```
/*Nirav Patel and Faisal Sayed
ECE 4760 Final Project
with Prof Bruce
Research Project with Hencey Lab
with Prof. Hencey */

#include <avr/io.h>
#include <stdlib.h>
#include<avr/interrupt.h>
#include "lookup.h"
/*--------Init global Variable------*/
volatile unsigned char timer_value;
volatile int pulses_per_second,pulses_per_second_2,wait=5000;
int e,error, setpoint = 300,setpoint2=290, inlet_temp, air_temp,cond_out, peltier
unsigned charflag_hotloopPump=0,flag_peltierHeat=0;

volatile int adc_value[8];

static unsigned char old_pin_state=0, old_pin_state_2=0;
int adc_value0, output;
int             adc_value1;//Variable used to store the value read from the ADC
uint16_t     adc_value2;
char                    buffer[5];//Output of the itoa function
uint8_t                    i=0;//Variable for the for() loop11
void adc_init(void);            //Function to initialize/configure the ADC
uint16_t   read_adc(uint8_t   channel);//Function to read an arbitrary analogic
float LPM ;

const int tempAdc[] = {   129, 167, 215, 277, 314, 354, 397, 444, 493, 543, 595,
const int tempCx10[] = { 1000, 900, 800, 700, 650, 600, 550, 500, 450, 400, 350,

Table1d temperatureTable =
{
         18,/* Number of columns */
    tempAdc,/* Input data */
   tempCx10,/* Output data */
};

/*---------------PeltierPIcontroller-----------------------*/
struct PIControl
{
  int                 kp;/**< Proportional gain constant */
  int                 ki;/**< Integral gain constant */
  unsigned char   shift;/**< Right shift to divide */
  int max;              /**< Maximum value */
  int min;              /**< Minimum value */
  long               i;/**< Current integrator value */
};
int pi_control (struct PIControl *p,int e)
{
      bool   int_ok;/* Whether or not the integrator should update */
  long        new_i;/* Proposed new integrator value */
  long            u;/* Control output */

   /* Compute new integrator and the final control output. */
  new_i = p->i + e;
  u = (p->kp * (long)e + p->ki * new_i)>> p->shift;
int_ok = true;

  /* Positive saturation? */
  if (u > p->max)
  {u = p->max;  /* Clamp the output */
  if (e > 0)/* Error is the same sign? Inhibit integration. */
     {int_ok =false;
    }  }
  /* Repeat for negative sign */
  else if (u < p->min)
  {
    u = p->min;
    if (e < 0)
    {
       int_ok =false;
    }
  }
/* Update the integrator if allowed. */
  if (int_ok)
  {
    p->i = new_i;
  }
  return (int)u;
}

void pi_control_init (struct PIControl *p)
{
  p->i = 0L;
}

struct PIControl Peltier_pi =
{
    20,// kp gain
     1,//ki gain
     1,// right s hift to divide
```

```
      255,// max
      10,// min
};
void control_Peltier (void)
{
  e =(setpoint - inlet_temp);

  output = pi_control(&Peltier_pi, e);
          OCR2A      =      output;// output of PID controller to D11 on Arduin
}

/*-------------------FanPIcontrol----------------------------------*/

struct PI_Control
{
  int                 kp;/**< Proportional gain constant */
  int                 ki;/**< Integral gain constant */
  unsigned char   shift;/**< Right shift to divide */
  int max;               /**< Maximum value */
  int min;               /**< Minimum value */
  long                 i;/**< Current integrator value */
};

//pi control function
int pi_control (struct PI_Control *p,int e)
{
      bool   int_ok;/* Whether or not the integrator should update */
  long        new_i;/* Proposed new integrator value */
  long            u;/* Control output */

   /* Compute new integrator and the final control output. */
  new_i = p->i + e;
  u = (p->kp * (long)e + p->ki * new_i)>> p->shift;
int_ok = true;

  /* Positive saturation? */
  if (u > p->max)
  {u = p->max;  /* Clamp the output */
  if (error > 0)/* Error is the same sign? Inhibit integration. */
     {int_ok =false;
    }  }
  /* Repeat for negative sign */
  else if (u < p->min)
  {
    u = p->min;
    if (error < 0)
    {
