# Basic Heater Control System – upliance.ai Embedded Systems Intern Assignment

Name : R.Johnson
Date : 14.08.2025
Submitted for : upliance.ai Embedded Systems Internship

## Introduction

This project designs a basic heater control system using a temperature sensor and microcontroller to maintain a target temperature. The heater switches ON or OFF based on preset thresholds, ensuring safety and energy efficiency.

## Sensors Required

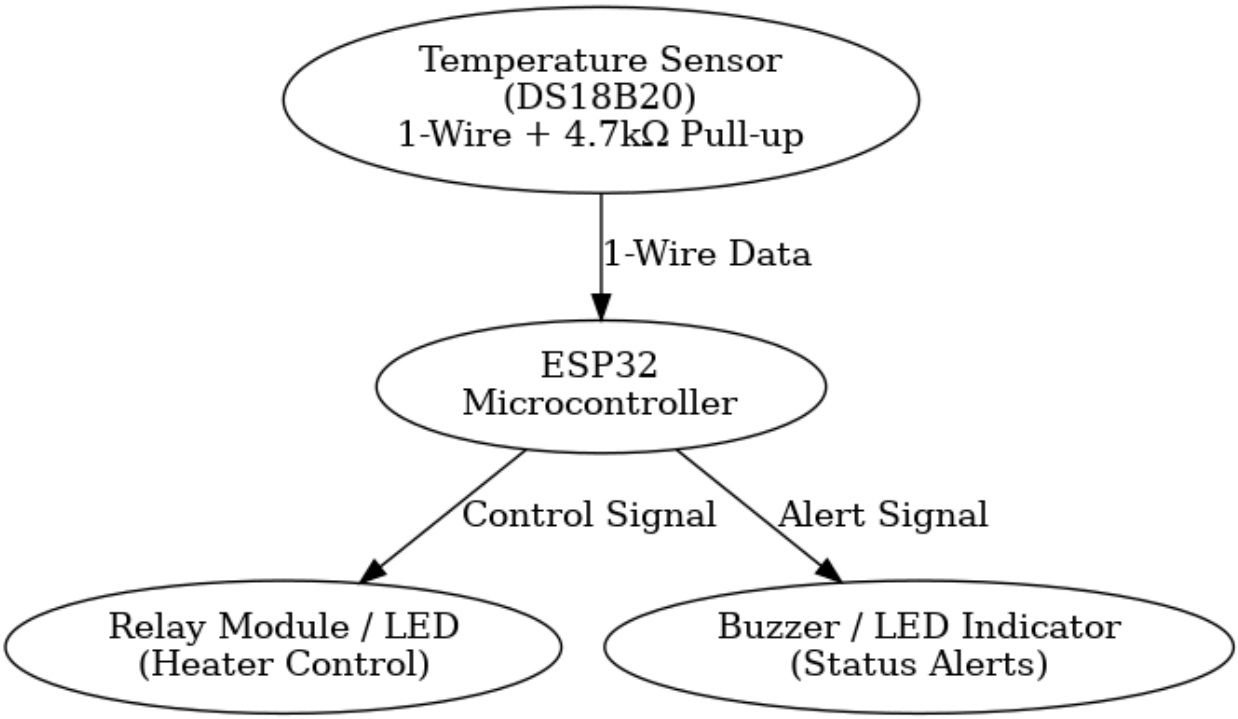
Temperature Sensor – DS18B20  Requires  ~4.7kΩ pull-up on DATA line (DQ→VCC).

- Microcontroller – ESP32 DevKit V1 (GPIO, Serial, BLE/Wi-Fi capable).
- Heater Control – Relay module or LED (simulation) for ON/OFF control.
- Indicators – Status LED (visual) and Buzzer (audio) for alerts

## Communication Protocol

1-Wire (for DS18B20). Single data line; supports multiple sensors on the same bus; requires ~4.7kΩ pull-up on DATA (DQ→VCC). DS18B20 is not I²C

## Block Diagram



## Future Roadmap

1 Add Overheat Protection – automatic shutdown if temperature exceeds safe limit.
2 Implement Multiple Heating Profiles – low, medium, high modes.
3 Integrate BLE/Wi-Fi Monitoring – remote monitoring & control.
4 Add Data Logging – store temperature history.

## Applications

- Room heater, Incubator, HVAC prototype.
- Industrial temperature monitoring & control.
- Smart home appliances integration.

