AN EXPERIMENTAL CHARACTERIZATION SYSTEM FOR DEEP ULTRA-VIOLET (UV) PHOTORESISTS

by

Dean M. Drako

Memorandum No. UCB/ERL M88/84

22 December 1988

COVER PAGE

AN EXPERIMENTAL CHARACTERIZATION SYSTEM FOR DEEP ULTRA-VIOLET (UV) PHOTORESISTS

by

Dean M. Drako

Memorandum No. UCB/ERL M88/84

22 December 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This research was supported by the Semiconductor Research Corporation and SEMATECH.

This research was supported by the Semiconductor Research Corporation and SEMATECH.

AN EXPERIMENTAL CHARACTERIZATION SYSTEM FOR DEEP ULTRA-VIOLET (UV) PHOTORESISTS

by

Dean M. Drako

Memorandum No. UCB/ERL M88/84

22 December 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This research was supported by the Semiconductor Research Corporation and SEMATECH.

An Experimental Characterization System
for Deep Ultra-Violet (UV) Photoresists

Dean M. Drako

Abstract

A versatile system designed specifically for experimental automated photoresist characterization has been constructed utilizing an excimer laser source for exposure at 248nm. The system was assembled, as much as possible, from commercially available components in order to facilitate its replication. The controlling software was written in "C" and is documented in this report. For these reasons other research groups can easily replicate the Characterization System.

An IBM PC-AT compatible computer controls the excimer laser, operates the Fourier Transform Infrared Spectrometer (FTIR), measures and records the energy of each laser pulse (incident, reflected, and transmitted), opens and closes shutters, and operates two linear stages for sample movement. All operations (except FTIR data reduction) are managed by a control program written in the "C" language.

The system is capable of measuring total exposure dose, performing bleaching measurements, creating and recording exposure pulse sequences, and generating exposure patterns suitable for multiple channel monitoring of the development. The total exposure energy, energy per pulse, and pulse rate are selectable over a wide range. The system contains an in-situ Fourier Transform Infrared Spectrometer for qualitative and quantitative analysis of the photoresist baking and exposure processes (baking is not done in-situ). FTIR may be performed in transmission or reflection. The FTIR data will form the basis of comprehensive multi-state resist models.

The system's versatility facilitates the development of new automated and repeatable experiments. Simple controlling software, utilizing the provided interface sub-routines, can be written to control new experiments and collect data.

Table of Contents

Chapter 1
Introduction

Motivation

The semiconductor industry currently uses photoresist and photolithography for pattern transfer. The pattern transfer resolution is reaching limits due, in part, to the exposing light's wavelength. This has prompted research into exposure systems with shorter wavelengths[1] and new photoresists suitable for use at these wavelengths[2-4]. To obtain the best performance from new photoresists they must be characterized, understood, and modeled[5]. No equipment or experimental apparatus is readily available to facilitate the large number of experiments needed.

An excimer laser (a candidate deep-uv, 248nm, light source) is a pulse type laser. Pulsed light is inherently different than the continuous light currently used for exposure. The laser itself and the light pulsing introduce additional exposure effects (eg. ablation, interference) and additional variables (eg. pulse rate) to the exposure process. The consequences must be examined and understood.

Commercially available equipment may be produced for the characterization of photoresist using excimer sources. Commercial equipment, however, rarely lends itself to the research environment. A commercial system typically provides data for well known models, well understood effects, or for a single variable. Rarely can one examine multiple effects simultaneously using different analysis techniques. In a research environment one wishes to pursue data for new models and less understood effects. Commercial equipment has repeatedly proven inadequate for these applications[6,7].

To overcome the problems with commercial equipment research labs typically must develop their own equipment to perform the desired experiments. This leads to a redundant effort by different laboratories working in the same field.

A Solution

A system or "experiment environment" was designed to overcome these three basic problems: 1) lack of equipment for deep-uv exposure characterization, 2) the problems associated with using commercial equipment in a research environment, and 3) the redundant development work performed by different laboratories. The results are a versatile hardware system, suitable for experimental research, constructed from commercially available components. A standard IBM PC-AT compatible computer controls numerous building block components. These components and other optics are configured on a standard optical table (this allows for changes in the optical path etc...). The computer controls the entire experiment and collects the data. Once an experiment is performed it may be easily repeated by a lab technician. Experiments are built on a semi-permanent basis and available as analysis equipment. A typical configuration is shown in Figure 1. It is useful for doing in-situ FTIR, DRM exposure matrix generation, and bleaching measurements.

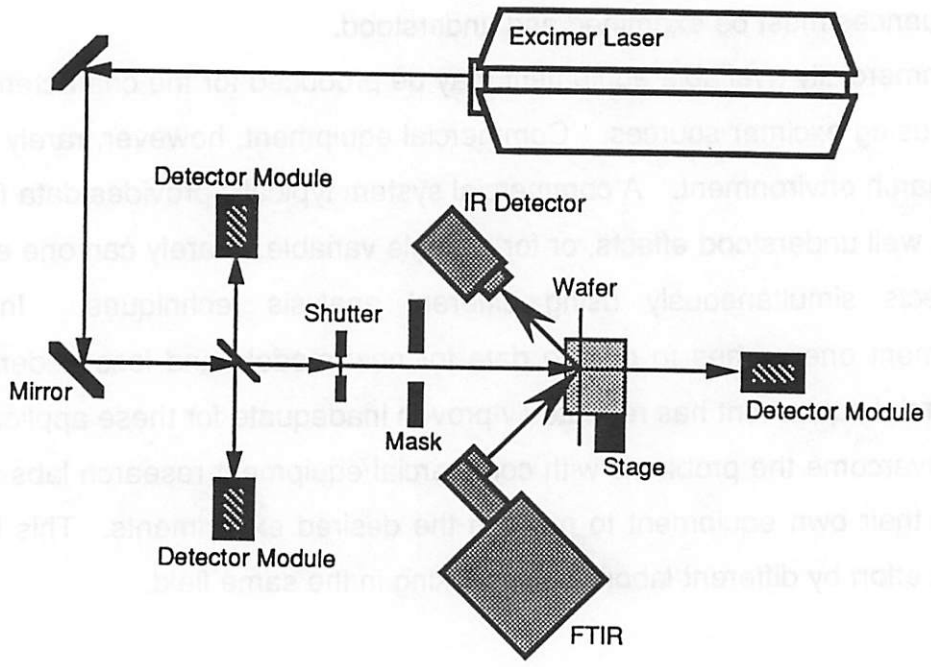


Figure 1. *Typical Optical Configuration*

Other laboratories may easily duplicate the Characterization System because it is constructed from commercially available hardware and is documented. The controlling software is written in a standard language, "C", and is also documented. The software includes interface sub-routines for writing additional programs for new experiments. It provides a foundation for experiment development.

This report documents the design and operation of the Photoresist Characterization System. It is split into three major sections: Chapter 2 describes the hardware designed and purchased (Appendix A lists vendors). Chapter 3 describes the software's operation. Chapter 4 describes the internals of the software (Appendix B contains variable listings and Appendix C contains program listings).

[1] Victor Pol, James H Bennewitz, Gary C. Escher, Martin Feldman, Victor A. Firtion, Tanya E. Jewell, Bruce E. Wilcomb, and James T. Clemens, SPIE Vol 633 Optical Microlithography V, 1986.

[2] T.M. Wolf, R.L. Hartless, A. Shugard, G.N. Taylor, "The Evaluation of Positive Acting Resists for Lithoography at 248 nm, J. Vac. Sci. Technology, Vol 5, No. 1, Jan/Feb 1987.

[3] Don E. Lyons, "Resist Materials For Deep UV Lithography", University of California, Berkeley, CA, EECS 243 Class Report, 1988.

[4] C. Grant Wilson, Robert D. Miller, Dennis R. McKean, Lester A. Pederson, and Manfred Regitz, "New Diazoketone Dissolution Inhibitors for Deep UV Photolithography", SPIE Vol 771, Advances in Resist Technology IV, 1987.

[5] F.H. Dill, A.R. Neureuther, J.A. Tuttle, and E.J. Walker, "Modeling Projection Printing of Positive Photoresists, IEEE Transactions on Electron Devices, Vol Ed-22, No. 7 July, 1975.

[6] M. Exterkamp, W. Wong, H. Damar, A.R. Neureuther, C.H. Ting, and W.G. Oldham, "Resist Characterization: procedures, parameters, and profiles," SPIE Vol 334 Optical Microlithography -- Technology for the Mid-1980's, 1982.

[7] K.L Konnerth and F.H. Dill, "In-Situ Measurement of Dielectric Thickness During Etching or Developing Processes," IEEE Transactions on Electron Devices, Vol ED-22, No. 7, JUly 1975.

Chapter 2
Hardware

This chapter describes the hardware of the Characterization System. An IBM PC-AT compatible computer is used to collect the data and control the experiments. Several basic subsystems have been added to expand its capabilities. A complete list of the equipment, product numbers, and vendors is provided in Appendix A. The subsystems are:

1) Data Translation DT2801-A -- an eight channel differential analog to digital (A/D) converter with 16 channels of digital input/output (I/O)
2) TECMAR Dual Stepper Motor Controller
3) Analect Instruments RFX-65 Fourier Transform Infrared Spectrometer (note that the IBM PC-AT compatible was actually supplied by Analect)

Each of the subsystems requires a board in the IBM PC-AT compatible. The DT2801-A (hereafter referred to as DT2801) is used for data collection and experiment control. The A/D channels are connected to three Integrator Units which measure the energy in a single laser pulse. The DT2801's digital I/O is utilized for firing the laser, opening and closing a shutter, and setting the Integrator Unit's gain. The TECMAR Stepper Motor Controller operates two linear stages. The Fourier Transform Infrared Spectrometer (FTIR) Interface controls the operation of, and collects data from the FTIR system.

As much of the system as possible was constructed from commercially available products. This allows other research facilities to easily duplicate the system's hardware.

Analog to Digital Subsystem

The DT2801 performs the general system I/O. It is a 12 bit A/D conversion system capable of operation at 27.5 kHz (36 microseconds per conversion). The eight differential channels are multiplexed to a single A/D converter pair. The system is configured for bipolar operation with a maximum input voltage swing of +-10V. The 16 channels of digital I/O are TTL compatible. The DT2801 is a single board system which sits in the

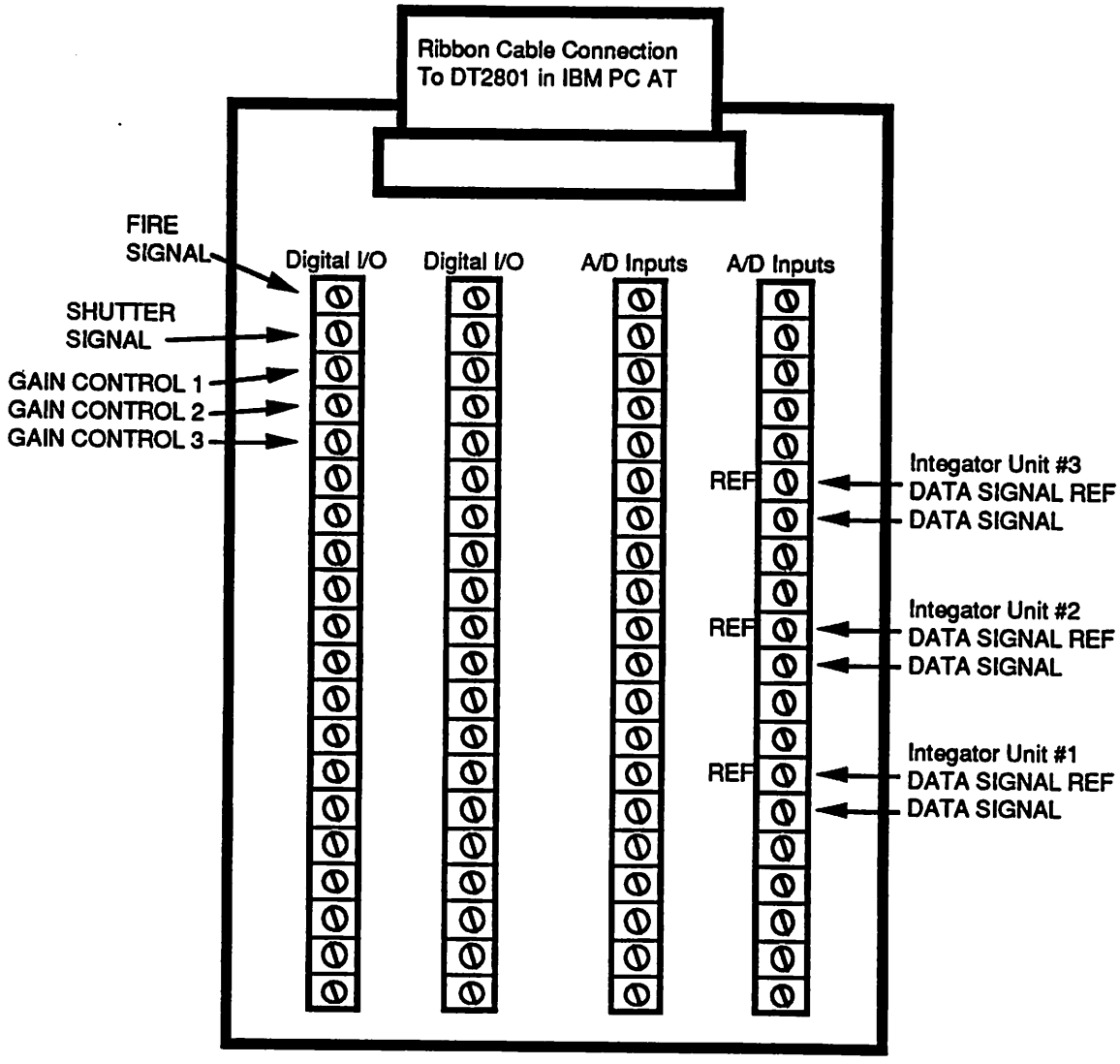


Figure 2. *Terminal Block Board and Connections*

IBM PC-AT compatible. A 50 wire ribbon cable connects the DT2801 to a Terminal

Block Board (Data Translation part no. DT707) used for all external connections (see Figure 2, page 5). The connections on the Terminal Block Board are detailed later.

The DT2801's 12 bit converters provide 4.88 mV resolution over 20 volts. If we assume half scale (10 volts) utilization and single bit noise we obtain 9.8 mV resolution and 0.1% error. These values are adequate for the laser energy measurement.

Stepper Motor Controller Subsystem

A TECMAR Stepper Motor Controller Board in the IBM PC-AT compatible and a simple Power Amplifier Board form the Stepper Motor Control Subsystem. The TECMAR board is capable of controlling two standard 4 phase stepper motors. Three 14 wire ribbon cables connect the TECMAR board to a Power Amplifier Board external to the IBM PC-AT compatible. The Power Amplifier Board interfaces the TECMAR board to the stepper motors and to the linear stages. The Motion Control System section in this chapter details the Power Amplifier Board and the linear stages.

Fourier Transform Infrared Spectrometer Subsystem

The FTIR system is a commercially available RFX-65 from Analect Instruments. The system interfaces to the IBM PC-AT compatible through a dedicated local area network interface. All data collection and analysis are performed by the IBM PC-AT compatible. The RFX-65 is ideal for a research environment because of its modular nature. The interferometer and detector modules are small and designed to mount directly to a standard optical table. Experimental set ups for in-situ FTIR measurement are easily constructed. The electronics and controlling hardware are removed from the optical table and the interferometer and detector maneuvered to set up the experiment (see Figure 3, page 7).

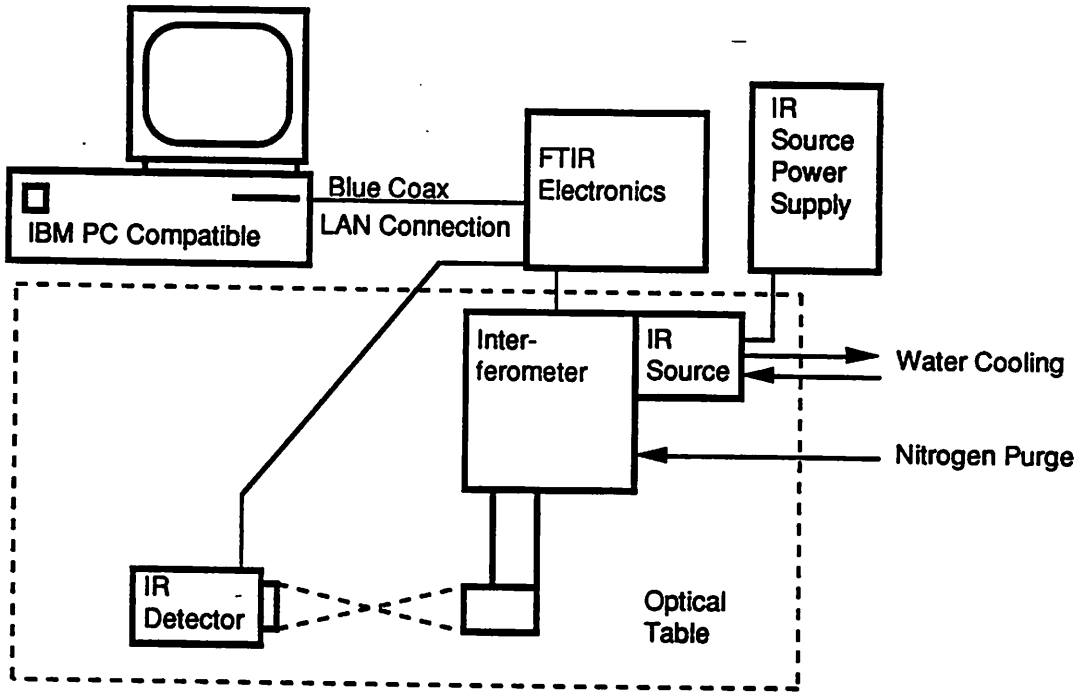


Figure 3. *FTIR Setup*

Laser Firing Control

The laser is a 248nm excimer laser manufactured by Cymer Laser, Inc. It provides pulsed output of 20 Watts. The laser firing can be directly controlled by a TTL level signal when the laser is in "triggered mode." The Laser is placed into triggered mode using its own control panel. When triggered the laser begins a "burst." The "burst count" of the laser should be set to one for proper laser energy measurement. This effectively gives complete control of the laser's firing to the IBM PC-AT compatible computer.

The Cymer laser is controlled by the computer through the DT2801's digital I/O. Digital I/O bit 0 is configured for output. It is connected to the trigger input of the laser (see Figure 4, page 8). When the laser is in the "triggered mode" the computer will-have direct control of the laser. The computer brings the FIRE SIGNAL high to begin pre-charge of the laser's capacitors. Pre-charge lasts from 4 to 8 milliseconds (depending on laser model). After completing pre-charge the computer drives the FIRE SIGNAL low and the laser fires (with a jitter of less than 2 microseconds). If the

computer drives the signal low before the pre-charge is complete the laser will not fire until after the pre-charge has finished. Note that if this occurs the laser energy measurement system will not obtain correct data. For a 200 Hz Cymer laser a 4.1 millisecond high pulse is recommended. The pulse is software configurable (see Software User's Guide -- Hardware Parameters, page 28).

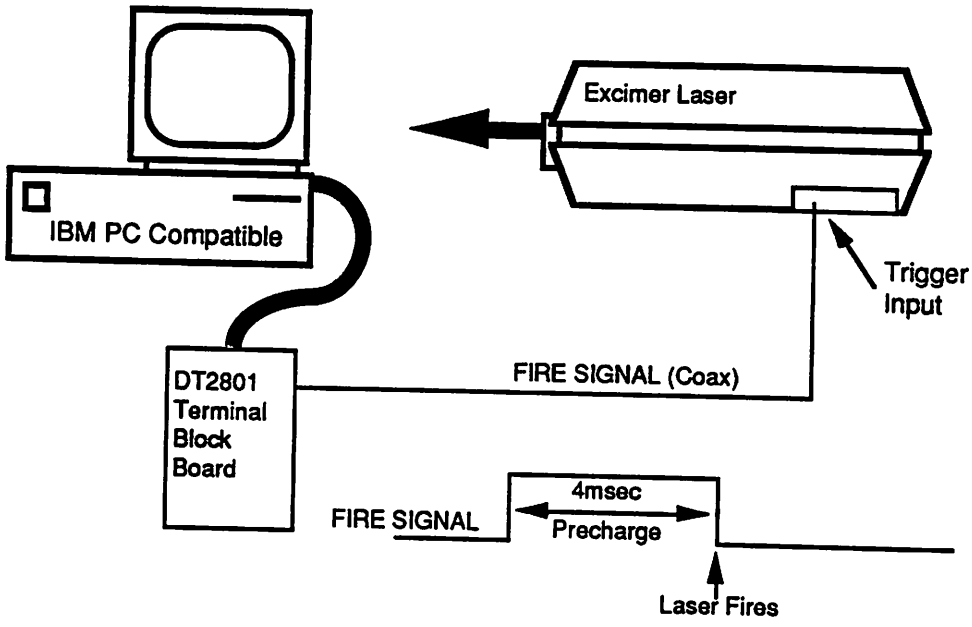


Figure 4. *Laser FIRE SIGNAL Connections*

The laser's capacitors may be held in the charged state for an extended period of time by holding the FIRE SIGNAL high. This is NOT recommended because of the extra stress on the capacitors which will reduce the laser's lifetime.

Laser Energy Measurement

The characterization system is capable of measuring the laser pulse energy at up to three locations (for example incident, reflected, and transmitted energy can be measured simultaneously). Any of the eight A/D channels may be used for data collection. Normally channels zero, one, and two are used. The operating channels, however, are software selectable. Each of the channels used for laser energy

measurement is connected to an Integrator Unit which in turn is connected to a Diode Module (see Figure 5, page 9). The Integrator Unit is responsible for integrating the energy in a single laser pulse to yield a voltage. It converts a charge pulse (which is proportional to light energy) from the Diode Module to a voltage that the A/D subsystem can use. Commercial units are available to perform this function, however, they are typically much more complex and much too costly for such a simple application. For these reasons a custom circuit was utilized.

