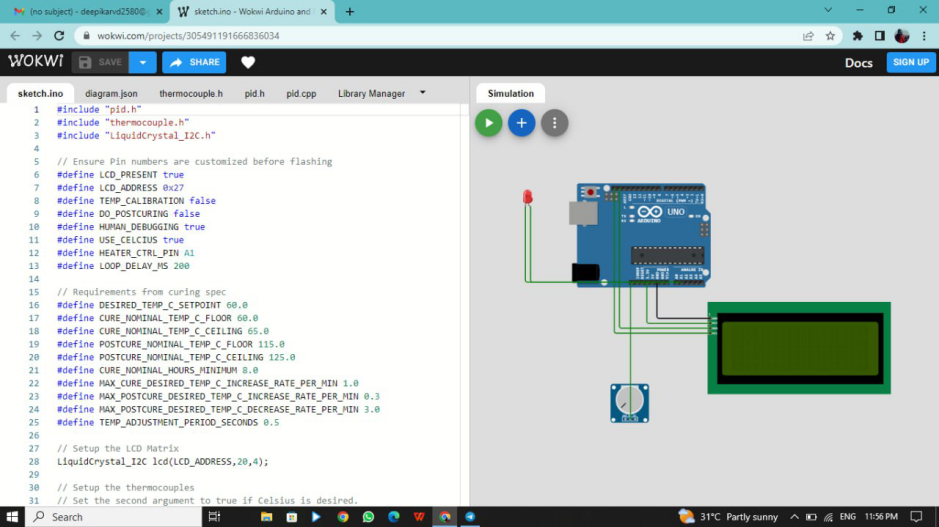
NAME : V.DEEPIKA

REG NO :732720121009

LINK :<https://wokwi.com/projects/305491191666836034>

SCREEN SHOT:

PROGRAM:

#include "pid.h"

#include "thermocouple.h"

#include "LiquidCrystal\_I2C.h"

// Ensure Pin numbers are customized before flashing

#define LCD\_PRESENT true

#define LCD\_ADDRESS 0x27

#define TEMP\_CALIBRATION false

#define DO\_POSTCURING false

#define HUMAN\_DEBUGGING true

#define USE\_CELCIUS true

#define HEATER\_CTRL\_PIN A1

#define LOOP\_DELAY\_MS 200

// Requirements from curing spec

#define DESIRED\_TEMP\_C\_SETPOINT 60.0

#define CURE\_NOMINAL\_TEMP\_C\_FLOOR 60.0

#define CURE\_NOMINAL\_TEMP\_C\_CEILING 65.0

#define POSTCURE\_NOMINAL\_TEMP\_C\_FLOOR 115.0

#define POSTCURE\_NOMINAL\_TEMP\_C\_CEILING 125.0

#define CURE\_NOMINAL\_HOURS\_MINIMUM 8.0

#define MAX\_CURE\_DESIRED\_TEMP\_C\_INCREASE\_RATE\_PER\_MIN 1.0

#define MAX\_POSTCURE\_DESIRED\_TEMP\_C\_INCREASE\_RATE\_PER\_MIN 0.3

#define MAX\_POSTCURE\_DESIRED\_TEMP\_C\_DECREASE\_RATE\_PER\_MIN 3.0

#define TEMP\_ADJUSTMENT\_PERIOD\_SECONDS 0.5

// Setup the LCD Matrix

LiquidCrystal\_I2C lcd(LCD\_ADDRESS,20,4);

// Setup the thermocouples

// Set the second argument to true if Celsius is desired.

#define THERMOCOUPLE\_COUNT 5

Thermocouple t1(A0, USE\_CELCIUS, 0.5);

Thermocouple t2(A0, USE\_CELCIUS, 0.5);

Thermocouple t3(A0, USE\_CELCIUS, 0.5);

Thermocouple t4(A0, USE\_CELCIUS, 0.5);

Thermocouple t5(A0, USE\_CELCIUS, 0.5);

Thermocouple thermocouples[THERMOCOUPLE\_COUNT] = {t1, t2, t3, t4, t5};

// PID object params

double dt = 0.1; // loop interval time

double max\_out = 1; // maximum allowable output from pid

double min\_out = -1; // minimum allowable output from pid

double Kp = 0.01; // proportional gain

double Kd = 0.01; // derivative gain

double Ki = 0.5; // integral gain

// Create pid object with params

PID pid = PID(dt, max\_out, min\_out, Kp, Kd, Ki); // Not used at the moment.