## Advantages

- Single-wire wiring (1-Wire) → simple setup; supports multiple sensors on one bus.
- Accurate digital readings; DS18B20 unique 64-bit ID for multi-sensor systems.
- Low pin count on MCU; strong ESP32 library support (OneWire + DallasTemperature).

## Safety Features

- Overheat cutoff: >60°C → Heater OFF + buzzer alert.
- Sensor fault handling: ignore 85°C (no conversion/power-up default) and −127°C (bus/wiring fault); fail-safe OFF + retry.
- Required pull-up: ~4.7kΩ on DS18B20 DATA (DQ→VCC) for reliable 1-Wire bus idle high.

## Conclusion

This basic heater control system is designed to operate efficiently with minimal components while ensuring temperature regulation and safety. By integrating future features such as overheat protection, multiple heating profiles, and wireless monitoring, the system can be enhanced for real-world applications, providing both safety and convenience to users.

# PRACTICAL IMPLEMENTATION

**Wokwi Simulation Link: https://wokwi.com/projects/439282182251172865**

This project uses an ESP32 microcontroller with a DS18B20 temperature sensor, two LEDs, and a buzzer for state indication and overheat alerts. The simulation demonstrates the state changes based on temperature thresholds.

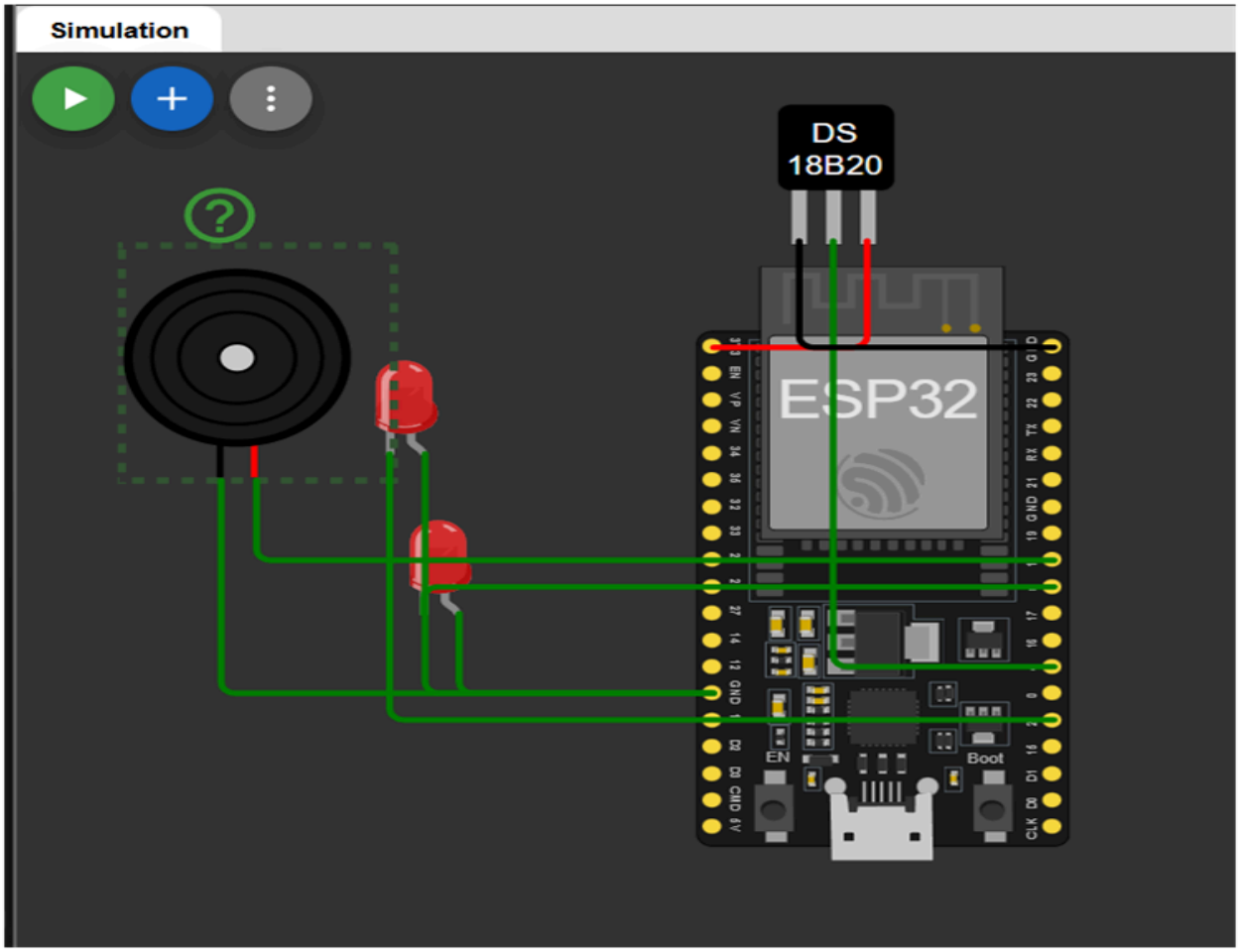
## Circuit Diagram - ESP32 Heater Control with DS18B20

