← Press or Scan for Demo Video.
https://rb.gy/zckiui

ECNG 3509 Project: Microcontroller-Based Smart Temperature Control System

Adham Elkhouly

900171543

Wednesday, 6th of December 2023

Instructor: Dr. Heba Draz

Graduate Teaching Assistant: Eng. Marwa Shaheen

Microcontroller System Design Lab (ECNG 459L/4509L)

Abstract

This project leverages microcontroller technology to provide a comprehensive and efficient solution to simulate temperature control of a bakery oven. A user-friendly interface of a keypad and an LCD display is utilized for real-time feedback and operator input. In addition, three temperature sensors are placed to monitor the surrounding temperature, which enables the user to dynamically achieve and maintain the preset temperature.

Moreover, the system is, then, used to orchestrate the operation of a fan and three LED lights to act as heating elements. The number of heating elements that are lit are modulated based on the difference between the pre-set temperature and the actual temperature surrounding the temperature sensor. To mitigate overheating scenarios, the system activates a ventilation mechanism, reversing the fan and disengaging heating elements as a safety precaution.

This project seamlessly blends hardware components and intelligent control algorithms to deliver an efficient, cost-effective, and safe temperature control system for bakery operations. The integration of user interfaces, safety features, and communication capabilities ensures a holistic solution for enhanced baking precision and operational oversight.

Table of Contents

1 List of Hardware

1.1 Arduino DUE

The Arduino Due, as shown in Figure 1, is a microcontroller board that belongs to the Arduino family. It stands out among other Arduino boards due to its powerful features, making it suitable for a wide range of applications, especially those requiring more processing power and memory.

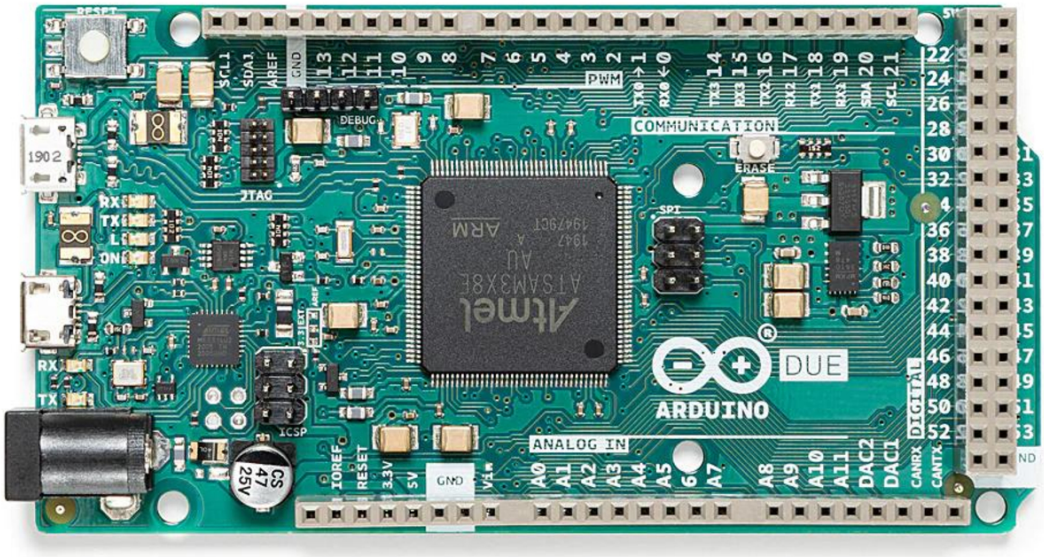


Figure 1

← Press or Scan for Demo Video.

1.2 LCD1602 Module

The LCD1602 module, as shown in Figure 2, is a widely used liquid crystal display (LCD) module. It is a 16x2 character alphanumeric display that consists of 16 columns and 2 rows, making it capable of displaying 32 characters at a time. Each character is formed by a 5x8 pixel matrix, allowing for the representation of alphanumeric characters, symbols, and custom characters. The schematic of this module is shown in Figure 3.

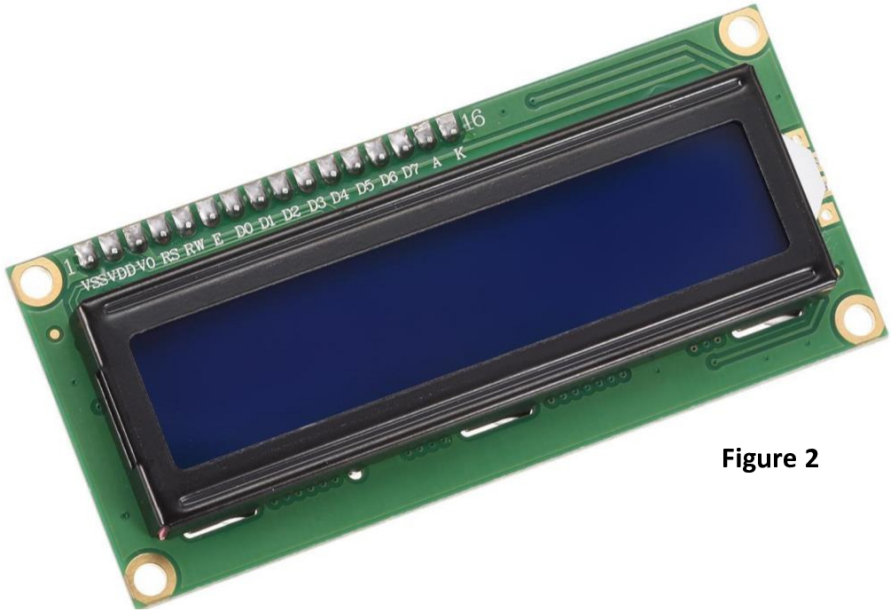


Figure 2

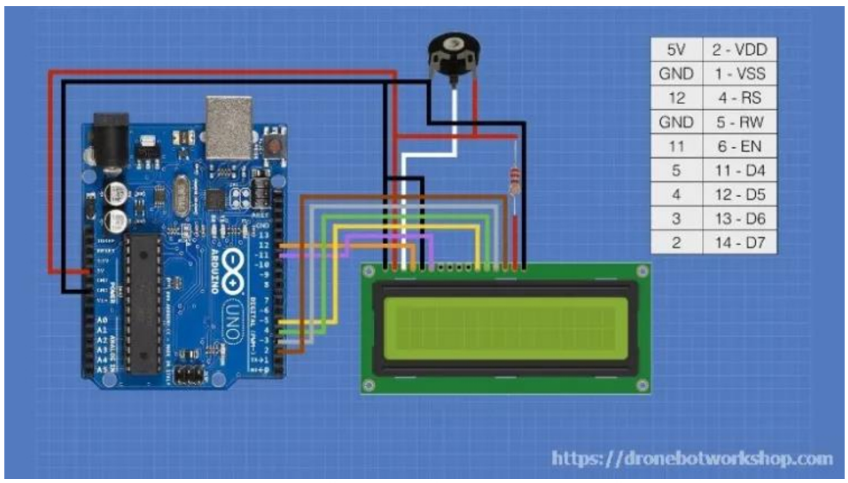


Figure 3

← Press or Scan for Demo Video.

