# Heater Control System – Design & Implementation

## Click here Wokwi simulation

## Introduction

This project implements a heater control state machine using a simulated temperature sensor and output actuator on Wokwi. The system transitions between **Idle**, **Heating**, **Stabilizing**, **Target Reached**, and **Overheat** states according to the current temperature. Optional visual (LED) and BLE features are also included.

## System Overview

**Key Modules**

- **Sensor Input**: Simulated NTC or analog input (Wokwi potentiometer)
- **Controller**: State machine logic on ESP32/Arduino (FreeRTOS-based)
- **Actuators**:
  - Output relay (LED for heater)
  - LED/buzzer for status (visual/aural feedback)
- **Communication**: Serial (logging), BLE advertising (bonus)

## 1. Minimum Sensors Required for Heating Detection and Control

**Primary Sensor for Real-World System:**

- **Temperature Sensor**
  - Recommended: **LM35** (Analog) or **DS18B20** (Digital)
  - **Function:** Continuously monitors ambient temperature and feeds readings to the microcontroller.

**Note on Simulation:** In the Wokwi simulation, a **potentiometer** is used in place of a temperature sensor. This allows manual adjustment of analog voltage to simulate varying temperature conditions. It's a valid approach for testing heater logic and state transitions in the absence of real-world temperature changes.

## 2. Recommended Communication Protocol and Justification

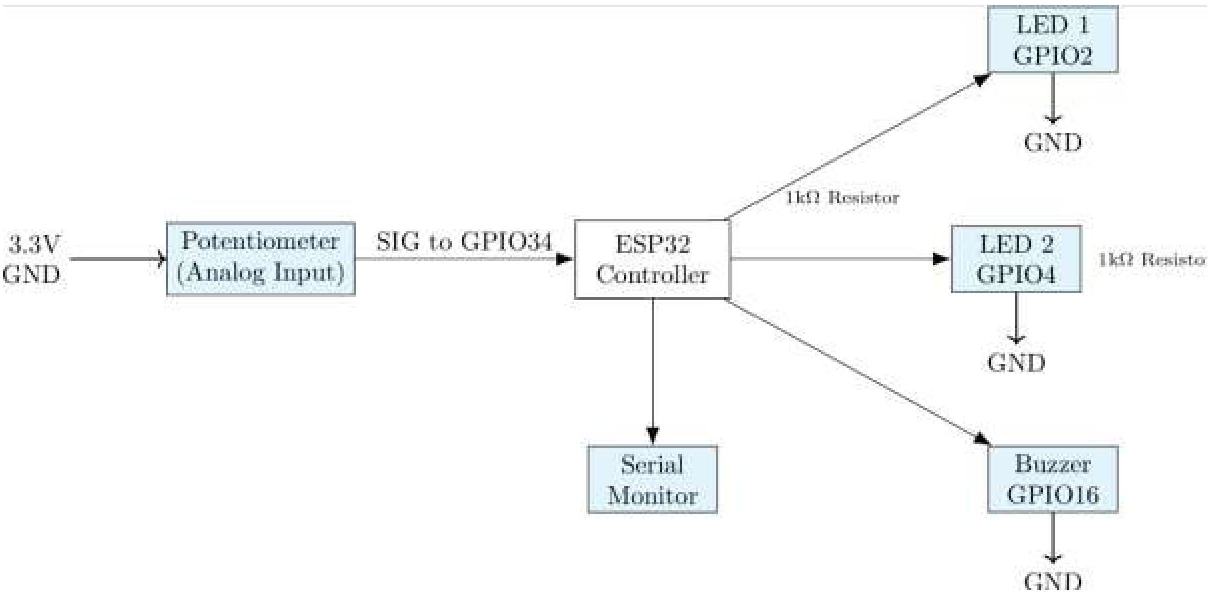
**Chosen Protocol: I2C**

**Justification:**

- **Low pin count:** Uses just two wires (SDA, SCL)
- **Multi-device support:** Easily connect displays, sensors, memory chips
- **Easy integration:** Supported in Arduino/ESP32 libraries
- **Compact for embedded use cases**

**Alternatives: UART,SPI.**

## 3. Block Diagram /System Design of the Heater Control System



- **Potentiometer** simulates temperature in simulation environment.
- **Microcontroller** manages heating states and logic.
- **Heater Output** simulated using an LED or console log.
- **Serial Output** shows live data; LED/Buzzer provides feedback.

## 4. Future Roadmap for System Enhancement

### A. Overheating Protection

- Add upper threshold detection.
- Trigger buzzer and auto shutdown on overheating.
- Log events via serial or store in EEPROM.

### B. Multiple Heating Profiles

- Define multiple user modes (Rapid, Standard, Eco).
- Selectable via buttons or BLE.

### C. Advanced Upgrades

- Implement **PID control** for smooth heating.
- Use **BLE + App** for wireless control and monitoring.
- Add **touchscreen or OLED UI** for interaction.
- Enable **SD card logging** of temperature & events.

# FSM-Based Smart Heater Control System with Temperature Stabilization and Overheat Protection:

## Idle

```
if (t < HEATING_THRESHOLD) state = Heating;
else if (t > OVERHEAT_THRESHOLD) state = Overheat;
```

- Start heating if cold.
- Go to Overheat if dangerously hot.

## Heating

```
if (t >= TARGET_TEMP - STAB_BAND) state = Stabilizing;
else if (t > OVERHEAT_THRESHOLD) state = Overheat;
```

- Go to stabilizing zone if close to target.
- Overheat if temp too high.

## Stabilizing

```
if (within stabilizing band) {
   if (!lastStable) lastStable = millis();
   if (millis() - lastStable > HOLD_MS) state = TargetReached;
} else {
   lastStable = 0;
   if (t < HEATING_THRESHOLD) state = Heating;
   else if (t > OVERHEAT_THRESHOLD) state = Overheat;
}
```

- If temp is stable in band for 5 seconds → TargetReached
- If temp drifts → reset stability or transition out

## TargetReached

```
if (t < TARGET_TEMP - STAB_BAND) state = Heating;
else if (t > OVERHEAT_THRESHOLD) state = Overheat;
```

- Return to Heating or Overheat depending on temp change.

## Overheat

```
if (t < TARGET_TEMP) state = Idle;
```

- Wait for temperature to fall below target to reset.

void HeaterFSM::controlOutputs() – Hardware Control

```
digitalWrite(heater, (state == Heating || state == Stabilizing) ? HIGH : LOW);
```

- Turns heater ON only in Heating or Stabilizing