A Programmable Heater Control Circuit for Spacecraft
(Center Director's Discretionary Fund Final Report Project No. 90-19)

D.D. Nguyen, J.W. Owen, D.A. Smith, W.J. Lewter

Structures and Dynamics Laboratory
and Astrionics Laboratory
Science Engineering Directorate

June 1994
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>APPROACH</td>
<td>3</td>
</tr>
<tr>
<td>SPACECRAFT APPLICATIONS</td>
<td>4</td>
</tr>
<tr>
<td>BREADBOARD DEVELOPMENT</td>
<td>5</td>
</tr>
<tr>
<td>SOFTWARE DEVELOPMENT AND VERIFICATION</td>
<td>7</td>
</tr>
<tr>
<td>PROTOTYPE DEVELOPMENT AND TESTING</td>
<td>8</td>
</tr>
<tr>
<td>QUALIFICATION REQUIREMENTS</td>
<td>9</td>
</tr>
<tr>
<td>CONCLUSION</td>
<td>10</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>11</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Typical thermostat</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Mechanical thermostat circuit</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>Analog control circuit</td>
<td>2</td>
</tr>
<tr>
<td>4.</td>
<td>Programmable heater control circuit with internal dc/dc converter</td>
<td>4</td>
</tr>
<tr>
<td>5.</td>
<td>Programmable heater control circuit with battery power logic circuit</td>
<td>5</td>
</tr>
<tr>
<td>6.</td>
<td>Programmable heater controller hybrid schematic</td>
<td>6</td>
</tr>
<tr>
<td>7.</td>
<td>The dimension of prototype hybrid</td>
<td>8</td>
</tr>
<tr>
<td>8.</td>
<td>Prototype hybrid system test configuration</td>
<td>9</td>
</tr>
</tbody>
</table>
## TERMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootloader</td>
<td>a program used for loading user programs into EEPROM</td>
</tr>
<tr>
<td>Breadboard</td>
<td>a printed circuit board that can be mounted and wired whatever circuitry is designed</td>
</tr>
<tr>
<td>CPU</td>
<td>central processor unit</td>
</tr>
<tr>
<td>D/A</td>
<td>digital-to-analog</td>
</tr>
<tr>
<td>EEPROM</td>
<td>electrically erasable programmable read-only memory</td>
</tr>
<tr>
<td>EMI</td>
<td>electromagnetic interference</td>
</tr>
<tr>
<td>mil-specification</td>
<td>Military Standard Mil-M-38510J, Mil-STD-883C...</td>
</tr>
<tr>
<td>MODB</td>
<td>operation mode pin B on 68 HC11 microcontroller</td>
</tr>
<tr>
<td>MOSFET</td>
<td>metal oxide semiconductor field-effect transistor</td>
</tr>
<tr>
<td>PHCC</td>
<td>programmable heater control circuit</td>
</tr>
<tr>
<td>RAM</td>
<td>random access memory</td>
</tr>
</tbody>
</table>
TECHNICAL MEMORANDUM

A PROGRAMMABLE HEATER CONTROL CIRCUIT FOR SPACECRAFT

Center Director’s Discretionary Fund Final Report
Project No. 90–19

INTRODUCTION

Spacecraft thermal control is accomplished for many components through use of multilayer insulation systems, electrical heaters, and radiator systems. The heaters are commanded to maintain component temperatures within design specifications. Control of the heater system has been accomplished in several ways including local thermostats, analog devices, and use of centralized digital control systems.

Many component heater control systems use thermostats to provide the desired temperature control. Thermostats are mechanical devices which make or break electrical contact due to change in temperature. This is usually accomplished via differential thermal expansion of a bimetallic disk that changes shape, and therefore position, with temperature. A typical thermostat is shown schematically in figure 1. Since a thermostat can conceivably fail in the closed position, two thermostats are used in spacecraft operation and are wired in series so that the circuit breaks if either of the thermostats open. Further, to provide the complete redundancy needed for spaceflight, two parallel circuits, each with two thermostats, are required as a safeguard against one of the thermostats failing in the open position. Figure 2 shows typical circuitry using thermostats.

Figure 1. Typical thermostat (courtesy of Elmwood Sensors, Inc.).
Alternatives to mechanical thermostats have included analog circuits in varying design applications, using solid state devices. Analog devices have the advantage of precise temperature control, for components with operational temperature requirements which must be maintained within small limits, such as spacecraft pointing and control systems. A schematic of an analog controller is shown in figure 3. The analog circuitry must also be designed with redundancy in spacecraft applications.

Some spacecraft have used central digital control systems to accomplish temperature control of components. This involves design of a central digital computer which receives many analog inputs from distributed component controllers. The analog signals are converted to digital data and the control functions are determined by the central processor unit (CPU). The heaters are commanded individually from the CPU based upon control logic. For some applications, proportional control is provided through voltage regulation.

Several disadvantages with these systems can be identified. For thermostatically controlled systems the mechanical thermostat itself has a potential for failure, either opened or closed. The thermostat, being mechanical, provides no temperature data; therefore, separate temperature sensors must be provided by a central spacecraft data system. The thermostats have a predetermined set point (temperature) and dead band (control range) that cannot be altered.
Analog devices eliminate the mechanical problems associated with thermostats, but they also have a predetermined set point and dead band that cannot be altered once physically integrated into the spacecraft. The analog device has no feedback of temperature data for monitoring the thermal condition of the component, even though one or more temperature sensors are integrated into the circuit. If the component temperature must be monitored, another sensor must be installed and integrated with the spacecraft central data system.

The central digital control systems provide the most versatility and flexibility with respect to control authority, data acquisition, reprogramming, and sometimes proportional control. These systems are usually expensive, require substantial integration during spacecraft assembly, and result in significant spacecraft wiring for the control instrumentation.

With the advanced technology in both electronic sensing and computer programming, this development introduces solid state design, use of control instrumentation as data available to the central data system, reprogramming capability of the local microprocessor during the spacecraft mission, if required, and the elimination of significant spacecraft wiring. The hybrid integrated circuit has a temperature sensing and conditioning circuit, a microprocessor, and a heater power and control circuit. This programmable heater control circuit (PHCC) is miniature and housed in a volume which allows physical integration with the component to be controlled. Applications might include alternate battery-powered logic-circuit configurations. A prototype unit with appropriate physical and functional interfaces was procured for testing. The physical functionality and feasibility of the hybrid integrated circuit were successfully verified.

**APPROACH**

The effort was to develop a small, integrated hybrid circuit that captured the advantages of both central digital controllers and local autonomous control systems. A simplified schematic of the device is shown in figure 4. Optional configurations are shown in figure 5. The advantages of this design include solid state design, use of control instrumentation as data available to the central data system (through a time-multiplexed data bus), reprogramming capability of the local microprocessor during the spacecraft mission, if required, and the elimination of significant spacecraft wiring (the device only requires a power bus interface and a data bus interface).

The hybrid integrated circuit has a temperature sensing and conditioning circuit, a microprocessor, and a heater power and control circuit. The device is miniature and housed in a volume which allows physical integration with the component to be controlled. In typical operation, the set point temperature and dead band control range are programmed in the heater control logic of the microprocessor. Temperature sensors reference the component temperature through signal conditioners and an analog-to-digital converter. Based on the reference temperature, the microprocessor issues commands to the heater controllers to turn on or off (voltage regulation, or proportional control is not provided in this design, with the exception of varying the heater duty cycle over time). The integrated circuit, through the microprocessor, also interfaces with the central data bus. Upon interrogation by the spacecraft CPU via this bidirectional communication link, component temperature data is passed upon demand to the spacecraft central processor. Additionally, the status of the heaters (on or off), the duty cycle of the heaters, and computed functions such as power utilization can be communicated and eventually telemetered to ground stations. The central spacecraft computer system also has the capability of reprogramming or reconfiguring the local microprocessor heater control logic to adjust to unexpected or anomalous events.
Additionally, the local microprocessor can be programmed to react to events such as temperature sensor failures, heater circuit failures, etc., and take action to use redundant systems. The versatility and utility of the device is quite good and offers substantial design and operational advantages to spacecraft thermal designers.

