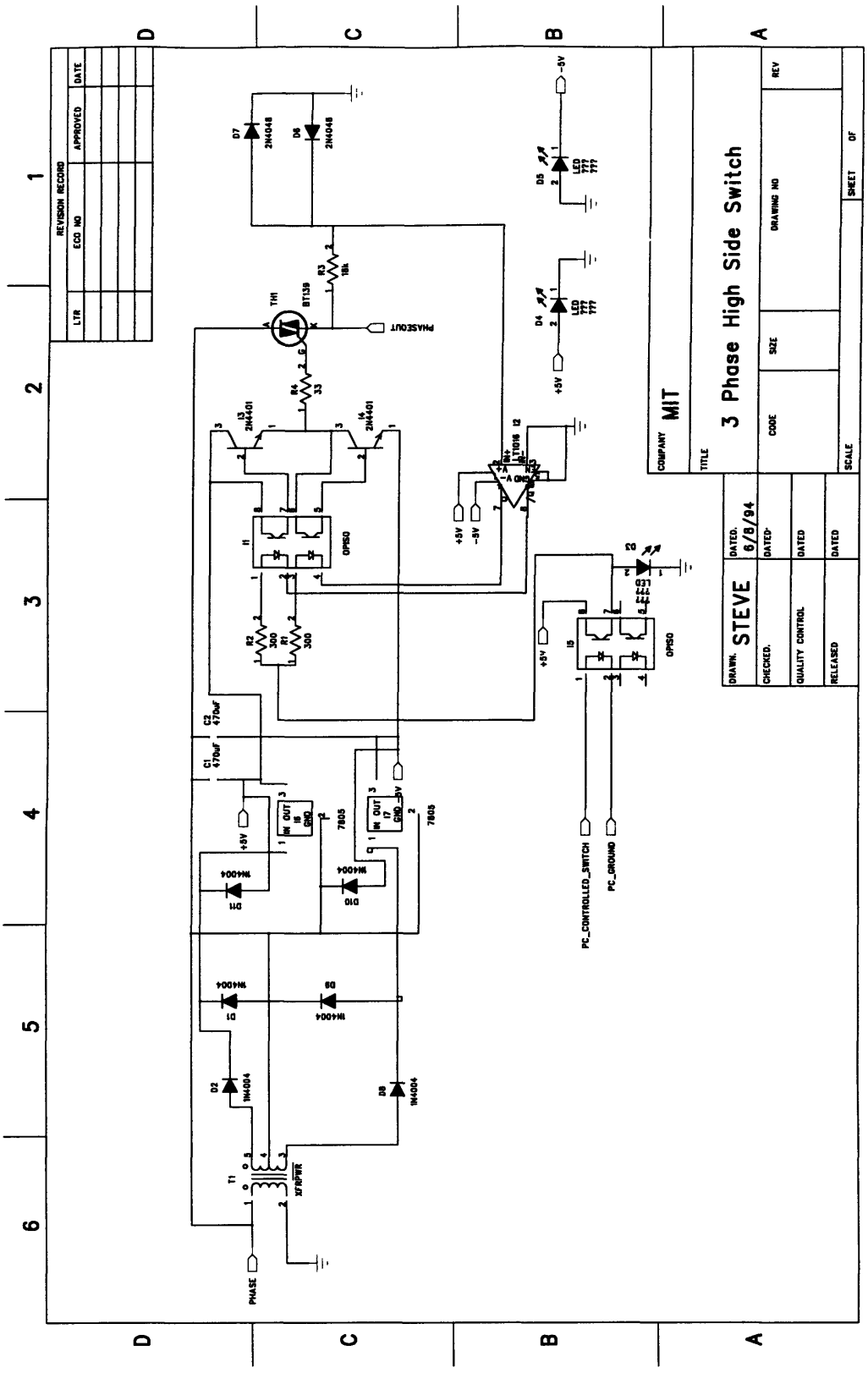


Figure A-1: Switch Board Schematic

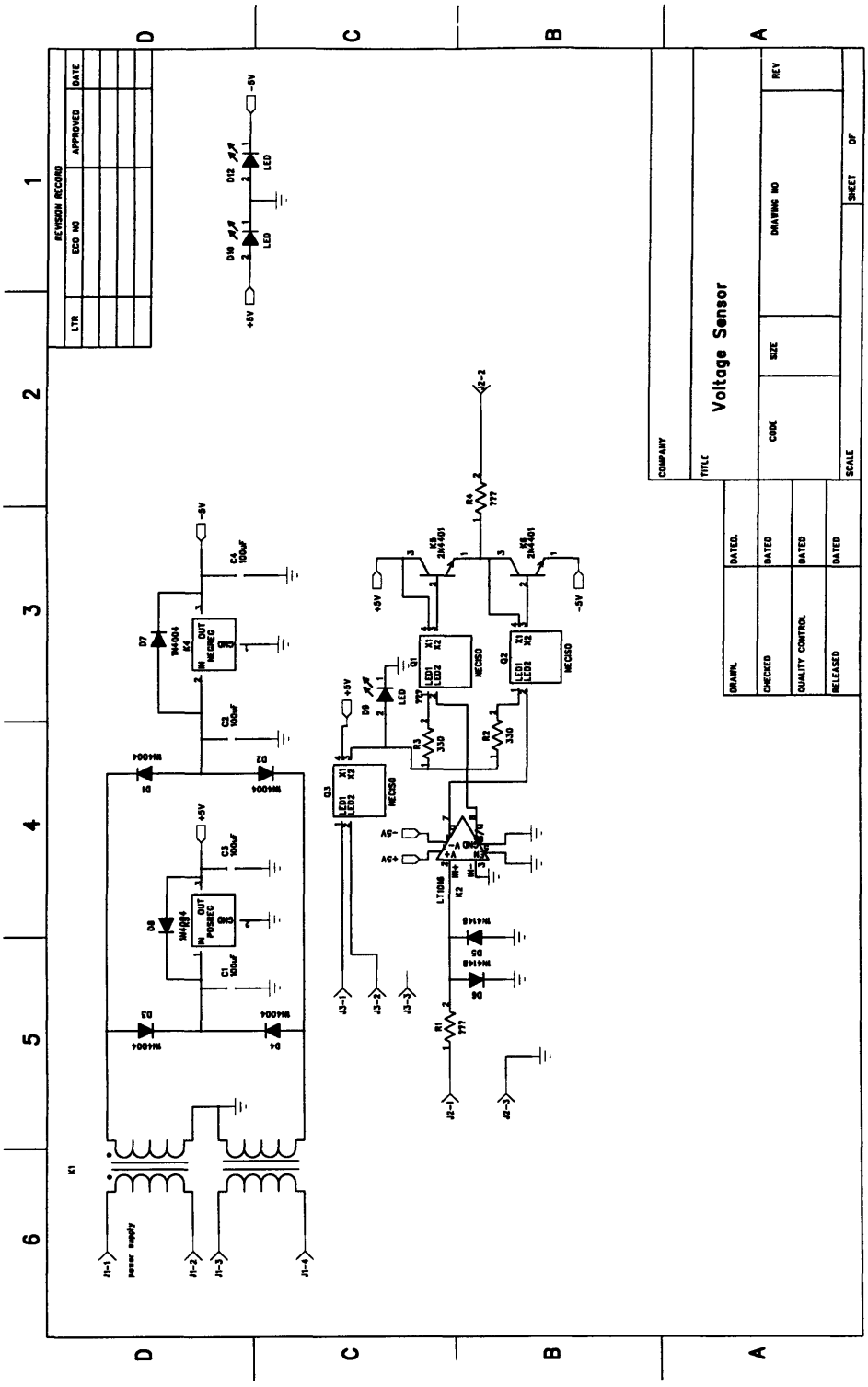


Figure A-2: Voltage Sensor Board Schematic

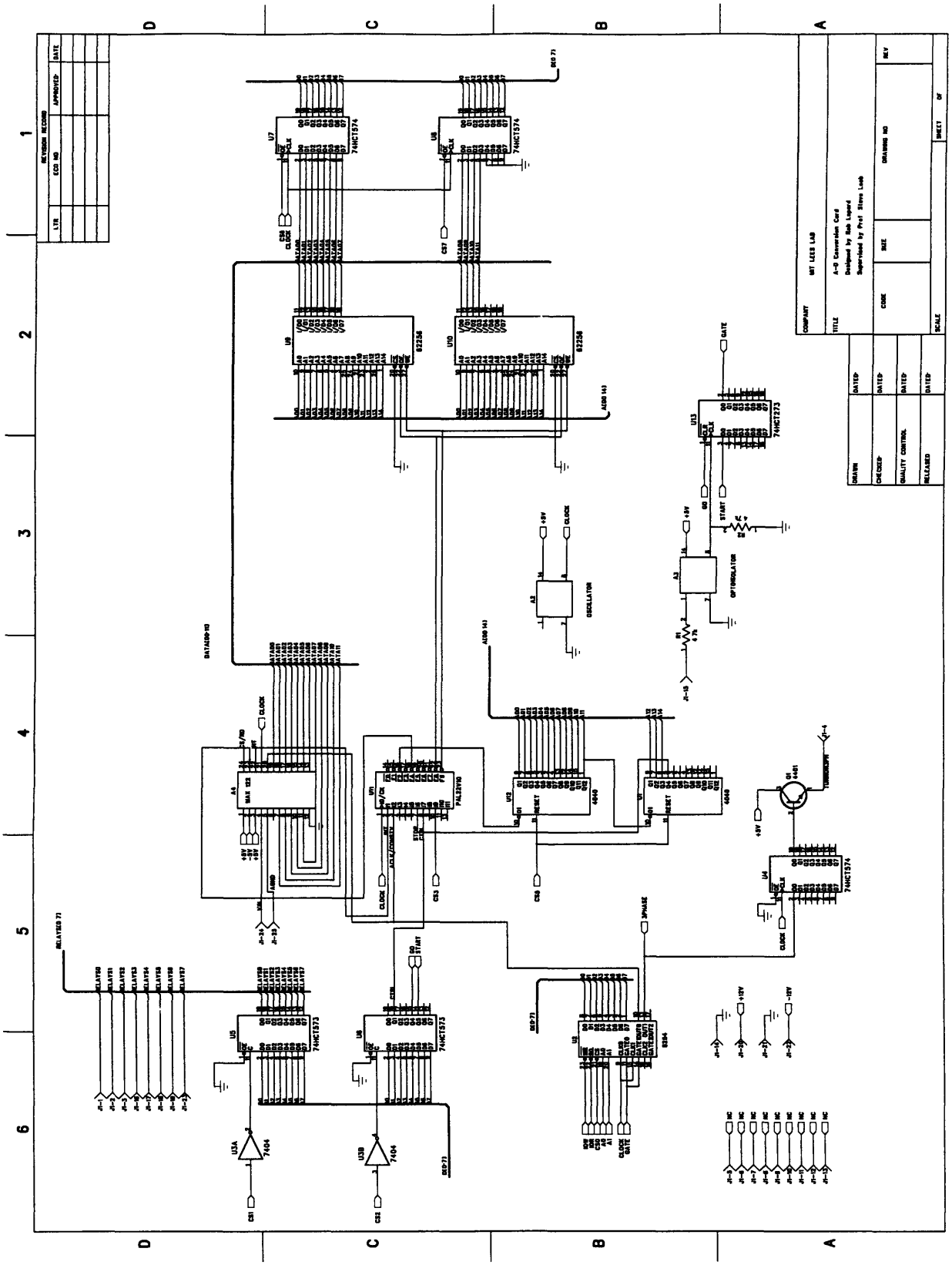


Figure A-3: PC Card Schematic

Appendix B

Code for Load Test Stand

B.1 Interface

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

/*
these routines interface the C code to the assembly code.

int far *get_pointer(void) is an assembly routine that returns a
pointer to 16 words located in the assembly
module. The first 4 words are the number of
60Hz line cycles that each 8254 should wait
before initiating the fine division delay.
The minimum number of 60Hz clock cycles to
delay is 1. The next 12 words are the number
of 61.4kHz PLL generated clock cycles that
will be counted before turning on the
alternistor.

The word AFTER the timing data is a running
counter of 60Hz line cycles, which can be
used to determine when it is safe to turn
off NMI and exit.

void install_isr(void);     is an assembly routine that enables
         NMI interrupts (x286 and up only. will not
run on PC's) and installs the interrupt
```

```
handler. This routine should be called
AFTER initializing the assembly module's
data area (with get_pointer)

void remove_isr(void); is an assembly routine that cleans up the
NMI interrupts. It replace the old NMI
handler and resets the motherboard's NMI
circuitry.
*/

typedef unsigned int word;

word far *get_pointer(void);
void install_isr(void);
void remove_isr(void);

void turn_off(void);

main()
{
word p[16] = {  10, 10, 10, 10,
255, 50000, 50000,
256, 256, 256,
256, 256, 256,
256, 256, 256};
volatile word far *ptr,*optr;
int i;
int start, last;

outp(0x315, 0);
turn_off();

ptr = get_pointer(); /* Get pointer to assembly data area */

optr = ptr;

for(i = 0; i < 16; i++)         /* Write our data...  */
*ptr++ = p[i];

/* NOTE: ptr now points to our running counter... */

install_isr();

getch();
```

```
printf("Hit a key to start...");
while(kbhit()) getch();
getch();

outp(0x310, 1);
outp(0x315, 1);

do {
 cprintf("\r%u, %u, %u, %u", optr[0], optr[1], optr[2], optr[3]);
} while(!kbhit());

while(kbhit()) getch();
getch();

turn_off();

outp(0x310, 0);
outp(0x315, 0);

printf("\n");

remove_isr();
}

void turn_off(void)
{
      int i;


for(i = 0; i < 4; i++) {
outp(0x303 + 4*i, 0x30);
outp(0x300 + 4*i, 0);
outp(0x303 + 4*i, 0x30 + 0x40);
outp(0x301 + 4*i, 0);
outp(0x303 + 4*i, 0x30 + 0x80);
outp(0x302 + 4*i, 0);
}

}
```

B.2 Assembly Code

```
;
; CONTAINS 286 ONLY ASSEMBLER INSTRUCTIONS!!!
```

```
;
; BOARDX.ASM - Interface module for 3 phase switch
;
; Contents:
; DOSgetivec - routine to return an interrupt vector.
; DOSsetivec - routine to set an interrupt vector.
; isr - interrupt service routine
; _install_isr - routine that installs the isr
; _remove_isr - routine that removes the isr
; _out_port - same as outportb
; _in_port - same as inportb
; _get_pointer
;
P286
DOSSEG
.MODEL LARGE
.DATA
dz dw 16 dup(?) ; 16 words of data.
cnt dw 0 ; running counter.
old dd 0 ; Old interrupt service vector
.CODE

BASE = 0300h ; Base address

PUBLIC _install_isr, _get_pointer
PUBLIC _out_port, _in_port, _remove_isr

;
; return a pointer to the data.
;
_get_pointer proc far
mov dx, SEG  dz
mov ax, OFFSET dz
retf
_get_pointer endp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; ISR
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
isr proc far
push ax ; Save the registers.
push cx
push dx
push di
```

```
push es
push ds
push bp
mov bp,DGROUP ; Load the data group.
mov ds,bp

mov dx,0310h ; Point to on-card NMI circuitry
xor al,al ; put a zero in AL
out dx,al ; clear NMI
inc al ; put a one in AL
out dx,al ; reset NMI


inc WORD PTR cnt ; running counter.

t1: cmp WORD PTR [dz+0], 0 ; Compare value to ZERO
je t2 ; Already have ZERO, go to next counter
dec WORD PTR [dz+0] ; Decrement 1st counter.
jnz t2 ; Not zero yet? try next counter

mov dx, BASE+0+3 ; point to our 8254
mov al, 30h + 0h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 3 ; point to counter0
mov ax, [dz+8+0] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+0+3 ; point to our 8254
mov al, 30h + 40h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 2 ; point to counter1
mov ax, [dz+8+2] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+0+3 ; point to our 8254
mov al, 30h + 80h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 1 ; point to counter2
mov ax, [dz+8+4] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out
```

```
t2: cmp WORD PTR [dz+2], 0 ; Compare value to ZERO
je t3 ; Already have ZERO, go to next counter
dec WORD PTR [dz+2] ; Decrement 2nd counter.
jnz t3 ; Not zero yet? try next counter

mov dx, BASE+4+3 ; point to our 8254
mov al, 30h + 0h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 3 ; point to counter0
mov ax, [dz+14+0] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+4+3 ; point to our 8254
mov al, 30h + 40h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 2 ; point to counter1
mov ax, [dz+14+2] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+4+3 ; point to our 8254
mov al, 30h + 80h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 1 ; point to counter2
mov ax, [dz+14+4] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

t3: cmp WORD PTR [dz+4], 0 ; Compare value to ZERO
je t4 ; Already have ZERO, go to next counter
dec WORD PTR [dz+4] ; Decrement 3nd counter.
jnz t4 ; Not zero yet? try next counter

mov dx, BASE+8+3 ; point to our 8254
mov al, 30h + 0h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 3 ; point to counter0
mov ax, [dz+20+0] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out
```

```
mov dx, BASE+8+3 ; point to our 8254
mov al, 30h + 40h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 2 ; point to counter1
mov ax, [dz+20+2] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+8+3 ; point to our 8254
mov al, 30h + 80h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 1 ; point to counter2
mov ax, [dz+20+4] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

t4: cmp WORD PTR [dz+6], 0 ; Compare value to ZERO
je t5 ; Already have ZERO, go to next counter
dec WORD PTR [dz+6] ; Decrement 3nd counter.
jnz t5 ; Not zero yet? try next counter

mov dx, BASE+12+3 ; point to our 8254
mov al, 30h + 0h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 3 ; point to counter0
mov ax, [dz+26+0] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+12+3 ; point to our 8254
mov al, 30h + 40h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 2 ; point to counter1
mov ax, [dz+26+2] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

mov dx, BASE+12+3 ; point to our 8254
mov al, 30h + 80h ; "interrupt on terminal count"
out dx, al ; send it out
sub dx, 1 ; point to counter2
```

