/\*\*

McCauley Propeller Systems

Temp Box Control v2.0

Programmer: Timothy Williams

V2.0 upgrades include:

- use non-contact thermometers, MLX90614, for blade monitoring and box

monitoring

- display current temperature

- display current setpoint

- allow changing of setpoint dynamically

- using EEPROM - "remember" settings after power cycle

- allow changing from "Hotbox" to "Coldbox" with a button push

//now include libraries.

\*/

#include <EEPROM.h>

#include <i2cmaster.h>

#include <SoftwareSerial.h>

byte buttons[] = {4, 9, 12, A0}; // the analog 0-5 pins are also known as 14-19

#define DEBOUNCE 10

// This handy macro lets us determine how big the array up above is, by checking the size

#define NUMBUTTONS sizeof(buttons)

// we will track if a button is just pressed, just released, or 'currently pressed'

volatile byte pressed[NUMBUTTONS], justpressed[NUMBUTTONS], justreleased[NUMBUTTONS];

//\*\*\*\*\* Arduino Mini Connections\*\*\*\*\*\*\*\*\*\*

SoftwareSerial lcd(200,7); //Serial Port for LCD display, '200' is a dummy channel

// and the RX is plugged into D7 of the Arduino

//D0 is being used for serial communication with the computer

//D1 is being used for serial communication with the computer

const int Pin\_AC1 = 2; // Control Output 1

const int Pin\_AC2 = 3; // Control Output 2

// Button1 = 4; // Mode Select Button - Input

const int Pin\_BladeOnePWM = 5; //PWM output for Blade 1 temp

const int Pin\_BladeTwoPWM = 6; //PWM output for Blade 2 temp

// D7 is being used by LCD RX Serial Communication

// D8 is SPARE, Not connected

// Button2 = 9; // Temp control UP button

const int Pin\_BoxPWM = 10; // the output for box temp data

const int Pin\_ShopPWM = 11; // output pin for shop ambient temp data

// Button3 = 12; // Temp control DOWN button

// Button4 = A0; // Display Select Button

const int Pin\_LED2 = 13; // Mode COLD indicator

const int Pin\_ShopThermistor = A1; // Analog input for shop temp thermistor

const int Pin\_LED3 = A2; //Mode HOT indicator

const int Pin\_LED4 = A3; // Control active indication

//A4 is being utilized in I2C-SDA for temp probes

//A5 is being utilized in I2C-SCL for temp probes

const int Pin\_BoxThermistor = A6; //A6 is connected to a thermistor located inside the box

//A7 is SPARE, Not connected

int device1Address = 0x2A<<1; // I2C address for Blade 1

// These are both plugged into (A4, A5)

int device2Address = 0x33<<1; // I2C address for Blade 2

float Celcius1 = 0; // Variable to hold temperature in Celcius

// for Blade 1.

float Fahrenheit1 = 0; // Variable to hold temperature in Fahrenheit

// for Blade 1.

float Celcius2 = 0; // Variable to hold temperature in Celcius

// for Blade 2.

float Fahrenheit2 = 0; // Variable to hold temperature in Fahrenheit

// for Blade 2.

float AmbientCelcius = 0; // Enviromental chamber temperature

float AmbientFahrenheit = 0;

float BoxFahrenheit = 0;

float LabTempFahrenheit = 0; // Temperature outside the box

// Variables for thermistor reading for Outside Ambient temperature

const int ThermistorNominal = 10000; // resistance at 25 degrees C

const int TemperatureNominal = 25; // temp. for nominal resistance (almost always 25 C)

const int Num\_Samples = 5; // how many samples to take and average, more takes longer

// but is more 'smooth'

const int BCoefficient = 3977; // The beta coefficient of the thermistor (usually 3000-4000)

const int Series\_Resistor = 10000; // the value of the 'other' resistor

float Steinhart;

//control parameters for air conditioner loop

int TempLow = 60; //the low temp point where heat gun turn on

int TempHi = 190; // the high temp point where heat gun turns off

int SetTemp;

const long ShortCycleDelay = 40000; //short cycle delay to allow compressor time to depressurize

int Samples[Num\_Samples];

long previousMillis;

long currentMillis;

byte AC = LOW;

//EEPROM addresses for variable retention

int EE\_ini = 3; //has the eeprom been initialized before?

int EE\_mode = 2; //what mode was the controller before shutdown?

