Electronic device with conditionally autonomous heat and humidity regulation for proofing bread and pastry

David Omanovic, Denis Lobe

June 10, 2024

Department of Natural Sciences, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway

Abstract

We describe an electronic system housed in a polypropylene box, equipped with an incandescent light bulb to create a warm environment necessary for proofing bread and pastry. The system utilizes a micro-controller unit, specifically the *Arduino UNO*, in conjunction with a relay system to manage both digital and analog signals, facilitate user interaction, and operate within a voltage range of 5V to 230V (50Hz AC). The autonomous systems of the device is implemented with C++ scripting, sensor data, communicated through an *Arduino UNO*. However, the mechanical design, use of materials, and the use of a light bulb significantly limits efficiency and results in energy losses, rendering the product less sustainable.

1 Introduction

To make bread you need a yeast and bacteria culture that can make your bread rise and ferment. In normal conditions like room temperature in your kitchen and low humidity, the potential rise and development of flavour is inconsistent. A way of ensuring consistent fermentation is crucial to get consistent results. The fermentation process is also quite tedious and slow in normal conditions. For that reason, a humid, warm and controlled environment like in a "proofing chamber", will accelerate the process. The largest volume increase in dough is anything around six hours [1].

A somewhat of a indicative technique to distinguish over/underproofing is by looking at the crumb structure. Using a proofing chamber, we wish to land on the middle row of the nicely fermented crumbs as we can see in figure 1 (most preferably the last one to the right). With a consistent temperature, you will minimize the chances for fermentation issues. Rules of thumb, like checking volume rise, springiness in the dough or just following a time-based recipe will not suffice if the environment is constantly changing. A constant temperature is achievable through a closed thermal system that regulates the heat. This project utilizes a plastic box and a light bulb as heat.

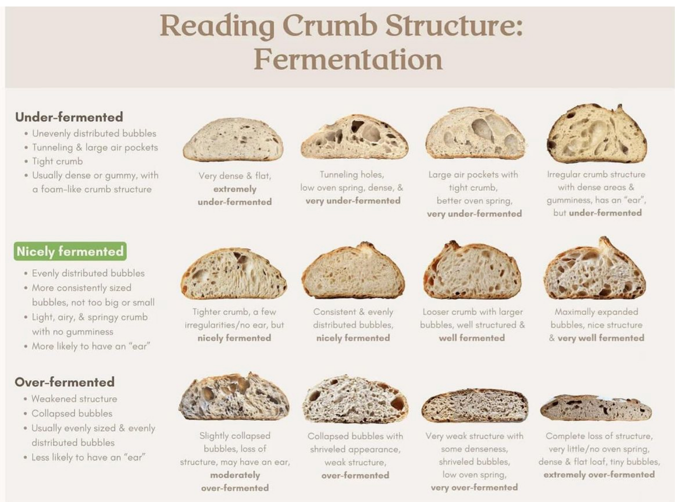

Figure 1: Different types of crumb structures specifically for sourdough loaf and how to read them [2].

2 Method

2.1 Mechanical build

A polypropylene box (storage box from Clas Ohlson) with dimensions of $59 \times 39,5 \times 24,5$ cm, as seen in figure 2, is the outer body/chassis of the proofing chamber where the dough will reside. The electronics is hidden underneath a black 3D printed box that is on top of the polypropylene lid (c.f. fig. 2). Holes are drilled in the black box for the buttons and display, with male-to-female connectors for each pin of the components drilled into the roof. The polypropylene lid also has a hole to insert the E27 light bulb socket, with belonging cable to power the heating element. The cable for the wall outlet plug comes through a hole at the back of the black box. With our $60W$ incandescent light bulb, we would reach our desired temperature in about **5 minutes** if we want an increment of $5°C$, for instance from room temperature $22°C$ to $27°C$ by using simple thermal equilibrium approximations and calculations with heat capacities. In this case, we use the fact that $q = mC_s\Delta T$, where q is the heat added, C_s is the specific heat capacity of the material, and ΔT is the temperature change [3]. For a clean and user-friendly design, we use a battery inside as a power source with a red latching switch at the top left corner (c.f. fig. 2).

Figure 2: Build of the box as a polypropylene box for the dough to rest inside and 3D printed black box at the top for the electronics. At the top of the black box is the user interface, OLED display, buttons and power switch. Blue LED indicates settings menu, Green LED indicates that program is running.