**SPACECRAFT APPLICATIONS**

Potential applications of the PHCC are believed to exist both in the spacecraft and payload areas. The primary application is believed to be free-flying spacecraft with somewhat tight component temperature-control requirements. These include applications such as nickel-hydrogen battery temperature control, pointing and control system temperature control, low temperature conditioning of science instruments, telescope mirror assembly temperature control, and heater control for variable conductance heat pipe systems. The payload applications might include alternate design configurations, such as battery-powered logic circuits, for short duration missions, eliminating the need for internal dc/dc conversion within the PHCC (see fig. 5). Control requirements would be similar to the spacecraft applications discussed above. Payload applications may not require an interface with a central data system but might require expanded memory in the microprocessor, with data retrieval occurring after flight. These design options can be made available through several off-the-shelf controllers, with varying design features selected by the spacecraft or payload designer. Depending upon the spacecraft market and unit cost for these devices, these solid state programmable systems could replace mechanical thermostats, analog devices, and central controllers on future spacecraft programs.
A fully functional breadboard of the PHCC was developed. The attached schematic, figure 6, shows the circuitry developed to accomplish this task. The Motorola M68HC811E2 microcontroller was chosen to be the heart of the design. This controller is in Motorola's HC11 family of microcontrollers. It features 2 kilobytes of electrically erasable programmable read-only memory (EEPROM), an 8-bit analog-to-digital converter with eight multiplexed inputs, an RS232 port, internal timers, an 8-bit bidirectional port, and an 8-bit output port. The Maxim MAX700 was chosen to provide the reset signal upon power-up. The microcontroller can read eight analog sensor input signals. These are input through an 8-to-1 analog multiplexer. The multiplexer used is the Analog Device 7503. The input are selected by three control lines from the microcontroller. Two of these inputs are AD590 temperature transducers that are mounted internal to the case. The other six inputs are set up to be AD590 inputs but can also be used to input other signals. Each external input has a pair of external pins with a 15-V reference signal on one pin and the other going to an input of an 8-to-1 multiplexer and to a 10-k ohm resistor to ground. The output of the multiplexer then goes to the plus input of the instrumentation amplifier. The minus input of the instrumentation amplifier is connected to the output of a 12-bit digital-to-analog (D/A) converter. The D/A used is the MicroNetworks MN371. The 12 input bits to the D/A come from the microcontroller digital output ports. The instrumentation amplifier used is the Burr Brown PGA200 programmable gain instrumentation amplifier. It has four selectable gain of 1, 10, 100, 1,000. The purpose of the D/A converter and the selectable gain is to provide a dc offset and amplification of the signal for better resolution and control around a setpoint.
Figure 6. Programmable heater controller hybrid schematic.
Communication with a main computer is accomplished through the microcontroller's RS232 port. The digital transmit and receive lines from the microcontroller are level-converted and buffered within the hybrid. The transmit line is buffered by a 1488 line driver and the receive line is buffered by a 1489 line receiver.

Initial programming of the 2k of EEPROM is done through the RS232 port. A “program mode” is entered when the MODB line that comes to an external pin is tied to ground. A “Bootloader” program is loaded into the random access memory (RAM) and takes control of the microcontroller to load the EEPROM. Software or “control code” is developed using the HC11 assembly language and assembler. It should be noted that the EEPROM can be reprogrammed under computer control through the RS232 port connected to the computer used for monitoring and control. Changing set points or control algorithms can be accomplished easily through the monitor and control computer also.

Four heaters can be controlled from this hybrid. There are two external lines provided for each heater. The load voltage brought in on the “28 V” external pin is connected to one of these lines. The other line is connected to the drain of an International Rectifier IRFF130. This metal oxide semiconductor field-effect transistor (MOSFET) is rated at 100 V and has an “on” resistance of less than 0.2 ohms. The MOSFET’s are controlled by outputs from the microcontroller that are optically isolated from the MOSFET by an HP2200 optical insulator.

SOFTWARE DEVELOPMENT AND VERIFICATION

The PHCC software is shown in appendix A. The software was developed using the breadboard unit discussed above. The software is designed to set, monitor, and program the heater controller and is as simple and flexible as possible. Some of the function features in the software include the following:

- Header files
- Turning cursor on and off
- Setting screen colors
- Screen locations
- User inputs
- Module functions
- Global variables
- Communicating with heater controller
- Controlling cursor
- Initializing stack, output ports, heaters
- Initial conditions
- Heater control
- Reading temperature.

Verification was also accomplished on the prototype unit when software was loaded and each function was successfully executed.
To verify the physical functionality and feasibility of fabrication of the hybrid integrated circuit, a prototype unit was procured and built from non Mil-certified parts. The prototype unit did not include the internal dc/dc converter required for the flight version. This function was simulated external to the prototype. The unit was delivered and burn-in tests were accomplished, and the software was loaded and verified. Of four units delivered and tested, one unit failed the burn-in test. The dimensions of the prototype are shown in figure 7. The width is 2 inch and length is 2.85 inch. The external pin connections are on 0.15-inch spacing and extend 0.25 inch from the body on each side. The mounting flanges extend 0.35 inch from each end and give an overall length of 3.85 inch. The base of the package is 0.0625 inch thick, and the diameter of the mounting holes is 0.16 inch with the centers 3.175 inch apart. The overall height is 0.5675 inch.

The prototype Hybrid weighs approximately 160 g (5.6 oz). The quiescent power consumed with all heaters "off" and including the dc/dc converter is 1.5 W. The complete system test configuration is shown in figure 8. The test procedures and results are described and tabulated as shown in appendix B.
QUALIFICATION REQUIREMENTS

Based upon the above discussion, the PHCC has been shown to be a feasible design concept. The software is fully developed and verified. Further development is required in the area of procurement and testing of a Mil-certified part with an internal dc/dc converter for generic qualification testing. The testing should include thermal vacuum performance testing, thermal cycling, vibration, and electromagnetic interference (EMI) testing. Appropriate test plans should be generated to qualify the device for a wide range of spacecraft applications as an off-the-shelf item. Military Standard 1540B provides qualification test requirements routinely accomplished at MSFC. Should a joint venture be initiated with a potential PHCC vendor, MSFC qualification testing is an option that is available.
CONCLUSION

The PHCC design has progressed to the definition of a very efficient and compact device including an internal dc/dc converter. The software is fully developed and verified. A breadboard unit has been developed and tested with applicable software. Also a prototype unit with appropriate physical and functional interfaces was procured for software testing and for burn-in testing. The remaining work to develop a flight-qualified device includes fabrication and testing of a Mil-certified part, with an internal dc/dc converter.