```
mov ax, [dz+26+4] ; get values
out dx,al ; send it out
mov al,ah
out dx,al ; send it out

t5:

;;
;; What follow are the special codes for dealing with the NMI hardware
;; the first code fragment is needed for the ON-CARD nmi hardware. The
;; second segment is needed for the ON-MOTHERBOARD nmi hardware.
;;
in al,61h ; Reset the motherboard.
or al, 00001000b
out 61h,al
and al, 11110111b
out 61h,al

pop bp ; Restore the registers.
pop ds
pop es
pop di
pop dx
pop cx
pop ax
iret
isr endp



;
; void out_port(int portnumber, unsigned char value);
;
;
_out_port proc far
ARG port:WORD, val:BYTE
push bp
mov bp,sp

mov dx, port
mov al, val
out dx,al

pop bp
retf
_out_port endp
```

```
;
; int IN_PORT(int portnumber)
;
_in_port proc far
ARG port:WORD
push bp
mov bp,sp

mov dx, port
in al,dx

pop bp
retf
_in_port endp


;
; Get a pointer to an interrupt vector.
;
; AL = Vector Number
;
; Returns pointer in es:bx
;
DOSgetivec proc
mov ah,35h
int 21h
ret
DOSgetivec endp

;
; Set an interrupt vector.
;
; AL  = Vector Number
;
; ES:BX = Pointer to interrupt vector routine
;
DOSsetivec proc
push ds
push bx

xchg bx,dx ; X'fer ES:BX to DS:DX
mov bx,es
mov ds,bx

mov ah,25h
```

```
int 21h
pop bx
pop ds
ret
DOSsetivec endp

;
; void install_isr(void);
;
; Procedure to install the ISR.
;
_install_isr proc far
mov al,2
call DOSgetivec ; Get and save the old interrupt.
mov word ptr [old+0], bx
mov word ptr [old+2], es

mov al,2 ; Set the interrupt vector (#2)
mov bx,seg isr ; Get the segment
mov es,bx
mov bx,offset isr ; Get the offset.
call DOSsetivec ; Set the vector.

in al,70h ; Enable NMI interrupts.
and al,7fh
out 70h,al

in al,61h
or al,00001000b
out 61h,al
and al,11110111b
out 61h,al

retf
_install_isr endp

;
; void remove_isr(void);
;
; Procedure to remove the interrupt service routine.
;
_remove_isr proc far
mov es, word ptr [old+2] ; Just set the ptr to the old address.
mov bx, word ptr [old+0]
mov al, 2
call DOSsetivec
```

```
retf
_remove_isr endp

END
```

Appendix C

DESIRE

C.1 Using DESIRE

Determination of Supply Inductance and Resistance (DESIRE) is a system which is designed to determine inductance and resistance looking into an electrical outlet for the purposes of predicting power quality. To accomplish this, a capacitive load is placed across line voltage at precisely the peak in the voltage waveform, and the current waveforms that the system draws are recorded. Based on our knowledge of such a system, a model has been created. The simulated current waveforms are then best fit to the real current waveforms by adjusting the model's R and L. These values are then returned to the user. This section describes the code of the user interface system.

Setup In Figure C-1, a diagram of DESIRE is shown. To properly setup the DESIRE system, several connections need to be made:

- Connect PC and monitor to powerstrip
- Connect monitor and keyboard to PC
- Connect Capacitor Box to PC (DB25 connection)
- Connect Load Test Stand to Capacitor Box (DB25 Connection)
- Connect Capacitor Box to electrical outlet on the front of the Load Test Stand

- Connect Powerstrip to Power Source
- Connect Load Test Stand to Source to be measured

In the current setup of DESIRE, the Gain Adjustment Switch [See Figure C-1] should be turned completely counter clockwise. This sets the gain resistor across the current sensor to 47Ω. The gain resistor can be increased to use the full range of the ADC by turning the Gain Adjustment Switch clockwise. The other settings of the gain resistor, moving clockwise from 47Ω are 75Ω, 100Ω, and 150Ω. When the gain resistor is changed, the variable *res* in "rob471" needs to be changed accordingly. If this variable is not changed, the magnitude of the current waveforms will be off, but the values that DESIRE calculates should not be affected.

Required Files

```
The files used to compile DESIRE include:
        mcc4.bat
        utils2.c
        mouse.lib
        llibce.lib
        globcon.h
        globfunc.h
        globvar.h
        button.h
        plot.h
        loadsave.h

The external files that the DESIRE System uses when running the code are:
        tmsrb.fon
        default.cap
        rob471.exe
        robmat.bat
        matlab
        transit.exe
        matorig.bat

These are the m-files used by Matlab:
        matlab.m
        matrob.m
        matold.m
        robir.m
```

peak.m
decay4.m
dec5.m

Description of Files

mcc4.bat	This is a batch file used to compile the DESIRE code and link it to utils2.c and mouse.lib. A typical command line would be''mcc4 filename" where filename.c is the file to be compiled.
utils2.c mouse.lib llibce.lib	These files are used to implement the mouse functions used in DESIRE. In utils2.c, there are several matrix programs that are not utilized in DESIRE.
globcon.h	This header file contains the global constants used by DESIRE. Specifically, the colornames are defined for ease of use and the origins of the plots are defined.
globvar.h	This header file defines varibles that DESIRE uses when it calls functions external functions.
globfunc.h	This file contains two of the functions that are called from within various parts of DESIRE.
button.h	Curiously enough, this header file defines the structure''pbutton", which is then used to create and hold information about each of the buttons that DESIRE uses (ie. capacitor numbers and values). The''lightup" and other similar routines are also included in this file.
plot.h	This header file contains the functions that are used to create the plot area for DESIRE. Some of the routines included in this file are init_plot (to plot blank graphs on the screen), plotme (a function called by DESIRE to begin the plot sequence), plotit2 (which prints two sets of data on the same axes) and other functions that are called by the above programs.
loadsave.h	Loadsave creates the routines used by the load and save buttons. These functions create a batch file that copies current data files to saved data files

and vice versa.

tmsrb.fon	This file contains information about the font that DESIRE uses to print information to the screen. Without this file, DESIRE will compile and run, but no text will be printed on the screen.
default.cap	Default.cap contains the capacitor numbers that DESIRE reads upon startup and when resetting the capacitor settings. These numbers correspond to the settings of the relays that determine the total capacitance closed across the line when running DESIRE program.
rob471.exe	This is the executable file that collects the data from the capacitor box. The program reads in the relay settings that correspond to the capacitor values that the user defines from the file''data.cap", which is created while DESIRE is running. The program then sets the relays for each setting and closes the line across the capacitor using the three phase switch, and records the current waveform. The data is stored in ram and is then downloaded to files with the root name''idata".
robmat.bat	This batch file is used to copy the file matrob.m to matlab.m and return to the desire directory, the directory which we are currently running DESIRE from.
matlab	This is the command used to call Matlab. This program is currently using the dos based 386 version of Matlab.
transit.exe	This program copies the output files from matlab, which are in exponetial notation, into floating point values of the proper length to be used by DESIRE.
matorig.bat	This file copies the file matold.m to matlab.m and returns to the desire directory.
matlab.m	This is the startup m-file that Matlab calls upon entering the program. In that way, it is similar to the autoexec.bat file used in dos

```
                based computer systems.

matrob.m        Matrob.m is the script file that matlab uses as
                a startup file when DESIRE is running.  It calls
                the script file robir.m and quits matlab.

matold.m        Matold.m is the original matlab startup file.
                It is copied here so that Matlab can be used
                when not within the DESIRE program.

robir.m         This script file reads the``idata" files and
                creates the best fit data files using the fmins
                command.  These files are saved with the``odata"
                root name.  The estimated parameters of L, R,
                and F are saved in lmat.mat, rmat.mat, and
                fmat.mat respectively.

peak.m          }
decay4.m        } These files are called from within the robir.m file.
dec5.m          }
```

Using DESIRE When the DESIRE.EXE file is run, the graphical user interface is entered. In this interface, the user has the ability to acquire new data, save the data, and load previously saved data. The interface contains area for four plots of data, a text area that displays the estimated R and L, eight capacitor setting buttons, and seven function buttons. At all times in the program, all buttons and four plots are shown, but the data in the plots and the functionality of the buttons depends on the status of the program. As the program is begun, all capacitor and function buttons appear, as well as four blank plots.

Each of the buttons can be"pressed" by clicking on it with the mouse or by hitting a keyboard shortcut. For each of the capacitor buttons, this shortcut is the number of the capacitor. For the function buttons, the shortcut is the first letter of the button name (ie."Cap 1"'s shortcut is "1", and"Plot"'s shortcut is"p").

```
Functionality of the buttons:

    Cap     The capacitor buttons control the capacitor value that
            will be placed across line voltage when the plot button is
```

pressed. The value of the capacitor affects both the frequency of oscillation of the transient and the amount of current that is drawn. A higher capacitor value will produce a slower oscillation that draws a larger amount of current than a smaller capacitor will. Clicking on a capacitor with the left mouse button will increase the value by about 0.5uF, while the right button will increase it by about 5uF. Using the keyboard shortcut will increase it by 0.5uF. The values of the capacitors can be changed only when no plotted data is shown on the screen.

Plot The plot button interupts the graphic interface to collect data using the capacitor settings that are currently displayed. The rob471 program is called to collect the raw data. At the completion of this program, matlab is run using the matrob.m file as the matlab startup program. The m-file robir.m is run to best fit the data and predict the supply R and L. This data is saved to the hard drive, then converted to floating point notation using the transit.exe program. After completing these steps, the graphical interface is reentered and the real and simulated data are plotted in the graphs and the estimated data is displayed in the text area. The data plotted in is found in the files with the root name''data". Plot can be run at any time.

View The view button allows the user to toggle between viewing the plots of the first four sets of data and the last four sets of data. View can only be used when data is plotted on the screen.

Save The save button saves data that is currently plotted to files with the root name''svdat". If there are''svdat" files at the time that save is called, these files will be copied over. The save name is functional only when data is plotted on the screen.

Load The load button copies the''svdat" files to''data" files and plots this data on the screen. Load can be called at any time, but assumes that there the''svdata" files exist and were created by the DESIRE program. The information contained in the''data" files will be written over when load is called.

Reset The reset button clears the plots and the text area, then resets the capacitor buttons to the settings in''default.cap". Reset can be pressed at any time. Pressing reset allows the user to change the capacitor settings to collect new data once plot has been run.

B&W/Color These buttons toggle between the color setting and the black and white setting. The black and white setting has been implemented for better pictures, and appears better on a black and white monitor than the color setting does. These buttons can be pressed at any time, and change only whether the screen appears in color or black and white.

Quit The quit button returns the user to the dos prompt, and can be called at any time. The data that has last been used by DESIRE still remains in the''data" files.

C.2 DESIRE Code

C.2.1 Desire.c

```
/****************************************/
/*              DeSIRe                  */
/*       Determination of Supply        */
/*      Inductance and Resistance       */
/*                                      */
/*       written by Tom Respress        */
/*       assisted by Rob Lepard         */
/****************************************/



#define MAIN 1
#include "globcon.h"
#include "globvar.h"
#include "globfunc.h"
#include "plot.h"
#include "button.h"
#include "loadsave.h"

void init(int colortype);
void refresh(pbutton pbuttons, int colortype, int view);
void refresh_all(pbutton pbuttons, int colortype, int view);
void refresh_button(pbutton pbuttons, int colortype);
void refresh_view(int colortype, int view);
void paint(int colortype);
void show_dos();
void init_text_area(int color_setting);
int print_R_and_L(pbutton pbutton_start, int color_setting);

/* change order of keyboard grab information - move near getmouse(...) */

/***************************/
```

```
/* MAIN --> executes DESIRE */
/*****************************/
main()
{
int captured_key, button_done;
/* boolean variable for termination of keystroke and button */
int xp, yp;
/* xp,yp are position variables */
        int bs=0, ks=0, flag=0, hit=0, plot=0, view=0;
/* others tell termination, and input */
int color_setting,color,i;
/* allows for change color to be on */
char charvalue, b[15];
/* grabs I/O character, stores the string of the button name */
pbutton pbuttons, pbutton_start;
/* initialize starting button and current button pointer */

/*************************************************************/
/************************** Initialization ***************/
/*************************************************************/
color_setting = 1;
/* color is on */
color = LIGHTGRAY;
/* set the button and text area color */
pbutton_start = pbuttons = init_buttons();
/* initialize buttons */

init(color_setting);
/* calls initializes mouse,fonts. */
refresh_all(pbutton_start,color_setting,view);
/* puts everything to screen */
show_mouse();

/*************************************************************/
/********************** Start I/O capture procedure *******/
/*************************************************************/

/* This function checks to see if there has been
any button presses or   */
/* key presses. The functionality buttons
(Load,Save,Plot,Reset,Color,   */
/* Quit, and View) can be accessed by
keystroke(first letter of function)*/
/* or by the left mouse button.  The capacitor
values can be adjusted by */
/* a factor of 1 or 8 depending, respectively, on
whether the left or    */
/* right mouse button was pressed.  Certain buttons
will be active or    */
/* inactive depending on the programs current state. i.e. you can only   */
/* save if there is something plotted.                                   */
while(flag==0)
```

```
{
button_done = 0;           /* reset constants upon every entry */
pbuttons = pbutton_start;  /* reset the pointer to the button array upon every entry */

getmouse(&xp, &yp, &bs);   /* grab mouse info */
if(bs!=0 || ks==1)
{
/*********************************************/
/****** Translate keystrokes to buttons ******/
/*********************************************/
if(ks==1)
{
char keystroke_name[10];  /* translate keystroke to string */

captured_key=0;
switch (charvalue)
{
case '1':
sprintf(keystroke_name,"Cap %d",1);
break;
case '2':
sprintf(keystroke_name,"Cap %d",2);
break;
case '3':
sprintf(keystroke_name,"Cap %d",3);
break;
case '4':
sprintf(keystroke_name,"Cap %d",4);
break;
case '5':
sprintf(keystroke_name,"Cap %d",5);
break;
case '6':
sprintf(keystroke_name,"Cap %d",6);
break;
case '7':
sprintf(keystroke_name,"Cap %d",7);
break;
case '8':
sprintf(keystroke_name,"Cap %d",8);
break;
case 'l':
strcpy(keystroke_name,"Load");
break;
case 'b':
strcpy(keystroke_name,"B&W");
break;
case 'c':
strcpy(keystroke_name,"Color");
break;
case 'p' :
strcpy(keystroke_name,"Plot");
```

```
break;
case 'q' :
strcpy(keystroke_name,"Quit");
break;
case 'r' :
strcpy(keystroke_name,"Reset");
break;
case 's' :
strcpy(keystroke_name,"Save");
break;
case 'v' :
strcpy(keystroke_name,"View");
break;
default :
strcpy(keystroke_name,"");
captured_key=1;
button_done=1;                                    /* no button */
ks=0;
break;
}
while(captured_key==0 && pbuttons!=NULL)                  /* move to appropriate button */
{
strcpy(b,pbuttons->name_type);
if(strcmp(b,"Cap")==0){strcpy(b,pbuttons->name_but);}
/* check for cap if so then change what b is */
if (strcmp(b, keystroke_name)==0)
captured_key = 1;                                  /* found button */
else
pbuttons=pbuttons->next;              /* not found goto next button */
}
if(captured_key==0){ks=0;}                         /* if no button then no keystroke */
}
/**************************/
/****** INPUT HANDLER ******/
/**************************/
while(button_done==0 && pbuttons!=NULL)
{
if (fall_in_range(pbuttons, xp, yp)==1|| ks==1)
  /* if the mouse was pressed on button or keystroke pressed */
{
strcpy(b,pbuttons->name_type);
/*****************************************************************/
/* Cap--> increments the cap button that was pressed by 1 or 8 */
/*****************************************************************/
if (strcmp(b,"Cap")==0 && plot==0)
{
button_done = 1;

hide_mouse();
lightup(pbuttons);
if(bs == 2)
change_cap_value(pbuttons,2);           /* change by 8 */
```

```
else
change_cap_value(pbuttons,1);           /* change by 1 */
print_cap_value(pbuttons, color_setting);
/* print the new value to screen */
show_mouse();
}
/*********************************************************************/
/* Checks if Plot button was pressed, if so, then executes plot command */
/*********************************************************************/
else if (strcmp(b,"Plot")==0)
{
int y;

button_done = 1;

hide_mouse();
lightup(pbuttons);
y = save_cap_value(pbutton_start, "data");    /* save current cap settings */
/* Err handling on save */
if(y==1)                                    /* if error then reinit screen */
{
refresh_all(pbutton_start,color_setting,view);
}
else                                          /* else continue with script */
{
show_dos();      /* allow message while waiting for script */
system("rob471");
system("robmat > crap.out");
system("matlab");
system("matorig > crap.out");
system("transit > crap.out");
init(color_setting);
refresh(pbutton_start,color_setting,view);
plot = 1;                               /* Data is currently plotted */
}
show_mouse();
}
/*********************************************************************/
/* Reset--> resets all cap values to default, clears text_area and plots */
/*********************************************************************/
else if (strcmp(b,"Reset")==0)
{
int y;

button_done = 1;
plot = 0;                                     /* no plot displayed  */

hide_mouse();
lightup(pbuttons);
y =load_cap_value(pbutton_start,"default");
/* load cap values into pbutton array */
refresh_all(pbutton_start,color_setting,view);  /* refresh the screen */
```

