Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Кіберфізична система «Розумний ліфт». Програмна частина
Назва теми

КвРКІ 022016.22.02.35 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент ІІІ курсу, група КІ2-22-2 _____ Артем КОЛОМІЄЦЬ
Підпис, Ініціали, прізвище

Керівник _____ Юрій ВОЙЧУР
Підпис, дата, Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата, Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис, Ініціали, прізвище

«12» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Коломієць Артем Анатолійович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Кіберфізична система «Розумний ліфт». Програмна частина
Керівник проекту (роботи) Войчур Юрій Олексійович д.ф., старший викладач.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 5

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Проєктування системи керування та обробки сигналів у кіберфізичній системі «Розумний ліфт» (програмна частина)
Вибір і підбір програмних компонентів для реалізації кіберфізичної системи «Розумний ліфт»
Розробка програмної частини проекту «Розумний ліфт» та тестування прототипу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Архітектура ПЗ проєкту
Архітектура ПЗ для кіберфізичної системи
Апаратне забезпечення проєкту

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «10» 01 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	виконано
4	Робота над розділом 2 – Проєктування програмної частини кіберфізичної системи «Розумний ліфт»	01.04.2025	виконано
5	Робота над розділом 3 – програмна реалізація кіберфізичної системи «Розумний ліфт»	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Артем.КОЛОМІЄЦЬ
Підпис Ініціали, прізвище

Керівник роботи _____ Юрій.ВОЙЧУР
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 022016.22.02.35 Е8	Пояснювальна записка	55		
			Графічні матеріали			
2		КвРКІ 022016.22.02.35 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 022016.22.02.35 Е8	Архітектура ПЗ для кіберфізичної системи	1		
4		КвРКІ 022016.22.02.35 Е8	Апаратне забезпечення проєкту	1		

КвРКІ 022016.22.02.35 ПЗ

Зм	Арк	№ докум	Підпис	Дата
Розробив		Коломієць		
Перевір.		Войчур		
Н. контр.		Кисіль		
Затв.		Павлова		

Відомість проекту

Літера	Аркуш	Аркушів
У	1	1

ХНУ, КІ2-22-2

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Кіберфізична система «Розумний ліфт». Програмна частина».

Автор роботи: Коломієць Артем Анатолійович.

Керівник роботи: Войчур Юрій Олексійович.

Пояснювальна записка: 55 с., 17 рис., 1 табл., 3 дод., 40 джерел.

Графічна частина: 3 креслення.

РОЗУМНИЙ ЛІФТ, КІБЕРФІЗИЧНА СИСТЕМА, ПРОГРАМНА АРХІТЕКТУРА, IoT, БАЗА ДАНИХ.

Метою дипломної роботи є розробка програмної частини кіберфізичної системи «Розумний ліфт» для підвищення ефективності управління ліфтовими перевезеннями та покращення взаємодії користувача з системою.

Об'єктом дослідження є процес функціонування інтелектуальної системи керування ліфтом у багатоповерхових будівлях.

Предметом дослідження є алгоритми, архітектурні підходи та технології реалізації програмної складової розумного ліфта з використанням принципів (IoT) та баз даних.

Під час дослідження застосовано метод систематичного огляду літератури та аналізу сучасних програмних рішень у сфері кіберфізичних систем управління для формування вимог до програмної архітектури системи та вибору оптимальних технологій її реалізації.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП

В умовах сучасного міського середовища особливу роль відіграють безпечні, ефективні та комфортні системи вертикального транспортування людей і вантажів, серед яких найбільш поширеними є ліфтові системи. Сучасні ліфти, будучи складними технічними засобами, вимагають постійного моніторингу та управління для забезпечення високого рівня надійності, швидкого реагування на аварійні ситуації та максимальної зручності для користувачів. Традиційні підходи до організації керування ліфтами, як правило, мають обмеження у можливостях гнучкої адаптації до змінних умов експлуатації, не забезпечують повноцінного дистанційного моніторингу та характеризуються низьким рівнем інтеграції із сучасними інформаційними системами.

Актуальність проведення цього дослідження обумовлена необхідністю створення ефективної кіберфізичної системи для «розумного ліфта», яка здатна здійснювати автоматизоване керування рухом кабіни, контролювати роботу дверних механізмів, забезпечувати своєчасне реагування на аварійні події та інтегруватися з сучасними хмарними платформами для збору й аналізу телеметричних даних. Реалізація програмної частини такої системи має на меті не лише забезпечити точність позиціонування і надійність експлуатації ліфта, але й значно знизити витрати на технічне обслуговування та мінімізувати ризики виникнення аварійних ситуацій.

