

**2005 Philips ARM Design Contest
Abstract of Project Number AR1793
CONTROL BUILDING TEMPERATURE FROM ANYWHERE IN THE WORLD WITH E-STAT
Remote Temperature Monitor and Control Project**

With the price of energy soaring and the heating season upon us, I am looking for creative ways to conserve energy. What I need is a way to slip into my home or office building when nobody is there and adjust the thermostat to an energy saving level. But wait a minute, if I drive to the building just to turn the heat down I would probably waste as much fuel for my vehicle as I would save by lowering the thermostat a few degrees. Enter E-stat, the remote temperature monitor and control project that allows you to control building temperature via Telnet or Telco.

E-stat is based on the Keil MCB2130 Evaluation Board, which utilizes the Philips LPC2138 ARM Processor. E-stat software was developed with the Keil uVision3 IDE, using C language for the source. Here is a list of the project features:

- Monitor and adjust building temperature via Internet or Telco
- E-stat connects through an RS232 port to a terminal server or modem
- PC communication software is Telnet or Hyperterm
- E-stat works with existing HVAC systems and thermostats
- Disables the existing thermostat on command
- Supports Fahrenheit or Celsius mode
- A two degree temperature threshold prevents HVAC short run cycles
- Simple to administer

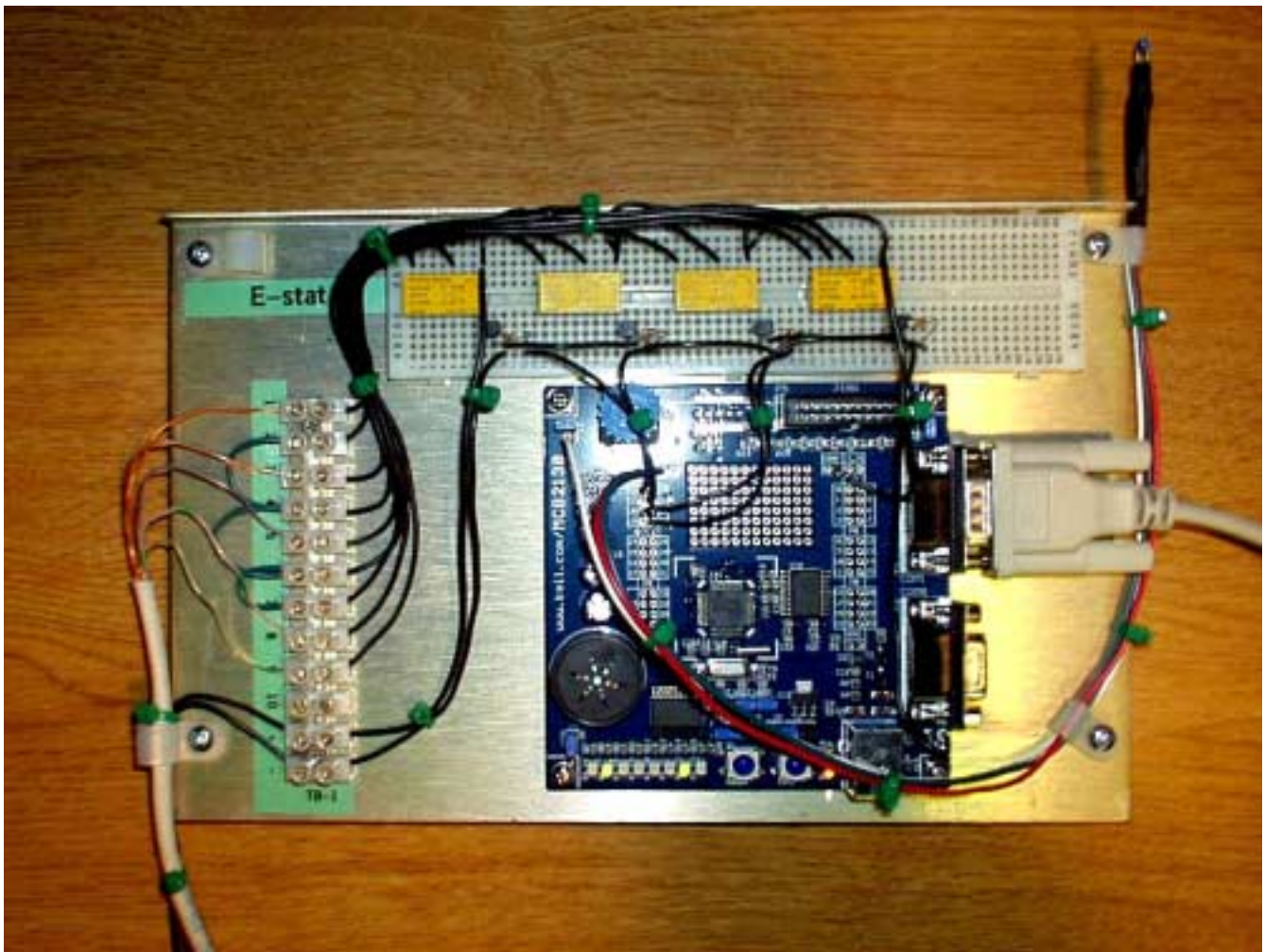


Figure 1 E-stat Hardware

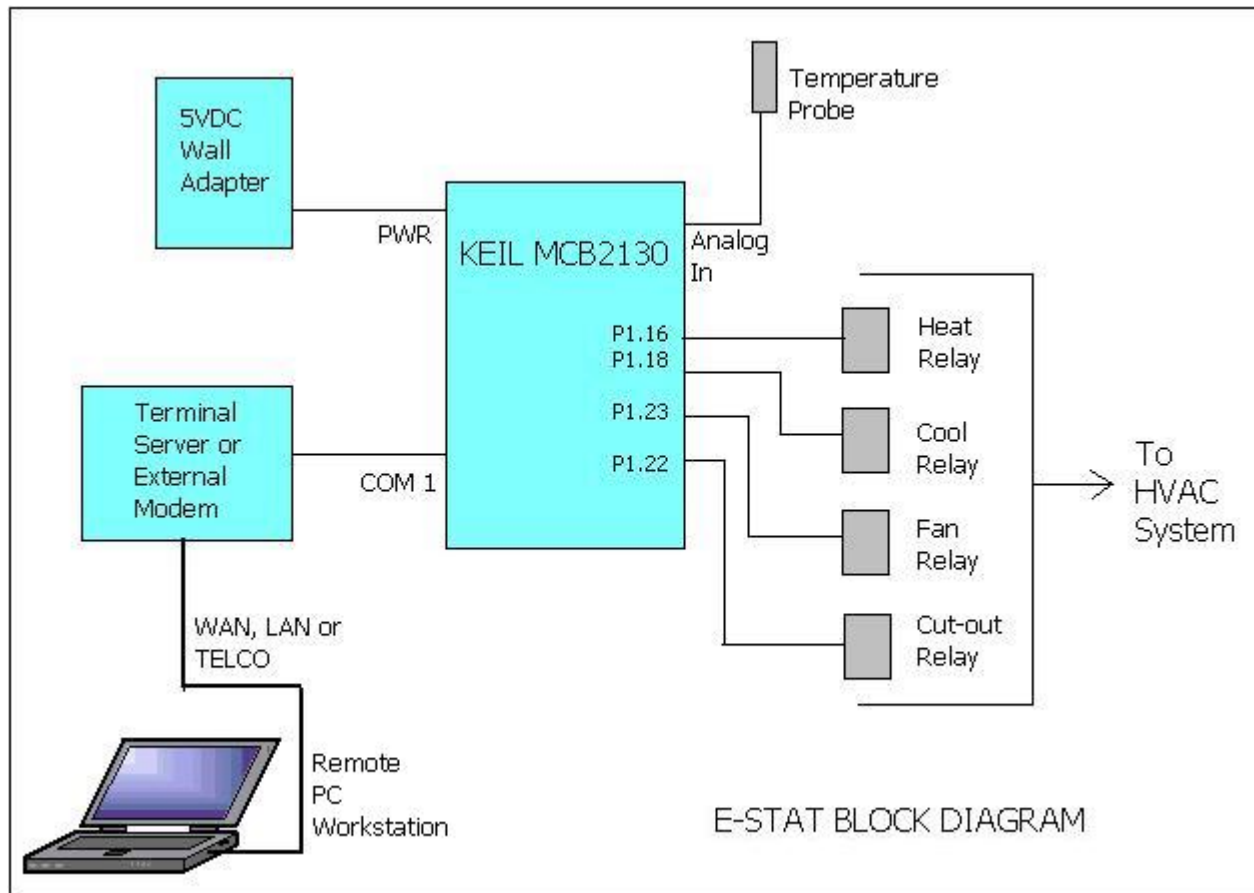


Figure 2 E-stat Hardware Block Diagram

E-stat Hardware Design

The E-stat hardware is based on the Keil MCB2130 evaluation board. This is one of the best evaluation boards that I have worked with. The board is clean, has ample external connection points and jumpers to disable any unwanted options. The board worked right out of the box and, as a bonus, steals its power from a PC USB port. The pre-loaded sample application, which plays a series of sound files, is the coolest thing I've seen on an evaluation board. This board just begs you to build something.

The schematic diagram can be seen in figure 6. Basically, the E-stat is the MCB2130 with the following components added:

- Four 5-volt SPDT relays, with contacts rated for 1 Amp at 50 Volts. The relays, K1 through K4, provide the dry contacts needed to control HVAC systems. K1 operates the heater, K2 operates the Chiller, K3 operates the fan or blower and K4 is the thermostat cut-out relay. K-4 is energized when the E-stat is enabled; this disables the existing building thermostat so that the E-stat has control of the HVAC system. K-4 does this by removing the control voltage from the building thermostat and applying the control voltage to the other three E-stat relays.
- Four relay driver circuits are comprised of 2N3904 NPN Transistors and 1K-Ohm base current limiting resistors. I was able to leave the 1N4001 relay clamping diodes out of the final design because they are built into the Mitsushita relays that I used.
- The temperature probe consists of an NC-103 thermistor and a 10K-ohm resistor in series. This provides a voltage divider network to the analog input of the LPC2138 processor. The resistance to temperature ratio of the NC-103 can be seen in figure 5. Note that the device has a negative temperature coefficient and offers a resistance of 10K-ohms at 25 degrees Celsius. The thermistor's non-linearity is handled by a software table and approximation formula.
- A 5-volt 500ma wall adapter is used for the power supply in the final design.

