U.S. DEPARTMENT OF THE INTERIOR

U.S. GEOLOGICAL SURVEY

A Control and Data Acquisition System for Use with a Hydrothermal Diamond-Anvil Cell

by

H. T. Haselton, Jr. and I-Ming Chou[1]

Open File Report 94-703

[1] U.S Geological Survey, 959 National Center, Reston, VA 22092

1994

Introduction

The diamond-anvil cell (DAC) described by Bassett et al. (1993) was designed for use in hydrothermal experimentation to pressures of 25 kbar and temperatures in the range of -190 to 1200°C. The purpose of this report is to describe ancillary components used in a computer based (PC-AT compatible) temperature control and NTSC video system for use with the DAC as set up in our U.S. Geological Survey laboratory in Reston, VA. In addition, we describe some of the maintenance procedures required to operate the cell. Descriptions are in the order (1) hydrothermal diamond-anvil cell and maintenance notes, (2) temperature control, (3) microscope and laser, and (4) video acquisition and frame capture. Sources of components and consumables and a listing of the computer program are attached as appendices.

Hydrothermal Diamond-Anvil Cell

The partially dissembled DAC and a schematic diagram of the diamond seats and heaters are shown in Fig. 1. Some of the key design features which facilitate hydrothermal experimentation are (1) a molybdenum wire resistance heater wound on the tungsten carbide seat of each diamond, (2) a type K thermocouple cemented to each diamond adjacent to the culet face, (3) attachment of heater and thermocouple leads to the cell by means of pin connectors, and (4) hollow base and posts through which a forced air flow can reduce dimensional changes in the cell due to heating and cooling. Because of these features, the temperature of each diamond can be controlled precisely and accurately; connections of electrical leads are secure and easily made; and the sample tends to maintain constant volume during heating. This DAC was fully described by Bassett et al. (1993); however, it has undergone several evolutionary design changes since described in response to user needs, the most significant changes being a reduction in overall height and modification of the upper seat parts to increase the accessible angle for incident and emergent radiation. The reduction in cell height increases the likelihood that it will fit in the sample chamber of spectrophotometers and on the stages of commercial microscopes without modifications or removal of components. A source list of components and consumable items, including the DAC, is given in Appendix A.

The thermocouples and heaters will need to be replaced with a frequency that depends on the temperature range of experimentation and the care of the user. The replacement of a thermocouple is straightforward as long as some care is taken to insulate it from the diamond seat. The following is a brief description of our procedure for winding a heater. To facilitate the winding process, we have constructed a jig (Fig. 2) which fits under a binocular microscope. Alternatively, a small format lathe (Bassett, oral communication, 1992) can also be adapted for this purpose. After the tungsten carbide seat has been cleaned and attached to the stainless steel axle of the winding jig with thermoplastic cement such as Buehler Lakeside 70C, a coating of thinned alumina cement is applied to the outside surface to electrically insulate the seat from the heater winding. We currently use 12 turns of 0.010" (0.254 mm) diameter wire for each heater. The leads of the heater tend to become fragile with use so the leads are doubled and twisted to lower their temperature. The doubling increases the life of the winding considerably. We find that the amount of wire needed for a winding is fairly constant (24.1 cm between twisted segments), we prepare the twisted leads at both ends before winding the heater. We do the twisting by hand; about 5 cm is sufficient. When winding the heater, we attach a weight to the free end of the wire in order to maintain some tension; hence, we postpone trimming the length of the free end until after the heater has been wound. Alternatively, the leads can be twisted after the winding is cemented to the seat; however, we find that the cement is somewhat delicate and commonly cracks during the twisting.

The end of the wire with the twisted lead is taped to the winding jig so that the twisted segment just reaches the tungsten carbide seat. With a binocular microscope it is not difficult to wind the furnace and maintain a spacing of half of the wire diameter or about 0.12 mm. After winding the heater, the free end is taped to the jig and the excess wire is cut off. The spacing of the winding is checked with particular attention given to the space between the twisted leads and the adjacent turn of wire. The application of several thin coats of cement anchors the winding to the seat. After the initial coats of cement are hard, the lead from the base of the seat is gently bent into position adjacent to the other lead. When working with

the bottom heater, this lower lead must lie fairly close to the winding so that the prepared seat fits properly in its stainless steel support. Additional cement is applied, and we build up fillets between the leads and winding for additional strength. The thermoplastic cement is softened with a heat gun and the seat is removed from the spindle. At this point, ensure that the base of seat is flat and that all thermoplastic cement has been removed. The seat is now cemented into its stainless steel holder. The orientation of the heater leads are checked to ensure that when the alignment screws on the bottom holder are engaged, the leads are directed toward the pin connectors. If the top seat has a radiation slot, the orientation of the heater leads is also important. Here, the alumina cement should be as thick as possible for this application in order to avoid getting cement between bearing surfaces. We have tried a couple of ways for holding the seat, insulating disc, and holder in alignment for this operation, but we are not totally satisfied with our methods. Finally, additional cement is applied to further secure the seats in their holders.

The seat assemblies are secured in the platens and the heater leads are attached to the pin connectors. The heater leads should be gently curved. Thermocouples are cemented in place and attached to the connector posts. Conductive silver paint is applied to the connections for both the heater leads and thermocouples. The cell is now ready for alignment.

A note on Ceramabond 569 alumina cement (Aremco Materials): This particular alumina cement sets up quickly, can be thinned with water, is quite tough, and is readily shaped using tungsten carbide scribes. Excessive thinning of the cement is undesirable, however, because it promotes shrinkage cracking. Excessive thinning of the cement is particularly undesirable when cementing diamonds into their seats. They will tend to loosen requiring the application of additional cement and a check of the alignment. The application of thin coats and the use of a hot air gun between successive applications of cement is helpful.

Temperature Control

Fabricated Type K thermocouples (0.003" (0.076 mm) diameter bare-wire) are available from Omega Engineering. The welded beads of these thermocouples are small and consistent. Thermocouple voltages are read with a PC by means of a Strawberry Tree MINI-16 A/D board (see Fig. 3 for a schematic of components related to the temperature control and video system). This board uses an external terminal panel with junction temperature compensation and can read up to 8 thermocouples with up to 16-bit resolution on a -5 to 50 mV range. For a Type K thermocouple, these specifications correspond to an ideal resolution of about 0.02°C in the range -50 to 1250°C. Of course, actual measurement precision is degraded somewhat due to electrical noise, but at moderate temperatures, temperature and gradient control to 0.1°C is practical. Power output to the DAC is controlled by a Strawberry Tree ACAO-12-2 D/A board which has two output channels with 12-bit resolution. We use a 4-20 mA output from the board to control a Eurotherm 15 A, phase angle fire power supply. At present, the two DAC heaters with trimmers are wired in parallel to a single supply. We intend to incorporate a second power supply so that we can heat the diamonds independently and control temperature gradients automatically. The Eurotherm 15 A power supplies are much larger than needed, but we have not found an appropriate substitute. The maximum power needed for both heaters is about 150 W.

Initially, we read the thermocouples directly with the MINI-16 board; however, at temperatures above about 450°C, signal noise which increased greatly with increasing temperature, became a severe problem. Grounding of thermocouples and electronic components helped somewhat but the situation was still unacceptable. We attribute the source of the noise to the switching of the Eurotherm power supply. The switching induces a voltage spike on the thermocouples which is not averaged out by the "low-noise" read mode of the MINI-16 software driver.

One solution to the noise problem is to read the thermocouples with meters that provide greater noise compensation and provide an output signal which is linearly proportional to temperature. This output signal from the meter is then read by the MINI-16 board. At present, we are using Newport Infinity series meters to read the thermocouples and the MINI-16 board to read the output signal from the Newport meters. The 0-10 volt DC signal from the Newport meters has been stable over the approximately 8

month period that we have been using them. In our system, the temperature reading on the Newport meters deviate slightly (maximum of 1-2°C at full scale) from the reading from the MINI-16 board. Because the readings from the MINI-16 are a permanent part of our video record, we use its readings and periodically check the response of the system to a millivolt source. Use of programmable DC power supplies may be a better solution that would allow us to bypass the Newport meters.

Another flaw in this system is random initial output current or voltage of the ACAO-12 D/A board when the computer boots. Until the board is initialized, the output is unspecified; therefore, if a power supply is left on by mistake, the cell windings may be burned out by a high initial current. Only when the temperature control program is started, is the output of the ACAO board set to zero.

For temperatures below ambient, a stream of cold nitrogen gas is directed at the sample. In our system, we can mix N_2 gas taken from the top of the dewar with liquid N_2. By changing the flow rate, temperature, and orientation of the gas stream, we have controlled the temperature and gradient precisely at temperatures above about -50°C. At lower temperatures, the gas stream tends to be a 2-phase mixture and the flow pulses; as liquid nitrogen hits the anvils and gasket, the sample cools very rapidly. With some adjustment of the nitrogen stream, precise control at temperatures below -50°C should be possible.

