Introduction

The goal of this assignment is to develop embedded firmware for a device that integrates a button, RGB LED, vibrator, and monitoring system. This project aims to assess skills in embedded programming, hardware integration, and real-time data monitoring. The firmware will enable the RGB LED to toggle between different states based on user input from the button, control the vibrator's operation based on a long press, and implement a monitoring system that logs the status of the device at regular intervals. Additionally, the assignment emphasizes the importance of error handling and power management to enhance the device's reliability and efficiency.

Design Decisions

In developing the embedded firmware for the button, RGB LED, and vibrator, several key design decisions were made to enhance functionality, maintainability, and performance. The following points summarize these decisions:

1. **Interrupt-Driven Architecture**:
   I implemented an interrupt-driven design using ISRs for button presses, periodic reporting, and vibrator timing to enhance responsiveness and reduce CPU load during idle times, ultimately improving real-time performance.
2. **Button Debouncing**:
   A debouncing mechanism was integrated into the button ISR to ensure reliable detection of button presses, thereby reducing erratic behavior and preventing unintended state changes due to mechanical noise.
3. **Periodic Monitoring with RTC**:
   I utilized a Real-Time Clock (RTC) to log system status every 2 seconds, providing valuable feedback for performance evaluation and facilitating easier debugging and diagnostics.
4. **Watchdog Timer Implementation**:
   The incorporation of a watchdog timer ensures that the system can recover from critical failures, enhancing robustness and minimizing downtime in case of unexpected errors.
5. **Low Power Mode**:
   By entering a low-power mode when idle, particularly when the RGB LED is off and the vibrator is inactive, I aimed to significantly extend battery life and reduce overall power consumption.
6. **Modular Code Structure**:
   I organized the code into multiple files to separate concerns, which improves maintainability, enhances readability, and facilitates the addition of new features without disrupting existing functionality.

Functionality Implementation

The firmware's main loop implements a state-driven system that manages LED patterns, button interactions, and power states. During normal operation, the system continuously monitors the time between events to control LED fading effects and watchdog updates. The RGB LED cycles through three states via button presses: off, static red, and an RGB fading pattern. In the fading mode, the

main loop updates color transitions every 10ms, creating smooth transitions between red, green, and blue using PWM control. The watchdog timer is refreshed every 10 seconds to ensure system stability.

While the main loop provides the core timing and state management, most functionality is handled through interrupts to improve efficiency. Button presses are detected through a GPIO interrupt, which determines between short presses (for LED state changes) and long presses (≥3 seconds for vibrator activation). The system monitors its idle state and automatically enters a low-power mode when the LED is off and the vibrator is inactive, with interrupts ensuring the system can wake when needed. Every 2 seconds, an RTC interrupt triggers system status logging, recording the current state of all components. This interrupt-driven approach minimizes CPU usage while maintaining responsive and reliable operation.

Decision Framework: Interrupt-Driven Solution vs. RTOS

In this project, I opted for an interrupt-driven architecture rather than implementing a RTOS for several reasons. First, the application’s complexity was relatively low, making a simple interrupt-driven design sufficient to meet the project requirements without the added overhead of an RTOS. This approach allows for efficient resource utilization, particularly in terms of memory and CPU cycles. Additionally, the interrupt-driven model simplifies the design by focusing on event-driven responses, enabling quick and efficient handling of user inputs and system monitoring tasks. Overall, this decision led to a straightforward and effective implementation that meets the project's goals while maintaining simplicity and clarity.