```
show_mouse();
}
/*************************************************************************/
/* B&W/Color button--> If it was color then it prints B&W, vice versa */
/*************************************************************************/
else if(strcmp(b,"B&W")==0 || strcmp(b,"Color")==0)
{
button_done = 1;
color_setting=!color_setting;     /* change color setting to opposite */

hide_mouse();
lightup(pbuttons);
if(plot==0)
refresh_all(pbutton_start,color_setting,view);
/* refresh the screen without plots   */
else
refresh(pbutton_start,color_setting,view);
/* refresh the screen,including plots */
change_color_name(pbuttons,color_setting);
/* change color name on button */
show_mouse();
}
/***********************************************/
/* View--> shows the first or second 4 cap plots */
/***********************************************/
else if (strcmp(b,"View")==0 && plot==1)
{
button_done = 1;
view = !view;                              /* view goes to other value */

hide_mouse();
lightup(pbuttons);
change_view_name(pbuttons,color_setting);
refresh_view(color_setting,view);
show_mouse();
}
/*****************************************************/
/* Load--> loads svdat*.* into data*.*, then plots it */
/*****************************************************/
else if (strcmp(b,"Load")==0)
{
int y;

button_done = 1;

hide_mouse();
lightup(pbuttons);
show_dos();
load_sequence();                                  /* do load commands */
y = load_cap_value(pbutton_start, "svdat");        /* load in cap values */
if(y==1)
refresh_all(pbutton_start,color_setting,view); /* refresh screen */
```

```
else
{
refresh(pbutton_start,color_setting,view);     /* refresh screen */
plot = 1;
}
show_mouse();
}
/************************************************/
/* Save--> saves current plot values to svdat*.* */
/************************************************/
else if (strcmp(b,"Save")==0 && plot==1)
{
button_done = 1;

hide_mouse();
lightup(pbuttons);
show_dos();                                    /* clear screen */
save_sequence();                               /* copy files to svdat */
refresh(pbutton_start,color_setting,view);     /* refresh screen */
show_mouse();
}
/***************************************/
/* QUIT button--> program quits into DOS. */
            /***************************************/
else if (strcmp(b,"Quit")==0)
{
button_done = 1;
flag=1;

hide_mouse();
lightup(pbuttons);
show_mouse();
_clearscreen(_GCLEARSCREEN);
_setvideomode(_DEFAULTMODE);
}  /* end else if */
}  /* end else if */
pbuttons = pbuttons->next;
}  /* end for */
bs=0;                              /* reset the button pressed counter */
ks=0;                              /* reset the keyboard pressed counter */
}  /* end if */
/*****************************/
/* grab keyboard information */
/*****************************/
else
{
/* This checks the number of keyboard hits and does the */
/* appropriate plots or quits out of the program.       */

charvalue = kbhit();
if(charvalue)
{
```

```
ks=1;
charvalue=getch();
}
else
ks=0;
}
}
}

/*************************************************************************/
/***************************** System ************************************/
/*************************************************************************/

/*************************************************************************/
/* Initializes the graphics mode with certain fonts which are to be used */
/*************************************************************************/
void init(int color_setting)
{
_setvideomode(_VRES16COLOR);
_registerfonts("tmsrb.fon");
_setfont("t'tms rmn'");
_setfont("h12w6");

/* This will paint the background in color and put up the appropriate */
/* title bar. It also initializes the mouse.                          */
paint(color_setting);
initmouse();
hide_mouse();
}
/***********************************************************/
/* This function is used preceeding any system() commands */
/* it will alert the user to the fact that it is thinking */
/***********************************************************/
void show_dos()
{
_setcolor(BLACK);
_rectangle(_GFILLINTERIOR, 0, 0, 640, 480);
putgtext("DESIRE is loading in values...", 100, 100, WHITE);
}

/***********************************************************/
/************************ REFRESH **************************/
/***********************************************************/

/**************************************/
/* REFRESH --> refreshes entire screen */
/**************************************/
void refresh(pbutton pbuttons, int color_setting, int view)
{
int y, z;

paint(color_setting);
```

```
if(color_setting==1){init_text_area(LIGHTGRAY);} else{init_text_area(BLACK);}
y = print_R_and_L(pbuttons, color_setting);            /* find out if can do it */
if(y==1)
/* if can't do it, then clean screen else finish refreshing the current screen */
refresh_all(pbuttons,color_setting,view);
else
{
z = plot_me(color_setting,view);                       /* find out if can do it */
if(z==1)
/* if can't do it, then clean screen else finish refreshing the current screen */
refresh_all(pbuttons,color_setting,view);
else
refresh_button(pbuttons,color_setting);
}
}
/*****************************************************/
/* REFRESH_ALL --> returns to initial startup screen */
/*****************************************************/
void refresh_all(pbutton pbuttons, int color_setting, int view)
{
paint(color_setting);
if(color_setting==1){init_text_area(LIGHTGRAY);} else{init_text_area(BLACK);}
init_plot(color_setting,view);
refresh_button(pbuttons,color_setting);
}
/*******************************************/
/* REFRESH_BUTTON --> refreshes the buttons */
/* and their corresponding values           */
/*******************************************/
void refresh_button(pbutton pbutton_start, int colortype)
{
pbutton pbuttons;
pbuttons = pbutton_start;

while(pbuttons!=NULL)
{
create_button(pbuttons, colortype);
if(strcmp(pbuttons->name_type,"Cap")==0)
print_cap_value(pbuttons, colortype);
pbuttons=pbuttons->next;
}
}
/*************************************/
/* REFRESH_VIEW --> refreshs plot area, */
/* specifically for View change button  */
/*************************************/
void refresh_view(int color_setting, int view)
{
if (color_setting==0)
{
_setcolor(BLACK);
_rectangle(_GFILLINTERIOR, 131, 14, 640,480);
```

```
_setcolor(WHITE);
_rectangle(_GBORDER, 0, 0, 639,479);
}
else if(color_setting==1)
{
_setcolor(DARKBLUE);
_rectangle(_GFILLINTERIOR, 131, 14, 639, 479);
}
plot_me(color_setting,view);
}

/*********************************************************/
/********************** Text area ************************/
/*********************************************************/

/*********************************************************/
/* INIT_TEXT_AREA --> this prints the text area for L and R */
/* Color is depedent on color setting                    */
/*********************************************************/
void init_text_area(int color_setting)
{
char str1[5], str2[5];

strcpy(str1,"L");
strcpy(str2,"R");
if (color_setting==0)
{
_setcolor(BLACK);
_rectangle(_GFILLINTERIOR, 55, 26, 130, 418);
_setcolor(WHITE);
_rectangle(_GBORDER, 55, 15, 130, 418);
putgtext(str1, 15, 75, WHITE);
putgtext(str2, 15, 110, WHITE);
}
else
{
_setcolor(LIGHTGRAY);
_rectangle(_GFILLINTERIOR, 55, 15, 130, 24);
putgtext(str1, 15, 75, BLACK);
putgtext(str2, 15, 110, BLACK);

_setcolor(LIGHTGRAY);
_rectangle(_GFILLINTERIOR, 55, 26, 130, 418);
}
}
/*****************************************************/
/* this prints Resistance and Inductance onto screen */
/* this function looks in two files: data.lma and    */
/* data.rma each are 8x1 matrix of floating numbers  */
/*****************************************************/
int print_R_and_L(pbutton pbutton_start, int color_setting)
{
```

```
char str1[20], var1[20], var2[20];
int color, invalid=0;
float num1, num2;
FILE *input1, *input2;
pbutton pbutton_now;
pbutton_now = pbutton_start;

/************ Err handling **************/
/* check if data.rma or data.lma exists */
/****************************************/
if((input1=fopen("data.rma","r"))==NULL || (input2=fopen("data.lma","r"))==NULL)
{
invalid = 1;
_setcolor(WHITE);
_rectangle(_GFILLINTERIOR, 100, 100, 300, 300);
if (input1==NULL)
putgtext("No data.rma", 200, 110, BLACK);
else
putgtext("No data.lma", 200, 110, BLACK);
putgtext("Hit any key to continue", 130, 110, BLACK);
}

if(invalid)
while(!kbhit()){}
else
{
if(color_setting==0)                          /* set color for text output */
color=WHITE;
else
color=BLACK;
while(strcmp(pbutton_now->name_type,"Cap")==0)
{
fscanf(input1,"%s",var1);
fscanf(input2,"%s",var2);
num1 = atof(var1);
num2 = atof(var2);
sprintf(str1,"%6.5f",num2);
putgtext(str1, pbutton_now->ystart+5, 55, color);
sprintf(str1,"%5.3f",num1);
putgtext(str1, pbutton_now->ystart+5, 100, color);
pbutton_now=pbutton_now->next;
}
fclose(input1);
fclose(input2);
}
return invalid;
}
```

C.2.2 Header Files

Plot.h

```
/*************************    PLOT.H    *************************/
/*********************************************************************/
/*  This header file contains all information relevant to the plot  */
/*  function.  This includes the plotit2 function itself as well as  */
/*  the initialization of the plot, and functions which plotit2 uses */
/*********************************************************************/
/*********************************************************************/

#define NUL2 0
void init_plot(int color_setting, int view);
int plot_me(int color_setting, int view);

static double manlist[MANLISTSIZE] = {1.0, 2.0, 4.0, 8.0, 10.0, 20.0, 40.0,
            80.0, 100.0, 200.0, 400.0, 800.0, 1000.0};
static int pticlabs[MANLISTSIZE] = {2, 20, 16, 16, 20, 20, 16, 16, 20, 20, 16,
            16, 20};
static int bticlabs[MANLISTSIZE] = {2, 16, 16, 16, 20, 16, 16, 16, 20, 16, 16,
            16, 20};
static int vlabpos[NVTICLAB] = {BLABPOS, Q1POS, MLABPOS, Q3POS, TLABPOS};

static int vlabposg[NVTICLAB] = {215, 170, 125, 80, 35};

static char plotvert[] = "Current";
static char plothoriz[] = "Time, ms";

/***************************/
/**** initialize plotit ****/
/***************************/
void init_plot(int color_setting, int view)
{
   int i, z, a[20];
   int color1, color2, color3;
   char plottitle[15], title[15];
   int xorg[4] = {XORG1,XORG2,XORG3,XORG4};
   int yorg[4] = {YORG1,YORG2,YORG3,YORG4};
   strcpy(title,"Capacitor");
   if(color_setting==0)
   {
      color1=BLACK;
      color2=BLACK;
      color3=WHITE;
   }
   else
   {
      color1=DARKBLUE;
      color2=DARKBLUE;
```

```
      color3=LIGHTGRAY;
   }

   /* sets array to be used in plotting blank screens for the initial setup */
   for(i=0;i<2; i++){a[i] = 1;}

   /* plots four empty graphs up on the screen */
   for(i=view*4+1,z=0;i<view*4+5;i++,z++)
   {
      sprintf(plottitle,"%s %d",title, i);
      plotit2(a, a, i, 0.0, 10.0, plottitle, plotvert,
plothoriz, xorg[z], yorg[z], color1, color2, color3);
   }
}

int plot_me(int color_setting, int view)
{
   int i, z, counter1, invalid=0;
   int color1, color2, color3;
   int xorg[4] = {XORG1,XORG2,XORG3,XORG4};
   int yorg[4] = {YORG1,YORG2,YORG3,YORG4};
   char plottitle[15],title[15],str1[40],str1f[40];
   FILE *input1, *input2;

   strcpy(title,"Capacitor");

   if(color_setting==0)
   {
      color1=WHITE;
      color2=WHITE;
      color3=WHITE;
   }
   else
   {
      color1=RED;
      color2=GREEN;
      color3=LIGHTGRAY;
   }

   for(i=view*4+1,z=0;i<view*4+5 && !invalid;i++,z++)
   {
      sprintf(plottitle,"%s %d",title, i);
      sprintf(str1,"data%d.mat",i);
      sprintf(str1f,"data%df.mat",i);

      if((input1 = fopen(str1,"r"))==NULL || (input2 = fopen(str1f,"r"))==NULL)
```

```
         invalid = 1;
      else
      {
         for(counter1=0;fscanf(input1, "%d", &ch1[counter1])!=EOF;){++counter1;}
         for(counter1=0;fscanf(input2, "%d", &ch1f[counter1])!=EOF;){++counter1;}
         plotit2(ch1, ch1f, counter1, 0.0, (double) (counter1)*.005,
plottitle, plotvert, plothoriz, xorg[z], yorg[z], color1, color2, color3);
         fclose(input1);
         fclose(input2);
      }
   }
   return invalid;
}

/* NOTE: This version of plotit2 only plots two variables vs. time.  This  */
/* is specific to this application. The number of variables to be plotted  */
/* can be altered depending on the need of the user.                        */

/* plotit2 puts up a plot of two variables against a fourth.
   It is intended to plot three phase currents, real or reactive power
   levels against time. The variables are:

      pa, pb and pc are double precision vectors of dimension n,
   and they are the variables which are plotted.

      ts is the start of the independent variable,
      tf is the end.

      titstring should be no more than 40 characters:
         it is the plot title.

      labstring should be no more than 14 characters:
         it labels the y axis. (it should be something like
         "watts" or "amperes"

      ordstring labels the x-axis (something like "hours").

   This procedure can handle the prefixes "kilo", "Mega" and "milli",
   and if they don't apply, will generate a factor of ten notation.
*/

/* Remember to note that this plotit2 will only plot two sets of points in*/
/* each graph. This version allows you to specify the color of the points */
/* as well as the color of the graph itself.                               */

/**** plotit2 FUNCTION ****/
```

```
plotit2(pa, pb, n, ts, tf, titstring, labstring, ordstring,
             xorg, yorg, plotcolor1, plotcolor2, graphcolor)
int pa[], pb[], n;
double ts, tf;
char *labstring, *titstring, *ordstring;
int xorg, yorg, plotcolor1, plotcolor2, graphcolor;
{
   short x, y, xc(), yc();
   int i, j, ltit, titoff, ym, yl, ls, s;
   double a, b, ymax, ymin, ymaxlim, yminlim, toplim();
   void xtics(), ytics(), vertprint(), magprt(), xaxis();
   char title[TITLENGTH];
   char vstring[ABLENGTH];
   double ymaxt, ymint, manpart, expart;
   int mx,my,mb,c;
   int flag = 0;
   FILE *pf, *fopen();

   pf = fopen("out.dat","w");
   for (i = 0; i < n; i++) {
      fprintf (pf,"%d %f\n",i,pa[i]);
   }
   fclose(pf);


   mx = my = mb = 0;

   /* vertical axis involves finding extrema */

   ymax = 0; ymin = 0;

   for (i=0; i<n; i++)
   {
      if (pa[i] > ymax) ymax = pa[i]; /* find max and min values */
      if (pb[i] > ymax) ymax = pb[i]; /* find max and min values */
      if (pa[i] < ymin) ymin = pa[i];
      if (pb[i] < ymin) ymin = pb[i];
   }

   ymaxt = fabs(ymax/10);                              /* biggest number */
   if ((ymint = fabs(ymin/10)) > ymaxt) ymaxt = ymint;

   if (ymaxt == 0) ym = 1;
   else ym = (int) floor(log10(ymaxt)/3.0); /* order of magnitude */
   yl = 3 * ym;                              /* in engineering notation */
```

```
expart = pow (10.0, (double) yl);
manpart = ymaxt / expart;

s = select(manpart, manlist, MANLISTSIZE);      /* index to list of
               mantissas for plot */

ymaxlim = manlist[s] * expart;          /* actual normalizing value */

if (ymin <0) yminlim = -ymaxlim;        /* bottom limit is */
else yminlim = 0.0;                     /* symmetrical or zero */

/* now write down variables in screen coordinates */

for (i=0; i<n; i++)
{
   pan[i] = ((double) YSIZE) * (pa[i] - 10*yminlim)/
      (10*(ymaxlim - yminlim));
   pbn[i] = ((double) YSIZE) * (pb[i] - 10*yminlim)/
      (10*(ymaxlim - yminlim));
}

/* The next two statements set up the screen (a microsoft 6.0 call)
and draw a rectangle around what will be the plot */

_setcolor(graphcolor);

_rectangle(_GBORDER, xorg, yorg-YSIZE, xorg+XSIZE, yorg);

labaxis(s, ymin, xorg, yorg, graphcolor); /* generate vertical axis markers */

ls = strlen(labstring);         /* label text length */
if (ls > ABSL) ls = ABSL;       /* must be limited */

/* This thing will prepend "kilo", "Mega" and "milli" to the
units for the abscissa (which is an argument to the procedure).
If none of these (or of course no prefix at all) applies, it
will generate a legend of the form "X 10^ xx"  to show order
of magnitude. This next fragment of code does that, and prints
the abscissa label, vertically, next to the plot */

switch(ym)                      /* ym describes order of magnitude */
{
   case 0: vertprint(labstring, xorg, yorg, graphcolor);
      break;
   case 1: vstring[0]='k';vstring[1]='i';vstring[2]='l';
      vstring[3]='o';
```

```
      for (i=0; i<ls; i++) vstring[i+4]=labstring[i];
      vstring[ls+4]=NUL2;
      vertprint(vstring, xorg, yorg, graphcolor);
      break;
   case 2: vstring[0]='M';vstring[1]='e';vstring[2]='g';
      vstring[3]='a';
      for (i=0; i<ls; i++) vstring[i+4]=labstring[i];
      vstring[ls+4]=NUL2;
      vertprint(vstring, xorg, yorg, graphcolor);
      break;
   case -1:vstring[0]='m';vstring[1]='i';vstring[2]='l';
      vstring[3]='l';vstring[4]='i';
      for (i=0; i<ls; i++) vstring[i+5]=labstring[i];
      vstring[ls+5]=NUL2;
      vertprint(vstring, xorg, yorg, graphcolor);
      break;
   default: vertprint(labstring, xorg, yorg, graphcolor);
       magprt(yl, xorg, yorg, graphcolor);
       break;
}