//Mode 0: Hot Temp Control

//Mode 1: Lo Temp Control

//Mode 2: No control, just monitor

int EE\_low = 1; //where is the low set point?

int EE\_hi = 0; //where is the high set point?

// Mode control variables

int CurrentMode;

char\* Current[ ]={"Hot" , "Cold" , "Mon"};

byte Press\_Once = 0;

unsigned long Mode\_Time = millis();

unsigned long Time\_Out = 1000;

// Select control variables

byte Select\_State; // Range of values: 0 - Blade Temps

// 1 - Amb/Box Temp

// 2 - Current Set Temp

// Display ON/OFF

boolean StateDisplay = true;

unsigned long TimeDisplay = 0;

unsigned long Timeout = 5 \* 60 \*1000; // the amount of time I want the display

// to remain on while buttons aren't being

// pressed

////////////////////////////////////////////////////////////////

//

// SETUP LOOP

//

void setup()

{

Serial.begin(9600); // Start serial communication at 9600bps.

lcd.begin(9600); //Start serial communication for LCD at 9600bps.

delay(5);

setBacklight(4); // Lower backlighting, consumes less power

clearDisplay(); //Clear the display.

setLCDCursor(0x01); // Set cursor to the 2nd spot, 1st line

lcd.print("McCauley Temp");

setLCDCursor(0x41); //Set cursor to the 2nd spot, 2nd line

lcd.print("Starting Up");

//\*\* Setup Arduino pins for Input/Output Functions \*\*

pinMode(Pin\_AC1, OUTPUT);

pinMode(Pin\_AC2, OUTPUT);

pinMode(Pin\_BladeOnePWM, OUTPUT);

pinMode(Pin\_BladeTwoPWM, OUTPUT);

pinMode(Pin\_BoxPWM, OUTPUT);

pinMode(Pin\_ShopPWM, OUTPUT);

pinMode(Pin\_LED2, OUTPUT);

pinMode(Pin\_ShopThermistor, INPUT);

pinMode(Pin\_LED3, OUTPUT);

pinMode(Pin\_LED4, OUTPUT);

pinMode(Pin\_BoxThermistor, INPUT);

//\*\* Initial values for LED indicators

digitalWrite(Pin\_LED2, HIGH);

digitalWrite(Pin\_LED3, HIGH);

digitalWrite(Pin\_LED4, HIGH);

//\*\*\* Check for EEPROM status \*\*\*\*\*

byte initial = EEPROM.read(EE\_ini);

if( initial == 0)

{

Serial.println("W,Using data from EEPROM");

CurrentMode = EEPROM.read(EE\_mode);

if (CurrentMode < 2) SetTemp = EEPROM.read(CurrentMode);

}

else

{

Serial.println("W,Writing default data to EEPROM");

EEPROM.write(EE\_ini, 0);

EEPROM.write(EE\_mode, 2);

EEPROM.write(EE\_low, TempLow);

EEPROM.write(EE\_hi, TempHi);

}

//\*\* Initialize I2C network with the MLX90614 IR temp sensors

i2c\_init(); // Initialise the i2c bus.