1.3 4 × 4 keypad (as shown in Figure 4 with a schematic shown in Figure 5)

Figure 4

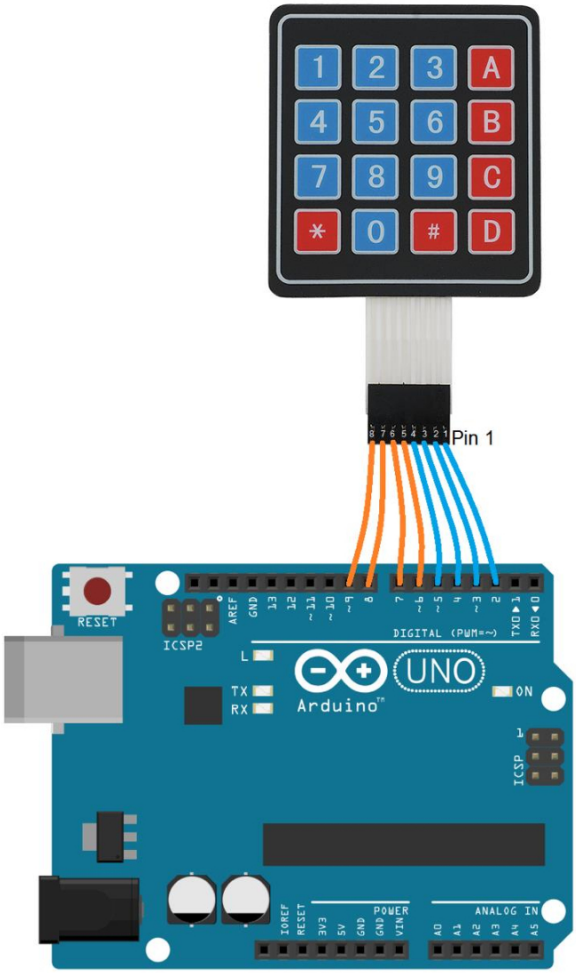


Figure 5

1.4 3 LED lights (to simulate the behavior of heating elements)

Figure 6

1.5 3 LM35 temperature sensors

The LM35 is a precision analog temperature sensor that is Manufactured by Texas Instruments. It is designed to provide an accurate and linear output voltage directly proportional to the Celsius temperature. It is simplicity, reliability, and ease of use.

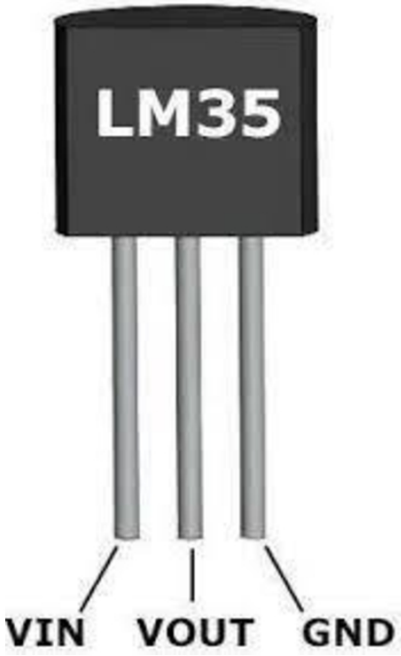


Figure 7

← Press or Scan for Demo Video.

1.6 Fan Blade and 3-6V Motor

Figure 8

1.7. L298N H-Bridge (to reverse the direction of rotation of the fan)

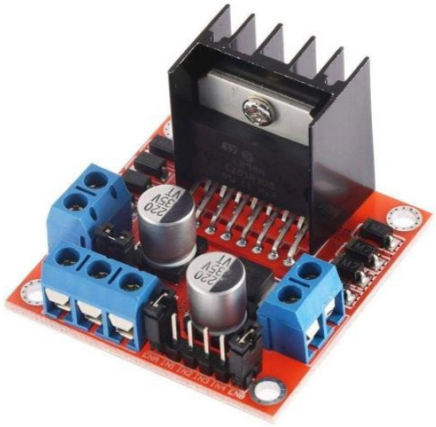


Figure 9

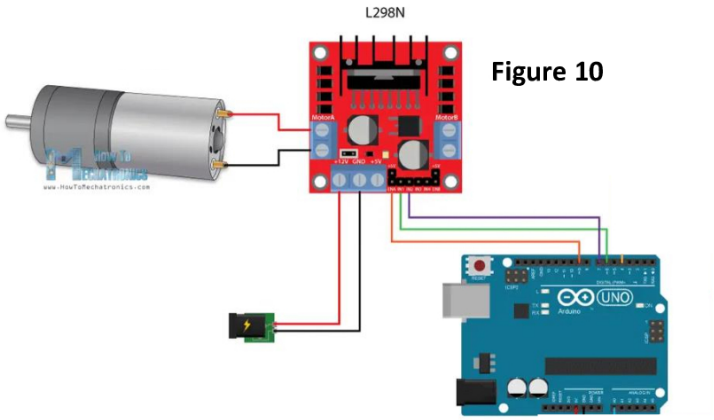


Figure 10

← Press or Scan for Demo Video.

1.8. Buzzer

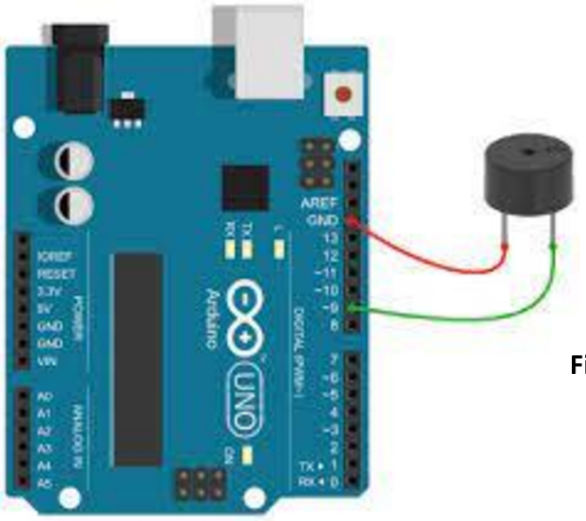


Figure 11

1.9 2 Breadboards

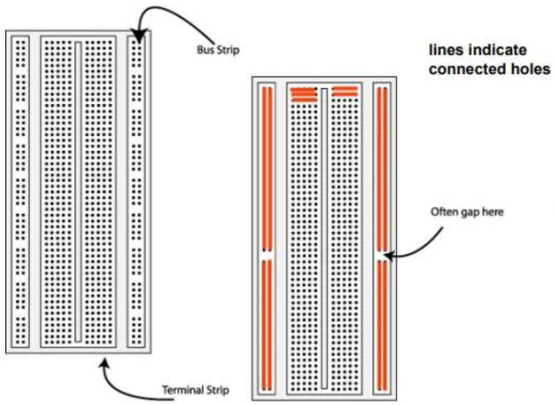


Figure 12

← Press or Scan for Demo Video.