The FIRE SIGNAL is connected to the Integrator Units in addition to the laser. When pre-charge of the laser begins (the FIRE SIGNAL is driven high) the Integrator Units zero their stored voltages. Once the computer fires the laser (drives the FIRE SIGNAL low) the Integrator Units begin integrating charge received from the Diode Module. The charge integration proceeds for approximately 40 microseconds after the

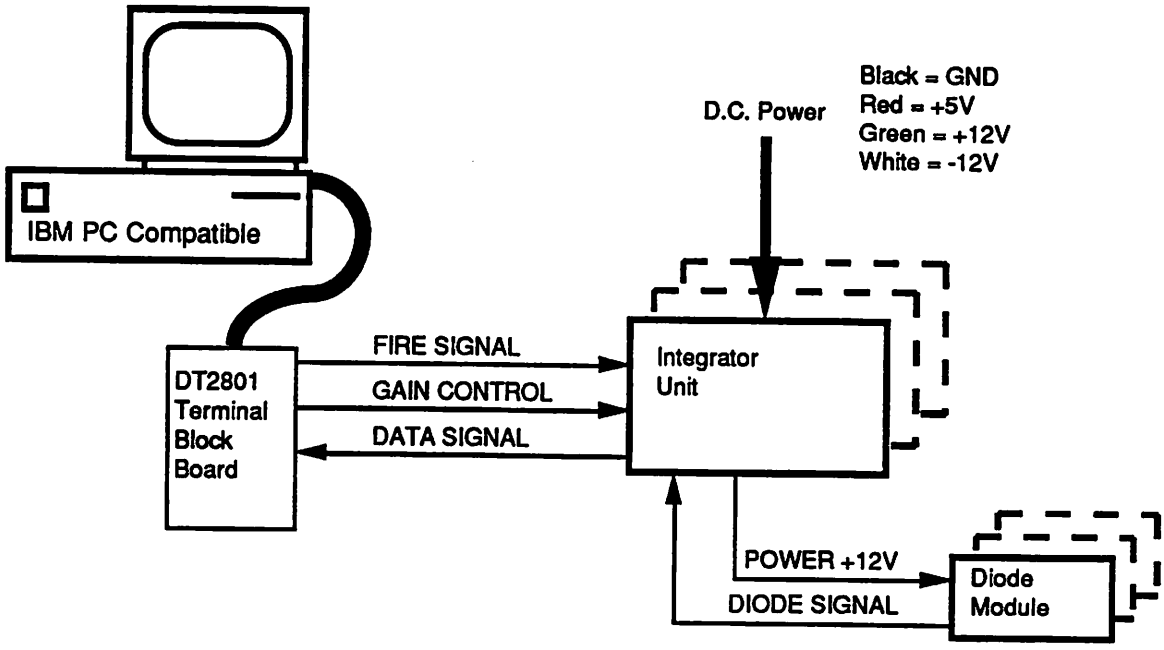


Figure 5. *Laser Pulse Energy Measurement*

laser fire pulse is given. The integration time is determined by an RC time constant in each Integrator Unit. The laser must fire and the diode respond fully during this 40 microsecond period. At the end of the 40 microseconds a sample a hold circuit records

the integrated voltage value. The integrated voltage is later converted to a digital number using the A/D subsystem. The short integration period eliminates data corruption from ambient light. If problems from ambient light are observed the integration period can be shortened by reducing the value of capacitor C2.

A circuit diagram of the Integrator Unit is given in Figure 6 on page 11. The Integrator Unit functions as follows. While the FIRE SIGNAL is high (the laser is precharging) the output Q of U4a will be low. This will effectively turn on the J-Fet (U2c) causing the output of the op amp (U1) to tend towards ground (this assume negligible input current from the diode, which is true). The integration capacitor (C3 or C4) is discharged during this period and the output voltage of the operational amplifier is zero. When the FIRE SIGNAL is driven low (the laser will fire in 2 microseconds), the J-Fet is turned off. The op amp then acts as a near ideal integrator (its input impedance is 10^{12} Ohms). When the laser light illuminates the photodiode it generates a current pulse. The charge in this pulse is integrated on the integration capacitor and a voltage is formed at the output of the Op Amp. When the J-Fet is turned off, however, it injects a certain amount of charge onto the integration capacitor because of its internal capacitance. This is canceled by capacitor C5 and an inverter (U4b) which brings the output reference back to zero volts.

At the falling edge of the FIRE SIGNAL (this fires the laser) the inverter(U4a) creates a rising edge. This rising edge triggers the Multistable (U5a) which outputs a 40-60 microsecond long logic high pulse. The length of this pulse is determined by R3 and C2. The pulse is used as a sample window for the sample and hold circuit (U3). The sample and hold circuit samples during the 40 to 60 microseconds following the FIRE SIGNAL's falling edge. It then holds the data until the laser fires again. The sample and hold provides ample time for the IBM PC-AT compatible and the DT2801 to perform the A/D conversion.

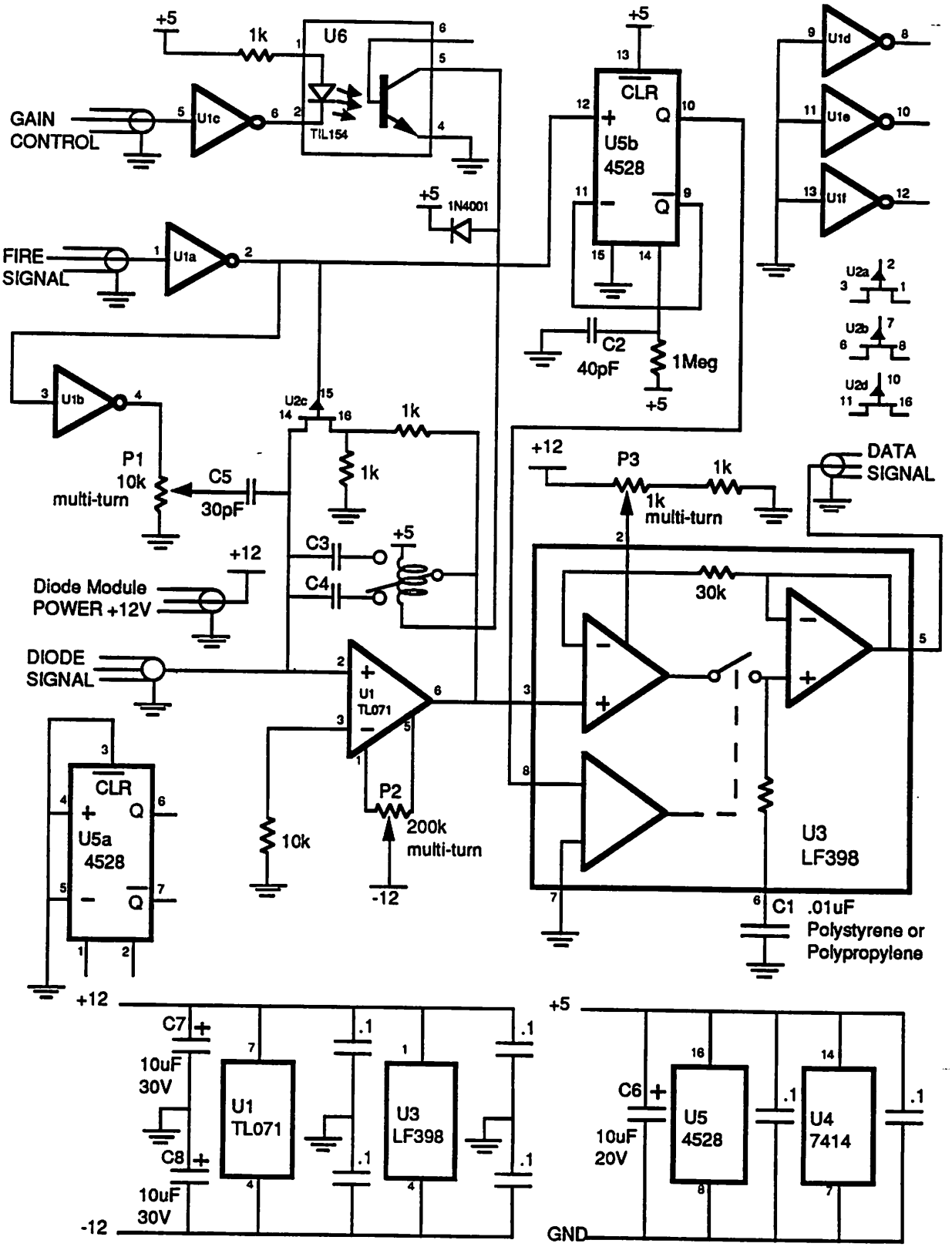


Figure 6. *Integrator Unit Circuit Diagram*

Adjustment of the Integrator Unit Potentiometers

The potentiometers should be adjusted using the following technique. The Diode Module should be disconnected or covered such that it receives no light. A clocking signal should be applied to the FIRE SIGNAL input (this can be accomplished by placing the computer into Continuous Fire Mode see Chapter 3 -- Manual Operations for details, page 30). An oscilloscope probe should be placed at pin 6 of the Op Amp (U1). A large gain setting should be used on the scope (100 mV per division or so). Adjust P1 until the signal observed on the scope is as flat as possible. Disconnect the scope. Connect an accurate voltmeter to the same location. Drive the FIRE SIGNAL high (the IBM PC-AT can be used to do this too). This will turn on the J-Fet. Adjust P2 until the voltage on the voltmeter is zero using as much accuracy as possible (approx. 2mV).

Connect the voltmeter to the Data Out line of the Integrator Unit. Connect a clocking signal to the FIRE SIGNAL input as was done earlier. Adjust P3 until a zero voltage reading is obtained.

After the laser fires the IBM PC-AT performs an A/D conversion on each of the inputs. A conversion takes approximately 36 microseconds per channel. The decay of the signal in the Integrator Unit's sample and hold circuit during this time is negligible. The computer uses the data collected to determine the amount of light energy in the pulse at each of the detector locations. See the Chapter 3 Software User's Guide -- Detector Parameters (page 26) for the exact calculations.

Diode Module

The Diode Modules consist of a PIN Silicon Photodiode with fused silica windows mounted in a holder. The diodes are reverse biased with 12 volts to improve their linearity. Components are included at the diode location to limit the D.C. current (see Figure 7, page 13). The photodiodes are mounted in an aluminum holder to provide simple mounting and movement. A lens is mounted in front of the diode to reduce the susceptibility to angular alignment and collect the light in a larger area. A collar on the front of the diode mounts is used to hold optical filters or other components. The diodes are extremely sensitive and a filter is typically required to avoid saturation. The photodiodes have an approximate sensitivity of 0.11 A/W at 248nm. (Hamamatsu part

no. S1722-02). If we assume a 2000 pF integration capacitor we can calculate:

$$\Phi = \left[\frac{1.0\ J}{0.11\ C}\right]\left[2000\ pF\right]$$

$$= 18.2\ nJ/Volt$$

Where Φ is the sensitivity of the Diode Module and Integrator Unit. A typical measured value for Φ is 18.75 nJ/Volt.

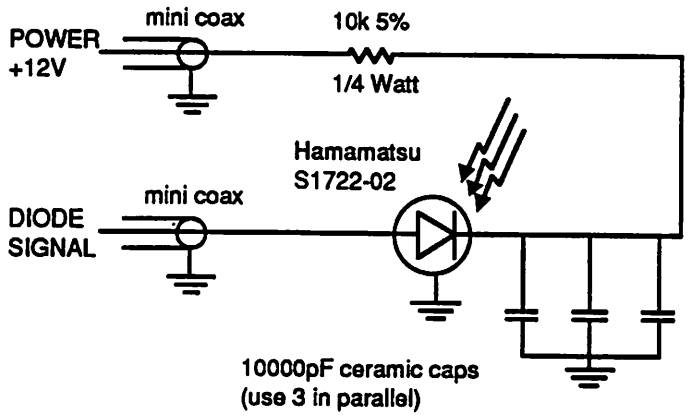


Figure 7. *Diode Module Circuit Diagram*

Shutter System

A large aperture shutter is integrated into the system for various experiments. It is primarily used as a laser beam block. The laser can be fired and its energy measured before the sample is illuminated. This allows a check on the laser's energy before proceeding with the experiment. The laser's energy varies with gas quality and voltage and should be checked before any experiments.

The shutter is operated through the DT2801 digital I/O section. It utilizes bit 1 of the interface. The shutter is a 35 mm electronic shutter from Ealing Electro-Optics (part

no. 22-8411) with a basic shutter power supply. The basic shutter power supply allows an operator to open and close the shutter with a switch. The basic shutter supply was modified to place the shutter under computer control. A circuit diagram is shown in Figure 8. The modifications allow the IBM PC-AT compatible to open and close the shutter with a TTL level signal from the DT2801.

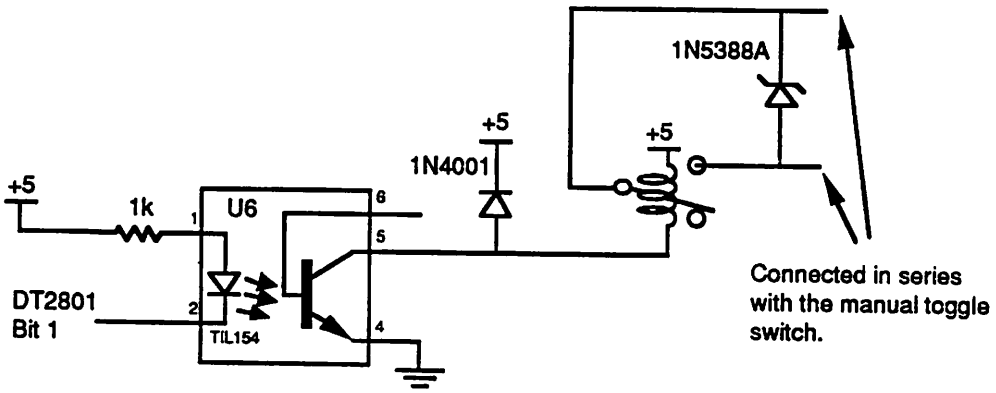


Figure 8. *Shutter System Circuit*

Motion Control System

The motion control system was added to the IBM PC-AT compatible to provide automated sample movement. The system consists of a TECMAR Stepper Motor Controller Board installed in the IBM PC-AT compatible computer. The TECMAR board is connected to the Power Amplifier Board which drives the stepper motors. The stepper motors are connected to a pair of linear stages which provide automated sample movement. The Power Amplifier Board also provides interface signals for monitoring the stages' limit switches. The stepper motor/stage combination provide a resolution of 1 mil. The accuracy is comparable to 1 mil but has never been accurately measured.

The TECMAR Stepper Motor Controller Board provides two semi-intelligent stepper motor controller chips (Cybernetic Micro Systems CY512). The TECMAR board interfaces the two chips to the IBM PC-AT internal bus. In addition, the TECMAR board provides several TTL input and output channels. The input channels are used to

monitor the stages' limit switches. All connections to the TECMAR board are made through the Power Amplifier Board. The two boards are connected to each other by three 16 conductor ribbon cables (see Figure 9). One cable is used for each stepper motor and another cable is used for the TTL input channels. Only one of the four TECMAR TTL I/O connectors is utilized (IN B). The Power Amplifier Board connects to the stepper motors and to the limit switches on the linear stages. It isolates the computer system from the noise and power associated with the motors. A separate power supply (+12 Volts) is utilized for the stepper motors. Completely separating the logic and motor power supplies increases the system's reliability. The motor power supply is connected to the Power Amplifier Board (see Figure 13, page 19). The TECMAR board provides the signals shown on the chart titled TECMAR STEPPER MOTOR RIBBON CABLE PINOUT for each of the stepper motors. The Phase 1-4 signals are the only ones, besides Vdd and GND, which are needed. These signals are provided directly by the CY512 stepper motor controller chips. For details see the CY512 data book. The

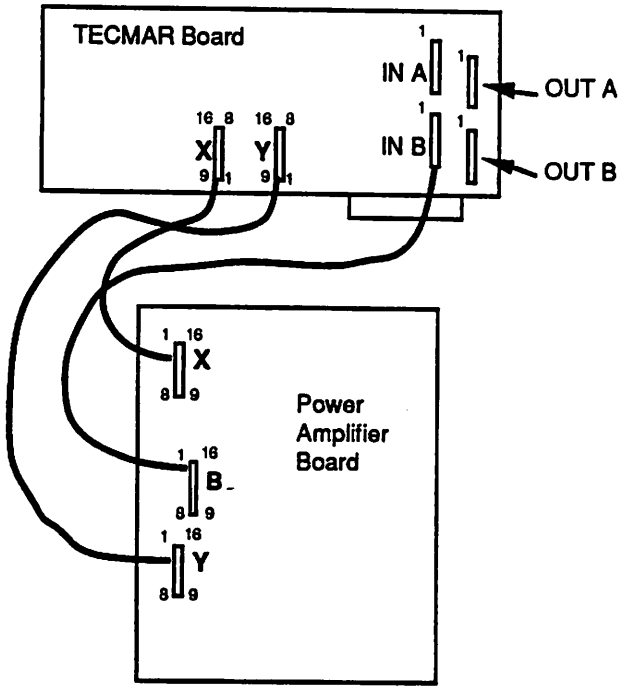


Figure 9. *TECMAR/Power Amplifier Connections*

Tecmar Stepper Motor Ribbon Cable Pinout

RIBBON CABLE PIN NUMBER	FUNCTION
1	Phase 1
2	Phase 2
3	Phase 3
4	Phase 4
5	DRCTN
6	CNTRL
7	PULSE
8	-MOT CNTL
10	-RUN
12	-SLEW
15	GND
16	Vdd (+5)

Power Amplifier Board amplifies the Phase 1-4 signals to create drive for the stepper motors. The Vdd and GND signals from the TECMAR board power the buffer chip on the Power Amplifier Board (see Figure 12).

A circuit diagram for one of the amplifier channels on the Power Amplifier Board is shown in Figure 12. The board actually contains two copies of this circuit (one for the X channel and one for the Y channel). The coils shown on the diagram are not on the Power Amplifier Board itself (they are in the motors) but are included for clarity. The Phase signals generated by the TECMAR board are buffered (by the TTL 7406), optically isolated from the motors (by the TIL154), and amplified (by the IRFD110). Note the complete separation of both the power and the ground planes between the computer and the motors. The IRFD110 is capable of conducting one ampere. This implies that the system is capable of driving stepper motors with up to one amp per phase.

The Power Amplifier Board is connected to the two stepper motors and to the limit switches of the linear stages. The connections are shown in Figure 13.

The stepper motors are standard 4 phase, 1.8 degrees/step, 0.5 amp per phase units manufactured by Oriental Motors. The motors operate directly from a 12 Volt power supply and provide 58 Ounce/In of torque. They have a standard NEMA 23 mount which connects directly to the linear stages. The motors have six wires which are connected to the Power Amplifier Board. The connections are clarified in Figure 13 and Figure 10.

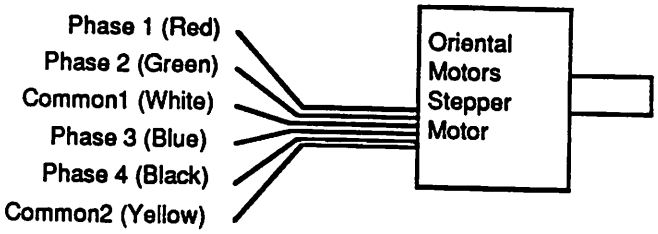


Figure 10. *Stepper Motor Wiring*

The linear stages include home and limit switches. The limit switches are connected to provide TTL level signals to the TECMAR board through the Power Amplifier Board. The TECMAR "IN B" port pinout is shown in the chart titled TECMAR IN B RIBBON CABLE PINOUT. The limit switch inputs to the TECMAR board (IN B) are connected to the Vdd power supply through 1k

TECMAR IN B
Ribbon Cable Pinout & Signals

RIBBON CABLE PIN NUMBER	FUNCTION
1	Data Bit 0
2	Data Bit 1
3	Data Bit 2
4	Data Bit 3
5	Data Bit 4
6	Data Bit 5
7	Data Bit 6
8	Data Bit 7
9-16	GND

pullup resistors. These inputs are grounded if the switch on the linear stage closes. A circuit diagram of the switches is shown in Figure 14. The "CW" and "CCW" abbreviations stand for clockwise and counter-clockwise respectively. The DAEDEL stages are provided complete with premounted limit switches and cables. Only four of the nine (see Figure 11, page 17) connections on the stage are connected to the Power Amplifier Board. The CW Common, CCW Common, CW Normally Open (N.O.), and CCW Normally Open connections are used. The remainder are not connected. The connections to the Power Amplifier Board are shown in Figure 13.

The DAEDEL linear stages provide a six inch square base with four inches of travel. They were drilled with an array of tapped holes on one inch centers. They provide a versatile base for sample movement.