/* the plot title appears above the plot on the screen. It is
plotted centered, and the next fragmet of code decides where to
put it and writes it there */

ltit = strlen(titstring);       /* length of plot title */

if (ltit < TITLENGTH-1)
{
   for (i=0; i<ltit; i++)
      title[i] = titstring[i];
   title[ltit] = NUL2;
   titoff = ltit/2;
}
else    /* stated title is too long and will be clipped */
{
   for (i=0; i<TITLENGTH-1; i++)
      title[i] = titstring[i];
   title[TITLENGTH-1] = NUL2;
   titoff = TITLENGTH/2;
}

/* This positions the title on the appropriate graph */

if(yorg<240)
{
```

```
      /*if the graph is on the top half of the screen */
      if(xorg<320)
      {
         /*if the graph is on the left half of the screen */
         putgtext(title, 25, 290-(titoff*8), graphcolor);
      }
      else
      {
         /*if the graph is on the right half of the screen */
         putgtext(title, 25, 550-(titoff*8), graphcolor);
      }
   }
   else
   {
      /*if the graph is on the bottom half of the screen */
      if(xorg<320)
      {
         /* if the graph is on the left half of the screen */
         putgtext(title, 265, 290-(titoff*8), graphcolor);
      }
      else
      {
         /* if the graph is on the right half of the screen */
         putgtext(title, 265, 550-(titoff*8), graphcolor);
      }
   }

   /* Ordinate title is placed below the chart on the screen.
   The next fragment of code prepends it with "Time", and
   centers the whole thing */

   ltit = strlen(ordstring);       /* length of ordinate title */

/*   title[0]='T';title[1]='i';title[2]='m';title[3]='e';
   title[4]=' ';title[5]=' ';
*/

   if (ltit < TITLENGTH-1)
   {
/*      for (i=0; i<ltit; i++)
         title[i+6]=ordstring[i];
*/
      ordstring[ltit] = NUL2;
      titoff = ltit/2 + 3;
   }
   else
```

```
   {
/*    for (i=0; i<TITLENGTH-7; i++)
         title[i+6] = ordstring[i];
*/
      ordstring[TITLENGTH-1]=NUL2;
      titoff = TITLENGTH/2;
   }

   /* This positions the ordinate title appropriately */

   if(yorg<240)
   {
      /* if the graph is on the top half of the screen */
      if(xorg<320)
      {
         /* if the graph is on the left half of the screen */
         putgtext(ordstring, 225, 290-(titoff*8), graphcolor);
      }
      else
      {
         /* if the graph is on the right half of the screen */
         putgtext(ordstring, 225, 550-(titoff*8), graphcolor);
      }
   }
   else
   {
      /* if the graph is on the bottom half of the screen */
      if(xorg<320)
      {
         /* if the graph is on the left half of the screen */
         putgtext(ordstring, 465, 290-(titoff*8), graphcolor);
      }
      else
      {
         /* if the graph is on the right half of the screen */
         putgtext(ordstring, 465, 550-(titoff*8), graphcolor);
      }
   }

   /* The routine "xaxis" has a fiendishly clever way of deciding
   how many "tics" to put on the x-axis. It also puts the starting
   and ending times at appropriate places on the axis, just below
   the plot. */

   xaxis(ts, tf, xorg,yorg, graphcolor);  /* puts time and x-axis tics */
```

```
   /* This routine plots three curves, in the colors GREEN, RED,
   and BLUE. These three sets follow. */

   _setcolor(plotcolor1);

   _moveto(xc(0.0, xorg), yc(pan[0], yorg));

   for (i = 1; i<n; i++) {
      x = xc((((double) i) * XSIZE)/((double) (n-1)), xorg);
      y = yc(pan[i], yorg);
      _lineto(x, y);
   }

   _setcolor(plotcolor2);

   _moveto(xc(0.0, xorg), yc(pbn[0], yorg));

   for (i = 1; i<n; i++) {
      x = xc((((double) i) * XSIZE)/((double) (n-1)), xorg);
      y = yc(pbn[i], yorg);
      _lineto(x, y);
   }

   _setcolor(graphcolor);
   _rectangle(_GBORDER,xorg, yorg-YSIZE, xorg+XSIZE, yorg);
   labaxis(s, ymin, xorg, yorg, graphcolor);
}

short xc(x, xorg)  /* Translate raw numbers into screen coordinates */
double x;
{
   return((short) (x + xorg));
}

short yc(y, yorg)       /* same for the vertical axis */
double y;
{
   return((short) (yorg - y));
}

/* xaxis does two things. It decides how many tics to put on the x
axis(and places them there). It also formats the beginning and
ending numbers for time and places them in the appropriate spaces
below the active plot area */

void xaxis (ts, tf, xorg,yorg, color)
```

```
double ts, tf;
int xorg, yorg, color;
{
   double time, dt, x, xf, dx, xt;
   char label[12];
   short xtic, i, ys, yf;

   time = fabs(tf - ts);   /* so it works if these are reversed */
   if(time==0)
      x=1;
   else
      x = log10(time)/log10(2.0);     /* log to base 2 */

   xf = floor(x);                  /* to pull out integer part */

   dt = pow (2.0, xf)/((double) TICSPEROCTAVE);

   dx = XSIZE * dt/time;           /* spacing between tics */

   ys = yorg;                      /* start and end of each tic */
   yf = yorg - TICLENGTH;

   for (i = 1; (xt = dx * i) < 200; i++)  /* tic by tic */
   {
      xtic = xc(xt, xorg);            /* horizontal position */
      _moveto(xtic, ys);
      _lineto(xtic, yf);
   }

   /* now do the end labels */

   sprintf (label,"%-5.2f",ts);

   /* prints the labels accordingly */
   if(yorg<240)
   {
      if(xorg<320)
      {
         putgtext(label, ORDROW2G, XLABSTART1G, color);
         sprintf(label,"%5.2f", tf);
         putgtext(label, ORDROW2G, XLABEND1G, color);
      }
      else
      {
         putgtext(label, ORDROW2G, XLABSTART2G, color);
         sprintf(label,"%5.2f", tf);
```

```
            putgtext(label, ORDROW2G, XLABEND2G, color);
         }
      }
      else
      {
         if(xorg<320)
         {
            putgtext(label, ORDROW3G, XLABSTART1G, color);
            sprintf(label,"%5.2f", tf);
            putgtext(label, ORDROW3G, XLABEND1G, color);
         }
         else
         {
            putgtext(label, ORDROW3G, XLABSTART2G, color);
            sprintf(label,"%5.2f", tf);
            putgtext(label, ORDROW3G, XLABEND2G, color);
         }
      }
}

void ytics(n, xorg, yorg)  /* This produces n uniformly spaced tics on the y axis */
int n;
{
   double spacing;
   short xs, xf, y, i;

   xs = xorg;
   xf = xorg + TICLENGTH;

   spacing = ((double) YSIZE) / ((double) n);
   for (i = 1; i<n; i++)
   {
      y = yc(((double) i) * spacing, yorg);
      _moveto(xs, y);
      _lineto(xf, y);
   }
}


void vertprint(vstring, xorg, yorg, color)
/* This one prints a label top to bottom */
char vstring[];
int xorg, yorg, color;
{
   int i, sl, vp, voff;
```

```
char ds[2];     /* actually, each piece is a character and a NUL2 */

sl = strlen(vstring);   /* this is length of the string */

if (sl > ABLENGTH) sl = ABLENGTH;       /* this clips so it can't
               be too long */

voff = ABLENGTH/2 -sl/2;        /* position on the screen */

for (i=0; i<sl; i++)            /* then print, character by */
{                               /* character */
   ds[0] = vstring[i];     /* note that we have hacked the */
   ds[1] = NUL2;           /* procedure _outtext*/
   vp = TLABPOSG + voff + i;

   /* prints the vertical label accordingly */
   if(yorg<240)
   {
      if(xorg<320)
      {
         if(ds[0]>64 && ds[0]<95)
         {
            putgtext(ds, vp*16, PRINTCOL1G-3, color);
         }
         else
         {
            putgtext(ds, vp*16, PRINTCOL1G, color);
         }
      }
      else
      {
         if(ds[0]>64 && ds[0]<95)
         {
            putgtext(ds, vp*16, PRINTCOL2G-3, color);
         }
         else
         {
            putgtext(ds, vp*16, PRINTCOL2G, color);
         }
      }
   }
   else
   {
      if(xorg<320)
      {
         if(ds[0]>64 && ds[0]<95)
```

```
            {
               putgtext(ds, vp*16+240, PRINTCOL1G-3, color);
            }
            else
            {
               putgtext(ds, vp*16+240, PRINTCOL1G, color);
            }
         }
         else
         {
            if(ds[0]>64 && ds[0]<95)
            {
               putgtext(ds, vp*16+240, PRINTCOL2G-3, color);
            }
            else
            {
               putgtext(ds, vp*16+240, PRINTCOL2G, color);
            }
         }
      }
   }
}


void magprt(m, xorg, yorg, color) /* this prints in cases where magnitude is not */
int m, xorg, yorg, color;                    /* within the milli to mega range */
{
   int vp;
   char str[9];

   sprintf(str, "X 10e%3d",m);
   str[8] = NUL2;
   vp = TLABPOS + ABLENGTH - 2;

   if(yorg<240){
      if(xorg<320){
         putgtext(str, vp*16, PRINTCOL1G, color);
      }
      else{
         putgtext(str, vp*16, PRINTCOL2G, color);
      }
   }
   else{
      if(xorg<320){
         putgtext(str, vp*16+240, PRINTCOL1G, color);
      }
```

```
      else{
         putgtext(str, vp*16+240, PRINTCOL2G, color);
      }
   }

}

labaxis(s, min, xorg, yorg, color)
/* this procedure generates the vertical axis */
int s;                  /* labels, including five numbers and a somewhat */
double min;             /* arbitrary number of tics */
int xorg, yorg, color;
{
   double max_norm, min_norm, tic_value;
   char tic_label[8];
   int i;

   max_norm = manlist[s];  /* orders taken from manlist */

   if (min < 0)    /* min value less than zero: two-valued axis */
   {
      min_norm = - max_norm;
      ytics(bticlabs[s], xorg, yorg);
   }
   else
   {
      min_norm = 0;
      ytics(pticlabs[s], xorg, yorg);
   }

   for (i=0; i<NVTICLAB; i++)      /* NVTICLAB is how many to print */
   {
      tic_value = min_norm + (max_norm - min_norm) *
         ((double) i)/((double) (NVTICLAB - 1));
      sprintf(tic_label, "%5.1lf", tic_value);

      if(yorg<240){
         if(xorg<320){
            putgtext(tic_label, vlabposg[i], XGTEXTORG1-7, color);
            }
         else{
            putgtext(tic_label, vlabposg[i], XGTEXTORG2-7, color);
            }
      }
      else{
         if(xorg<320){
```

```
            putgtext(tic_label, vlabposg[i]+240, XGTEXTORG1-7, color);
         }
         else{
            putgtext(tic_label, vlabposg[i]+240, XGTEXTORG2-7, color);
         }

      }
   }
}

int select(y, manlist, n)        /* this procedure gets the mantissa */
double y, manlist[];             /* (mod 1000) of the number used to */
int n;                           /* normalize the vertical dimension */
{
   int i;
   for (i=0; y >= manlist[i] && i<n; i++);
   return(i);
}
```

Button.h

```
typedef struct button
{
   int xstart, ystart, xfinish, yfinish, cap_number;
   double cap_value;
   char *name_type, *name_but;
   struct button *next;
} button, *pbutton;

pbutton init_buttons();
void create_button(pbutton pbuttons, int color_setting);
void change_color_name(pbutton pbuttons, int color_setting);
void change_view_name(pbutton pbuttons, int color_setting);
void change_cap_value(pbutton pbuttons, int increment_mode);
void print_cap_value(pbutton pbuttons, int color_setting);
int load_cap_value(pbutton pbuttons, char prefix[]);
int save_cap_value(pbutton pbuttons, char prefix[]);
void lightup(pbutton pbuttons);
int fall_in_range(pbutton pbuttons, int xp, int yp);
double capvalue(int cap);


/*****************************************************************/
/* INIT_BUTTONS --> this creates all the buttons for use in the program */
/* It initializes each button's coordinate system and values and then   */
```

```
/* uses a linked list format to tie them together                    */
/*********************************************************************/
pbutton init_buttons()
{
   char name[15];
   int i;
   pbutton pbutton_start, pbutton_prev, pbutton_now;

   pbutton_start = pbutton_prev = NULL;

   /* create all 8 capacitor buttons */
   for(i=1;i<=8;i++)
   {
      pbutton_now = (pbutton) malloc(sizeof(button));
      if(pbutton_now == NULL)
      {
         putgtext("I've got serious problems",85,85,BLACK);
         exit(2);
      }
      if(pbutton_prev == NULL)
         pbutton_start = pbutton_now;
      else
         pbutton_prev->next = pbutton_now;

      strcpy(name, "Cap");
      pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
      strcpy(pbutton_now->name_type, name);
      sprintf(name, "%s %d", name, i); /* make cap 1, cap 2,... as names */
      pbutton_now->name_but = (char *) malloc(strlen(name) + 1);
      strcpy(pbutton_now->name_but, name);
      if(pbutton_now->name_type == NULL || pbutton_now->name_but == NULL)
      {
         putgtext("I've got more serious problems",105,85,BLACK);
         exit(2);
      }
      pbutton_now->xstart=10;
      pbutton_now->ystart=50*i-25;
      pbutton_now->xfinish=50;
      pbutton_now->yfinish=50*i-25+18;
      pbutton_prev = pbutton_now;
   }

   /* create Load button */
   strcpy(name,"Load");
   pbutton_now = (pbutton) malloc(sizeof(button));
   pbutton_prev->next = pbutton_now;
```

```
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = pbutton_now->name_type;
pbutton_now->xstart=10;
pbutton_now->ystart=421;
pbutton_now->xfinish=10+40;
pbutton_now->yfinish=421+18;
pbutton_prev = pbutton_now;

/* create Save button */
strcpy(name,"Save");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = pbutton_now->name_type;
pbutton_now->xstart=55;
pbutton_now->ystart=421;
pbutton_now->xfinish=55+35;
pbutton_now->yfinish=421+18;
pbutton_prev = pbutton_now;

/* create plot button */
strcpy(name,"Plot");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = pbutton_now->name_type;
pbutton_now->xstart=95;
pbutton_now->ystart=421;
pbutton_now->xfinish=95+35;
pbutton_now->yfinish=421+18;
pbutton_prev = pbutton_now;

/* create Reset button */
strcpy(name,"Reset");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = pbutton_now->name_type;
pbutton_now->xstart=10;
pbutton_now->ystart=421+20;
pbutton_now->xfinish=10+40;
pbutton_now->yfinish=421+20+18;
```

```
pbutton_prev = pbutton_now;

/* create Color/B&W button */
strcpy(name,"Color");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_but, name);
pbutton_now->xstart=55;
pbutton_now->ystart=421+20;
pbutton_now->xfinish=55+35;
pbutton_now->yfinish=421+20+18;
pbutton_prev = pbutton_now;

/* create Quit button */
strcpy(name,"Quit");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
pbutton_now->name_but = pbutton_now->name_type;
pbutton_now->xstart=95;
pbutton_now->ystart=421+20;
pbutton_now->xfinish=95+35;
pbutton_now->yfinish=421+20+18;
pbutton_prev = pbutton_now;

/* create View button */
strcpy(name,"View");
pbutton_now = (pbutton) malloc(sizeof(button));
pbutton_prev->next = pbutton_now;
pbutton_now->name_type = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_type, name);
sprintf(name, "%s 1-4", name, i); /* make cap 1, cap 2,... as names */
pbutton_now->name_but = (char *) malloc(strlen(name) + 1);
strcpy(pbutton_now->name_but, name);
pbutton_now->xstart=45;
pbutton_now->ystart=421+40;
pbutton_now->xfinish=65+35;
pbutton_now->yfinish=421+40+18;
pbutton_now->next=NULL;   /* Last button */
pbutton_prev = pbutton_now;

/* load default values into pbuttons */
```

```
   load_cap_value(pbutton_start,"default");
   return pbutton_start;
}
/*************************************************************************/
/* CREATE_BUTTON -->prints a button up on the screen given the coordinates */
/* of the button and the string name that should be printed on the button. */
/* It also takes a mode argument. When the mode=1, then it will create the */
/* button for a color setting. Otherwise, it creates the button for a black*/
/* and white setting.                                                      */
/*************************************************************************/
void create_button(pbutton pbuttons, int color_setting)
{
   int xs, ys, xf, yf;
   char *buttonstring;

   /* initialize button characteristics for function */
   xs = pbuttons->xstart;
   ys = pbuttons->ystart;
   xf = pbuttons->xfinish;
   yf = pbuttons->yfinish;
   buttonstring = pbuttons->name_but;

   if(color_setting==1){
      _setcolor(BLACK);
   }
   else{
      _setcolor(WHITE);
   }

   _rectangle(_GBORDER,xs,ys,xf,yf);

   if(color_setting==1){
      _setcolor(LIGHTGRAY);
   }
   else{
      _setcolor(BLACK);
   }

   _rectangle(_GFILLINTERIOR,xs+1,ys+1,xf-1,yf-1);

   if(color_setting==1){
      _setcolor(GRAY);
   }
   else{
      _setcolor(BLACK);
   }
```

```
   _moveto(xs+1,yf-1);
   _lineto(xf-1,yf-1);
   _moveto(xs+1,yf-2);
   _lineto(xf-1,yf-2);

   _moveto(xf-1,ys+1);
   _lineto(xf-1,yf-1);
   _moveto(xf-2,ys+1);
   _lineto(xf-2,yf-1);
   _moveto(xf-3,ys+1);
   _lineto(xf-3,yf-1);
   _moveto(xf-4,ys+1);
   _lineto(xf-4,yf-1);

   if(color_setting==1){
      _setcolor(WHITE);
   }
   else{
      _setcolor(BLACK);
   }