Temperature Acquisition and Heater Control Program

An acquisition and control program written in C is listed in Appendix B. The code makes extensive use of graphics and event processing functions in the MetaWindows package from MetaGraphics Software Corp. Functions for reading and writing to the MINI-16 and ACAO-12-2 boards rely on the MS_CALL.OBJ module provided with the boards by Strawberry Tree. Except for the two keyboard commands, Alt-x for exiting the program and Alt-r for redrawing the control area of the screen, all controls are mouse driven. In operation, the code cycles repeatedly through a loop checking for mouse or keyboard input. Time and temperature information is updated to the screen at one second intervals. The program is well-tested for manual-mode control. A function for automatic 3-term (PID) control of one power supply is included, but although it has been tested with a heater simulated in software, it has not been tested with the diamond cell. We plan eventually to use two power supplies, one per heater, and to implement ramp and gradient functions. At present, with the program in manual mode, trim resistors must be adjusted frequently in order to eliminate the temperature gradient between thermocouples.

He-Ne Laser and Microscope

Our microscope is a Leitz Aristophot from which the substage condenser system has been removed. The original 12 V transmitted light system provides sufficient illumination. The microscope is fixed to a 3' x 3' optical breadboard with vibration absorbing pads. If a laser system is added to detect phase transitions by shifting interference fringes, the vibration dampening of an optical is very desirable. With our present set up, the stability of the laser interference fringes is not satisfactory.

A low-power He-Ne laser is used to detect phase transitions by means of a shift in the interference fringes created by reflections from the top and bottoms surfaces of a doubly-polished sample. For example, the α–β quartz transition (Shen et al., 1993), is easily observed by this technique. The ability to easily adjust the intensity of the laser light is desirable so that its intensity can be adjusted to optimize the images of the sample and the interference fringes. The laser is held and oriented with commonly available optical components.

Video Acquisition and Frame Capture

A visual record of experiments is indispensable, and therefore we try to get as much information about the experiment on video tape as possible. This is the primary reason that we rely on a computer-based temperature control system. A monochrome video camera (786x488 element CCD) with NTSC

(RS-170A) output is mounted on the microscope with a parfocal c-mount adapter. The output from the camera is connected to a video overlay card (Truevision VideoVGA) in the PC which is then connected to a video recorder and monitor. As shown in Fig. 4, a vertical strip on the left hand side of the computer screen displays time and temperature information and contains the heater control buttons; the remainder of the computer screen is painted dark gray. By means of a resident utility, the dark gray portion of the computer screen is replaced in the output NTSC signal by the image from the video camera. The resulting NTSC video signal has the temperature and control information combined with the image of the sample. Audio information can also be recorded.

Video frames of interest are captured by means of a Data Translation DT-55 Vision EZ board. This monochrome board can digitize a NTSC frame (2 fields) at 640x480 pixels (256 gray levels) in real time (1/30 sec). Data Translation supplies both MS Windows version 3.1 and MS-DOS capture utilities. Although the board can digitize a frame in real time, the digitized image is not displayed in real time; hence, the video source is usually displayed on a second RGB monitor. We have found that the MS-DOS utility (QuickCapture) is more useful for our needs because fewer keystrokes are required to switch between viewing a live or captured image. Captured images, 307.2 kbyte each in TIFF format (also PCX and DT-IRIS) are relatively low resolution and tend to be of low contrast; therefore, some image modification is usually desirable. A second problem can occur when printing the image. Laser printers simulate levels of gray by varying the dot density. The resulting apparent resolution of the printer is lowered significantly, and printed images may not be sharp even when reduced in size.

The signal to noise ratio, and the resolution to a modest extent, of the video record can be improved by switching to s-video components. The principal advantage of s-video is that the noise level of edited tapes is less apparent.

References Cited

Bassett, W.A., Shen, A.H., Bucknam, M., and Chou, I-Ming (1993) A new diamond-anvil cell for hydrothermal studies to 2.5 GPa and from -190 to 1200°C. Reviews of Scientific Instruments v. 64, p. 2340-2345.

Shen, A.H., Bassett, W.A., and Chou, I-Ming (1993) The α–β quartz transition at high temperatures and pressures in a diamond-anvil cell by laser interferometry. American Mineralogist v. 78, p. 694-698.

Acknowledgments

Prof. William. A. Bassett is the expert on hydrothermal diamond anvil cells and he introduced us to many of the techniques and materials needed for diamond cell experimentation. We appreciate the reviews of this report by Harvey E. Belkin and Gary. L. Cygan.

Appendix A

Sources for many of the components and consumable items are given in the following lists. Prices are from the period 1992-94 and may have changed significantly for some items. Items for which the price is not given were obtained as a part of collaborative work or as a sample.

Diamond-Anvil Cell and Consumables

Hydrothermal Diamond Anvil Cell	W.A. Bassett Foxwood Instruments 765 Bostwick Road Ithaca, NY 14850 (605) 255-7502	
Type K 0.003" unsheathed thermocouples (pkg of 5), CHAL-003. Type K extension wire 24 awg. 50 ft, TT-K-24SLE Pin Connectors SMTC-CU-S copper socket SMTC-CU-P copper plug SMTC-CH-S chromel socket SMTC-CH-P chromel plug SMTC-AL-S alumel socket SMTC-AL-P alumel plug	Omega Engineering, Inc. P.O. Box 4047 Stamford, CT 06907-0047 (800) 826-6342	22.00 46.00 35.00 35.00 35.00 35.00 35.00 35.00
18 mm diameter micro cover glasses, No. 1 Red Label circles (box) 6662-F43	Thomas Scientific P.O Box 99 Swedesboro, NJ 08085-6099 (800) 345-2100	21.00
Silver Iodide (#11419) 99.9%, 25 grams	Johnson-Mathey / AESAR Group 892 Lafayette Rd. Seabrook, NH 03874-5511 (800) 343-1990	36.70
Type 1, 1/8 carat, low-birefringent diamonds (no culet facet) set of 12.	Lazar Kaplan 529 5th Ave. New York, NY 10017 (212) 972-9700 Attn: Diana Cuevas	3660.00
Engineering 30 liter liquid nitrogen dewar, Minnesota Valley Engineering.	AIRCO Gas & Gear 5 Rodney Lane Fredericksburg, VA 22401 (703) 373-1782	453.00
Molybdenum wire 0.010" diameter T5010280K (140 meters)	Philips Lighting 1560 Lisbon Road Lewiston, ME 04240 (800) 343-8008	
FeCrAl875 (Kanthal A-1) 0.010" diameter wire (3 lb spool)	National Electric 7 Executive Drive Toms River, NJ 08755 (908) 240-5383	150.00
Alumina cement Ceramabond 569 (pint)	AREMCO Products Inc. P.O. Box 429 23 Snowden Ave. Ossining, NY 10562 (914) 762-0685	75.00

Rhenium sheet 0.010", 0.015" thickness	Rhenium Alloys Box 245 1329 Taylor Street Elyria, OH 44036 (216)385-7388	

Computer Boards

VGA graphics adapter with NTSC overlay and ouput, 512 kbytes DRAM, register compatible VGA with simultaneous interlaced composite or s-video output, software, Truevision VideoVGA.	Source Computer 1420 Spring Hill Road, Suite 400 McLean, VA 22102 (703) 821-6800	995.00
Analog input board, 8 differential input channels, 16-bit resolution on 50mV scale, software, Strawberry Tree MINI-16. T21 thermocouple terminal panel and cable for MINI-16	Strawberry Tree, Inc. 160 South Wolfe Rd. Sunnyvale, CA 94086	995.00 270.00
Analog output board, 2 channels, 12-bit resolution, selectable outputs, software Strawberry Tree ACAO-12-2. T31 general purpose panel and cable for ACAO-12-2	Strawberry Tree, Inc. 160 South Wolfe Rd. Sunnyvale, CA 94086	395.00 149.00
Frame grabber , monochrome 640x480, 8-bit, software, Data Translation DT-55 Vision-EZ. EP277 cable for DT55	Data Translation 100 Locke Drive Marlboro, MA 01752-1192 (800) 525-8528	995.00 165.00

Heater Power Supply and Thermocouple Readouts

Heater power supply PAP 15 amp SCR (425/15A240V/115V4-20ADC/PA/CL//) Fuse Holder and fuse (409/15A250V) 15 A fuse (CH260024	Eurotherm Corporation 11485 Sunset Hills Road Reston, VA 22090 (703) 471-4870	372.00 20.00 5.50
Thermocouple meters, Newport Infinity series meter INFT0-0-1-1-KC	Newport Electronics, Inc. 2229 South Yale Street Santa Ana, CA 92704 (800) 639-7678	418.50