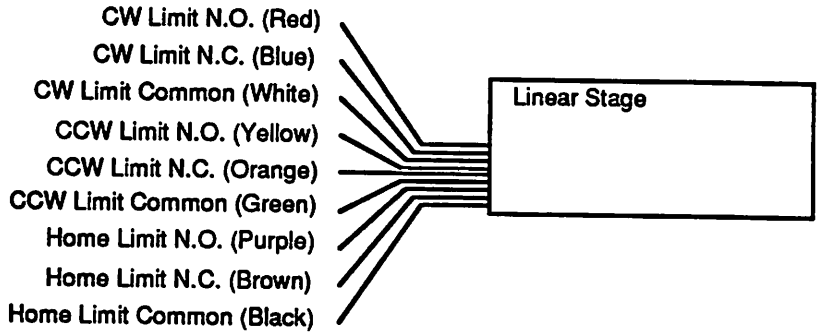


Figure 11. *DAEDEL Stage Limit Switch Cable*

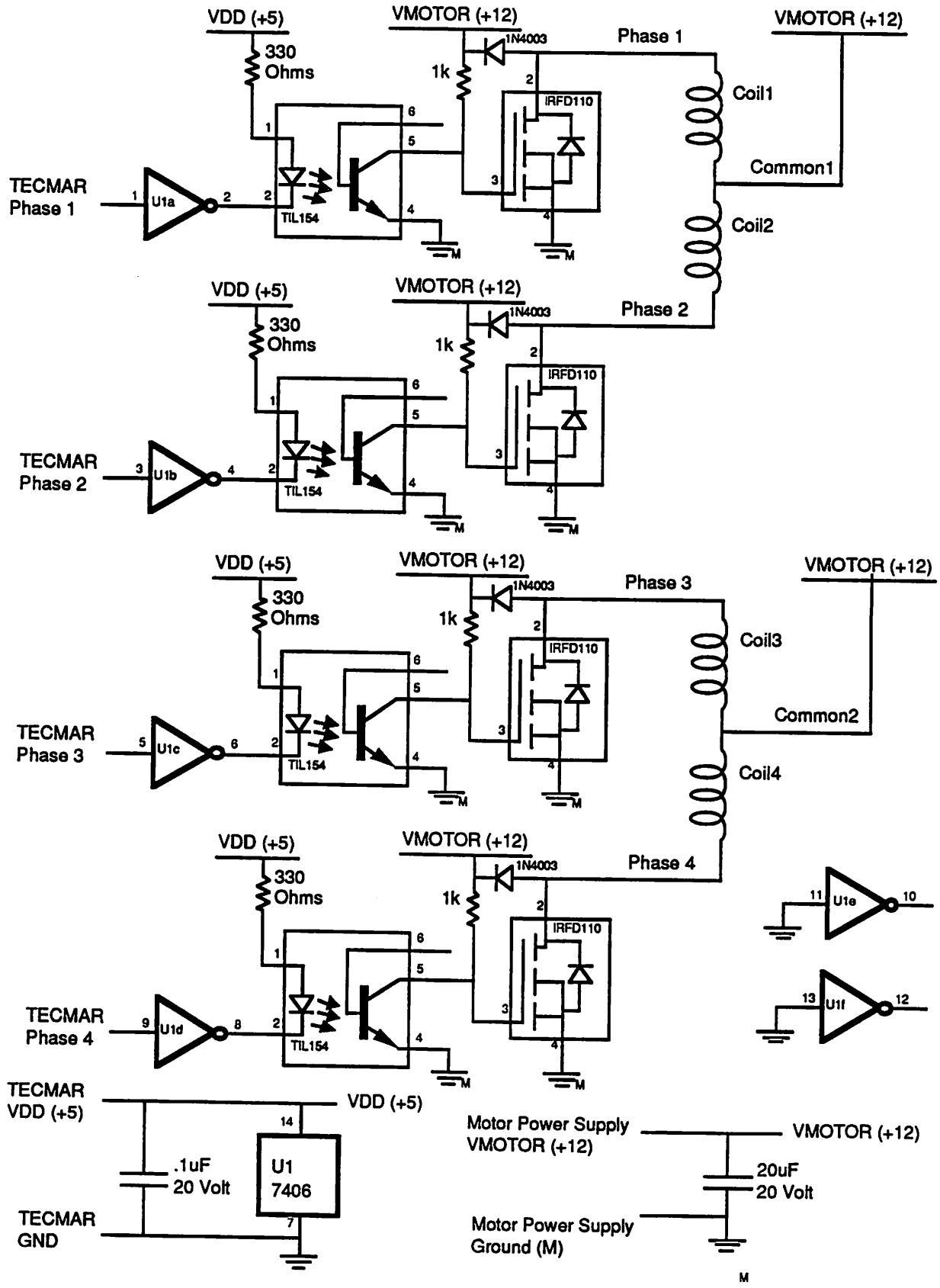


Figure 12. *Power Amplifier Board Circuit Diagram*

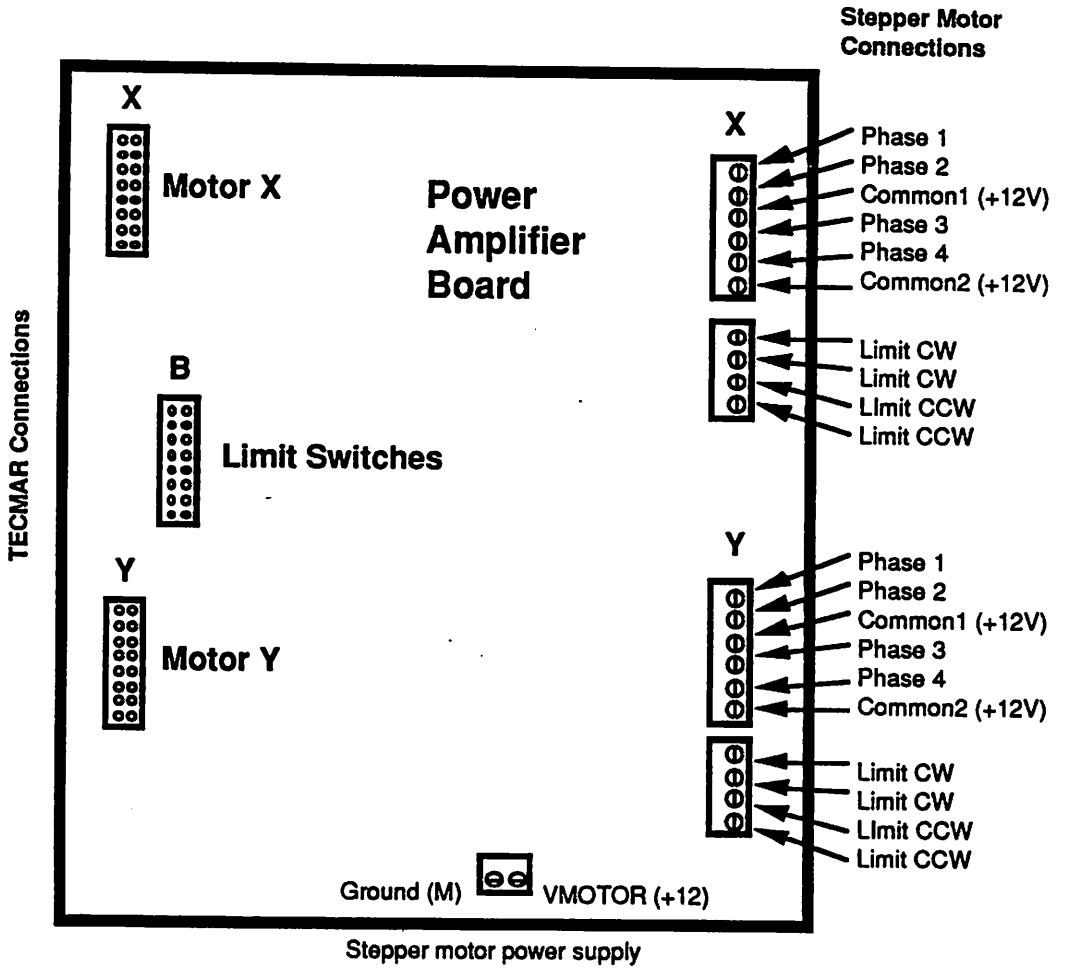


Figure 13. *Power Amplifier Board Diagram*

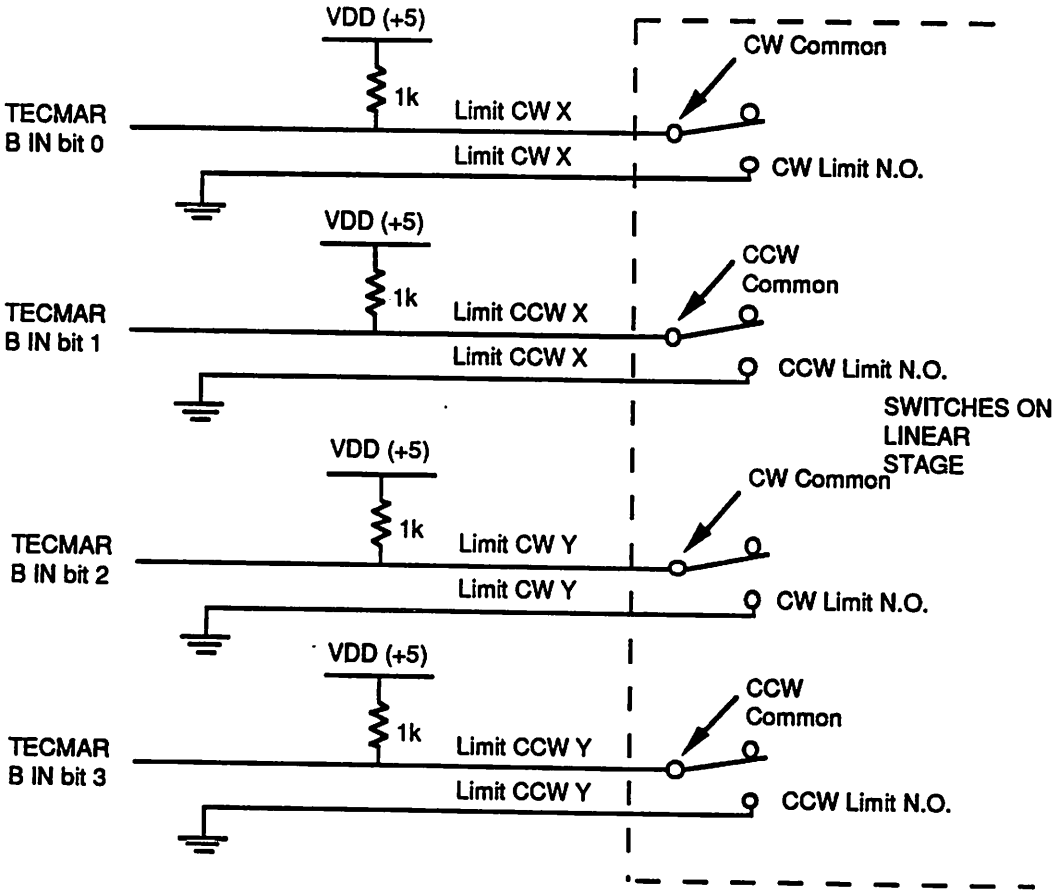


Figure 14. *Limit Switch Circuit on Power Amplifier Board*

Chapter 3
Software Users Guide

This chapter describes the operation of the Photoresist Characterization System Software. Details on the software routines and information for writing additional software are not provided in this chapter. Chapter 4 provides internal and programming related detail. This chapter discusses only the software's operation.

Executing the Program

The Exposure Characterization program is normally called "expose.exe" and is located in the "ecs" directory on the computer's C: drive. The program is executed by changing to the ecs directory (using the cd command) and entering "expose". In order to execute the expose program the PCLAB driver must be installed as a system device. The PCLAB driver acts as a link between the Characterization System Software and the DT2801 A/D system. The driver is installed by changing the config.sys file. A line containing "device=pcldrv.sys" is entered into the file. The pcldrv.sys file must be located in the root directory.

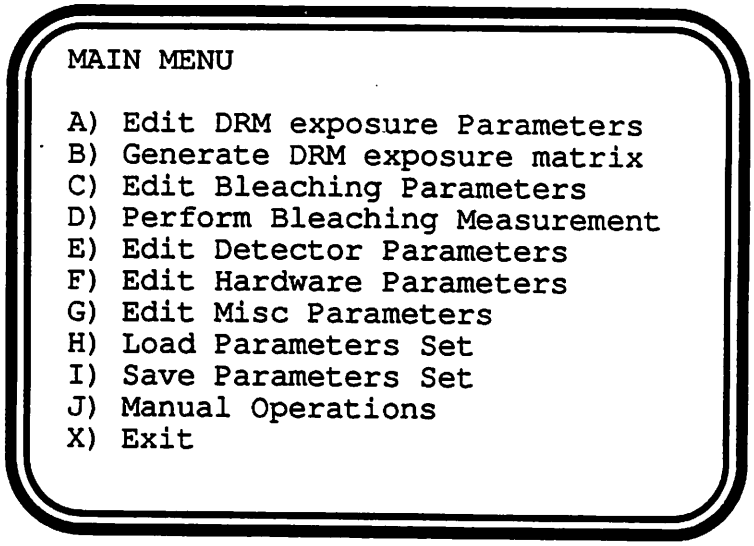
When executed the Exposure Characterization program will attempt to initialize the hardware subsystems. If any problems are discovered it will inform the operator. If the hardware is satisfactory (as far as the computer can tell) it will ask for the Experiment Session Data. The operator may enter two lines of descriptive data. The Experiment Session Data will be saved and/or printed with all the experiment data collected by the system. The Experiment Session Data can be changed from the Miscellaneous Parameters Menu (see page 29).

After entering the Experiment Session Data the system displays the main menu. The main menu and its options are described in the remainder of this chapter.

Main Menu

The main menu is shown on the screen below. The first two options are for creating matrices which vary the resist's exposure level over a specified range. The

samples may be used in the multi-channel dissolution rate monitor (DRM) to observe the development process with varying exposure dose. Option A is used to change the parameters associated with creating such samples. Option B actually fires the laser, moves the sample, and creates the exposure matrix. Options C & D are used for bleaching measurements. The sample may be exposed and data collected on transmittance and reflectance as the exposure proceeds. The data can be used to obtain ABC[1] parameters for the modeling. Options E, F, & G are used to edit parameters which are hardware and preference dependent. Options I & J will load and save complete parameter sets. All parameters are initialized to appropriate defaults at start up. The operator, however, can load and save parameter sets which he has created for particular experiments. Option J provides access to lower level functions via another menu (see the Manual Operations section in this chapter on page 30).



DRM Exposure Parameters Menu

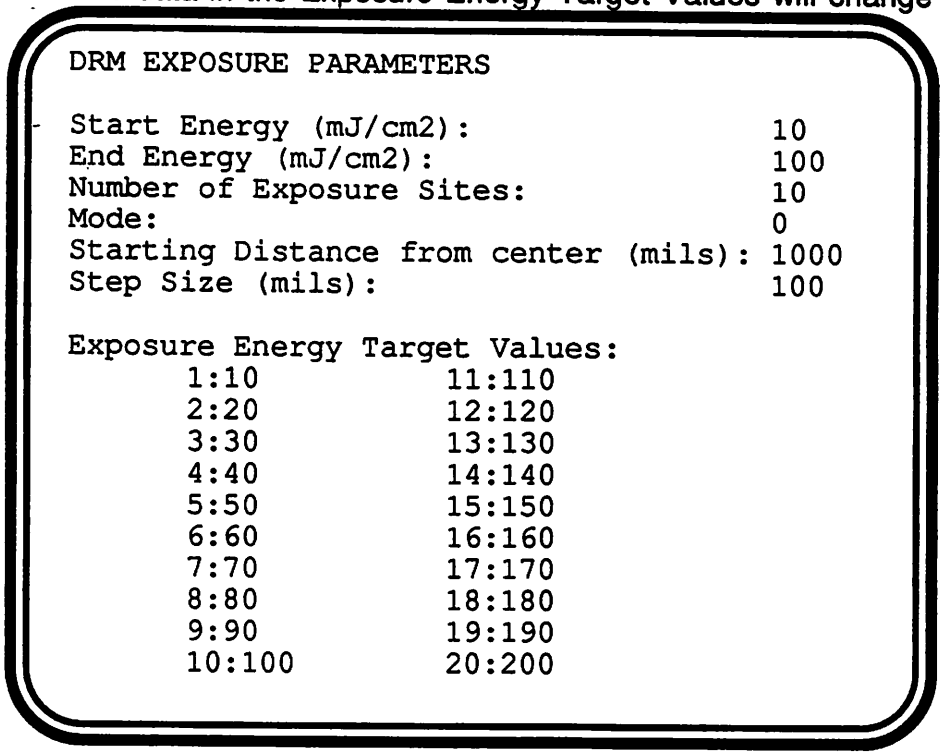
The DRM Exposure Parameters menu is shown below. The six parameters at the top determine the energy and position of each of the exposure sites. The parameters may be changed. Use the arrow keys to move the cursor around the screen. When the cursor is over a parameters value it may be changed. Press the F10 Function Key when done making changes. The PAGE DOWN and PAGE UP keys may be used to move the cursor left or right an entire field (the left and right cursor keys move the cursor within a particular field). The menu's operation depends on the Mode selection made on the menu. The Mode is used to select automated sequences of exposure energies. Three modes are possible:

0) Arithmetic Mode -- The Start Energy and End Energy will be used to calculate an arithmetic sequence spanning the Number of Exposure Sites.

1) Exponential Mode -- The Start Energy and End Energy will be used to calculate an exponential sequence spanning the Number of Exposure Sites.

2) The Start Energy and End Energy will be ignored and the values entered in the lower portion of the menu (Exposure Energy Target Values) will be used for the specified Number of Exposure Sites.

If mode 0 or 1 is selected the data in the Exposure Energy Target Values will change when the menu is exited (by pressing F10 or moving the cursor off the bottom). The system will calculate new Exposure Energy Target Values and ask the operator if they are acceptable. If mode 2 is selected the Exposure Energy Target Values will not be changed when the menu is exited.



```
DRM EXPOSURE PARAMETERS

Start Energy (mJ/cm2):                    10
End Energy (mJ/cm2):                      100
Number of Exposure Sites:                 10
Mode:                                     0
Starting Distance from center (mils):     1000
Step Size (mils):                         100

Exposure Energy Target Values:
      1:10          11:110
      2:20          12:120
      3:30          13:130
      4:40          14:140
      5:50          15:150
      6:60          16:160
      7:70          17:170
      8:80          18:180
      9:90          19:190
      10:100        20:200
```

The Step Size parameter controls the distance the linear stage will move between each exposure site. A large value can cause the stage to jam (by asking it to move further than is possible). Values up to 300 mils are reasonable. The Starting DIstance From Center specifies how far from stage center the system should begin the matrix exposure.

Generate DRM Exposure Matrix

Option B on the main menu will generate the exposure matrix. When option B is selected the DRM Parameter Menu will be displayed and the operator will be asked if the energies and parameters are acceptable. If the operator indicates the parameters are appropriate he is queried about the laser status. The laser must be on, in triggered mode, and have the burst count set equal to one. If the operator indicates the laser is ready he is warned that the laser will fire when a key is pressed. Goggles and other protective measures should be in place at this point. The operator is given an opportunity to abort. If he continues the software fires the laser and displays the energies measured by each detector. The laser is repeatedly fired until another key is pressed. The operator can use the continuous firing to adjust the optics until appropriate energies are measured. After the operator stops the firing, the system asks if the energies are acceptable. A positive reply obtains a request for wafer installation and a warning that the exposure will begin when a key is pressed.

The exposure process consists of the following steps: 1) Align and center the linear stage, 2) offset stage from center by the amount specified in DRM Parameter Menu, 3) open the shutter, 4) pulse the laser until the specified total energy is reached for the current exposure site, 5) move the stage to the next site (distance specified in DRM Parameter Menu), 6) repeat 4 & 5 until all sites have been exposed. The Exposure Target Energies, the actual energies, and the number of pulses are displayed while the exposure proceeds.

Once the exposure matrix is complete the system allows the operator to save and/or print the data. If the data is printed or saved the operator will be asked to enter two lines of data which describe the experiment. When the data is saved and/or printed these two lines, along with the Experiment Session Data (entered at start-up), and the date and time will be saved and/or printed too. The filename used to save the data must be specified by the operator. A file extension of ".drm" is automatically appended and the file is placed in the "C:\ecs\ecsdata" directory. The directory where the data is saved may be changed on the Miscellaneous Parameters Menu (see page 29).

Bleaching Parameters Menu

The Bleaching Parameters menu is shown below. Option C on the main menu selects the Bleaching Parameter menu. It contains four parameters which control the bleaching process. The menu may be traversed and changes made using the methods explained for the DRM Exposure Parameters Menu. The Maximum Dose is the bleaching process ending energy. The exposure site will be exposed until the integrated energy is equal to the Maximum Dose. The Energy per Measurement (high) initially controls the data collected during bleaching. The system keeps track of the total energy incident on the sample. When the total incident energy reaches the Energy per Measurement (high) value the system saves the total incident energy, reflected energy, transmitted energy, and pulse count. The totals are then zeroed and the process repeated until the grand total incident energy reaches the Measurement Density Drop Off Point. Once the Drop Off is reached data is saved using the Energy per Measurement (low) parameter. The process continues until the grand total incident energy reaches the Maximum Dose. This simple technique allows a high density of data points at the beginning, a large total dose, and keeps the number of data points within reason. WARNING -- The total number of data points may not exceed 2000. The system will stop taking data if the number of points exceeds 2000.

```
BLEACHING PARAMETERS MENU

Maximum Dose:                                100
Measurement Density Drop Off Point:          10
Energy per Measurement (high):               .1
Energy per Measurement (low):                1
```

Perform Bleaching Measurement

The Perform Bleaching Measurement function exposes the sample and saves the energy measurement data. The operator is queried about the status of the laser. The laser must be on, in triggered mode, and have its burst count set to one. If the operator indicates all of the above are true he is warned that the laser will fire when he presses a

key. Goggles and other protective measures should be in place at this point. If the operator continues the system repeatedly fires the laser and displays the energies measured by each detector. The system stops when another key is pressed. The operator can use the continuous firing to adjust the optical set-up until appropriate energies are observed. After the operator stops the continuous firing, the system asks if the energies are acceptable. An affirmative reply obtains a request for wafer installation and a warning that the exposure will begin when a key is pressed.

The shutter will open and the exposure will begin. The sample will be bleached using the parameters set on the Bleaching Parameter Menu. Data will be collected and stored in memory. Once the bleaching is complete the system allows the operator to save and/or print the data. If the data is printed or saved the operator is asked to enter two lines of experiment description. When the data is saved and/or printed these two lines, along with the Experiment Session Data (entered at start-up), and the date and time will be saved and/or printed too. The filename used to save the data must be specified by the operator. A file extension of ".ble" is automatically appended and the file is placed in the "C:\ecs\ecsdata" directory. The directory where the data is saved may be changed on the Miscellaneous Parameters Menu. If the file already exist the operator is given an opportunity to append, overwrite, or abort.