2.2 Software

The software part is easily implemented with some logic programming in Arduino C. You start off at a menu screen, you scroll through options and pick your desired temperature and desired time for the program to run. For the electric circuit, we are controlling our heat and humidity by first setting input variables for desired temperature, humidity and time. After starting the program, the output temperature and humidity will be displayed on the LCD and the time will start a countdown. If we reach undesirable values, the heating elements turns off to release heat and turns on again as we reach a lower temperature. In this way, we should approximately stay at thermal equilibrium at the user input temperature. As the time is counted down to zero, the whole programs turns off and the user gets sent back to the main menu. The entirity of the code for the Arduino is given in Appendix A at page 8-12, and the methods for controlling pins and much more are found in "Arduino Projects" [4]. Respectively, we have libraries and manuals for the DHT11 sensor and SSD1306 display from AdaFruit, which you can download from Arduino IDE directly.

2.3 Electrical connections

The device works as a simple feedback system using the Arduino UNO as the control operator and the temperature/humidity sensor as the input device. Additionally, the LED lights and screen functions as a human guide for the operation of this device in the form of a user interface (UI).

Table 1: Electrical components for proofing chamber with recommended voltages and currects. Note that the circuit consists of AC and DC sources.

Component	V_{in}/V_{out} (recommended)
Arduino UNO	7-12 V DC
Adafruit SSD1306 - I2C OLED 0.96" 128 × 64	3-5 V DC
DHT11 - temp & humid sensor	3-5.5 V DC
60W OSRAM Incandescent bulb	230V AC (50Hz)
Procell - Alkaline Battery	9 V DC
Relay Module JQC-3FF-S-Z	5V/230V AC (50Hz)
Restored E27 light bulb socket	230V AC (50 Hz)
2x 74HC595 Shift Register	2-6V DC
SMD RGB LED	1.8-3V DC
RGB LED Bar Graph	1.8-3V DC

Table 2: Pin connections for components to the Arduino UNO. **'D'** is the digital channels, which are either input or output pinmodes on the Arduino. **'A'** indicates the analog channels.

Component	Pinout	Arduino Pin
SSD1306 Display	V_{in}	5V
SSD1306 Display	GND	GND
SSD1306 Display	SCL	**A1**
SSD1306 Display	SDA	**A2**
Tactile switch (+)	V_{in}, GND	**D2**, GND
Tactile switch (-)	V_{in}, GND	**D3**, GND
Tactile switch (OK)	V_{in}, GND	**D4**, GND
Relay module	IN	**D5**
Relay module	V_{cc}	5V
Relay module	GND	GND
DHT11 sensor	Data	**D6**
DHT11 sensor	V_{cc}	5V
DHT11 sensor	GND	GND
Red - RGB LED + 470Ω	Anode	**D7**
Green - RGB LED + 470Ω	Anode	**D8**
Blue - RGB LED + 470Ω	Anode	**D9**
GND - RGB LED	Cathode	GND
74HC595	DATA	**D10**
74HC595	LATCH	**D11**
74HC595	CLOCK	**D12**
9V Battery	Cathode	V_{in}
9V Battery	Anode	GND

Table 3: Pin connections for components to components.

From	Pinout from	To	Pinout to
E27 Socket	V_{supply}	Relay	COM
E27 Socket	Neutral (N)	Relay	NO
9V Battery	Cathode (+)	Latching switch	Arduino V_{in}
9V Battery	Anode (-)	Arduino UNO	GND