Метою дипломної роботи є розробка архітектури та програмної реалізації кіберфізичної системи «Розумний ліфт», а також оцінка ефективності її функціонування у віртуальному середовищі моделювання.

Об’єктом дослідження виступає процес автоматизованого керування рухом ліфтової кабіни та її окремими механізмами. Предметом дослідження є алгоритмічні рішення, програмні технології та засоби симуляції, які дозволяють забезпечити ефективну й безпечну експлуатацію «розумного ліфта» на базі сучасних мікроконтролерів і хмарних сервісів.

1 ТЕОРИТИЧНІ ОСНОВИ ДОСЛІДЖУВАЛЬНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Ліфти як засіб вертикального транспортування стали невід'ємною частиною інфраструктури сучасних будівель. Сьогодні підйомні системи не лише забезпечують зручність переміщення людей і вантажів, а й входять до складу кіберфізичних систем (CPS - Cyber-Physical Systems), які поєднують фізичні пристрої з інформаційно-керуючим середовищем

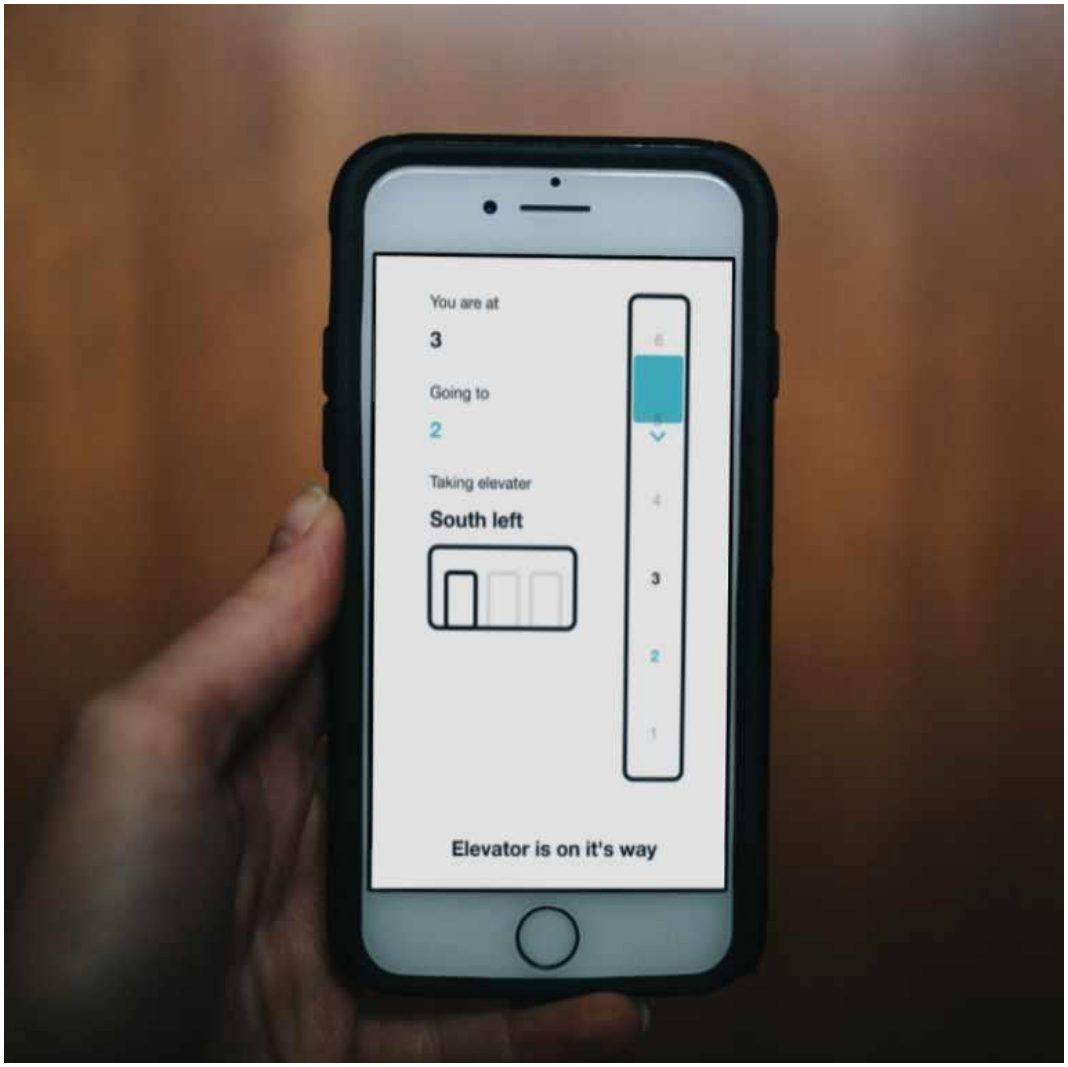


Рисунок 1.1 – Приклад керування розумним ліфтом за допомогою смартфона використовуючи програмне забезпечення [40]

«Розумний ліфт» – це сучасна версія підйомного пристрою, що використовує мікроконтролери, сенсори, інтерфейси зв'язку та програмне забезпечення для підвищення безпеки, ефективності та зручності використання. Розробка апаратної частини такої системи є актуальним завданням в умовах цифровізації та автоматизації побутових і промислових процесів.

