Basic Heater Control System using Arduino Uno and DHT22

Purpose:

This project design and implement a heater control system using an Arduino Uno that reads temperature from a DHT22 sensor, tracks system states (Idle, Heating, Stabilizing, Target Reached, Overheat), and controls a simulated heater (LED) with overheat protection (buzzer alert). The system displays real-time temperature and state on a 16x2 LCD and logs data over the Serial Monitor for monitoring and debugging.

It ensures:

- Heating when temperature is below the set threshold.
- Stabilization as it nears the target.
- Alerts when overheating occurs.

Objective

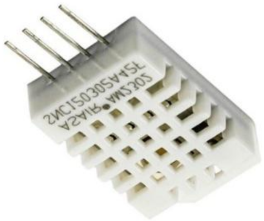
Automatically control a heater using a DHT22 sensor and potentiometer to dynamically adjust the target temperature. System states:

- **HEATING**: temp < (target − 5 °C) → heater ON (LED on, buzzer off)
- **STABILIZING**: between (target − 5 °C) and target → heater remains ON
- **TARGET_REACHED**: temp ≥ target → heater turns OFF
- **OVERHEAT**: temp ≥ overheat threshold (e.g. 40 °C) → buzzer beeps

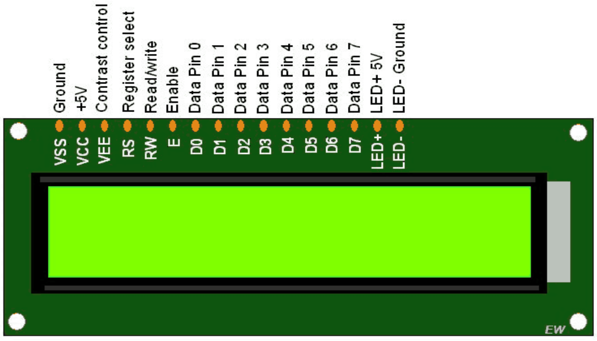
Main Components & Their Roles:

1. Arduino Uno – The brain of the system. Reads sensor data, processes logic, and controls outputs.

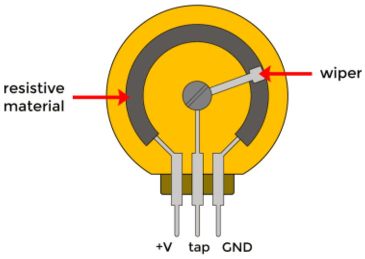
2. DHT22 Temperature Sensor – Measures current ambient temperature.



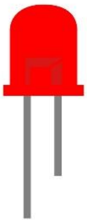
3. 16×2 LCD Display – Shows real-time temperature, target temperature, and system state.



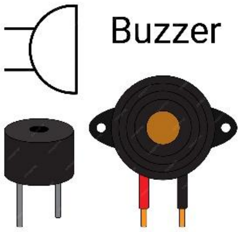
4. Potentiometer – Used to set the target temperature (between 20°C and 35°C).



5. LED – Indicates when the heating system is active.



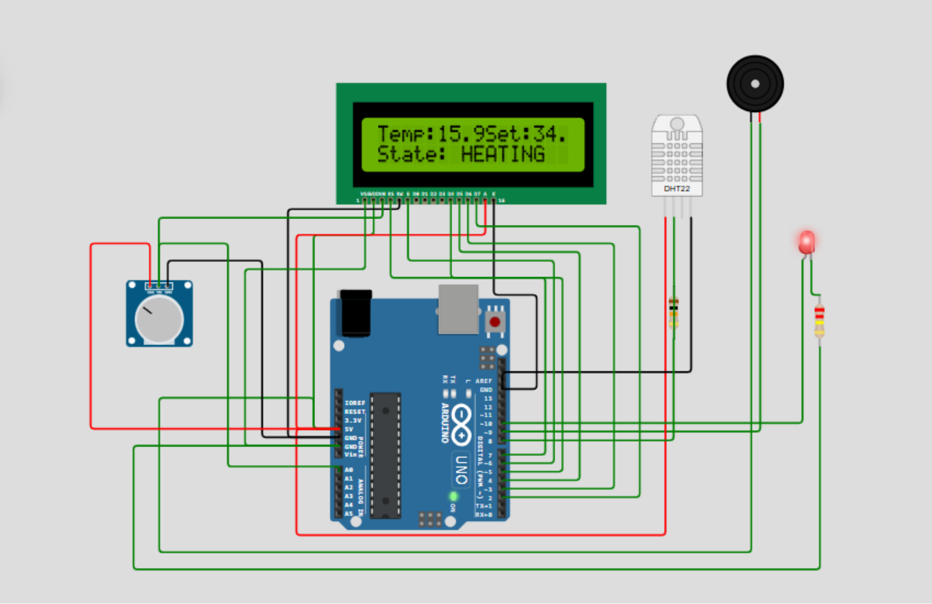
6. Buzzer – Alerts when the system enters the overheat state.