      int_ok =false;
    }
  }
/* Update the integrator if allowed. */
  if (int_ok)
  {
    p->i = new_i;
  }
  return (int)u;
}

void pi_control_init (struct PI_Control *p)
{
  p->i = 0L;
}

//Control Varaibles for the PI structure
struct PIControl fan_pi =
{
    20,// kp gain
     1,//ki gain
     1,// right s hift to divide
      255,// max
      0,// min
};

//PI controller for the heat exchanger fans on D9
void control_fan (void)
{
  error =-(setpoint2 - cond_out);
  output = pi_control(&fan_pi, error);
      OCR1A   =   output;// output of PID controller to D9 on Arduino
}
/*----------------------------------------------*/

#define TEMP_FILTER_SHIFT 2
#define TEMP_FILTER_SIZE 4
inttemp_filter[TEMP_FILTER_SIZE];
int temp_filter_sum;
unsigned char temp_filter_idx;
volatile inttime1,pause=0,iterations;
volatile unsigned char flag=0, flag_transmit=0;
int main(void)
{
  unsigned char timer_saved, timer_copy;
```

```
int data, value=0, identifier;

Serial.begin (9600);
        adc_init();//Setup the ADC
 DDRD  |=  0x13;//0b10010  D2 & D4 on
  DDRB=0x30;//0b110000
// PWM set up using timer2
 DDRB  |=  0x0E;// PD5 is now an output pin  D9 on arduino
DDRD |= 0x88;

   OCR2A   =0;// D11 Connected to the peltier Heater
      OCR2B=255;//D3 Connected to the pump on the hot loop
 TCCR2A  |=  (1  <<  COM2A1)  |  (1<<COM2B1);// set non-inverting mode
 TCCR2A  |=  (1  <<  WGM21)  |  (1  <<  WGM20);// set fast PWM Mode
TCCR2B |= (1 << CS20)| (1 << CS22);
  OCR1A   =   128;// set D9 Connected to the heat exchanger fans
  OCR1B=0;//D10
 TCCR1A  |=  (1  <<  COM1A1)  |  (1<<COM1B1);// set non-inverting mode
 TCCR1A  |=  (1  <<  WGM12)  |  (0  <<  WGM11)  |  (1  <<  WGM10);// set 8 bit fast PWM
TCCR1B |= (1 << CS12);

// set 1ms interrupt using timer0
            TCCR0A        =        0XC2;//11000010  // Set the Timer Mode to
             OCR0A         =         124;// Set the value that you want to cou
                 TIMSK0           =0X02;// Enable match interrupts
            TCCR0B        =        0x03;// set prescaler to 64 8Mhz/64=7.815

   sei();

for(;;)
        {//Our infinite loop
  timer_copy = timer_value;
  PORTD |= 0x10;
  //check for commands from Matlab GUI
   if(Serial.available()>0)
   {
    identifier=Serial.read();
    switch(identifier)
     {
       case 'r':
       data_transmit();
       identifier=0;
       break;

       case 'w':
       update_setpoint();
```

```
          identifier=0;
           break;

           case 'g':
          update_gains();
          identifier=0;
           break;
         }

    }
      if (((unsigned)(timer_value - timer_saved) & 0x7F) >= 100)
      {
       timer_saved=timer_copy;
        //look up the temperature values from the table
        lookup1d(&temperatureTable, adc_value[3], &inlet_temp);
       lookup1d(&temperatureTable, adc_value[4], &cond_out);
       lookup1d(&temperatureTable, adc_value[5], &air_temp);
       lookup1d(&temperatureTable, adc_value[1], &peltier_temp);
       control_fan();
        //safety checks for peltier for overheat or hot loop pump not working
        if(peltier_temp<=900 && flag_hotloopPump==0 && flag_peltierHeat==0)
         control_Peltier();
        else
          {
          //turns off the peltier in case of overheat or pump failure
          //demands a manual reset for safety purpose
          OCR2A=0;
         flag_peltierHeat=1;
          }
      }
      //check if the flow is zero in the hot loop pump for atleast 5 seconds
      if(pulses_per_second==0 && wait==0)
      {
          //turn off the pump if no flow is detected for 5 seconds continously
          //demands a manual reset for safety concerns
         PORTD=(0<<PIN2);
         flag_hotloopPump=1;
      }
      else
        wait=5000;

      //sync the ADC reads with PWM on time to get instantaneous values of current
      if(TCNT1L>(OCR2A-5))
       {
        flag=0;
       PORTD=(0<<PIN7);
      }