PORTC = (1 << PORTC4) | (1 << PORTC5); // Enable pullups.

for (int i=0; i < NUMBUTTONS; i++)

{ pinMode(buttons[i], INPUT);

digitalWrite(buttons[i], HIGH);

}

// Run timer2 interrupt every 15 ms

TCCR2A = 0;

TCCR2B = 1<<CS22 | 1<<CS21 | 1<<CS20;

//Timer2 Overflow Interrupt Enable

TIMSK2 |= 1<<TOIE2;

}

//////////////////////////////////////////////////////////////////////////////////////

//

// Start Main Loop

//

void loop()

{

//Begin gathering data needed

Celcius1 = temperatureCelcius(device1Address); // Read's data from MLX90614

Celcius2 = temperatureCelcius(device2Address); // with the given address,

AmbientCelcius = ambientCelcius(device1Address, device2Address); // transform's it into

// temperature in Celcius and

// store's it in the celcius1

// or celcius2 variables.

Fahrenheit1 = (Celcius1\*1.8) + 32; // Converts celcius into Fahrenheit

Fahrenheit2 = (Celcius2\*1.8) + 32; // and stores in Fahrenheit1 or

AmbientFahrenheit = (AmbientCelcius \* 1.8)+32; // Fahrenheit2 variables.

LabTempFahrenheit = readThermistor(Pin\_ShopThermistor);

BoxFahrenheit = readThermistor(Pin\_BoxThermistor);

//Begin Control Section

currentMillis = millis();

if (CurrentMode == 1)

{

digitalWrite(Pin\_LED2, LOW);

digitalWrite(Pin\_LED3, HIGH);

lowtemp();

}

if (CurrentMode == 0)

{

digitalWrite(Pin\_LED2, HIGH);

digitalWrite(Pin\_LED3, LOW);

hitemp();

}

if (CurrentMode == 2)

{

digitalWrite(Pin\_LED2, HIGH);

digitalWrite(Pin\_LED3, HIGH);

}

//send temps to analog output

analogWrite(Pin\_BladeOnePWM, map(Fahrenheit1, 90, 290, 0, 255));

analogWrite(Pin\_BladeTwoPWM, map(Fahrenheit2, 90, 290, 0, 255));

analogWrite(Pin\_BoxPWM, map(BoxFahrenheit, 30, 250, 0, 255));

analogWrite(Pin\_ShopPWM, map(LabTempFahrenheit, 50, 120, 0, 255));

if (Select\_State == 0)

{

clearbottomrow();

setLCDCursor(0x40);

lcd.print(" B1:");

lcd.print(Fahrenheit1,0);

setLCDCursor(0x48);

lcd.print(" B2:");

lcd.print(Fahrenheit2,0);

}

if (Select\_State == 1)

{

clearbottomrow();

setLCDCursor(0x41);

lcd.print("Box:");

lcd.print(BoxFahrenheit,0);

setLCDCursor(0x49);

lcd.print("Amb:");

lcd.print(LabTempFahrenheit,0);

}

if (Select\_State == 2)

{

clearDisplay();

setLCDCursor(0x01);

lcd.print("Current:");

lcd.print(Current[CurrentMode]);

setLCDCursor(0x41);

lcd.print("Set:");

lcd.print(SetTemp);

setLCDCursor(0x49);

lcd.print("Box:");

lcd.print(BoxFahrenheit,0);

}

if (Select\_State == 3 && millis()- Mode\_Time > Time\_Out)

{

clearDisplay();

delay(2);

setLCDCursor(0x01);

delay(1);

lcd.print("McCauley Temp");

Press\_Once = 0;

Select\_State = 0;

}

//\*\*\* Now we will check the status on the buttons

if (justpressed[0])

{

justpressed[0] = 0;

Select\_State = 3;

clearDisplay();

setLCDCursor(0x01);

lcd.print("Current:");

delay(1);

lcd.print(Current[CurrentMode]);

digitalWrite(Pin\_LED2, HIGH);

digitalWrite(Pin\_LED3, HIGH); //reset LED indicators

Mode\_Time = millis();

if (Press\_Once == 0)

{

Press\_Once ++;

}

else if (Press\_Once == 1)

{

CurrentMode ++;

setLCDCursor(0x10);

delay(1);

if (CurrentMode > 2)

{CurrentMode = 0;}

changemode(CurrentMode);

setLCDCursor(0x01);

lcd.print("Current:");

delay(1);

lcd.print(Current[CurrentMode]);

delay(500);