   _moveto(xs+1,ys+1);
   _lineto(xf-1,ys+1);
   _moveto(xs+1,ys+1);
   _lineto(xs+1,yf-1);

   /* place button text */
   if(color_setting==1)
   {
      putgtext(buttonstring, ys+3, xs+3,BLACK);
   }
   else
   {
      putgtext(buttonstring, ys+3, xs+3, WHITE);
   }
}
/*********************************************************/
/* CHANGE_COLOR_NAME --> changes the color name between */
/* Color and B&W depending which state its in           */
/*********************************************************/
void change_color_name(pbutton pbuttons, int color_setting)
{
   pbutton pbutton_now;
   pbutton_now = pbuttons;

   if(strcmp(pbutton_now->name_but,"Color")==0)
```

```
   {
      strcpy(pbutton_now->name_type, "B&W");
      strcpy(pbutton_now->name_but, "B&W");
   }
   else
   {
      strcpy(pbutton_now->name_type, "Color");
      strcpy(pbutton_now->name_but, "Color");
   }
   create_button(pbutton_now, color_setting);
}
/*****************************************************/
/* CHANGE_VIEW_NAME --> changes the view name between */
/* View 1-4 and View 5-8 depending which state its in */
/*****************************************************/
void change_view_name(pbutton pbuttons, int color_setting)
{
   pbutton pbutton_now;
   pbutton_now = pbuttons;

   if(strcmp(pbutton_now->name_but,"View 1-4")==0)
      strcpy(pbutton_now->name_but, "View 5-8");
   else
      strcpy(pbutton_now->name_but, "View 1-4");
   create_button(pbutton_now, color_setting);
}
/*****************************************************/
/* CHANGE_CAP_VALUE --> this increments the capacitor */
/* value by 1 or 8 depending on increment_mode        */
/*****************************************************/
void change_cap_value(pbutton pbuttons, int increment_mode)
{
   pbutton pbutton_now;
   pbutton_now = pbuttons;

   if(increment_mode==1)
      pbutton_now->cap_number = pbutton_now->cap_number + 1;
   else
      pbutton_now->cap_number = pbutton_now->cap_number + 8;
   if(pbutton_now->cap_number>=128)
      pbutton_now->cap_number = 0;
   pbutton_now->cap_value = capvalue(pbutton_now->cap_number);
}
/*************************************************************/
/* PRINT_CAP_VALUE --> prints the current cap values stored in */
/* pbuttons and prints them to the screen                      */
```

```
/*************************************************************/
void print_cap_value(pbutton pbuttons, int color_setting)
{
   char str1[10];
   int xs, ys, xf, yf;

   xs = pbuttons->xstart;
   ys = pbuttons->ystart;
   xf = pbuttons->xfinish;
   yf = pbuttons->yfinish;

   if (color_setting==0)                              /* print in B&W */
   {
      /* create description */
      _setcolor(BLACK);
      _rectangle(_GFILLINTERIOR, xs, yf+1, xf, yf+25);
      /* create text box */
      sprintf(str1, "%4.3f", pbuttons->cap_value);
      putgtext(str1, yf+2, xs, WHITE);
      sprintf(str1, "%s", "uF");
      putgtext(str1, yf+15, xs, WHITE);
   }
   else                                               /* print in color */
   {
      /* create description */
      _setcolor(WHITE);
      _rectangle(_GFILLINTERIOR, xs, yf+1, xf, yf+25);
      /* create text box */
      sprintf(str1, "%4.3f", pbuttons->cap_value);
      putgtext(str1, yf+2, xs, BLACK);
      sprintf(str1, "%s", "uF");
      putgtext(str1, yf+15, xs, BLACK);
   }
}
/*********************************************************************/
/* LOAD_CAP_VALUE --> takes in the button pointer and the string to */
/* LOAD capacitor values as.  it appends suffix .cap                 */
/*********************************************************************/
int load_cap_value(pbutton pbuttons, char prefix[])
{
   FILE *cap;
   char str1[30];
   int var1,invalid=0;
   pbutton pbutton_now;
   pbutton_now = pbuttons;
```

```
    sprintf(str1, "%s.cap",prefix);
    if((cap = fopen(str1,"r")) == NULL)
    {
       invalid = 1;
       sprintf(str1,"%s does not exist",str1);
       _setcolor(WHITE);
       _rectangle(_GFILLINTERIOR, 100, 100, 300, 300);
       putgtext(str1, 200, 110, BLACK);
       putgtext("Hit any key to continue", 130, 110, BLACK);
    }

    if(invalid)
       while(!kbhit()){}
    else
    {
       while(strcmp(pbutton_now->name_type,"Cap")==0)
       {
          int j;

          if((j = fscanf(cap, "%s", str1))==1)
             var1 = atoi(str1);
          else
             var1 = 0;
          pbutton_now->cap_number = var1;
          pbutton_now->cap_value = capvalue(pbutton_now->cap_number);
          pbutton_now = pbutton_now->next;
       }
       fclose(cap);
    }
    return invalid;
}
/*********************************************************************/
/* SAVE_CAP_VALUE --> takes in the button pointer and the string to */
/* save capacitor values as.  it appends suffix .cap                 */
/*********************************************************************/
int save_cap_value(pbutton pbuttons, char prefix[])
{
    FILE *cap;
    char str1[30];
    int var1, invalid=0;
    pbutton pbutton_now;
    pbutton_now = pbuttons;

    sprintf(str1, "%s.cap",prefix);
    if((cap = fopen(str1, "w"))==NULL)
    {
```

```
      invalid = 1;
      sprintf(str1,"%s does not exist",str1);
      _setcolor(WHITE);
      _rectangle(_GFILLINTERIOR, 100, 100, 300, 300);
      putgtext(str1, 200, 110, BLACK);
      putgtext("Hit any key to continue", 130, 110, BLACK);
   }

   if(invalid)
      while(!kbhit()){}
   else
   {
      while(strcmp(pbutton_now->name_type,"Cap")==0)
      {
         fprintf(cap, "%d\n", pbutton_now->cap_number);
         pbutton_now = pbutton_now->next;
      }
      fclose(cap);
   }
   return invalid;
}
/*************************************************************************/
/* This will "lightup" the button. It draws a yellow box around the button */
/* when the button is clicked, to reenforce to the user that he/she clicked*/
/* on the appropriate button.                                              */
/*************************************************************************/
void lightup(pbutton pbuttons)
{
   int xs, ys, xf, yf;

   xs = pbuttons->xstart;
   ys = pbuttons->ystart;
   xf = pbuttons->xfinish;
   yf = pbuttons->yfinish;

   _setcolor(YELLOW);
   _rectangle(_GBORDER, xs, ys, xf, yf);
   release_button();
   _setcolor(BLACK);
   _rectangle(_GBORDER, xs, ys, xf, yf);
   _setcolor(WHITE);
   _moveto(xs+1, ys+1);
   _lineto(xf-1, ys+1);
   _moveto(xs+1, ys+1);
   _lineto(xs+1, yf-1);
}
```

```
/*****************************************************************/
/* FALL_IN_RANGE --> tells whether the x and y coordinates of the */
/* mouse are within the current buttons borders                   */
/*****************************************************************/
int fall_in_range(pbutton pbuttons, int xp, int yp)
{
  if(xp >= pbuttons->xstart && xp <= pbuttons->xfinish &&
     yp >= pbuttons->ystart && yp <= pbuttons->yfinish) {return 1;}
  else
   return 0;
}
/*****************************************************************/
/* CAPVALUE --> takes in an integer and converts that integer */
/* To the appropriate capacitor value                         */
/*****************************************************************/
double capvalue(int cap)
{
   int cap2 = cap;
   int i;
   double caps[7] = {0.539,0.985,2.89,4.95,14.54,25.4,52.3};
   double capval = 0;

   for(i=0;i<7;i++)
   {
      capval+=caps[i]*(cap&1!=0);
      cap>>= 1;
   }
   return(capval);
}
```

Loadsave.h

```
void copy_files(char *file_start, char *file_end)
{
   FILE *output;
   int i;
   char str1[100], str2[100], command[100];

   output = fopen("bat.bat","w");
   fprintf(output,"%s","@echo off\n");
   /* send to system */
   sprintf(str1, "%s.lma", file_start);
   sprintf(str2, "%s.lma", file_end);
   sprintf(command,"copy %s %s > junk.foo",str1,str2);
   sprintf(str1, "%s.fma", file_start);
   sprintf(str2, "%s.fma", file_end);
```

```
   sprintf(command,"copy %s %s > junk.foo",str1,str2);
   sprintf(str1, "%s.rma", file_start);
   sprintf(str2, "%s.rma", file_end);
   sprintf(command,"copy %s %s > junk.foo",str1,str2);
   sprintf(str1, "%s.cap", file_start);
   sprintf(str2, "%s.cap", file_end);
   sprintf(command,"copy %s %s > junk.foo",str1,str2);
   fprintf(output,"%s\n",command);

   for(i=1;i<=8;i++)
   {
      sprintf(str1, "%s%d.mat", file_start,i);
      sprintf(str2, "%s%d.mat", file_end,i);
      sprintf(command,"copy %s %s > junk.foo",str1,str2);
      fprintf(output,"%s\n",command);
      sprintf(str1, "%s%df.mat", file_start, i);
      sprintf(str2, "%s%df.mat", file_end, i);
      sprintf(command,"copy %s %s > junk.foo",str1,str2);
      fprintf(output,"%s\n",command);
   }
   fprintf(output,"%s","cls\n");
   fclose(output);
   system("bat");
}

void load_sequence()
{
   copy_files("svdat","data");                    /* restore corresponding data files */
}

void save_sequence()
{
/*   char a, str1[20], str2[2];
   int i, bool=0, length=0, tot_length=0;

   _setcolor(WHITE);
   _rectangle(_GFILLINTERIOR, 190, 190, 370, 270);
   putgtext("Enter file prefix to save as", 200, 200, BLACK);
   putgtext("Hit <CR> to continue", 220, 210, BLACK);
   for(i=0;a!='\r' && i<20;i++)
   {
      while(!bool)
      {
         bool=kbhit();
         if(bool)
         {
            a = getch();
            if ((a==(char)8 || a==(char)127) && i>0)
            {
               str1[i]='\0';
               i=i-1;
            }
```

```
            else
            {
               str1[i] = a;
               sprintf(str2,"%c",a);
               length = _getgtextextent(str2);
               tot_length = tot_length+length;
               putgtext(str2, 250, 200 + tot_length, BLACK);

            }
         }
      }
      bool=0;
   }
*/
   copy_files("data","svdat");              /* save corresponding 16 data files */
}
```

Globfunc.h

```
/*************************/
/*************************/
/*   Universal functions  */
/*************************/
/*************************/
void putgtext(char *work, int r, int c, int col);
void paint(int color_setting);

/*****************************************************************************/
/* Writes text on the screen using graphical fonts instead of regular fonts */
/*****************************************************************************/
void putgtext(char *work, int r,int c, int col)
{
_moveto(c,r);
_setcolor(col);
_outgtext(work);
}

/* Paints the background and draws the title and title bar at the top of  */
/* the screen. color_setting=0 prints black and white, color_setting=1 prints in color.     */
void paint(int color_setting)
{
   if(color_setting==0)   /* Black and White background */
   {
      _setcolor(BLACK);
      _rectangle(_GFILLINTERIOR, 0, 0, 640, 480);
      _setcolor(WHITE);
      _rectangle(_GBORDER, 0, 0, 639, 479);
      _setcolor(WHITE);
      _rectangle(_GBORDER, 0, 0, 639, 13);
      putgtext("DESIRE", 1, 295, WHITE);
   }
```

```
   else if(color_setting==1)  /* Color background */
   {
      _setcolor(DARKBLUE);
      _rectangle(_GFILLINTERIOR, 0, 0, 640, 480);
      _setcolor(LIGHTGRAY);
      _rectangle(_GFILLINTERIOR, 0, 0, 640, 13);
      putgtext("DESIRE", 1, 295, BLACK);
   }
}

/* This function will take a string, and put it up on the screen at some   */
/* location. This location is specified by not only the column and row     */
/* numbers, but also from the mode argument. If mode=0, then the text will */
/* be displayed in the console/contacts window. Otherwise, it will be      */
/* displayed in the time window. (note the different use of mode. In       */
/* previous functions, mode was used to specify the color scheme, whereas  */
/* here, it is used to specify the location of the text.)                  */
readandprint(char *text, int *column, int *row, int color, int color_setting)
{
  int i=0, length=0, linenumber=0, columnnumber=0;
  char string[100], newtext[100];

  columnnumber=*column;
  linenumber=*row;

  length=strlen(text);

  for(i=0; i<length; i++)
  {
     if(text[i]=='=')
     {
        text[i]=' ';
     }
     else{}
  }

  if(color_setting==0)
  {
     if(linenumber>22 | length>20){}
     else
     {
        putgtext(text, 225+10*linenumber, 15, WHITE);
     }
  }
  else
  {
     if(linenumber>0 | length>20){}
     else
     {
        putgtext(text, 50+10*linenumber, 15, color);
     }
  }
```

```
  *row=linenumber+1;
}
```

Globvar.h

```
#define NUL
#ifdef MAIN
#define EXTERN NUL
#else
#define EXTERN extern
#endif


EXTERN TRANMOD tran1,tran2,tran3,tran4,tran5,tran6,tran7,tran8;
EXTERN PTRANMOD ptrans[8];
EXTERN int mousepresent;
EXTERN int finished;
EXTERN int sc;
EXTERN char fname[30], savename[30];
EXTERN double pan[3000], pbn[3000];
EXTERN int ch1[3000], ch2[3000], ch3[3000], ch4[3000];
EXTERN int ch1f[3000], ch2f[3000], ch3f[3000], ch4f[3000];

/*  "Expert" information from the .tab files */
EXTERN double wireinfo[AMPS][MILS];
/*[MILS=0] gives the maximum amp rating         */
        /*[MILS=1] gives the wire gauge in circular mils*/
        /*[MILS=2] gives the number of wires            */

EXTERN double motorinfo[POW][INFO];
        /*[INFO=0] gives the motor's rated shaft power  */
        /*[INFO=1] gives fractional efficiency          */
        /*[INFO=2] gives power factor in degrees        */

EXTERN double legalmils[100];
EXTERN double trip[50];
EXTERN double conduit[AMPS];
EXTERN double condtypes[50];
EXTERN double def_pfact;
EXTERN double AWG;
EXTERN int MAXSW;
EXTERN int UNBALANCE;
EXTERN int OVERCURR;
EXTERN int currentpanel;
EXTERN int num_of_devices;
EXTERN int num_of_panels;
EXTERN int num_of_breakers;
EXTERN int num_of_conds;
EXTERN int num_of_fileblanks;
EXTERN int num_of_motors;
EXTERN int num_of_wires;
```

```
EXTERN int num_of_ltypes;
EXTERN int num_of_mtypes;
EXTERN int num_of_trans;
EXTERN int busflag;
EXTERN int transflag;
EXTERN int powerflag;
/*0 is apparent pow,S;1 means real power,P;*/
EXTERN char AWGSTRING[15];
EXTERN char *netname;
EXTERN char *netstor;
EXTERN char *gage[AMPS];
EXTERN char *legalgages[100];
EXTERN char *loads[NUM_OF_LOADS];
EXTERN char *devices[101][2];
EXTERN char ltypes[50][30];
EXTERN char mtypes[50][30];
EXTERN char origpath[100];
EXTERN char workpath[100];
EXTERN int blanks[100];
EXTERN int network[MAX_PANELS][2];
/*[0] gives the # of "children" panels   */
        /*[1] gives the # of the parent panel    */

EXTERN char *netstring[MAX_PANELS][2];
    /*[0] gives the title of the panel          */
       /*[1] gives the panel's filename            */

EXTERN char *displaynet[11];
EXTERN int  framecolor;
EXTERN int  highlightcolor;
EXTERN long  highlightbkcolor;
EXTERN int  textcolor;
EXTERN long backcolor;
EXTERN int  xscale;
```

Globcon.h

```
/*  Load p#s will always give the apparent phase power */

#include<dos.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<malloc.h>
#include<string.h>
#include<direct.h>
#include<graph.h>
#include<bios.h>
#include<conio.h>

#define DOUBLE 1
```

```
#define INTEGER 2
#define STRING 3
#define CURS 219
#define SPACE 32
#define AMPS  80
#define MILS  3
#define POW 50
#define INFO 3
#define ROOT3 1.7321
#define INFINITY 1.2e304
#define NUM_OF_LOADS 8
#define MAX_PANELS 100
#define MAX_SWITCHES 50
#define CONVERT_TO_RADIANS (double) .017453
#define HP_TO_WATTS (double) 746
#define NUL
#define MAX_ABCISSA 460
#define PI 3.14159

#define BLACK 0
#define DARKBLUE 1 /* For convenience: color definitions */
#define DARKGREEN 2
#define TEAL 3
#define DARKRED 4
#define PURPLE 5
#define ORANGE 6
#define LIGHTGRAY 7
#define GRAY 8
#define BLUE 9
#define GREEN 10
#define LIGHTBLUE 11
#define RED 12
#define LIGHTPURPLE 13
#define YELLOW 14
#define WHITE 15

/* Coordinates of the origins of the
four graphs, in VGA coordinates */
#define XORG1 180
#define YORG1 220
#define XORG2 439
#define YORG2 220
#define XORG3 180
#define YORG3 460
#define XORG4 439
#define YORG4 460

#define XSIZE 200 /* horizontal width, in pixels */
#define YSIZE 180 /* vertical height, in pixels */

#define TICLENGTH 5 /* for axis markings, in pixels */
```

```
#define XTEXTORG1 18 /* the column number where
horizontal text starts */
#define XTEXTORG2 50
#define XGTEXTORG1 150
#define XGTEXTORG2 410