// Variables for test/debug

double test\_setpoint = 60;

// Varibles for the oven controller

float prev\_temp;

unsigned long prev\_meas\_timestamp = 0;

unsigned long last\_heat\_adjustment\_timestamp = 0;

unsigned long setpoint\_reached\_timestamp = 0;

unsigned long last\_lcd\_update\_timestamp = 0;

bool update\_setpoint\_timestamp;

bool heater\_on;

long votes\_for\_heat = 0;

double toFahrenheit(double celcius) {

return celcius \* 1.8 + 32.0;

}

double toCelcius(double fahrenheit) {

return (fahrenheit - 32.0) / 1.8;

}

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

Serial.print("Serial started.\n");

if (LCD\_PRESENT) {

// Initialize LCD, wait for 5 sec

Serial.print("Initializing LCD.\n");

lcd.init();

lcd.backlight();

lcd.clear();

}

heater\_on = false;

prev\_temp = 0.0;

prev\_meas\_timestamp = millis();

update\_setpoint\_timestamp = true;

last\_heat\_adjustment\_timestamp = millis();

}

void loop() {

if (TEMP\_CALIBRATION) {

// Log the temperature to serial output

// We'll save this to an output file on a companion computer

for (int i=0;i<THERMOCOUPLE\_COUNT;i++) {

Serial.print(thermocouples[i].read());

Serial.print(USE\_CELCIUS ? "C": "F");

Serial.print(", ");

}

Serial.print("\n");

// Delay the loop for human readable debugging

if (HUMAN\_DEBUGGING) {

delay(300);

}

}

// Measure average temperature in the oven

float current\_temp = 0.0;

for (int i=0;i<THERMOCOUPLE\_COUNT;i++) {

current\_temp += thermocouples[i].read();

}

current\_temp /= THERMOCOUPLE\_COUNT;

if (!USE\_CELCIUS) {

current\_temp = toCelcius(current\_temp);

}

// Record current timestamp

unsigned long current\_meas\_timestamp = millis();

if (LCD\_PRESENT && (current\_meas\_timestamp - last\_lcd\_update\_timestamp > 1000)) {

// Refresh the LCD screen

lcd.clear(); // Clear the screen

lcd.setCursor(0,0);

lcd.print("Temp: ");

lcd.print(USE\_CELCIUS ? current\_temp : toFahrenheit(current\_temp));

lcd.print(USE\_CELCIUS ? "C.": "F.");

lcd.setCursor(0,1);

lcd.print("Time left (min): ");

unsigned long time\_left\_min = (CURE\_NOMINAL\_HOURS\_MINIMUM \* 1000 \* 3600 - (millis() - setpoint\_reached\_timestamp)) / (60.0 \* 1000);

lcd.print(time\_left\_min);

lcd.setCursor(0,2);

lcd.print("Time on (min): ");

unsigned long time\_on\_min = millis() / (60.0 \* 1000);

lcd.print(time\_on\_min);

lcd.setCursor(0,3);

lcd.print("Heater is ");

lcd.print(heater\_on ? "ON" : "OFF");

last\_lcd\_update\_timestamp = current\_meas\_timestamp;

}

// Stop the heater after the desired curing time, and let the oven cool down

if (!DO\_POSTCURING && (millis() - setpoint\_reached\_timestamp) > CURE\_NOMINAL\_HOURS\_MINIMUM \* 1000 \* 3600) {

return;

}

// Limit heat control to the specified adjustment period

if (millis() - last\_heat\_adjustment\_timestamp >= TEMP\_ADJUSTMENT\_PERIOD\_SECONDS \* 1000) {

if (votes\_for\_heat > 0) {

analogWrite(HEATER\_CTRL\_PIN, 255);

heater\_on = true;

}

else {

analogWrite(HEATER\_CTRL\_PIN, 0);

heater\_on = false;

}

// Reset the timer var and votes

last\_heat\_adjustment\_timestamp = millis();

votes\_for\_heat = 0;

}

// Set the timestamp of when the oven first reaches the desired setpoint

if (current\_temp >= DESIRED\_TEMP\_C\_SETPOINT && update\_setpoint\_timestamp) {

setpoint\_reached\_timestamp = millis();

update\_setpoint\_timestamp = false;

}

else if (update\_setpoint\_timestamp) {

setpoint\_reached\_timestamp = millis();

}

// Calculate this interval's control output

// Not currently used

double control\_output = pid.calculate(test\_setpoint, current\_temp);

// Controller Logic

// Evaluate the rate of change in temperature between now and the previous measurement

double temperature\_change\_rate = (current\_temp - prev\_temp) / (1.0 \* (current\_meas\_timestamp/100.0 - prev\_meas\_timestamp/100.0));

if (!DO\_POSTCURING) {

// Initial curing mode

if (temperature\_change\_rate <= (MAX\_CURE\_DESIRED\_TEMP\_C\_INCREASE\_RATE\_PER\_MIN / (60.0 \* 10)) &&

current\_temp < (CURE\_NOMINAL\_TEMP\_C\_CEILING + CURE\_NOMINAL\_TEMP\_C\_FLOOR)/2.0) {

// Heat up vote

votes\_for\_heat++;

// Human readable output

if (HUMAN\_DEBUGGING) {

Serial.print("Vote for heat++\n");

}

}

else {

// Heat down vote

votes\_for\_heat--;

// Human readable output

if (HUMAN\_DEBUGGING) {

Serial.print("Vote for heat--\n");

}

}

}

else {

// Postcuring mode

if (temperature\_change\_rate <= (MAX\_POSTCURE\_DESIRED\_TEMP\_C\_INCREASE\_RATE\_PER\_MIN / (60.0 \* 10))) {

// Heat up votes

votes\_for\_heat++;

}

else {

votes\_for\_heat--;

}

}

prev\_temp = current\_temp;

prev\_meas\_timestamp = current\_meas\_timestamp;

delay(LOOP\_DELAY\_MS);

}