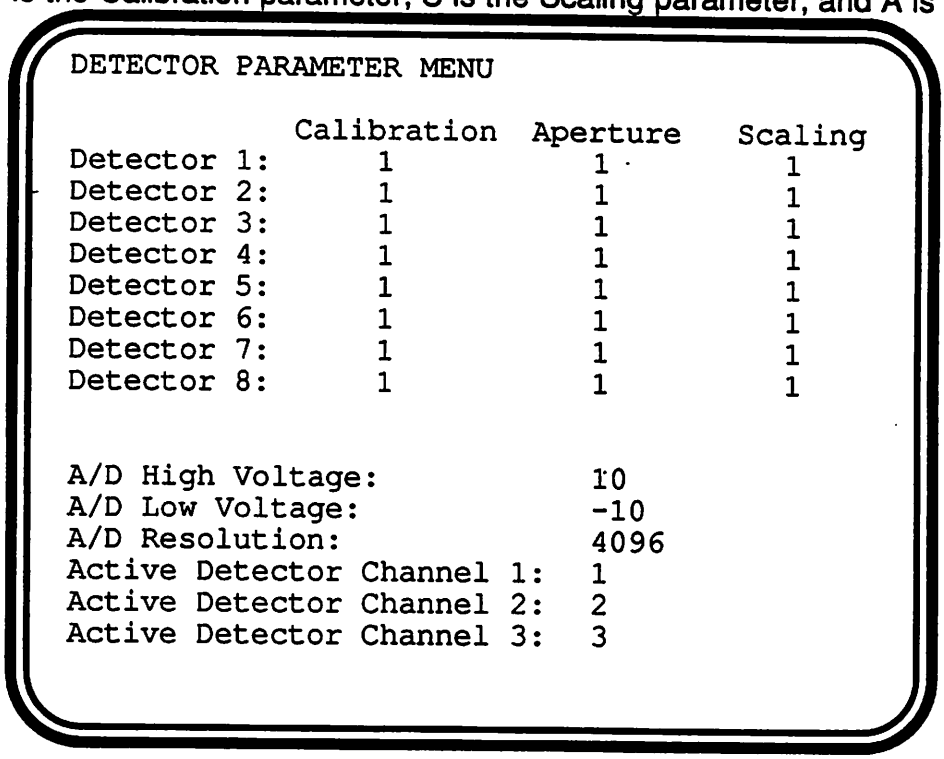
Detector Parameter Menu

The Detector Parameters control the operation of the DT2801 A/D system and the conversion from voltage to energy. The Detector Parameter Menu is shown below. Each of the eight possible detectors has 3 constants associated with it. Calibration is the milliJoules per volt of the detector hardware. This value depends on the electronics of the detector and is obtained by comparing the detector to an absolutely calibrated Joulemeter. Aperture is the amount of area (cm^2) which the detector receives light energy from. These two parameters allow the calculation of milliJoules per cm^2 (energy density) from the DT2801 digitized voltages. The scaling constant is used to account for beam splitters and the like. For example the detector used for incident energy measurement

typically receives only 50% of the incident energy. The Scaling parameter would be set to 2 to account for this. The equation used to obtain detector energy is:

$$E = \frac{(V)(C)(S)}{A}$$

Where E is the detector energy density returned, V is the voltage from the DT2801 A/D conversion system, C is the Calibration parameter, S is the Scaling parameter, and A is the Aperture parameter.



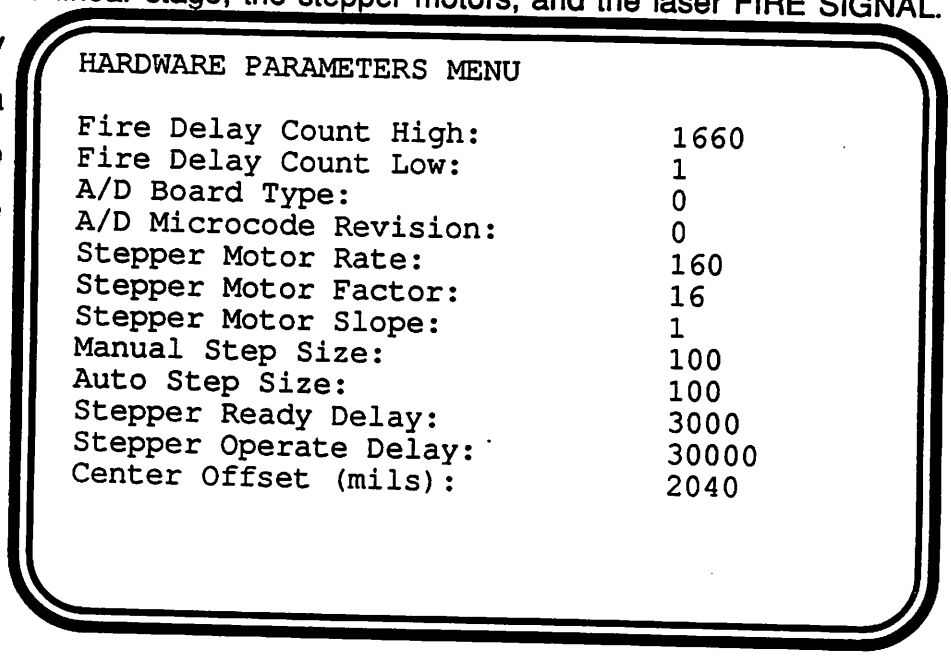
```
DETECTOR PARAMETER MENU

                Calibration  Aperture   Scaling
Detector 1:         1           1          1
Detector 2:         1           1          1
Detector 3:         1           1          1
Detector 4:         1           1          1
Detector 5:         1           1          1
Detector 6:         1           1          1
Detector 7:         1           1          1
Detector 8:         1           1          1


A/D High Voltage:                10
A/D Low Voltage:                 -10
A/D Resolution:                  4096
Active Detector Channel 1:       1
Active Detector Channel 2:       2
Active Detector Channel 3:       3
```

The three A/D parameters listed after the detector parameters are hardware dependent. The two voltages define the voltages which correspond to the minimum and maximum values returned by the DT2801. The A/D Resolution defines the number of discrete values the A/D system has over the specified voltage range. The DT2801 voltage is calculated using these parameters.

The remaining three parameters, the Active Detector Channels (1,2,3), define which of the eight detector channels are used for energy measurement. The first detector specified is the incident energy detector, the second is the transmitted energy, and the third is the reflected energy.

Hardware Parameters Menu

The Hardware Parameters Menu is shown below. The Hardware Parameters control the operation of the linear stage, the stepper motors, and the laser FIRE SIGNAL. The parameters may be changed in a manner identical to that explained in the DRM Exposure Parameters section. The Fire Delay Count High and the Fire Delay Count Low control the shape and size of the pulse used to fire the laser. See Chapter 2 -- Laser Fire Control for details (page 7). Fire Delay Count High is how long the pulse will remain high. The Fire Delay Count Low is used to slow the rate at which the laser fires (never use the High parameter for this because it will place the laser in pre-charge for an extended period of time). The FIRE SIGNAL remains low for a time determined by the sum of the Fire Delay Count Low and the time needed to collect and process the detector data. The two counts are simply the number of times a null loop is executed to create the delay. The IBM PC-AT environment does not provide an absolute time delay. All delays will depend on the CPU clock rate.



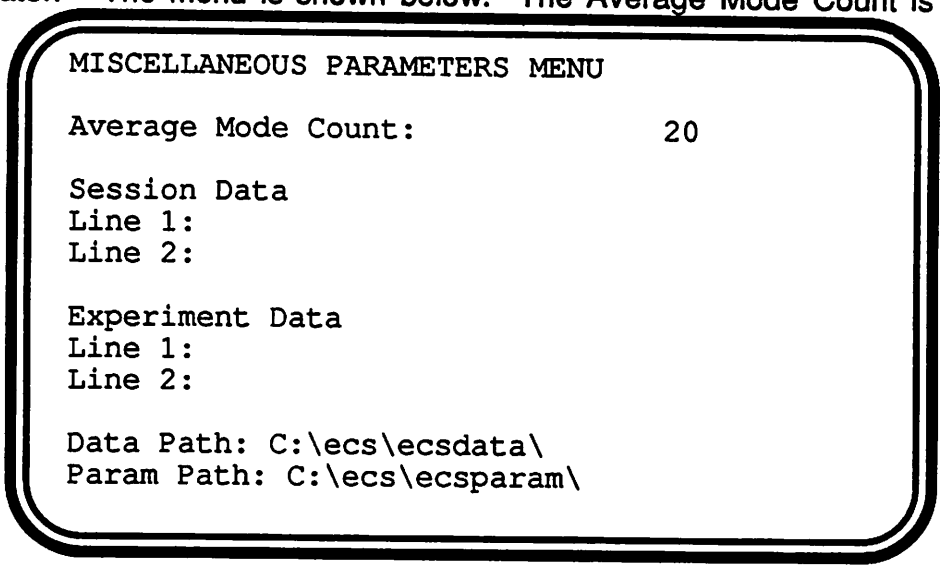
The A/D Board Type and A/D Microcode Revision parameters are set by the software. They are not currently utilized but provide information for the operator.

The Stepper Motor Rate is the speed at which the stepper motors are driven. The Stepper Motor Factor is a control parameter for the TECMAR Stepper Motor Controller Board. See the TECMAR manual for details. The Stepper Motor Slope is the rate at which the stepper motor is accelerated to full speed. The Manual Step Size defines how many steps the system will make when told to "Step Left" or "Step Right" from the Manual Operation Menu. The Auto Step Size defines how many steps will be made in a

single operation when homing the linear stage. A value larger than 100 may cause the stage to jam. The Stepper Ready Delay and Stepper Operate Delay parameters account for the finite amount of time it takes the stepper motor to operate. They control internal delays. The Center Offset (mils) is the distance from the limit switch to the center. When the system attempts to home and center the stage it moves the stage until a limit switches closes. It then moves the stage the amount specified by the Center Offset in order to obtain the center position.

Miscellaneous Parameter Menu

The Miscellaneous Parameter Menu contains more parameters which may be changed by the operator. The menu is shown below. The Average Mode Count is used by the Average Fire Mode on the Manual Operations Menu. It sets the number of laser pulses which be be averaged (2000 maximum). The Session Data and Experiment Data are the 4 lines of text used to describe data whenever it is saved or printed. The Session Data is entered when the system is started. It can be changed on this menu. The Experiment Data is reentered every time data is saved or printed. It can also be changed on this menu. Either set of data may be defaulted to its current value rather than reentered. The Data Path is the MSDOS Path used to store all data files. The Param Path is the MSDOS Path used to save all parameter files. The paths may be changed using this menu. Their default values are shown above.

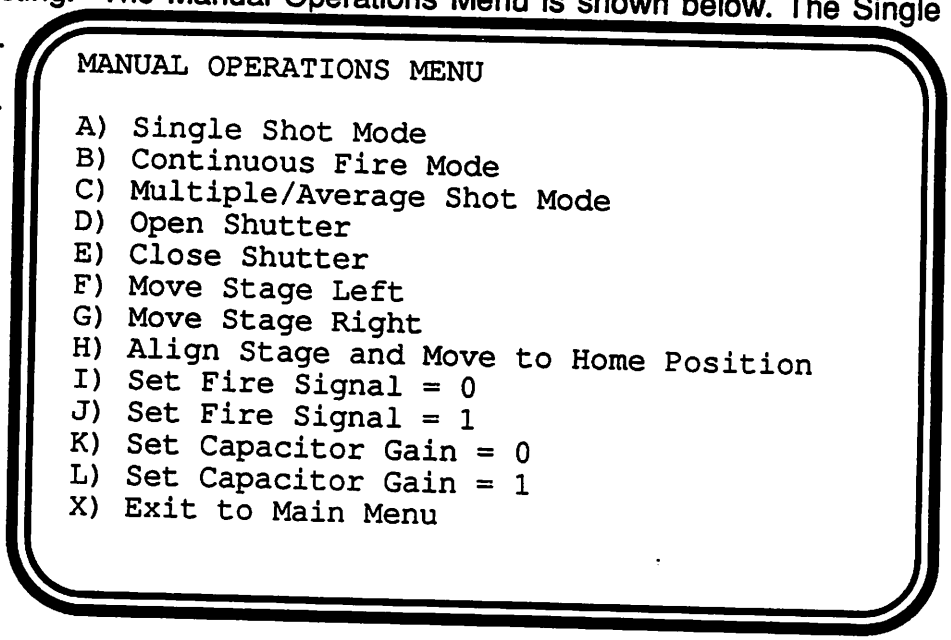


Load and Save Parameter Set

Parameter sets consist of the data on all the aforementioned menus. They can be loaded and saved to facilitate different experimental set ups. When either option is requested the operator must enter a filename. The entered filename will have the extension ".ecf" added.. The system will then use the directory specified in the Param Path to either load or save the parameters.

Manual Operations

The Manual Operations Menu contains functions useful for the calibration of the system and development of new experiments. It also contains functions which assist optical alignment and testing. The Manual Operations Menu is shown below. The Single Shot Mode is entered by selecting A. In Single Shot Mode every press of the space bar fires the laser and displays the measured energies. Any other key exits the mode. Continuous Fire Mode is entered by selecting B. It fires the laser repeatedly until a key is pressed. Multiple Average Shot Mode fires the laser the number of times specified on the Miscellaneous Parameter Menu. The energy data and averages are displayed and may saved or printed. Open Shutter and Close Shutter open and close the shutter. Move Stage Left and Move Stage Right move the stage the number of steps specified in the Manual Step Size (see Hardware Parameter Menu, page 28). Align Stage and Move to Home Position is self descriptive. The Set Fire Signal commands are used for adjusting the potentiometers in the Integrator Units (see Hardware Chapter). The Set Capacitor Gain options switch the relays in the Integrator



Units to obtain a different integration capacitor and therefore a different gain. These functions should be used with caution because the energy calculations will be in error if the Detector Calibration parameters are not modified.

[1] F.H. Dill, W.P. Hornberger, P.S. Hauge, and J.M. Shaw, "Characterization of Positive Photoresists," IEEE Transactions on Electron Devices, Vol ED-22, No. 7, 1975.

Chapter 4
Software Internals

The Software Internals chapter details the "C" programs and the development environment of the Characterization System. This chapter contains information for operators wishing to change or write "C" programs related to the Characterization System.

Development Environment

An IBM PC-AT compatible was used for all development work. Microsoft C version 5.0 compiled all the programs. Microsoft C was chosen because of its ANSI C Standard compatability, its completeness, and its widespread acceptance. Two specialized libraries were utilized to augment the standard Microsoft libraries. The first library was PCLAB from Data Translation. PCLAB is a collection of C calls which provide a simple interface between the DT2801 A/D converter and the C language. The second library was Windows For Data (WFD) from Vermont Creative Software. WFD is a collection of routines for window management. It will be difficult to examine any of the expose related programs without the WFD documentation[1].

Module and Library Names

The program modules and libraries needed for the complete Characterization System are listed below:

pcc4llib.lib	Library of the PCLAB software package. Provides C calls for A/D conversion.
pcldrv.sys	Device driver which must be installed in order for the above library to function.
wfdl.lib & wfcl.lib	Libraries for Windows For Data (WFD).
expose.mak	Make file for automatically compiling and linking the Characterization System Software.
expose.c	The expose program. Defines global variables and contains main().
expconf.c	Global variable definitions. Included in most modules to define the global variables.

1. Windows For Data Reference and Windows For C Reference (version 2.1), Vermont Creative Software, Richford, VT, 1987.

wfd.h	Include file needed for Windows For Data. Included in all modules which make WFD calls. Note that this file includes other files supplied with the WFD system.
pclerrs.c	File supplied with PCLAB which contains ASCII descriptions of errors. Included to describe errors (if they occur).
expmenu.c	C Module which contains all menu operations.
expfunc.c	C module which contains all major exposure functions.
filer.c	C module which performs all file related operations.
graphics.c	C module which performs all graphics.
query.c	C module which contain sub-routines to query the operator.
laser.c	C module which fires the laser and reads the detectors.
stepper.c	C module which controls the Linear Stages via the TECMAR Stepper Motor Controller Board.

Additional standard include files are used from the Microsoft C libraries. These are all ANSI standard include files. The following sections will describe the sub-routines and some of the variables in each of the modules. Listings for all modules are given in Appendix C. Appendix B contains global variable descriptions.

Variable Standardization

All variables used in the program include an extension which defines their type. The conventions used are shown in the table below. The STANDARD column contains the extension used for normal variables. The POINTER column contains the extension used for a pointer to a variable.

VARIABLE TYPE	STANDARD	POINTER
char	_c	_pc
constant char	_C	_PC
int	_i	_pi
short	_s	_ps
long	_l	_pl
unsigned char	_uc	_puc
unsigned	_u	_pu
unsigned short	_us	_pus
unsigned long	_ul	_pul
float	_f	_pf
double	_d	_pd
long double	_e	_pe
window	_w	_pw
boolean	_b	_pb

form	_fm	_pfm
field	_fd	_pfd
memory file	_mf	_pmf

Expose.c

Expose.c is the main program. It defines general global variables and initializes the configuration structure (config_u). The configuration structure type (config_type_union) is defined in expconf.c. This structure is loaded and saved by the parameter load and save options on the main menu.

Expose.c contains only a single routine, main(). Main() performs initialization functions and then calls the menu() function. Menu() displays the main menu and executes the functions requested by the operator using WFD calls.

Expconf.c

Expconf.c contains the configuration structure definition. Expconf.c defines a union type (config_type_union). This type must be used in the including module to define the structure as either internal or external. Expose.c declares the union as internal (defines it). All other modules declare it as external. The variables included in the union are listed below with a brief description. All the variables appear on one of the parameter menus.

Variable and Type	Description
long fire_delay_counth_l	Fire Delay Count High
long fire_delay_countl_l	Fire Delay Count Low
int average_count_i	Average Mode Count
double high_voltage_d	A/D High Voltage
double low_voltage_d	A/D Low Voltage
double noc_i	A/D Resolution
int board_type_i	A/D Board Type
int microcode_i	A/D Microcode Revision
double det_calib_pd[8]	Detector Calibration Values
double det_apera_pd[8]	Detector Aperture Values
double det_scale_pd[8]	Detector Scale Values
long toggledelay_i	not used
int detchannel_i[8]	Detector Channels in use
double startenergy_d	DRM Start Energy
double endenergy_d	DRM End Energy
int sequence_i	DRM Mode

int steps_i	DRM Number of Exposure Sites
double targetenergy_pd[21]	DRM Exposure Target Energies
int stepperrate_i	Stepper Motor Rate
int setpperfactor_i	Stepper Motor Factor
int steppersiope_i	Stepper Motor Slope
int manualstepsize_i	Manual Step Size
int autostepsize_i	Auto Step Size
int readydelay_i	Stepper Ready Delay
long stepperdelay_i	Stepper Operate Delay
int centeroffset_i	Center Offset
int drnstart_i	DRM Starting Distance From Center
int drmstepsize_i	DRM Step Size
double elow_d	Bleaching Energy Per Measurement Low
double ehigh_d	Bleaching Energy Per Measurement High
double maxdose_d	Bleaching Maximum Dose
double measdens_d	Bleaching Measurement Density Drop Off Point

Expmenu.c

Expmenu.c contains the menus and two functions to control the window environment. Initwindows() is called by main() to initialize the WFD package. Endwindows() is called by main() prior to exiting the program to close all windows. Menu() is called by main() once and does not return until the operator selects the exit option. Menu() displays the main menu and requests the operator's selection. Based on the selection it executes a call. Manualmenu() is called from menu(). It displays an additional menu containing more options and requests the operator's selection again. Based on the selection it also executes a call.

Editdetparams() is called from menu() to edit the Detector Parameters. The Detector Parameters are displayed and the operator may modify them. Editdetparams() returns to menu(). Edithardparams() functions as editdetparams() except the operator may change the Hardware Parameters. Editdrmparams() is used to edit the DRM Exposure Parameters. Editdrmparams() may call calcenergy() to calculate the target energies if an automatic mode is selected by the operator. Editabcparams() allows the operator to change the Bleaching Parameters. Editmiscparams() allows the operator to change the Miscellaneous Parameters. All editing is accomplished using WFD calls.

Exitmenu() is called either by menu() or by manualmenu(). It tells the calling function to return up a level in the menu system. Exitmenu() will cause Manualmenu() to re-

turn to menu() and menu() to return to main().

Calcenergy() is called by editdrmparams() to calculate the Exposure Target Energies. The Exposure Target Energies are calculated based on the DRM Exposure Parameter settings. Either an arithmetic or a exponential sequence is inserted into the Exposure Target Energy parameters. The routine verifies that the sequence is of finite length.

Stepleft1(), stepright1(), and findhome1() may be called from manualmenu(). They call their more general counterparts (stepleft() and stepright()) to perform the action requested. Stepleft1() steps stage one left by the Manual Step Size. Findhome1() performs a homing and centering operation on stage one. These functions are necessary because the WFD menu system will not allow parameter passing.

Expfunc.c

Expfunc.c contains the major exposure functions. Routines to perform bleaching, fire the laser, and perform drm matrix generation are included. Most of the functions are called directly from menu() or manualmenu().

Abcexpose() performs a bleaching measurement. Several variables are established by this routine which include:

Variable	Description
int pointcount_i	maintains the current number of data points
FILE *outfile	file pointer used if the operator saves data to a file
char filename_pc[80]	filename data will be saved into
double totalenergyi_d	total incident energy during bleaching
double totalenergyr_d	total reflected energy during bleaching
double totalenergyt_d	total transmitted energy during bleaching
double energyi_d	incremental incident energy during bleaching
double energyr_d	incremental incident energy during bleaching
double energyt_d	incremental incident energy during bleaching

Abcexpose() calls laserready(), testfire(), and genmessage() to verify that the laser is ready, that the laser energy is acceptable, and that the wafer is loaded. If everything is correct it opens the shutter and begins pulsing the laser. Two WHILE loops pulse the laser, one for the Measurement Density (high) and one for Measurement Density (low).

Data returned from pulsetoenergy() is displayed and saved in 4 arrays. The shutter is closed and the data is graphed by graphbleachdata(). Savedata() is called to save the bleaching data.