This circuit diagram shows the connection of an ESP32 microcontroller with the following components:

1. DS18B20 Temperature Sensor
2. Two LEDs for indicating states
3. Active Buzzer for overheating alert

### Connections:

- DS18B20 connected to GPIO 4 (Data), 3.3V (VCC), and GND.
- LED1 and LED2 connected to GPIO pins (with series resistors in practical setup).
- Active Buzzer connected to a digital GPIO pin and GND.




### Working Principle

The ESP32 microcontroller is connected to a DS18B20 temperature sensor.
The system continuously reads the temperature from the sensor and controls the heater, LEDs, and buzzer based on the temperature range.

## Operation:

- Temperature < 45°C → Heater ON, LED1 ON, LED2 OFF, Buzzer OFF.
- Temperature 45°C to 70°C → Heater OFF, LED1 OFF, LED2 ON, Buzzer OFF.
- Temperature ≥ 70°C → Heater OFF, LED1 OFF, LED2 ON, Buzzer ON (overheat alert).
  This logic ensures that the system maintains the desired temperature and provides an immediate alert in case of overheating.

## Arduino Code

```
#include <OneWire.h>

#include <DallasTemperature.h>



// Pin connections

#define ONE_WIRE_BUS 4    // DS18B20 data pin

#define HEATER_LED 5      // Heater LED

#define STATUS_LED 19     // Status LED

#define BUZZER_PIN 18     // Buzzer



// Temperature limits

#define TARGET_TEMP 45

#define OVERHEAT_TEMP 70



OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature sensors(&oneWire);



void setup() {

  Serial.begin(115200);

  sensors.begin();



  pinMode(HEATER_LED, OUTPUT);

  pinMode(STATUS_LED, OUTPUT);
```

```
  pinMode(BUZZER_PIN, OUTPUT);

  Serial.println("=== Basic Heater Control (ESP32 + DS18B20) ===");
}

void loop() {
  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);

  Serial.print("Temp = ");
  Serial.print(tempC);
  Serial.print(" °C, ");

  if (tempC < TARGET_TEMP) {
    Serial.println("state=HEATING, heater=ON");
    digitalWrite(HEATER_LED, HIGH);
    digitalWrite(STATUS_LED, LOW);
    digitalWrite(BUZZER_PIN, LOW);
  }
  else if (tempC >= TARGET_TEMP && tempC < OVERHEAT_TEMP) {
    Serial.println("state=TARGET_REACHED, heater=OFF");
    digitalWrite(HEATER_LED, LOW);
    digitalWrite(STATUS_LED, HIGH);
```

