# **Heater Control System: System Design Document**

### **Project Overview**

This document presents a comprehensive design for a foundational heater control system, fulfilling Part 1 of the Embedded Systems Intern assignment for upliance.ai. The objective is to design and simulate a system that activates or deactivates a heating element based on defined temperature thresholds. The proposed solution is an ESP32-based system utilizing a DS18B20 digital temperature sensor, with the logic and behavior simulated on the Wokwi platform.

The complete source code and simulation files for this project are available on GitHub: [**https://github.com/Kazim71/SMart-Heater-Control-System**](https://github.com/Kazim71/SMart-Heater-Control-System)

### **1. System Components and Architecture**

#### **1.1. Core Sensing Requirement**

A **temperature sensor** is the critical component for this system. It provides the essential feedback loop, allowing the control algorithm to make intelligent decisions about activating or deactivating the heating element.

#### **1.2. Selected Components for Simulation**

* Microcontroller: ESP32  
  This powerful module was chosen for its dual-core processor, ample GPIO, and integrated Wi-Fi/BLE capabilities. While a simpler microcontroller could meet the basic requirements, the ESP32 is a strategic choice that directly supports the future roadmap of creating a scalable, connected smart appliance.
* Temperature Sensor: DS18B20 Digital Temperature Sensor  
  The DS18B20 is superior to analog alternatives for this application due to its high accuracy and direct digital output. It comes pre-calibrated, eliminating the need for complex ADC conversion and linearization. Its digital nature provides excellent noise immunity, and its unique 64-bit serial number on each sensor allows for robust and scalable multi-sensor deployments.
* Actuating Mechanism (Heater): Red LED  
  In the simulation, a red LED visually represents the heater's state. An ON state signifies active heating.
* **Status Indicators:**
  + **Green LED:** Indicates that the target temperature has been reached and the system is stable.
  + **Buzzer:** Serves as an audible alarm for a critical **Overheat** condition.

### **2. Communication Protocol**

For communication between the ESP32 and the DS18B20 sensor, the **1-Wire Protocol** is the optimal standard.

#### **2.1. Justification**

* **Minimal I/O Usage:** As its name implies, the 1-Wire protocol requires only a single data line (plus ground), streamlining the circuit design and conserving microcontroller pins for other functions.
* **Scalability:** The protocol natively supports a multi-device bus architecture. Each device has a unique 64-bit ROM code, enabling future expansion with multiple temperature sensors without significant hardware or software redesign.
* **Robust Software Support:** Mature and well-documented libraries (e.g., OneWire and DallasTemperature in the Arduino ecosystem) are readily available, abstracting low-level protocol details and simplifying implementation.

#### **2.2. Alternative Protocols Considered**

* **I2C/SPI:** While versatile, these protocols are not natively supported by the DS18B20 and would require more I/O pins, adding unnecessary complexity.
* **UART:** This protocol is primarily used for serial communication with a host PC for debugging and logging, not direct sensor interfacing in this context.

### **3. System Block Diagram**

The following diagram illustrates the primary components and their interconnections, showing the flow of data and control within the system.

#### **Module Descriptions:**

The **ESP32 Microcontroller** serves as the brain of the system. It executes the control logic, polls the **DS18B20 Sensor** for temperature data via the 1-Wire bus, and drives all outputs. Based on the temperature reading, it controls a **Relay/SSR Driver** (simulated by a direct GPIO connection) to switch the heating element. A **Serial Interface (UART)** provides a channel for debugging and logging system status to a PC, while the **Buzzer and LED Indicators** offer immediate visual and auditory feedback on the system's operational state (e.g., heating, target reached, error).

### **4. Future Roadmap for System Evolution**

This foundational design can be evolved through a phased approach to enhance safety, functionality, and connectivity, transforming it into a market-ready smart appliance.

#### **Phase 1: Overheating Protection (Safety Enhancements)**

* **Redundant Sensing:** Implement a secondary, independent temperature sensor or a thermal fuse. If this backup sensor detects a temperature exceeding a critical safety limit, it will physically de-energize the heating element, overriding all software control.
* **Watchdog Timer:** Utilize a hardware or software watchdog timer to monitor the microcontroller's health. If the software freezes, the watchdog will trigger a system reset, preventing uncontrolled heating.
* **Sensor Fault Detection:** Develop logic to detect sensor failures (e.g., disconnection, out-of-range readings). Upon detecting a fault, the system will enter a safe, inactive state and signal an error.

#### **Phase 2: Multiple Heating Profiles & Precision Control (Functional Expansion)**

* **User Interface (UI):** Add an OLED display and pushbuttons or a rotary encoder to allow users to select pre-defined heating profiles (e.g., "Boil," "Simmer," "Keep Warm").
* **Profile Persistence:** Store user-defined temperature setpoints and durations in the ESP32's non-volatile memory (Flash or EEPROM) to retain settings across power cycles.
* **PID Control:** Upgrade from a simple on/off algorithm to a **Proportional-Integral-Derivative (PID) controller**. PID control will provide far more precise temperature regulation, minimizing overshoot and maintaining a stable temperature, which also improves energy efficiency by avoiding constant full-power on/off cycles.

#### **Phase 3: Connectivity and Advanced Features (IoT Integration)**

* **Wi-Fi & Cloud Connectivity:** Leverage the ESP32's built-in Wi-Fi to connect the device to a cloud platform (e.g., Firebase, AWS IoT). This enables remote monitoring and control via a companion smartphone application.
* **Data Logging & Analytics:** Store historical temperature and energy usage data on an SD card or in the cloud. This data can be used for performance analysis, optimization, and providing insights to the user.
* **Energy Consumption Monitoring:** Integrate a current sensor (e.g., ACS712) to monitor the heater's real-time power consumption, enabling energy-saving features and user feedback.
* **Firmware Over-The-Air (FOTA) Updates:** Implement a FOTA mechanism to deploy new features, performance improvements, and security patches remotely, without requiring physical access to the device.