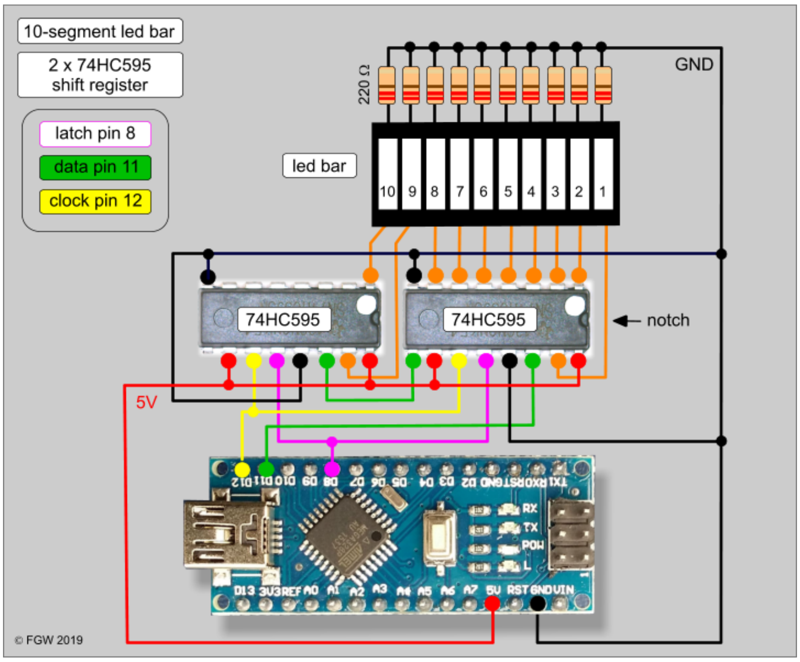

Figure 3: Shift register wiring diagram retrieved from [5]. Two 74HC595, 10x 220Ω resistors and the 10-segment led bar connected to an MCU like Arduino UNO. Latch pin 8 updates the LED states by transferring data from the shift register to the output pins upon receiving a high signal. Data pin 11 receives the serial data that determines the on/off states of the LEDs, one bit per LED. Clock pin (SRCLK) advances the shift register by one position with each pulse, allowing sequential data entry from data pin.

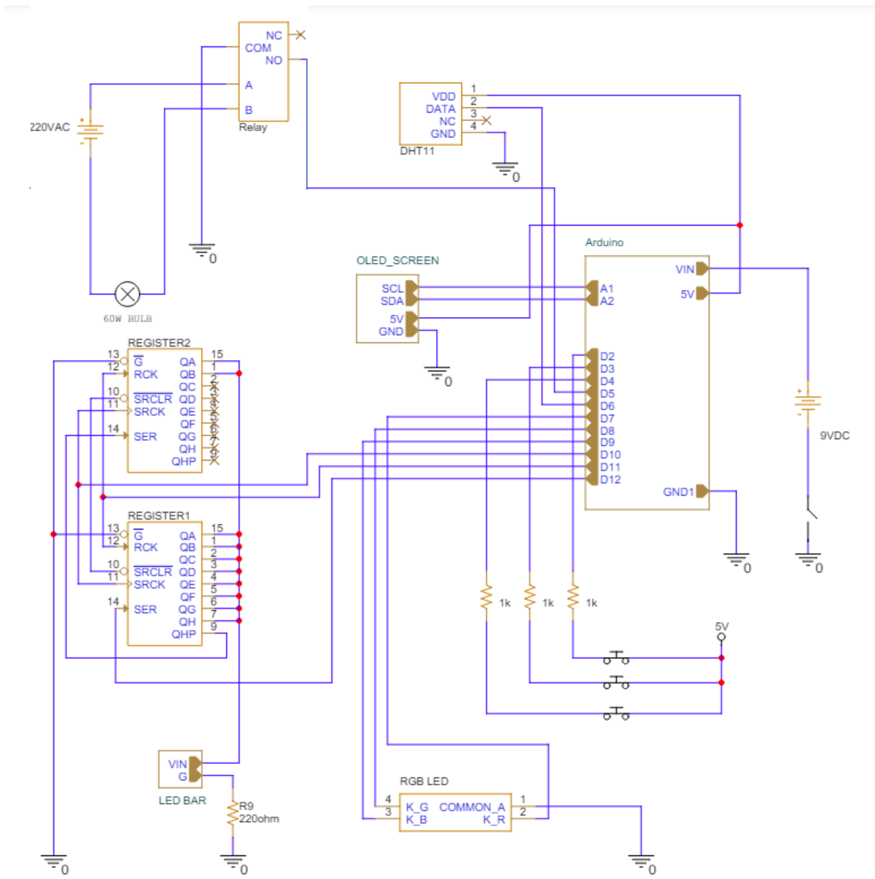

Figure 4: Circuit schematics of figure 5 and tables 2, 1, 3 constructed in *OrCAD™* software.