Video and Optical Components

High-resolution, B/W NTSC video monitor, 800 line horizontal resolution, Sony PVM-122	Professional Products 4964 Fairmont Ave. Bethesda, MD 22814-5090 (301) 657-2141	560.00
SVHS VCR with serial interface, NEC PV-S98A	Professional Products 4964 Fairmont Ave. Bethesda, MD 22814-5090 (301) 657-2141	1800.00
High-resolution video camera (786x488 element CCD) , Hitachi KP-161	Professional Products 4964 Fairmont Ave. Bethesda, MD 22814-5090 (301) 657-2141	887.00

C-mount adapter for Leitz Aristophot petrographic microscope, 1.0x, parfocal.	Diagnostic Instruments 6540 Burroughs Sterling Heights, MI 48314 (313) 731-6000	395.00
High-resolution color video monitor, 13", 750 line horizontal resolution, underscan, s-video and NTSC, Panasonic BT-H1350Y	Panasonic c/o CTI 9301 Georgia Ave. Silver Spring, MD 20910 (301) 585-6311	986.00
He-Ne laser head, 1.5 mW, polarized 500:1, 0.63 mm beam, angular drift cold start 0.1 mrad, Uniphase 1101P Power supply 1201-1 for model 1101P head	Uniphase 1096 Mellon Ave. Manteca, CA 95337 (209) 239-9348	365.00 310.00
SVHS VCR, JVC SR-S378U	Professional Products 4964 Fairmont Ave. Bethesda, MD 22814-5090 (301) 657-2141	977.00
Optical components Model C standard rod (qty. 2) Model 340-C rod clamp Model 300-P universal mounting platform Model 812 precision adjustable laser mount Model 900 spatial filter (M 10x, 900PH-25) Model 460A-XZ stage Model SM-05 micrometer (qty. 2)	Newport Corp. 1791 Deere Ave. Irvine, DA 92714 (714) 963-2015	 84.00 75.00 147.00 384.00 682.00 441.00 55.00