An option for completing the PHCC flight qualification testing is to enter into a joint venture with industry. The government contribution would include the design, software, and development history; also, the government could provide the qualification testing and data analysis. The industry participation would include fabrication of the units for testing, per Mil-specification. Successful completion of the test program would result in a commercially marketable device or family of devices, with applications to spacecraft thermal control systems.
REFERENCES


/*
 * include statements for header files
 */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <bios.h>
#include "rs232.h"

/*
 * definitions used in turning cursor on and off
 */
#define VIDEO_IO 0x10
#define SET_CRSR 1

/*
 * definitions used for setting screen colors
 */
#define BLACK 0x00
#define BLUE 0x10
#define GREEN 0x20
#define CYAN 0x30
#define RED 0x40
#define MAGENTA 0x50
#define BROWN 0x60
#define WHITE 0x70
#define BLACK_ON 0x00
```c
#define ELUE ON 0x01
#define GREEN ON 0x02
#define CYAN ON 0x03
#define RED ON 0x04
#define MAGENTA ON 0x05
#define BROWN ON 0x06
#define WHITE ON 0x07
#define GREY ON 0x08
#define BRIGHT BLUE ON 0x09
#define BRIGHT GREEN ON 0x0A
#define BRIGHT CYAN ON 0x0B
#define BRIGHT RED ON 0x0C
#define BRIGHT MAGENTA ON 0x0D
#define YELLOW ON 0x0E
#define BRIGHT WHITE ON 0x0F

/*
 * definitions used for screen locations
 */

#define PLATE_UL_ROW 1
#define PLATE_UL_COLUMN 12
#define PLATE_ROWS 15
#define PLATE_COLUMNS 55

#define HEATER1_UL_ROW 2
#define HEATER1_UL_COLUMN 14
#define HEATER1_ROWS 5
#define HEATER1_COLUMNS 19

#define HEATER2_UL_ROW 7
#define HEATER2_UL_COLUMN 54
#define HEATER2_ROWS 3
#define HEATER2_COLUMNS 11

#define SENSOR1_UL_ROW 4
#define SENSOR1_UL_COLUMN 18
#define SENSOR1_ROWS 1
#define SENSOR1_COLUMNS 6

#define SENSOR2_UL_ROW 4
#define SENSOR2_UL_COLUMN 56
#define SENSOR2_ROWS 1
#define SENSOR2_COLUMNS 6

#define SENSOR3_UL_ROW 8
#define SENSOR3_UL_COLUMN 26
#define SENSOR3_ROWS 1
#define SENSOR3_COLUMNS 6

#define SENSOR4_UL_ROW 12
#define SENSOR4_UL_COLUMN 18
#define SENSOR4_ROWS 1
#define SENSOR4_COLUMNS 6
```
#define SENSOR5_UL_ROW 12
#define SENSOR5_UL_COLUMN 55
#define SENSOR5_ROWS 1
#define SENSOR5_COLUMNS 6

#define ROW_query_window 18
#define COLUMN_query_window 40

#define ROW_HEATER1_SETPOINT 3
#define COLUMN_HEATER1_SETPOINT 1

#define ROW_HEATER2_SETPOINT 9
#define COLUMN_HEATER2_SETPOINT 70

#define ROW_NUMBER_TO_AVERAGE 23
#define COLUMN_NUMBER_TO_AVERAGE 30

#define ROW_ERROR_MESSAGE 16
#define COLUMN_ERROR_MESSAGE 20

/**********************************************************
* definitions used for getting information from user
**********************************************************/

#define RETURN 13
#define ESCAPE 27
#define BACKSPACE 8
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define CURSOR 176
#define F1 59
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F7 65
#define F8 66
#define F9 67

/**********************************************************
* function declarations for functions in this module
**********************************************************/

void send_byte_to_heater( unsigned char c );
unsigned char receive_byte_from_heater( unsigned comm_port );
void wait_for_comm_port_data( void );

void turn_on_cursor( void );
void turn_off_cursor( void );
void change_cursor( int top, int bottom );

void background_on_screen( void );
void update_screen( void );
void display_screen( void );

void write_string_in_screen_buffer( char *string, int row, int column );
void write_double_in_screen_buffer( char *format_string,
     double number, int row, int column );

void cut_string( char *destination, char *source, int first_char, int last_char );

void user_edit_field_on_screen( char *string,
     int row, int start_column, int end_column,
     char field_foreground, char field_background,
     char cursor_foreground, char cursor_background );

void copy_characters_from_screen( char *string,
     int start_row, int end_row, int start_column, int end_column );
void copy_characters_to_screen( char *string,
     int start_row, int end_row, int start_column, int end_column );
void copy_attributes_from_screen( char *string,
     int start_row, int end_row, int start_column, int end_column );
void copy_attributes_to_screen( char *string,
     int start_row, int end_row, int start_column, int end_column );

void fill_character_block( char c,
     int start_row, int end_row, int start_column, int end_column );
void fill_attribute_block( char c,
     int start_row, int start_column, int end_column );

char query_user_for_key( char *prompt );
double query_user_for_number( char *prompt );

void open_window( char *character_array, char *attribute_array,
     int start_row, int end_row, int start_column, int end_column,
     char color_byte );
void close_window( char *character_array, char *attribute_array,
     int start_row, int end_row, int start_column, int end_column );

char screen_characters[2001], screen_attributes[2001];
unsigned comm_port;
int debug_mode = 0;

/*
   global variable declarations
*/

the main program
int main( int argc, char *argv[] )
{
    static char string[256];
    char key = ' ';
    unsigned char heater_status;
    unsigned char sensor1_data;
    unsigned char sensor2_data;
    unsigned char sensor3_data;
    unsigned char sensor4_data;
    unsigned char sensor5_data;

    double sensor1_temperature = 0.0;
    double sensor2_temperature = 0.0;
    double sensor3_temperature = 0.0;
    double sensor4_temperature = 0.0;
    double sensor5_temperature = 0.0;
    double averages = 0.0;
    double number_of_readings_to_average = 10.0;

    unsigned char heater_select;
    unsigned char sensor_select;
    unsigned char set_point;

    FILE *fileptr;

    /*
    Check for debug mode.
    */

    if ( argc > 1 )
    {
        if ( (argv[1])[0] == 'd' || (argv[1])[0] == 'D' )
            debug_mode = 1;
    }

    /*
    Set up the screen.
    */

    turn_off_cursor();
    background_on_screen();

    /*
    Set up the comm port.
    */

    do
    {
        key = query_user_for_key( "Enter 1 for COM1 or 2 for COM2" );
    } while( key != '1' && key != '2' && key != ESCAPE );

    if ( key == ESCAPE )
if ( key == '2' ) comm_port = COM2;
else comm_port = COM1;

initialize_comm_port( comm_port,
    _COM_CH8 | _COM_STOP1 | _COM_NOPARITY | _COM_9600 );

if ( check_status( comm_port ) & DATA_READY_BIT )
    receive_byte( comm_port );

/*
Main program loop.
*/
do
{
    do
    {
        send_byte_to_heater( 'D' );

        heater_status = receive_byte_from_heater( comm_port );
        sensor1_data = receive_byte_from_heater( comm_port );
        sensor2_data = receive_byte_from_heater( comm_port );
        sensor3_data = receive_byte_from_heater( comm_port );
        sensor4_data = receive_byte_from_heater( comm_port );
        sensor5_data = receive_byte_from_heater( comm_port );

        sensor1_temperature += (double) sensor1_data * 50.0 / 255.0;
        sensor2_temperature += (double) sensor2_data * 50.0 / 255.0;
        sensor3_temperature += (double) sensor3_data * 50.0 / 255.0;
        sensor4_temperature += (double) sensor4_data * 50.0 / 255.0;
        sensor5_temperature += (double) sensor5_data * 50.0 / 255.0;
        ++averages;

        if ( averages >= number_of_readings_to_average )
            {
                write_double_in_screen_buffer( "%6.1lf",
                    ( sensor1_temperature / averages ),
                    SENSOR1_UL_ROW, SENSOR1_UL_COLUMN );

                write_double_in_screen_buffer( "%6.1lf",
                    ( sensor2_temperature / averages ),
                    SENSOR2_UL_ROW, SENSOR2_UL_COLUMN );

                write_double_in_screen_buffer( "%6.1lf",
                    ( sensor3_temperature / averages ),
                    SENSOR3_UL_ROW, SENSOR3_UL_COLUMN );

                write_double_in_screen_buffer( "%6.1lf",
                    ( sensor4_temperature / averages ),
                    SENSOR4_UL_ROW, SENSOR4_UL_COLUMN );

                write_double_in_screen_buffer( "%6.1lf",
                    ( sensor5_temperature / averages ),
                    SENSOR5_UL_ROW, SENSOR5_UL_COLUMN );

