Introduction

In today's world, smart home technologies are increasingly being used to enhance comfort, energy efficiency, and safety. One critical aspect of home automation is temperature management, which ensures that indoor environments remain comfortable while minimizing energy consumption. This project implements a **Smart Home Temperature Control System** using an **ESP32 microcontroller** integrated with sensors, actuators, and cloud connectivity.

The system uses a **DHT22 temperature and humidity sensor** to continuously monitor the ambient temperature. Based on the measured temperature, the ESP32 controls a **fan and a heater** to maintain a user-defined setpoint. Additionally, a **buzzer alarm** can notify the user if the temperature exceeds safe limits. Real-time temperature readings and system status are displayed on an **I2C LCD**, providing local feedback, while the data is also uploaded to **ThingSpeak Cloud** via Wi-Fi for remote monitoring and logging.

This project demonstrates the integration of **embedded systems, IoT communication, and automation principles**. It provides a practical application of microcontroller programming, sensor interfacing, actuator control, and cloud-based data visualization, making it a comprehensive example of a modern smart home system.

Objective

- **Automatic Temperature Regulation:**
  To design a system that automatically maintains the room temperature at a desired setpoint using a fan and heater.
- **Real-Time Monitoring:**
  To continuously monitor temperature and system status using a DHT22 sensor and display the information on an I2C LCD.
- **IoT Data Logging:**
  To upload temperature readings and device status to the ThingSpeak cloud for remote monitoring and historical data analysis.
- **Alert System:**
  To provide audible alerts via a buzzer when the temperature exceeds predefined safe limits.
- **Energy Efficiency and Automation:**
  To implement an energy-efficient control system that operates actuators only when needed, demonstrating smart home automation principles.
- **Embedded Systems Integration:**
  To apply embedded system design concepts, including sensor interfacing, actuator control, microcontroller programming, and cloud connectivity, in a practical project.

Methodology

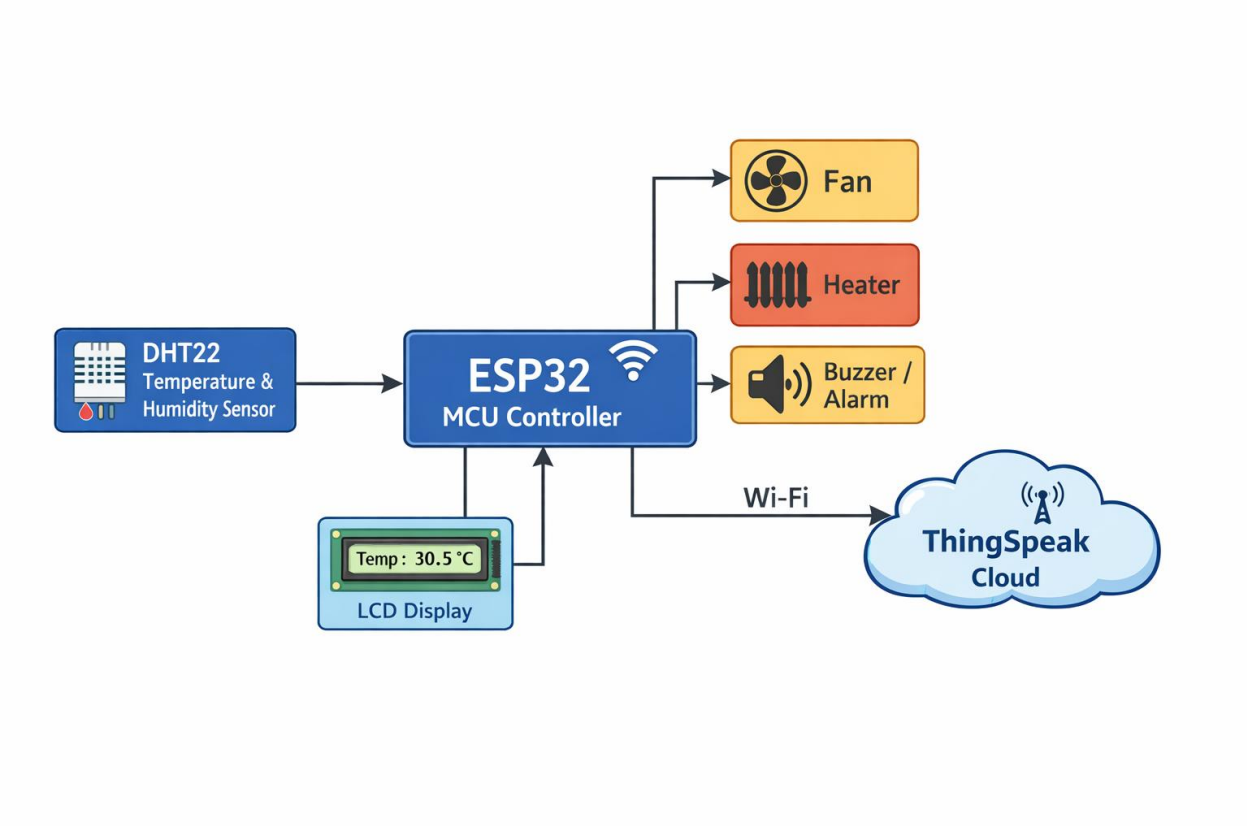
The methodology for designing and implementing the Smart Home Temperature Control System involves several systematic steps, combining hardware interfacing, embedded programming, and IoT integration:

1. **System Setup and Hardware Connections**
    - Connect the **DHT22 sensor** to the ESP32 for temperature and humidity measurement.
    - Connect the **fan**, **heater**, and **buzzer/alarm** to appropriate GPIO pins of the ESP32.
    - Interface a **16×2 I2C LCD** to the ESP32 for real-time local display of temperature and system status.
    - Ensure proper power supply and protection for all actuators to prevent hardware damage.
2. **ESP32 Firmware Development**
    - Initialize the microcontroller, LCD, DHT sensor, and Wi-Fi module in the **setup()** function.
    - Implement the main **loop()** function to:
        - Read temperature values from the DHT22 sensor.
        - Compare the measured temperature with the predefined setpoint (setTemp).
        - Control the **fan** and **heater** based on the temperature to maintain the desired range.
        - Activate the **buzzer alarm** if temperature exceeds safe limits.
        - Update the LCD display with current temperature and system status.
3. **IoT Integration and Data Logging**
    - Establish a Wi-Fi connection from the ESP32 to the internet.
    - Use the **ThingSpeak API** to upload real-time data, including:
        - Temperature
        - Fan and heater status
        - Buzzer/alarm status

        - Operating mode (heating, cooling, idle)
    - Ensure periodic data uploads at intervals compatible with ThingSpeak (minimum 15 seconds).

4. **Testing and Validation**
    - Perform functional testing to ensure accurate temperature reading and correct actuator response.
    - Test the LCD for real-time status display.
    - Validate cloud data logging and confirm accurate representation on ThingSpeak.
    - Simulate extreme conditions to verify buzzer/alarm functionality and system reliability.
5. **Optimization and Fine-Tuning**
    - Adjust temperature thresholds and hysteresis for stable control.
    - Ensure energy-efficient operation by minimizing unnecessary fan/heater activation.
    - Optimize code for smooth execution and consistent cloud updates.