#define TLABPOSG 2
#define TLABPOS 3 /* vertical line position of
top axis label */
#define Q3POS 6 /* third quartile of middle axis
label-not actually displayed
on the screen */
#define MLABPOS 9 /* vertical line position
of middle axis label */
#define Q1POS 12 /* first quartile position-not
actually displayed */
#define BLABPOS 14 /* vertical line position of
bottom axis label */

#define NVTICLAB 5
#define TOPLABCTR1 36   /* horizontal column position */
#define TOPLABCTR2 68
#define LABROW 0 /* vertical position of title row */
#define TITLENGTH 40 /* maximum printable title length */

#define PRINTCOL1 16 /* column positions of abscissa labels */
#define PRINTCOL2 49

#define PRINTCOL1G 135
#define PRINTCOL2G 395

#define ABLENGTH 12 /* maximum length of abscissa label */

#define ABSL 12 /* maximum length of abscissa text */

#define ORDROW1 0 /* rows where ordinate labels go */
#define ORDROW2 15
#define ORDROW3 30
#define ORDROW1G 0
#define ORDROW2G 225
#define ORDROW3G 465

#define MANLISTSIZE 13 /* number of entries in the mantissa list */

#define XLABSTART1 23 /* start locations of time labels */
#define XLABSTART2 55
#define XLABEND1 44 /* end locations of time labels */
#define XLABEND2 75
#define XLABSTART1G 180
#define XLABSTART2G 440
#define XLABEND1G 360
#define XLABEND2G 610
```

```
#define TICSPEROCTAVE 8 /* on the x-axis */

#define TEMPSIZE 60
#define MAXN 50
#define MAXTEMPS 50

typedef double *pd;
typedef pd vect;
typedef vect *matrix;
typedef matrix *pmatrix;

typedef struct {double t0,t1,t2,t3,t4,p1,p2,p3,sp1,sp2,sp3;
                double d1,d2,d3,st1,st2,st3;
char name[10];
int num_of_breaks;} TRANMOD, *PTRANMOD;
typedef struct new_load
    { int ltype; double pa,qa,pb,qb,pc,qc,
/*ltype is the load type*/
     spa,sqa,spb,sqb,spc,sqc,
/*like vapor lamp, etc. */
     pfact,trip;} NEW_LOAD;
typedef struct new_motor
{ int mtype; double shaftpow,
 p,q,sp,sq,
 mpfact,mtrip,effic;} NEW_MOT;
typedef struct receptacle
                { double number,rtrip,rp;} REC;
typedef struct spares
                { double strip;} SPAR;
typedef struct space
                { int nothing; } SPAC;
typedef struct pan
                { char *pfile; double ppa,ppb,ppc,pqa,pqb,pqc;
                  double ptrip,pvolts;} PAN;
typedef struct trans
                { char *tfile; int windtype;
                  double secvolts,primvolts,ttrip,kVa,
                         tpa,tpb,tpc,tqa,tqb,tqc;} TRANS;
typedef struct bus
                { char *bfile; double bpa,bpb,bpc,bqa,bqb,bqc;
                  double btrip,bvolts;} BUS;
typedef struct new_switches
                { char *swname; int kind,wiretype;
/* wiretype is 3,4,5wire*/
          double wire,wirelen,numwires;
                  double conduit_diameter,phase; char *wirestring;
  TRANMOD TM; NEW_LOAD L; NEW_MOT M; REC R; SPAR S;
                  SPAC N; PAN P; TRANS T; BUS B;} SWITCHES;
typedef struct panel
                {
                  char *title;
                  char *filename;
```

```
    char *parenttitle;
    char *parentfile;
    char *update;
    double voltage;
    double breaker;
    double Pa,Pb,Pc;
    double Qa,Qb,Qc;
    int    dirty;
    int    num_of_switches;
    struct new_switches *switchvect[MAX_SWITCHES];
   } PANEL, *PPANEL;


typedef struct {double sa, psia, sb, psib, sc, psic, reserved, trip;}
c_load, *p_c_load;
```

Utils.c

```
#include "globcon.h"
#include "globvar.h"

putrc(r,c)
int r,c;
{
_settextposition(r,c);
}


initmouse()
{
int c,xp,yp,m1,m2,m3,m4;
int ox,oy,stat;
int flag = 0;

m1 = m2 = m3 = m4 = 0;
cmousel(&m1,&m2,&m3,&m4);
if (m1 == 0) mousepresent = 0;
else {
mousepresent = 1;
m1 = 15; m2 = 2; m3 = 4; m4 = 8;
cmousel(&m1,&m2,&m3,&m4);
m1 = 4; m2 = 6; m3 = 320; m4 = 160;
cmousel(&m1,&m2,&m3,&m4);
m1 = 1; m2 = 6; m3 = 320; m4 = 160;
cmousel(&m1,&m2,&m3,&m4);
  }
}

clearmouse()
{
int c,xp,yp;
int stat;
```

```
if (mousepresent) c = mouseint(2,0,0,0,&xp,&yp,&stat);
else return(0);
}


makehand()
{
 static int cursor[32] =
 {
 0xE1FF, 0xE1FF, 0xE1FF,
 0xE1FF, 0xE1FF, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0x1E00, 0x1200, 0x1200, 0x1200,
0x1200, 0x13FF, 0x1249, 0x1249,
0xF249, 0x9001, 0x9001, 0x9001,
0x8001, 0x8001, 0x8001, 0xFFFF

 };
 int *cptr=cursor;
 int m1, m2, m3, m4;

 m1 = 9;
 m2 = 0;
 m3 = 0;
 cmousel(&m1, &m2, &m3, &cptr);
 m1 = 1;
 cmousel(&m1, &m2, &m3, &m4);
}


makecross()
{
 static int cursor[32] =
 {
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff,0xffff,0xffff,
0, 0, 0, 0x0380, 0x0c60, 0x1010,
0x2108, 0x2388, 0x2108, 0x1010,
0x0c60, 0x0380, 0x0, 0x0, 0x0, 0x0


 };
 int *cptr=cursor;
 int m1, m2, m3, m4;

 m1 = 9;
 m2 = 8;
 m3 = 8;
 cmousel(&m1, &m2, &m3, &cptr);
 m1 = 1;
```

```
 cmousel(&m1, &m2, &m3, &m4);
}


makequest()
{
 static int cursor[32] =
 {
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff, 0xffff, 0xffff,
0xffff, 0xffff,0xffff,0xffff,
0, 0x3FFC, 0x2004, 0x2004, 0x0004,
0x0004, 0x0004,0x0004, 0x00FC,
0x0080,0x0080,0x0080,0x0080,
0x0080, 0, 0x0080


 };
 int *cptr=cursor;
 int m1, m2, m3, m4;

 m1 = 9;
 m2 = 0;
 m3 = 0;
 cmousel(&m1, &m2, &m3, &cptr);
 m1 = 1;
 cmousel(&m1, &m2, &m3, &m4);
}


hide_mouse()
{
int m1,m2,m3,m4;

if (mousepresent) {
 m1 = 2; m2 = 0; m3 = 0; m4 = 0;
 cmousel(&m1,&m2,&m3,&m4);}
}

show_mouse()
{
int m1,m2,m3,m4;

if (mousepresent) {
m1 = 1; m2 = 0; m3 = 0; m4 = 0;
cmousel(&m1,&m2,&m3,&m4);}
}

release_button()
{
int flag = 0;
```

```
int x = 0,y = 0,b = 0;

 while (flag == 0)
  {
if (mousepresent) {getmouse(&x,&y,&b);}
else flag = 1;
if (b == 0) flag = 1;
else flag = 0;
  }
 flag = 0;
}



press()
{
char c;
int x,y,b,safeb;
int flag = 0;
int flag2 = 0;

x = y = b = 0;
while (flag == 0)
 {
getmouse(&x,&y,&b);
c = kbhit();
if (c != 0 | b!= 0) {
flag = 1;
if (c != 0) c = getch();
else if (b != 0) {
safeb = b;
while (flag2 == 0) {
getmouse(&x,&y,&b);
if (b == 0) flag2 = 1;}
}
 }
 }
 if (flag2 == 1) return(safeb);
}


getmouse(x,y,b)
int *x,*y,*b;
{
  int stat;

  if (mousepresent) {
*b = mouseint(3,0,0,0,x,y,&stat);}
  else *b = *x = *y = 0;
}
```

```
initmouse2()
{
int c,xp,yp,m1,m2,m3,m4;
int ox,oy,stat;
int flag = 0;

m1 = m2 = m3 = m4 = 0;
cmousel(&m1,&m2,&m3,&m4);
if (m1 == 0) mousepresent = 0;
else {
mousepresent = 1;
makehand();
m1 = 15; m2 = 2; m3 = 4; m4 = 8;
cmousel(&m1,&m2,&m3,&m4);
m1 = 4; m2 = 6; m3 = 320; m4 = 160;
cmousel(&m1,&m2,&m3,&m4);
m1 = 1; m2 = 6; m3 = 320; m4 = 160;
cmousel(&m1,&m2,&m3,&m4);
  }
}


int mouseint(m1,m2,m3,m4,xp,yp,stat)
int m1,m2,m3,m4,*xp,*yp,*stat;
{ extern int sc;
  cmousel(&m1,&m2,&m3,&m4);
  *xp = m3;
  *yp = m4;
  return(m2);
}


load_templates(ptemplates,qtemplates,rows,cols,idnum)
matrix ptemplates, qtemplates;
int *rows,*cols,*idnum;
{
int i,j, r, c;
double num;
FILE *tf, *fopen();

if ((tf = fopen("temps.dat","r")) == 0) {
printf ("Unable to open temps.dat\n");
exit(0);
}

fscanf (tf,"%d",&c); *cols = c;
fscanf (tf,"%d",idnum);
r = TEMPSIZE; *rows = TEMPSIZE;

for (i = 0; i < c; i++) {
for (j = 0; j < r; j++) {
fscanf (tf,"%lf",&num);
```

```
ptemplates[j][i] = num;
}
for (j = 0; j < r; j++) {
fscanf (tf,"%lf",&num);
qtemplates[j][i] = num;
}
}
}


matoutput(Amat,r,c)
matrix Amat;
int r,c;
{
 int i,j;

 printf ("\n");
 for (i = 0; i < r; i++){
for (j = 0; j < c; j++){
printf(" %g ",Amat[i][j]);
}
printf ("\n");
 }
printf ("\n");
}


fmatoutput(Amat,r,c)
matrix Amat;
int r,c;
{
 char work[100];
 int i,j;
 FILE *af, *fopen();

 af = fopen(savename,"w");

 for (i = 0; i < r; i++){
for (j = 0; j < c; j++){
fprintf(af," %g ",Amat[i][j]);
}
fprintf (af,"\n");
 }
fprintf (af,"\n");

fclose(af);
}


transpose_matoutput(Amat,r,c)
matrix Amat;
int r,c;
```

```
{
 int i,j;

 printf ("\n");
 for (i = 0; i < c; i++){
for (j = 0; j < r; j++){
printf(" %g ",Amat[j][i]);
}
printf ("\n");
 }
printf ("\n");
}


ftranspose_matoutput(Amat,r, c)
matrix Amat;
int r,c;
{
 char work[100];
 int i,j;
 FILE *af, *fopen();

 printf ("File name ==> ");
 scanf("%s",work);
 af = fopen(work,"w");

 printf ("\n");
 for (i = 0; i < c; i++){
for (j = 0; j < r; j++){
fprintf(af," %g ",Amat[j][i]);
}
fprintf (af,"\n");
 }
fprintf (af,"\n");
fclose(af);
}

void nrerror(error_text)
char error_text[50];
{
 printf ("Error in Matrix Inversion\n");
 printf ("%s\n",error_text);
}


double *vector(nl,nh)
int nl,nh;
{
double *v;

v = (double *) malloc((unsigned) (nh - nl + 1)*sizeof(double));
if (!v) nrerror("allocation failure in vector()");
```

```
return v-nl;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
free((char *) (v+nl));
}


ludcmp(a,n,indx,d)
int n,indx[MAXN];
double *d;
matrix a;
{
int i,imax,j,k;
double big,dum,sum,temp;
double *vv,*vector();
double tiny;
void nrerror(),free_vector();

tiny = 1e-20;
vv = vector(1,n);
*d = 1.0;

for (i = 1; i <= n; i++) {
big = 0.0;
for (j = 1; j <= n; j++)
if ((temp = fabs(a[i][j])) > big) big = temp;
if (big == 0) nrerror("Singular matrix in LUDCMP");
vv[i] = 1.0/big;
}
for (j = 1; j <= n; j++) {
for (i = 1; i < j; i++) {
sum = a[i][j];
for (k = 1; k < i; k++) sum -= a[i][k]*a[k][j];
a[i][j] = sum;
}
big = 0.0;
for (i = j; i<=n; i++) {
sum = a[i][j];
for (k = 1; k < j; k++)
 sum -= a[i][k]*a[k][j];
  a[i][j] = sum;
if ((dum=vv[i]*fabs(sum)) >= big) {
big = dum;
imax = i;
}
 }
if (j != imax) {
```

```
for (k = 1; k<=n; k++) {
dum = a[imax][k];
a[imax][k] = a[j][k];
a[j][k] = dum;
}
*d = -(*d);
vv[imax] = vv[j];
}
indx[j] = imax;
if (a[j][j] == 0) a[j][j] = tiny;
if (j != n) {
dum = 1.0/a[j][j];
for (i = j+1; i <=n; i++) a[i][j] *= dum;
}
 }
 free_vector(vv,1,n);
}


lubksb(a,n,indx,b)
matrix a;
double *b;
int n,*indx;
{
int i,ii=0,ip,j;
double sum;

for (i = 1; i <= n; i++) {
ip = indx[i];
sum = b[ip];
b[ip] = b[i];
if (ii)
for (j = ii; j <= i-1; j++) sum -= a[i][j]*b[j];
  else if (sum) ii = i;
b[i] = sum;
}
for (i = n; i >= 1; i--) {
sum = b[i];
for (j = i+1; j <=n; j++) sum -= a[i][j]*b[j];
b[i] = sum/a[i][i];
 }
}


matinvert(n,Ac,y)
int n;
matrix Ac, y;
{
double d,col[MAXN];
matrix Amat;
int i,j,indx[MAXN];
```

```
AllocMatrix(&Amat,MAXN,MAXN);


for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
Amat[i+1][j+1] = Ac[i][j];
ludcmp(Amat,n,indx,&d);
for (j = 1; j <= n; j++) {
for (i=1; i<=n; i++) col[i] = 0.0;
col[j] = 1.0;
lubksb(Amat,n,indx,col);
for (i=1; i<=n; i++) y[i][j] = col[i];
 }
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
Amat[i][j] = y[i+1][j+1];
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
y[i][j] = Amat[i][j];

FreeMatrix(Amat,MAXN);
}



transpose(source,target,r,c)
matrix source, target;
int r,c;
{
int i,j;

for (i = 0; i < r; i ++) {
for (j = 0; j < c; j++) {
target[j][i]  = source[i][j];
}
}
}


matmult(a,b,cm,r,c)
matrix a, b, cm;
int r,c;
{
int arow, bcol, inner;
double sum;

for (arow = 0; arow < c; arow++) {
for (bcol = 0; bcol < c; bcol++) {
sum = 0.0;
for (inner = 0; inner < r; inner++)
sum += a[arow][inner]*b[inner][bcol];
cm[arow][bcol] = sum;
```

```
}
}
}


matmult2(a,b,cm,r,c)
matrix a, b, cm;
int r,c;
{
int arow, bcol, inner;
double sum;

for (arow = 0; arow < c; arow++) {
for (bcol = 0; bcol < r; bcol++) {
sum = 0.0;
for (inner = 0; inner < c; inner++)
sum += a[arow][inner]*b[inner][bcol];
cm[arow][bcol] = sum;
}
}
}


compute_tapweights(ptemplates,qtemplates,Gp,Gq,r,c)
matrix ptemplates, qtemplates, Gp, Gq;
int r,c;
{
 int i,j;
 matrix tempt, invG, copyM;

 AllocMatrix(&tempt,MAXTEMPS,TEMPSIZE);
 AllocMatrix(&invG,MAXTEMPS,TEMPSIZE);
 AllocMatrix(&copyM,MAXN,MAXN);

 /* compute orthonormal tap weights for real part */

 transpose(ptemplates,tempt,r,c);
 matmult(tempt,ptemplates,Gp,r,c);
 for (i = 0; i < c; i++)
 for (j = 0; j < c; j++)
 copyM[i][j] = Gp[i][j];
 matinvert(c,copyM,copyM);
 for (i = 0; i < c; i++)
 for (j = 0; j < c; j++)
 invG[i][j] = copyM[i][j];
 matmult2(invG,tempt,Gp,r,c);

 /* compute orthonormal tap weights for reactive part */

 transpose(qtemplates,tempt,r,c);
 matmult(tempt,qtemplates,Gq,r,c);
 for (i = 0; i < c; i++)
```

```
 for (j = 0; j < c; j++)
 copyM[i][j] = Gq[i][j];
 matinvert(c,copyM,copyM);
 for (i = 0; i < c; i++)
 for (j = 0; j < c; j++)
 invG[i][j] = copyM[i][j];
 matmult2(invG,tempt,Gq,r,c);

 FreeMatrix(tempt,MAXTEMPS);
 FreeMatrix(invG,MAXTEMPS);
 FreeMatrix(copyM,MAXN);
}

AllocMatrix(a,row,col)
pmatrix a;
int row,col;
{
  int j;
  matrix mat;

  mat = (matrix) calloc(row,sizeof(pd));
  if (mat == NULL) {printf ("Sorry, no memory\n"); exit(0);}
  for (j = 0; j < row; j++) {
  if((mat[j] = (vect) calloc(col,sizeof(double))) == NULL) {
  printf ("Sorry, no memory\n");
 exit(0);
  }
  }
  *a = mat;
}


