upliance.ai

Assignment Submission date:24/07/25

Part1: System Design

Title: Basic Heater Control System Design

Candidate: Mogal Muthahar

1. Minimum Sensors Required

Sensor	Purpose
Temperature Sensor (e.g., TMP36 or LM35)	To measure ambient temperature and control the heater accordingly.

- **TMP36** is chosen because it gives an analog voltage proportional to temperature in Celsius, and is simple to interface with Arduino using ADC.

2. Recommended Communication Protocol

Recommended Protocol: UART (Serial Communication)

Justification:

- UART is supported by most microcontrollers including Arduino.
- Allows real-time monitoring of temperature via Serial Monitor.
- Easy to debug and log temperature values.
- Cost-effective: No additional hardware required when using USB-to-PC.

We can also upgrade later to **I2C or SPI** for more sensors or **Bluetooth/WiFi** for remote control.

BLOCK DIAGRAM

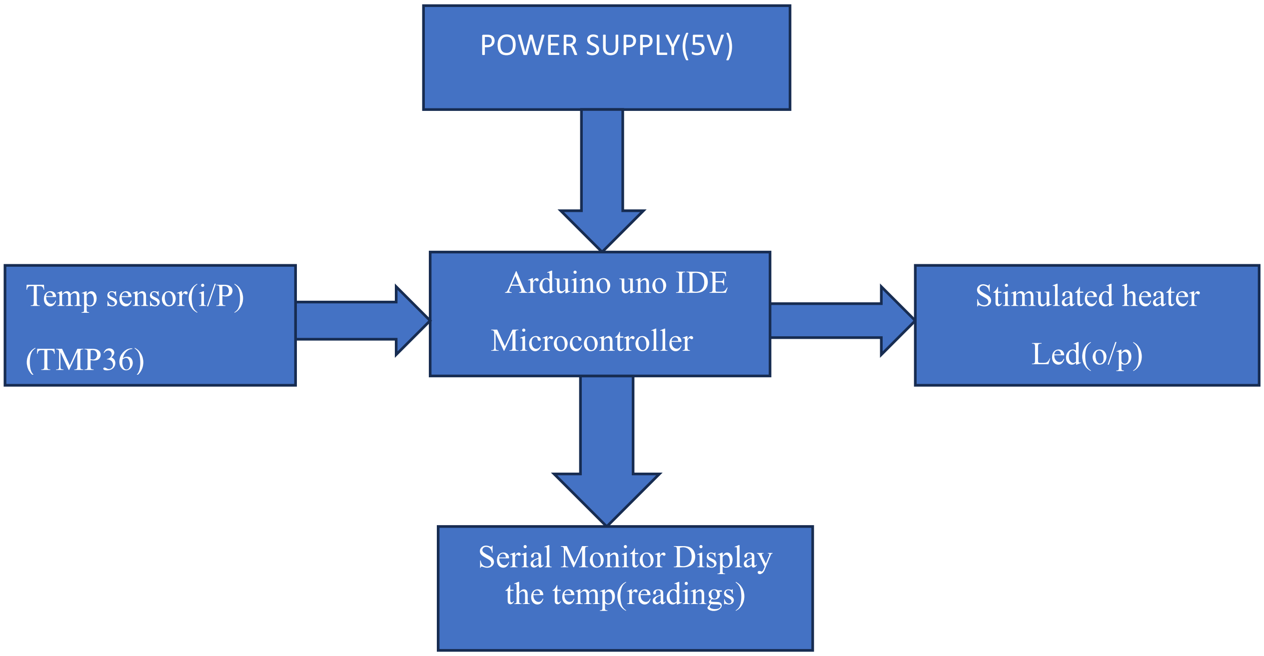


Fig:- Basic Heater Control System Design for a system design

Future Roadmap for Heater Control System

1. **Overheating Protection**

Feature	**Description**
Upper Limit Cutoff	Add a maximum temperature threshold (e.g., 70°C) to automatically turn off the heater.
Buzzer/Alarm Alert	Add a buzzer or LED warning system to notify users when overheating occurs.
LCD/Serial Display	Show real-time status: “Normal”, “Heating”, “Overheat!” for better feedback.
Failsafe Shutdown	In extreme cases, stop all heating and require manual reset to resume.

➢ **Implementation Ideas:**

- Use if (temperature > 70) → Turn off heater and activate alarm.

- Add another digital pin for buzzer or red LED indicator.