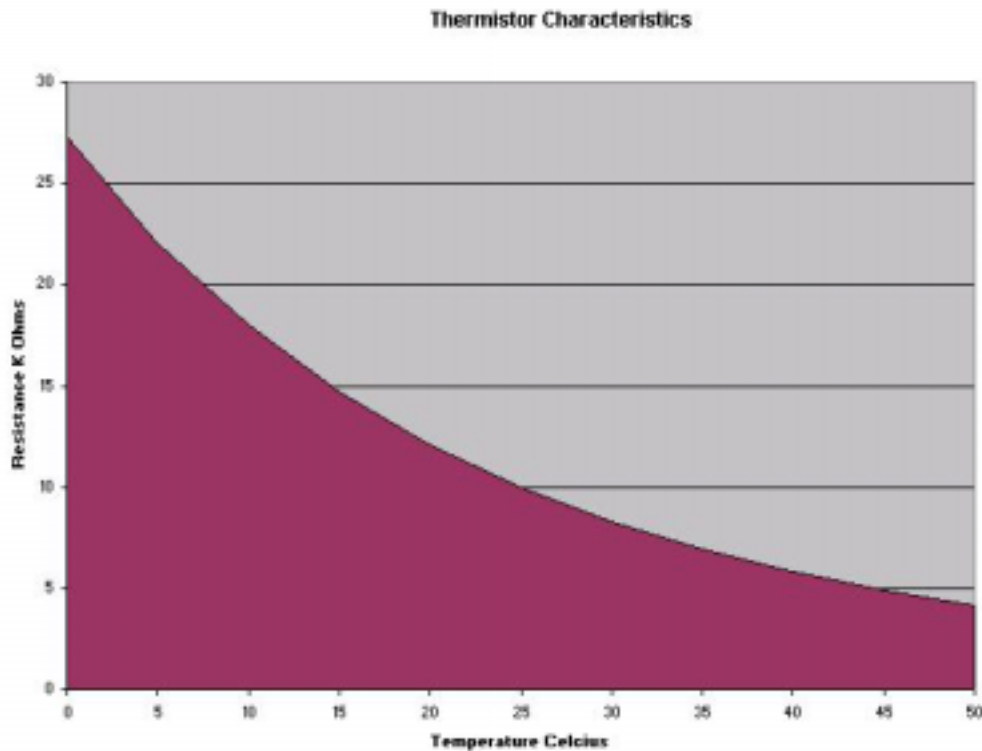
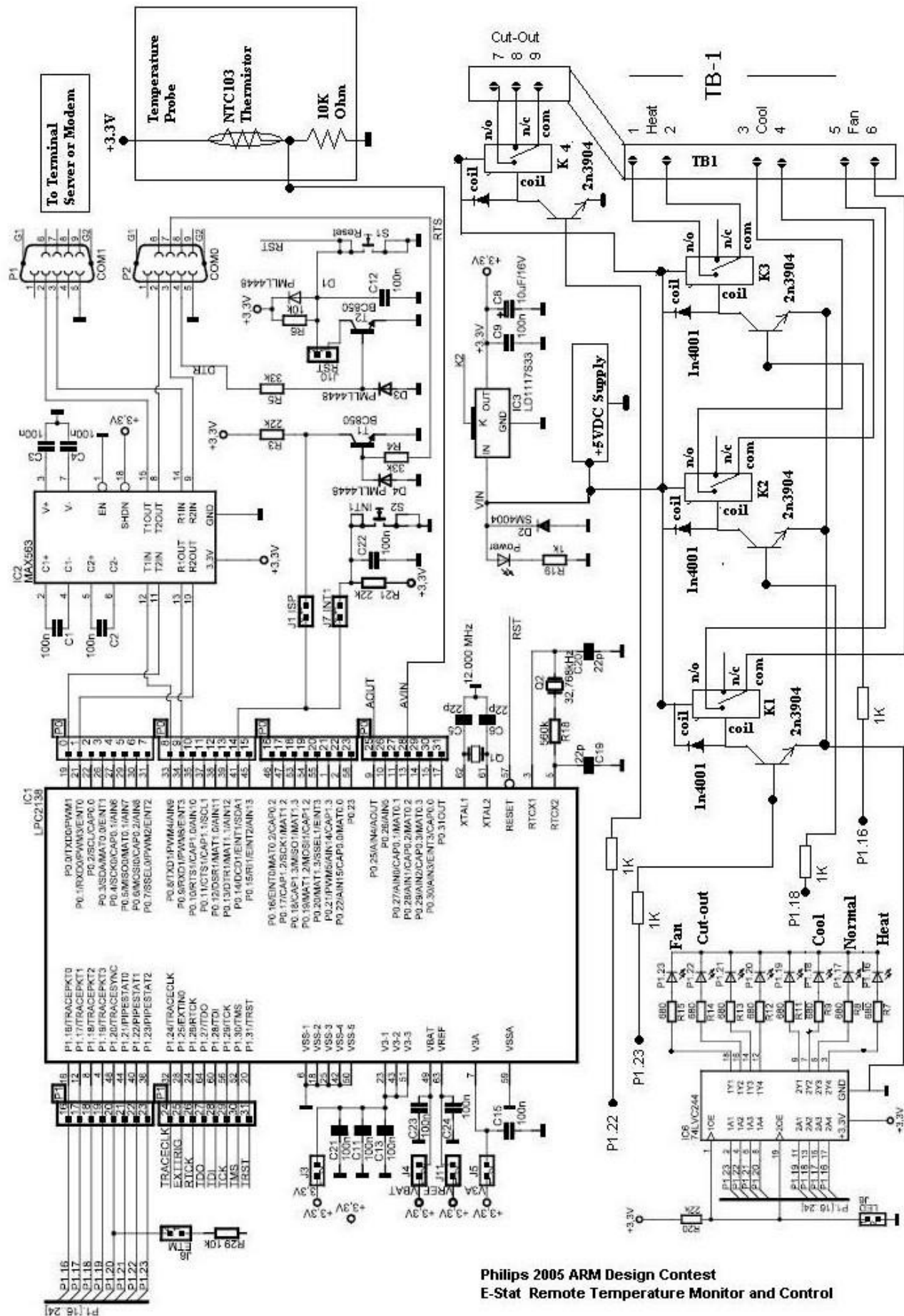


