# **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 temperature thresholds. The proposed solution is an ESP32-based system utilizing a DS18B20 digital temperature sensor, with the logic and behavior to be simulated on the Wokwi platform as required.

### **1. System Components and Identification**

#### **1.1. Minimum Sensors Required**

A **Temperature Sensor** is the indispensable component for the effective operation of a basic heater control system. Its primary function is to continuously acquire temperature data, which serves as critical feedback for the control algorithm to determine the activation or deactivation state of the heating element.

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

* Microcontroller: ESP32  
  This powerful module is chosen for its dual-core processor, ample GPIO, and integrated Wi-Fi/BLE capabilities. While a simpler microcontroller could fulfill the basic requirements, the ESP32 is a forward-looking choice that aligns perfectly with the future roadmap for creating a scalable, smart appliance prototype.
* Temperature Sensor: DS18B20 Digital Temperature Sensor  
  This sensor is highly recommended over analog alternatives for its inherent accuracy and direct digital output. It arrives pre-calibrated, obviating the complexities of analog-to-digital conversion and linearization. Its digital nature provides excellent noise immunity, and its unique 64-bit serial number allows for robust and scalable multi-sensor deployments.
* Actuating Mechanism (Simulated Heater): Red LED  
  This will visually represent the state of the heater. When the LED is ON, the system is actively heating.
* **Feedback & Status Indicators:**
  + A **Green LED** will indicate when the system has successfully reached its target temperature.
  + A **Buzzer** will serve as an audible alarm to signal a critical Overheat condition.

### **2. Recommended Communication Protocol and Justification**

For data exchange between the microcontroller and the selected DS18B20 temperature sensor, the **1-Wire Protocol** is identified as the optimal communication standard.

#### **2.1. Justification**

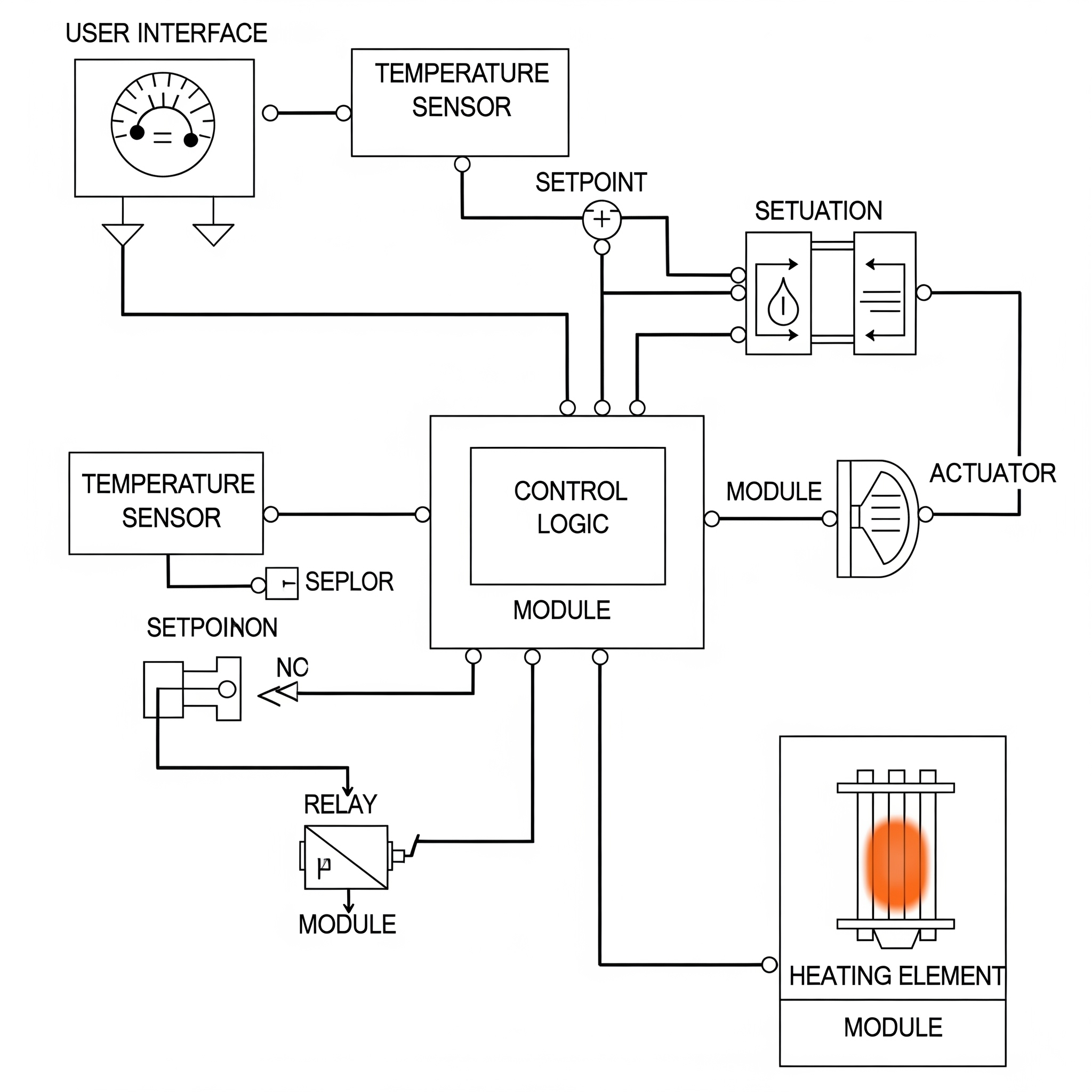
* **Minimal Interconnection Requirements:** The 1-Wire protocol, as its name implies, requires only a single data line (plus ground) for comprehensive communication. This streamlines circuit design and minimizes the use of microcontroller I/O pins.
* **Multi-Device Scalability:** The 1-Wire bus architecture supports the concurrent connection of multiple devices. Each device is uniquely identifiable via a 64-bit ROM code, providing a clear pathway for future system expansions requiring multi-point temperature sensing.
* **Extensive Software Support:** Well-documented libraries (e.g., OneWire and DallasTemperature in the Arduino ecosystem) are readily available. These libraries abstract the low-level protocol intricacies, simplifying implementation.