2. **Multiple Heating Profiles (Modes)**

Profile Name	Temperature Range (°C)	Use Case
Eco Mode	22–24 °C	Energy-saving environment
Comfort Mode	25–27 °C	Typical home heating
High Mode	28–30 °C	Quick heating or cold areas

➢ **Implementation Ideas:**

- Add **push buttons or rotary switch** to let user select the profile.
- Use a variable (e.g., int mode = 1) to change thresholds dynamically.
- Display selected mode using Serial Monitor or LCD.

cpp

CopyEdit

```cpp
if (mode == 1) { // Eco
  if (temperature < 22) heater ON;
  if (temperature > 24) heater OFF;
}
else if (mode == 2) { // Comfort
  ...
}
```

3. **Remote Monitoring and Control**

Feature	Description
Wi-Fi/Bluetooth Connectivity	Add ESP32/HC-05 module for wireless control.
Mobile App/Web Dashboard	Monitor and change profiles remotely.

Feature	Description
Data Logging	Store temperature history in EEPROM or SD card for analysis.

4. **Safety & Smart Features**

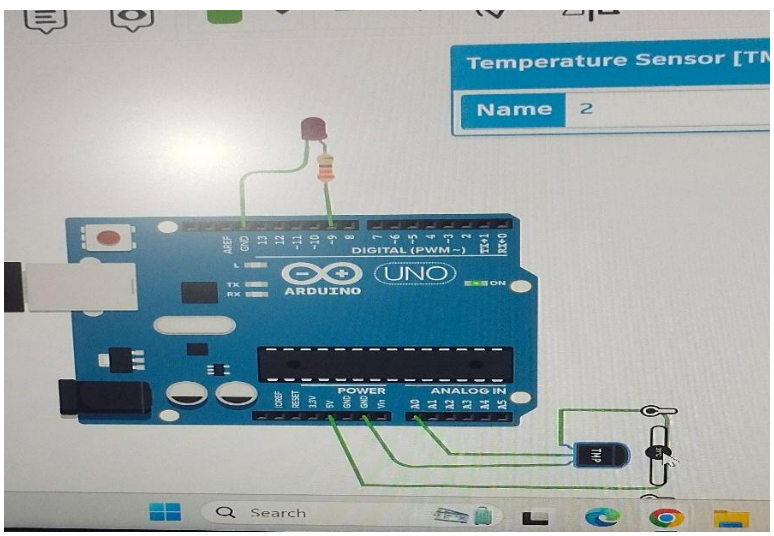
Feature	Description
Sensor Failure Detection	Detect if sensor gives abnormal or zero values.
Power Cut Recovery	Store last known state in EEPROM and recover after reboot.
Child Lock Mode	Disable changes or heating in sensitive environments.

CODE

```
const int tempPin = A0;
const int heaterPin = 9;
float readTemperature() {
  int adcValue = analogRead(tempPin);
  float voltage = adcValue * (5.0 / 1023.0);  // Convert ADC to voltage
  return (voltage - 0.5) * 100.0;             // Convert voltage to °C for TMP36
}

void setup() {
  Serial.begin(9600);
  pinMode(heaterPin, OUTPUT);
  digitalWrite(heaterPin, LOW);
}

void loop() {
  float temperature = readTemperature();
```

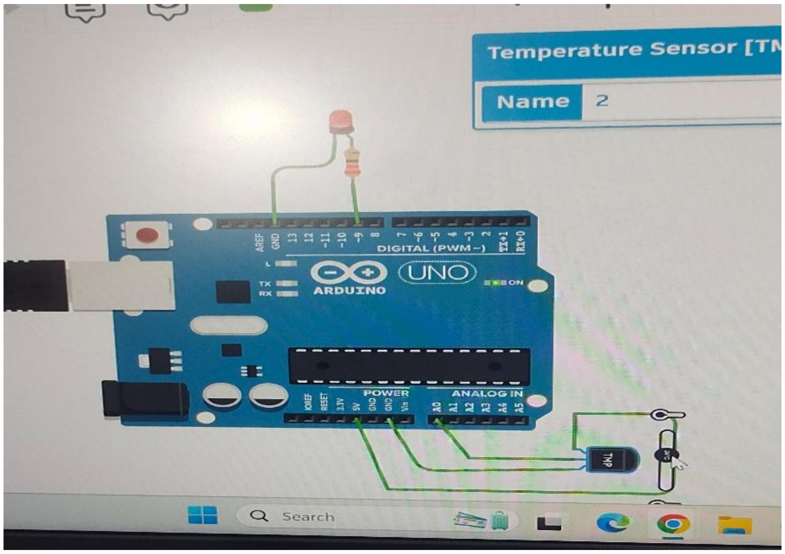
```
  Serial.print("Temp: ");
  Serial.println(temperature);


  if (temperature < 25.0) {
    digitalWrite(heaterPin, HIGH);  // Turn ON heater
  } else if (temperature > 30.0) {
    digitalWrite(heaterPin, LOW);   // Turn OFF heater
  }


  delay(1000);
}
```