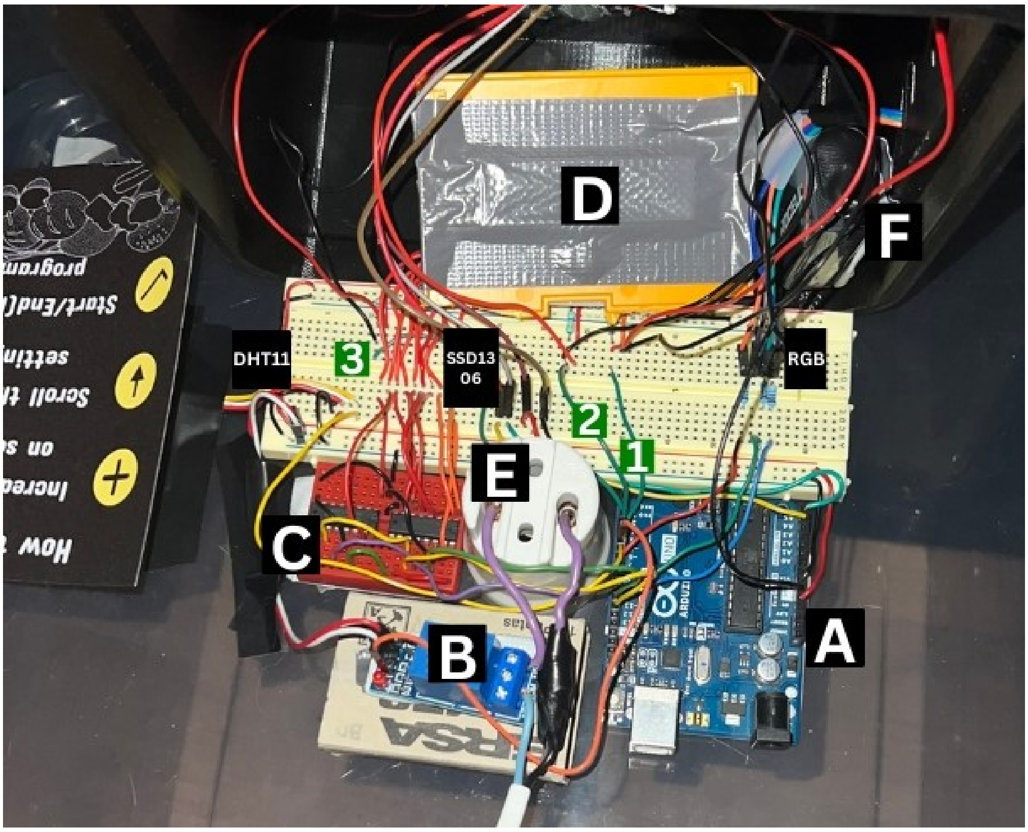

Figure 5: **(A)** Arduino UNO pinout connections as in table 2. **(B)** Relay Module listed in table 1 connected as in table 3. **(C)** 8-bit shift register using two 74HC595 to control the 10-segment led bar for heat indication, software logic in Appendix A, p.12, line 292-337. **(D)** The 10-segment led bar connected with 10x 220Ω resistors to cathode legs, and rest connects to **(C)** as schematics in figure 3. **(E)** E27 bulb socket connected to the relay module, screwed into the box and plugged into wall outlet of 230V. **(F)** 9V battery plugged into a switch at the top of the lid so user can manually turn off the device without plugging out of the wall, anode goes to Arduino V_{in} and cathode to GND. **(1)** OK button, tactile switch with logic in Appendix A, p. 11, line 238-246. **(2)** Scroll button, tactile switch with logic in Appendix A, p.11, line 248-256. **(3)** Increment/Plus button, tactile switch with logic in Appendix A, p.11, line 260-268. **DHT11** is the temperature and humidity sensor, **SSD1306** is the OLED display, and **RGB** is the RGB Micro LED component for program indication. These three components are connected in accordance to table 1.

3 Results and discussion

The price of the device lands on about 300,- NOK with an Arduino UNO, all electrical components and the plastic boxes. This is good news since similar products at the market costs up to 2500,- NOK. The device does indeed use about 5 minutes to heat up from $22°C$ to $27°C$ as stated in the method. We have tested, and it can easily go up to higher temperatures like $32°C$. The regulation technique for the temperature works fairly well as simply as turning the bulb on and off by closing the relay. From sensor readings, the environment will reach a thermal equilibrium as the temperature reaches what the user has picked as input. It could be a bit annoying that the light turns on and off every minute. Take for instance that the user picked a temperature of $27°C$. The code is constructed such that if the system hits $27.0°C$, it turns off and then on again immediately as the DHT11 registers 26.9°, which happens frequently in this case. An easy fix is implementing a threshold of $\pm 0.5°C$ in the software, letting it heat up over the input temperature, and cool down more. Despite the good heating capabilities, a poor insulator like our very thin chassis build in polypropylene plastic suffers loss of heat. The budget severely affects its efficiency. The box is easily heated up, but does waste a lot of energy in the long run, which is not sustainable. On the other hand, when it comes to the finished product, it is apparent that it makes a difference in bread and pastry proofing. Comparing bread without using the bread proofer and after using the bread proofer, shows a noticable difference in the quality of the fermentation as seen in figure (6).