            }
( sensor5_temperature / averages ),
SENSOR5_UL_ROW, SENSOR5_UL_COLUMN );

sensor1_temperature = 0.0;
sensor2_temperature = 0.0;
sensor3_temperature = 0.0;
sensor4_temperature = 0.0;
sensor5_temperature = 0.0;
averages = 0.0;
}

if ( heater_status & 1 )
{
  fill_attribute_block( WHITE_ON | BROWN,
    HEATER1_UL_ROW, HEATER1_UL_ROW + HEATER1_ROWS - 1,
    HEATER1_UL_COLUMN, HEATER1_UL_COLUMN + HEATER1_COLUMNS - 1
  fill_attribute_block( BRIGHT WHITE_ON | BROWN,
    SENSOR1_UL_ROW + 1, SENSOR1_UL_ROW + 1 + SENSOR1_ROWS - 1,
    SENSOR1_UL_COLUMN, SENSOR1_UL_COLUMN + SENSOR1_COLUMNS - 1
}
else
{
  fill_attribute_block( WHITE_ON | RED,
    HEATER1_UL_ROW, HEATER1_UL_ROW + HEATER1_ROWS - 1,
    HEATER1_UL_COLUMN, HEATER1_UL_COLUMN + HEATER1_COLUMNS - 1
  fill_attribute_block( BRIGHT WHITE_ON | RED,
    SENSOR1_UL_ROW + 1, SENSOR1_UL_ROW + 1 + SENSOR1_ROWS - 1,
    SENSOR1_UL_COLUMN, SENSOR1_UL_COLUMN + SENSOR1_COLUMNS - 1
}

if ( heater_status & 2 )
{
  fill_attribute_block( WHITE_ON | BROWN,
    HEATER2_UL_ROW, HEATER2_UL_ROW + HEATER2_ROWS - 1,
    HEATER2_UL_COLUMN, HEATER2_UL_COLUMN + HEATER2_COLUMNS - 1
  fill_attribute_block( BRIGHT WHITE_ON | BROWN,
    SENSOR5_UL_ROW + 1, SENSOR5_UL_ROW + 1 + SENSOR5_ROWS - 1,
    SENSOR5_UL_COLUMN, SENSOR5_UL_COLUMN + SENSOR5_COLUMNS - 1
}
else
{
  fill_attribute_block( WHITE_ON | RED,
    HEATER2_UL_ROW, HEATER2_UL_ROW + HEATER2_ROWS - 1,
    HEATER2_UL_COLUMN, HEATER2_UL_COLUMN + HEATER2_COLUMNS - 1
  fill_attribute_block( BRIGHT WHITE_ON | RED,
    SENSOR5_UL_ROW + 1, SENSOR5_UL_ROW + 1 + SENSOR5_ROWS - 1,
    SENSOR5_UL_COLUMN, SENSOR5_UL_COLUMN + SENSOR5_COLUMNS - 1
}

fill_attribute_block( BLACK_ON | WHITE,
  SENSOR1_UL_ROW, SENSOR1_UL_ROW + SENSOR1_ROWS - 1,
  SENSOR1_UL_COLUMN, SENSOR1_UL_COLUMN + SENSOR1_COLUMNS - 1 );

fill_attribute_block( BLACK_ON | WHITE,
  SENSOR5_UL_ROW, SENSOR5_UL_ROW + SENSOR5_ROWS - 1,
  SENSOR5_UL_COLUMN, SENSOR5_UL_COLUMN + SENSOR5_COLUMNS - 1 );
```c
sprintf( string, "%01f %01f", number_of_readings_to_average, averages + 1.0 );

write_string_in_screen_buffer( string,
    ROW_NUMBER_TO_AVERAGE, COLUMN_NUMBER_TO_AVERAGE );

update_screen();
}while( !kbhit() );

/*
Get the key that was pressed.
*/

do {
    key = (char) getch();
    if ( key == 0 ) getch();
}while( kbhit() );

/*
Set point command.
*/

if ( key == 's' || key == 'S' ) {
    do {
        heater_select = (unsigned char) query_user_for_number(
            "Enter 1 or 2 to select the heater" );
    } while( heater_select < 1 || heater_select > 2 );

do {
    sensor_select = (unsigned char) query_user_for_number(
        "Enter 1, 2, 3, 4, or 5 to select the sensor" );
}while( sensor_select < 1 || sensor_select > 5 );

do {
    set_point = (unsigned char) ( query_user_for_number(
        "Enter 0 ½C to 50 ½C for set point" ) * 255.0 / 50.0 )
} while( set_point < 0 || set_point > 255 );

send_byte_to_heater( 'S' );
send_byte_to_heater( heater_select );
send_byte_to_heater( sensor_select );
send_byte_to_heater( set_point );

if ( heater_select == 1 )
    write_string_in_screen_buffer( "maintain", 
```
void send_byte_to_heater( unsigned char c )
{
    unsigned char echo;
    if ( debug_mode ) return;
    send_byte( c, comm_port );
    wait_for_comm_port_data();
    echo = receive_byte( comm_port );
    if ( echo != c )
    {
        write_string_in_screen_buffer( "Incorrect byte echoed back", ROW_ERRORMESSAGE, COLUMN_ERRORMESSAGE );
        update_screen();
        turn_on_cursor();
        exit( 1 );
    }
}

unsigned char receive_byte_from Heater( unsigned comm_port )
{
    if ( debug_mode )
        return (unsigned char) ( (double) rand() * 255.0 / 32767.0 );
    wait_for_comm_port_data();
    return receive_byte( comm_port );
}

void wait_for_comm_port_data( void )
{
    int key, key_was_hit = 0;
    while ( 0 == ( check_status( comm_port ) & DATA_READY_BIT ) )
    {
        if ( kbhit() )
        {
            key_was_hit = 1;
            key = getch();
        }
ROW_HEATER1_SETPOINT, COLUMN_HEATER1_SETPOINT);

sprintf( string, "sensor %-2d", (int) sensor_select);
write_string_in_screen_buffer( string, 
    ROW_HEATER1_SETPOINT + 1, COLUMN_HEATER1_SETPOINT);

sprintf( string, "at %.1lf°C ", 
    (double) ( set_point * 50.0 / 255.0 ) );
write_string_in_screen_buffer( string, 
    ROW_HEATER1_SETPOINT + 2, COLUMN_HEATER1_SETPOINT );
}

if ( heater_select == 2 )
{
write_string_in_screen_buffer( "maintain", 
    ROW_HEATER2_SETPOINT, COLUMN_HEATER2_SETPOINT );

sprintf( string, "sensor %-2d", (int) sensor_select);
write_string_in_screen_buffer( string, 
    ROW_HEATER2_SETPOINT + 1, COLUMN_HEATER2_SETPOINT );

sprintf( string, "at %.1lf°C ", 
    (double) ( set_point * 50.0 / 255.0 ) );
write_string_in_screen_buffer( string, 
    ROW_HEATER2_SETPOINT + 2, COLUMN_HEATER2_SETPOINT );
}

/includes/serial.h

Change number of averages.
*/

else if ( key == 'a' || key == 'A' )
{
do

    number_of_readings_to_average = query_user_for_number( 
        "Enter number of readings to average for screen display" )

    while( number_of_readings_to_average < 1.0 );

}

/*
Setup to save values to a file.
*/

else if ( key == 'f' || key == 'F' )
{
}

while( key != ESCAPE );