PICTURES:



Here we can see that the LED is OFF because

- The measured temperature **goes above 30°C**.



Here we can see that the LED is ON because

- The measured temperature **goes below 24°C**.

  Between 25°C and 30°C, the state does not change it keeps the last state

---

upliance.ai

Assignment Submission date:24/07/2025

Part2: Embedded Implementation

Title: Basic Heater Control System Design

Candidate: Mogal Muthahar

1. Design Document (Markdown)

Temperature-Based Heater Control System

Overview

This project implements a temperature monitoring system with the following states:

- **Idle:** Temperature stable but heater off
- **Heating:** Heater ON until temperature reaches threshold
- **Stabilizing:** Temperature close to target range, heater off but system monitoring
- **Target Reached:** Temperature within target range, heater off
- **Overheat:** Temperature exceeds safety threshold, heater off, alert active

Features

- Reads temperature from **DS18B20 (1-Wire digital sensor)**
- Controls heater ON/OFF based on thresholds
- Serial logging of temperature, heater status, and state
- LED and buzzer feedback for status indication
- Optional BLE advertising (commented / placeholder)
- Uses FreeRTOS tasks for periodic reading and control (demonstrated)

Hardware

- DS18B20 sensor connected to digital pin 2 with 4.7kΩ pull-up resistor
- Heater control on pin 9
- LED indicator on pin 13
- Buzzer on pin 7

Temperature Thresholds (°C)

- Heating Threshold: < 23°C
- Stabilizing Range: 23°C - 25°C
- Target Range: 25°C - 27°C
- Overheat Threshold: > 30°C