Figure 6: To the left **(Before)** a sourdough loaf is baked without proofing and fermentation in the environments of the box. **(Before)** was proofing at the kitchen bench at approximately $22°C$ for 8 hours and 12 minutes. To the right **(After)**, the loaf proofed constantly at $27°C$ for 5 hours and 25 minutes. It is apparent that it takes almost half the time, and also ferments the dough in an even and controlled way, resulting in an uniform crumb structure. It subjectively tastes better with better fermentation as well.

Another issue with the design of the product is that all the electrical components are on the lid/roof of the polypropylene box (c.f fig. 5), essentially right next to the heating element. The second issue that arises by being on the lid of the box, is that the light bulb is also placed on top of the lid. In this sense, the breadboard and arduino also receive heat from the bulb, which can damage and wear out the electrical components. Even though the operating temperature is up to $85°C$ for the Arduino UNO, as specified by the official Arduino support page [6], it is not a safe environment to keep the rest of the components and for user. For that reason, it is not recommended to use the program for more than a couple of hours. Luckily, with higher temperatures it is not neccessary to proof for longer than four to five hours, whereas it would otherwise take anything between eight to 12 hours at the kitchen bench [1]. Another design flaw is that most components are taped down with duct tape for modularity purposes, but poses a risk of loosening.

A design improvement could be to have a slim heating element instead of a light bulb that is isolated at the bottom of the box, and have the electronics in a small seperate box that is glued to the front of the proofing box. There was a lack of resources to make this mechanical work less demanding, so the only viable option was to use a 3D printed plastic box at the top of the lid with cables hanging down from it. Another nice additional functionality would be some sort of speaker that plays a noise as the program is done, or if its too hot.

The program cycle works as promised and is user friendly. A demonstration of how the heating works is illustrated in figure 7. The led bar graph solution is slightly unstable as some cables can jump out of the breadboard caused from its bad positioning inside the black box. The user has to be careful when lifting the lid, which can be annoying. However, there seems to be no electromagnetic interference despite all of the connections, indicating good isolation. Most of the issues arise by the fact that the positioning and placements of the electronics are subpar and the lack of space inside the black box. More functionalities, slimmer design and less fuss with wires could be improved by making a PCB or having a 3D printed box with slots where you could easily insert the electronics.

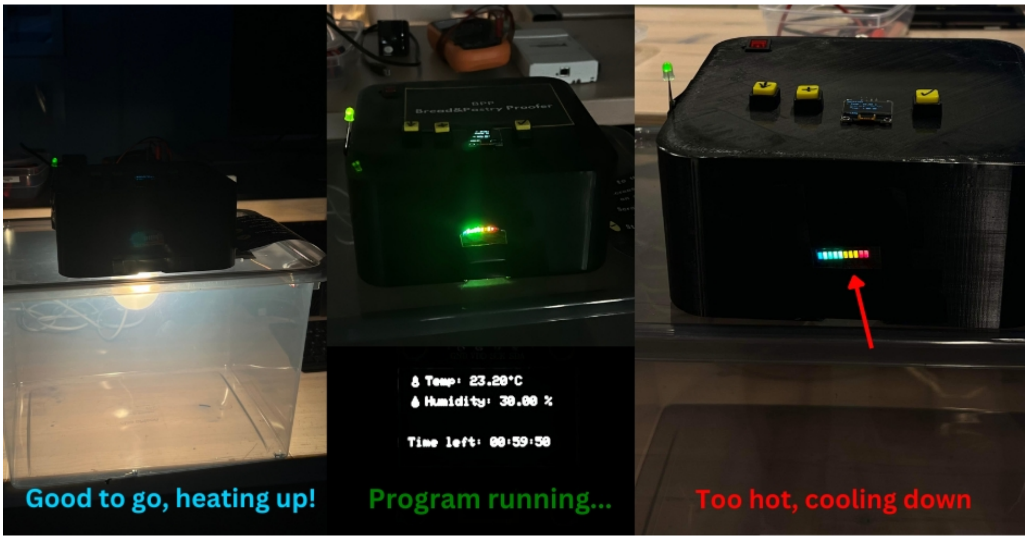

Figure 7: Example of program running where user has input a certain temperature to heat up to. To the left, user has input values on the settings page, pressed the "OK" button and then the light bulb turns on to heat the box. The middle shows an example of the program running and the OLED display indicating the sensor readings and time left for the program. The led bar segment to the right shows that it is too hot and turns off the heating element until the sensor registers that it is under the user input temperature.