/*
Exit the program after restoring the cursor.
*/

24
if ( key == 0 ) getch();
if ( key == ESCAPE )
{
    turn_on_cursor();
    exit( 1 );
}
}
}
if ( key_was_hit && key != 0 ) ungetch( key );

void turn_on_cursor( void )
{
    change_cursor( 6, 7 );
}

void turn_off_cursor( void )
{
    change_cursor( 63, 63 );
}

void change_cursor( int top, int bottom )
{
    union REGS regs;
    regs.h.ah = (unsigned char) SET_CRSR;
    regs.h.ch = (unsigned char) top;
    regs.h.cl = (unsigned char) bottom;
    int86(VIDEO_IO, &regs, &regs);
}

void background_on_screen( void )
{
    char *cp = screen_characters;
    sprintf( cp, "%s", "
    sprintf( cp, "%s", "

sprintf( cp, "%s", "Heater 1°C" );
sprintf( cp, "%s", " 1°C" );
sprintf( cp, "%s", "aaaaaaaai" );

sprintf( cp, "%s", "Heater 1°C" );
sprintf( cp, "%s", " 1°C" );
sprintf( cp, "%s", "aaaaaaaai" );

sprintf( cp, "%s", "Heater 1°C" );
sprintf( cp, "%s", " 1°C" );
sprintf( cp, "%s", "aaaaaaaai" );

sprintf( cp, "%s", "Esc ... exit to dos
sprintf( cp, "%s", "S ..... set point command
sprintf( cp, "%s", "A ..... set sensor averages
sprintf( cp, "%s", "F ..... save readings in a file

fill_attribute_block( WHITE_ON | BLACK, 0, 24, 0, 79 );

fill_attribute_block( WHITE_ON | BROWN,
   PLATE_UL_ROW, PLATE_UL_ROW + PLATE_ROWS - 1,
   PLATE_UL_COLUMN, PLATE_UL_COLUMN + PLATE_COLUMNS - 1 );

fill_attribute_block( WHITE_ON | BROWN,
   HEATER1_UL_ROW, HEATER1_UL_ROW + HEATER1_ROWS - 1,
   HEATER1_UL_COLUMN, HEATER1_UL_COLUMN + HEATER1_COLUMNS - 1 );

fill_attribute_block( WHITE_ON | BROWN,
   HEATER2_UL_ROW, HEATER2_UL_ROW + HEATER2_ROWS - 1,
   HEATER2_UL_COLUMN, HEATER2_UL_COLUMN + HEATER2_COLUMNS - 1 );

fill_attribute_block( BLACK_ON | WHITE,
   SENSOR1_UL_ROW, SENSOR1_UL_ROW + SENSOR1_ROWS - 1,
   SENSOR1_UL_COLUMN, SENSOR1_UL_COLUMN + SENSOR1_COLUMNS - 1 );

fill_attribute_block( BLACK_ON | WHITE,
   SENSOR2_UL_ROW, SENSOR2_UL_ROW + SENSOR2_ROWS - 1,
   SENSOR2_UL_COLUMN, SENSOR2_UL_COLUMN + SENSOR2_COLUMNS - 1 );

fill_attribute_block( BLACK_ON | WHITE,
   SENSOR3_UL_ROW, SENSOR3_UL_ROW + SENSOR3_ROWS - 1,
   SENSOR3_UL_COLUMN, SENSOR3_UL_COLUMN + SENSOR3_COLUMNS - 1 );

fill_attribute_block( BLACK_ON | WHITE,
   SENSOR4_UL_ROW, SENSOR4_UL_ROW + SENSOR4_ROWS - 1,
   SENSOR4_UL_COLUMN, SENSOR4_UL_COLUMN + SENSOR4_COLUMNS - 1 );
```c
fill_attribute_block( BLACK_ON : WHITE,
SENSOR5_UL_ROW, SENSOR5_UL_ROW + SENSOR5_ROWS - 1,
SENSOR5_UL_COLUMN, SENSOR5_UL_COLUMN + SENSOR5_COLUMNS - 1 );

fill_attribute_block( BRIGHT_WHITE_ON | BROWN,
SENSOR1_UL_ROW + i, SENSOR1_UL_ROW + 1 + SENSOR1_ROWS - i,
SENSOR1_UL_COLUMN, SENSOR1_UL_COLUMN + SENSOR1_COLUMNS - 1 );

fill_attribute_block( BRIGHT_WHITE_ON | BROWN,
SENSOR2_UL_ROW + i, SENSOR2_UL_ROW + 1 + SENSOR2_ROWS - i,
SENSOR2_UL_COLUMN, SENSOR2_UL_COLUMN + SENSOR2_COLUMNS - 1 );

fill_attribute_block( BRIGHT_WHITE_ON | BROWN,
SENSOR3_UL_ROW + i, SENSOR3_UL_ROW + 1 + SENSOR3_ROWS - i,
SENSOR3_UL_COLUMN, SENSOR3_UL_COLUMN + SENSOR3_COLUMNS - 1 );

fill_attribute_block( BRIGHT_WHITE_ON | BROWN,
SENSOR4_UL_ROW + i, SENSOR4_UL_ROW + 1 + SENSOR4_ROWS - i,
SENSOR4_UL_COLUMN, SENSOR4_UL_COLUMN + SENSOR4_COLUMNS - 1 );

fill_attribute_block( BRIGHT_WHITE_ON | BROWN,
SENSOR5_UL_ROW + i, SENSOR5_UL_ROW + 1 + SENSOR5_ROWS - i,
SENSOR5_UL_COLUMN, SENSOR5_UL_COLUMN + SENSOR5_COLUMNS - 1 );

update_screen();

void update_screen( void )
{
    display_screen();
}

void display_screen( void )
{
    /* draw the screen with the contents of the character and attribute arrays */

    int i;
    char *ap = screen_attributes;
    char *cp = screen_characters;
    unsigned char far *video_mem_ptr = (unsigned char far *) (0xB800L <<

    for ( i = 0; i < 2000; ++i )
    {
        *video_mem_ptr++ = (unsigned char) *cp++;
        *video_mem_ptr++ = (unsigned char) *ap++;
    }
}
```
void write_string_in_screen_buffer( char *string, int row, int column )
{
    /* copy the string to the specified location in the screen buffer */
    int location = row * 80 + column;
    while ( *string )
    {
        screen_characters[ location++ ] = *string++;
    }
}

void write_double_in_screen_buffer( char *format_string,
                     double number, int row, int column )
{
    /* copy the double precision floating point number to the specified
    location in the screen buffer using the format string */
    char string[80];
    sprintf( string, format_string, number );
    write_string_in_screen_buffer( string, row, column );
}

void cut_string( char *destination, char *source, int first_char, int last_char)
{
    /* copy specified piece of character string from source to
    destination */
    char * ptr;
    for ( ptr = source + first_char; ptr <= source + last_char; ++ptr )
    {
        *destination++ = *ptr;
    } *destination = 0;
}

void user_edit_field_on_screen( char *string,
                     int row, int start_column, int end_column,
                     char field_foreground, char field_background,
                     char cursor_foreground, char cursor_background )
{
    /* let the user edit a field on the screen and return the edited
     field contents in the string */
    int column, key = 0, key2;
    char saved_field_chars[100], saved_field_attrs[100], saved_attr;
    copy_characters_from_screen( saved_field_chars, row, row, start_column );
    copy_attributes_from_screen( saved_field_attrs, row, row, start_column );
    fill_attribute_block( field_foreground | field_background,
                              row, row, start_column, end_column );
column = start_column;
saved_attr = screen_attributes[ row * 80 + column ];
screen_attributes[ row * 80 + column ] = cursor_foreground | cursor_back;

while ( key != RETURN )
{
    while ( !kbhit() )
    {
        update_screen();
    }

    key = getch();