FreeMatrix(a,row)
matrix a;
int row;
{
  int j;

  for (j = 0; j < row; j++) free(a[j]);
  free(a);
}
```

C.2.3 Rob471.c

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

FILE *capfile;
```

```
void go(int res, int cap2, int len, double capval, char *work);
double capvalue(int cap2);

main()
{
char w1[15],temp[5],c1[5];
char *work;
double capval;

int res = 47;
int len = 1000;
int cap, cap2,i;

capfile = fopen("data.cap","r");

for(i=1;i<=8;i++)
{
fgets(temp, 5, capfile);

cap = atoi(temp);

cap2 = cap;
capval=capvalue(cap);

sprintf(work,"idata%d.mat",i);
go (res, cap2, len, capval, work);

}
fclose(capfile);
}



double capvalue(int cap)
{
int cap2 = cap;
int i;
double caps[7] = {0.539,0.985,2.89,4.95,14.54,25.4,52.3};
double capval = 0;

for(i = 0; i < 7; i++) {
capval += caps[i] * ( cap & 1 != 0 );
cap >>= 1;
}

return( capval );
}


void go(int res, int cap2, int len, double capval, char *work)
{
```

```
time_t start = time((time_t *)NULL);


int i,j,pdelay,constv, dl, dh,out,hold;

int highword0, lowword0, highword1, lowword1;
double temp, rtime, freq, scale, current, maxval;
FILE *pf, *fopen();



pf = fopen(work,"w");

constv = 22;
pdelay = 18480;
freq = 4435000;
scale = .00244;

highword0 = constv/256;          /* high word for cntr 0 of 8254*/
lowword0 = constv - highword0;  /* low  "   "   "   "   "   "  */

highword1 = pdelay/256;      /* high word for cntr 1 of 8254*/
lowword1 = pdelay-highword1; /* low "   "   "   "   "   "   */


outp(0x304,cap2);    /* sets the capacitor relays */

while (time((time_t *)NULL) < 2+start);

outp(0x308,2); /* gate low*/

outp(0x303,54); /* turn off 8254 */
outp(0x300,0);

outp(0x303,112);
outp(0x301,0);

outp(0x314,0);    /* reset counters*/


outp(0x308,0);    /* set CTL = 0 to get to initial state*/
outp(0x308,2);
outp(0x314,0);    /* reset counters*/

outp(0x303,54);   /* sets the constv rate of the system*/
outp(0x300,lowword0);
outp(0x300,highword0);

outp(0x303,112);   /* sets one shot to control
the 3phase switch.*/
outp(0x301,lowword1);
outp(0x301,highwo~d1);
```

```
outp(0x308,18);
outp(0x308,50);         /* tell brd to begin sampling and it
will continue until 32K worth of
data is stored by making gate high on
the 8254.*/

fprintf(pf,"%f  999\n", capval);
maxval = 0;

while (time((time_t *)NULL) < 5+start);

outp(0x308,2); /* gate low*/

outp(0x303,54); /* turn off 8254 */
outp(0x300,0);

outp(0x303,112);
outp(0x301,0);

outp(0x314,0);    /* reset counters*/

for(i=1;i<850; i++)
{
outp(0x30c,0);
}

for(i=850; i<850+len; i++)
{

dl = inp(0x318);
dh = inp(0x31c);

/* This subroutine combines the low
  word and the high byte of data
* taken from the RAM into its decimal
  value (ranging from -2048
* to 2047). This value, called out, is
  then scaled by 4.88mV per division
* (to calculate the voltage into the 7874)
  then divided by the value of the
* resistor value to find the current
  flowing thru the capacitor.
*/


if(dl < 0)
dl = dl + 256;

if(dh < 8)
{
out = 256 * dh + dl;
```

```
}
else
{
out = 256 * (dh - 8) + dl - 2048;
}
current = 1000*(((double) out) *
((double) scale)) / res;

if(out>maxval) maxval=out;
if((-1*out)>maxval) maxval = (-1*out);

rtime = (((double) i-850) * ((double) constv))/freq;
fprintf(pf,"%f  %f  \n",current,rtime);
outp(0x30c,0);
}
 fclose(pf);
 }
```

C.2.4 Matlab Script/Function Files

Robir.m

```
% This sequence determines the R and L of a line automatically and
% places the values in a matrix.

% echo on

% !mc47 1 data1.mat 1000
% !mc47 2 data2.mat 1000
% !mc47 3 data3.mat 1000
% !mc47 4 data4.mat 1000
% !mc47 5 data5.mat 1000
% !mc47 6 data6.mat 1000
% !mc47 7 data7.mat 1000
% !mc47 8 data8.mat 1000

echo off;

global t Data C filename filenumbers timename dataname
ename fmat rmat lmat caps
global t1 t2 t3 t4 t5 t6 t7 t8
global d1 d2 d3 d4 d5 d6 d7 d8
global e1 e2 e3 e4 e5 e6 e7 e8


filename=['idata1';'idata2';'idata3';'idata4';
'idata5';'idata6';'idata7';'idata8'];
filenumbers = [1, 2, 3, 4, 5, 6, 7, 8];
timename = ['t1';'t2';'t3';'t4';'t5';'t6';'t7';'t8'];
dataname = ['d1';'d2';'d3';'d4';'d5';'d6';'d7';'d8'];
estname = ['est1';'est2';'est3';'est4';
'est5';'est6';'est7';'est8'];
```

```
ename = ['e1';'e2';'e3';'e4';'e5';'e6';'e7';'e8'];



for counter=1:8

        file = filename(counter,:);
        fn = filenumbers(counter);
        eval(['load ',file]);
        eval(['C = ',file,'(1,1)*1e-6;']);
        caps(counter)=C;
        eval(['t = ',file,'(2:length(',file,'(:,2)),2);']);
        eval(['Data = ',file,'(2:length(t)+1,1);']);
        w = peak(Data);
        t= t(w:length(t))- 5e-6*(w-1);
        Data  = Data(w:length(Data));
        eval([dataname(counter,:),' = Data;']);
        eval([timename(counter,:),' = t;']);
        est = fmins('decay4',[2,.0013,5,-1]',1e-3);
        e = round(10*dec5(est));
% round after fit so that rmat, lmat not effected
        eval([dataname(counter,:),' =
round(10*',dataname(counter,:),');'])
% factor of 10 so that it can be rounded without losing much precision
        eval([ename(counter,:),' = round(10*dec5(est));']);
        eval([estname(counter,:),' = est;']);
        rmat(counter) = est(1);
        lmat(counter) = est(2);
        fmat(counter) = 1/(2*pi*sqrt(lmat(counter)*caps(counter)));
%       rplotf(counter);
end

save odata1.mat d1 /ascii;
save odata2.mat d2 /ascii;
save odata3.mat d3 /ascii;
save odata4.mat d4 /ascii;
save odata5.mat d5 /ascii;
save odata6.mat d6 /ascii;
save odata7.mat d7 /ascii;
save odata8.mat d8 /ascii;

save odata1f.mat e1 /ascii;
save odata2f.mat e2 /ascii;
save odata3f.mat e3 /ascii;
save odata4f.mat e4 /ascii;
save odata5f.mat e5 /ascii;
save odata6f.mat e6 /ascii;
save odata7f.mat e7 /ascii;
save odata8f.mat e8 /ascii;

fmat=fmat';
save ormat.mat rmat /ascii;
```

```
save olmat.mat lmat /ascii;
save ocaps.mat caps /ascii;
save ofmat.mat fmat /ascii;
```

Peak.m

```
function q = peak(p)

% PEAK  this function is to find the peak current of the file (p) and
%       return the number of the point in p.

q=1;
for i=1:length(p),
        if(p(i)>p(q))
                q = i;
        end
end
```

Dec5.m

```
function x = dec5(estimate)
%

x = estimate(4)  +
estimate(3)*exp(-(estimate(1)/(2*estimate(2)))*t).*
cos((1/sqrt(C*estimate(2)))*t);
```

Decay4.m

```
function q=decay4(p)
% decay4 is a function to allow the user to
% estimate a transient using the fmins command
% with R & L as the variables.

r=p(1);
l=p(2);

tc=r/(2*l);
omega=1/sqrt(C*l);
const=p(3);
offset = p(4);

q = offset+const*exp(-tc*t).*cos(omega*t) - Data;
q = sum(q.^2);
```

C.2.5 Other Program Files

Matrob.m

```
% Master startup M-file, executed by MATLAB at startup time.  On
% multi-user or networked systems, the system manager can put here
% any messages, definitions, etc. that apply to all users.

c = computer;
if c(1:2) == 'PC' & exist('serno') == 0,
mkserno
end
clear c;
echo off;
robir;
quit;
```

Matold.m

```
% Master startup M-file, executed by MATLAB at startup time.  On
% multi-user or networked systems, the system manager can put here
% any messages, definitions, etc. that apply to all users.

disp('                    HELP, DEMO, and INFO are available.')
disp('')
c = computer;
if c(1:2) == 'PC' & exist('serno') == 0,
mkserno
end
clear c;
```

Transit.c

```
#include <stdio.h>

main()
{
  int i;
  float num;
  FILE *pf, *of, *fopen();

  pf = fopen("odata1.mat","r");
  of = fopen("holdit.dat","w");

  while (fscanf(pf,"%f",&num)!=EOF) {
  fprintf(of,"%d\n",(int) num);
  }
  fclose(pf);
  fclose(of);
  system("copy holdit.dat data1.mat");
```

```
pf = fopen("odata2.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);

system("copy holdit.dat data2.mat");

pf = fopen("odata3.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data3.mat");

pf = fopen("odata4.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data4.mat");

pf = fopen("odata5.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data5.mat");

pf = fopen("odata6.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data6.mat");

pf = fopen("odata7.mat","r");
```

```
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data7.mat");

pf = fopen("odata8.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data8.mat");



pf = fopen("odata1f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data1f.mat");

pf = fopen("odata2f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);

system("copy holdit.dat data2f.mat");

pf = fopen("odata3f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data3f.mat");
```

```
pf = fopen("odata4f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);

system("copy holdit.dat data4f.mat");

pf = fopen("odata5f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data5f.mat");

pf = fopen("odata6f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data6f.mat");

pf = fopen("odata7f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data7f.mat");

pf = fopen("odata8f.mat","r");
of = fopen("holdit.dat","w");

while (fscanf(pf,"%f",&num)!=EOF) {
fprintf(of,"%d\n",(int) num);
}
fclose(pf);
fclose(of);
system("copy holdit.dat data8f.mat");

pf = fopen("olmst.mat","r");
```

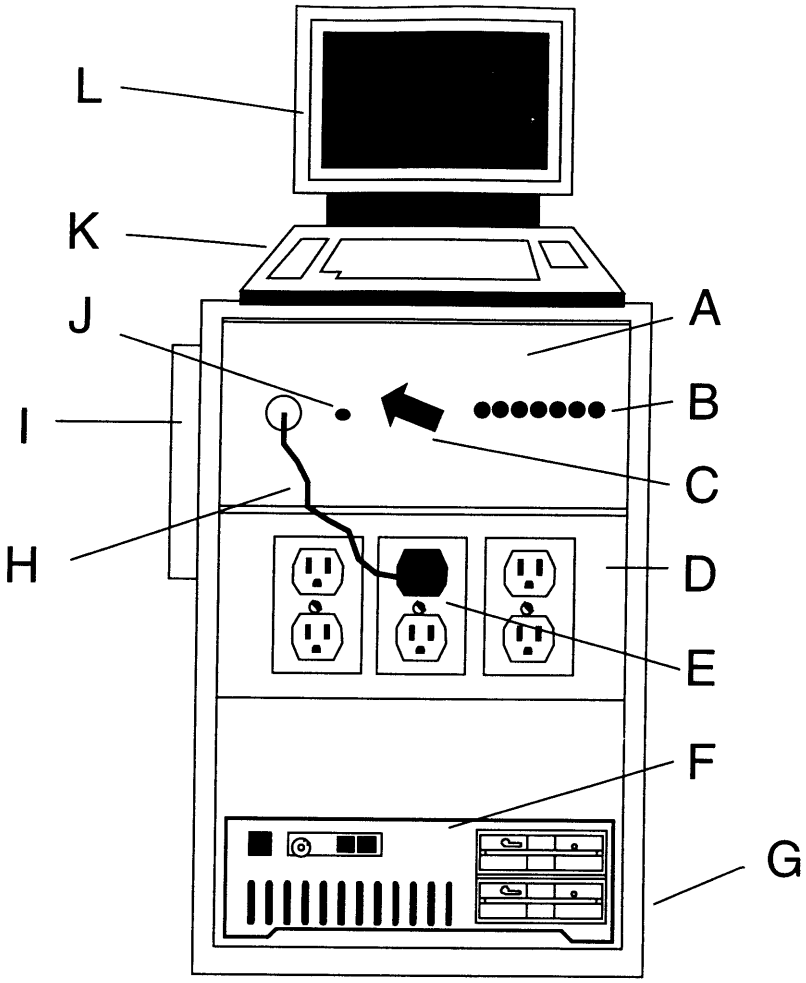
```
  of = fopen("holdit.dat","w");

  while (fscanf(pf,"%f",&num)!=EOF) {
  fprintf(of,"%6.5f\n",(float) num);
  }
  fclose(pf);
  fclose(of);
  system("copy holdit.dat data.lma");

  pf = fopen("ormat.mat","r");
  of = fopen("holdit.dat","w");

  while (fscanf(pf,"%f",&num)!=EOF) {
  fprintf(of,"%5.3f\n",(float) num);
  }
  fclose(pf);
  fclose(of);
  system("copy holdit.dat data.rma");

  pf = fopen("ofmat.mat","r");
  of = fopen("holdit.dat","w");

  while (fscanf(pf,"%f",&num)!=EOF) {
  fprintf(of,"%6.5f\n",(float) num);
  }
  fclose(pf);
  fclose(of);
  system("copy holdit.dat data.fma");

}
```

Robmat.bat

```
@echo off
cd \matlab\matlab
copy matrob.m matlab.m
cd \desire
```

Matorig.bat

```
@echo off
cd \matlab\matlab
copy matold.m matlab.m
cd \desire
```



A - Capacitor Box
B - Relay Setting Indicators
C - Gain Adjustment Switch
D - Load Test Stand
E - Phase for Connection to Cap. Box
F - Personal Computer
G - Disk Drive Access
H - Connection Between Cap. Box and Test Stand
I - Power Strip
J - Power Indicator
K - Keyboard
L - Monitor

Figure C-1: Diagram of DESIRE

Appendix D

Code for Prediction of Waveforms

D.1 Rfft2.m

```
function [f,Pyy] = rfft(t,x,n)
%
% [f,Pyy] = rfft(t,x,n)
% t = time frame
% x = signal
% n = number points
%


Y = fft(x,n);
Pyy = sqrt(Y.*conj(Y))/n;

f = 1/(t(2)-t(1))/n*(0:(n/2-1));

plot(f,Pyy(1:(n/2)));
```

D.2 Pred5.m

```
function ve = pred5(r,l,y)
% ve = pred5(r,l,y)
% r = res
% l = induct
% y = scaling of current harmonics

t=linspace(0,0.1,500);
```

```
vi = 178*sin(377*t);
vrd = r*(-y(1)*sin(377*t)+y(2)*sin(377*3*t)
-y(3)*sin(377*5*t)+y(4)*sin(377*7*t));
vld = l*(-y(1)*377*cos(377*t)+y(2)*377*3*cos(377*3*t)
-y(3)*377*5*cos(377*5*t)+y(4)*377*7*cos(377*7*t));
vo1=vi+vrd+vld;
dt=linspace(.05,.1,250)-.05;
vdt=34*(r*sin(377*dt).*exp(-10.6*dt)
-l*10.6*sin(377*dt).*exp(-10.6*dt)
+l*377*cos(377*dt).*exp(-10.6*dt));
ve=[vo1(1:250),vo1(251:500)-vdt];
```

D.3 Predi5.m

```
function ve = predi5(y)
r=1;
l=0;
t=linspace(1e-4,0.1,500);
vrd = r*(-y(1)*sin(377*t)+y(2)*sin(377*3*t)
-y(3)*sin(377*5*t)+y(4)*sin(377*7*t));
vld = l*(-y(1)*377*cos(377*t)+y(2)*377*3*cos(377*3*t)
-y(3)*377*5*cos(377*5*t)+y(4)*377*7*cos(377*7*t));
vo1=vrd+vld;
dt=t(246:500)-t(246);
vdt=r*34*sin(377*dt).*exp(-10.6*dt)
-l*10.6*34*sin(377*dt).*exp(-10.6*dt)
+l*34*377*cos(377*dt).*exp(-10.6*dt);
ve=-[vo1(1:245),vo1(246:500)-vdt];
```

D.4 Predigen.m

```
function c=predigen(real,imag,scale)
%
% c = predigen(real,imag,scale)
%
% real = coefficients of in phase (real) harmonic components
% imag = coef. of  out of phase (imaginary) harmonic components
% scale is the scale factor


t = linspace(0,.1,1000);
c = zeros(1,1000);
```

```
for i=1:length(real)
c = c+real(i)*sin(377*t*(2*i-1));
end

for i=1:length(imag)
c = c+imag(i)*cos(377*t*(2*i-1));
end

c=c*scale;
```

