Architecture of the Temperature Monitoring System

The **Temperature Monitoring System** is designed to sample temperature values using an ADC-connected sensor, process the readings, update LED indicators based on defined thresholds, and provide system information via serial output. The software supports two hardware revisions with different temperature sensor resolutions.

Note: C++ project is in the same structure as C project.

1. System Overview

- **Microcontroller/PC Simulation**: The system runs on a microcontroller but is implemented in a PC environment with mocked hardware interfaces.
- **Temperature Sampling**: A sensor connected via ADC provides temperature readings at a precise interval.
- **LED Indicators**: Three LEDs (Green, Yellow, Red) visualize different temperature conditions.
- **EEPROM Configuration**: Stores the hardware revision type and a serial number.
- **PC-Based Execution**: Uses a loop with a delay (usleep or Sleep) to mimic real-time sensor readings.

2. System Components & Responsibilities

2.1 Hardware Interfaces (Mocked in PC Simulation)

1. **ADC (Analog-to-Digital Converter)**
    - Reads raw sensor values at regular intervals.
    - Converts the ADC value to temperature based on hardware revision.
    - Supports two sensor types:
        - **Rev-A**: 1°C per ADC digit.
        - **Rev-B**: 0.1°C per ADC digit.
2. **GPIOs (LED Indicators)**
    - **Green (G)**: Normal operation temperature (< 85°C).
    - **Yellow (Y)**: Warning level temperature (≥ 85°C).
    - **Red (R)**: Critical temperature (≥ 105°C or < 5°C).
3. **EEPROM (Configuration Memory)**
    - Stores hardware revision (Rev-A or Rev-B).
    - Stores a placeholder serial number.

2.2 Software Modules & Responsibilities

(1) main.c (Application Layer)

- Initializes the system.
- Reads ADC values.
- Converts ADC readings to temperature.

- Decides which sensor to display based on temperature format.
- Updates LED indicators based on temperature conditions.
- Implements a **loop** to simulate real-time temperature monitoring.
- Stops execution if temperature reaches MAX_TEMP.

(2) adc_handler.c (ADC Abstraction Layer)

- Mocks ADC readings.
- Manages temperature increments in **steps**:
    - **Rev-A**: Whole numbers only.
    - **Rev-B**: 0.2°C increments.
- Converts raw ADC values to temperature.

(3) led_controller.c (LED Control)

- Controls LED states based on temperature.
- Implements **threshold-based logic**:
    - **Green** = Safe (< 85°C).
    - **Yellow** = Warning (85°C ≤ T < 105°C).
    - **Red** = Critical (≥ 105°C or < 5°C).

(4) eeprom_handler.c (EEPROM Simulation)

- Mocks EEPROM storage.
- Provides functions to retrieve:
    - Hardware revision.
    - Serial number.

(5) system_config.h (Global Configuration)

- Defines constants:
    - **Temperature limits** (MIN_TEMP, MAX_TEMP).
    - **Step size** for temperature increments (STEP_B).
    - **Scaling factors** for ADC conversion.

3. Control Flow

1. **System Initialization**
    - Retrieve EEPROM values (revision type, serial number).
    - Initialize LED states.
2. **Main Monitoring Loop**
    - Read ADC values.
    - Convert ADC to temperature.
    - Determine the sensor type to display.

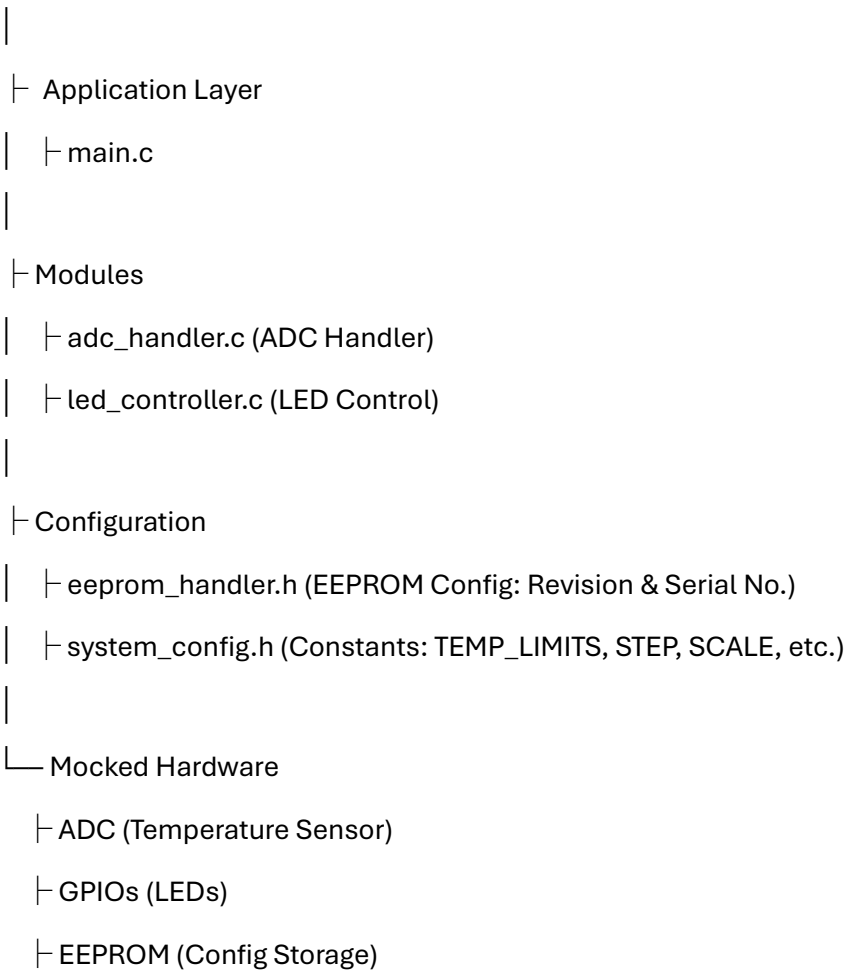
- Update LED states.
- Print temperature & LED status.
- Sleep for 100ms.
- **Exit if MAX_TEMP is reached.**

3. **System Shutdown**
   - Print summary message with iteration count.
   - Exit program.

4. Diagram

The architecture can be visualized as:

Temperature Monitoring System



5. Summary

- **ADC reads temperature** at a fixed interval.
- **EEPROM provides configuration** (hardware revision, serial number).
- **LEDs indicate safe, warning, or critical temperatures**.
- **The system stops when temperature reaches MAX_TEMP**.