    if ( key == ESCAPE )
    {
        screen_attributes[ row * 80 + column ] = saved_attr;
        sprintf( string, "%s", saved_field_chars );
        write_string_in_screen_buffer( string, row, start_column);
        column = start_column;
        saved_attr = screen_attributes[ row * 80 + column ];
    }
    else if ( key >= 32 && key <= 126 )
    {
        screen_characters[ row * 80 + column ] = (char) key;
        screen_attributes[ row * 80 + column ] = saved_attr;
        if ( column < end_column ) ++column;
        saved_attr = screen_attributes[ row * 80 + column ];
        screen_attributes[ row * 80 + column ] = cursor_foreground;
    }
    else if ( key == BACKSPACE && column > start_column )
    {
        screen_attributes[ row * 80 + column ] = saved_attr;
        --column;
        saved_attr = screen_attributes[ row * 80 + column ];
        screen_attributes[ row * 80 + column ] = cursor_foreground;
    }
    else if ( key == 0 )
    {
        key2 = getch();
        if ( key2 == LEFT_ARROW && column > start_column )
        {
            screen_attributes[ row * 80 + column ] = saved_attr;
            --column;
            saved_attr = screen_attributes[ row * 80 + column ];
            screen_attributes[ row * 80 + column ] = cursor_foreground;
        }
        else if ( key2 == RIGHT_ARROW && column < end_column )
        {
            screen_attributes[ row * 80 + column ] = saved_attr;
            ++column;
            saved_attr = screen_attributes[ row * 80 + column ];
            screen_attributes[ row * 80 + column ] = cursor_foreground;
        }
    }
}
copy_characters_from_screen(string, row, row, start_column, end_column);
copy_attributes_to_screen(saved_field_attrs, row, row, start_column, end_column);
update_screen();

void copy_characters_from_screen(char *string,
    int start_row, int end_row, int start_column, int end_column)
{
    int row, column;

    for (row = start_row; row <= end_row; ++row)
        for (column = start_column; column <= end_column; ++column)
            *string++ = screen_characters[row * 80 + column];
    *string = 0;
}

void copy_characters_to_screen(char *string,
    int start_row, int end_row, int start_column, int end_column)
{
    int row, column;

    for (row = start_row; row <= end_row; ++row)
        for (column = start_column; column <= end_column; ++column)
            screen_characters[row * 80 + column] = *string++;
}

void copy_attributes_from_screen(char *string,
    int start_row, int end_row, int start_column, int end_column)
{
    int row, column;

    for (row = start_row; row <= end_row; ++row)
        for (column = start_column; column <= end_column; ++column)
            *string++ = screen_attributes[row * 80 + column];
    *string = 0;
}

void copy_attributes_to_screen(char *string,
    int start_row, int end_row, int start_column, int end_column)
{
    int row, column;

    for (row = start_row; row <= end_row; ++row)
        for (column = start_column; column <= end_column; ++column)
            screen_attributes[row * 80 + column] = *string++;
}
void fill_character_block( char c,
in_ start_row, int end_row, int start_column, int end_column )
{
    int row, column;
    for ( row = start_row; row <= end_row; ++row )
        for ( column = start_column; column <= end_column; ++column )
            screen_characters[ row * 80 + column ] = c;
}

void fill_attribute_block( char c,
in_ start_row, int end_row, int start_column, int end_column )
{
    int row, column;
    for ( row = start_row; row <= end_row; ++row )
        for ( column = start_column; column <= end_column; ++column )
            screen_attributes[ row * 80 + column ] = c;
}

char query_user_for_key( char *prompt )
{
    /* Prompt the user for a single key response */

    char back_characters[256], back_attributes[256];
    char front_characters[256], front_attributes[256];
    int length = 0;
    char * p = prompt, key = 0;
    while ( *p++ ) ++length;

    open_window( back_characters, back_attributes,
                 ROW_query_window, ROW_query_window + 2,
                 COLUMN_query_window - length/2 - 1,
                 COLUMN_query_window + length/2 + 5,
                 WHITE_ON | BLACK );

    open_window( front_characters, front_attributes,
                 ROW_query_window - 1,
                 ROW_query_window + 1,
                 COLUMN_query_window - length/2 - 3,
                 COLUMN_query_window + length/2 + 3,
                 BLACK_ON | WHITE );

    write_string_in_screen_buffer( prompt,
                                   ROW_query_window,
                                   COLUMN_query_window - length/2 );

    while ( !kbhit() ) update_screen();
    key = (char) getch();

    close_window( front_characters, front_attributes,
                  ROW_query_window - 1,
                  ROW_query_window + 1,
                  COLUMN_query_window - length/2 - 3,
                  COLUMN_query_window + length/2 + 3 );
double query_user_for_number( char *prompt )
{
/* Prompt the user for a number */

char back_characters[256], back_attributes[256];
char front_characters[256], front_attributes[256];
int length = 0;
char string[80], * p = prompt;
double number = 0.0;

while ( *p++ ) ++length;
length += 10;

open_window( back_characters, back_attributes,
             ROW_query_window,
             ROW_query_window + 2,
             COLUMN_query_window - length/2 - 1,
             COLUMN_query_window + length/2 + 5,
             WHITE_ON | BLACK );

open_window( front_characters, front_attributes,
             ROW_query_window - 1,
             ROW_query_window + 1,
             COLUMN_query_window - length/2 - 3,
             COLUMN_query_window + length/2 + 3,
             BLACK_ON | WHITE );

write_string_in_screen_buffer( prompt,
                                ROW_query_window,
                                COLUMN_query_window - length/2 );

user_edit_field_on_screen( string,
                           ROW_query_window,
                           COLUMN_query_window + length/2 - 7,
                           COLUMN_query_window + length/2 + 1,
                           BRIGHT_WHITE_ON, BLUE,
                           BRIGHT_WHITE_ON, BLACK );

sscanf( string, "%lf", &number );

close_window( front_characters, front_attributes,
              ROW_query_window - 1,
              ROW_query_window + 1,
              COLUMN_query_window - length/2 - 3,
              COLUMN_query_window + length/2 + 3 );

32

close_window( back_characters, back_attributes,
ROW_query_window,
ROW_query_window - 2,
COLUMN_query_window - length/2 - 1,
COLUMN_query_window + length/2 + 5);

update_screen();
return( number );
}

void open_window( char *character_array, char *attribute_array,
int start_row, int end_row, int start_column, int end_column,
char color_byte )
{
    /* open a blank window on the screen, saving the previous contents
    in the character and attribute arrays */
    copy_characters_from_screen( character_array,
        start_row, end_row,
        start_column, end_column );

    copy_attributes_from_screen( attribute_array,
        start_row, end_row,
        start_column, end_column );

    fill_character_block( ' ', start_row, end_row,
        start_column, end_column );

    fill_attribute_block( color_byte,
        start_row, end_row,
        start_column, end_column );
}

void close_window( char *character_array, char *attribute_array,
    int start_row, int end_row, int start_column, int end_column )
{
    /* close a window on the screen, copying the contents of the
    character and attribute arrays to the screen */
    copy_characters_to_screen( character_array,
        start_row, end_row,
        start_column, end_column );

    copy_attributes_to_screen( attribute_array,
        start_row, end_row,
        start_column, end_column );
; THIS PROGRAM IS THE INITIAL ATTEMPT TO COMMUNICATE WITH A PC
; AND ACTIVELY SET TEMPERATURE SET POINTS, DEFINE CONTROL SENSORS
; AND READ DATA TO BE DISPLAYED. THIS IS FOR THE PROGRAMMABLE
; HEATER CONTROLLER USING THE 68HC811E2 MICROCONTROLLER.