Appendix E

V-Section Code

E.1 Vsmaker.m

```
function [data,out,input,diffmax] =
          vsmaker(filtertype,order,thresh,vstype, vsnum)

% This program is designed to take a filter type and order
% and threshhold and then filter the data using a separate
% m-file.  This filtered data will then be sliced and diced
% using "diff" and the threshold in the m-file vspart1.m
%
%  [data,out,input,diffmax] = vsmaker(filtertype,order,thresh,vstype,vsnum)
%
% filtertype = 1 for butterworth filter
% filtertype = 2 for median
%
% vstype = 1 for vs with overlap of all vsections
% vstype = 2 for vs with overlap + "vsnum" points
% vstype = 3 for vs with "vsnum" points

%% global to1 to2 to3 to4 to5 to6 to7 to8 to9 to10

input = [];
if exist('to1')==1;
        input=[input,to1];
```

```
end
if exist('to2')==1;
        input=[input,to2];
end
if exist('to3')==1;
        input=[input,to3];
end
if exist('to4')==1;
        input=[input,to4];
end
if exist('to5')==1;
        input=[input,to5];
end
if exist('to6')==1;
        input=[input,to6];
end
if exist('to7')==1;
        input=[input,to7];
end
if exist('to8')==1;
        input=[input,to8];
end
if exist('to9')==1;
        input=[input,to9];
end
if exist('to10')==1;
        input=[input,to10];
end

if filtertype == 1
        filtdat = vsfiltb2(input,order);
end

if filtertype == 2
        filtdat = vsfiltm2(input,order);
end

[m,n] = size(filtdat);
```

```
data =[];
for i=1:n
        data = [data,vspart1(filtdat(:,i),thresh)];
end

t=linspace(1,n*length(filtdat(:,1)),n*length(filtdat(:,1)));

%%%%%
%% dmean is the average of each of all of the ''to'' files
%%    at each point - it is used for determining the
%%    values of the vsections
%%%%%


for i=1:m
        dmean(i)=mean(input(i,:));
end
dmean = dmean';

num = vspart2(data);

vs = vspart3(data,num);

vso = vspart4b(vs,num);
tempout = vso;

[min,nin]=size(input);

med = mednum(vso(:,nin));
[mm,mn] = size(med);

%%%%%%
%%  This routine sets the number of points per v-section
%%%%%%

if vstype == 3
        for i=1:mm
```

```
                med(i,2) = vsnum;
        end
        out = medvs(med,dmean);
else
        if vstype == 2
                for i=1:mm
                        med(i,2) = med(i,2)+vsnum;
                end
                out = medvs(med,dmean);
        end
end



if vstype == 1
        out = medvs(med,dmean);
end

%%%%%
%% Plots the info on the screen
%%%%%

f2 = [];
for i=1:n
        f2 = [f2;input(:,i)];
end

d=[];
tout=[];
for i=1:n
        tout = [tout;out];
        d=[d;data(:,i)];
end

data = dmean;

plot(t,f2,t,tout)
```

```
%%%%%
%% Finds maximum gaussian diffrerence for v-section to be
%% identified
%%%%%

diffmax = [];
wl=out;
while m ~=0
        [vs,wl] = dicer2(wl);
        [m,n] = size(vs);
        temp=[];
        for i=1:nin
                temp = [temp;testvs3(input(:,i),vs)];
        end
        if sum(temp) ~= 0
                diffmax = [diffmax;max(temp)];
        end
end
```

E.2 Vsfiltb2.m

```
function filtdat = vsfiltb2(input,order)

% part of vscontrol1
%
% filtdat = vsfiltb2(input,order)
%

[b,a] = butter(order,0.4);

[m,n] = size(input);

filtdat=[];
for i=1:n
        filtdat = [filtdat,filter(b,a,input(:,i))];
end
```

E.3 Vsfiltm2.m

```
function filtdat = vsfiltm2(input,order)

% part of vscontrol1
%
% filtdat = vsfiltm2(input,order)
%

[m,n] = size(input);


for i=1:n
        filtdat = [filtdat,medfilt(input(:,i),order)];
end
```

E.4 Vspart1.m

```
function d = vspart1(in,thresh)
% This function is designed to return a matrix of the points
% that correspond to the absolute value of the slope of the input
% [special case of vst4: on=1, data already filtered.
%
% d = vspart(in,thresh)
% --- in=input stream, thresh = slope
%     threshold
%
%
on=1;

filtd = in;

diffdat = abs(diff(filtd));   % lpf, differentiate, then absolute value
                              % to find mag. of slope of input

y=[];

for i=1:length(diffdat),         % y is a matrix of every point value
  if diffdat(i) > thresh;        % that has a slope greater than thresh
```

```
                y = [y;i];
  end
end

%%%%%%%%%%%%%%%%%%%%
% finds length of individual vsections
%%%%%%%%%%%%%%%%%%%%

n=1;
o=[1];
                                % o is a matrix that has
                                % the length of the number of
                                % vsections in the input.
                                % the values is o  correspond to
                                % the length of each vsection.
                                % ie if length(o) = 5, then there
                                % were 5 total vsections in the input.
                                % if o(2) = 7, then the second
                                % vsection that was recorded has
                                % a length of 7


for i=1:length(y)-1,
        if y(i+1)==y(i)+1;
                o(n)=o(n)+1;
        else
                n=n+1;
                o=[o;1];
        end
end

%%%%%%%%%%%%%%%%%%%%%%%%%
% organizes into matrices for each vsection
%%%%%%%%%%%%%%%%%%%%%%%%%
% This section takes the filtered data and isolates the vsections.
% H is the # of vsections per turn on, and is used to make a matrix
% of the first vsection of each turn on (vs1), the second vsection of each
% turn on (vs2), etc.
```

```
h = length(o)/on;       % number of vsect per turn on

if h ~= floor(h)  % if h is not an integer
        disp('Vsections per turnon inconsistent')
        disp('Try adjusting the threshold')
        error('or number of turn on times.')
end


x = 1;                  % x is a pointer to the first value
                        % of a vsection

vs1=[];
vs2=[];
vs3=[];
vs4=[];
vsl1=[]; % These matrices correspond to the length of the
vsl2=[]; % vsection with the least # of points.
vsl3=[];
vsl4=[];

if h == 0;
        error('Threshold too high.  No vsections in this portion.')
end




if h==1;
        vsl1 = min(o)-1;
        i=1;
        while i <= length(o)
                        temp = filtd((y(x)):(y(x+vsl1)));
                        vs1 = [vs1,temp];
                        x=x+o(i);
                        i=i+1;
        end
```

```
end

if h==2;
        i=1;
        for z=1:on;
                vsl1 = [vsl1,o(2*z-1)];
                vsl2 = [vsl2,o(2*z)];
        end

        vsl1=min(vsl1)-1;
        vsl2=min(vsl2)-1;

        while i <= length(o)
                        temp = filtd((y(x)):(y(x+vsl1)));
                        vs1 = [vs1,temp];
                        x=x+o(i);
                        i=i+1;
                        temp = filtd((y(x)):(y(x+vsl2)));
                        vs2 = [vs2,temp];
                        x=x+o(i);
                        i=i+1;
        end
end

if h==3;
        i=1;
        for z=1:on;
                vsl1 = [vsl1,o(3*z-2)];
                vsl2 = [vsl2,o(3*z-1)];
                vsl3 = [vsl3,o(3*z)];
        end

        vsl1 = min(vsl1)-1;
        vsl2 = min(vsl2)-1;
        vsl3 = min(vsl3)-1;

        while i <= length(o)
```

```
                        temp = filtd((y(x)):(y(x+vsl1)));
                        vs1 = [vs1,temp];
                        x=x+o(i);
                        i=i+1;

                        temp = filtd((y(x)):(y(x+vsl2)));
                        vs2 = [vs2,temp];
                        x=x+o(i);
                        i=i+1;

                        temp = filtd((y(x)):(y(x+vsl3)));
                        vs3 = [vs3,temp];
                        x=x+o(i);
                        i=i+1;

        end
end

if h==4;
        i=1;
        for z=1:on;
                vsl1 = [vsl1,o(4*z-3)];
                vsl2 = [vsl2,o(4*z-2)];
                vsl3 = [vsl3,o(4*z-1)];
                vsl4 = [vsl4,o(4*z)];
        end

        vsl1 = min(vsl1)-1;
        vsl2 = min(vsl2)-1;
        vsl3 = min(vsl3)-1;
        vsl4 = min(vsl4)-1;

        while i <= length(o)

                        temp = filtd((y(x)):(y(x+vsl1)));
                        vs1 = [vs1,temp];
                        x=x+o(i);
                        i=i+1;
```

```
                        temp = filtd((y(x)):(y(x+vsl2)));
                        vs2 = [vs2,temp];
                        x=x+o(i);
                        i=i+1;

                        temp = filtd((y(x)):(y(x+vsl3)));
                        vs3 = [vs3,temp];
                        x=x+o(i);
                        i=i+1;

                        temp = filtd((y(x)):(y(x+vsl4)));
                        vs4 = [vs4,temp];
                        x=x+o(i);
                        i=i+1;

        end
end
%%%%%%%%%%%%%%%%%%%%%%
% averages vsections and returns
% vsx(mean,std)
%%%%%%%%%%%%%%%%%%%%%%
vs1o = vs1;
vs2o = vs2;
vs3o = vs3;
vs4o = vs4;



%%%%%%%%%%%%%%%%%%%%%%
% plots real data and vsections
%%%%%%%%%%%%%%%%%%%%%%
temp = size(vs1o);      % this routine insures that the number
m1 = temp(1);           % of rows of each vsection is not
temp = size(vs2o);      % exceded when setting up for
m2 = temp(1);           % display in the next routine
temp = size(vs3o);
m3 = temp(1);
```

```
temp = size(vs4o);
m4 = temp(1);

x=0;
d=zeros(length(in),1);          % d is a zero matrix
                                % d is changed to be the
                                % value of the average v-section
                                % at the points that the v-sections
                                % occur

if h==1
        i=1;
        while i <= length(o)
                for z=1:m1
                        d(y(x+z))=vs1o(z,1);
                end
                x=x+o(i);
                i=i+1;
        end
end

if h==2
        i=1;
        while i <= length(o)
                for z=1:m1
                        d(y(x+z))=vs1o(z,1);
                end
                x=x+o(i);
                i=i+1;
                for z=1:m2
                        d(y(x+z))=vs2o(z,1);
                end
                x=x+o(i);
                i=i+1;
                end
        end
end
```

```
if h==3
        i=1;
        while i < length(o)
                for z=1:m1
                        d(y(x+z))=vs1o(z,1);
                end
                x=x+o(i);
                i=i+1;
                for z=1:m2
                        d(y(x+z))=vs2o(z,1);
                end
                x=x+o(i);
                i=i+1;
                for z=1:m3
                        d(y(x+z))=vs3o(z,1);
                end
                x=x+o(i);
                i=i+1;
                end
        end
end

if h==4
        i=1;
        while i < length(o)
                for z=1:m1
                        d(y(x+z))=vs1o(z,1);
                end
                x=x+o(i);
                i=i+1;
                for z=1:m2
                        d(y(x+z))=vs2o(z,1);
                end
                x=x+o(i);
                i=i+1;
                for z=1:m3
                        d(y(x+z))=vs3o(z,1);
                end
```

```
                x=x+o(i);
                i=i+1;
                for z=1:m4
                        d(y(x+z))=vs4o(z,1);
                end
                x=x+o(i);
                i=i+1;
                end
        end
end

if h>4
        h
        disp('There are more than four vsections per turn on.')
        disp('Make sure that you have input the correct number')
        disp('turn on times.')
end

% this routine plots the input and the vsections to give the user
% a rough idea of how close the two are.

t=1;
while d(t) == 0
        t=t+1;
end


% this routine returns a "complete" vsection -- it includes
% all vsections for one turn on with the relative time spacing
% intact.  Between vsections, there are zeros.

holder = 1;
h2 = 0;
for i=t:length(d)
        if d(i) == 0
                if holder == 1
                        holder = 0;
                        h2=h2+1;
```

```
                end
        end
        if d(i) ~= 0
                if holder == 0
                        holder = 1;
                end
        end
        if h2 < h
                vx(i-t+1) = d(i);
        end
end

vx=vx';

hold off;
```

E.5 Vspart2.m

```
function num = vspart2(d)

% This function takes d from vspart1
% and returns a string corresponding to the number of
% nonzero columns of d for each row of d.  (A nonzero
% value indicates that there is a vsection for that point
% in that column.)

[m,n] = size(d);

for i=1:m
        temp=0;
        for j=1:n
                if d(i,j) ~= 0
                        temp = temp+1;
                end
        end
        num(i)=temp;
end
```

```
num=num';
```

E.6 Vspart3.m

```
function vs = vspart3(d,num)

% This function takes in d (size [4,length]) and
% num(from vspart2) and returns an average vsection for
% the turn on.  The vsection will be the average of the
% the vsections that are present (eg d(5,:) = [4,6,0,0]
% the average vsection will be 5.)

[m,n] = size(d);
vs=[];
for i=1:m
        if num(i)==0
                vs=[vs;0];
        else
                vs=[vs;sum(d(i,:))/num(i)];
        end
end
```

E.7 Vspart4b.m

```
function [vso] = vspart4b(vs,num)

% This function takes in vs (from vspart3) and num, then returns
% four matrices:vs for num>=1,num>=2,num>=3,num=4.
%
% the higher the column number, the greater the reliability of
% the vsection.

vs1=[];
vs2=[];
vs3=[];
vs4=[];
vs5=[];
```

```
vs6=[];
vs7=[];
vs8=[];
vs9=[];
vs10=[];
for i=1:length(num)
        if num(i)>=1
                vs1=[vs1;vs(i)];
        else
                vs1=[vs1;0];
        end
        if num(i)>=2
                vs2=[vs2;vs(i)];
        else
                vs2=[vs2;0];
        end
        if num(i)>=3
                vs3=[vs3;vs(i)];
        else
                vs3=[vs3;0];
        end
        if num(i)>=4
                vs4=[vs4;vs(i)];
        else
                vs4=[vs4;0];
        end
        if num(i)>=5
                vs5=[vs5;vs(i)];
        else
                vs5=[vs5;0];
        end
        if num(i)>=6
                vs6=[vs6;vs(i)];
        else
                vs6=[vs6;0];
        end
        if num(i)>=7
                vs7=[vs7;vs(i)];
```

```
        else
                vs7=[vs7;0];
        end
        if num(i)>=8
                vs8=[vs8;vs(i)];
        else
                vs8=[vs8;0];
        end
        if num(i)>=9
                vs9=[vs9;vs(i)];
        else
                vs9=[vs9;0];
        end
        if num(i)>=10
                vs10=[vs10;vs(i)];
        else
                vs10=[vs10;0];
        end
end
vso=[vs1,vs2,vs3,vs4,vs5,vs6,vs7,vs8,vs9,vs10];
```

E.8 Dicer2.m

```
function [vs,whatsleft] = dicer2(in)
% This function takes an input from vsmaker that consists of
% vsections and zeros wherever there is no vsection.  Dicer the returns
% the first vsection and input with the vsection removed.
%
% [vs,whatsleft] = dicer2(in)
%

vs=[];
whatsleft=[];
l = length(in);
bvs=1;
%flag = 0;
%while flag == 0
if sum(in) ~=0
```

```
        while in(bvs)==0
%                if bvs == 1
%                        flag = 1;
%                end
                bvs=bvs+1;
        end
%        flag=1;
%end

evs=bvs;
%while flag == 1
        while in(evs)~=0
%                if evs == 1;
%                        flag == 2;
%                end
                evs=evs+1;
        end
%        flag == 2;
%end


        vs=in(bvs:evs-1);
        whatsleft=in(evs:1);
end
```

E.9 Testvs3.m

```
function diff = testvs3(data, vs)
% testvs is designed to take a stream of data and a vsection
% (note that this is a "complete" vsection, meaning that is
% has all of the vsections of a transient with zeros between
% vsections).  The output will be a [m,1] matrix with the vsection
% data best placed where it has the least Gaussian difference with
% the input stream.  All other values will be zero.
%
% diff = testvs3(data,vs)
%
% see also testvs, vsmaker
```

```
%

ld = length(data);
lv = length(vs);
x = 0;                          % x is the # of zeros before
                        % the vsection data

xbest = 0;                % best holds the x that has the lowest
                        % gaussian difference and the difference

diff = sum(abs(data));          % sum(data) is used as a starting point
                        % for the gauss diff.  If there isn't a
                        % difference that's less than this, the
                        % vsection is not worth diddley.



for x=0:ld-lv-1
        gausshold=0;
        holder = [zeros(x,1);vs;zeros(ld-lv-x,1)];
        for i=1:ld
                if holder(i) ~= 0
                        gausshold = gausshold+abs(holder(i)-data(i));
                end
        end
        if gausshold < diff
                diff = gausshold;
                xbest = x;
                out = [zeros(x,1);vs;zeros(ld-lv-x,1)];
        end
end

% clg;
% hold on;
% plot(data,'r');
% plot(out,'+g');
% hold off;
```

E.10 Medvs.m

```
function medout = medvs(med,afdata)

% Medvs takes in med (from mednum) and afdata (usually the average of the
% filterered data) and returns the vsection of length med(xx,2)
% centered around the median point, given by med(xx,1);
%
% medout = medvs(med,afdata);
%

[md,nd] = size(afdata);
[mm,nm] = size(med);

medout = zeros(md,1);

for i=1:mm
        for j=med(i,1)-round(med(i,2)/2)+1:med(i,1)+round(med(i,2)/2)
                medout(j) = afdata(j);
        end
end
```

E.11 Mednum.m

```
function med = mednum(d)

% This function takes in the data stream from vscontrol1 and returns a matrix
% that has the median point of a vsection and the number of points of
% the vsection. Each row of med corresponds to a vsection.
%
% med = mednum(d)
%

med=[];

j=1;
while j < length(d)
        if d(j) == 0
                j=j+1;
```

```
        else        x=j;
                while d(j) ~= 0
                        j=j+1;
                end
                m = round(mean([x,j]));
                p = j-x;
                med = [med;m,p];
                j=j+1;
        end
end
```

E.12 Medfilt.m

```
function out = medfilt(in,order)

% this function takes the median of a sliding window (of length order)
% for each set of points in "in"

for i=1:length(in)-order
        temp = in(i:i+order);
        out(i) = median(temp);
end

fixl = ones(order,1)*mean(temp);
out=[out';fixl];
```