Appendix B

The following program has been used to read and control temperatures of the DAC in our experimental setup for about two years. It was written for a PC-AT compatible with VGA graphics. We make extensive use of the MetaWINDOW Plus, version 3.1, graphics library (Metagraphics Software Corp.) for event processing as well as screen graphics. The program was compiled with the C compiler in the Borland C++ package. With the exception of device control functions, the only significant deviation from ANSI C is the use of the C++ commenting convention. Copies of the current version of the program are available from the authors by e-mail, anonymous ftp, or floppy disk.

```
// ---------------------------------------------------------------------------
//  run_dac.c - routine for controlling diamond anvil cell heater
//
//              Strawberry Tree MINI-16 and ACAO a/d and d/a cards
//              Tren Haselton, USGS, Reston, VA  haselton@rgborafsa.er.usgs.gov
//                   Tel: (703) 648-6171  Fax: (703) 648-6252
//
//              2/17/93 added check for Alt-r to refresh screen in manual
//                      mode
//              1/26/94 cell top and bottom tempertures scaled from 4-20 ma
//                      input from Newports on block positions 4 and 5
//              3/02/94 changed current to temperature equations, vcr functions
//                      vcr_init and vcr_close commented out
//              3/04/94 cell top and bottom temperatures scaled from 0-10 vdc
//                      input from Newports on block positions 4 and 3
//             10/19/94 removed ability to increase furnace power by 10%
//                      rect check and button drawing removed
// ---------------------------------------------------------------------------

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <alloc.h>

#include "GRconst.h"          // MetaWINDOW headers
#include "GRports.h"
#include "GRextrn.h"

#define  BUT_X      11        // xy dimensions of buttons in pixels
#define  BUT_Y      11
#define  DBUT_X      9        // xy dim of up/down buttons in pixels
#define  DBUT_Y      7

#define  IN_ANACNT   8        // total # of analog input channels MINI-16
#define  OUT_ANACNT  2        // total # of analog output channels ACAO-2
#define  DIGCNT     20        // sum of digital i/o channels MINI=12, ACAO=8
#define  IN_CHAN     4        // actual # of analog-in channels in use
#define  OUT_CHAN    2        // actual # of analog-out channels in use

#define  POWER_LIMIT 100.0     // limit for analog out % of range (0 - 4095)

void controls_setup(void);
void video_area_setup(void);
void cleanup(void);
void time_temp(void);
void plot(float temperature);

void display_MINI16_config(void);
void calibrate_MINI16(void);
void initialize_MINI16(void);
void read_MINI16(void);

void auto_control(float set_point, float current_temp, int reset_flag);
int heater_power(float level);
void analog_out_ACAO(int out_val, int channel);

void vcr_init(void);
int com_to_vcr(char *out_str);
int vcr_close(void);

rect     sR, infoR, videoR,             // screen, info and video areas
         auto_B, manual_B,              // temperature control mode buttons
         power_uB[4], power_dB[4],      // power level buttons
         temp_uB[4], temp_dB[4],        // temperature set point buttons
```

```
        ramp_uB[4], ramp_dB[4],        // ramp rate buttons
        ok_B, reset_B,                 // ok and cancel on setpoint and ramp
        kb_uB, kb_dB,                  // PID proportional band buttons
        ki_uB, ki_dB,                  // PID integral buttons
        kd_uB, kd_dB;                  // PID derivative buttons

point pt;

char  s_str[80];               // general purpose string for screen writes
char  time_str[30];            // time string for screen write
char  *fnt1_ptr, *fnt2_ptr;    // pointers to screen fonts

float tc[IN_CHAN];             // temperatures returned from MINI-16

float power = 0.0;             // initial power set to zero
float kb =  0.30,              // PID parameters
      ki =  8.00,
      kd =  0.02;
float set_pt =   0.0, set_pt_new  =  0.0;    // initialization of set_point
float ramp =   100.0, ramp_new =   100.0;    // and ramp


int main(void)
{
   int       i;
   long      ltime;
   int       ramp_slope;
   int       ramp_on;
   int       ramp_reset;
   int       manual_flag = 1;

   struct tm *time_now;
   time_t    t, t0;


   float p_inc[] = {10.0, 1.0, 0.1, 0.01};    // power increments
   float r_inc[] = {100.0, 10.0, 1.0, 0.1};   // ramp increments
   float t_inc[] = {100.0, 10.0, 1.0, 0.1};   // set_point increments

   float ramp_del_t;             // temperature increment (/sec) during a ramp
   float ramp_set_pt;

   int       evCnt;
   int       load_err;                              // font file load error
   mapArray  b_map;                                 // cursor style array
   event     evnt;
   point     m_xy;                                      // mouse position

   // ---------------------------------------------------------------------

   // load graphics fonts - must reside in curent directory
   if ((fnt1_ptr = (char *)malloc(4000)) == NULL) {
      printf("Memory allocation error for font 1\n");
      exit(1);
   }
   if ((load_err = FileLoad("system16.fnt", fnt1_ptr, 4000)) < 0) {
      printf("FileLoad error - system16.fnt = %d\n", load_err);
      exit(1);
   }

   if ((fnt2_ptr = (char *)malloc(5000)) == NULL) {
      printf("Memory allocation error for font 2\n");
      exit(1);
   }
   if ((load_err = FileLoad("system96.fnt", fnt2_ptr, 5000)) < 0) {
      printf("FileLoad error - system96.fnt = %d\n", load_err);
      exit(1);
   }


   initialize_MINI16();              // check acquisition and control boards
   // vcr_init();

   InitGrafix(EGA640x480);                 // setup graphics, font, and mouse
   InitMouse(MsDriver);
   ScaleMouse(8,8);
   SetDisplay(GrafPg0);
   ScreenRect(&sR);
   SetFont((fontRec *)fnt1_ptr);

   controls_setup();                              // Draw initial screen
```

```
   video_area_setup();

   LimitMouse(0, 0, infoR.Xmax-10, infoR.Ymax);  // keep cursor in info Rect
   MoveCursor(sR.Xmax/12, sR.Ymax/2);            // initial cursor position

   for (i = 0; i < 8; i++)                        // fix cursor style - arrow
      b_map[i] = 2;
   CursorMap((mapArray far *)b_map);

   TrackCursor(1);                               // start tracking and queue
   EventQueue(True);

   while (KeyEvent(False, &evnt) ==  True)             // flush event queue
      ;

   evCnt = 0;                                             //event counter

   ShowCursor();
// ---------------------------------------------------------------------------

   do  {
      SetPt(&m_xy, evnt.CursorX, evnt.CursorY); // define point - cursor xy
      PenColor(0);

      if ((KeyEvent(False,&evnt))==True)  {            // start event queue
         ++evCnt;
         HideCursor();

         // check cursor position and process event only if the right mouse
         // button has been hit.
         if ((evnt.State & 0x0F00) == 0x0400)  {

            if (PtInRect(&m_xy, &auto_B)) {  // switch to auto mode
               BackColor(15); EraseRect(&manual_B); FrameRect(&manual_B);
               BackColor(13); EraseRect(&auto_B);   FrameRect(&auto_B);
               manual_flag = 0;
               ramp_reset  = 1;
            }
            if (PtInRect(&m_xy, &manual_B)) {  // switch to manual mode
               BackColor(13); EraseRect(&manual_B); FrameRect(&manual_B);
               BackColor(15); EraseRect(&auto_B);   FrameRect(&auto_B);
               manual_flag = 1;
            }

            // manual power control (permitted only in manual mode)
            PenColor(0); BackColor(7);
            if (manual_flag) {
               for (i = 1; i < 4; i++) {
                  if (PtInRect(&m_xy, &power_uB[i])) {         // boost power
                     power += p_inc[i];
                     if (power > POWER_LIMIT)
                        power = POWER_LIMIT;
                     heater_power(power);
                     MoveTo(85, 288);
                     sprintf(s_str,"%6.2f", power); DrawString(s_str);
                  }
               }
               for (i = 0; i < 4; i++) {
                  if (PtInRect(&m_xy, &power_dB[i])) {         // drop power
                     power -= p_inc[i];
                     if (power < 0.0)
                        power = 0.0;
                     heater_power(power);
                     MoveTo(85, 288);
                     sprintf(s_str,"%6.2f", power); DrawString(s_str);
                  }
               }

               // changes in PID parameters permitted only in manual mode
               // increasing proportional band, integral, or derivative increases
               // heater response to deviations from the setpoint.

               if (PtInRect(&m_xy, &kb_uB)) {       // boost proportional band
                  kb += 0.01;
                  sprintf(s_str, "%3.0f", kb * 100.);
                  MoveTo(18, 170); DrawString(s_str);
               }
               if (PtInRect(&m_xy, &kb_dB)) {        // drop proportional band
                  kb -= 0.01;
                  if (kb < 0.)
                     kb = 0.;
```

```
      sprintf(s_str, "%3.0f", kb * 100.);
      MoveTo(18, 170); DrawString(s_str);
   }

   if (PtInRect(&m_xy, &ki_uB)) {                  // boost integral
      ki += 1.0;
      sprintf(s_str, "%3.0f", ki);
      MoveTo(65, 170); DrawString(s_str);
   }
   if (PtInRect(&m_xy, &ki_dB)) {                   // drop integral
      ki -= 1.0;
      if (ki < 0)
         ki = 0;
      sprintf(s_str, "%3.0f", ki);