How the Project Could Have Been Implemented with an RTOS

If an RTOS were used for this project, the implementation could have been structured as follows:

1. **Task Creation**:
    - **Button Task**: A dedicated task to handle button inputs and debouncing, ensuring reliable detection of button presses and managing state transitions.
    - **LED Task**: A task responsible for managing the RGB LED states, including smooth fading transitions.
    - **Vibrator Task**: A task that controls the vibrator operation, activating it on a long button press and managing the timing for its deactivation.
    - **Monitoring Task**: A periodic task that logs the system status every 2 seconds, facilitating easy monitoring and diagnostics.
2. **Inter-task Communication**:
   Utilize message queues or semaphores to facilitate communication between tasks, allowing them to share data and notify each other about state changes without conflicts.
3. **Timer Services**:
   Leverage built-in RTOS timer services for managing time-based events, such as the timing for the vibrator and periodic monitoring, instead of manually implementing timers.
4. **Low Power Management**:
   Use the RTOS's built-in low-power management features to allow the microcontroller to enter sleep modes while maintaining responsiveness to interrupts, thus optimizing power consumption.

5. **Error Handling**:
   Implement robust error handling and recovery mechanisms using the RTOS features, such as task watchdogs, to ensure the system remains resilient in case of unexpected failures.

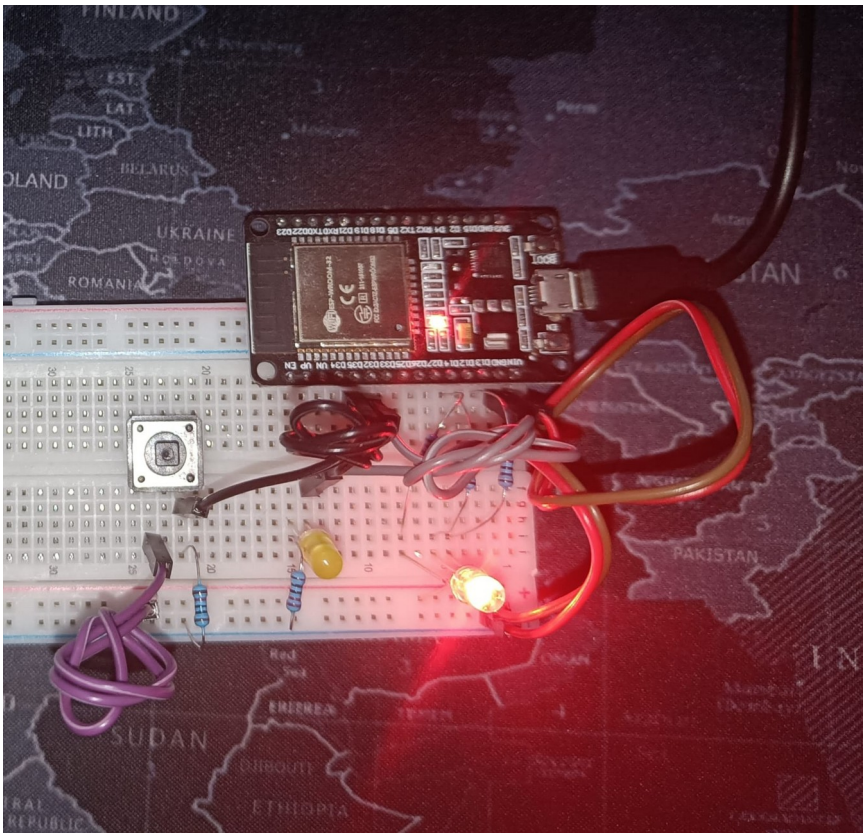
Testing

To validate the design and functionality of the system, I created a test implementation using an ESP32 DevKit board and the Arduino IDE. This approach allowed for rapid prototyping and verification of the core functionality while maintaining the same basic architecture as the original design. The main adaptations included using Arduino's built-in PWM functions (ledcWrite) instead of direct PWM register manipulation, utilizing the ESP32's interrupt handling mechanisms, and implementing serial monitoring for system status logging.