4 Conclusion

Our conclusion is clear and the Bread proofer is far away for an ideal and efficient baking instrument. However, through trial and error, the project has given valuable information regarding how a micro controller with a simple feedback control system can be applied in a practical environment. For future applications one can apply a more suitable insulation box, or a mechanism that regulates the humidity in a better fashion. Therefore, this experiment can be used as a good example for future uses and applications, and is a good fundamental basis to heating systems. A good outcome is that a typical proofer costs up to 2500,- NOK, but this proofer landed at a tenth of the price.

References

[1] K. S. Aplevicz, P. J. Ogliari, E. S. Sant'Anna, Influence of fermentation time on characteristics of sourdough bread, Brazilian Journal of Pharmaceutical Sciences 49(2):233-239, (2013).

[2] Sourdough Wiki, Reading Crumb, Reddit, Available: r/Sourdough, Accessed: March 20th, 2024, (2024)

[3] H.D. Young, R.A. Freedman, *University Physics*, (Pearson, 14th ed., 2015).

[4] S. Fitzgerald, M. Shiloh, The Arduino Projects book, in *Arduino UNO starter kit*, (Torino, Italy, 2012).

[5] F. Wouterlood, Two ways to control a ten-segment led bar with an Arduino, thesolaruniverse.wordpress.com, (2019).

[6] Arduino Support, What is the operating temperature range for Arduino boards?, Available: support.arduino.cc, Accessed: April 8th 2024, (2024)

Note: ***Appendix A*** *containing all Arduino code is located at the next following pages.*