      MoveTo(65, 170); DrawString(s_str);
   }

   if (PtInRect(&m_xy, &kd_uB)) {                // boost derivative
      kd += 0.01;
      sprintf(s_str, "%4.2f", kd);
      MoveTo(110, 170); DrawString(s_str);
   }
   if (PtInRect(&m_xy, &kd_dB)) {                 // drop derivative
      kd -= 0.01;
      if (kd < 0)
         kd = 0;
      sprintf(s_str, "%4.2f", kd);
      MoveTo(110, 170); DrawString(s_str);
   }
}

for (i = 0; i < 4; i++) {                // check set_pt u/d buttons
   if (PtInRect(&m_xy, &temp_uB[i])) {
      set_pt_new += t_inc[i];
      if (set_pt_new > 999.9)
         set_pt_new = 999.9;
      sprintf(s_str, "%5.1f", set_pt_new);
      MoveTo(60, 241);
      PenColor(4); DrawString(s_str);
   }
   if (PtInRect(&m_xy, &temp_dB[i])) {
      set_pt_new -= t_inc[i];
      if (set_pt_new < -50.)
         set_pt_new = -50;
      sprintf(s_str, "%5.1f", set_pt_new);
      MoveTo(60, 241);
      PenColor(4); DrawString(s_str);
   }
}

for (i = 0; i < 4; i++) {                  // check ramp u/d buttons
   if (PtInRect(&m_xy, &ramp_uB[i])) {
      ramp_new += r_inc[i];
      if (ramp_new > 999.9)
         ramp_new = 999.9;
      sprintf(s_str,"%5.1f", ramp_new);
      MoveTo(115, 241);
      PenColor(4); DrawString(s_str);
   }
   if (PtInRect(&m_xy, &ramp_dB[i])) {
      ramp_new -= r_inc[i];
      if (ramp_new < 0.0)
         ramp_new = 0.0;
      sprintf(s_str,"%5.1f", ramp_new);
      MoveTo(115, 241);
      PenColor(4); DrawString(s_str);
   }
}

if (PtInRect(&m_xy, &ok_B)) {           // new set_point and ramp ok
   set_pt = set_pt_new;
   sprintf(s_str, "%5.1f", set_pt_new);
   MoveTo(60, 241);
   PenColor(0); DrawString(s_str);

   ramp = ramp_new;
   sprintf(s_str,"%5.1f", ramp);
   MoveTo(115, 241);
   PenColor(0); DrawString(s_str);

   ramp_reset = 1;
```

```
      }
      if (PtInRect(&m_xy, &reset_B)) {     // restore set_point and ramp
         set_pt_new = set_pt;
         sprintf(s_str, "%5.1f", set_pt_new);
         MoveTo(60, 241);
         PenColor(0); DrawString(s_str);

         ramp_new = ramp;
         sprintf(s_str,"%5.1f", ramp);
         MoveTo(115, 241);
         PenColor(0); DrawString(s_str);
      }


   }

   ShowCursor();

}  // end of queue processing for mouse hits

t = time(NULL);
time_now = localtime(&t);
strftime(time_str, 80,"%m/%d/%y %H:%M:%S", time_now);

// t, t0 comparison prevents multiple updates in the same second
if (t != t0) {

   MoveTo(12, 460); BackColor(7); DrawString(time_str);
   t0= t;

   // thermocouples read here
   read_MINI16(); // get cell and gas stream temperatures

   // update temperature info on screen ------------------------------
   sprintf(s_str,"%5.1f", tc[0]);              // top diamond temp in C
   if (tc[0] > tc[1])
      PenColor(4);
   else
      PenColor(1);
   MoveTo(68, 430); DrawString(s_str);

   sprintf(s_str,"%5.1f", tc[1]);           // bottom diamond temp in C
   if (tc[1] > tc[0])
      PenColor(4);
   else
      PenColor(1);
   MoveTo(68, 375); DrawString(s_str);

   sprintf(s_str, "%5.1f", tc[0]-tc[1]);               // delta T
   MoveTo(10, 403); PenColor(0); DrawString(s_str);

   sprintf(s_str,"%4.1f", tc[2]);                      // room temp in C
   MoveTo(123, 403); PenColor(0); DrawString(s_str);

   time_temp();

   // auto mode control section ---------------------------------------
   if (manual_flag == 0) {

      // reset ramp on switch to auto mode or new set_pt, ramp rate
      if (ramp_reset) {
         ramp_set_pt = tc[0];
         if (set_pt >= tc[0])
            ramp_slope =  1;
         else if (set_pt < tc[0])
            ramp_slope = -1;
         ramp_del_t = ramp_slope * ramp / 60.;
         ramp_on = 1;
         auto_control(ramp_set_pt, tc[0], 1); // eliminate transfer bump
         ramp_reset = 0;
      }
      // turn off ramping if abs(set_pt - tc[0]) <= abs(ramp_del_t)

      if ( ((ramp_slope >= 0.0) && (set_pt-tc[0] <= ramp_del_t))
        || ((ramp_slope <= 0.0) && (set_pt-tc[0] >= ramp_del_t)) )
         ramp_on = 0;

      if (ramp_on) {                          // ramp turned on
         auto_control(ramp_set_pt, tc[0], 0);
         ramp_set_pt += ramp_del_t;
      }
```

```
            else                                        // ramp turned off
               auto_control(set_pt, tc[0], 0);

            sprintf(s_str,"%6.2f", power);
            MoveTo(85, 288);  PenColor(0);  DrawString(s_str);

            }

         plot(tc[0]);
         }
/*
         // display event structure for debugging
         sprintf(s_str," %3d, x=%4d, y=%4d, ascii= %2x, scan= %4x, state= %4x",
            evCnt, evnt.CursorX, evnt.CursorY, evnt.ASCII, evnt.ScanCode, evnt.State);
         MoveTo(180, 20); DrawString(s_str);
*/
         //screen refresh
         if (manual_flag && (evnt.ASCII==00 && evnt.ScanCode==0x13))      //Alt-r
            controls_setup();


   } while ( !((evnt.ASCII==00) && (evnt.ScanCode==0x2D)));    // Alt-x = exit

   HideCursor();
   cleanup();
   //vcr_close();
   display_MINI16_config();
   analog_out_ACAO(0, 1);           // set analog_out channels low before exit
   analog_out_ACAO(0, 2);
   display_MINI16_config();

   return 0;
}

// ---------------------------------------------------------------------------
void controls_setup(void)        // all MetaWINDOW structures declared global
{
   int  i;

   PenColor(0);

   SetRect(&infoR, 0, 0, 159, sR.Ymax);                     // info area
   BackColor(7);                                            // black on gray
   EraseRect(&infoR);
   FrameRect(&infoR);

   // diamond schematic ------------------------------------------------------

   MoveTo(50, 450);                                           // top diamond
   LineRel(15, -40); LineRel(40, 0); LineRel(15, 40); LineRel(-70, 0);

   MoveTo(55, 410);                                                 // gasket
   LineRel(0, -5); LineRel(20, 0); LineRel(0, 5); LineRel(-20, 0);
   MoveTo(95, 410);                                                 // gasket
   LineRel(0, -5); LineRel(20, 0); LineRel(0, 5); LineRel(-20, 0);

   MoveTo(125, 420); LineRel(30, 0);                                // heater
   MoveTo(125, 395); LineRel(30, 0);                                // heater

   MoveTo(50, 365);                                          // bottom diamond
   LineRel(15, 40); LineRel(40, 0); LineRel(15, -40); LineRel(-70, 0);

   MoveTo(0, 355); LineRel(159, 0);            // line under diamond schematic

   // heater section - manual control ----------------------------------------

   SetPt(&pt, 10, 315);                                        // manual button
   CenterRect(&pt, BUT_X, BUT_Y, &manual_B);
   BackColor(13); EraseRect(&manual_B);
   FrameRect(&manual_B);

   BackColor(15);                                          // power u/d buttons

      SetPt(&pt, 104, 302);
      CenterRect(&pt, DBUT_X, DBUT_Y, &power_uB[1]);
      EraseRect(&power_uB[1]); FrameRect(&power_uB[1]);

   for (i = 0; i < 2; i++) {
      SetPt(&pt, 95 + 9*i, 282);
      CenterRect(&pt, DBUT_X, DBUT_Y, &power_dB[i]);
      EraseRect(&power_dB[i]); FrameRect(&power_dB[i]);
```

```
}

for (i = 2; i < 4; i++) {
   SetPt(&pt, 102 + 9*i, 302);
   CenterRect(&pt, DBUT_X, DBUT_Y, &power_uB[i]);
   EraseRect(&power_uB[i]); FrameRect(&power_uB[i]);

   SetPt(&pt, 102 + 9*i, 282);
   CenterRect(&pt, DBUT_X, DBUT_Y, &power_dB[i]);
   EraseRect(&power_dB[i]); FrameRect(&power_dB[i]);
}

// heater section - automatic control ----------------------------------

SetPt(&pt, 10, 270);                                          // auto button
CenterRect(&pt, BUT_X, BUT_Y, &auto_B);
BackColor(15);
EraseRect(&auto_B);
FrameRect(&auto_B);

BackColor(15);  PenColor(0);
for (i = 0; i < 4; i++) {                          // temperature u/d buttons
   SetPt(&pt, 63 + 9*i, 255);
   if (i == 3)
      SetPt(&pt, 95, 255);
   CenterRect(&pt, DBUT_X, DBUT_Y, &temp_uB[i]);
   EraseRect(&temp_uB[i]); FrameRect(&temp_uB[i]);

   SetPt(&pt, 63 + 9*i, 235);
   if (i == 3)
      SetPt(&pt, 95, 235);
   CenterRect(&pt, DBUT_X, DBUT_Y, &temp_dB[i]);
   EraseRect(&temp_dB[i]); FrameRect(&temp_dB[i]);
}

for (i = 0; i < 4; i++) {                                  // ramp u/d buttons
   SetPt(&pt, 117+9*i, 255);
   if (i == 3)
      SetPt(&pt, 149, 255);
   CenterRect(&pt, DBUT_X, DBUT_Y, &ramp_uB[i]);
   EraseRect(&ramp_uB[i]); FrameRect(&ramp_uB[i]);

   SetPt(&pt, 117+9*i, 235);
   if (i == 3)
      SetPt(&pt, 149, 235);
   CenterRect(&pt, DBUT_X, DBUT_Y, &ramp_dB[i]);
   EraseRect(&ramp_dB[i]); FrameRect(&ramp_dB[i]);
}

SetPt(&pt, 94, 219);
CenterRect(&pt, BUT_X, BUT_Y, &ok_B);
SetPt(&pt, 150, 219);
CenterRect(&pt, BUT_X, BUT_Y, &reset_B);
BackColor(15);
EraseRect(&ok_B); FrameRect(&ok_B);
EraseRect(&reset_B); FrameRect(&reset_B);

// heater PID controls -------------------------------------------------

SetPt(&pt, 48, 177);                                 // prop_band u/d buttons
CenterRect(&pt, DBUT_X, DBUT_Y, &kb_uB);
BackColor(15); EraseRect(&kb_uB); FrameRect(&kb_uB);
SetPt(&pt, 48, 171);
CenterRect(&pt, DBUT_X, DBUT_Y, &kb_dB);
BackColor(15); EraseRect(&kb_dB); FrameRect(&kb_dB);

SetPt(&pt, 95, 177);                                 // integral u/d buttons
CenterRect(&pt, DBUT_X, DBUT_Y, &ki_uB);
BackColor(15); EraseRect(&ki_uB); FrameRect(&ki_uB);
SetPt(&pt, 95, 171);
CenterRect(&pt, DBUT_X, DBUT_Y, &ki_dB);
BackColor(15); EraseRect(&ki_dB); FrameRect(&ki_dB);

SetPt(&pt, 149, 177);                               // derivative u/d buttons
CenterRect(&pt, DBUT_X, DBUT_Y, &kd_uB);
BackColor(15); EraseRect(&kd_uB); FrameRect(&kd_uB);
SetPt(&pt, 149, 171);
CenterRect(&pt, DBUT_X, DBUT_Y, &kd_dB);
BackColor(15); EraseRect(&kd_dB); FrameRect(&kd_dB);