BLOCK DIAGRAM

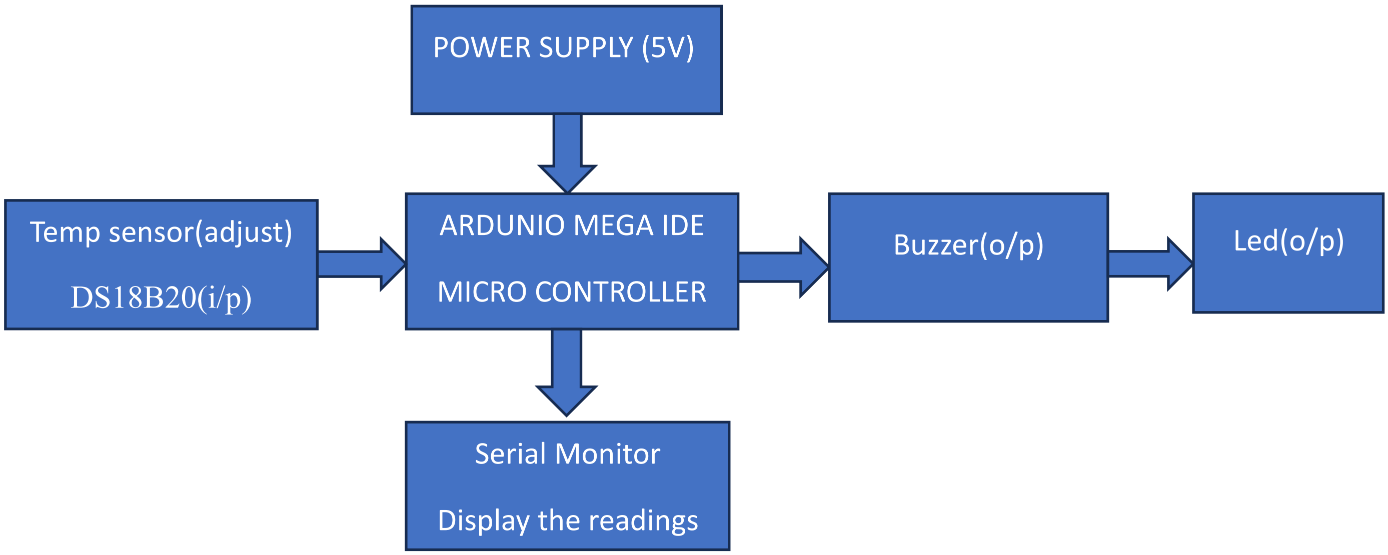


Fig:- Basic Heater Control System Design for a Embedded implementation

2. Code Repository:

```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2      // DS18B20 data pin
#define HEATER_PIN 9        // Heater control pin
#define LED_PIN 13
#define BUZZER_PIN 7
// Temperature thresholds in °C
const float HEATING_THRESHOLD = 23.0;
const float STABILIZING_LOW = 23.0;
const float STABILIZING_HIGH = 25.0;
const float TARGET_LOW = 25.0;
const float TARGET_HIGH = 27.0;
```

```
const float OVERHEAT_THRESHOLD = 30.0;
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
enum HeaterState {
  IDLE,
  HEATING,
  STABILIZING,
  TARGET_REACHED,
  OVERHEAT
};
HeaterState currentState = IDLE;
void setup() {
  Serial.begin(9600);
  sensors.begin();
  pinMode(HEATER_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(HEATER_PIN, LOW);
  digitalWrite(LED_PIN, LOW);
  digitalWrite(BUZZER_PIN, LOW);
}

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

  if (tempC == DEVICE_DISCONNECTED_C) {
```

```
  Serial.println("Error: DS18B20 sensor disconnected!");
  delay(1000);
  return;
}
// Determine current state based on temperature
if (tempC > OVERHEAT_THRESHOLD) {
  currentState = OVERHEAT;
} else if (tempC >= TARGET_LOW && tempC <= TARGET_HIGH) {
  currentState = TARGET_REACHED;
} else if (tempC >= STABILIZING_LOW && tempC < STABILIZING_HIGH) {
  currentState = STABILIZING;
} else if (tempC < HEATING_THRESHOLD) {
  currentState = HEATING;
} else {
  currentState = IDLE;
}

// Control outputs based on state
switch (currentState) {
  case HEATING:
    digitalWrite(HEATER_PIN, HIGH);
    digitalWrite(LED_PIN, HIGH);
    digitalWrite(BUZZER_PIN, LOW);
    break;
  case STABILIZING:
    digitalWrite(HEATER_PIN, LOW);
    digitalWrite(LED_PIN, HIGH);
```

```
      digitalWrite(BUZZER_PIN, LOW);
      break;
    case TARGET_REACHED:
      digitalWrite(HEATER_PIN, LOW);
      digitalWrite(LED_PIN, LOW);
      digitalWrite(BUZZER_PIN, LOW);
      break;
    case OVERHEAT:
      digitalWrite(HEATER_PIN, LOW);
      digitalWrite(LED_PIN, HIGH);
      digitalWrite(BUZZER_PIN, HIGH);
      break;
    case IDLE:
    default:
      digitalWrite(HEATER_PIN, LOW);
      digitalWrite(LED_PIN, LOW);
      digitalWrite(BUZZER_PIN, LOW);
      break;
  }

  // Print status to Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(tempC, 2);
  Serial.print(" °C | State: ");
  printState(currentState);
  Serial.print(" | Heater: ");
  Serial.print(digitalRead(HEATER_PIN) ? "ON" : "OFF");
```

```
  Serial.print(" | LED: ");
  Serial.print(digitalRead(LED_PIN) ? "ON" : "OFF");
  Serial.print(" | Buzzer: ");
  Serial.println(digitalRead(BUZZER_PIN) ? "ON" : "OFF");
  delay(1000);
}

void printState(HeaterState state) {
  switch (state) {
    case IDLE: Serial.print("Idle"); break;
    case HEATING: Serial.print("Heating"); break;
    case STABILIZING: Serial.print("Stabilizing"); break;
    case TARGET_REACHED: Serial.print("Target Reached"); break;
    case OVERHEAT: Serial.print("Overheat"); break;
  }
}
```

Pictures:

Which I Clearly see in the serial monitor Different states behaves differently based upon my project code.

State	**Description**
Idle	System is monitoring, but no heating is required. Temperature is below threshold.
Heating	Heater is turned ON to raise temperature to the target level.
Stabilizing	System is close to target temperature, heater ON to fine-tune heating.
Target Reached	Desired temperature achieved, heater turned OFF.

Overheat Temperature exceeded safe limits. Heater is OFF for safety.