Testfire() is called by several different routines. It warns the operator that the laser is about to fire and requests that he press a key to begin. After the keypress it fires the laser and displays the energy measured. It uses readdet() to read the detector energy data and printdetdata() to display it. It asks the operator if these energies are acceptable and returns a TRUE if the operator indicates they are. It returns a FALSE if the operator indicates they are not. The return value is typically used to determine if a process should proceed (such as in abcexpose()) or abort.

Fireone() is called from the manualmenu() for single shot mode. It verifies that the laser is ready and then fires the laser every time the space bar is pressed. It uses the readdet() routine to read the detector data and printdetdata() to display it.

Fireaverage() is called from manualmenu() to fire the laser a specified number of times, record the energies, and calculate the averages. The number of times it fires are passed as a parameter (pulsecount_i). A FOR loop executes pulsecount_i times to fire the laser. Firelaser() is called to fire the laser and readdet() is called to obtain the energies. Data from each pulse is stored in an array and also displayed for the operator. The percent transmittance and percent reflectance are calculated and displayed too. A running energy total (total_pd[3]) is maintained for each detector. Once the laser has fired the specified number of times, averages are calculated from the totals and the number of pulses. The operator may save or print the data (savedata()).

Firecont() is similar to fireone() except it fires the laser continuously instead of waiting for the space bar to be pressed. When a key is pressed it stops firing.

Drmexpose() creates an exposure matrice. It calls editdrmparams(), laserready(), testfire(), and genmessage() to verify that the parameters are acceptable, the laser is ready, the laser energies are acceptable, and the wafer is loaded, respectively. If all the above are fine it proceeds. The stage is homed (findhome()) and then moved to the start position (stepleft(), distance determined by Starting Distance From Center). It then sequences creating the specified number of exposures. Pulsetoenergy() is called with the appropriate energy value and the actual energies returned by Pulsetoenergy() are

saved. Stepright() is called to move the stage to the next exposure site. After the specified number of exposures have been created the operator may save the data (savedata()).

Filer.c

Filer.c contains the routines for handling file input, file output, and printing. Loadparams() is called by main() to load the parameter file. It changes the directory to ecsparam_path_pc and requests a filename from the operator using getfilename(). It concatenates filepost_ecf_PC (this is normally ".ecf") to the filename end and attempts to read the parameter file using readparamfile(). If an error occurs it reports the problem.

Readparamfile() is called by loadparams() with a filename (filename_pc). It attempts to open the file (open()), read the data (read()) and close the file (close()). If an error occurs at any stage it reports the problem using fileerror(). The data read is the structure config_u which is of type config_type_union defined in expconf.c.

Saveparams() is called by main() to save a parameter file. It changes directories and requests a filename similar to loadparams(). It checks the directory (directory()), checks the file (checkfile()), and calls writeparamfile() to write the data if the checks are passed.

Writeparamfile() is called by saveparams() to write the data. The filename is passed to writeparamfile() via filename_pc. It attempts to open the file (open()), write the data (write()) and close the file (close()). If an error occurs at any stages it reports the problem using fileerror(). The data written is the structure config_u. Config_u is of type config_type_union which is defined in expconf.c.

Checkfile() is called with a three parameters. The first parameter is the window title (title_pc), the second is the filename to be checked (path_pc), and the third is a boolean control variable controlling concatenation (concat_b). Concat_b is only important if the file exists when the check is performed. Checkfile() attempts to open the file. If the file does not exist it quits and returns a 1 to the calling routine. If the file exists the operator is informed so and asked if the file should be overwritten or the operation aborted. If concat_b is TRUE the operator is offered the option of concatenating to the file's end.

The returned value is 1 if the file does not exist or the operator selects overwrite. The returned value is 2 if the operator selects concatenate. The returned value is 0 if the operator aborts. Querychar() is used to question the operator.

Savedata() is called from all functions which collect data and give the operator a chance to save or print it. Five parameters are passed to savedata(): a file pointer (outfile), the experiment title (experimenttit_pc), two description lines (experiment1_pc, experiment2_pc), and a filename extension (filepost_pc). Savedata() uses querychar() to ask the operator if he wants to save or print the data. If the operator selects F to save the data savedata() requests a filename using getfilename(). Filepost_pc is appended to the the filename. Savedata() performs a checkfile() on the filename If the operator selects P to print the data the filename request is bypassed and a filename of "PRN" is automatically used (this will cause MSDOS to print on the printer). The file is opened (append mode may be observed if checkfile() indicated this was necessary) and getexpdata() is called to request the experiment description from the operator. The experiment title, date, time, description, and session description are all written to the file (or printed). Savedata() returns a 1 if a file is open. This indicates to the calling routine that it should write its data and close the file. If it returns a zero the operator indicated he did not want to save or print. Typically savedata() is repeatedly called until the operator quits (thus allowing the operator to print and save the same data). Savedata() does not actually write the data to the file. The calling program must do this. This task division allows the calling routines to use different data formats.

Directory() is passed a pointer to a string variable (path_pc). It attempts to change directories to path_pc. If an error occurs it uses genmessage() to report the problem.

Graphics.c

Graphics.c contains only four routines. Initgraphics() is called by main() to initialize the graphics system at start up. Graphicson() is called to turn the graphics screen on. Graphicsoff() is called turn the graphics screen off and return to normal text display.

Graphbleachdata() is called by abcexpose() to graph the bleaching data on the screen. This allows the operator to view the data before saving or printing it. It turns the graphic screen on (graphicson()) and uses Microsoft library routines to create a sim-

ple x-y graph. It plots %transmission versus total incident energy and labels the graph.

Laser.c

Laser.c contains functions related to controlling the laser and the DT2801 interface. Initpclab() is called by main() to initialize the DT2801 and the PCLAB subroutine library. It verifies that the DT2801 card is installed and enables the digital I/O section for output. If an error occurs it displays the PCLAB error number using errorwno().

Readdet() is the routine which reads and calculates the energies observed by the detectors. Readdet() is a passed a pointer to an array of doubles (det_pd). Det_pd is updated to the energy values read from each detector. The three detector channels are read using the ADC_VALUE() function of PCLAB. After the channels are read energy calculations are performed to obtain the values for det_pd. A check is made on the gain utilized for the A/D conversion. If the values are out of the ideal resolution range the gain for the next conversion is changed. The dynamic gain scaling of the A/D converter yields constant resolution even at low voltage readings.

Printdetdata() is called to print a single line which contains the detector energy readings. The routine is used by almost all the routines which display energy data.

Openshutter() and closeshutter() output a digital value on the DT2801 digital I/O port to open or close the shutter. Gain0() and gain1() are called by manualmenu() to change the capacitor gain. Firesignal0() and firesignal1() are called by manualmenu() to change the FIRE SIGNAL. The routines simply toggle a digital output bit on the DT2801

Laserready() is called by functions to query the user about the laser's status. It uses querybool() to verify that the laser is on, in triggered mode, and has the burst count set equal to one. It returns the operators reply.

Firelaser() is called to fire the laser. It uses the Fire Delay Count (low) to wait in a loop. After the loop is complete it changes the FIRE SIGNAL to a high value to begin pre-charge. It uses Fire Delay Count (high) to wait in another loop. After the second loop is complete it changes the FIRE SIGNAL to a low value.

Pulsetoenergy() is passed 5 variables. The target energy (target_d) is the only one used by the routine. The other four are used to return data to the calling routine. Pulsetoenergy() uses firelaser() and readdet() to fire the laser until the specified total

energy (target_d) is reached. It returns the actual total incident energy (totali_pd), the total transmitted energy (totalt_pd), the total reflected energy (totalr_pd), and the pulse count (pulsecount_pl). The returned values are determined by summing the energies of each pulse.

Query.c

Query.c contains routines for displaying data and obtaining data from the operator. These routines are used by many of the functions in the Characterization System Software.

Querybool() queries the operator on a yes/no question. It creates a window, displays the question, waits for an answer, and returns the answer to the calling routine.

Querychar() operates as querybool, however, it allows a single character response.

Querybool2() has a slightly different display format than querybool() but functions the same.

Fileerror() is called when a file related error occurs. It uses genmessage() to display the error number and error message.

Errorwno() is used to display a general error message which contains a number. It converts the number to a string and then uses genmessage() to display the error message.

Genmessage() is passed a window title (errormesstit_pc) and three lines of information (errormess1_pc, errormess2_pc, and errormess3_pc). It creates a window with the title and displays the three lines. It asks the operator to press a key to continue. It erases the window and returns.

Getexpdata() is called with four parameters. It creates a window with a title (qtit_pc) and asks the operator to enter two lines of ASCII data (qvar1_pc and qvar2_pc). It describes the requested lines using qdescription_pc. This function is called to obtain the Session Description Data and the Experiment Description Data.

Getfilename() is called whenever a filename is requested. It asks the operator to enter a filename and returns the result (reply_pc) to the calling routine. It allows only 8 characters to be entered and titles the window with qtit_pc. Q1_pc is used to describe the requested filename.

Stepper.c

Stepper.c contains the functions related to stepper motor control and stage movement. It also translates from CCW and CW to left and right motion.

Initstepper() initializes the TECMAR Stepper Motor Controller Board and the two CY512s located on the board. The commands sent to the CY512s and their functions will not be detailed here. The interested reader is referred to the CY512 databook[2] and TECMAR manual[3]. Some documentation is supplied in the Characterization System Software. Stepperok_b is set TRUE to indicate that the stepper motors are functioning. Stepright() and stepleft() are called to move the stage and verify they really are functioning. Stepright() and stepleft() will set stepperok_b to false if any problems are discovered. If stepperok_b is false all stepper related routines will be skipped.

Stepclockwise() is called to change the direction to clockwise. The routine is passed the motor number (motor_i). It sends commands to the appropriate CY512 using dochara() to change the direction. It sets the variable stagedirection_i to one, indicating clockwise.

Stepcclockwise() is called to change the direction to counter-clockwise. The routine is passed the motor number (motor_i). It sends commands to the appropriate CY512 using dochara() to change the direction. It sets stagedirection_i to two, indicating counter-clockwise.

Stepleft() is called to step the stage left a specified stepsize. The routine is passed the motor number (motor_i) and the stepsize (stepsize_i). If stepperok_b is FALSE (ie. the stepper motors are not functioning) the routine performs no actions. If stepperok_b is TRUE the routine proceeds. The routine checks the stage direction. If it is correct it continues. If the stage direction is incorrect it calls stepclockwise() to change it. Stepleft() checks the limit switch on the stage. If the limit switch is open it proceeds with a series of commands to move the stage. If the limit switch is closed (0) it does nothing. After moving the stage it updates the stage position variable (stageposition_i). Stepright() functions identically to steplef()t except in the other direction.

2. Cybernetic Micro Systems CY512 Intelligent Positioning Stepper Motor Controller, Cybernetic Micro Systems, Inc. P.O. Box 3000 San Gregorio, CA 94074, (415) 726-3000, 1983
3. TECMAR IBM Stepper Motor Controller, TECMAR, 1982

Findhome() is called by the drmexpose() and manualmenu() functions. It puts the stage through a series of action in order to center it. If the stage has been homed previously the function will simply execute movehome() to center the stage. If the function has not been executed before it moves the stage to the left using stepleft() until the limit switch closes. It uses the step size set by the Auto Step Size parameter (Hardware Parameters Menu). When the limit switch closes the routine moves the stage to the right two Auto Step Size parameters and then moves back to the left using a stepsize of one mil. This finds the left edge very accurately. It then moves the stage to the center using stepright() with the Stage Center Offset Parameter.

Movehome() uses the current position of the stage (stageposition_i) to either stepright() or stepleft() in order to bring the stage to position zero (the center).

Dochara() outputs a single character command to the CY512 Stepper Motor Controllers on the TECMAR board. It is passed a motor number (motor_i) and a character (databyte_uc).

Checkready(), waitstepper(), and delaystepper() are used to control communication timing with the TECMAR board.

Controlling The FTIR System

No software functions currently utilize the FTIR system. This, however, can easily be implemented. All FTIR operations can be controlled by execution of DOS like commands with parameters[4]. These could be directly executed from the Characterization System Software using one of the Microsoft C library routines: "exec", "spawn", or "system". The call needed will depend on the desired operation.

4. RFX-65 User's Guide, Analect Instruments, Irvine, CA, 1988

Appendix A

Vendors

Analect Instruments, Inc
17819 Gillette Avenue
Irvine, CA 92714
(714) 660-8801
RFX-65 Fourier Transform Infrared Spectrometer with 0.05 wavenumber resolution, High Power IR Source, and MCT detector.

Cybernetic Micro Systems, Inc.
P.O. Box 3000
San Gregorio, CA 94074
(415) 726-3000
Stepper motor controller chips (these were actually supplied by TECMAR)

Daedel, Inc
P.O. Box G
Sandy Hill Road
Harrison City, PA 15636-4451
(800) 245-6903 (412) 744-4451
Linear Stage model 106041P-10E

Data Translation, Inc
100 Locke Dr.
Marlboro, MA 01752
DT2801-A IBM/PC/XT/AT A/D and I/O Board HIgh Speed two channel
PCLAB subroutine library for Microsoft C version 5.0
DT707 Screw Terminal Panel for DT2801-A board

Ealing Electro-Optics
22 Pleasant Street
South Natick, MA 01760
(617) 651-8100
22-8411 Electronic Shutter, 35mm Aperture
2-8445 Electronic Shutter Pin Mount, 35mm Shutter

Hamamatsu
2444 Moorpark Avenue
Suite 312
San Jose, CA 95128
(408) 292-8603
PIN Silicon Photodiodes S1722-02 with Fused SIlica Windows and High UV sensitivity

Melles Griot
1770 Kettering Street
Irvine, CA 92714
(714) 261-5600
Optical components

Newport Corporation
P.O. Box 8020
18235 Mt. Baldy Circle
Fountain Valley, CA 92728-8020
(714) 963-9811
Optical components

TecMar, Inc. also know as Scientific Solutions, Inc.
6225 Cochran Road
Solon, Ohio 44139
(216) 349 4030
Stepper Motor Controller Board

Vermont Creative Software
21 Elm Avenue
Richford, VT 05476
(802) 848 3502
Windows for Data for Microsoft C version 5.0

Appendix B

Global Variables

Variable	Description
double energyi_pd[2000]	incident energy array used to store sequence of incident energys.
double energyt_pd[2000]	transmitted energy array
double energyr_pd[2000]	reflected energy array
long pulsecount_pl[2000]	number of pulses array
expdesc1_pc[82]	Experiment Description line #1
expdesc2_pc[82]	Experiment Description line #2
expdesc3_pc[82]	Session Experiment Description line #1
expdesc4_pc[82]	Session Experiment Description line #2
ecsdata_path_pc[82]	path to store data with.
ecsparam_path_pc[82]	path to load and store parameter files with
config_u	Parameter configuration structure/union. See the expconf.c section for a full description.