MoveTo(infoR.Xmin, 160); LineRel(infoR.Xmax, 0);  // line under htr section
```

```
   // initial setup text
   PenColor(0);   BackColor(7);

   MoveTo(55, 335);   DrawString("Heater");
   MoveTo(20, 310);   DrawString("Manual (max=100%)");
   sprintf(s_str,"%6.2f", power);
   MoveTo(85, 288);   DrawString(s_str);

   MoveTo(20, 265);   DrawString("Auto Set_Pt Ramp");
   sprintf(s_str,"%5.1f", set_pt);
   MoveTo(60, 241);   DrawString(s_str);
   sprintf(s_str,"%5.1f", ramp);
   MoveTo(115, 241);  DrawString(s_str);
   MoveTo(70, 215);   DrawString("OK");
   MoveTo(103, 215);  DrawString("Reset");

   MoveTo(20, 190);   DrawString("Prop");
   MoveRel(10, 0);    DrawString("Integ");
   MoveRel(10, 0);    DrawString("Deriv");
   sprintf(s_str,"%3.0f", kb * 100.);
   MoveTo(18, 170);   DrawString(s_str);
   sprintf(s_str,"%3.0f", ki);
   MoveTo(65, 170);   DrawString(s_str);
   sprintf(s_str,"%4.2f", kd);
   MoveTo(110, 170);  DrawString(s_str);

}

//-----------------------------------------------------------------------------
void video_area_setup(void)
{
   SetRect(&videoR, 159, 0, sR.Xmax, sR.Ymax);              // video area
   BackColor(8);
   EraseRect(&videoR);
}

//-----------------------------------------------------------------------------
//  standard MetaWINDOW cleanup() - switches to text and checks QueryError()

void cleanup(void)
{
   int   i;

   StopMouse();
   StopEvent();
   HideCursor();
   SetDisplay(TextPg0);
   ClearText();

   i = QueryError();
   if (i)
      printf("\n QueryError = %d / %d", i >> 7, i & 127);
}

// ----------------------------------------------------------------------------
int heater_power(float power)
{
   int out_val;
   char s_str[20];

   out_val = (int)(4095. * power / 100.);

   analog_out_ACAO(out_val, 1);
   sprintf(s_str,"analog_out=%4d", out_val);
   BackColor(7); PenColor(0);
   MoveTo(20,10); DrawString(s_str);

   return out_val;
}

// ----------------------------------------------------------------------------
void auto_control(float set_point, float current_temp, int reset_flag)
{
   float del_err, sample_int = 1./60., t_PID;
   float b_term, i_term, d_term;
   float temp_to_power = 0.05;
   static float err0, err1, err2, err3, sum_err;

   if (reset_flag) {
      err1 = err2 = err3 = 0.0;
```

```
      sum_err = power / (kb * ki * sample_int * temp_to_power);
   }
   else {
      err3 = err2;
      err2 = err1;
      err1 = err0;
   }
   err0 = set_point - current_temp;
   sum_err += err0;
   del_err = (err0 + 3.*(err1-err2) - err3) / 6.;

   // Although the temperature calculated by the PID line below (t_PID) is
   // proportional to the actual control temperature, there is no explicit
   // relationship.  The proportionality depends on the exact coupling of
   // power to the process temperature.

   b_term = kb * err0;
   i_term = kb * ki * sample_int * sum_err;
   d_term = (kb * kd / sample_int) * del_err;
   t_PID = b_term + i_term + d_term;

   power = t_PID * temp_to_power;
   if (power > POWER_LIMIT)
      power = POWER_LIMIT;
   else if (power < 0.0)
      power = 0.0;

   heater_power(power);
}

// ---------------------------------------------------------------------------
void plot(float tc)
{
   static int col = 160;
   static int row;
   static int init_flag = 1;
   int        i;

   BackColor(8); PenColor(14);

   if(init_flag) {
      for (i = 1; i < 10; i++) {
         MoveTo(160, 48 * i); LineRel(3, 0);
      }
      init_flag = 0;
   }

   if ((col+1) >= 639) {
      EraseRect(&videoR);
      for (i = 1; i < 10; i++) {
         MoveTo(160, 48 * i); LineRel(3, 0);
      }
      col = 160;
      row = (int)(480. * tc / 1000.);
      MoveTo(col,row);
   }
   else {
      MoveTo(col, row);
      row = (int)(480. * tc / 1000.);
      LineTo(++col, row);
   }
   PenColor(0);
}

// ---------------------------------------------------------------------------
void time_temp(void)
{
   char timebuf[9];

   SetFont((fontRec *)fnt2_ptr);
   PenColor(0); BackColor(8);
   TextExtra(2);
   TextFace(cBold);
   _strtime(timebuf);
   sprintf(s_str, "%s %7.1f", timebuf, tc[1]);    // use bottom temp
   MoveTo(300, 50);
   DrawString(s_str);
   SetFont((fontRec *)fnt1_ptr);
   PenColor(0); BackColor(7);
   TextExtra(0); TextFace(cNormal);
}
```

```
// ---------------------------------------------------------------------------
//    2 sets of functions are contained below. 1st is a set of dummy fuctions
//    which are included for programming purposes.  The operating set follows.
//    Comment out the appropriate set of functions.
// ---------------------------------------------------------------------------

/*
// ===========================================================================
// Initialize and calibrate A-D board.  Check and report any errors to user.
void initialize_MINI16(void)
{
}
// ---------------------------------------------------------------------------
void display_MINI16_config(void)
{
}
// ---------------------------------------------------------------------------
void read_MINI16(void)
{
   static float tc_last = 0.0;

   tc[0] = tc_last + (21. * power - tc_last) / 10.;
   tc[1] = tc[0];
   tc[2] = set_pt;

   tc_last = tc[0];
}
// ---------------------------------------------------------------------------
void calibrate_MINI16(void)
{
}
// ---------------------------------------------------------------------------
void analog_out_ACAO(int out_val, int channel)
{
   char s_str[80];

   sprintf(s_str, "analog out %2d - %4d", channel, out_val);
   //PenColor(0);
   //MoveTo(180, 10); DrawString(s_str);
}
// ---------------------------------------------------------------------------
void vcr_init(void)
{
}
// ---------------------------------------------------------------------------
void vcr_close(void)
{
}

*/

// ===========================================================================
// Initialize and calibrate A-D board.  Check and report any errors to user.

void initialize_MINI16(void)
{
   float    analog[16];                          // analog channel array
   unsigned digital[20];                        // digital channel array
   int i;

   // Read CALIB.DAT file and check the hardware setup
   CCALL( "Fn" , analog, digital );         // Call A-D driver

   if (digital[0] == 0 && digital[2] == 0)  {
      printf("\nDriver, ADRIVE.COM, not installed; ");
      printf(" or analog card not installed.\n");
      exit(1);
   }
   if (digital[0] == 0 && digital[2] != 0)  {
      printf("\nNo analog card selected. BRD SEL switch set to 0.\n");
      exit(1);
   }
   if (digital[0] != 0 && digital[6] == 0)  {
      printf("\nCALIB.DAT file not correct or FIND.EXE was not run.\n");
      exit(1);
   }
   if (digital[0] > digital[6])  {
      printf("\nCalibration numbers are not correct.\n");
      exit(1);
   }
   if (digital[0]!=IN_ANACNT || digital[1]!=OUT_ANACNT || digital[2]!=DIGCNT) {
```

```
      printf("\nMix-up on number of channels installed.\n");
      printf("Change size of IN_ANACNT, OUT_ANACNT, and/or DIGCNT\n");
      for (i = 0; i < 8; i++)
         printf("digital[%d] = %u\n", i, digital[i]);
      exit(1);
   }

   // Set analog-out channels low

   analog_out_ACAO(0, 1);
   analog_out_ACAO(0, 2);

   // Specify # of analog_in channels in use
   digital[0] = IN_CHAN;
   CCALL( "N" , analog, digital );

   // Set analog channel resolution to 16 bits - low noise
   digital[0] = 18;
   CCALL( "a" , analog, digital );

   // Set delay for all analog input channels
   for (i = 0; i < IN_CHAN; i++)
      digital[i] = 5000;
   CCALL( "D" , analog, digital );

   // Select analog channel range and calibrate
   for (i = 0; i < IN_CHAN; ++i)
      digital[i] = 22;                          // 22 = type K - 50 mV
      digital[2] =  2;     // block position 3  0-10 vdc  bottom tc
      digital[3] =  2;     // block position 4  0-10 vdc  top tc
   CCALL( "rc" , analog, digital );

   // Select up digital channel direction
   for (i = 0; i < DIGCNT; ++i)
      digital[i] = 0;
   CCALL( "S" , analog, digital );
}

// ---------------------------------------------------------------------------
void display_MINI16_config(void)
{
   float    analog[16];                              // analog channel array
   unsigned digital[20];                            // digital channel array

   CCALL("R", analog, digital);
   CCALL("n", analog, digital);
   printf("\n\n# of analog outs %u # of analog outs w/ cal #s %u",
           digital[1], digital[4]);
//   CCALL("b", analog, digital);
//   printf("\n\nchannel 1 = %u\nchannel 2 = %u", digital[0], digital[1] );
}

// ---------------------------------------------------------------------------
void read_MINI16(void)
{
   int i;
   float    analog[16];                              // analog channel array
   unsigned digital[20];                            // digital channel array


   // Calibrate and read analog inputs
   CCALL( "m" , analog, digital );          // Call A-D driver
   //for (i = 0; i < IN_CHAN && i < IN_ANACNT; i++)
   //   tc[i] = analog[i];
   tc[0] = 164.2 * analog[3] - 270. + 0.27;
   tc[1] = 164.2 * analog[2] - 270. + 0.38;
   tc[2] = analog[0];
}

// ---------------------------------------------------------------------------
void analog_out_ACAO(int out_val, int channel)
{
   // out_val must be in the range 0 - 4095; channel = 1 or 2
   CCALL("w", &out_val, &channel);
}

// ---------------------------------------------------------------------------
void calibrate_MINI16(void)
{
   float    analog[16];                              // analog channel array
   unsigned digital[20];                            // digital channel array
```

```
   CCALL("c", analog, digital);
}

// ---------------------------------------------------------------------------
```