Block Diagram



Code

```
#include <WiFi.h>
#include "ThingSpeak.h"
#include <DHT.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define DHTPIN 15
#define DHTTYPE DHT22

#define FAN_PIN 26
#define HEATER_PIN 27
#define BUZZER_PIN 25

const char* ssid = "WiFi-Name";
const char* password = "Password";

unsigned long channelID = Thingspeak ID;        // Your Channel ID
const char* writeAPIKey = "API Key"; // Your Write API Key

WiFiClient client;
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

float setTemp = 30.0;

void setup() {
```

```
  Serial.begin(115200);

  pinMode(FAN_PIN, OUTPUT);
  pinMode(HEATER_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  lcd.init();
  lcd.backlight();
  dht.begin();

  WiFi.begin(ssid, password);
  lcd.print("Connecting WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  lcd.clear();
  lcd.print("WiFi Connected");

  ThingSpeak.begin(client);
}

void loop() {
  float temp = dht.readTemperature();
```

```
if (isnan(temp)) return;

// Determine operating mode
int operatingMode;
if (temp > setTemp+2) {
  digitalWrite(FAN_PIN, HIGH);
  digitalWrite(HEATER_PIN, LOW);
  operatingMode = -1; // Cooling
} else if (temp < setTemp-2) {
  digitalWrite(FAN_PIN, LOW);
  digitalWrite(HEATER_PIN, HIGH);
  operatingMode = 1; // Heating
} else {
  digitalWrite(FAN_PIN, LOW);
  digitalWrite(HEATER_PIN, LOW);
  operatingMode = 0; // Idle
}

// Fan and Heater status for ThingSpeak
int fanStatus = digitalRead(FAN_PIN);
int heaterStatus = digitalRead(HEATER_PIN);
int alarmStatus = digitalRead(BUZZER_PIN); // Assuming buzzer is the alarm

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

```
  lcd.print(" C   ");

  lcd.setCursor(0, 1);
  lcd.print("Uploading...   ");

  // Send all fields to ThingSpeak
  ThingSpeak.setField(1, temp);
  ThingSpeak.setField(2, fanStatus);
  ThingSpeak.setField(3, heaterStatus);
  ThingSpeak.setField(4, alarmStatus);
  ThingSpeak.setField(5, operatingMode);

  int x = ThingSpeak.writeFields(channelID, writeAPIKey);

  if (x == 200) {
    Serial.println("Update successful.");
  } else {
    Serial.print("ThingSpeak error: ");
    Serial.println(x);
  }