Figure 5 NC-103 Thermistor Characteristics



Philips 2005 ARM Design Contest
E-Stat Remote Temperature Monitor and Control

Figure 6 E-stat Schematic Diagram

2005 Philips ARM Design Contest Project Number AR1793 E-stat Software Description

The get_temperature function, part of main.c, calculates the ambient temperature. The variable temp_val is a short table with the known A/D readings of the temperature probe. The known readings were calculated using the device data sheet and are for Celsius temperature starting at 0 degrees and ending at 50 degrees. The get_temperature function gets an A/D reading of the temperature probe. The function then reads through the temp_val table until it finds a value greater than the A/D value. This process brings the temperature reading within 5 degrees. A downward approximation to the whole single degree celsius is then made with these statements:

```
gp0 = temp_val[counter-1]; //gp0 = min value, gp1 = max value, thermistor lies in between
temp_cel = (counter * 5); //This is the initial temperature, this will be high
gp2 = (gp1-gp0)/(thermistor - gp0); //Approximate downward using this formula
temp_cel = temp_cel - gp2;
```

The regulate_temp routine, located in main.c, makes the decision to heat, cool or do nothing based on the ambient temperature and the minimum and maximum temperature settings. The regulate_temp routine is called on 3 second intervals by the timer0 interrupt service routine, regulate_temp then calls get_temperature for a current ambient temperature reading. A comparison is done against minimum and maximum temperatures and a decision is made. Note that heat and cool set a temperature threshold of 2 degrees, which must be overcome during the heat and cool cycle to return to normal. The 2 degree threshold prevents HVAC short heat and cool cycles.