Appendix A. Software code

Below the entire Arduino code is given. Note that to implement this, you have to use exactly the same pins as specified from lines 1-24, or change them accordingly. The rest are functionalities.

```
1 #include <DHT_U.h> // Temperature/Humidity sensor Library
2 #include <Wire.h> // I2C Communications library
3 #include <Adafruit_GFX.h> // OLED Display library
4 #include <Adafruit_SSD1306.h>  // OLED Display library
5
6 // ARDUINO DIGITAL PINOUT
7 #define OLED_RESET -1 // Reset pin
8 #define BUTTON_PIN 2  // OK button
9 #define INCREMENT_PIN 3 // Plus button
10 #define SCROLL_PIN 4 // Scroll button
11 #define HEAT_PIN 5 // Relay
12 #define DHTPIN 6    // DHT sensor
13 #define RED_RGB 7 // Red RGB pin
14 #define GREEN_RGB 8 // Green RGB pin
15 #define BLUE_RGB 9 // Blue RGB pin
16 #define LATCH_PIN 10 // 74HC595 Latch
17 #define DATA_PIN 11 // 74HC595 Data
18 #define CLOCK_PIN 12 // 74HC595 Clock
19
20 // OLED CONFIGURATION
21 #define SCREEN_WIDTH 128 // OLED display width
22 #define SCREEN_HEIGHT 64 // OLED display height
23 #define DHTTYPE DHT11    // DHT 11
24 DHT dht(DHTPIN, DHTTYPE);
25
26 //  PROGRAM CONFIGURATION
27 float T_c = 22; // degrees celsius
28 float H_c = 40; // humidity in percent
29 int Time_c = 60;
30 int currentSelection = 0; // Display
31 int heatOn = LOW;
32 boolean registers[10]; // heatbar pins
33
34 // OUTPUT STATES
35 int displayState = 0; // (0 = Menu, 1 = Program)
36 int heatState = LOW;
37 int lastButtonState;
38 int currentButtonState;
39 int currentScroll;
40 int lastScroll;
41 int currentIncrement;
42 int lastIncrement;
43
44 // Small arrow bitmap (8x8)
45 static const unsigned char PROGMEM arrowIcon[] = {
46   0b00100, 0b01110,
47   0b11111, 0b00100,
48   0b00100, 0b00100,
49   0b00100, 0b00100,
50 };
51
52 // Temperature Icon bitmap (8x8)
53 static const unsigned char PROGMEM tempIcon[] =
54 {
55   0b00100, 0b01010,
56   0b01010, 0b01010,
57   0b01010, 0b10001,
58   0b10001, 0b01110,
59 };
60
61 // Water Droplet Icon bitmap (8x8)
62 static const unsigned char PROGMEM waterIcon[] =
63 {
64   0b00100, 0b00100,
65   0b01110, 0b01110,
66   0b11111, 0b11111,
67   0b11111, 0b01110,
68 };
```

```
// MISC
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
unsigned long previousMillis = 0; // stores time
long initialTime = 10; // Countdown time in s

// RUN CODE ON STARTUP
void setup() {
  // LCD SCREEN SETUP
  Serial.begin(9600);
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
  }

  display.display();

  delay(2000); // Pause for 2 seconds

  display.clearDisplay();

  // START TEMP&HUMIDITY SENSOR
  dht.begin();

  // SET DIGITAL PIN MODES AS I/O ARDUINO UNO
  // ===========================================
  pinMode(BUTTON_PIN, INPUT);   // DIGITAL PIN 2
  pinMode(SCROLL_PIN, INPUT); // DIGITAL PIN 3
  pinMode(INCREMENT_PIN, INPUT); // DIGITAL PIN 4
  pinMode(HEAT_PIN, OUTPUT); // DIGITAL PIN 5
  pinMode(RED_RGB,  OUTPUT); // DIGITAL PIN 7
  pinMode(GREEN_RGB, OUTPUT); // DIGITAL PIN 8
  pinMode(BLUE_RGB, OUTPUT); // DIGITAL PIN 9
  pinMode(DATA_PIN, OUTPUT); // DIGITAL PIN 10
  pinMode(LATCH_PIN, OUTPUT); // DIGITAL PIN 11
  pinMode(CLOCK_PIN, OUTPUT); // DIGITAL PIN 12
  // ===========================================
  currentButtonState = digitalRead(BUTTON_PIN);
  displayMenu();
}

// ____ FUNCTION SECTION ____

// CONTROL "MENU" DISPLAY STATE (HELPER FUNCTION)

void displayMenu(){

  // Display design
  display.clearDisplay();
  display.setTextSize(1);      // Double the normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0,0);
  display.println(F("Settings"));

  // Temperature
  display.setCursor(0,16);
  if (currentSelection == 0) display.print("> "); // Arrow for selection
  else display.print("  ");
  display.println("Temp: " + String(T_c) + " C");

  // Print Humidity on display
  display.setCursor(0,32);
  if (currentSelection == 1) display.print("> "); // Arrow for selection
  else display.print("  ");
  display.println("Humidity: " + String(H_c) + "%");

  // Print Time on display
  display.setCursor(0,48);
  if (currentSelection == 2) display.print("> "); // Arrow for selection
  else display.print("  ");
  display.println("Time: " + String(Time_c) + ":00:00");
}
//
```

```
// VALUE CONTROLLER FOR INCREMENT BUTTON
void increment() {
  switch (currentSelection) {
    case 0: // Temperature
      T_c += 1;
      if (T_c > 32) T_c = 22;
      break;
    case 1: // Humidity
      H_c += 10; // Increment by 10 for example
      if (H_c > 80) H_c = 40;
      break;
    case 2: // Time
      Time_c += 1; // Increment by 1 hour for example
      if (Time_c > 12) Time_c = 1; // Reset to 1 if it exceeds 12
      // Convert Time_c to seconds and update initialTime
      initialTime = Time_c * 3600;
      break;
  }
}

// DISPLAY DHT11 SENSOR READINGS
void displaySensorReadings()
{
  display.clearDisplay();
  display.setTextSize(1);      // Double the normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Display Temperature Icon
  display.drawBitmap(0, 0, tempIcon, 8, 8, SSD1306_WHITE);
  display.setCursor(14,0);     // Start text right after the icon
  display.print(F("Temp: "));
  display.print(t);
  int x = display.getCursorX() + 2; // Adjust X as needed
  int y = display.getCursorY() + 1; // Adjust Y to align with the top of the text