7. Heater (simulated by LED) – Represents the heating element being controlled.

8. Power Supply – Provides power to Arduino and components.

Block Diagram:



Working Principle:

1. **Read Temperature:** The Arduino reads temperature from the DHT22 sensor.
2. **State Tracking:** Based on the temperature thresholds:
   - Below 25°C: HEATING (heater ON, LED ON)
   - 25°C - 30°C: STABILIZING (heater ON, LED ON)
   - 30°C - 35°C: TARGET_REACHED (heater OFF, LED OFF)
   - Above 35°C: OVERHEAT (heater OFF, LED OFF, buzzer ON)
3. **Display:** The system displays the current temperature and system state on the LCD.
4. **Logging:** Logs the temperature and system state over the Serial Monitor for debugging.

Observations:

- The system successfully reads and logs the temperature.
- System states transition correctly based on the test and actual temperatures.
- The LED (heater) responds as per the system state.
- The buzzer activates when the temperature exceeds the overheat threshold.

- The LCD displays the temperature and system state clearly once the potentiometer is correctly adjusted.

1. Arduino Uno Controller

- **Type:** Microcontroller board (ATmega328P)
- **Role:**
    - Reads sensor data (temperature from DHT22, voltage from potentiometer).
    - Runs the **state machine logic** to decide whether to heat, stabilize, shut off, or trigger overheat alarm.
    - Controls output devices (LED, buzzer).
    - Updates the **LCD** and **Serial Monitor** with live system data.
- **Connections:**
    - Digital pins for DHT22, LED, buzzer, LCD.
    - Analog pin for potentiometer.
- **Working:**
    - Continuously runs the loop() function, making decisions based on thresholds:
        - Below (target − 5) → heater ON
        - Between (target − 5) & target → stabilizing
        - Above target → heater OFF
        - Above 40 °C → Overheat alarm (buzzer)

2. DHT22 Sensor (Temperature & Humidity)

- **Type:** Digital temperature & humidity sensor.
- **Role:**
    - Provides the **current ambient temperature** (in °C).
    - (Humidity data is available but not used here).
- **Connections:**

- VCC → 5 V
- GND → GND
- Data → Arduino **D8** (with a pull-up resistor).

- **Working:**
  - Internally has a thermistor and a microcontroller to convert analog temperature data to a **digital signal**.
  - Arduino reads it using the DHT library.
  - Data is processed every ~2 seconds (DHT22 has a slow refresh rate).

3. Potentiometer (Target Temperature Adjuster)

- **Type:** 10 kΩ variable resistor.
- **Role:**
  - Used by the user to **set the target temperature** (e.g., between 20 °C and 35 °C).
- **Connections:**
  - One end → 5 V
  - Other end → GND
  - Wiper (middle) → Arduino **A0**.
- **Working:**
  - Acts as a **voltage divider**.
  - The wiper outputs a voltage between 0–5 V depending on knob position.
  - Arduino reads this as an analog value (0–1023), which is mapped to a temperature range using map().

4. LCD Display (16×2 Character Display)

- **Type:** Parallel interface LCD (HD44780 controller).
- **Role:**
    - Displays the **current temperature**, **target temperature**, and **system state** (HEATING/STABILIZING/TARGET/OVERHEAT).
- **Connections:**
    - RS → **D7**
    - E → **D6**
    - D4–D7 → **D5, D4, D3, D2**
    - VCC → 5 V
    - RW → GND (write-only mode).
    - Contrast pin (V0) → Potentiometer → GND.
- **Working:**
    - Controlled by Arduino using the **LiquidCrystal library**.
    - Updated in real-time with new readings every loop cycle.

