Technical and Operational Documentation for Task Scheduler

The task scheduler in the embedded thermostat system is designed to manage various periodic tasks, such as reading the temperature, controlling an LED, processing button inputs, and sending data over UART. This scheduler ensures that all these tasks are executed at specified intervals without blocking the main execution flow, thereby keeping the system responsive to user inputs and environmental changes.

Functional Overview

The task scheduler is implemented within the scheduler function, which operates in the main loop of the program. The scheduler uses a custom timekeeping variable, elapsedTime, to track the passage of time and trigger tasks at predefined intervals. The primary inputs to this scheduler include the timer interrupt, which periodically increments the elapsedTime variable, and button inputs, which set flags when the user presses the buttons. The temperature is also periodically read from the sensor.

The outputs of the scheduler include controlling the LED state based on the temperature relative to the set-point and sending formatted data over UART, which includes the temperature, set-point, and system time. The expected results are that the system should adjust the set-point temperature when a button is pressed, reflect the heating status through the LED, and output the current temperature and time via UART every second.

Scheduler Implementation

The scheduler operates by checking specific conditions based on elapsed time. It handles button inputs every 200 milliseconds, reads the temperature every 500 milliseconds, and sends data over UART every second. If the scheduler detects that a button has been pressed, it updates the set-point temperature and logs the action. Similarly, the scheduler reads the current temperature from the sensor and updates the LED state to indicate whether the heating should be on or off.

Error handling is integrated into the scheduler and peripheral initialization functions. If any peripheral fails to initialize or if there is an issue with reading from the temperature sensor, an error message is sent over UART, and the system halts to prevent further issues.

State-Machine Diagram

The task scheduler can be represented as a state-machine diagram, where each state corresponds to a specific task, such as waiting for the next interval, processing button inputs, reading the temperature, controlling the LED, or sending data over UART. Transitions between these states occur based on elapsed time and input conditions, ensuring that all tasks are executed in a timely and non-blocking manner.

Thermostat Peripherals Support and Architecture Analysis

The thermostat system supports various peripherals, including the temperature sensor (connected via I2C), an LED (controlled via GPIO), buttons (also connected via GPIO), and UART for communication with external systems. The task scheduler efficiently manages these peripherals, ensuring that the system can monitor and control the temperature while providing feedback to the user.

Hardware Architectures Comparison

When considering the hardware architecture for the thermostat, three options were analyzed: the TI CC3220, the Microchip SAMW25, and the Freescale/NXP Kinetis KW41Z. The TI CC3220 is particularly well-suited for this application, offering 256KB of RAM, up to 1MB of Flash, and integrated Wi-Fi connectivity. It also provides extensive support for the necessary peripherals, including GPIO, I2C, and UART, and is designed with IoT applications in mind, making it ideal for cloud-connected thermostat systems.

In contrast, the Microchip SAMW25, while also providing integrated Wi-Fi and a range of peripheral support, offers less RAM and Flash memory, which may limit future scalability. The Freescale/NXP Kinetis KW41Z, although rich in peripherals, lacks native Wi-Fi support, requiring additional modules for cloud connectivity, which increases complexity and cost.

Cloud Connectivity via Wi-Fi

The thermostat connects to the cloud over Wi-Fi, utilizing the integrated Wi-Fi module in the selected microcontroller (e.g., TI CC3220). This module facilitates secure communication with cloud services, allowing the thermostat to send temperature data and receive control commands remotely. For successful cloud connectivity, the architecture must support secure communication protocols, such as TLS, to protect data in transit. Efficient data handling is also crucial to ensure minimal latency and reliable transmission, which the TI CC3220 is well-equipped to handle.

Flash and RAM Considerations

Flash memory is used to store the firmware, which includes the task scheduler, peripheral drivers, and communication protocols, while RAM is required for runtime operations such as stack space, buffer management, and handling real-time tasks. The TI CC3220 provides sufficient Flash and RAM to support the thermostat’s functionality, ensuring that all tasks can be implemented without memory constraints.

Conclusion

The task scheduler is a critical component of the thermostat system, ensuring that all tasks are executed efficiently and without blocking the main program. The TI CC3220 is recommended as the most suitable hardware architecture due to its integrated Wi-Fi, extensive peripheral support, and adequate Flash and RAM. This architecture enables the thermostat to be cloud-connected, providing real-time monitoring and control capabilities. The provided documentation and analysis ensure that the system is well-understood and ready for further development and deployment.