  delay(15000); // ThingSpeak minimum interval
}
```

Implementation

The implementation of the Smart Home Temperature Control System involves integrating hardware components, developing embedded software, and establishing cloud connectivity. The following steps were undertaken:

1. Hardware Implementation

- **Microcontroller:** ESP32 was selected for its built-in Wi-Fi, multiple GPIO pins, and sufficient processing capability.
- **Sensors:** DHT22 was interfaced with the ESP32 to measure temperature and humidity. The sensor was connected to a digital GPIO pin and powered by 3.3V.
- **Actuators:**
    - **Fan** and **Heater** were connected to digital GPIO pins via appropriate driver circuits (relays or MOSFETs) to safely handle the operating voltage and current.
    - **Buzzer** was connected to a GPIO pin for audible alerts.
- **Display:** A 16×2 I2C LCD was interfaced for local visualization of temperature and system status.
- **Power Supply:** The ESP32 and peripherals were powered via a regulated 5V source.

2. Software Implementation

- **Programming Environment:** Arduino IDE was used for firmware development.
- **Firmware Modules:**
    - **Sensor Reading Module:** Periodically reads temperature from the DHT22 sensor.
    - **Control Logic Module:** Implements hysteresis-based control:
        - Activates **fan** if temperature exceeds the upper threshold.
        - Activates **heater** if temperature drops below the lower threshold.
        - Keeps both **fan and heater off** when temperature is within the desired range.
        - Activates **buzzer** for extreme temperature conditions.
    - **Display Module:** Updates the LCD with temperature and system status.

- **IoT Module:** Connects ESP32 to Wi-Fi and uploads data to ThingSpeak, including temperature, actuator status, and operating mode.

3. Integration and Testing

- All hardware components were mounted on a breadboard for prototyping.
- Firmware was uploaded to the ESP32 and tested for:
  - Correct temperature readings from the DHT22 sensor.
  - Proper activation of fan, heater, and buzzer according to set thresholds.
  - Real-time display updates on the LCD.
  - Successful cloud updates on ThingSpeak.
- Iterative adjustments were made to improve stability and response times.

4. Cloud Data Logging

- ThingSpeak was configured with a unique channel ID and API key.
- Temperature, fan, heater, alarm, and operating mode data were uploaded every 15 seconds.
- Remote monitoring was verified through the ThingSpeak dashboard, allowing visualization and historical analysis of temperature trends.

Results

The Smart Home Temperature Control System was implemented and tested under controlled conditions. The following outcomes were observed:

1. Temperature Regulation

- The system successfully maintained the room temperature around the setpoint of **30°C**.
- When the temperature exceeded **32°C**, the **fan** was activated automatically to reduce the temperature.
- When the temperature fell below **28°C**, the **heater** was turned on to raise the temperature.
- Within the **28°C–32°C range**, both fan and heater remained off, demonstrating correct hysteresis-based control.

2. Real-Time Monitoring

- The **16×2 I2C LCD** continuously displayed the current temperature and system status (Idle, Heating, or Cooling).
- The system responded to temperature changes within a few seconds, providing a stable and user-friendly local interface.

3. IoT Data Logging

- Temperature readings, actuator status (fan, heater), alarm status, and operating mode were successfully uploaded to **ThingSpeak Cloud**.
- Remote monitoring via ThingSpeak confirmed accurate and timely updates every 15 seconds.
- Historical data allowed analysis of temperature trends and actuator operations over time.

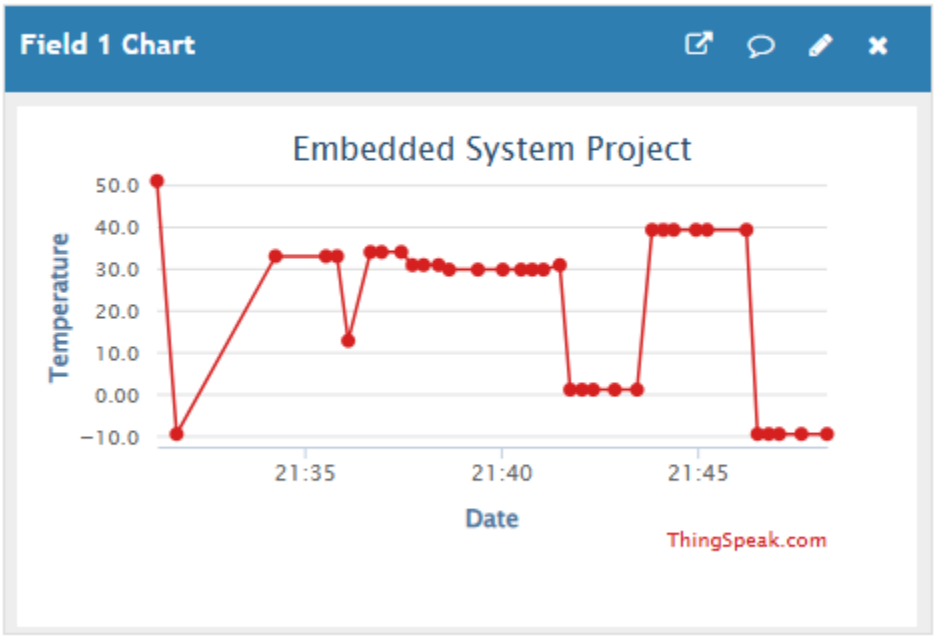
4. Alert System

- The **buzzer alarm** was triggered when the temperature exceeded predefined safe limits, providing an audible warning of extreme conditions.