}

}

if (justpressed[1]) //Temp Up button

{

justpressed[1] = 0;

SetTemp ++;

if (CurrentMode == 1)

{EEPROM.write(EE\_low, SetTemp);}

else if (CurrentMode == 0)

{EEPROM.write(EE\_hi, SetTemp);}

}

if (justpressed[2]) //Temp Down Button

{

justpressed[2] = 0;

SetTemp --;

if (CurrentMode == 1)

{EEPROM.write(EE\_low, SetTemp);}

else if (CurrentMode == 0)

{EEPROM.write(EE\_hi, SetTemp);}

}

if (justpressed[3])

{

justpressed[3] = 0;

Select\_State ++;

if (Select\_State > 2)

{

{Select\_State = 0;}

clearDisplay();

}

}

Serial.print("H");

Serial.print(",");

Serial.print(Fahrenheit1);

Serial.print(",");

Serial.print(Fahrenheit2);

Serial.print(",");

Serial.print(AmbientFahrenheit);

Serial.print(",");

Serial.print(LabTempFahrenheit);

Serial.print(",");

Serial.print(BoxFahrenheit);

Serial.print(",");

Serial.print(Select\_State);

Serial.print(",");

Serial.print(CurrentMode);

Serial.print(",");

Serial.print(SetTemp);

Serial.print(",");

Serial.print(freeRam());

Serial.print(",");

Serial.print(Press\_Once);

Serial.print(",");

Serial.print(millis());

Serial.print(",");

Serial.println(previousMillis);

}

////////////////////////////////////////////////////////////////

//

// Read IR Target Temp

//

float temperatureCelcius(int address)

{

int dev = address;

int data\_low = 0;

int data\_high = 0;

int pec = 0;

// Write

i2c\_start\_wait(dev+I2C\_WRITE);

i2c\_write(0x07);

// Read

i2c\_rep\_start(dev+I2C\_READ);

data\_low = i2c\_readAck(); // Read 1 byte and then send ack.

data\_high = i2c\_readAck(); // Read 1 byte and then send ack.

pec = i2c\_readNak();

i2c\_stop();

// This converts high and low bytes together and processes temperature,

// MSB is a error bit and is ignored for temps.

double tempFactor = 0.02; // 0.02 degrees per LSB (measurement

// resolution of the MLX90614).

double tempData = 0x0000; // Zero out the data

int frac; // Data past the decimal point

// This masks off the error bit of the high byte, then moves it left

// 8 bits and adds the low byte.

tempData = (double)(((data\_high & 0x007F) << 8) + data\_low);

tempData = (tempData \* tempFactor)-0.01;

float celcius = tempData - 273.15;

// Returns temperature in Celcius.

return celcius;

}

/////////////////////////////////////////////////////////////////

//

// Read IR Ambient Temperature

//

float ambientCelcius(int address1, int address2) {

int dev1 = address1;

int dev2 = address2;

int data\_low = 0;

int data\_high = 0;

int pec = 0;

// Write for the first address

i2c\_start\_wait(dev1+I2C\_WRITE);

i2c\_write(0x06);

// Read for the first address

i2c\_rep\_start(dev1+I2C\_READ);

data\_low = i2c\_readAck(); // Read 1 byte and then send ack.

data\_high = i2c\_readAck(); // Read 1 byte and then send ack.

pec = i2c\_readNak();

i2c\_stop();

// This converts high and low bytes together and processes temperature,

// MSB is a error bit and is ignored for temps.

double tempFactor = 0.02; // 0.02 degrees per LSB (measurement

// resolution of the MLX90614).

double tempData = 0x0000; // Zero out the data

int frac; // Data past the decimal point

// This masks off the error bit of the high byte, then moves it left

// 8 bits and adds the low byte.

tempData = (double)(((data\_high & 0x007F) << 8) + data\_low);

tempData = (tempData \* tempFactor)-0.01;

float celcius1 = tempData - 273.15;