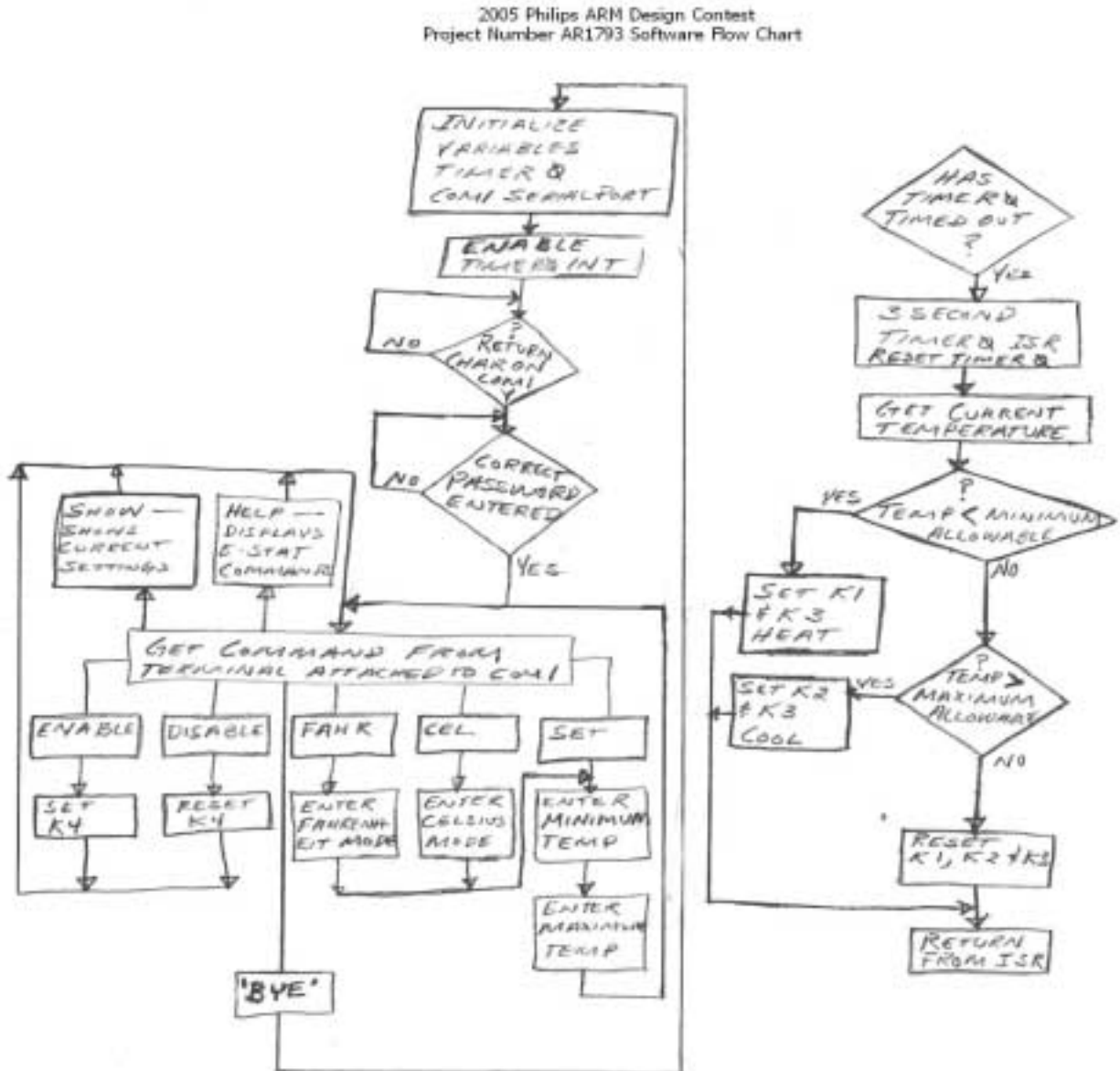


Figure 7 E-stat Software Flow Chart

2005 Philips ARM Design Contest

Project Number AR1793

E-stat Partial Software Code Listing

```

/*****/
/* Circuit Cellar 2005 Philips ARM Design Contest */
/* Project Number AR1793 , Project Name E-Stat */
/* PHILIPS LPC2138 REMOTE TEMPERATURE MONITOR AND CONTROL PROJECT */
/* Based on the Keil MCB2130 Evaluation Board */
/*****/
/* This application monitors and manages building temperature from a */
/* remote location via Telnet or Modem connection */
/* */
/* MAIN.C: */
/*****/

#include <stdio.h> /* standard I/O .h-file */
#include <LPC213x.H> /* LPC21xx definitions */
#include <math.h>
#include <string.h>

//extern int getkey(void);
extern void init_serial (void); /* Initialize Serial Interface */
extern void init_timer (void); /* 3 second Timer Interrupt Routine */
extern unsigned long timeval;

//This table contains the known values for A/D reading the thermistor, in 5°C increments
extern signed int temp_val[11] = {8789,10239,11719,13277,14894,16384,17892,19341,20702,21973,23139};
signed int thermistor;

int temp_cel;
int temp_fahr;
int temp_work;
int min_set = 60; //Set the default min temp
int max_set = 90; //set the default max temp
int threshold = 0; //Set temperature threshold to 0 initially
int gp0; //General purpose registers, used for temporary storage and calculations
int gp1;
int gp2;
int status = 0;
int mode = 0x46; //0x46 = F for Fahrenheit, 0x43 = C for Celsius
char str_comp;
char key_str[10];
int counter;
char buf [12];
char password [12] = ("therm123");//Put your 8 char password here

//Read thermistor voltage and convert it to temperature
void get_temperature(void) { /* Get Thermistor Value */
    unsigned int val;
    AD0CR |= 0x01000000; /* Start A/D Conversion */
    do {
        val = AD0DR; /* Read A/D Data Register */
    } while ((val & 0x80000000) == 0); /* Wait for end of A/D Conversion */
    AD0CR &= ~0x01000000; /* Stop A/D Conversion */
    thermistor = ((val >> 1) & 0x7FE0); /* Extract AIN0 Value */

    counter = 0; //perform conversion to celcius temp
    gp1 = 0;
    while (thermistor > gp1) {
        gp1 = temp_val[counter]; //Read through the array, increment counter
        ++counter;
    }
    counter --;
    gp0 = temp_val[counter-1]; //gp0 = min value, gp1 = max value, thermistor lies in between
    temp_cel = (counter * 5); //This is the initial temperature, this will be high
    gp2 = (gp1-gp0)/(thermistor - gp0); //Approximate downward using this formula
    temp_cel = temp_cel - gp2;