1.10. 9V Battery (for the H-bridge)

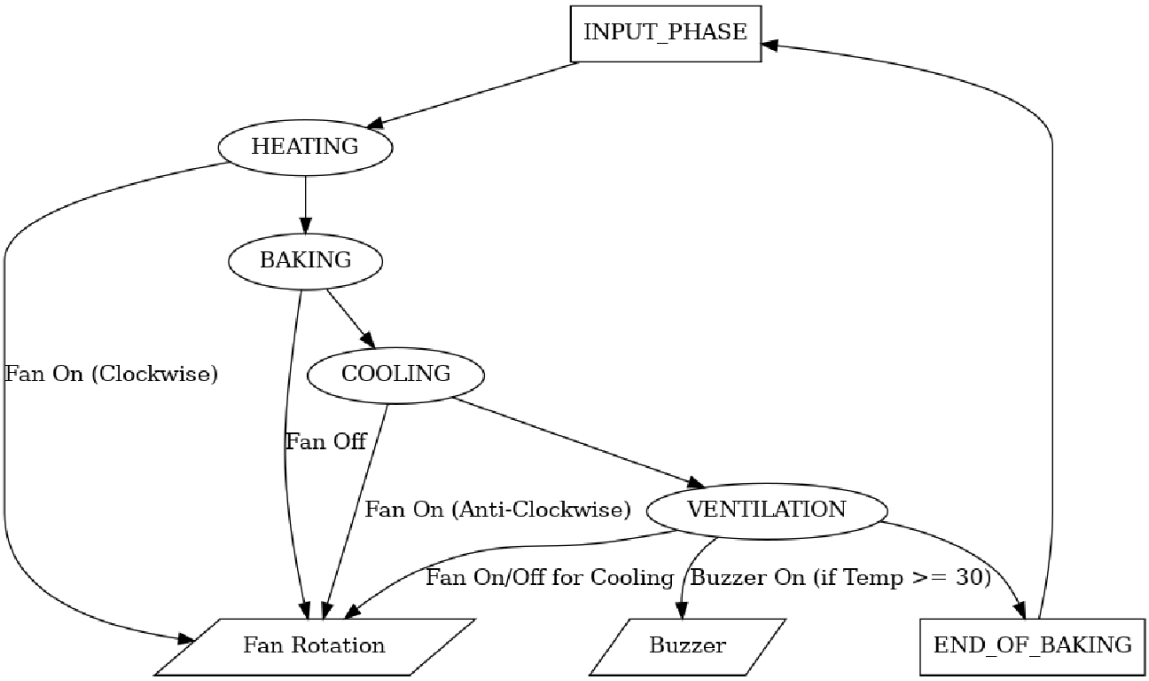
Figure 13

1.11 Male to Male Jumper Wires

Figure 14

2 The Code

The code (as shown in Appendix II), developed in Arduino using Arduino IDE (Version: 2.2.1), orchestrates a bespoke oven control system through a meticulously designed set of functions. Each function encapsulates a specific facet of the overarching project, creating a modular structure. Employing a keypad for user input, a Liquid Crystal Display (LCD) for output, and LM35 temperature sensors for precise temperature measurements, the system allows users to establish a target temperature and baking duration. Throughout the baking process, the system continuously monitors the temperature, dynamically adjusting the fan direction based on temperature conditions. This intelligent control logic facilitates a range of features including heating, cooling, baking, and ventilation. In essence, this code presents a sophisticated and all-encompassing mechanism for managing an oven with diverse operational modes.



2.1 Taking Inputs

2.1.1 Temperature Input Phase:

```
void readingInputs() {
  if (tempOrTime == 0 && inputPhase) {
    char customKey = customKeypad.getKey();
    if (customKey) {
      if (customKey >= '0' && customKey <= '9') {
        tempInputStr += customKey;
        lcd.print(customKey);
      } else if (customKey == '#') {
        if (tempInputStr.length() > 0) {
          targetTemperature = tempInputStr.toInt();
          tempInputStr = "";
          tempOrTime = 1;
          lcd.clear();
          lcd.print("Enter time: ");
        }
      }
    }
  }
}
```

In this block, the **readingInputs** function is responsible for capturing user input during the temperature input phase. It checks if the system is in the temperature input phase (**tempOrTime == 0**), and if the input phase is active (**inputPhase** is **true**). It then listens for key presses using **customKeypad.getKey()**.

- If a numeric key ('0' to '9') is pressed, it accumulates the key in the **tempInputStr** string and displays it on the LCD.
- If the '#' key is pressed, it checks if any temperature input has been entered. If yes, it converts the accumulated temperature string to an integer (**targetTemperature**) and clears the string. It then transitions to the time input phase, clearing the LCD and prompting the user to enter the baking time.

2.1.2 Time Input Phase:

```
void readingInputs() {
  // ... (previous code)

  else if (tempOrTime == 1 && inputPhase) {
    char customKey = customKeypad.getKey();
    if (customKey) {
      if (customKey >= '0' && customKey <= '9') {
        tempInputStr += customKey;
        lcd.print(customKey);
      } else if (customKey == '#') {
        if (tempInputStr.length() > 0) {
          bakingTime = tempInputStr.toInt();
          tempInputStr = "";
          inputPhase = false;
        }
      }
    }
  }
}
```

In this part of the code, the system is in the time input phase (**tempOrTime == 1**). Similar to the temperature input phase, it captures numeric key presses, accumulates them in **tempInputStr**, and displays them on the LCD. If the '#' key is pressed, it checks if any time input has been entered. If yes, it converts the accumulated time string to an integer (**bakingTime**) and clears the string. Finally, it sets **inputPhase** to **false**, indicating the completion of the input phase.

These blocks together create a user-friendly interface for setting the target temperature and baking time using the keypad. The LCD provides real-time feedback to the user during the input process.

2.2 Temperature Control

2.2.1 Temperature Sensors:

- Three analog temperature sensors are used, connected to analog pins A0, A1, and A2.
- These sensors sample the temperature at three different locations within the oven.

```
void initializeTemperatureSensors() {
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
}
```

2.2.2 Temperature Calculation:

- The function **calculateAverageTemperature()** calculates the average temperature based on the readings from the three sensors.
- The voltage readings are converted to temperature values.

```
int calculateAverageTemperature() {
  double voltage1 = analogRead(A0) / 2.048;
  double voltage2 = analogRead(A1) / 2.048;
  double voltage3 = analogRead(A2) / 2.048;
  return static_cast<int>((voltage1 + voltage2 + voltage3) / 3 - 10);
}
```

2.2.3 Heating Elements:

- The oven has up to three heating elements controlled by pins 48, 50, and 52.
- The function **activateHeatingElements()** is responsible for activating the appropriate number of heating elements based on the temperature difference.

```
void activateHeatingElements(char numElements) {
  // Logic to activate heating elements based on the temperature difference.
}
```

2.2.4 Fan Control:

- The oven is equipped with a fan, controlled by pins 22 and 24.
- The functions **turnFanClockwise()**, **turnFanAntiClockwise()**, and **turnOffFan()** control the fan's direction and turning it off.

```
void initializeFan() {
  pinMode(22, OUTPUT);
  pinMode(24, OUTPUT);
  turnOffFan();
}

void turnFanClockwise() {
  // Logic to turn the fan clockwise.
}

void turnOffFan() {
  // Logic to turn off the fan.
}

void turnFanAntiClockwise() {
  // Logic to turn the fan counterclockwise.
}
```

I) Fan Direction Control:

The fan plays a crucial role in regulating the temperature within the oven. Depending on the temperature conditions, the fan is controlled to rotate in either a clockwise or counterclockwise direction.