; TEMP0 EQU $10
; TEMP1 EQU $11
; TEMP2 EQU $12
; TEMP3 EQU $13
; TEMP4 EQU $14
; TCONT1 EQU $15
; TSET1 EQU $16
; TCONT2 EQU $17
; TSET2 EQU $18
; SENSOR EQU $0F
; HEATNUM EQU $19
; SENSNUM EQU $1A
; SETPOINT EQU $1B
; STATUS EQU $20

PORTA EQU $1000
PORTB EQU $1003
PORTC EQU $1003
DDRC EQU $1007
PORTD EQU $1008
DDRD EQU $1009
PORTE EQU $100A
PACTL EQU $1026

DEFSEG PROGRAM, ABSOLUTE
SEG PROGRAM
ORG $F800

;******************************************************************************
; INITIALIZE STACK, OUTPUT PORTS AND HEATERS
;******************************************************************************

LDS #$00FF ; INITIALIZE STACK
LDY #$1000 ; LOAD Y REG WITH SEGMENT OFFSET
LDAA #$FF ; SET ALL PINS OF PORT C AS OUTPUTS
STAA DDRC ; BY PUTTING FF IN DDRC
BSET $26,Y,$80 ; SETUP PA7 OF PORT A AS OUTPUT
BSET $00,Y,$88 ; TURN HEATERS OFF (0=ON, 1=OFF)
BSET $09,Y,$02 ; SETUP BIT1 PORT D AS OUTPUT
LDAA #$30 ; SET UP SCI FOR 9600 BAUD
STAA $102B
LDAA #$0C ; ENABLE TRANSMIT AND RECEIVE SCI
STAA $102D
LDAA#$04
STAA PORTC
LDAA#$5E
STAA PORTB
BSET$03,Y,$10

;POWER ON A/D CONVERTERS
;LOAD 12 BIT D/A WITH 2.73 VOLT VALUE
;SET GAIN OF PGA200 TO 10

;INITIAL CONDITIONS

LDAA#$03
STAA TCONT1
STAA TCONT2
LDAA#$30
STAA TSET1
STAA TSET2

;READ TEMPS 1-5

MAINLOOP:
BCLR$00,Y,$70
LDAB PORTA
JSR READTEMP
STAA TEMP0
ADDDB#$10
STAB PORTA
JSR READTEMP
STAA TEMP1
ADDDB#$10
STAB PORTA
JSR READTEMP
STAA TEMP2
ADDDB#$10
STAB PORTA
JSR READTEMP
STAA TEMP3
ADDDB#$10
STAB PORTA
JSR READTEMP
STAA TEMP4

;HEATER CONTROL

HEATER1:
LDAB TCONT1
CLRA
XGDX

;LOAD TCONT1 IN X REG
BRCLR $00,Y,$08,H1_ON ;IF BIT 3 IS 0 THEN BRANCH TO H1_ON

H1_OFF:
LDAA TSET1 ;LOAD A WITH HEATER 1 SET POINT TEMP
SUBA #$0A ;SUBTRACT 2 DEGREE
SUBA SENSOR,X ;SUBTRACT CONTROL1 SENSOR TEMPERATURE
BMI HEATER2 

BCLR $00,Y,$08 ;TURN HEATER 1 ON
BRA HEATER2

H1_ON:
LDAA TSET1 ;LOAD A WITH HEATER 1 SET POINT TEMP
SUBA SENSOR,X ;SUBTRACT CONTROL1 SENSOR TEMPERATURE
BPL HEATER2 

BSET $00,Y,$08 ;TURN HEATER 1 OFF

;**********HEATER 2******************************************************************************

HEATER2:
LDAB TCONT2
CLRA
XGDX

BRCLR $00,Y,$80,H2_ON ;IF BIT 0 IS 0 THEN BRANCH TO H2_ON

H2_OFF:
LDAA TSET2 ;LOAD A WITH HEATER 2 SET POINT TEMP
SUBA #$0A ;SUBTRACT 2 DEGREE
SUBA SENSOR,X ;SUBTRACT CONTROL2 SENSOR TEMPERATURE
BMI HEATEND 

BCLR $00,Y,$80 ;TURN HEATER 2 ON
BRA HEATEND

H2_ON:
LDAA TSET2 ;LOAD A WITH HEATER 2 SET POINT TEMP
SUBA SENSOR,X ;SUBTRACT CONTROL2 SENSOR TEMPERATURE
BPL HEATEND 

BSET $00,Y,$80 ;TURN HEATER 2 OFF

HEATEND:

;********** SERIAL PORT CHECK ***********************************************

HERE:
BRCLR $2E,Y,$20,HERE ;IF DATA RECEIVED BYTE NOT SET THEN
;GOTO NOBYTE

LDAA $102F ;READ DATA RECEIVED
CMPA #$44 ;CHECK IF BYTE IS "D"
BNE CHECKS ;IF NOT "D" THEN GOTO CHECKS
JSR SENDBYTE ;ECHO "D" BACK TO PC

CLRA
STAA STATUS
BRCLR $00,Y,$08,H2STAT ;READ STATUS OF HEATER 1
BSET STATUS,$01

H2STAT:
BRCLR $00,Y,$80,SENDSTAT ;READ STATUS OF HEATER 2
BSET STATUS,$02

SENDSTAT:
LDAA STATUS ;SEND STATUS WORD
JSR SENDBYTE
LDAA TEMP0
  JSR SENDBYTE

LDAA TEMP1
  JSR SENDBYTE

LDAA TEMP2
  JSR SENDBYTE

LDAA TEMP3
  JSR SENDBYTE

LDAA TEMP4
  JSR SENDBYTE

JMP NOBYTE

CHECKS:
  CMPA #$53
  BNE NOBYTE
  JSR SENDBYTE

WAITN:
  BRCLR $2E,Y,$20,WAITHN
  LDAA $102F
  STAA HEATNUM
  JSR SENDBYTE

WAITSN:
  BRCLR $2E,Y,$20,WAITSN
  LDAA $102F
  STAA SENSNUM
  JSR SENDBYTE

WAITSP:
  BRCLR $2E,Y,$20,WAITSP
  LDAA $102F
  STAA SETPOINT
  JSR SENDBYTE

FIGURE1:
  LDAA HEATNUM
  CMPA #$02
  BEQ FIGURE2
  LDAA SENSNUM
  STAA TCONT1
  LDAA SETPOINT
  STAA TSET1

  JMP NOBYTE

FIGURE2:
  LDAA SENSNUM
  STAA TCONT2
  LDAA SETPOINT
  STAA TSET2

NOBYTE: JMP MAINLOOP
SENDBYTE SUBROUTINE

SENDBYTE:  STAA $102F ; TRANSMIT BYTE FROM A REG
TWAIT:  BRCLR $2E,Y,$80,TWAIT ; WAIT FOR BYTE TO BE SENT

READTEMP SUBROUTINE

READTEMP:  CLRA
LDAA #$0A ; SET A/D FOR SINGLE SCAN CHANNEL 0

NOTYET1:  DECA
BNE NOTYET1 ; WAIT FOR MUX TO SETTLE
STAA $1030 ; AND START CONVERSION
LDAA #$06

NOTYET:  DECA
BNE NOTYET ; CONVERSION COMPLETE
LDAA $1031 ; READ A/D RESULT
RTS

DEFSEG reset, ABSOLUTE
SEG reset
ORG $FFFE

DB $F8
DB $00

END
APPENDIX B

PHCC TEST PROCEDURES/RESULTS
Materials Required:

- 286 (or better) IBM-compatible PC with a 9-pin serial port
- 9-pin female computer cable wired as shown in figure 1
- PHC program disk with PHCCDE6.BIN, BOOT.BIN, HEATER.EXE, and PROGCOM1.EXE.
- Bipolar 15-V power supply (i.e., +15 V and −15 V)
- 5-V power supply
- Variable 5-V power supply.