make file for run_dac.exe

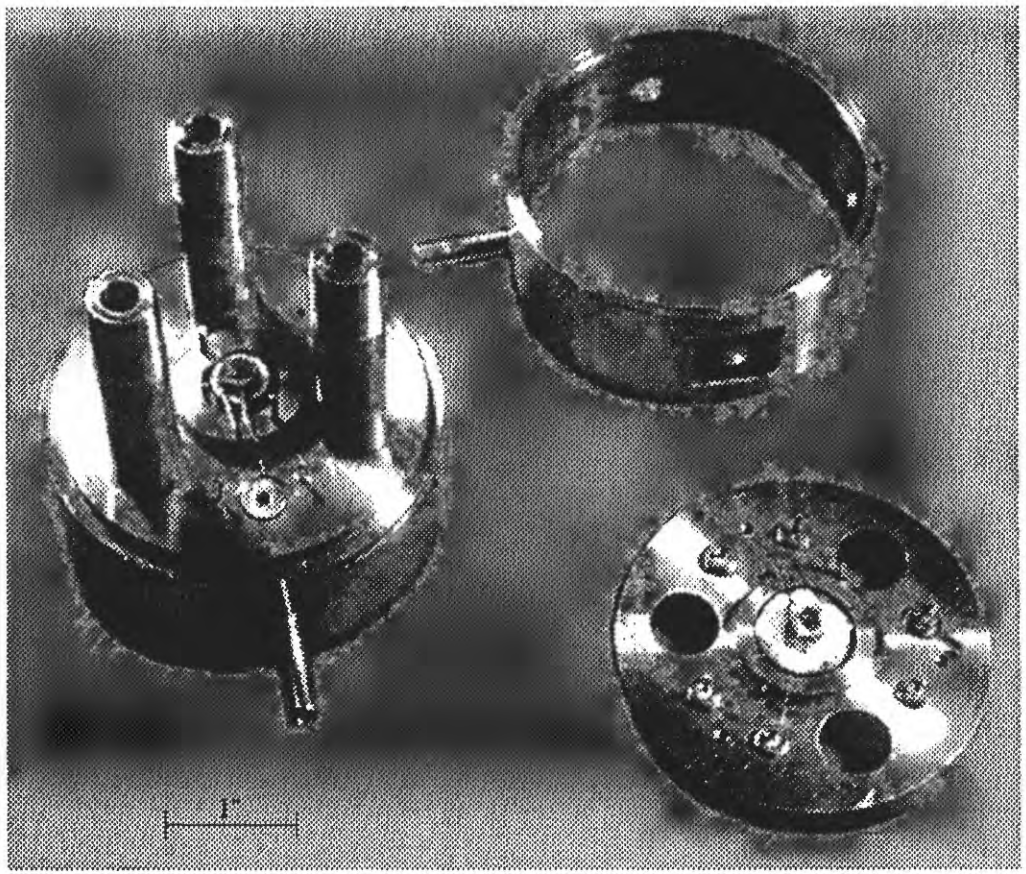
```
COMPILE = bcc  -ml -v -O- -f87
LIBPATH = D:\BC\LIB
INCLUDEPATH = D:\BC\INCLUDE

.c.obj:
  $(COMPILE) -c {$< }

run_dac.exe: run_dac.obj
   tlink /c /x /v c0l run_dac ms_call async2, run_dac, , wndowblm FP87 mathl cl

run_dac.obj: run_dac.c
```

(A)



(B)

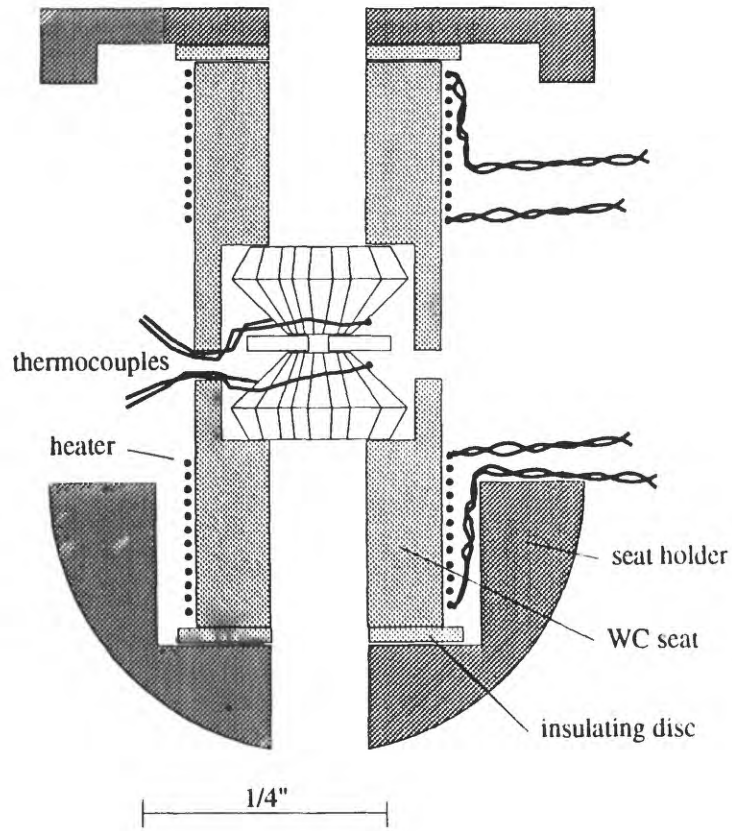


Fig 1. (A) Partially dissembled hydrothermal diamond anvil cell (Basset et al., 1993). The bottom platen is at the left, top platen at lower right, and outer sleeve at upper right. The three pairs of electrical lead-throughs are visible on the upper platen. During a high-temperature experiment, air is forced through the lower platen and posts to reduce thermal expansion, and 1% hydrogen in argon is fed through the outer sleeve to inhibit oxidation of the diamonds and heaters. (B) A schematic view of the heater and thermocouple arrangement. The tungsten carbide seat, insulating disc, and seat holder are held together with alumina cement. The top assembly can be translated horizontally and the bottom assembly tilted by alignment screws. Scales are approximate.

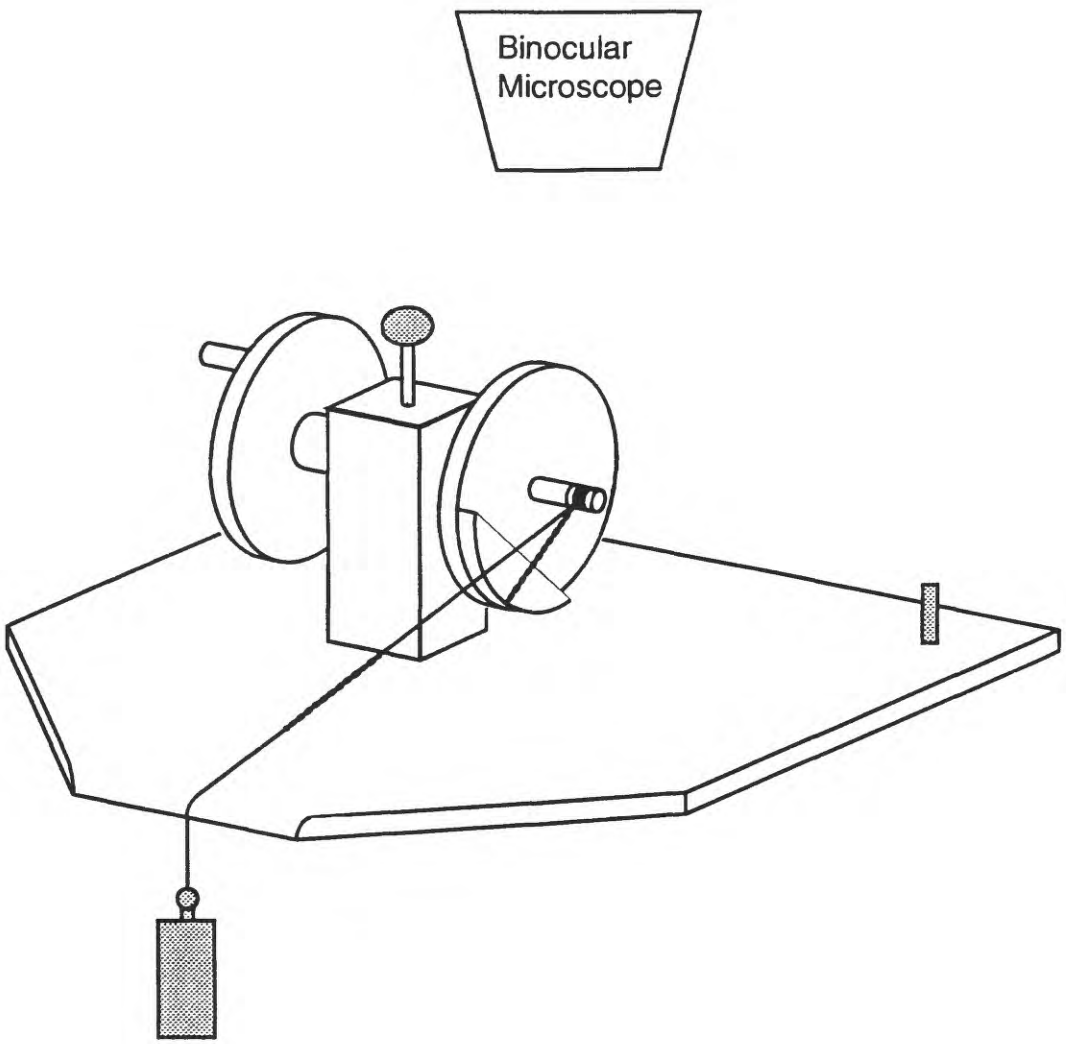


Fig. 2 Schematic drawing of a jig used for winding a wire heater on the tungsten carbide seat. The jig is used with a binocular microscope and is secured in place by pins which match holes in the microscope stage.