Clockwise rotation is activated when the temperature inside the oven is below the user-set temperature. This direction is intended to evenly distribute heat within the oven space, ensuring consistent temperature throughout. The clockwise rotation is achieved by setting the appropriate control signals on the fan motor using the **turnFanClockwise()** function.

```
void turnFanClockwise() {
  digitalWrite(22, HIGH);  // Signal to rotate clockwise
  digitalWrite(24, LOW);
}
```

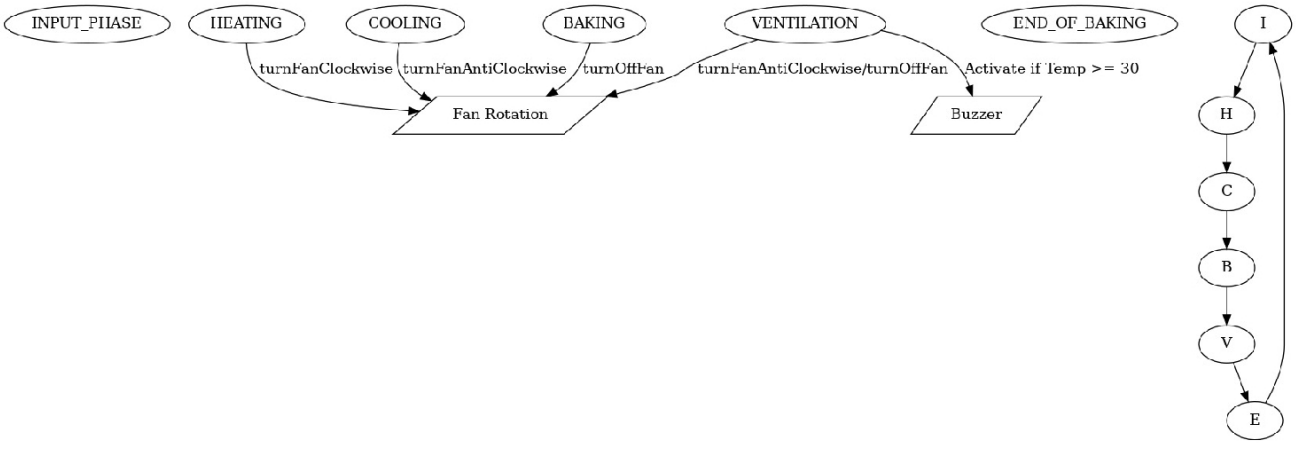
Counterclockwise rotation is initiated when the temperature inside the oven surpasses the desired temperature by a certain threshold. This direction aims to dissipate excess heat and prevent overheating. The counterclockwise rotation is activated using the **turnFanAntiClockwise()** function.

```
void turnFanAntiClockwise() {
  digitalWrite(22, LOW);
  digitalWrite(24, HIGH);  // Signal to rotate counterclockwise
}
```

The fan is turned off when it is not required for temperature regulation. This occurs during the baking phase when the temperature is within the acceptable range. The turnOffFan() function sets both control signals to low, effectively turning off the fan.

```
void turnOffFan() {
  digitalWrite(22, LOW);
  digitalWrite(24, LOW);
}
```

The decision to change the fan direction is based on the temperature conditions and is an integral part of the overall temperature control strategy within the oven. The fan direction control enhances the efficiency of the oven in achieving and maintaining the desired baking temperature.

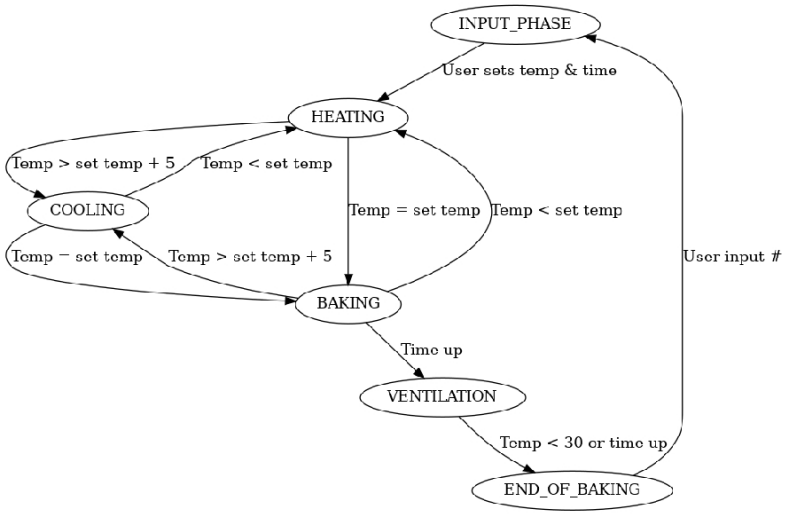


← Press or Scan for Demo Video.

2.2.5 State Machine:

- The overall temperature control logic is implemented using a state machine.
- States such as HEATING, COOLING, BAKING, VENTILATION, and END_OF_BAKING are used to control the oven's behavior based on temperature conditions.

```
switch (ovenState) {
  case INPUT_PHASE:
    // Read user inputs for temperature and time.
    break;
case HEATING:
    // Control heating elements and fan during the heating phase.
    break;
case COOLING:
    // Control fan and heating elements during the cooling phase.
    break;
case BAKING:
    // Control heating elements during the baking phase.
    break;
case VENTILATION:
    // Control fan and heating elements during the ventilation phase.
    break;
case END_OF_BAKING:
    // Perform actions at the end of the baking process such as buzzer sound.
    break;
}
```



I) Oven States:

The state machine manages the overall operation of the oven, transitioning between different states based on user inputs and temperature conditions.

II) **Transition Conditions**:

The transition between states is governed by specific conditions that are continuously monitored during the operation of the oven.

Temperature Conditions:

The temperature conditions are crucial in determining the transitions between the HEATING, COOLING, and BAKING states.

For example, in the performHeating() function:

```
void performHeating() {
  // Existing code...

  if (calculateAverageTemperature() < inputTemperature) {
    turnFanClockwise();

    if ((inputTemperature - calculateAverageTemperature()) >= 30) {
      activateHeatingElements(3);
    } else if ((inputTemperature - calculateAverageTemperature()) >= 20) {
      activateHeatingElements(2);
    } else {
      activateHeatingElements(1);
    }
  } else if (calculateAverageTemperature() > inputTemperature + 5) {
    turnOffFan();
    ovenState = COOLING;
    lcd.clear();
  } else {
    ovenState = BAKING;
    lcd.clear();
  }
}
```

Time Conditions:

The time conditions are used to track the elapsed time and initiate transitions, particularly in the loop() function:

```
if (ovenState != INPUT_PHASE && inputTime > 0) {
  startTime2 = millis();

  if (startTime2 - startTime1 >= 1000) {
    startTime1 = startTime2;
    inputTime--;

    if (inputTime == 0) {
      ovenState = VENTILATION;
    }
  }
}
```

This checks if the baking time has elapsed, triggering a transition to the VENTILATION state.

Ventilation Threshold:

The ventilation state is entered when the temperature falls below a specified threshold after baking:

```
if (calculateAverageTemperature() >= 30) {
  turnFanAntiClockwise();
  activateHeatingElements(0);
} else if (calculateAverageTemperature() < 30) {
  for (char i = 0; i < 5; i++) {
    digitalWrite(26, HIGH);
    delay(300);
    digitalWrite(26, LOW);
    delay(300);
  }
  ovenState = END_OF_BAKING;
  lcd.clear();
}
```

- The oven transitions to the **END_OF_BAKING** state when the temperature is sufficiently low for ventilation.