  int radius = 1; // Small radius for the degrees circle
  display.drawCircle(x, y, radius, SSD1306_WHITE);

  // Move the cursor to the right of the circle before printing "C"
  display.setCursor(x + 4, 0); // Adjust X to leave space after the circle
  display.println(F("C"));

  // Display Humidity Icon
  display.drawBitmap(0, 16, waterIcon, 8, 8, SSD1306_WHITE);
  display.setCursor(14,16); // Start text right after the icon
  display.print(F("Humidity: "));
  display.print(h);
  display.println(F(" %"));
}

// TIMER COUNTDOWN FOR DISPLAY
void displayCountdown() {
    if (initialTime > 0) { // Calculate time
        initialTime--;
        int hours = initialTime / 3600;
        int minutes = (initialTime % 3600) / 60;
        int seconds = initialTime % 60;

        // Display logic
        display.setTextSize(1);
        display.setCursor(0, 42);
        display.print(F("Time left: "));
        if(hours < 10) display.print('0');
        display.print(hours);
        display.print(':');
        if(minutes < 10) display.print('0');
        display.print(minutes);
        display.print(':');
        if(seconds < 10) display.print('0');
        display.print(seconds);
        delay(1000);
    } else {
```

```
        // When time is 0
        display.setTextSize(1);
        display.setCursor(0, 42);
        display.print(F("                    "));
        display.setCursor(0, 42);
        display.print(F("Program over!"));
        delay(200);
        displayState = 0; // Return to main menu
    }
}

// DISPLAY UPDATER (MAIN LOOP FUNCTION)
void updateDisplay()
{
    display.clearDisplay();
    if (displayState == 0) {
        RGB(0,0,255); // Blue at menu
        displayMenu();
        display.display();
    } else {
        RGB(0,255,0); // Green during program
        displaySensorReadings();
        displayCountdown();
        display.display();
    }
}

// BUTTON CONTROLLER (MAIN LOOP FUNCTION)
void buttons(){
    // CONFIRM BUTTON //
    lastButtonState = currentButtonState;
    currentButtonState = digitalRead(BUTTON_PIN);
    if (lastButtonState == HIGH && currentButtonState == LOW){
        // Toggle displayState between menu (0) and sensor readings (1)
        displayState ^= 1;
        heatState = !heatState;
        digitalWrite(HEAT_PIN, heatState);
    }

    // SCROLL MENU OPTIONS BUTTON //
    int scrollReading = digitalRead(SCROLL_PIN);
    if (scrollReading == LOW && lastScroll == HIGH) {
        // Add a small delay for button debounce
        delay(50);
        if (displayState == 0) {
            currentSelection =
            (currentSelection + 1) % 3;
        }
    }

    lastScroll = scrollReading;
    // INCREMENT BUTTON //
    int currentIncrementReading = digitalRead(INCREMENT_PIN);
    if (lastIncrement == HIGH && currentIncrementReading == LOW){
        // Add a small delay for button debounce
        delay(50);
        increment();
    }
    lastIncrement = currentIncrementReading;
}
// HEAT ELEMENT CONTROLLER (MAIN LOOP FUNCTION)
void heatControl(){
  if(dht.readTemperature() >= T_c && displayState == 1){
      digitalWrite(HEAT_PIN, LOW); // Set relay pin to low = turn off heat
    }
    else if (displayState == 1) // if you are inside the program, turn on heat
    {
      digitalWrite(HEAT_PIN, HIGH);
    }
    else if (displayState == 0){ // if you are in settings menu, no heat
      digitalWrite(HEAT_PIN, LOW);
    }
}
```

```

// RGB CONTROLLER (MAIN LOOP FUNCTION)
void RGB(int r,int g,int b){
   digitalWrite(RED_RGB, r & 0x04 ? HIGH : LOW);
   digitalWrite(GREEN_RGB, g & 0x02? HIGH : LOW);
   digitalWrite(BLUE_RGB, b & 0x01 ? HIGH : LOW);
}

// Control the shift register to manipulate the led bar graph showing temperature
void writereg(){
  digitalWrite(LATCH_PIN, LOW);
  for (int i = 9; i >= 0; i--){
    digitalWrite(CLOCK_PIN, LOW);
    digitalWrite(DATA_PIN, registers[i]);
    digitalWrite(CLOCK_PIN, HIGH);
  }
  digitalWrite(LATCH_PIN, HIGH);
}

// 10-segment LED bar logic for temperature indicator
void heatBar(){
  int T = dht.readTemperature();

  // if DHT11 reads less than 25 celsius, we only light up the first four LEDs
  if(T < 25 && displayState == 1){
    for (int i = 0; i < 4; i++) {
        registers[i] = HIGH; // Color = (1x blue, 3x green)
        writereg();
        delay(10);
      }
    }
    // if DHT11 reads between 25 to 28, we light up the first seven
    else if(T > 25 && T < 28  && displayState == 1){
      for (int i = 0; i < 7; i++) {
        registers[i] = HIGH; // Color = (1x blue, 3x green, 3x orange)
        writereg();
        delay(10);
      }
    }
    // anything above 28 is too hot for the dough to handle long periods
    else if((T > 28 && displayState == 1)){
      for (int i = 0; i < 10; i++) {
        registers[i] = HIGH; // Lights up all the LEDs
        writereg();
        delay(10);
      }
    }
    else if (displayState == 0){ // If you are in main menu > no light
      for(int i = 0; i < 10; i++){
        registers[i] = LOW;
        writereg();
        delay(10);
      }
    }
}

// __________ MAIN LOOP ______________________
void loop() {
    heatControl(); // Heat element controller
    buttons(); // Button UI controller
    updateDisplay(); // Display update controller
    heatBar(); // LED Bar graph controller
}
```