    temp_fahr = ((temp_cel*1.8)+32); //Calculate fahrenheit temp

    if (mode == 0x46) //0x46 = F for Fahrenheit
        temp_work = temp_fahr;

```

```

if (mode == 0x43) //0x43 = C for Celsius
temp_work = temp_cel;
}
//Display E-Stat settings and temperature on terminal
void show_settings (void){
get_temperature();
if (status == 1)
printf ("\nE-STAT IS CURRENTLY ENABLED");
else if (status != 1)
printf ("\nE-STAT IS CURRENTLY DISABLED");
if (mode == 0x43)
printf ("\nTEMPERATURE MODE IS SET TO CELSIUS");
else if (mode != 0x43)
printf ("\nTEMPERATURE MODE IS SET TO FAHRENHEIT");
printf ("\nTHE CURRENT TEMPERATURE IS %d'%c",temp_work,mode);
printf ("\nTHE MINIMUM TEMPERATURE IS %d'%c",min_set,mode );
printf ("\nTHE MAXIMUM TEMPERATURE IS %d'%c",max_set,mode );
}

//Set minimum and maximum temperatures
void set_temperature (void){
set_min:
printf("\n\nSET NEW MINIMUM TEMPERATURE '%c: ",mode); scanf("%d",&min_set);
printf("\nTHE MINIMUM TEMPERATURE IS SET TO %d'%c",min_set,mode);
set_max:
printf("\n\nSET NEW MAXIMUM TEMPERATURE '%c: ",mode); scanf("%d",&max_set);
if (max_set<min_set+5) //do over if max_set not > min_set
goto set_min;
printf("\nTHE MAXIMUM TEMPERATURE IS SET TO %d'%c",max_set,mode);
}

//Print the help information on the terminal
void help (void){
printf ("\nE-STAT Command listing, three letter commands are shown ");
printf ("\nbelow in parenthesis and must be entered in lower case");
printf ("\n(cel)sius temperature mode");
printf ("\n(fah)renheit temperature mode");
printf ("\n(ena)ble E-Stat" );
printf ("\n(dis)able E-Stat" );
printf ("\n(hel)p");
printf ("\n(sho)w current settings");
printf ("\n(set) temperature");
printf ("\n(bye) - logout" );
}

//Decide if temperature is too cold, hot or normal, activate relays accordingly
void regulate_temp (void){
get_temperature(); //Get the current ambient temperature
if (temp_work < (min_set+threshold))
goto heat;
if (temp_work > (max_set-threshold))
goto cool;
goto normal;

heat: //It's too cold, fire up the heater
IOCLR1 = 0xbf0000; //Clear the relays, except for the cut-out
IOSET1 = 0x810000; //Set K1 and K3
threshold = 2; //temperature threshold set to 2
goto reg_done;

cool: //It's too hot, turn on the chiller
IOCLR1 = 0xbf0000; //Clear the relays, except for the cut-out
IOSET1 = 0x840000; //Set K2 and K3
threshold = 2; // temperature threshold set to 2
goto reg_done;

normal: //The temp is in range, do nothing
IOCLR1 = 0xbf0000; //Clear the relays, except for the cut-out
IOSET1 = 0x020000; //Set the 'Normal LED'
threshold = 0;
goto reg_done;
}