3. Time Management

3.1. Input Time:

During the **INPUT_PHASE**, the user inputs the desired baking time using the keypad.

- The **readInputs()** function reads the user input for time after the temperature input phase.
- The entered time is then used to set the **inputTime** variable.

3.1.1 Countdown Mechanism:

Once the oven transitions to the **HEATING** state, the countdown mechanism begins.

- The countdown is implemented in the **loop()** function using the **startTime1** and **startTime2** variables.
- The **startTime1** variable is initialized with the current time in milliseconds when the countdown starts.
- In each iteration of the **loop()**, the elapsed time is calculated, and the **inputTime** is decremented accordingly.

```
if (ovenState != INPUT_PHASE && inputTime > 0) {
  startTime2 = millis();

  if (startTime2 - startTime1 >= 1000) {
    startTime1 = startTime2;
    inputTime--;

    if (inputTime == 0) {
      ovenState = VENTILATION;
    }
  }
}
```

3.2 Elapsed Time Tracking:

Elapsed time is tracked using the **startTime1** and **startTime2** variables.

- The **startTime1** variable is updated at regular intervals, and the elapsed time is calculated by subtracting the previous start time from the current time.

```
startTime2 = millis();

if (startTime2 - startTime1 >= 1000) {
  startTime1 = startTime2;
  // Perform actions every second
}
```

3.2.1 State Transition on Time Up:

When the countdown reaches zero, the oven transitions to the **VENTILATION** state.

3 If **inputTime** becomes zero, the oven state is set to **VENTILATION** in the **loop()** function.

```
if (inputTime == 0) {
  ovenState = VENTILATION;
}
```

This time management mechanism ensures that the oven operates for the specified baking time, and it transitions to the ventilation phase once the time is up. The countdown and elapsed time tracking contribute to precise time management in the oven operation.

4. Bill of Materials (BOM)

Item	Cost (EGP)	Quantity	Total (EGP)
Arduino DUE	-	1	-
LCD1602 Module	60	1	60
4×4 keypad	40	1	40
LED Set	80	1	80
3 LM35 temperature sensors	75	3	225
Fan Blade and 3-6V Motor	50	1	50
L298N H-Bridge	65	1	65
Buzzer	6.5	1	6.5
2 Breadboards	35	2	70
9V Battery	40	1	40
Male to Male Jumper Wires	40	1	40
Tax (14%)	-	-	76.3
Total	-	-	722.8

← Press or Scan for Demo Video.

5. Conclusion

In conclusion, the Arduino-based bakery oven control system has not only met the initial project goals but has also provided a platform for ongoing exploration and improvement. The experience gained from this project contributes to our continuous pursuit of innovation and excellence in embedded systems design.

6. Appendix

6.1 Appendix 1:

THE AMERICAN UNIVERSITY IN CAIRO

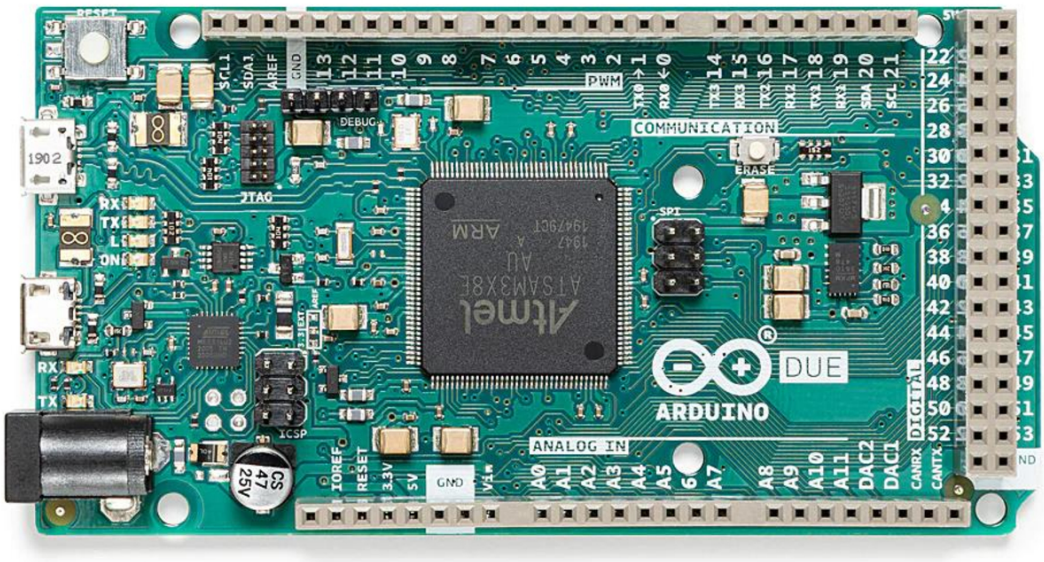


Figure 1

← Press or Scan for Demo Video.

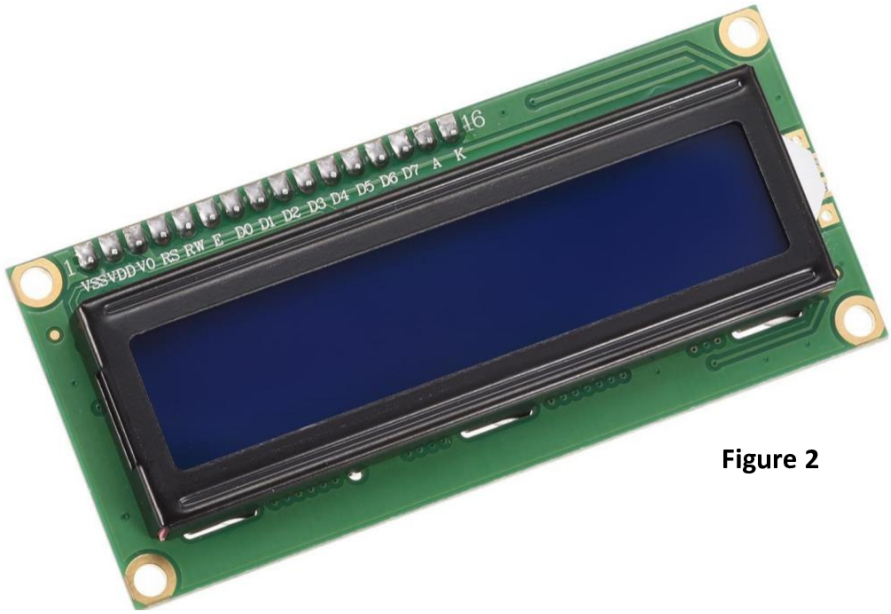


Figure 2

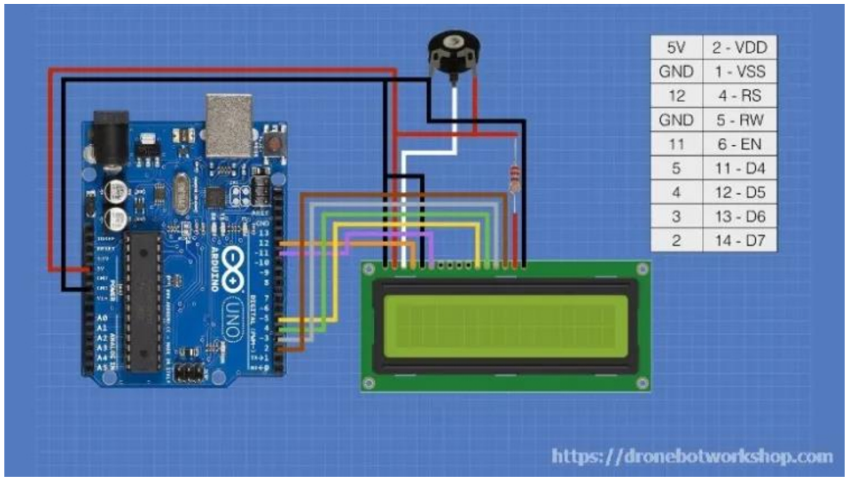


Figure 3

← Press or Scan for Demo Video.