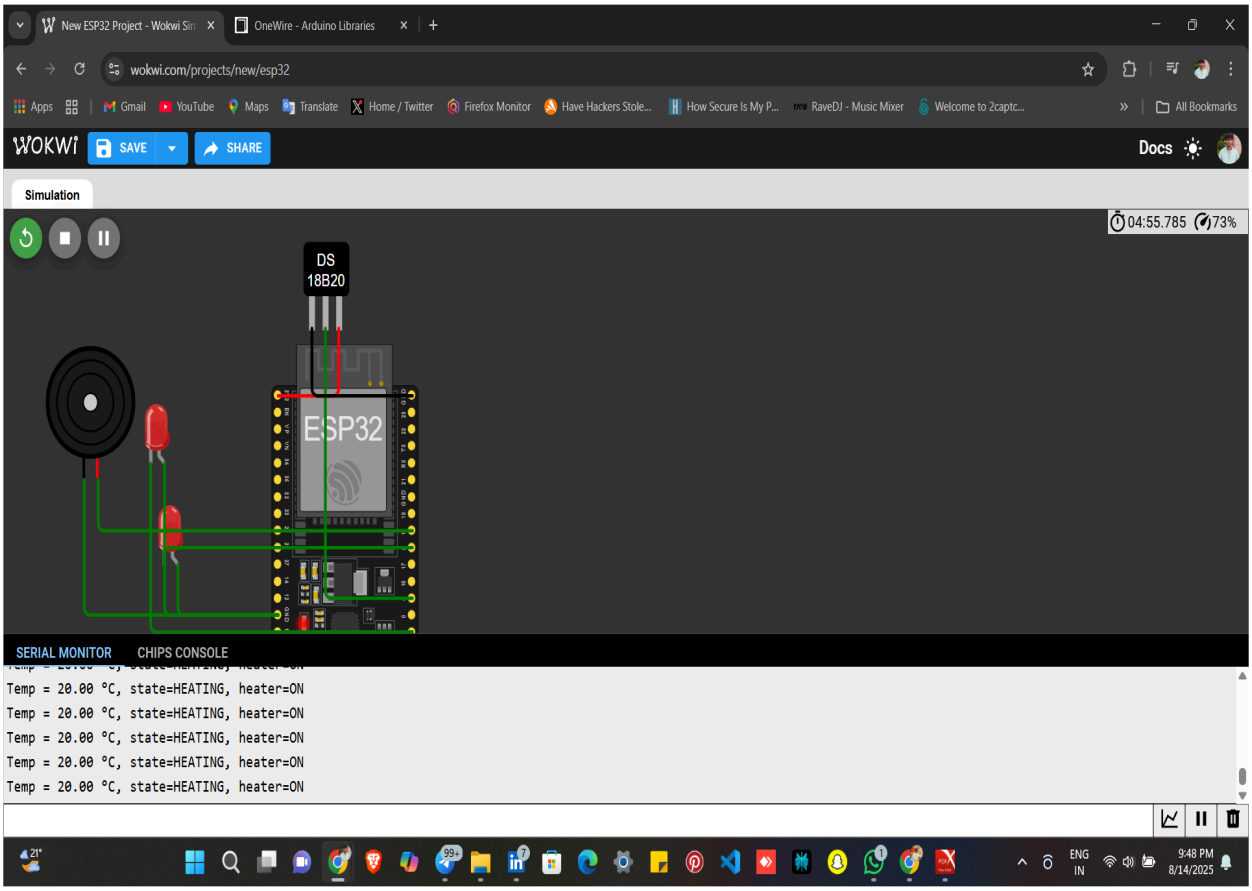
```
    digitalWrite(BUZZER_PIN, LOW);
  }
  else if (tempC >= OVERHEAT_TEMP) {
    Serial.println("state=OVERHEAT, buzzer=ON");
    digitalWrite(HEATER_LED, LOW);
    digitalWrite(STATUS_LED, HIGH);
    digitalWrite(BUZZER_PIN, HIGH);
  }


  delay(1000);
}
```