#### **2.2. Alternative Protocol Considerations**

* **I2C/SPI:** While versatile, these protocols are not natively supported by the DS18B20 and would require more I/O pins, which is unnecessary for this application.
* **UART:** This protocol is customarily employed for serial communication with a host PC for debugging and logging, making it less suitable for direct sensor interfacing.

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

The following block diagram delineates the principal components of the heater control system and their interconnections.



#### **Module Descriptions:**

* **Microcontroller (ESP32):** The central processing unit, responsible for acquiring sensor data via the 1-Wire bus, executing the control algorithm, and managing all outputs.
* **DS18B20 Sensor:** Measures the ambient temperature and transmits the data digitally.
* **Relay/SSR Driver:** An interface circuit (simulated by a direct GPIO connection to the LED) that enables the low-power microcontroller to safely switch the high-power heating element in a real-world application.
* **Serial Interface (UART):** A channel used for debugging, logging temperature and system status to a PC.
* **Buzzer/LED Indicators:** Provide immediate auditory and visual feedback on the system's operational status.

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

The foundational system can be enhanced through a phased approach to augment safety, expand functionality, and improve user experience.

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

* **Redundant Temperature Sensing:** Implement a secondary, independent temperature sensor or a dedicated thermal fuse. Should this auxiliary sensor register a temperature exceeding a predefined safety threshold, it will immediately de-energize the heating element, overriding all standard control logic.
* **Software Watchdog Timer:** Integrate a watchdog timer to monitor the microcontroller's operational integrity. In the event of a software deadlock, the watchdog will initiate a system reset, mitigating the risk of uncontrolled heating.
* **Fault Detection Logic:** Develop logic to identify sensor malfunctions (e.g., disconnections, out-of-range readings). Upon fault detection, the system will enter a safe state, deactivating the heater and activating an error indicator.

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

* **User Interface (UI) Integration:** Incorporate a small OLED display and pushbuttons or a rotary encoder to enable users to select from pre-defined heating profiles (e.g., "Boil," "Simmer," "Keep Warm").
* **Profile Persistence:** Store temperature setpoints and durations for each profile in the microcontroller's non-volatile memory (e.g., Flash or EEPROM).
* **PID Control Implementation:** Transition from simple on/off control to a **Proportional-Integral-Derivative (PID)** control algorithm. PID control offers superior precision in temperature regulation, minimizing overshoot and oscillations for more stable and energy-efficient heating.

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

* **Wi-Fi/Cloud Connectivity:** Enable the ESP32's Wi-Fi to facilitate remote monitoring and control via a cloud platform (e.g., Firebase, AWS IoT). This would support a companion smartphone app.
* **Data Logging and Analytics:** Implement mechanisms to store historical temperature data and energy consumption metrics to an SD card or cloud database for analysis and optimization.
* **Energy Consumption Monitoring:** Integrate a current sensor (e.g., ACS712) to monitor the heater's power usage, providing valuable data for energy-saving features and user feedback.
* **Firmware Over-The-Air (FOTA) Updates:** Provision for remote firmware updates, enabling the deployment of new features and bug fixes without requiring physical access to the device.