Figure 4

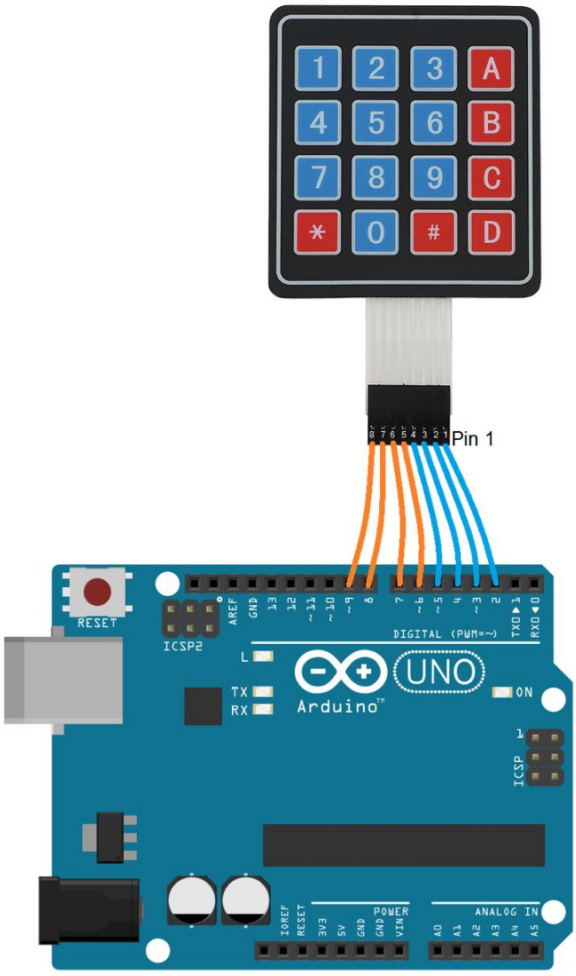


Figure 5

← Press or Scan for Demo Video.

Figure 6

Figure 7

Figure 8

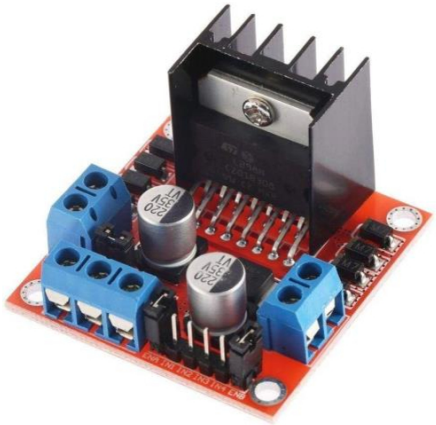


Figure 9

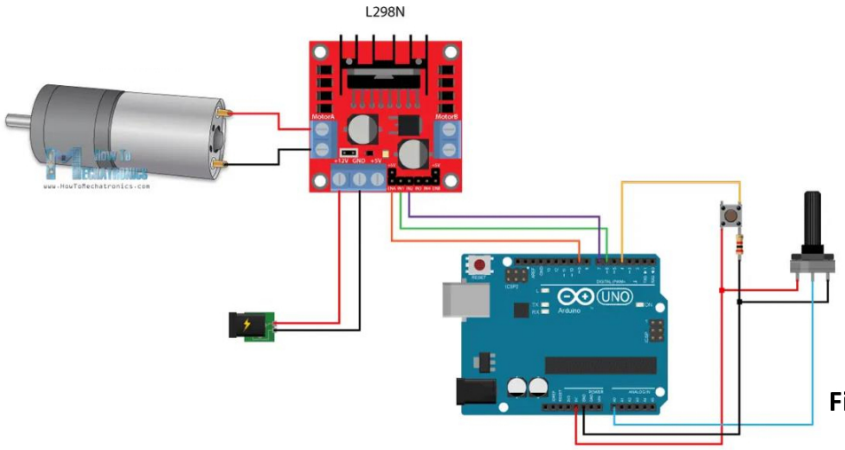


Figure 10

← Press or Scan for Demo Video.