Appendix C
Software Listings

```
/*****************************************************************************/
/*  expconf.c                                                                */
/*  GLOBAL DEFINES for the expose programs                                   */
/*****************************************************************************/

#define DT2801_A    0x5              /* DATA I/O Board ID codes */
#define DAC_O   0
#define MAX_DET 8


/*****************************************************************************/
/* CONFIGURATION STRUCTURE DEFINITION                                        */
/*****************************************************************************/

struct config_type_structure {
  long fire_delay_counth_l;
  long fire_delay_countl_l;
  int average_count_i;
  double high_voltage_d;           /* highest possible voltage */
  double low_voltage_d;            /* lowest possible voltage */
  int gain_i;                      /* gain of A/D converters */
  double noc_d;                    /* Resolution for 12 bit A/D */
  int board_type_i;
  int microcode_i;
  double det_calib_pd[MAX_DET];
  double det_apera_pd[MAX_DET];
  double det_scale_pd[MAX_DET];
  long toggledelay_l;
  int detchannel_i[MAX_DET];
  double startenergy_d;
  double endenergy_d;
  int sequence_i;
  int steps_i;
  double targetenergy_pd[21];
  int stepperrate_i;
  int stepperfactor_i;
  int stepperslope_i;
  int manualstepsize_i;
  int autostepsize_i;
  int readydelay_i;
  long stepperdelay_l;
  int centeroffset_i;
  int drmstart_i;
  int drmstepsize_i;
  double elow_d;
  double ehigh_d;
  double maxdose_d;
  double measdens_d;
  };

union config_type_union {
  struct config_type_structure config_st;
  char buff[512];
  };
```

```
/*****************************************************************************/
/*  expfunc.c                                                                */
/*****************************************************************************/


/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <math.h>
#include <stdio.h>
#include <string.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>
#include "expconf.c"


/*****************************************************************************/
/*  GLOBAL VARIABLES                                                         */
/*****************************************************************************/

extern double energyi_pd[2000];
extern double energyt_pd[2000];
extern double energyr_pd[2000];
extern long pulsecount_pl[2000];

extern union config_type_union config_u;

static char filepost_bleach_PC[]=".ble";
static char filepost_drm_PC[]=".drm";
static char filepost_pulse_PC[]= ".pul";
static char drmexposure_PCT[]=" DRM Exposure ";
static char bleachingmeasurement_PCT[]=" Bleaching Measurement ";
static char continuousfiremode_PCT[]=" Continuous Fire Mode ";
static char averagefiremode_PCT[]=" Average Fire Mode ";
static char laserenergy_PCT[]=" Laser Energy ";
static char singleshotmode_PCT[]=" Single Shot Mode ";

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


abcexpose() {
  char retu_c;
  int count_i,pointcount_i=0;
  FILE *outfile;
  char filename_pc[80],user1_pc[80],user2_pc[80],temp_pc[80];
  double totalenergyi_d=0,totalenergyt_d=0,totalenergyr_d=0;
  double energyi_d,energyt_d,energyr_d;
  int datapoint_i;
  WINDOW disp_pw;
  defs_wn(&disp_pw,2,2,18,76,BDR_DLNP);
  sw_name(bleachingmeasurement_PCT,&disp_pw);
```

```
    sw_popup(ON,&disp_pw);
    set_wn(&disp_pw);
    if (laserready(bleachingmeasurement_PCT)== TRUE
      && testfire(bleachingmeasurement_PCT)==TRUE &&
      genmessage(" Load Wafer ","Load wafer for bleaching measurement",
      "Bleaching will begin at press of a key.","Press 'A' to abort")!='A')
      {
      openshutter();
      v_printf(&disp_pw,"Incident Energy     %%Transmitted    %%Reflected \n");
      while (pointcount_i<2000 && totalenergyi_d < config_u.config_st.measdens_d)
{
        pulsetoenergy(config_u.config_st.ehigh_d,&pulsecount_pl[pointcount_i],
                      &energyi_pd[pointcount_i],
                      &energyt_pd[pointcount_i],&energyr_pd[pointcount_i]);
        totalenergyi_d=totalenergyi_d+energyi_pd[pointcount_i];
        v_printf(&disp_pw,"%+11.4fmJ/cm2    %6.2f           %6.2f \n",
                 totalenergyi_d,100*energyt_pd[pointcount_i]/energyi_pd[pointcount
_i],
                 100*energyr_pd[pointcount_i]/energyi_pd[pointcount_i]);
        pointcount_i=pointcount_i+1;
        }
      while (pointcount_i<2000 && totalenergyi_d < config_u.config_st.maxdose_d) {
        pulsetoenergy(config_u.config_st.elow_d,&pulsecount_pl[pointcount_i],
                      &energyi_pd[pointcount_i],
                      &energyt_pd[pointcount_i],&energyr_pd[pointcount_i]);
        totalenergyi_d=totalenergyi_d+energyi_pd[pointcount_i];
        v_printf(&disp_pw,"%+11.4fmJ/cm2    %6.2f           %6.2f \n",
                 totalenergyi_d,100*energyt_pd[pointcount_i]/energyi_pd[pointcount
_i],
                 100*energyr_pd[pointcount_i]/energyi_pd[pointcount_i]);
        pointcount_i=pointcount_i+1;
        }
      closeshutter();
      graphbleachdata(pointcount_i,totalenergyi_d);
      while (savedata(&outfile,bleachingmeasurement_PCT,
        "Bleaching measurement data for photo resist",
        "Total incident energy, percent transmission, percent reflection, pulsecou
nt",
        filepost_bleach_PC) == 1)
        {
        totalenergyi_d=0;
        for (count_i=0; count_i<pointcount_i; count_i++) {
          totalenergyi_d=totalenergyi_d+energyi_pd[count_i];
          fprintf(outfile,"%+11.4f  %+11.4f  %+11.4f  %5i\n",
                totalenergyi_d,100*energyt_pd[count_i]/energyi_pd[count_i],
               100*energyr_pd[count_i]/energyi_pd[count_i],pulsecount_pl[count_i])
;
          }
        fclose(outfile);
        }
      }
    unset_wn(&disp_pw);
    }

testfire(title_pc)
  char *title_pc;
  {
  int key_i=0;
  int retu_i=0;
  WINDOW disp_pw;
  double detectors_pd[MAX_DET];
  int count_i;
  bool looping_b=TRUE;
  closeshutter();
  defs_wn(&disp_pw,16,2,6,76,BDR_DLNP);
  sw_popup(ON,&disp_pw);
```

```
  sw_name(laserenergy_PCT,&disp_pw);
  set_wn(&disp_pw);
  while (looping_b==TRUE) {
    if (genmessage(title_pc,"Ready to test laser pulse energy",
      "Laser will fire when you press a key",
      "Press 'A' to abort")=='A')
      {
      looping_b=FALSE;
      retu_i=0;
      }
    else {
      key_i=0;
      while (key_i==0) {
        firelaser();
        readdet(detectors_pd);
        printdetdata(detectors_pd,&disp_pw);
        key_i=ki_chk();
        }
      key_i=ki();
      retu_i=querybool(title_pc,"Energy of laser.", "",
          "Are these energies acceptable (Y/N):",1);
      if (retu_i==1) {
        looping_b=FALSE;
        }
      }
    }
  unset_wn(&disp_pw);
  return(retu_i);
  }


fireone(pass_pc)
  char *pass_pc;
  {
  double detectors_pd[MAX_DET];
  int count_i;
  WINDOW disp_pw;
  char ch_c=' ';
  if (laserready(singleshotmode_PCT)==TRUE) {
    defs_wn(&disp_pw,10,2,10,76,BDR_DLNP);
    sw_popup(ON,&disp_pw);
    sw_name(singleshotmode_PCT,&disp_pw);
    set_wn(&disp_pw);
    while (ch_c==' ') {
      firelaser();
      readdet(detectors_pd);
      printdetdata(detectors_pd,&disp_pw);
      ch_c=ki();
      }
    unset_wn(&disp_pw);
    }
  return(0);
  }


fireaverage1() {
 if (laserready(averagefiremode_PCT)==TRUE) {
   fireaverage(config_u.config_st.average_count_i);
   }
 }


fireaverage(pulsecount_i)
  int pulsecount_i;
  {
  double detectors_pd[MAX_DET];
```

```
    FILE *outfile;
    double total_pd[MAX_DET];
    int count_i,count2_i;
    WINDOW disp_pw;
    char ch_c=' ';
    for (count2_i=0; count2_i<3; count2_i++) {
      total_pd[count2_i]=0;
      }
    defs_wn(&disp_pw,2,2,18,76,BDR_DLNP);
    sw_popup(ON,&disp_pw);
    sw_name(averagefiremode_PCT,&disp_pw);
    set_wn(&disp_pw);
    for (count_i=0; count_i<pulsecount_i; count_i++) {
      firelaser();
      readdet(detectors_pd);
      for (count2_i=0; count2_i<3; count2_i++) {
        total_pd[count2_i]=total_pd[count2_i]+detectors_pd[count2_i];
        }
      energyi_pd[count_i]=detectors_pd[0];
      energyt_pd[count_i]=detectors_pd[1];
      energyr_pd[count_i]=detectors_pd[2];
      v_printf(&disp_pw,"%4d: %6.2f %6.2f ",count_i+1,
            100*energyt_pd[count_i]/energyi_pd[count_i],
            100*energyr_pd[count_i]/energyi_pd[count_i]);
      printdetdata(detectors_pd,&disp_pw);
      }
    for (count2_i=0; count2_i<3; count2_i++) {
        total_pd[count2_i]=total_pd[count2_i]/count_i;
        }
    if (total_pd[0]!=0) v_printf(&disp_pw,"Avg:  %6.2f %6.2f ",
            100*total_pd[1]/total_pd[0],
            100*total_pd[2]/total_pd[0]);
    else  v_printf(&disp_pw,"Avg:                    ");
    printdetdata(total_pd,&disp_pw);
    v_printf(&disp_pw,"Press a key:");
    ch_c=ki();
    while (savedata(&outfile,averagefiremode_PCT,
      "Detector data",
      "Pulse no, incident energy, transmitted, reflected, %%transmitted, %%reflect
ed",
      filepost_pulse_PC) == 1)
      {
      for (count_i=0; count_i<pulsecount_i; count_i++) {
        fprintf(outfile,"%5i  %+11.4f  %+11.4f  %+11.4f  %6.2f  %6.2f\n",
              count_i+1,energyi_pd[count_i],energyt_pd[count_i],
              energyr_pd[count_i],
              100*energyt_pd[count_i]/energyi_pd[count_i],
              100*energyr_pd[count_i]/energyi_pd[count_i]);
        }
      fclose(outfile);
      }
    unset_wn(&disp_pw);
    return(0);
    }


firecont(pass_pc)
  char *pass_pc;
  {
  double detectors_pd[MAX_DET];
  int count_i;
  WINDOW disp_pw;
  int key_i=0;
  if (laserready(continuousfiremode_PCT)==TRUE) {
    defs_wn(&disp_pw,2,2,18,76,BDR_DLNP);
    sw_popup(ON,&disp_pw);
```

```
    sw_name(continuousfiremode_PCT,&disp_pw);
    set_wn(&disp_pw);
    while (key_i==0) {
      firelaser();
      readdet(detectors_pd);
      printdetdata(detectors_pd,&disp_pw);
      key_i=ki_chk();
      }
    key_i=ki();
    key_i=ki();
    unset_wn(&disp_pw);
    }
  return(0);
  }


drmexpose(pass_pc)
  char *pass_pc;
  {
  int count_i,pointcount_i;
  FILE *outfile;
  char filename_pc[80],user1_pc[80],user2_pc[80],temp_pc[80];
  WINDOW disp_pw;
  defs_wn(&disp_pw,2,2,18,76,BDR_DLNP);
  sw_name(drmexposure_PCT,&disp_pw);
  sw_popup(ON,&disp_pw);
  set_wn(&disp_pw);
  if (editdrmparams()!=-1 && laserready(drmexposure_PCT)== TRUE
    && testfire(drmexposure_PCT)==TRUE &&
    genmessage(drmexposure_PCT,"Load wafer for DRM exposure pattern",
    "Exposure will begin at press of a key.","Press 'A' to abort")!='A')
    {
    v_printf(&disp_pw,"Aligning stage please wait....\n");
    findhome(1);
    stepleft(1,config_u.config_st.drmstart_i);
    for (pointcount_i=0; pointcount_i<config_u.config_st.steps_i; pointcount_i++
) {
      v_printf(&disp_pw,"Target energy=%fmJ/cm2  ",config_u.config_st.targetener
gy_pd[pointcount_i]);
      pulsetoenergy(config_u.config_st.targetenergy_pd[pointcount_i],
                  &pulsecount_pl[pointcount_i],
                  &energyi_pd[pointcount_i],
                  &energyt_pd[pointcount_i],&energyr_pd[pointcount_i]);
      v_printf(&disp_pw,"  Actual=%fmJ/cm2",energyi_pd[pointcount_i]);
      v_printf(&disp_pw,"  Shots=%i\n",pulsecount_pl[pointcount_i]);
      stepright(1,config_u.config_st.drmstepsize_i);
      }
    while (savedata(&outfile,drmexposure_PCT,
      "DRM exposure data",
      "Rectangle No, Incident energy, Transmitted energy, Reflected energy, Puls
e count",
      filepost_drm_PC) == 1)
      {
      for (count_i=0; count_i<pointcount_i; count_i++) {
        fprintf(outfile,"%4i  %+11.4f  %+11.4f  %+11.4f  %5i\n",
              count_i+1,
              energyi_pd[count_i],energyt_pd[count_i],
              energyr_pd[count_i],pulsecount_pl[count_i]);
        }
      fclose(outfile);
      }
    }
  unset_wn(&disp_pw);
  }
```

```
calibrate() {
  }


/*************************************************************************/
/*  expmenu.c                                                            */
/*************************************************************************/


/*************************************************************************/
/*  SYSTEM INCLUDE FILES                                                 */
/*************************************************************************/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>


/*************************************************************************/
/*  OTHER INCLUDE FILES                                                  */
/*************************************************************************/
#include <general.h>
#define WN_DEBUG                /* enable window debugging system */
#define F_FLOAT                 /* enable floating point in WFD */
#include <wfd.h>
#include "expconf.c"



/*************************************************************************/
/*  GLOBAL FORMS FOR WINDOWS FOR DATA                                    */
/*************************************************************************/
DFORMPTR main_pfm;
DFORMPTR fkeys_pfm;                    /* form for function key window */
DFORMPTR fback_pfm;


/*************************************************************************/
/*  FUNCTION DEFINITIONS                                                 */
/*************************************************************************/
int firecont(), fireone(), drmexpose(),exitmenu(),fireaverage1();
int toggleshutter(),stepleft1(),stepright1(),openshutter(),closeshutter();
int manualmenu(),edithardparams(),editdetparams();
int abcexpose(), editabcparams(),editdrmparams(),editmiscparams();
int loadparams(), saveparams(),calibrate();
int gain0(),gain1(),firesignal0(),firesignal1();
int findhome1();


/*************************************************************************/
/*  GLOBAL STRING DEFINITIONS                                            */
/*************************************************************************/
static char commandline_PC[]="F3=Undo   F5=Clear   F10=Done   "
      "PGUP=Moveleft   PGDN=Moveright   ESC=Abort";
static char picture_float_PC[]="######@#####E####";
static char picture_int_PC[]="999999";
static char picture_string60_PC[]="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX";

static char numbers_PC[21][4] ={"0: ","1: ","2: ","3: ","4: ","5: ","6: ",
                                "7: ","8: ","9: ","10:","11:","12:","13:",
                                "14:","15:","16:","17:","18:","19:","20:"};

extern union config_type_union config_u;
extern char expdesc[ pc[82];
```

```
extern char expdesc2_pc[82];
extern char expdesc3_pc[82];
extern char expdesc4_pc[82];

extern char ecsparam_path_pc[82];
extern char ecsdata_path_pc[82];



/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/

initwindows()  {
  int retu_i=0;
  init_wfd();
  mod_wn(16,0,7,80,&help_wn);
  sw_name(" Help Window ",&help_wn);
  mod_wn(23,0,1,80,&msg_wn);
  mod_wn(23,0,1,80,&mnu_msgw);
  mod_wn(23,0,1,80,&err_wn);
  se_fldopt(F_INT,RTADJUST+CLEAR);
  se_fldopt(F_FLOAT,RTADJUST+CLEAR);
  se_fldopt(F_LONG,RTADJUST+CLEAR);
  main_pfm = fm_def(0,0,23,80,LNORMAL,  BDR_DLNP);
  sw_name(" Exposure Characterization System ",main_pfm->wnp);
  sw_namelocation(TOPCENTER,main_pfm->wnp);
  fm_up(main_pfm);
  fkeys_pfm = fm_def(24,0,1,80,LREVERSE,BDR_OP);
  sfm_opt(FORMCLEAR,OFF,fkeys_pfm);
  ftxt_def(0,0,commandline_PC,LREVERSE,fkeys_pfm);
  se_mnmsg(MANUAL);
  pl_mnmsg(23,0);
  fback_pfm = fm_def(23,0,1,80,LNORMAL,BDR_OP);
  fm_up(fback_pfm);
  return(retu_i);
  }


endwindows() {
  fm_dn(main_pfm);
  fm_free(main_pfm);
  fm_free(fkeys_pfm);
  fm_dn(fback_pfm);
  fm_free(fback_pfm);
  return(0);
  }



menu() {
  DFORMPTR menu_pfm;
  menu_pfm=mnfm_def(5,15,15,50,LNORMAL,BDR_OP);
  sfm_opt(MNTOPESCAPE,  OFF,  menu_pfm);
  mnf_def(1,1,"A) Edit DRM exposure parameters",
    "Edit the parameters associated with DRM exposure",
    NULLP,editdrmparams,menu_pfm);
  mnf_def(2,1,"B) Generate DRM exposure matrix",
    "Fire the laser and move the wafer while recording exposure dose",
    NULLP,drmexpose,menu_pfm);
  mnf_def(3,1,"C) Edit bleaching parameters",
    "Edit the parameters associated with bleaching measurement",
    NULLP,editabcparams,menu_pfm);
  mnf_def(4,1,"D) Perform bleaching measurement",
    "Measure the bleaching of photo-resist",
```

```
    NULLP,abcexpose,menu_pfm);
  mnf_def(5,1,"E) Edit detector parameters",
    "Edit the parameters associated with the detectors",
    NULLP,editdetparams,menu_pfm);
  mnf_def(6,1,"F) Edit hardware parameters",
    "Edit the parameters associated with the hardware",
    NULLP,edithardparams,menu_pfm);
  mnf_def(7,1,"G) Edit misc parameters",
    "Edit parameters",
    NULLP,editmiscparams,menu_pfm);
  mnf_def(8,1,"H) Load parameter set",
    "Load a configuration file",
    NULLP,loadparams,menu_pfm);
  mnf_def(9,1,"I) Save parameters set",
    "Save a configuration file",
    NULLP,saveparams,menu_pfm);
  mnf_def(10,1,"J) Manual Operations",
    "Manually operate the system",
    NULLP,manualmenu,menu_pfm);
  mnf_def(12,1,"X) Exit",
    "Exit the program",
    NULLP,exitmenu,menu_pfm);
  mn_proc(0,menu_pfm);
  fm_free(menu_pfm);
  return(0);
  }


manualmenu() {
  DFORMPTR menu_pfm;
  menu_pfm=mnfm_def(5,15,17,50,LNORMAL,BDR_OP);
  sfm_opt(MNTOPESCAPE, OFF, menu_pfm);
  mnf_def(1,1,"S) Single shot mode",
    "Fire a single shot with each press of spacebar and record energy",
    NULLP,fireone,menu_pfm);
  mnf_def(2,1,"C) Continuous fire mode",
    "Fire the laser and display the energy of each detector",
    NULLP,firecont,menu_pfm);
  mnf_def(3,1,"M) Multiple/average shot mode",
    "Fire the laser a specified number of time and average the energy",
    NULLP,fireaverage1,menu_pfm);
  mnf_def(4,1,"O) Open shutter",
    "Open shutter",
    NULLP,openshutter,menu_pfm);
  mnf_def(5,1,"C) Close shutter",
    "Close shutter",
    NULLP,closeshutter,menu_pfm);
  mnf_def(6,1,"L) Move left",
    "Move wafer stage left",
    NULLP,stepleft1,menu_pfm);
  mnf_def(7,1,"R) Move right",
    "Move wafer stage right",
    NULLP,stepright1,menu_pfm);
  mnf_def(8,1,"A) Align stage and move to home position",
    "Move stage to its home position and calibrate",
    NULLP,findhome1,menu_pfm);
  mnf_def(9,1,"C) Calibrate diodes to each other",
    "Calibrate two diodes",
    NULLP,calibrate,menu_pfm);
  mnf_def(10,1,"0) Set fire signal = 0",
    "",
    NULLP,firesignal0,menu_pfm);
  mnf_def(11,1,"1) Set fire signal = 1",
    "",
    NULLP,firesignal1,menu_pfm);
  mnf_def(12,1,"2) Set capacitor gain = 0",
```

```
    "",
    NULLP,gain0,menu_pfm);
  mnf_def(13,1,"3) Set capacitor gain = 1",
    "",
    NULLP,gain1,menu_pfm);
  mnf_def(14,1,"X) Exit to main menu",
    "Exit the menu",
    NULLP,exitmenu,menu_pfm);
  mn_proc(0,menu_pfm);
  fm_free(menu_pfm);
  return(0);
  }


editdetparams(pass_pc)
  char *pass_pc;
  {
  int count_i;
  static char calibration_PC[]="Calibration (mJ/V)";
  static char aperature_PC[]="Aperature (cm2)";
  static char scaling_PC[]="Scaling Factor";
  int retu_i=1;
  DFORMPTR form_pfm;
  DFIELDPTR temp_pfd;
  unset_wn(&mnu_msgw);
  form_pfm = fm_def(2,2,20,76,LNORMAL, BDR_DLNP);
  sfm_opt(CURSORFREE,ON,form_pfm);
  sfm_opt(VERIFYEXIT,OFF,form_pfm);
  sw_name(" Detector Data ",form_pfm->wnp);
  ftxt_def(1,15,calibration_PC,LNORMAL,form_pfm);
  ftxt_def(1,37,aperature_PC,LNORMAL,form_pfm);
  ftxt_def(1,58,scaling_PC,LNORMAL,form_pfm);
  for (count_i=0; count_i<MAX_DET; count_i++) {
    ftxt_def(count_i+2,1,"Detector",LNORMAL,form_pfm);
    temp_pfd=fld_def(count_i+2,15,numbers_PC[count_i+1],
      FADJACENT,picture_float_PC, F_FLOAT,
      (char *) &config_u.config_st.det_calib_pd[count_i], form_pfm);
    fld_tdef(temp_pfd,count_i+2,35,"",FADJACENT,picture_float_PC, F_FLOAT,
      (char *) &config_u.config_st.det_apera_pd[count_i], form_pfm);
    fld_tdef(temp_pfd,count_i+2,55,"",FADJACENT,picture_float_PC, F_FLOAT,
      (char *) &config_u.config_st.det_scale_pd[count_i], form_pfm);
    }
  fld_tdef(temp_pfd,11,1,"A/D High Voltage:          ",FADJACENT,
    picture_float_PC, F_FLOAT,
    (char *) &config_u.config_st.high_voltage_d, form_pfm);
  fld_tdef(temp_pfd,12,1,"A/D Low Voltage:           ",FADJACENT,
    picture_float_PC, F_FLOAT,
    (char *) &config_u.config_st.low_voltage_d, form_pfm);
  fld_tdef(temp_pfd,13,1,"A/D Resolution:            ",FADJACENT,
    picture_float_PC, F_FLOAT,
    (char *) &config_u.config_st.noc_d, form_pfm);
  fld_tdef(temp_pfd,14,1,"Active Detector Channel 1: ",FADJACENT,
    picture_int_PC, F_INT,
    (char *) &config_u.config_st.detchannel_i[0], form_pfm);
  fld_tdef(temp_pfd,15,1,"Active Detector Channel 2: ",FADJACENT,
    picture_int_PC, F_INT,
    (char *) &config_u.config_st.detchannel_i[1], form_pfm);
  fld_tdef(temp_pfd,16,1,"Active Detector Channel 3: ",FADJACENT,
    picture_int_PC, F_INT,
    (char *) &config_u.config_st.detchannel_i[2], form_pfm);
  fm_up(fkeys_pfm);
  fm_proc(0,form_pfm);
  fm_free(form_pfm);
  fm_dn(fkeys_pfm);

  return(retu_i);
```

```
    }


  edithardparams(pass_pc)
    char *pass_pc;
    {
    int retu_i=0;
    DFORMPTR form_pfm;
    DFIELDPTR temp_pfd;
    unset_wn(&mnu_msgw);
    form_pfm = fm_def(2,2,20,76,LNORMAL, BDR_DLNP);
    sfm_opt(VERIFYEXIT,OFF,form_pfm);
    sw_name(" Hardware Data ",form_pfm->wnp);
    temp_pfd = fld_def(1,1,"Fire Delay Count High:                ",FADJACENT,
      picture_int_PC, F_LONG,
      (char *) &config_u.config_st.fire_delay_counth_l, form_pfm);
    fld_tdef(temp_pfd,2,1,"Fire Delay Count Low:                 ",FADJACENT,
      picture_int_PC, F_LONG,
      (char *) &config_u.config_st.fire_delay_countl_l, form_pfm);
    fld_tdef(temp_pfd,5,1,"A/D Board Type:                       ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.board_type_i, form_pfm);
    fld_tdef(temp_pfd,6,1,"A/D Microcode Revision:               ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.microcode_i, form_pfm);