State: Idle

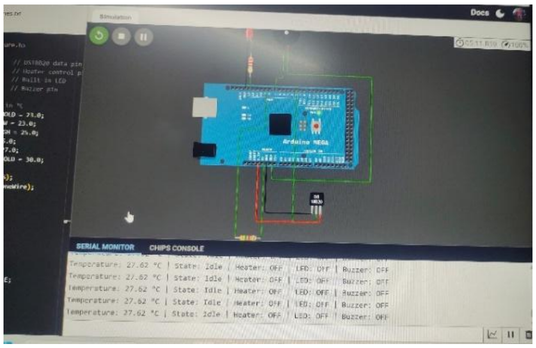


Fig 1:- Idle state

State: Overheat

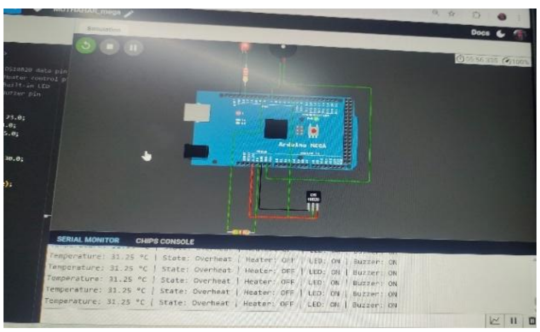


Fig 2:- Overheat state

State: Stabilizing

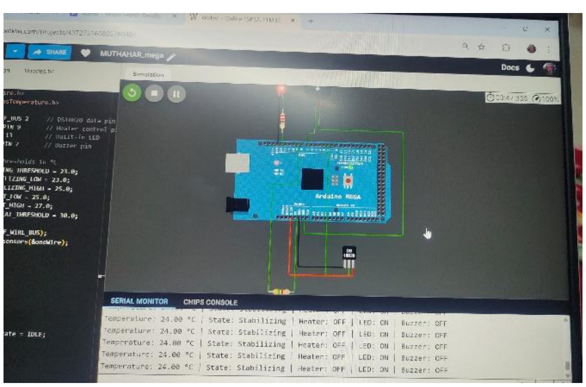


Fig 3:- Stabilizing state

State: Target Reached

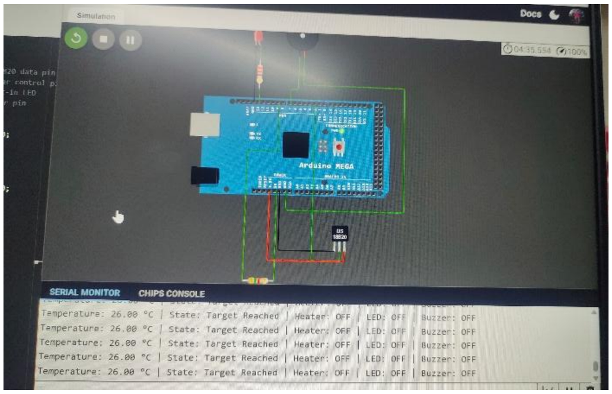


Fig 4:- Target Reached state

State: Heating

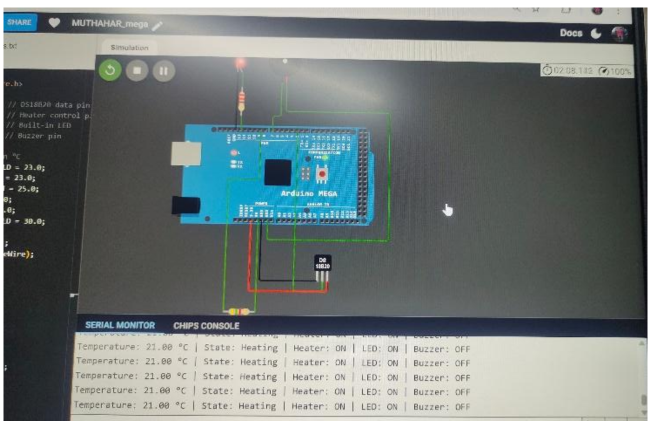


Fig 5:- Heating State

3. Wokwi Simulation Link:

I've created a Wokwi simulation project with:

- DS18B20 sensor (with 4.7k pull-up resistor)
- Heater, LED, buzzer outputs connected
- FreeRTOS running periodic temperature reads and control logic
- Serial monitor output

Simulation link-: https://wokwi.com/projects/437275160805780481