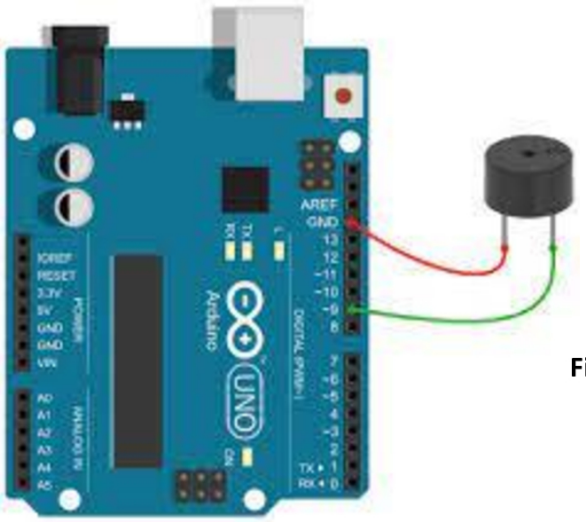


Figure 11

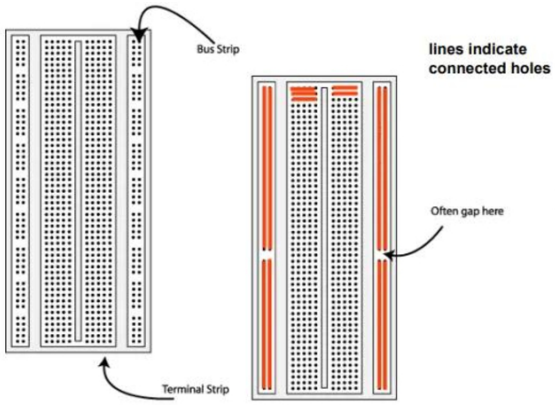


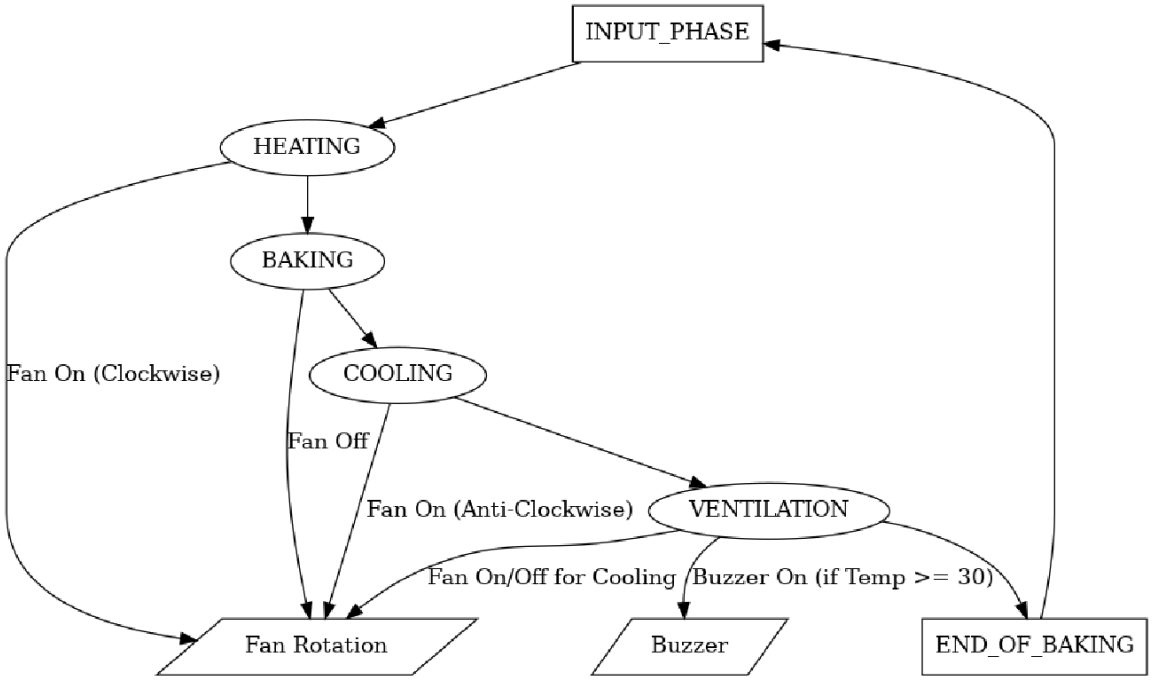
Figure 12

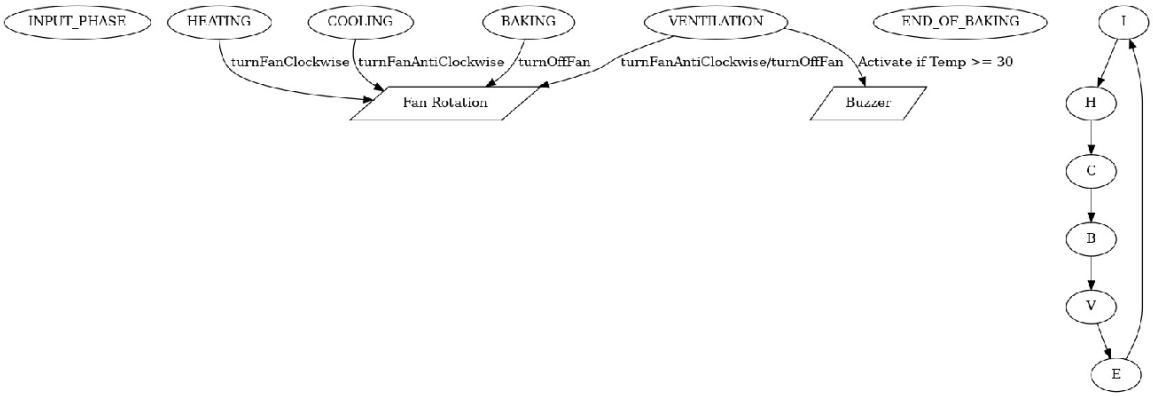
← Press or Scan for Demo Video.

Figure 13

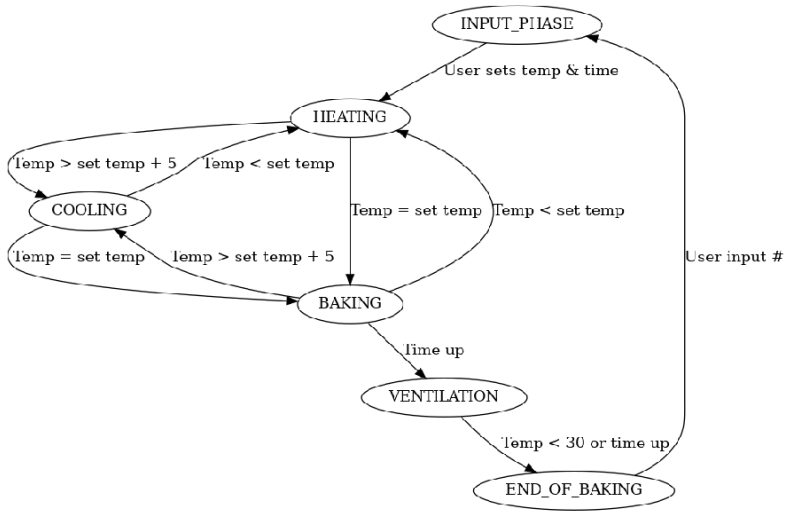
Figure 14

← Press or Scan for Demo Video.





← Press or Scan for Demo Video.



Item	Cost (EGP)	Quantity	Total (EGP)
Arduino DUE	-	1	-
LCD1602 Module	60	1	60
4×4 keypad	40	1	40
LED Set	80	1	80
3 LM35 temperature sensors	75	3	225
Fan Blade and 3-6V Motor	50	1	50
L298N H-Bridge	65	1	65
Buzzer	6.5	1	6.5
2 Breadboards	35	2	70
9V Battery	40	1	40
Male to Male Jumper Wires	40	1	40
Tax (14%)	-	-	76.3
Total	-	-	722.8