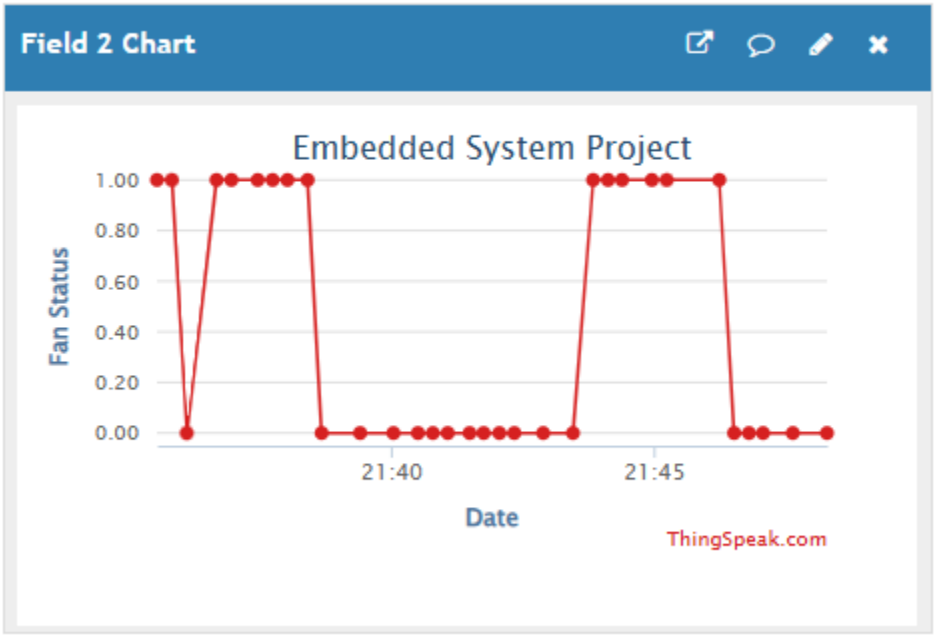
5. System Stability and Reliability

- The system operated continuously for extended periods without failures.
- Temperature control remained consistent, and actuator response was reliable.
- Wi-Fi connectivity and cloud data uploads were stable and error-free.

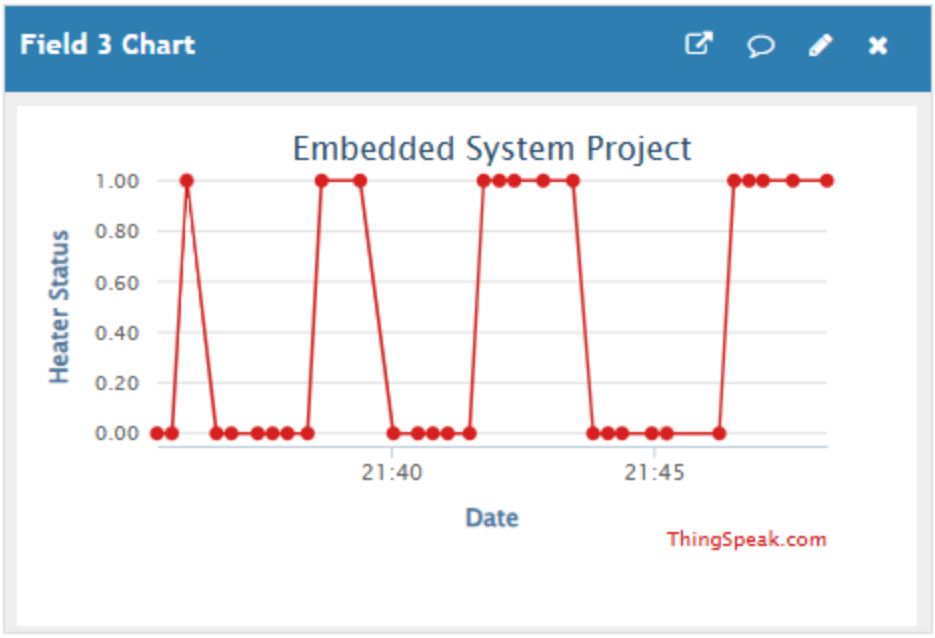
Temperature



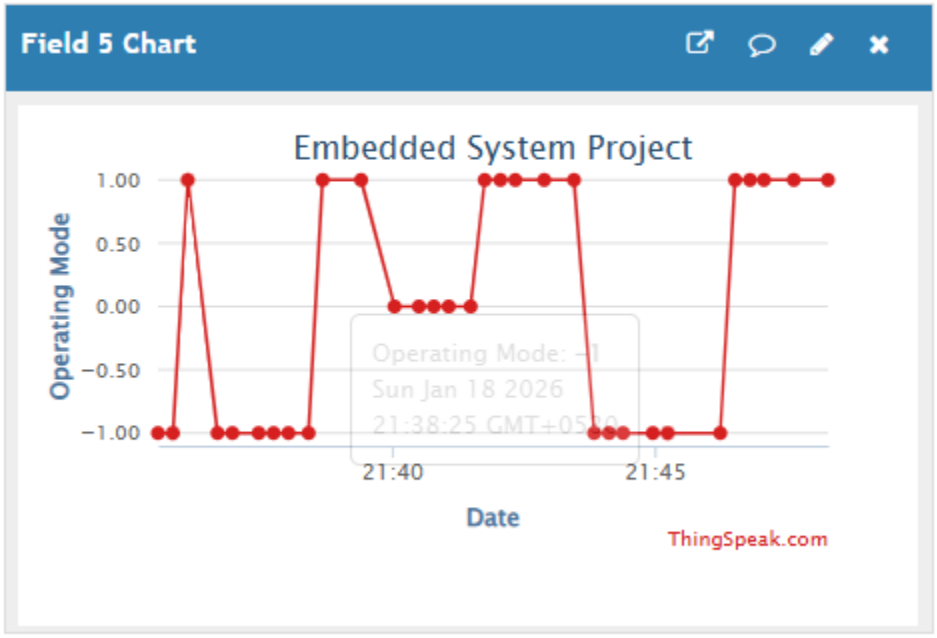
Fan Status



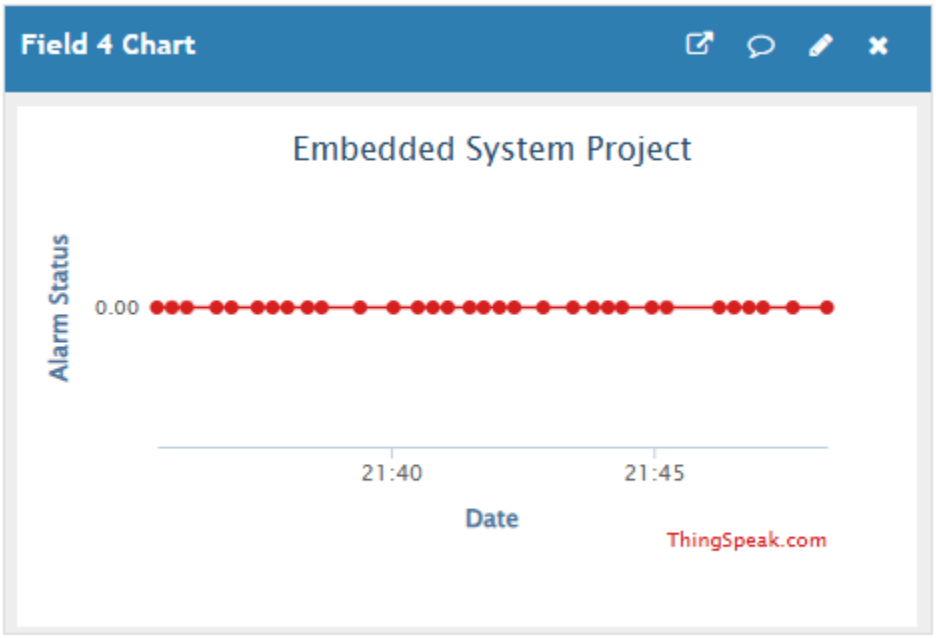
Heater Status



Operation Status



Fault/Alarm Status



Conclusion

The Smart Home Temperature Control System was successfully designed and implemented using the ESP32 microcontroller, integrating sensor interfacing, actuator control, and IoT cloud connectivity. The system effectively monitored ambient temperature using the DHT22 sensor and maintained the room temperature around the desired setpoint by automatically controlling the fan and heater. The use of hysteresis ensured stable operation and prevented frequent switching of actuators, improving both reliability and energy efficiency.

Real-time temperature data and system status were displayed locally on an I2C LCD, providing immediate user feedback. Additionally, the integration with ThingSpeak Cloud enabled remote monitoring and data logging, allowing users to analyze temperature trends and system behavior over time. The buzzer-based alert mechanism further enhanced system safety by providing audible warnings during extreme temperature conditions.

Overall, this project successfully met all the stated objectives and demonstrated the practical application of embedded systems, IoT communication, and automation principles. It highlights how microcontrollers like the ESP32 can be used to build intelligent and energy-efficient smart home solutions. Future enhancements could include mobile app integration, adaptive control using machine learning, additional environmental sensors, and improved security features for cloud communication.