```

```
reg_done:
;
}
```

```
int main (void) {
    IODIR1 = 0x00FF0000;          /* P1.16..23 defined as Outputs */
    ADOCR  = 0x00200402;          /* Setup A/D: 10-bit AINO @ 3MHz */
    PINSEL1 = 0x01080000;          /* enable DAC */

    init_serial();                /* Initialize Serial Interface to 1200 baud */
    init_timer ();

    while (1) {                    /* Loop forever */
        get_password:
        printf("\nENTER E-STAT PASSWORD: ");
        gets (buf, sizeof(buf)-1);
        str_comp = strncmp (buf, password, 8);          //Compare 1st 8 characters of the password with buf
        if (str_comp != 0) goto get_password;          //Incorrect password = start over
        printf("\nWELCOME TO E-STAT");

        menu:
        printf("\n>");
        gets (buf, sizeof(buf)-1);
        // printf("%s",buf);
        str_comp = strncmp (buf, "help", 3);          //Compare 1st 3 characters of the buf with "help"
        if (str_comp == 0) help();          //

        str_comp = strncmp (buf, "show", 3);          //Compare 1st 3 characters of the buf with "show"
        if (str_comp == 0) show_settings();          //

        str_comp = strncmp (buf, "set", 3);          //Compare 1st 3 characters of the buf with "set"
        if (str_comp == 0) set_temperature();          //

        str_comp = strncmp (buf, "bye", 3);          //Compare 1st 3 characters of the buf with "bye"
        if (str_comp == 0) main();          //

        str_comp = strncmp (buf, "ena", 3);          //Compare 1st 3 characters of the buf with "ena"
        if (str_comp == 0){
            IOSET1 = 0x400000;          //Set the cut-out relay
            printf ("\nE-STAT IS NOW ENABLED");
            status = 1;          //Set status flag to indicate the E-stat is enabled
        }
        str_comp = strncmp (buf, "dis", 3);          //Compare 1st 3 characters of the buf with "dis"
        if (str_comp == 0){
            IOCLR1 = 0x400000;          //Reset the cut-out relay
            printf ("\nE-STAT IS NOW DISABLED");
            status = 0;          //reset the status flag to indicate the E-stat is disabled
        }
        str_comp = strncmp (buf, "celsius", 3);          //Compare 1st 3 characters of the buf with "cel"
        if (str_comp == 0){
            printf ("\nMODE IS NOW SET TO CELSIUS");
            mode = 0x43;          //0x43 = C for celsius
            set_temperature();
        }
        str_comp = strncmp (buf, "fahr", 3);          //Compare 1st 3 characters of the buf with "fah"
        if (str_comp == 0){
            printf ("\nMODE IS NOW SET TO FAHRENHEIT");
            mode = 0x46;          //0x46 = F for fahrenheit
            set_temperature();
        }
        goto menu;
    }
}
```