6.2 Appendix II:

```
/*
 * @file CustomOvenControl.ino
 * @version 1.0
 * @author Adham Elkhouly
 * @contact Adham_Elkhouly@aucegypt.edu
 *
 * @description
 * This Arduino code regulates a bakery electric oven's hot-air temperature based on user-set presets for temperature and baking time. It samples temperature at three locations, controls a fan, and activates up to three heating elements based on the temperature difference.
 */

#include <Keypad.h>
#include <LiquidCrystal.h>

// states for the oven
#define INPUT_PHASE 0
#define HEATING 1
#define COOLING 2
#define BAKING 3
#define END_OF_BAKING 4
#define VENTILATION 5

char ovenState = INPUT_PHASE; // Current state of the oven

// Function prototypes
void activateHeatingElements(char numElements);
void initializeHeatingElements();
void readInputs();
int calculateAverageTemperature();
void initializeLCD();
void initializeKeypad();
void initializeTemperatureSensors();
void initializeFan();
void turnFanClockwise();
void turnFanAntiClockwise();
void turnOffFan();
void performCooling();
void performBaking();
void performVentilation();
void performEndOfBaking();

// Input variables
String inputString = "";
long inputTemperature;
```

```
long inputTime;
char tempOrTime = 0; // 0 for temperature input, 1 for time input
long startTime1 = 0, startTime2 = 0;

// LCD and Keypad pin configurations
const int rsPin = 12, enPin = 11, d4Pin = 5, d5Pin = 4, d6Pin = 3, d7Pin = 2;
LiquidCrystal lcd(rsPin, enPin, d4Pin, d5Pin, d6Pin, d7Pin);
const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {39, 41, 43, 45};
byte colPins[COLS] = {47, 49, 51, 53};
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

// time tracking
int elapsedTime = 0;

void setup() {
  pinMode(26, OUTPUT); // pin for a possible output (fan or buzzer)

  initializeTemperatureSensors();
  initializeHeatingElements();
  initializeFan();
  initializeLCD();
}

void loop() {
  //  State machine for oven operation
  switch (ovenState) {
    case INPUT_PHASE:
      readInputs();
      break;
    case HEATING:
      performHeating();
      break;
    case COOLING:
      performCooling();
      break;
    case BAKING:
      performBaking();
      break;
    case VENTILATION:
      performVentilation();
```

```
      break;
    case END_OF_BAKING:
      performEndOfBaking();
      break;
  }

  // update  time and switch to VENTILATION state if time is up
  if (ovenState != INPUT_PHASE && inputTime > 0) {
    startTime2 = millis();

    if (startTime2 - startTime1 >= 1000) {
      startTime1 = startTime2;
      inputTime--;

      if (inputTime == 0) {
        ovenState = VENTILATION;
      }
    }
  }
}

void initializeTemperatureSensors() {
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
}

void initializeHeatingElements() {
  pinMode(52, OUTPUT);
  pinMode(50, OUTPUT);
  pinMode(48, OUTPUT);
}

void activateHeatingElements(char numElements) {
  switch (numElements) {
    case 0:
      digitalWrite(52, LOW);
      digitalWrite(50, LOW);
      digitalWrite(48, LOW);
      break;
    case 1:
      digitalWrite(52, HIGH);
      digitalWrite(50, LOW);
      digitalWrite(48, LOW);
      break;
    case 2:
      digitalWrite(52, HIGH);
      digitalWrite(50, HIGH);
```

```
      digitalWrite(48, LOW);
      break;
    case 3:
      digitalWrite(52, HIGH);
      digitalWrite(50, HIGH);
      digitalWrite(48, HIGH);
      break;
  }
}

void initializeFan() {
  pinMode(22, OUTPUT);
  pinMode(24, OUTPUT);
  turnOffFan();
}

void turnFanClockwise() {
  digitalWrite(22, HIGH);
  digitalWrite(24, LOW);
}

void turnOffFan() {
  digitalWrite(22, LOW);
  digitalWrite(24, LOW);
}

void turnFanAntiClockwise() {
  digitalWrite(22, LOW);
  digitalWrite(24, HIGH);
}

void initializeLCD() {
  lcd.begin(16, 2);
  lcd.print("Enter temp: ");
}

void initializeKeypad() {
}

void readInputs() {
  if (tempOrTime == 0) {
    char customKey = customKeypad.getKey();
    if (customKey) {
      if (customKey >= '0' && customKey <= '9') {
        inputString += customKey;
        lcd.print(customKey);
      } else if (customKey == '#') {
        if (inputString.length() > 0) {
```

```
        inputTemperature = inputString.toInt();
        inputString = "";
        tempOrTime = 1;
        lcd.clear();
        lcd.print("Enter time:");
      }
    }
  }
 } else if (tempOrTime == 1) {
  char customKey = customKeypad.getKey();
  if (customKey) {
    if (customKey >= '0' && customKey <= '9') {
      inputString += customKey;
      lcd.print(customKey);
    } else if (customKey == '#') {
      startTime1 = millis();
      if (inputString.length() > 0) {
        inputTime = inputString.toInt();
        inputString = "";
        ovenState = HEATING;
        tempOrTime = 0;
        lcd.clear();
      }
    }
  }
 }
}

int calculateAverageTemperature() {
 double voltage1 = analogRead(A0) / 2.048;
 double voltage2 = analogRead(A1) / 2.048;
 double voltage3 = analogRead(A2) / 2.048;
 return static_cast<int>((voltage1 + voltage2 + voltage3) / 3 - 10);
}

void performHeating() {
 lcd.setCursor(0, 0);
 lcd.print("HEATING");
 lcd.setCursor(0, 1);
 lcd.print("Temp:");
 lcd.print(calculateAverageTemperature());
 lcd.print(" Time:");
 lcd.print(inputTime);

 if (calculateAverageTemperature() < inputTemperature) {
  turnFanClockwise();

  if ((inputTemperature - calculateAverageTemperature()) >= 30) {
```

```
      activateHeatingElements(3);
    } else if ((inputTemperature - calculateAverageTemperature()) >= 20) {
      activateHeatingElements(2);
    } else {
      activateHeatingElements(1);
    }
  } else if (calculateAverageTemperature() > inputTemperature + 5) {
    turnOffFan();
    ovenState = COOLING;
    lcd.clear();
  } else {
    ovenState = BAKING;
    lcd.clear();
  }
}

void performCooling() {
  lcd.setCursor(0, 0);
  lcd.print("COOLING");
  lcd.setCursor(0, 1);
  lcd.print("Temp:");
  lcd.print(calculateAverageTemperature());
  lcd.print(" Time:");
  lcd.print(inputTime);

  if (calculateAverageTemperature() > inputTemperature + 5) {
    turnFanAntiClockwise();
    activateHeatingElements(0);
  } else if (calculateAverageTemperature() < inputTemperature) {
    turnOffFan();
    ovenState = HEATING;
    lcd.clear();
  } else {
    ovenState = BAKING;
    lcd.clear();
  }
}

void performBaking() {
  lcd.setCursor(0, 0);
  lcd.print("BAKING");
  lcd.setCursor(0, 1);
  lcd.print("Temp:");
  lcd.print(calculateAverageTemperature());
  lcd.print(" Time:");
  lcd.print(inputTime);
  turnOffFan();
  activateHeatingElements(0);
```

```
  if (calculateAverageTemperature() > inputTemperature + 5) {
    ovenState = COOLING;
    lcd.clear();
  } else if (calculateAverageTemperature() < inputTemperature) {
    ovenState = HEATING;
    lcd.clear();
  }
}

void performVentilation() {
  lcd.setCursor(0, 0);
  lcd.print("VENTILATION");
  lcd.setCursor(0, 1);
  lcd.print("Temp:");
  lcd.print(calculateAverageTemperature());
  lcd.print(" Time:");
  lcd.print(inputTime);

  if (calculateAverageTemperature() >= 30) {
    turnFanAntiClockwise();
    activateHeatingElements(0);
  } else if (calculateAverageTemperature() < 30) {
    for (char i = 0; i < 5; i++) {
      digitalWrite(26, HIGH);
      delay(300);
      digitalWrite(26, LOW);
      delay(300);
    }
    ovenState = END_OF_BAKING;
    lcd.clear();
  }
}

void performEndOfBaking() {
  turnOffFan();
  activateHeatingElements(0);
  lcd.setCursor(0, 0);
  lcd.print("END_OF_BAKING");
  lcd.setCursor(0, 1);
  lcd.print("Temp:");
  lcd.print(calculateAverageTemperature());
  lcd.print(" Time:");
  lcd.print(inputTime);

  char customKey = customKeypad.getKey();
  if (customKey) {
    if (customKey >= '#') {
```

← Press or Scan for Demo Video.


```
    lcd.clear();
    lcd.print("Enter temp: ");
    ovenState = INPUT_PHASE;
   }
  }
}
```