    fld_tdef(temp_pfd,8,1,"Stepper motor rate:                   ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.stepperrate_i, form_pfm);
    fld_tdef(temp_pfd,9,1,"Stepper motor factor:                 ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.stepperfactor_i, form_pfm);
    fld_tdef(temp_pfd,10,1,"Stepper motor slope:                 ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.stepperslope_i, form_pfm);
    fld_tdef(temp_pfd,11,1,"Manual step size:                    ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.manualstepsize_i, form_pfm);
    fld_tdef(temp_pfd,12,1,"Auto step size:                      ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.autostepsize_i, form_pfm);
    fld_tdef(temp_pfd,13,1,"Stepper ready delay:                 ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.readydelay_i, form_pfm);
    fld_tdef(temp_pfd,14,1,"Stepper operate delay:               ",FADJACENT,
      picture_int_PC, F_LONG,
      (char *) &config_u.config_st.stepperdelay_l, form_pfm);
    fld_tdef(temp_pfd,15,1,"Center offset (mils):                ",FADJACENT,
      picture_int_PC, F_INT,
      (char *) &config_u.config_st.centeroffset_i, form_pfm);
    fm_up(fkeys_pfm);
    fm_proc(0,form_pfm);
    fm_free(form_pfm);
    fm_dn(fkeys_pfm);
    return(retu_i);
    }


  editdrmparams()
    {
    bool reply_b=0;
    int retu_i=0;
    int count_i;
    DFORMPTR form_pfm;
    DFIELDPTR temp_pfd;
    unset_wn(&mnu_msgw);
    form_pfm = fm_def(2,2,20,76,LNORMAL, BDR_DLNP);
```

```
  sfm_opt(FORMCLEAR,OFF,form_pfm);
  sfm_opt(VERIFYEXIT,OFF,form_pfm);
  sfm_opt(CURSORFREE,ON,form_pfm);
  sw_name(" DRM Exposure Data ",form_pfm->wnp);
  temp_pfd = fld_def(1,1,"Start Energy (mJ/cm2)                    ", FADJACENT,
    picture_float_PC, F_FLOAT,
    (char *) &config_u.config_st.startenergy_d, form_pfm);
  fld_tdef(temp_pfd,2,1,"End energy (mJ/cm2):                      ",FADJACENT,
    picture_float_PC,F_FLOAT,
    (char *) &config_u.config_st.endenergy_d,form_pfm);
  fld_tdef(temp_pfd,3,1,"Number of exposure sites:                 ",FADJACENT,
    picture_int_PC,F_INT,
    (char *) &config_u.config_st.steps_i,form_pfm);
  fld_tdef(temp_pfd,4,1,"Mode (0=Arith 1=Exponential 2=Manual):    ",FADJACENT,
     picture_int_PC,F_INT,
    (char *) &config_u.config_st.sequence_i,form_pfm);
  fld_tdef(temp_pfd,5,1,"Starting distance from center (mils):     ",FADJACENT,
    picture_int_PC, F_INT,
    (char *) &config_u.config_st.drmstart_i, form_pfm);
  fld_tdef(temp_pfd,6,1,"Step size (mils):                         ",FADJACENT,
    picture_int_PC, F_INT,
    (char *) &config_u.config_st.drmstepsize_i, form_pfm);
  ftxt_def(7,1,"Exposure Energy Target Values:",LNORMAL,form_pfm);
  for (count_i=0; count_i<10; count_i++) {
    temp_pfd=fld_def(count_i+8,15,numbers_PC[count_i+1],
      FADJACENT,picture_float_PC, F_FLOAT,
      (char *) &config_u.config_st.targetenergy_pd[count_i], form_pfm);
    temp_pfd=fld_def(count_i+8,45,numbers_PC[count_i+11],
      FADJACENT,picture_float_PC, F_FLOAT,
      (char *) &config_u.config_st.targetenergy_pd[count_i+10], form_pfm);
      }
  fm_up(form_pfm);
  while (reply_b==0 && retu_i==0) {
    fm_up(fkeys_pfm);
    if (fm_rd(0,form_pfm)== EXIT_FORM) {
      calcenergy();
      fm_upd(form_pfm);
      fm_dn(fkeys_pfm);
      if (querybool2("Are these parameters acceptable? (Y/N):",1)==1) {
        retu_i=1;
        }
      }
    else {
      fm_dn(fkeys_pfm);
      retu_i=-1;
      }
    }
  fm_dn(form_pfm);
  fm_free(form_pfm);
  return(retu_i);
  }


editabcparams(pass_pc)
  char *pass_pc;
  {
  int retu_i=0;
  DFORMPTR form_pfm;
  DFIELDPTR temp_pfd;
  unset_wn(&mnu_msgw);
  form_pfm = fm_def(2,2,20,76,LNORMAL, BDR_DLNP);
  sfm_opt(VERIFYEXIT,OFF,form_pfm);
  sw_name(" Bleaching Data ",form_pfm->wnp);
  temp_pfd = fld_def(1,1,"Maximum dose:                          ", FADJACENT,
    picture_float_PC, F_FLOAT,
    (char *) &config_u.config_st.maxdose_d, form_pfm);
```

```
   fld_tdef(temp_pfd,2,1,"Measurement density drop off point:",FADJACENT,
     picture_float_PC, F_FLOAT,
     (char *) &config_u.config_st.measdens_d, form_pfm);
   fld_tdef(temp_pfd,3,1,"Energy per measurement (high dens):",FADJACENT,
     picture_float_PC, F_FLOAT,
     (char *) &config_u.config_st.ehigh_d, form_pfm);
   fld_tdef(temp_pfd,4,1,"Energy per measurement (low dens): ",FADJACENT,
     picture_float_PC, F_FLOAT,
     (char *) &config_u.config_st.elow_d, form_pfm);
   fm_up(fkeys_pfm);
   fm_proc(0,form_pfm);
   fm_free(form_pfm);
   fm_dn(fkeys_pfm);
   return(retu_i);
   }


editmiscparams(pass_pc)
   char *pass_pc;
   {
   int retu_i=0;
   DFORMPTR form_pfm;
   DFIELDPTR temp_pfd;
   unset_wn(&mnu_msgw);
   form_pfm = fm_def(2,2,20,76,LNORMAL, BDR_DLNP);
   sfm_opt(VERIFYEXIT,OFF,form_pfm);
   sw_name(" Miscellaneous Data ",form_pfm->wnp);
   temp_pfd = fld_def(1,1,"Average Mode Count:                ",FADJACENT,
     picture_int_PC, F_INT,
     (char *) &config_u.config_st.average_count_i, form_pfm);
   ftxt_def(3,1," Session Data: ",LNORMAL,form_pfm);
   temp_pfd=fld_def(4,1,"Line 1: ",FADJACENT,picture_string60_PC,F_STRING,
          (char *) expdesc3_pc, form_pfm);
   sf_opt(INITIALBLANKS+RTADJUST,OFF,temp_pfd);
   fld_tdef(temp_pfd,5,1,"Line 2: ",FADJACENT,picture_string60_PC,F_STRING,
          (char *) expdesc4_pc, form_pfm);
   ftxt_def(7,1," Experiment Data: ",LNORMAL,form_pfm);
   fld_tdef(temp_pfd,8,1,"Line 1: ",FADJACENT,picture_string60_PC,F_STRING,
          (char *) expdesc1_pc, form_pfm);
   fld_tdef(temp_pfd,9,1,"Line 2: ",FADJACENT,picture_string60_PC,F_STRING,
          (char *) expdesc2_pc, form_pfm);
   fld_tdef(temp_pfd,11,1,"Data path: " ,FADJACENT,picture_string60_PC,F_STRING,
          (char *) ecsdata_path_pc, form_pfm);
   fld_tdef(temp_pfd,12,1,"Param path:",FADJACENT,picture_string60_PC,F_STRING,
          (char *) ecsparam_path_pc, form_pfm);
   fm_up(fkeys_pfm);
   fm_proc(0,form_pfm);
   fm_free(form_pfm);
   fm_dn(fkeys_pfm);
   if (config_u.config_st.average_count_i >2000) config_u.config_st.average_count
_i=2000;
   directory(ecsparam_path_pc);
   directory(ecsdata_path_pc);
   return(retu_i);
   }


exitmenu(pass_pc)
   char *pass_pc;
   {
   return(-2);
   }


calcenergy() {
   double addon_d.base_d;
```

```
  int count_i;
  if (config_u.config_st.steps_i < 1) return (-1);
  if (config_u.config_st.sequence_i==2) return(0);
  for (count_i=0; count_i<20; count_i++) {
    config_u.config_st.targetenergy_pd[count_i]=0;
    }
  if (config_u.config_st.sequence_i==0) {
    config_u.config_st.targetenergy_pd[0]=config_u.config_st.startenergy_d;
    config_u.config_st.targetenergy_pd[config_u.config_st.steps_i-1]=config_u.co
nfig_st.endenergy_d;
    if (config_u.config_st.steps_i<3) return(0);
    addon_d = (config_u.config_st.endenergy_d - config_u.config_st.startenergy_d
)/(config_u.config_st.steps_i-1);
    for (count_i=0; count_i<config_u.config_st.steps_i-1; count_i++) {
      config_u.config_st.targetenergy_pd[count_i]=config_u.config_st.startenergy
_d+ (count_i)*addon_d;
      }
    }
  if (config_u.config_st.sequence_i==1) {
    config_u.config_st.targetenergy_pd[0]=config_u.config_st.startenergy_d;
    config_u.config_st.targetenergy_pd[config_u.config_st.steps_i-1]=config_u.co
nfig_st.endenergy_d;
    if (config_u.config_st.steps_i<3) return(0);
    addon_d= config_u.config_st.startenergy_d-1.0;
    base_d=pow(config_u.config_st.endenergy_d - addon_d, (double)
         (1.0/((double)config_u.config_st.steps_i-1.0)));
    for (count_i=0; count_i<config_u.config_st.steps_i-1; count_i++) {
      config_u.config_st.targetenergy_pd[count_i]=addon_d+pow(base_d,count_i);
      }
    }
  return(0);
  }


stepleft1() {
  stepleft(1,config_u.config_st.manualstepsize_i);
  }


stepright1() {
  stepright(1,config_u.config_st.manualstepsize_i);
  }


findhome1() {
  findhome(1);
  }
```

```
/*****************************************************************************/
/*    expose.c                                                               */
/*    EXPOSURE CHARACTERIZATION SYSTEM CONTROL PROGRAM                       */
/*    VERSION 1.0                                                            */
/*    DEAN M. DRAKO                                                          */
/*    9-17-88                                                                */
/*    U.C. Berkeley                                                          */
/*****************************************************************************/

/*****************************************************************************/
/*    HARDWARE DESCRIPTION                                                   */
/*
   This program is designed to operate with a specific hardware
   configuration.  The hardware is briefly summarized here.

   A DATA I/O 2801_A Digital to Analog and I/O card is needed.
   This card is used to control firing of the laser (through the
   laser trigger input), to open and close the shutter,
   and also to read the power measurements from the
   diode detectors.  A device driver must be installed in the system for
   these routines to properly function.  The device driver is named
   "pcldrv.sys".  It must be in the root directory.  The following line
   must be in the config.sys file:  "device=pcldrv.sys".  This will
   install the device driver when the system is powered on.  The
   connections to the DATA I/O card are as follows:
         A/D port 0:  diode detector #1
         A/D port 1:  diode detector #2
         A/D port 2:  diode detector #3
         Digital I/O port 0: (configured as output)
              Bit 0:  Laser Fire trigger & diode detector trigger
              Bit 1:  Shutter operation
              Bit 2-4: diode module gain
              Bit 5-7: not used
         Digital I/O port 1: (configured as input)
              Bit 0-7: not used

   TECMAR stepper motor controller with two Cyber MicroSystems CY512
   stepper motor controllers is used in the system to control the stepper
   motors.  The board is configured for I/O address hex 2C0.  No additional
   software is needed for this board.  The board has three cables which
   go to an interface board which actually powers the motors.  The TECMAR
   board also has several digital inputs which are connected to the
   limit switches of the linear stages.
        Stepper motor 1: linear stage 1 control
        Stepper motor 2: linear stage 2 control


      SOFTWARE DESCRIPTION

   The software was written in Microsoft C 5.0.  A general window package
   (Windows for Data) from Vermont Creative SOftware was used for the
   window generation.  The following modules make up the entire software
   suite:    expconf.c    -- global variable definitions
             expfunc.c    -- actual functions performed by system
             expmenu.c    -- menus
             expose.c     -- the main program
             filer.c      -- file operations sub-routines
             graphics.c   -- graphics routines
             laser.c      -- laser related routines
             pclerrs.c    -- error info for DT2801
             query.c      -- subroutines that query the user
```

```
            stepper.c    -- stepper motor controller sub-routines
            expose.mak   -- make file used to automatically build program

            wfdl.lib     -- windows for c library
            wfcl.lib     -- another windows for c library
            pcc4llib.lib -- library for DT2801 card
/*                                                                              */
/********************************************************************************/



/********************************************************************************/
/*  SYSTEM INCLUDE FILES                                                        */
/********************************************************************************/
#include <string.h>


/********************************************************************************/
/*  OTHER INCLUDE FILES                                                         */
/********************************************************************************/
#include <general.h>
#include "expconf.c"          /* configuration structure */
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>
#include <wfd_glob.h>


/********************************************************************************/
/*  GLOBAL VARIABLES                                                            */
/********************************************************************************/
double energyi_pd[2000];
double energyt_pd[2000];
double energyr_pd[2000];
long pulsecount_pl[2000];

char expdesc1_pc[82];
char expdesc2_pc[82];
char expdesc3_pc[82];
char expdesc4_pc[82];

static char blank80_PC[]="
                          ";

char ecsdata_path_pc[82]="C:\\ecs\\ecsdata";
char ecsparam_path_pc[82]="C:\\ecs\\ecsparam";



union config_type_union config_u = {
    1660,
    1,
    20,
    10.0,
    -10.0,
    1,
    4096.0,
    0,
    0,
    {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0},
    {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0},
    {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0},
    100000,
    {0,1,2,3,4,5,6,7},
    10,
```

```
    100,
    0,
    10,
    {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200},
    0xa0,
    16,
    1,
    100,
    100,
    3000,
    30000,
    2040,
    1500,
    250,
    0.1,
    0.01,
    2000,
    100,
    };

/*****************************************************************************/
/*  MAIN                                                                     */
/*****************************************************************************/

main() {
  int retu_i=0;
  strcpy(expdesc1_pc,blank80_PC);
  strcpy(expdesc2_pc,blank80_PC);
  strcpy(expdesc3_pc,blank80_PC);
  strcpy(expdesc4_pc,blank80_PC);
  initwindows();
  initgraphics();
  retu_i=initpclab();
  if (retu_i==0) {
    endwindows();
    exit(-1);
    }
  closeshutter();
  directory(ecsdata_path_pc);
  initstepper();

  getexpdata(" Experiment Session Data ",
             "Enter the experiment session description data:",
             expdesc3_pc,expdesc4_pc);
  menu();

  endwindows();
  }
```

```
/*****************************************************************************/
/*  filer.c                                                                  */
/*  file related functions for expose.c                                      */
/*****************************************************************************/


/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>
#include <errno.h>
#include <direct.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>
#include "expconf.c"



/*****************************************************************************/
/* GLOBAL VARIABLES                                                          */
/*****************************************************************************/

char config_filename_pc[80];
extern union config_type_union config_u;
static char filepost_ecf_PC[]=".ecf";

extern char expdesc1_pc[82];
extern char expdesc2_pc[82];
extern char expdesc3_pc[82];
extern char expdesc4_pc[82];

extern char ecsdata_path_pc[82];
extern char ecsparam_path_pc[82];

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


loadparams() {
  int retu_i;
  if (directory(ecsparam_path_pc)!=0) return(0);
  getfilename(" Parameter Filename ",
              "Enter filename to load parameters from.",config_filename_pc);
  if (strlen(config_filename_pc)<3) {
    return(0);
    }
```

```
  strcat(config_filename_pc,filepost_ecf_PC);
  retu_i=readparamfile(config_filename_pc);
  if (retu_i==-2) {
    genmessage(" File Error ","Attempted to load parameter file:",
      config_filename_pc,"file not found.");
    }
  return(1);
  }


readparamfile(filename_pc)
  char *filename_pc;
  /* returns -2 if file does not exist */
  /* returns -1 if a file error occurs */
  /* returns 0 if no errors */
  {
  int infile;
  int retu_i=0;
  infile = open(filename_pc, O_RDONLY | O_BINARY);
  if (infile==-1) {
    if (errno==ENOENT) { retu_i=-2;  goto END; }
    fileerror("Opening Configuration File",filename_pc);
    retu_i=-1;
    goto END;
    }
  retu_i = read(infile,config_u.buff,sizeof(union config_type_union));
  if (retu_i==-1) {
    fileerror("Reading Configuration File",filename_pc);
    close(infile);
    retu_i=-1;
    goto END;
    }
  retu_i = close(infile);
  if (retu_i==-1) {
    fileerror("Closing Configuration File",filename_pc);
    retu_i=-1;
    goto END;
    }
  END:
  return(retu_i);
  }



saveparams() {
  if (directory(ecsparam_path_pc)!=0) return(0);
  getfilename(" Parameter Filename ",
              "Enter filename to save parameters in.",config_filename_pc);
  if (strlen(config_filename_pc)<3) {
    return(0);
    }
  strcat(config_filename_pc,filepost_ecf_PC);
  if (directory(ecsparam_path_pc) && checkfile(config_filename_pc,0)==1) {
    writeparamfile(config_filename_pc);
    }
  return(1);
  }


writeparamfile(filename_pc)
  char *filename_pc;
  {
  int retu_i=0;
  int outfile;
  outfile=open(filename_pc,O_CREAT | O_WRONLY | O_BINARY, S_IWRITE);
  if (outfile == -1) {
```

```
    fileerror("Opening configuration file for writing",filename_pc);
    retu_i=-1;
    goto END;
    }
  retu_i=write(outfile, config_u.buff, sizeof(union config_type_union));
  if (retu_i==-1) {
    fileerror("Writing configuration file",filename_pc);
    retu_i=close(outfile);
    retu_i=-1;
    goto END;
    }
  retu_i=close(outfile);
  if (retu_i==-1) {
    fileerror("Closing configuration file",filename_pc);
    retu_i=-1;
    return(-1);
    goto END;
    }
END:
  return(0);
  }


checkfile(title_pc,path_pc,concat_b)
  char *path_pc,*title_pc;
  bool concat_b;
  {
  char reply_c='C';
  FILE *inputfile;
  inputfile=fopen(path_pc,"r");
  if (inputfile==NULL) {
    return(1);
    }
  fclose(inputfile);
  if (concat_b) {
    querychar(title_pc,path_pc,"file already exists.",
        "Overwrite, concatenate, or abort (O/C/A):",&reply_c);
    if (reply_c=='O') return(1);
    if (reply_c=='C') return(2);
    }
  else {
    querychar(title_pc,path_pc,"file already exists.",
        "Overwrite or abort (O/A):",&reply_c);
    if (reply_c=='O') return(1);
    }
  return(0);
  }


savedata(outfile,experimenttit_pc,experiment1_pc,experiment2_pc,filepost_pc)
  FILE *(*outfile);
  char *experimenttit_pc,*experiment1_pc, *experiment2_pc,*filepost_pc;
  {
  char filename_pc[80];
  char temp_pc[80];
  char retu_c;
  int retu_i=0;
  bool looping_b=TRUE;
  int append_i=0;
  retu_c='N';
  while (looping_b) {
    querychar(experimenttit_pc,"Press 'F' to save data to a file,",
                        "'P' to print data, or 'N' for neither.",
                        "File, Print or Neither (F/P/N):",&retu_c);
    if (retu_c=='N') looping_b=FALSE;
    if ((retu_c=='F' && directory(ecsdata_path_pc)==0) || retu_c=='P') {
```

```
        if (retu_c=='F') getfilename(experimenttit_pc,
                "Enter a filename to save the data in.",filename_pc);
        if (retu_c=='P') {
          strcpy(filename_pc,"prn");
          }
        if (strlen(filename_pc)!=0) {
          strcat(filename_pc,filepost_pc);
          append_i=2;
          if (retu_c=='F') append_i=checkfile(experimenttit_pc,filename_pc,1);
          if (append_i!=0) {
            looping_b=FALSE;
            getexpdata(experimenttit_pc,"Enter the experiment description:",
                       expdesc1_pc,expdesc2_pc);
            if (append_i==1) *outfile=fopen(filename_pc,"w");
            if (append_i==2) *outfile=fopen(filename_pc,"a");
            fprintf(*outfile,"\n\n\n");
            fprintf(*outfile,experiment1_pc);
            fprintf(*outfile,"\n\n");
            _strdate(temp_pc);
            fprintf(*outfile,temp_pc);
            fprintf(*outfile,"\n");
            _strtime(temp_pc);
            fprintf(*outfile,temp_pc);
            fprintf(*outfile,"\n");
            fprintf(*outfile,expdesc3_pc);
            fprintf(*outfile,"\n");
            fprintf(*outfile,expdesc4_pc);
            fprintf(*outfile,"\n");
            fprintf(*outfile,expdesc1_pc);
            fprintf(*outfile,"\n");
            fprintf(*outfile,expdesc2_pc);
            fprintf(*outfile,"\n\n");
            fprintf(*outfile,experiment2_pc);
            fprintf(*outfile,"\n\n");
            retu_i=1;
            }
          }
        }
      }
    return(retu_i);
    }


directory(path_pc)
  char *path_pc;
  {
  int retu_i;
  char temp_pc[80];
  strncpy(temp_pc,path_pc,2);
  temp_pc[2]='\0';
  retu_i=chdir(path_pc);
  if (retu_i) {
    genmessage(" Directory Error ","Attempted to change to directory:",
      path_pc,"directory not found.");
    }
  return(retu_i);
  }
```

```
/*****************************************************************************/
/*  graphics.c                                                               */
/*****************************************************************************/



/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <graph.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>
#include "expconf.c"


/*****************************************************************************/
/*  GLOBAL VARIABLES                                                         */
/*****************************************************************************/
int graphics_b=FALSE;

int saved_video_mode_i;

struct videoconfig vc;

extern union config_type_union config_u;

extern double energyi_pd[2000];
extern double energyt_pd[2000];
extern double energyr_pd[2000];
extern long pulsecount_pl[2000];


/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


initgraphics() {
  graphicson();
  graphicsoff();
  }


graphicson() {
  saved_video_mode_i=v_mode;
  sav_wi(&wn0);
  vid_mode(6);
  if( setvideomode( ERESCOLOR))  graphics_b=TRUE; else
```

```
  if(_setvideomode(_HRES16COLOR)) graphics_b=TRUE; else