5. LED (Heater Indicator)

- **Type:** Standard 5 mm LED.
- **Role:**
    - Indicates when the **heater is ON**.
- **Connections:**
    - Anode → Arduino **D10** (via current-limiting resistor ~220Ω).
    - Cathode → GND.
- **Working:**

    - Arduino sets D10 **HIGH** to turn the LED ON when heating is active.
    - Set to **LOW** when target is reached or overheat is detected.

6. Buzzer

- **Type:** Piezoelectric buzzer.
- **Role:**
    - Alerts when the system enters the **overheat state** (e.g., above 40 °C).
- **Connections:**
    - Positive → Arduino **D11**.
    - Negative → GND.
- **Working:**
    - Arduino toggles the buzzer ON/OFF in intervals (millis() function) to create a **beeping sound**.

7. Power Supply

- **Type:** Arduino 5 V regulated supply (USB or adapter).
- **Role:**
    - Provides power to Arduino and all peripherals (DHT22, LCD, LED, Buzzer).
- **Working:**
    - Stable 5 V ensures proper readings and safe operation for all modules.

8. Logic Flow – How Everything Works Together

1. **Startup:**
   - LCD initializes, Arduino reads the potentiometer to set the target temperature.
2. **Temperature Reading:**
   - DHT22 sends the current temperature to Arduino.
3. **Decision Making (State Machine):**
   - If temp < target − 5 → **HEATING** (LED ON).
   - If temp < target but > lower threshold → **STABILIZING** (LED ON).
   - If temp ≥ target → **TARGET REACHED** (LED OFF).
   - If temp ≥ 40 °C → **OVERHEAT** (Buzzer ON, LED OFF).
4. **Outputs:**
   - Update LCD with live readings.
   - Print logs on Serial Monitor.
   - Turn ON/OFF LED or Buzzer depending on the state.

Wiring Guide

- **DHT22** → Data pin to Arduino D8, VCC to 5 V, GND to ground.
- **Potentiometer** → Wiper (middle pin) to A0; outer pins to 5 V and GND.
- **LCD Parallel**:
  - RS → D7
  - E → D6
  - D4 → D5

  - D5 → D4
  - D6 → D3
  - D7 → D2
  - V0 (contrast) → pot to GND
  - VCC, RW, ground appropriately
- **LED (heater indicator)** → Digital pin 10 (via resistor) to GND
- **Buzzer** → Digital pin 11 to GND

Arduino code:

```
#include <DHT.h>
#include <LiquidCrystal.h>


#define DHTPIN 8
#define LED_PIN 10
#define BUZZER_PIN 11
#define POT_PIN A0
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);


float lowerThreshold;
float targetTemp = 30.0;
const float overheatTemp = 40.0;
enum State {IDLE, HEATING, STABILIZING, TARGET_REACHED, OVERHEAT};
State currentState = IDLE;
```