//Rezero certain parameters

data\_low = 0;

data\_high = 0;

pec = 0;

tempData = 0x0000;

// Write for the second address

i2c\_start\_wait(dev2+I2C\_WRITE);

i2c\_write(0x06);

// Read for the second address

i2c\_rep\_start(dev2+I2C\_READ);

data\_low = i2c\_readAck(); // Read 1 byte and then send ack.

data\_high = i2c\_readAck(); // Read 1 byte and then send ack.

pec = i2c\_readNak();

i2c\_stop();

// This converts high and low bytes together and processes temperature,

// MSB is a error bit and is ignored for temps.

tempFactor = 0.02; // 0.02 degrees per LSB (measurement

// resolution of the MLX90614).

tempData = 0x0000; // Zero out the data

frac; // Data past the decimal point

// This masks off the error bit of the high byte, then moves it left

// 8 bits and adds the low byte.

tempData = (double)(((data\_high & 0x007F) << 8) + data\_low);

tempData = (tempData \* tempFactor)-0.01;

float celcius2 = tempData - 273.15;

//Average the two temps together

float celcius = (celcius1 + celcius2)/2;

// Returns temperature in Celcius.

return celcius;

}

////////////////////////////////////////////////////////////////

//

// Read thermistor temp

//

int readThermistor(int pin)

{

for (int i=0; i< Num\_Samples; i++)

{

Samples[i] = analogRead(pin);

delay(10);

}

// average all the samples out

float average = 0;

for (int c=0; c< Num\_Samples; c++) {

average += Samples[c];

}

average /= Num\_Samples;

// convert the value to resistance

average = (1023 / average) - 1;

average = Series\_Resistor / average;

Steinhart = average / ThermistorNominal; // (R/Ro)

Steinhart = log(Steinhart); // ln(R/Ro)

Steinhart /= BCoefficient; // 1/B \* ln(R/Ro)

Steinhart += 1.0 / (TemperatureNominal + 273.15); // + (1/To)

Steinhart = 1.0 / Steinhart; // Invert

Steinhart -= 273.15; // convert to C

Steinhart \*= 9; // begin conversion to F

Steinhart /= 5;

Steinhart += 32; // finish conversion to F

return Steinhart;

}

//////////////////////////////////////////////////////////////////////

//

// Control Section

//

void lowtemp()

{

currentMillis = millis();

Serial.println("W,We have entered lowtemp");

if (BoxFahrenheit < SetTemp && AC==HIGH)

{

Serial.println("W,We have turned off the airconditioners!");

digitalWrite(Pin\_AC1, LOW);

digitalWrite(Pin\_AC2, LOW);

digitalWrite(Pin\_LED4, HIGH);

AC = LOW;

previousMillis = millis();

}

if (BoxFahrenheit > SetTemp)

{

if (currentMillis - previousMillis < ShortCycleDelay)

{Serial.println("W,Short Cycle Delay");}

else

{

Serial.println("W,We have turned on the airconditioners!");

digitalWrite(Pin\_AC1, HIGH);

digitalWrite(Pin\_AC2, HIGH);

digitalWrite(Pin\_LED4, LOW);

AC=HIGH;

}

}

}

void hitemp()

{

Serial.println("W,We have entered hitemp!");

if (BoxFahrenheit < SetTemp)

{

digitalWrite(Pin\_AC1, HIGH);

digitalWrite(Pin\_AC2, HIGH);

digitalWrite(Pin\_LED4, LOW);

previousMillis = currentMillis;

}

if (BoxFahrenheit > SetTemp)

{

digitalWrite(Pin\_AC1, LOW);

digitalWrite(Pin\_AC2, LOW);

digitalWrite(Pin\_LED4, HIGH);

}

}

void changemode(byte newmode)

{

EEPROM.write(EE\_mode, newmode);

if (newmode == 1)

{SetTemp = EEPROM.read(EE\_low);}

if (newmode == 0)

{SetTemp = EEPROM.read(EE\_hi);}

}