The code was modified to accommodate the ESP32's specific requirements, including:

- Using Arduino's GPIO interrupt attachments instead of direct interrupt handling
- Implementing PWM through ESP32's LED Control (LEDC) peripheral
- Utilizing built-in Arduino timing functions (millis()) for timing management
- Adding Serial communication for real-time monitoring and debugging

The hardware setup consisted of a pushbutton, an RGB LED, and a yellow LED (used as a substitute for the vibrator module):



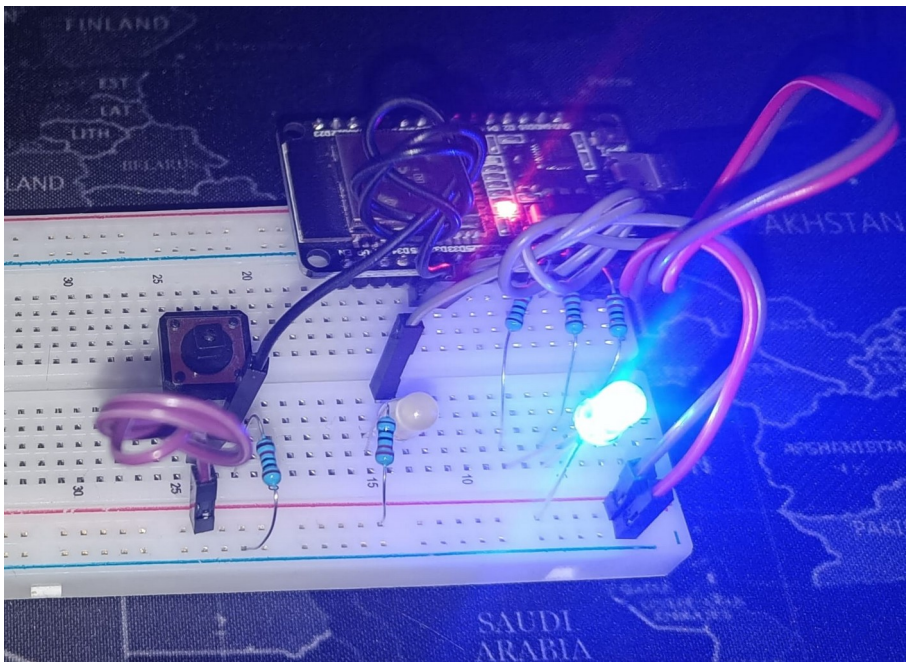
Test Cases Performed:

1. Functionality Testing
   - Verify LED state transitions through short button presses
   - Confirm smooth color transitions in RGB fading mode

- Test long press detection and yellow LED activation timing (simulating vibrator)
- Validate debouncing effectiveness with rapid button presses

2. Reliability Testing
    - Test system operation over extended periods
    - Verify consistent timing of status updates
    - Check button response reliability under different pressing patterns
    - Monitor serial output for any anomalies in system state reporting
3. Timing Verification
    - Measure yellow LED activation duration accuracy (2-second timeout)
    - Verify RGB fade transition smoothness
    - Confirm 2-second interval of status updates
    - Test button press duration detection accuracy (3-second threshold)

```
03:52:21.942 -> ==================
03:52:23.964 -> === System Status ===
03:52:23.964 -> Button: Released
03:52:23.964 -> LED State: Static Red
03:52:23.964 -> Vibrator: Off
03:52:23.964 -> ==================
03:52:25.953 -> === System Status ===
03:52:25.953 -> Button: Pressed
03:52:25.953 -> LED State: Static Red
03:52:25.953 -> Vibrator: Off
03:52:25.953 -> ==================
03:52:27.942 -> === System Status ===
03:52:27.942 -> Button: Pressed
03:52:27.942 -> LED State: Static Red
03:52:27.942 -> Vibrator: On
03:52:27.942 -> ==================
03:52:29.963 -> === System Status ===
03:52:29.963 -> Button: Released
03:52:29.963 -> LED State: Static Red
03:52:29.963 -> Vibrator: Off
03:52:29.963 -> ==================
03:52:31.952 -> === System Status ===
03:52:31.952 -> Button: Released
03:52:31.952 -> LED State: Fading RGB
03:52:31.952 -> Vibrator: Off
03:52:31.952 -> ==================
```


FINLAND
EST
LAT
LTH
SAUDI
ARABIA