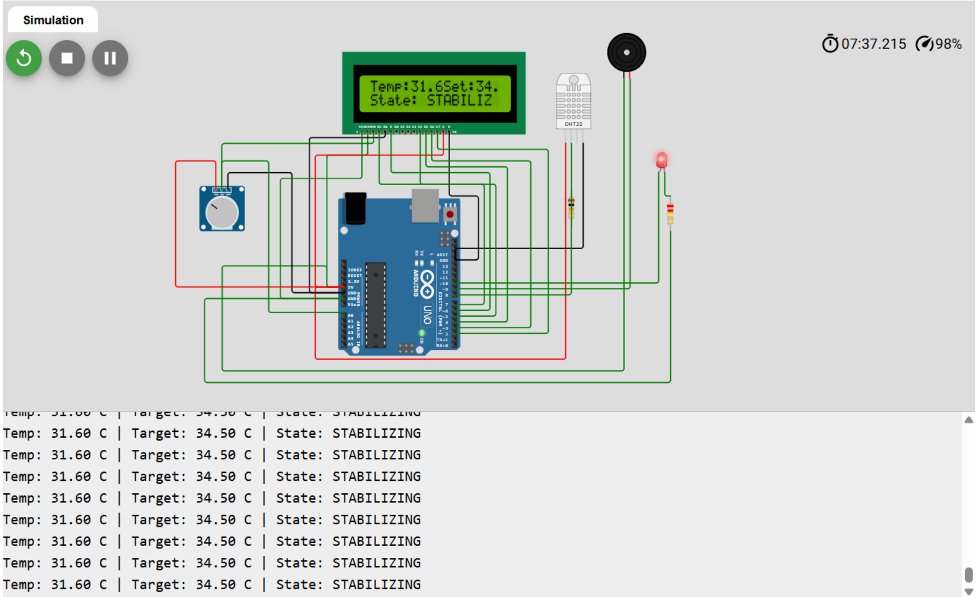
```
void setup() {
pinMode(LED_PIN, OUTPUT);
pinMode(BUZZER_PIN, OUTPUT);
pinMode(POT_PIN, INPUT);
digitalWrite(LED_PIN, LOW);
digitalWrite(BUZZER_PIN, LOW);

lcd.begin(16, 2);
lcd.print("Initializing...");
delay(1500);
lcd.clear();

Serial.begin(9600);
dht.begin();
}

void loop() {
int potValue = analogRead(POT_PIN);
targetTemp = map(potValue, 0, 1023, 200, 350) / 10.0;
lowerThreshold = targetTemp - 5;

float temp = dht.readTemperature();
if (isnan(temp)) {
Serial.println("Error reading temperature");
lcd.setCursor(0, 0);
lcd.print("Sensor Error      ");
return;
}

if (temp < lowerThreshold) {
currentState = HEATING;
digitalWrite(LED_PIN, HIGH);
digitalWrite(BUZZER_PIN, LOW);
} else if (temp >= lowerThreshold && temp < targetTemp) {
currentState = STABILIZING;
digitalWrite(LED_PIN, HIGH);
digitalWrite(BUZZER_PIN, LOW);
} else if (temp >= targetTemp && temp < overheatTemp) {
currentState = TARGET_REACHED;
digitalWrite(LED_PIN, LOW);
digitalWrite(BUZZER_PIN, LOW);
} else if (temp >= overheatTemp) {
currentState = OVERHEAT;
digitalWrite(LED_PIN, LOW);
// Make buzzer beep every 500ms
digitalWrite(BUZZER_PIN, (millis() / 500) % 2);

}
```

```
// Display on LCD
lcd.setCursor(0, 0);
lcd.print("Temp:");
lcd.print(temp, 1);
lcd.print((char)223);
lcd.print("C ");

lcd.setCursor(9, 0);
lcd.print("Set:");
lcd.print(targetTemp, 1);
lcd.print("C");

lcd.setCursor(0, 1);
switch(currentState) {
case IDLE: lcd.print("State: IDLE     "); break;
case HEATING: lcd.print("State: HEATING  "); break;
case STABILIZING: lcd.print("State: STABILIZ "); break;
case TARGET_REACHED: lcd.print("State: TARGET   "); break;
case OVERHEAT: lcd.print("State: OVERHEAT "); break;
}

// Log to Serial
Serial.print("Temp: ");
Serial.print(temp);

Serial.print(" C | Target: ");

Serial.print(targetTemp);

Serial.print(" C | State: ");

switch(currentState) {

case IDLE: Serial.println("IDLE"); break;

case HEATING: Serial.println("HEATING"); break;

case STABILIZING: Serial.println("STABILIZING"); break;

case TARGET_REACHED: Serial.println("TARGET_REACHED"); break;

case OVERHEAT: Serial.println("OVERHEAT"); break;

}

delay(1000);

}
```

Testing Methodology:

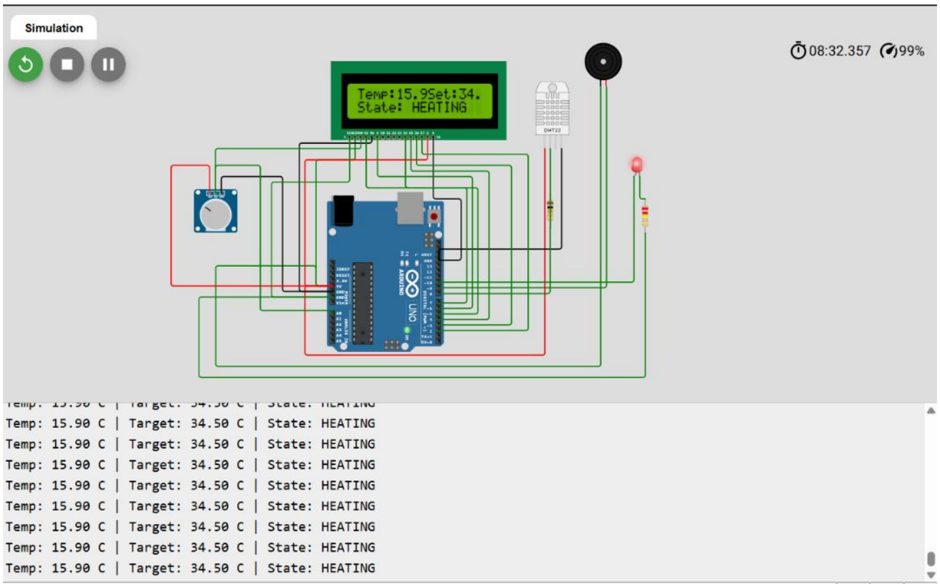
- The system was first tested using actual sensor readings from the DHT22 to ensure accurate temperature acquisition.
- To validate all system states, the DHT22 reading code was temporarily replaced with hardcoded temperature values for simulation:
  - 36.0 °C → For testing the OVERHEAT state
  - 28.5 °C → For testing the STABILIZING state
  - 30.0 °C → For testing the TARGET_REACHED state
  - 24.0 °C → For testing the HEATING state
- The LCD contrast was adjusted using a potentiometer for clear visibility of readings.

- The LED and buzzer were carefully observed to verify correct operation during each state transition.



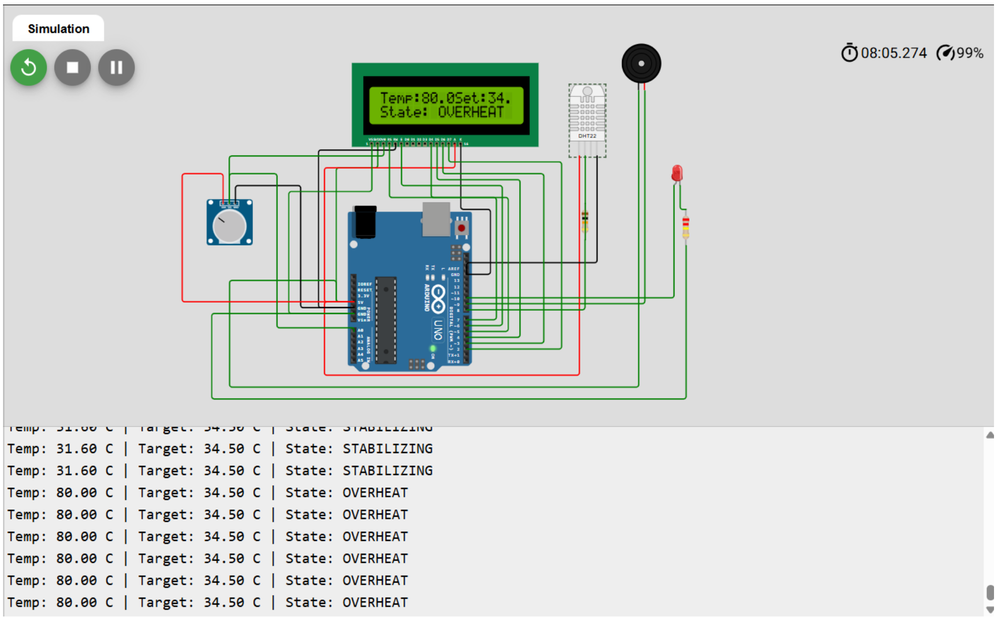
```
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
```

HEATING:



```
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
Temp: 15.90 C | Target: 34.50 C | State: HEATING
```

OVERHEAT:



```
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 31.60 C | Target: 34.50 C | State: STABILIZING
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
Temp: 80.00 C | Target: 34.50 C | State: OVERHEAT
```

Conclusion:

This heater control system project demonstrates a practical and complete embedded systems application, integrating sensors, actuator control, state tracking, and real-time monitoring. It successfully meets the requirements of the upliance.ai Embedded Systems Intern Assignment and is fully prepared for submission.

SIMULATION LINK: https://wokwi.com/projects/437464546492553217

github.com LINK: https://github.com/naveencheery4-creator/Basic-Heater-Control-System-using-Arduino-Uno-and-DHT22.git