  if(_setvideomode(_MRES16COLOR)) graphics_b=TRUE; else
  if(_setvideomode(_MRES4COLOR)) graphics_b=TRUE; else return(0);
  _getvideoconfig(&vc);
  _setbkcolor((long)1);
  _clearscreen(_GCLEARSCREEN);
  _setlogorg(0,vc.numypixels-1);
  _setcolor(2);
  }


graphicsoff(){
  _setvideomode(_DEFAULTMODE);
  vid_mode(saved_video_mode_i);
  unsav_wi(&wn0);
  }



graphbleachdata(pointcount_i,totalenergyi_d)
  int pointcount_i;
  double totalenergyi_d;
  {
  static char ble_ver_label_PC[]="% Transmission";
  int x,y,count_i;
  double totalenergyit_d=0;
  char temp_pc[2];
  graphicson();
  temp_pc[1]='\0';
  _setcolor(2);
  _moveto(25,-25);
  _lineto(25,-(vc.numypixels-10));
  _moveto(25,-25);
  _lineto(vc.numxpixels-10,-25);
  for (count_i=0; count_i<14; count_i++) {
    _settextposition(count_i+1,1);
    temp_pc[0]=ble_ver_label_PC[count_i];
    _outtext(temp_pc);
    }
  _settextposition(vc.numtextrows-1,5);
  _outtext("Total Energy");
  _settextposition(vc.numtextrows,5);
  _outtext("Press return:");
  _setcolor(3);
  for (count_i=0; count_i<pointcount_i; count_i++) {
    totalenergyit_d=totalenergyit_d+energyi_pd[count_i];
    x=25+totalenergyit_d*(vc.numxpixels-30)/totalenergyi_d;
    y=25-(vc.numypixels-30)*energyt_pd[count_i]/energyi_pd[count_i];
    if (x>25 && y<-25 && x<vc.numxpixels && y>-vc.numypixels) _setpixel(x,y);
    }
 getchar();
 graphicsoff();
 }
```

```
/*****************************************************************************/
/*  laser.c                                                                  */
/*****************************************************************************/


/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>
#include "pclerrs.c"          /* Include PCLAB error codes for Microsoft C */
#include "expconf.c"


/*****************************************************************************/
/*  GLOBAL VARIABLES                                                         */
/*****************************************************************************/

int gain_i[MAX_DET]={1,1,1,1,1,1,1,1};
int gainold_i[MAX_DET]={1,1,1,1,1,1,1,1};
int capacitorgain_i[MAX_DET]={0,0,0,0,0,0,0,0};

extern union config_type_union config_u;


/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


initpclab() {
   int unit_id_i,retu_i;
   bool valid_board_b;
   valid_board_b = TRUE;
   config_u.config_st.board_type_i=0;
   SET_ERROR_CONTROL_WORD(0);
   retu_i=INITIALIZE();           /* Initialize the PCLAB subroutines */
   if (retu_i==0) {
       retu_i=RESET_DT(&unit_id_i);        /* perform a board reset */
       if (retu_i==0) {
         config_u.config_st.board_type_i = (unit_id_i / 16);  /* mask out Board
ID. code */
         config_u.config_st.microcode_i = (unit_id_i & 0xf);  /* mask out Microc
ode Rev. */
         }
       }
   switch (config_u.config_st.board_type_i) {
      case DT2801_A:
         ENABLE_FOR_OUTPUT(0);
```

```
            ENABLE_FOR_OUTPUT(1);
            OUTPUT_DIGITAL_VALUE(1,0xf,0x0);
            break;
         default:
            valid_board_b = FALSE;  /* set valid_board to false */
            retu_i=errorwno(" Hardware Error ",
                     "Unable to initialize DT2801-A A/D board.",
                     "Verify installation of board and pcldrv.sys.",
                     "PCLAB error number (see appendix A-1) ",retu_i);
            if (retu_i=='C') valid_board_b=TRUE;
         }
      return(valid_board_b);
      }


readdet(det_pd)
   double *det_pd;
   {
   int ad_data_i[MAX_DET];
   int channel_i;
   double v_range_d;
   double lsb_d, scaled_low_d, scaled_lsb_d;
   v_range_d = (config_u.config_st.high_voltage_d - config_u.config_st.low_volta
ge_d); /* total voltage range */
   lsb_d = (v_range_d/config_u.config_st.noc_d);                /* voltage of Lea
st significant bit */

   for (channel_i=0; channel_i<3; channel_i++) {
     ad_data_i[channel_i]  = 0x0000;
     ADC_VALUE(config_u.config_st.detchannel_i[channel_i],gain_i[channel_i],&ad_
data_i[channel_i]);
     }
   for (channel_i=0; channel_i<3; channel_i++) {
     scaled_lsb_d = (lsb_d / gain_i[channel_i]);          /* calculate scaled LSB
 */
     scaled_low_d = (config_u.config_st.low_voltage_d / gain_i[channel_i]); /* c
alculate scaled low voltage */

     det_pd[channel_i] = (ad_data_i[channel_i] * scaled_lsb_d) + scaled_low_d;
     det_pd[channel_i] = - det_pd[channel_i]
                         * config_u.config_st.det_calib_pd[channel_i]
                         * config_u.config_st.det_scale_pd[channel_i]
                         / config_u.config_st.det_apera_pd[channel_i];
     gainold_i[channel_i]=gain_i[channel_i];
     if (ad_data_i[channel_i] <0x80 && gain_i[channel_i] >1 ) {
       gain_i[channel_i]=gain_i[channel_i]/2;
       }
     if (ad_data_i[channel_i] >0x500 && gain_i[channel_i]<8) {
       gain_i[channel_i]= gain_i[channel_i]*2;
       }
     }
   }


printdetdata(det_pd, disp_ppw)
   double *det_pd;
   WINDOWPTR disp_ppw;
   {
   int count2_i;
   for (count2_i=0; count2_i<3; count2_i++) {
     v_printf(disp_ppw,"%+7.4fmJ/cm2(%1i) ",det_pd[count2_i],gainold_i[count2_i]
);
     }
   v_printf(disp_ppw,"\n");
   }
```

```
openshutter() {
  OUTPUT_DIGITAL_VALUE(0,0x0002,0);
  }


closeshutter() {
  OUTPUT_DIGITAL_VALUE(0,0x0002,2);
  }

gain0() {
  OUTPUT_DIGITAL_VALUE(1,0xf,0x0);
  }

gain1() {
  OUTPUT_DIGITAL_VALUE(1,0xf,0xf);
  }

firesignal0() {
  OUTPUT_DIGITAL_VALUE(0,0x1,0x0);
  }

firesignal1() {
  OUTPUT_DIGITAL_VALUE(0,0x1,0x1);
  }


laserready(title_pc)
  char *title_pc;
  {
  bool retu_b;
  retu_b=querybool(title_pc,"Laser must be 'ON', in triggered mode",
           "and burst count set=1.",
           "Are all of the above true? (Y/N):",TRUE);
  return(retu_b);
  }


firelaser() {
  long count_l;
  count_l=1;
  while (count_l <config_u.config_st.fire_delay_countl_l) {
    count_l=count_l+1;
    }
  OUTPUT_DIGITAL_VALUE(0,0x0001,1);
  count_l=1;
  while (count_l <config_u.config_st.fire_delay_counth_l) {
    count_l=count_l+1;
    }
  OUTPUT_DIGITAL_VALUE(0,0x0001,0);
  }



pulsetoenergy(target_d,pulsecount_pl,totali_pd,totalt_pd,totalr_pd)
  long *pulsecount_pl;
  double target_d;
  double *totali_pd,*totalt_pd,*totalr_pd;
  {
  double detectors_pd[MAX_DET];
  *totali_pd=0;
  *totalt_pd=0;
  *totalr_pd=0;
  *pulsecount_pl=0;
```

```
while (*totali_pd< target_d) {
  firelaser();
  readdet(detectors_pd);
  *totali_pd=*totali_pd + detectors_pd[0];
  *totalt_pd=*totalt_pd + detectors_pd[1];
  *totalr_pd=*totalr_pd + detectors_pd[2];
  *pulsecount_pl=*pulsecount_pl+1;
  }
return(0);
}
```

```
/*****************************************************************************/
/*  query.c                                                                  */
/*  general routines to query the user or display information                */
/*****************************************************************************/



/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#define WN_DEBUG              /* enable window debugging system */
#define F_FLOAT               /* enable floating point in WFD */
#include <wfd.h>


/*****************************************************************************/
/*  GLOBAL STRING DEFINITIONS                                                */
/*****************************************************************************/

static char picture_string60_PC[]="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXX";



/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/



querybool(title_pc, mess1_pc, mess2_pc, prompt_pc, default_b)
  char *title_pc, *mess1_pc, *mess2_pc, *prompt_pc;
  bool default_b;
  {
  DFORMPTR form_pfm;
  DFIELDPTR temp_pfd;
  int reply_b;
  reply_b=default_b;
  form_pfm=fm_def(8, 10, 8, 60, LHIGHLITE, BDR_DLNP);
  sfm_opt(VERIFYEXIT | CURSORFREE, OFF, form_pfm);
  sfm_opt(AUTOEXIT | AUTOMOVE, ON, form_pfm);
  sw_name(title_pc, form_pfm->wnp);
```

```
    ftxt_def( 1,CENTER_TEXT,mess1_pc, LNORMAL, form_pfm);
    ftxt_def( 2,CENTER_TEXT,mess2_pc, LNORMAL, form_pfm);
    fld_def(4, 30-strlen(prompt_pc)/2, prompt_pc,FADJACENT,"A",F_BOOL,
           (char *) &reply_b, form_pfm);
    fm_proc(0,form_pfm);
    fm_free(form_pfm);
    return(reply_b);
    }


querychar(title_pc,mess1_pc,mess2_pc,prompt_pc,default_pc)
    char *title_pc,*mess1_pc,*mess2_pc,*prompt_pc;
    char *default_pc;
    {
    DFORMPTR form_pfm;
    DFIELDPTR temp_pfd;
    form_pfm=fm_def(8,10,8,60,LHIGHLITE,BDR_DLNP);
    sfm_opt(VERIFYEXIT | CURSORFREE, OFF, form_pfm);
    sfm_opt(AUTOEXIT | AUTOMOVE, ON, form_pfm);
    sw_name(title_pc,form_pfm->wnp);
    ftxt_def( 1,CENTER_TEXT,mess1_pc, LNORMAL, form_pfm);
    ftxt_def( 2,CENTER_TEXT,mess2_pc, LNORMAL, form_pfm);
    fld_def(4, 30-strlen(prompt_pc)/2, prompt_pc,FADJACENT,"!",F_CHAR,
           (char *) default_pc, form_pfm);
    fm_proc(0,form_pfm);
    fm_free(form_pfm);
    return(1);
    }



querybool2(prompt_pc,default_b)
    char *prompt_pc;
    bool default_b;
    {
    DFORMPTR form_pfm;
    DFIELDPTR temp_pfd;
    int reply_b;
    reply_b=default_b;
    form_pfm=fm_def(23,0,1,80,LHIGHLITE,BDR_OP);
    sfm_opt(VERIFYEXIT | CURSORFREE, OFF, form_pfm);
    sfm_opt(AUTOEXIT | AUTOMOVE, ON, form_pfm);
    fld_def(0,0, prompt_pc,FADJACENT,"A",F_BOOL,
           (char *) &reply_b, form_pfm);
    fm_proc(0,form_pfm);
    fm_free(form_pfm);
    return(reply_b);
    }



fileerror(errormess_pc,filename_pc)
    char *errormess_pc,*filename_pc;
    {
    genmessage(" File Access Error ",strerror(errno),filename_pc,errormess_pc);
    }


errorwno(errormesstit_pc,errormess1_pc,errormess2_pc,errormess3_pc,errorno_i)
    char *errormesstit_pc,*errormess1_pc,*errormess2_pc,*errormess3_pc;
    int errorno_i;
    {
    char concat_pc[120],buffer[20];
    strcpy(concat_pc,errormess3_pc);
    strcat(concat_pc,itoa(errorno_i,buffer,10));
    genmessage(errormesstit_pc,errormess1_pc,errormess2_pc,concat_pc);
```

```
}


genmessage(errormesstit_pc,errormess1_pc,errormess2_pc,errormess3_pc)
  char *errormesstit_pc,*errormess1_pc,*errormess2_pc,*errormess3_pc;
  {
  DFORMPTR form_pfm;
  char key_c = ' ';
  form_pfm=fm_def(8,10,8,60,LHIGHLITE,BDR_DLNP);
  sfm_opt(VERIFYEXIT | CURSORFREE, OFF, form_pfm);
  sfm_opt(AUTOEXIT | AUTOMOVE, ON, form_pfm);
  sw_name(errormesstit_pc,form_pfm->wnp);
  ftxt_def( 1,CENTER_TEXT,errormess1_pc, LNORMAL,
    form_pfm);
  ftxt_def( 2,CENTER_TEXT,errormess2_pc, LNORMAL, form_pfm);
  ftxt_def( 3,CENTER_TEXT,errormess3_pc, LNORMAL, form_pfm);
  fld_def(4, 22,
         "Press a key: ",FADJACENT,"!",F_CHAR,
         (char *) &key_c, form_pfm);
  fm_proc(0,form_pfm);
  fm_free(form_pfm);
  return(key_c);
  }


getexpdata(qtit_pc,qdescription_pc,qvar1_pc,qvar2_pc)
  char *qtit_pc, *qdescription_pc, *qvar1_pc, *qvar2_pc;
  {
  DFIELDPTR temp_pfd;
  DFORMPTR form_pfm;
  form_pfm=fm_def(8,8,8,65,LHIGHLITE,BDR_DLNP);
  sfm_opt(AUTOMOVE | VERIFYEXIT, OFF, form_pfm);
  sfm_opt(AUTOEXIT | CURSORFREE, ON, form_pfm);
  sw_name(qtit_pc,form_pfm->wnp);
  ftxt_def(1,CENTER_TEXT,qdescription_pc, LNORMAL, form_pfm);
  temp_pfd=fld_def(3, 1,"",FADJACENT,picture_string60_PC,F_STRING,
         (char *) qvar1_pc, form_pfm);
  sf_opt(INITIALBLANKS+RTADJUST,OFF,temp_pfd);
  fld_tdef(temp_pfd,4, 1,"",FADJACENT,picture_string60_PC,F_STRING,
         (char *) qvar2_pc, form_pfm);
  fm_proc(0,form_pfm);
  fm_free(form_pfm);
  return(0);
  }


getfilename(qtit_pc,q1_pc,reply_pc)
  char *qtit_pc,*q1_pc,*reply_pc;
  {
  DFORMPTR form_pfm;
  strcpy(reply_pc,"         ");
  strcpy(reply_pc,"");
  form_pfm=fm_def(8,10,8,60,LHIGHLITE,BDR_DLNP);
  sfm_opt(AUTOMOVE | VERIFYEXIT | CURSORFREE, OFF, form_pfm);
  sfm_opt(AUTOEXIT, ON, form_pfm);
  sw_name(qtit_pc,form_pfm->wnp);
  ftxt_def( 2,CENTER_TEXT,q1_pc, LNORMAL, form_pfm);
  fld_def(4, 21,
         "Filename: ",FADJACENT,"XXXXXXXX",F_STRING,
         (char *) reply_pc, form_pfm);
  fm_proc(0,form_pfm);
  fm_free(form_pfm);
  return(0);
  }
```

```
/*****************************************************************************/
/*  stepper.c                                                                */
/*****************************************************************************/


/*****************************************************************************/
/*  SYSTEM INCLUDE FILES                                                     */
/*****************************************************************************/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>


/*****************************************************************************/
/*  OTHER INCLUDE FILES                                                      */
/*****************************************************************************/
#include <general.h>
#include "expconf.c"


/*****************************************************************************/
/*  GLOBAL VARIABLES                                                         */
/*****************************************************************************/
unsigned stepperport=0x2c0;
unsigned limitport=0x2c7;
bool stepperok_b=TRUE;
bool stagehomed_b=FALSE;
int stageposition_i=0;
int stagedirection_i=0;

extern union config_type_union config_u;



/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


initstepper() {
  char ch_c;

  stepperok_b=TRUE;

  outp(stepperport,0x8a);
  outp(stepperport+1,0x8a);
  outp(stepperport,0x9a);
  outp(stepperport+1,0x9a);

  dochara(1,0x49);          /* Initialize Cy512 */
  dochara(1,0);
```

```
    dochara(2,0x49);
    dochara(2,0);

    dochara(1,0x46);          /* Set Factor Parameter */
    dochara(1,1);
    dochara(1,(unsigned char)config_u.config_st.stepperfactor_i % 256);
    dochara(2,0x46);
    dochara(2,1);
    dochara(2,(unsigned char)config_u.config_st.stepperfactor_i % 256);

    dochara(1,0x41);          /* At Home command to guarantee we can move */
    dochara(1,0);
    dochara(2,0x41);
    dochara(2,0);

    dochara(1,0x52);          /* Set rate parameter */
    dochara(1,1);
    dochara(1,(unsigned char)config_u.config_st.stepperrate_i % 256);
    dochara(2,0x52);
    dochara(2,1);
    dochara(2,(unsigned char)config_u.config_st.stepperrate_i % 256);

    dochara(1,0x53);          /* set slope parameter */
    dochara(1,1);
    dochara(1,(unsigned char)config_u.config_st.stepperslope_i % 256);
    dochara(2,0x53);
    dochara(2,1);
    dochara(2,(unsigned char)config_u.config_st.stepperslope_i % 256);

    dochara(1,0x2b);          /* set clockwise direction */
    dochara(1,0);
    dochara(2,0x2b);
    dochara(2,0);
    stepleft(1,1);
    stepright(1,1);
    return(1);
    }


stepclockwise(motor_i)
  int motor_i;
  {
  dochara(motor_i,0x2b);          /* set clockwise direction */
  dochara(motor_i,0);
  delaystepper();
  stagedirection_i=1;
  }


stepcclockwise(motor_i)
  int motor_i;
  {
  dochara(motor_i,0x2d);          /* set counter clockwise direction */
  dochara(motor_i,0);
  delaystepper();
  stagedirection_i=2;
  }



stepleft(motor_i,stepsize_i)
  int motor_i;
  int stepsize_i;
  {
  unsigned char input_uc;
  unsigned char high_uc;
```

```
    if (stepperok_b==TRUE) {
      if (stagedirection_i!=1) stepclockwise(motor_i);
      input_uc=inp(limitport);
      if (( input_uc & 0x01) == 1 ) {
        dochara(motor_i,0x4e);
        dochara(motor_i,2);
        dochara(motor_i,(unsigned char)stepsize_i % 256);
        high_uc=stepsize_i/256;
        dochara(motor_i,high_uc);
        dochara(motor_i,0x47);
        dochara(motor_i,0);
        waitstepper(motor_i);
        stageposition_i=stageposition_i-stepsize_i;
        }
      }
    }


stepright(motor_i,stepsize_i)
  int motor_i;
  int stepsize_i;
  {
  unsigned char input_uc;
  unsigned char high_uc;
  if (stepperok_b==TRUE) {
    if (stagedirection_i!=2) stepcclockwise(motor_i);
    input_uc=inp(limitport);
    if (( input_uc & 0x02) == 2 ) {
      dochara(motor_i,0x4e);
      dochara(motor_i,2);
      dochara(motor_i,(unsigned char)stepsize_i % 256);
      high_uc= stepsize_i/256;
      dochara(motor_i,high_uc);
      dochara(motor_i,0x47);
      dochara(motor_i,0);
      waitstepper(motor_i);
      stageposition_i=stageposition_i+stepsize_i;
      }
    }
  }


findhome(motor_i)
  int motor_i;
  {
  int count_i;
  unsigned char input_uc;
  if (stepperok_b==TRUE && stagehomed_b==FALSE) {
    input_uc=inp(limitport);
    while (( input_uc & 0x01) == 1 && stepperok_b==TRUE ) {
      stepleft(motor_i,config_u.config_st.autostepsize_i);
      input_uc=inp(limitport);
      }
    for (count_i=0; count_i<2; count_i++) {
      stepright(motor_i,config_u.config_st.autostepsize_i);
      }
    input_uc=inp(limitport);
    while (( input_uc & 0x01) == 1 && stepperok_b==TRUE ) {
      stepleft(motor_i,1);
      input_uc=inp(limitport);
      }
    stepright(motor_i,config_u.config_st.centeroffset_i);
    stageposition_i=0;
    stagehomed_b=TRUE;
    }
  movehome(motor_i);
```

```
  return(1);
  }


movehome(motor_i)
  int motor_i;
  {
  if (stageposition_i<0) {
    stepright(motor_i,-stageposition_i);
    }
  if (stageposition_i>0) {
    stepleft(motor_i,stageposition_i);
    }
  }



dochara(motor_i,databyte_uc)
  unsigned char databyte_uc;
  int motor_i;
  {
  if (motor_i >0 && motor_i<3) {
    checkready(motor_i);
    outp(stepperport+2,databyte_uc);
    outp(stepperport+motor_i-1,0x9a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x1a);
    outp(stepperport+motor_i-1,0x9a);
    }
  }


checkready(motor_i)
  int motor_i;
  {
  int count_i;
  long count_l;
  unsigned char ready_uc;
  ready_uc=0;
  for (count_i=0; count_i<config_u.config_st.readydelay_i; count_i++) {
    }
  count_l=0;
  while (ready_uc < 0xa0  && count_l++ <40000) {
    ready_uc=inp(stepperport+motor_i -1);
    }
  for (count_i=0; count_i<config_u.config_st.readydelay_i; count_i++) {
    }
  return(0);
  }


waitstepper(motor_i)
  int motor_i;
  {
  long count_l=0;
  unsigned char ready_uc;
  ready_uc=255;
  while (ready_uc != 0 && count_l<1000000) {
    ready_uc=(inp(stepperport+motor_i -1) & 0x40);
```

```
    count_1=count_1+i;
    }
  if (count_1>999999) {
    stepperok_b=FALSE;
    genmessage(" Stage Error ","Stage not responding properly.",
               "Program will continue, but linear stage will not",
               "function.");
    }
  return(0);
  }


delaystepper() {
  long count_1;
  for (count_1=0; count_1<config_u.config_st.stepperdelay_1; count_1++) {
    }
  }
```

```
/*                                  PCLERRS.C

  Global error definitions.  these error codes are returned from PCLAB
  subroutines.

*/

#define E_OUTTMO        7       /* user timeout exceeded on output */
#define E_INPTMO        6       /* user timeout exceeded on input */
#define E_DMABSY        5       /* DMA channel currently busy */
#define E_2FAST         4       /* board clocked too fast */
#define E_NORMAL        0       /* success! */
#define E_NOPCL         -1      /* device driver PCL not found */
#define E_MANYFIL       -2      /* too many files open */
#define E_GATING        -4      /* illegal gating source for clock routines */
#define E_NOTFUNC       -5      /* board not capable of requested function */
#define E_DMAFREE       -6      /* DMA not currently in use */
#define E_DMAASN        -7      /* DMA channel not assigned to unit */
#define E_ILLSBX        -8      /* illegal iSBX slot, chip select, or channel
                                   number */
#define E_GAIN          -10     /* illegal gain specification */
#define E_DMABND        -13     /* DMA buffer crosses 64K boundary */
#define E_ILLFUN        -15     /* illegal function call */
#define E_NVALUE        -16     /* non-positive number of values argument */
#define E_NOFUNC        -18     /* unimplemented function call */
#define E_TIMING        -19     /* illegal timing source value */
#define E_HIFREQ        -22     /* requested frequency too high */
#define E_LOFREQ        -23     /* requested frequency too low */
#define E_SMALLP        -24     /* requested period too small */
#define E_LARGEP        -25     /* requested period too large */
#define E_CLKDIV        -26     /* illegal clock divider */
#define E_PORT          -27     /* illegal digital port */
#define E_CLKTIM        -28     /* illegal clock period or clock freq.
                                   period or frequency too small or
                                   too large */
#define E_CHANNL        -29     /* illegal channel number */
#define E_DACSEL        -30     /* illegal DAC select */
#define E_BOARD         -31     /* illegal board number */
#define E_STARTC        -32     /* illegal start channel */
#define E_NODIN         -33     /* DIO port enabled for output */
#define E_NODOUT        -34     /* DIO port enabled for input */
#define E_TYPE          -35     /* Illegal thermocouple type. Must be ASCII of
                                   B, E, J, K, R, S, T upper or lower case */
#define E_OVTAB         -36     /* specified voltage in routines XVTD, or XMT
                                   is not in the thermocouple type's range */
#define E_OTTAB         -37     /* temperature specified in routines XDTV or
                                   XMT is not in thermocouple type's range */
#define E_CJC           -38     /* compensated voltage (DT707-T channel zero)
                                   is out of linear range. Linear range
                                   is 0 degrees C to  +40 degrees C */
#define E_INIDAC        -42     /* DAC not initialized */
#define E_INIADC        -43     /* ADC not initialized */
#define E_INPUT         -44     /* board timeout on input */
#define E_OUTPUT        -45     /* board timeout on output */
#define E_READY         -46     /* board timeout on ready */
#define E_ILIV          -49     /* Voltage value in routine XATV is not between
                                   current board's negative full scale and
                                   positive full scale */
#define E_UNEXP         -100    /* unexpected error */

/*  end PCLERRS.C  */
```

Index

Index