Université Saint-Joseph de Beyrouth
Faculté d'ingénierie
Institut national des télécommunications et de l'informatique

Firmware Design: Elevator Controller

Report Submitted by

Jane Abi Saad
Patrick Soueid

Bachelor of Computer Science
Saint Joseph University of Beirut

Submitted to

Dr. Majdi Richa
Firmware Design

Submission Date: January 10, 2025

Contents

I Abstract . . . 2
II Introduction . . . 2
III System Overview . . . 3
IV Firmware Architecture . . . 3
V Flowcharts . . . 5
VI System Operation . . . 6
VI.1 Initialization . . . 6
VI.2 Task Execution . . . 6
VII Challenges and Resolutions . . . 7
VIII Future Enhancements . . . 7
IX Conclusion . . . 7

I Abstract

This document explores the design and implementation of a firmware-based elevator control system tailored for a four-floor building. By employing a structured task-oriented programming approach, this project demonstrates efficient task scheduling, real-time data acquisition, and seamless interaction with hardware components such as sensors, displays, and motor drivers. The modular architecture, complemented by the integration of advanced sensors, ensures precise operation and scalability for future enhancements.

II Introduction

The elevator controller project is an embedded firmware application that showcases the integration of real-time operational requirements with modular and task-based programming techniques. This system is designed to meet the demands of efficient and safe elevator operations, leveraging robust hardware-software interactions and advanced firmware architecture.

The core functionality revolves around:

- Managing floor requests in an optimized manner to ensure minimal waiting times and efficient navigation.
- Monitoring and displaying temperature readings using an LM35 sensor and ADC modules to enhance passenger comfort.
- Utilizing a DS1307 RTC module for accurate timekeeping, displayed via a 7-segment interface.
- Implementing a real-time scheduler to manage non-blocking task execution, enabling seamless integration of multiple concurrent operations.

This firmware operates on a hardware abstraction layer, enabling compatibility and scalability across different microcontroller platforms. By separating hardware-specific details from the application logic, the design ensures modularity, maintainability, and future expansion potential. Additionally, non-blocking programming principles are employed to avoid delays and ensure continuous system responsiveness.

The firmware incorporates a cooperative real-time scheduler that governs tasks such as:

- Reading and processing floor request inputs via push buttons.
- Driving the motor to navigate between floors based on prioritized requests.

- Displaying real-time status updates on 7-segment displays, including floor numbers and temperature.
- Communicating with external peripherals such as the DS1307 RTC and LM35 temperature sensor over I2C and ADC interfaces, respectively.

This project exemplifies the integration of fundamental firmware principles, such as abstraction, modularity, and concurrency, within an embedded system. The document explores the architectural breakdown, implementation strategies, testing procedures, encountered challenges, and proposed expansions for the system. Ultimately, the report aims to provide a comprehensive understanding of how an efficient and scalable elevator control system is developed using firmware design methodologies.

III System Overview

The elevator controller is developed as an embedded system that combines hardware and firmware to ensure efficient operation. Key components include:

- **Microcontroller**: The PIC18F67K22, responsible for orchestrating all operations.
- **Inputs**: Floor call buttons, direction indicators, and cabin floor selectors.
- **Outputs**: Seven-segment displays, LED indicators, and motor signals.
- **Sensors**: LM35 temperature sensor and I2C-based RTC for environmental data.
- **Motor Driver**: Enables precise movement and speed control of the elevator cabin.

IV Firmware Architecture

The firmware design adopts a modular approach, ensuring maintainability and scalability. Each module operates independently while collaborating to achieve the overall system goals. The architecture includes:

- **Scheduler**: Ensures task prioritization and avoids blocking operations.
- **Input Handler**: Processes button presses and queues requests.
- **Motor Controller**: Manages cabin movement and stopping logic.

- **Display Updater**: Cycles through floor indicators and environmental data.
- **Error Detection**: Monitors system integrity and triggers safety mechanisms.