Special Notes:

External pin numbers, EP1 through EP29, may be different than used in the actual hybrid design. Check the MSFC drawings to ensure connections are to the right points.

Computer Interface Set-Up:

Connect the computer com1 port to the hybrid circuit as shown in figure 1.

![Diagram of DB9 Female Connector](image)

Next, connect the power supplies to the appropriate pins as shown in figure 2; i.e., +15 V to EP26 and EP24, −15 V to EP27, +5 V to EP28, and GND to EP29 and EP25.
Last, install 10-k ohm \( \frac{1}{4} \)-W resistors in place of heater-1, heater-2, heater-3, and heater-4. These resistors will model the actual heaters and will be connected to pins EP16 through EP23, as shown in figure 3.

**System Testing:**

1. Once all electrical connections are made, power should be turned on to the hybrid circuit. The sum total current of all supplies should not be greater than 100 mA. Record the currents on each power supply.

<table>
<thead>
<tr>
<th>Supply</th>
<th>Measured Value (mA)</th>
<th>Acceptable Range (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+15 V P.S.</td>
<td>30</td>
<td>&lt;200</td>
</tr>
<tr>
<td>-15 V P.S.</td>
<td>30</td>
<td>&lt;200</td>
</tr>
<tr>
<td>+5 V P.S.</td>
<td>20</td>
<td>&lt;200</td>
</tr>
</tbody>
</table>

2. Turn all power supplies off and short EP15 to ground. Once shorted, turn on all power supplies.
3. Insert PHC software disk in the PC and change the active drive to the one containing the disk. At the proper disk prompt, type PROGCOM1 PHCCODE6.BIN. The computer screen should appear as shown in figure 4 and the value for byte number automatically increment up to the value shown in figure 4. Once this completes, the message 'eeprom code successfully transferred' will display.

A:>progcom1 phccode6.bin
The bootstrap code is 256 bytes.
byte #255 ...bootstrap successfully transferred.
The eeprom code is 2048 bytes.
byte #2047 ...eeprom code successfully transferred.
A:>

Figure 4.

4. Remove the shorting wire on EP15.

5. Cycle all power supplies off and then on.

6. Type HEATER on the PC followed by a return. The screen should appear as shown in figure 5.

Figure 5.
7. Enter a 1 (for communication port 1) followed by a carriage return.

8. Values should appear in the spaces for T1 through T8. T1 and T2 are the values for the internal hybrid AD590s which should read between 24 and 33 °C. The rest of the sensors should read '0.' Record the values on the computer screen for T1 and T2 below.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>50 °C</td>
</tr>
<tr>
<td>T2</td>
<td>50 °C</td>
</tr>
</tbody>
</table>

9. Set the variable power supply to 5 V and connect it through a serial 10-k ohm resistor individually to each of the AD590 inputs (see fig. 6). Record the values shown on the computer screen for the channel stimulated by the supply. Repeat this process for the power supply set to 6 V and again at 6.5 V.

<table>
<thead>
<tr>
<th>AD590 Input</th>
<th>Measured Value at 5 V</th>
<th>Measured Value at 6 V</th>
<th>Measured Value at 6.5 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP1 (T3)</td>
<td>0</td>
<td>28</td>
<td>49.5</td>
</tr>
<tr>
<td>EP2 (T4)</td>
<td>0</td>
<td>28</td>
<td>49.8</td>
</tr>
<tr>
<td>EP3 (T5)</td>
<td>0</td>
<td>29</td>
<td>50</td>
</tr>
<tr>
<td>EP4 (T6)</td>
<td>0</td>
<td>29</td>
<td>50</td>
</tr>
<tr>
<td>EP5 (T7)</td>
<td>0</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>EP6 (T8)</td>
<td>0</td>
<td>30</td>
<td>50</td>
</tr>
</tbody>
</table>

Note: At 5 V the measured value should be between 0.0 and 1.0 °C, at 6 V the measured value should be between 28 and 32 °C, and at 6.5 V the measured value should be between 49 and 50 °C.

10. On the PC, type 'S' to select the 'set point command' option. This option will allow the heater controls to be tested. At the prompt, 'Enter 1, 2, 3, or 4 to select the heater,' enter a 1 to make heater 1 active. The software will then prompt, 'Enter 3, 4, 5, 6, or 7 to select the sensor,' enter a 7 to select an AD590 input for control. The next prompt, 'Enter 0 to 50 °C for set point,' enter a value of 30 °C. This sets up the heater to turn on below 30 °C and off at temperatures above 30 °C. The screen on
the computer should reflect the information in one of the four corners, depending on which heater is selected.

11. Using the same variable power supply setup as in step 9, set the voltage to 5 V and inject on EP8. The computer screen should verify that the heater 1 is on by a different color block around the heater. Hook a voltmeter to the heater 1 resistor (the meter should indicate 15 V). Slowly increase the variable voltage supply (maximum of 7 V) until the heater 1 voltage drops to 0, as indicated on the voltmeter. Verify that the computer screen also indicates the heater cutoff and record the voltage on the variable supply at cutoff in the table below.

<table>
<thead>
<tr>
<th>Heater</th>
<th>Cutoff Voltage (V)</th>
<th>Acceptable Range (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heater 1</td>
<td>5.5051</td>
<td>2.9 → 3.1</td>
</tr>
</tbody>
</table>

12. Repeat steps 10 and 11 for the remaining three heaters using the same sensor and set point, but choose a different heater. Remember to move the voltage meter to the active heater and record the values in the table below.

<table>
<thead>
<tr>
<th>Heater</th>
<th>Cutoff Voltage (V)</th>
<th>Acceptable Range (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heater 2</td>
<td>5.503</td>
<td></td>
</tr>
<tr>
<td>Heater 3</td>
<td>5.503</td>
<td></td>
</tr>
<tr>
<td>Heater 4</td>
<td>5.503</td>
<td></td>
</tr>
</tbody>
</table>
Connections to Programmable Heater Controller Hybrid

Package P/U Numbers Correspond With EP Numbers on Schematic

Argo Transdata Corp.
July 21, 1992
A PROGRAMMABLE HEATER CONTROL CIRCUIT FOR SPACECRAFT

by

D.D. Nguyen, J.W. Owen, D.A. Smith, and W.J. Lewter

The information in this report has been reviewed for technical content. Review of any information concerning Department of Defense or nuclear energy activities or programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.
**A Programmable Heater Control Circuit for Spacecraft**

(Leaves blank)

**REPORT DATE**
June 1994

**REPORT TYPE AND DATES COVERED**
Technical Memorandum

**AUTHOR(S)**
D.D. Nguyen, J.W. Owen, D.A. Smith, and W.J. Lewter

**SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Aeronautics and Space Administration
Washington, DC 20546

**ABSTRACT (Maximum 200 words)**

Spacecraft thermal control is accomplished for many components through use of multilayer insulation systems, electrical heaters, and radiator systems. The heaters are commanded to maintain component temperatures within design specifications. The programmable heater control circuit (PHCC) was designed to obtain an effective and efficient means of spacecraft thermal control. The hybrid circuit provides use of control instrumentation as temperature data, available to the spacecraft central data system, reprogramming capability of the local microprocessor during the spacecraft's mission, and the elimination of significant spacecraft wiring. The hybrid integrated circuit has a temperature sensing and conditioning circuit, a microprocessor, and a heater power and control circuit. The device is miniature and housed in a volume which allows physical integration with the component to be controlled. Applications might include battery-powered logic-circuit configurations. A prototype unit with appropriate physical and functional interfaces was procured for testing. The physical functionality and the feasibility of fabrication of the hybrid integrated circuit were successfully verified. The remaining work to develop a flight-qualified device includes fabrication and testing of a Mil-certified part. An option for completing the PHCC flight qualification testing is to enter into a joint venture with industry.