      //transmits when demanded at user defined intervals
      if(flag_transmit==1 && iterations>0)
        {
        if(time1==0)
          {
          Serial.print("~");
          Serial.print(inlet_temp);
          Serial.print ("~");
          Serial.print ("~");
          Serial.print (cond_out);
          Serial.print ("~");
          Serial.print ("~");
          Serial.print(pulses_per_second,DEC);
          Serial.print ("~");
          Serial.print ("~");
          Serial.print(pulses_per_second_2,DEC);
          Serial.print ("~");
          Serial.print ("~");
          Serial.print(adc_value[6],DEC);
          Serial.print ("~");
          Serial.print ("~");
          Serial.print(adc_value[7],DEC);
          Serial.println ("~");
          if(iterations==1)
            flag_transmit=0;
          --iterations;
          time1=pause;
          }
        }
       //LPM=0.307*pulses_per_second;



        }//end for
  }    //end main

  ISR  (TIMER0_COMPA_vect)// timer0 Compare match interrupt
  {
    static int prescaler;
    static int pulse_count, pulse_count_2;
    unsigned char new_pin_state  = PINB  & 0x10;// Read PORT B pin 4 - flow sensor
    unsigned char new_pin_state_2 = PINB & 0x20;// Read PORT B pin 5 - flow sensor
    static unsigned char adcChannel = 0;

    //event to be executed every 1ms here
```

```
 ++timer_value;

 if (old_pin_state != new_pin_state)
 {
   if ((old_pin_state == 0) && (new_pin_state != 0))
   {
     ++pulse_count;
   }

   old_pin_state = new_pin_state;
 }
  if (old_pin_state_2 != new_pin_state_2)
 {
   if ((old_pin_state_2 == 0) && (new_pin_state_2 != 0))
   {
     ++pulse_count_2;
   }

   old_pin_state_2 = new_pin_state_2;
 }

 ++prescaler;
 if(time1>0) --time1;
 if(pulses_per_second==0) --wait;
 if (prescaler == 1000)//update the flow value each second
 {
   pulses_per_second = pulse_count;
  pulses_per_second_2=pulse_count_2;
   pulse_count=0;
   pulse_count_2=0;
   prescaler = 0;
  }

//read ADC when PWM is on to get instantaneous
if(TCNT1L>3 && TCNT1L<(OCR2A-5))
   {
     if (0 == (ADCSRA & (1 << ADSC)) && flag==0)
       {
            PORTD=(1<<PIN7);
       flag=1;
       adc_value[adcChannel] = ADCW;
       /* Go to the next channel.  Wrap past 8. */
       adcChannel = (adcChannel + 1) & 0x07;
       ADMUX = (ADMUX & 0xF0) | adcChannel;
       /* Start the next conversion */
       ADCSRA |= (1<<ADSC);
       PORTD=(0<<PIN7);
        }
    }
  if(TCNT1L>(OCR2A-5))
     {
       flag=0;
      PORTD=(0<<PIN7);
    }
}
//end the timer 0 compare match interrupt

//init the ADC
void adc_init(void)
{
             PRR        &=        ~0x01;/* Disable ADC power-down logic */
   ADCSRA  |=  ((1<<ADPS2)|(1<<ADPS1));//8Mhz/64 = 125Khz the ADC reference clo
  ADMUX = 0x40;/* Use 5V reference, start at channel 0 */
         ADCSRA     |=     (1<<ADEN);//Turn on ADC
         ADCSRA     |=     (1<<ADSC);//Do an initial conversion because this or
}
//end ADC init

//Data trasmit sets the variable for transmit operation
void data_transmit(void)
{
      iterations=getdata();//gets the iteration value
      pause=getdata();//gets the iteration interval
     time1=pause;
      flag_transmit=1;//sets the flag for transmit
}
//end data_transmit

void update_setpoint(void)
{
   setpoint=getdata();//gets the new setpoint
  Serial.println("OK");//acknowledges successful update
}
//end updatae_setpoint

void update_gains(void)
  {
  //get the new gain values
 Peltier_pi.kp=getdata();
 Peltier_pi.ki=getdata();
  //acknowledges successful update
  Serial.println("OK");
  }
//end update_gains

//gets data from UART - Terminating Character is Linefeed '\n
int getdata(void)
{
  int value;
  value=0;
  while(1)
    {
     int data=Serial.read();
     if(data=='\n')
        {
        return  value;
        break;
        }
     else if (data>47 && data<58)
        {
        data=data-0x30;
        value=value*10+data;
        }
    }

}  //end getdata()
```