V Flowcharts

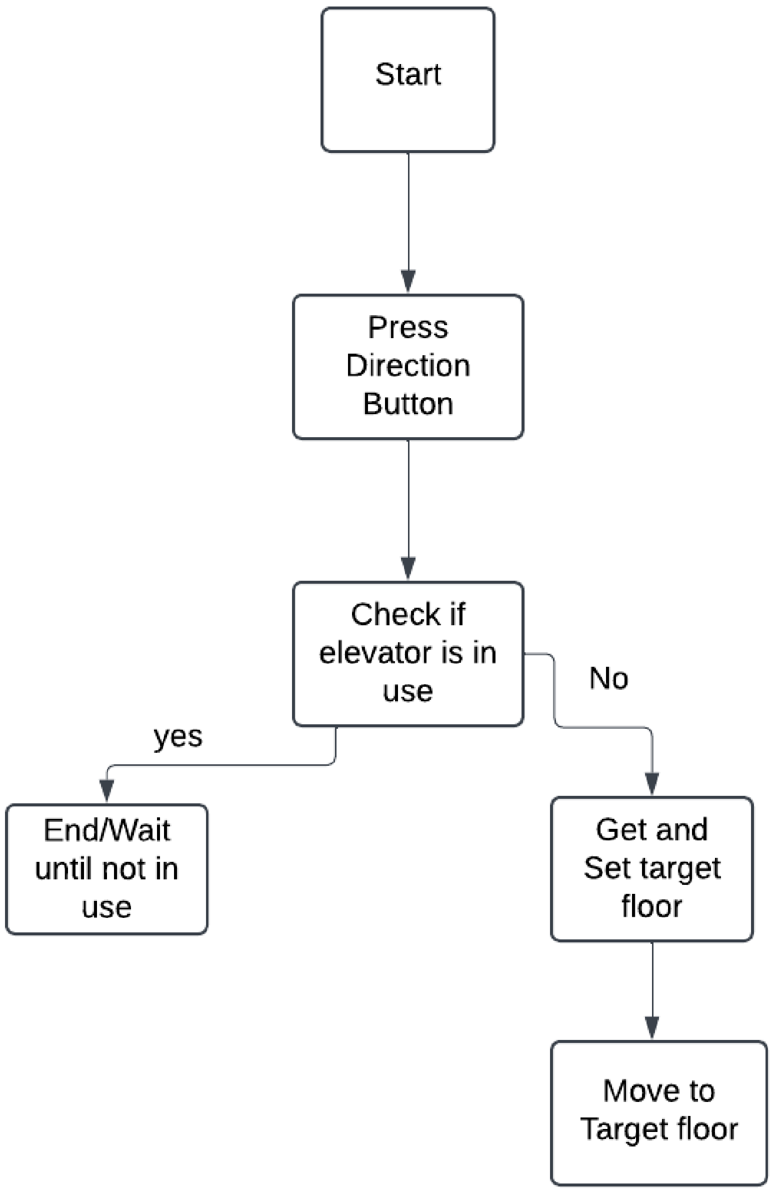

Figure 1: Elevator Motor Flowchart

The flowchart illustrates the process of operating the elevator system based on user input and the system's current state. The process begins when a user presses a directional button, either requesting the elevator to move up or down. The system then checks if the elevator is already in use. If the elevator is in use, the process ends or waits until the elevator becomes available. If the elevator is not in use, the system retrieves and sets the target floor based on the user's request. Once the target floor is set, the elevator moves toward the target floor, completing the operation. This streamlined process ensures efficient handling of user requests while maintaining logical control over elevator operations.

VI System Operation

VI.1 Initialization

Upon startup, the system:

- Initializes the scheduler with `scheduler_init(get_ticks_counter)` for task management.
- Configures hardware I/O pins (GPIO) for inputs and outputs.
- Initializes the system, including global configurations and interrupts.
- Sets up the ADC peripheral for reading temperature from the LM35 sensor.
- Initializes the cabin interior display for the elevator.
- Initializes elevator tasks for handling requests and managing elevator movement.

VI.2 Task Execution

- **Button Monitoring**: Detects floor requests from buttons inside the cabin and outside on each floor, and immediately processes them to control the elevator's movement.
- **Motor Control**: Controls the elevator's movement between floors, adjusting the direction (up or down) based on the current floor and the requested floor.
- **Display Updates**: Includes a shared display inside the cabin that alternates between showing the temperature, time, and date, alongside a display that shows the current floor number.

VII Challenges and Resolutions

- **Queue Floor Direction Requests**: One of the challenges faced during development was implementing a queue-based algorithm to manage floor requests efficiently, as it required careful handling of request priorities, elevator direction, and ensuring no conflicts occurred when processing multiple requests simultaneously.
- **Resolution:** Implemented the elevator to serve one request at a time as a temporarily solution.
- **CPU Overload**: A significant issue encountered was CPU overload caused by scheduling too many tasks, which led to inefficient processing and delays in responding to elevator requests
- **Resolution:** Removed some of the tasks and directly called their function.

VIII Future Enhancements

- Implementing the queue functionality for more efficient elevator requests handling.
- Integrating a weight sensor to prevent overloading
- Enhancing accessibility through voice-based announcements.

IX Conclusion

This project demonstrates the implementation of a robust elevator control system using advanced firmware principles. By prioritizing modularity, real-time data processing, and safety, the system achieves efficient operation and offers scope for future innovations. It serves as a practical application of embedded system design, combining technical rigor with user-focused functionality.