## Output

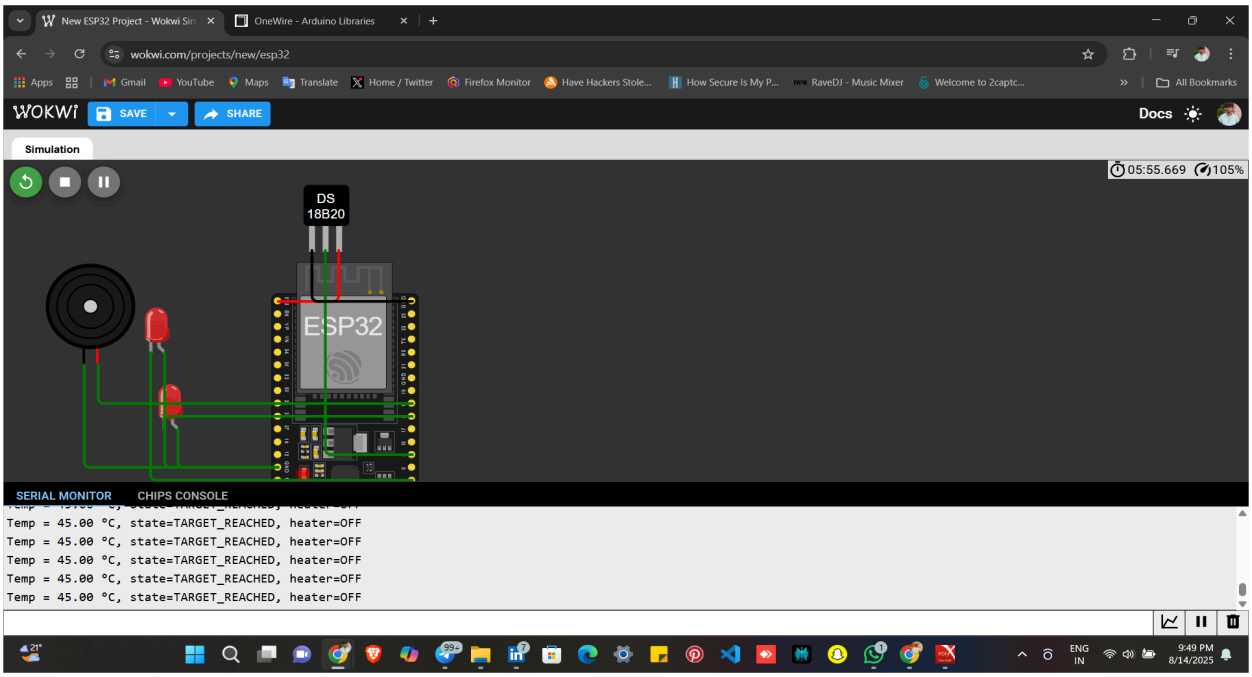
20°C Test

- Heater ON, LED1 ON, LED2 OFF, Buzzer OFF.



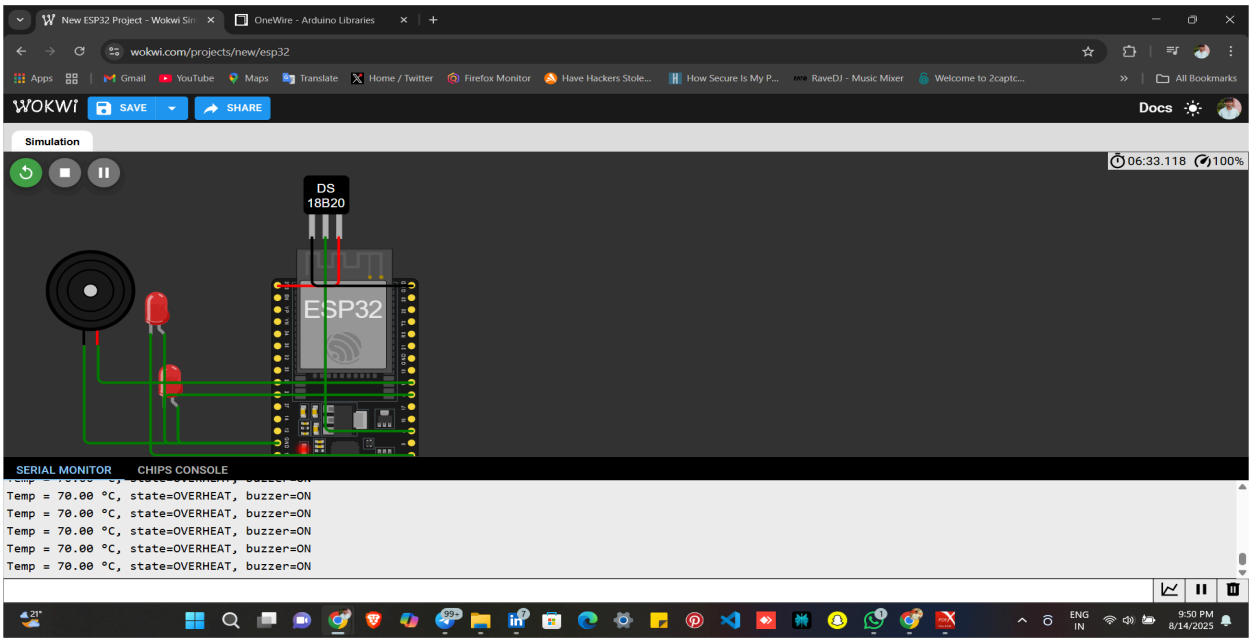

45°C Test

- Heater OFF, LED1 OFF, LED2 ON, Buzzer OFF.




70°C Test

- Heater OFF, LED1 OFF, LED2 ON, Buzzer ON (Overheat alert).




## Conclusion

The ESP32-based temperature control system using the DS18B20 sensor successfully monitors temperature and controls the heater, LEDs, and buzzer according to preset

thresholds.

The system turns ON the heater below 45 °C, switches to the status LED between 45 °C and 70 °C, and activates the buzzer above 70 °C to indicate overheating.

This ensures safe operation and prevents thermal damage.