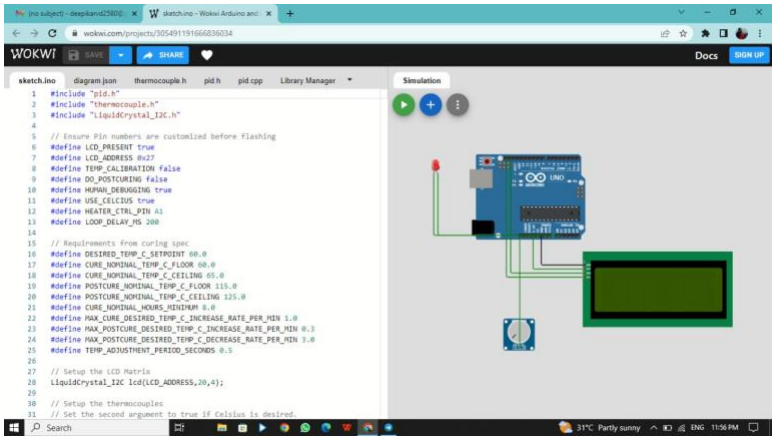
NAME:V. DEEPIKA

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;

// Variables 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);
}
```