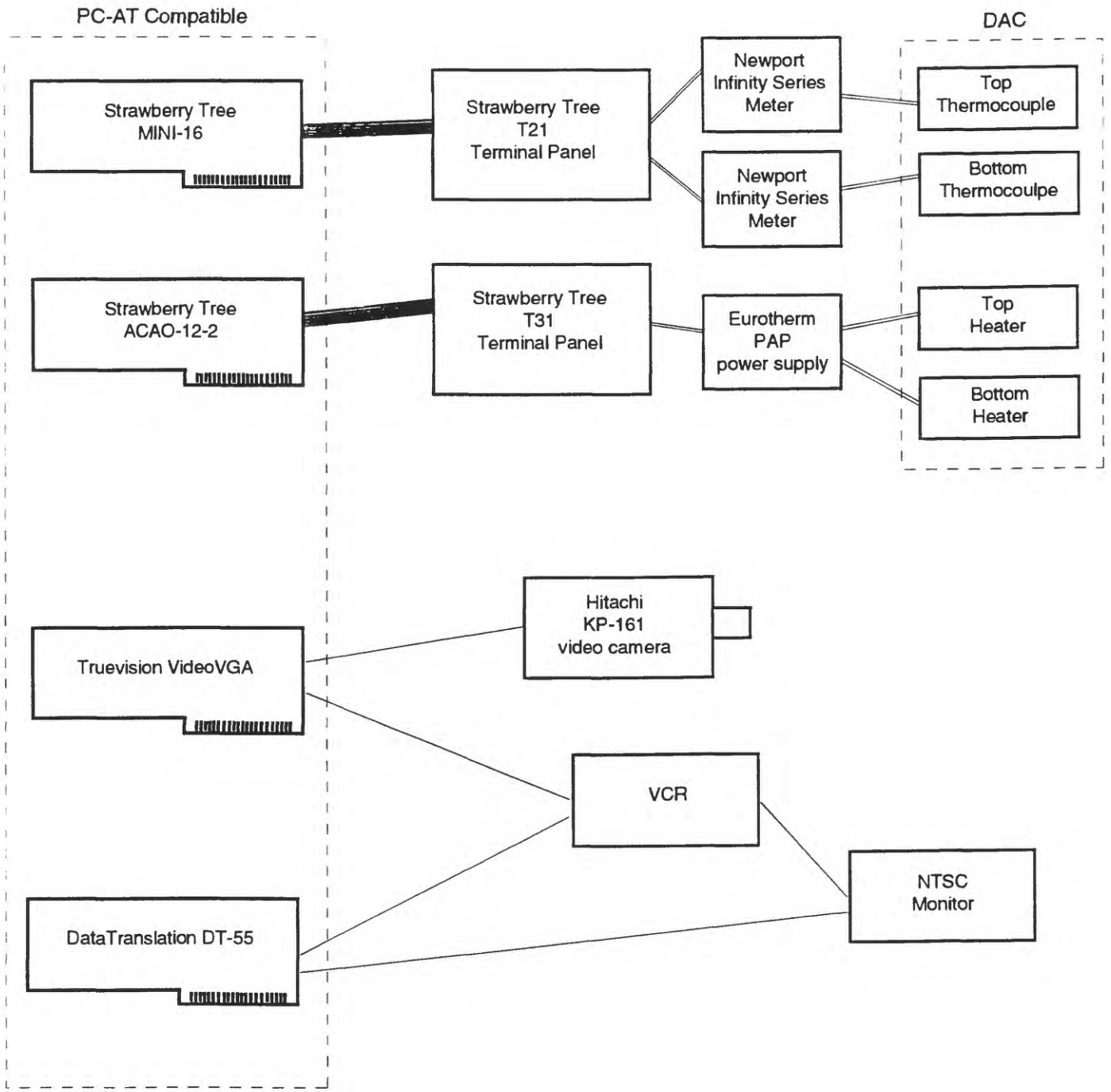


Fig 3. Schematic diagram of the temperature measurement, heater control, and video electronic components. Dashed outlines indicate the physical limits of the PC-AT compatible computer and the diamond anvil cell.

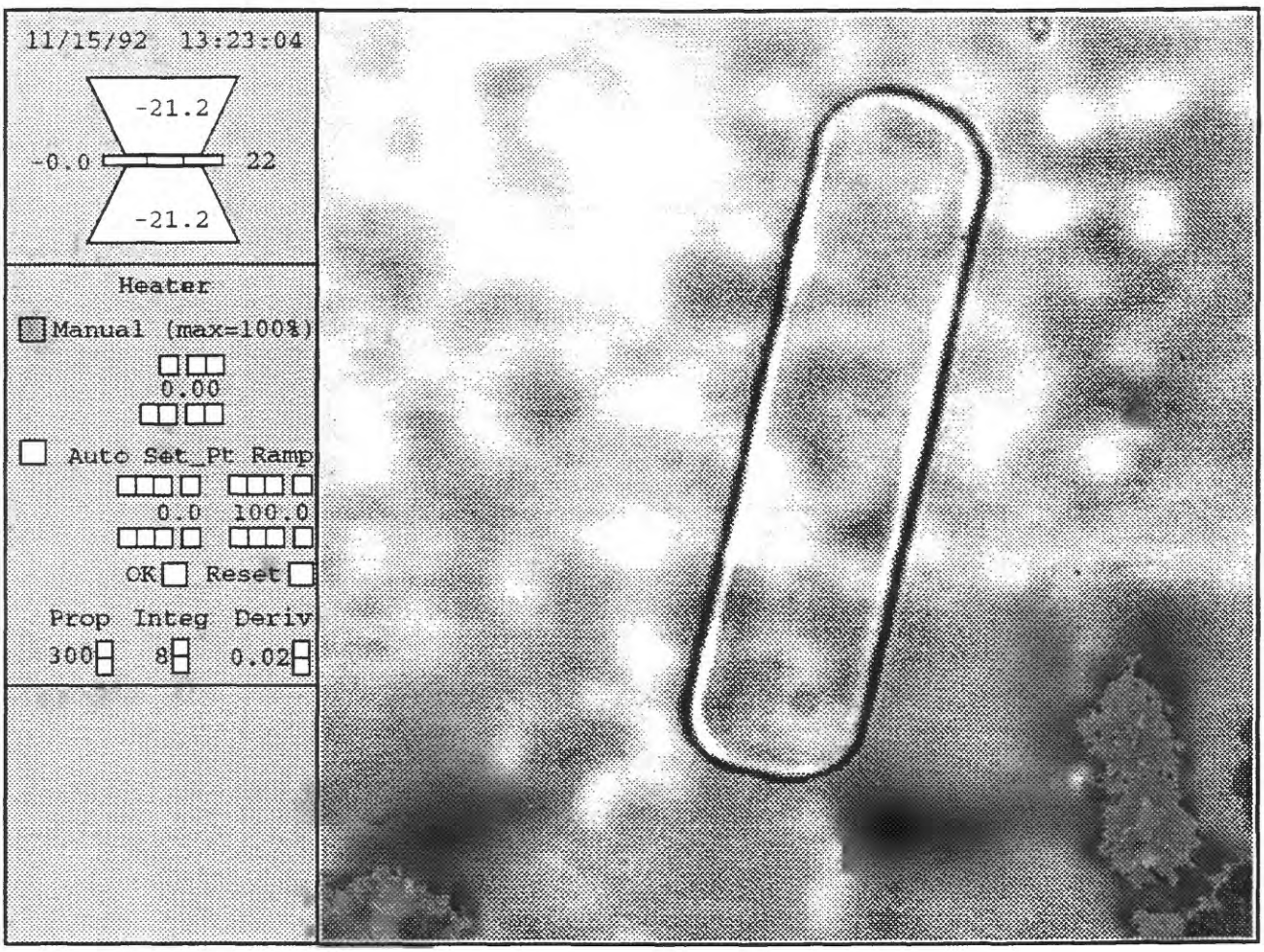


Fig. 4. Schematic showing the combined NTSC video signal. The control strip on the left side is converted to an analog signal and is combined with the image from the microscope video camera.