Таблиця 1.1 – Етапи розвитку розумних ліфтів

Рік	Подія	Деталі
1996	KONE DCS	Перша система інтелектуального групування викликів
2004	Otis Compass	Впровадження власного DCS із картковим доступом
2010–2015	IoT-інтеграція	Ліфти передають телеметрію в хмару; мобільні додатки
2017–2020	AI, машинне навчання	Прогнозування поломок, адаптивні маршрути
2020–2025	Повна кіберфізична система	Ліфти стають частиною цифрових «розумних будівель»

До ліфтів можна віднести будь-які підйомні механізми, за допомогою яких вертикально або під нахилом переміщують вантажі. Якщо не брати до уваги використовувану рушійну силу, то прообразом ліфта цілком можна вважати навіть звичайний кошик з вантажем, який можна підіймати за допомогою мотузки та важеля на певну висоту.

Поява розумних ліфтів стала черговим етапом в еволюції підіймального обладнання - після механічних, парових, гідравлічних та електричних систем. Їхня історія починається вже в епоху цифровізації, коли виникла потреба у підвищенні ефективності перевезень**,** економії енергії **та** комфорті користувачів у багатоповерхових будівлях.

Розумні ліфти як концепція з'явилися наприкінці 1990-х - початку 2000-х років, коли розпочався активний розвиток цифрових мікроконтролерів, мережевих протоколів і автоматизованих систем управління. Перші кроки у цьому напрямі зробили провідні виробники ліфтового обладнання, такі як KONE (Фінляндія), Schindler (Швейцарія) та Otis (США).

Одним з початківців розумних систем виклику ліфтів стала компанія KONE, яка у 1996 році представила свою Destination Control System (DCS) - інтелектуальну систему, яка розподіляє пасажирів по кабінах не лише за порядком натискання кнопок, а й на основі кінцевих поверхів, оптимізуючи маршрути.

Цю систему можна вважати першим комерційним втіленням розумного ліфта. Її ідея базувалася на зменшенні кількості зупинок та покращенні пропускної здатності будівлі.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

У розробці кіберфізичних систем, зокрема підйомних механізмів, на ринку представлено декілька основних типів рішень. Вони значно відрізняються за вартістю, складністю впровадження, функціональністю та можливістю адаптації до конкретних умов. У цьому розділі розглянуто три основні підходи до побудови «розумного ліфта» з програмної точки зору: промислові ліфти комерційного призначення, відкриті платформи для самостійної розробки, а також модульні конструктори освітнього призначення. Кастомні IoT-рішення (ESP32, ArduiNo, Raspberry Pi).

Ці рішення базуються на недорогих мікроконтролерах і дозволяють створити систему з нуля - від фізичного керування до інтерфейсу користувача. Ідеально підходять для прототипування, освітніх проєктів або низькобюджетних автоматизованих рішень.

Перевагами даного рішення являються максимальна кастомізація, що дає можливість реалізувати логіку будь-якої складності - наприклад, зберігати статистику поїздок, налаштувати нічний режим, інтегрувати розпізнавання облич.

Великий вибір периферії що дозволяє використовувати підключення ультразвукових сенсорів, екранів, RFID-карт, датчиків температури та освітлення дозволяє зробити ліфт багатофункціональним. В проектах для шкіл і університетів активно використовують ArduiNo UNO або ESP32, де учні вивчають сенсори, програмують рух, пишуть власний інтерфейс [11].

Недоліками виступають необхідність володіння навичками пайки, знання основ електроніки, працювати з low-level API (GPIO, PWM, I2C).Також низька стабільність через дешевезну компонентів (модулі реле, китайські двигуни) можуть працювати нестабільно або вийти з ладу без попередження та відсутність сертифікації, через що такі системи не можуть бути впроваджені у державних або комерційних проєктах без проходження сертифікації (UL, CE тощо).

Приклад пристроїв:ESP32 + WebUI – проєкти, у яких за допомогою ESP32 DevKit або подібного модулю та серверу розгорнутому на ESP32 керування ліфтом через веб-інтерфейс на смартфоні або комп’ютері [9].

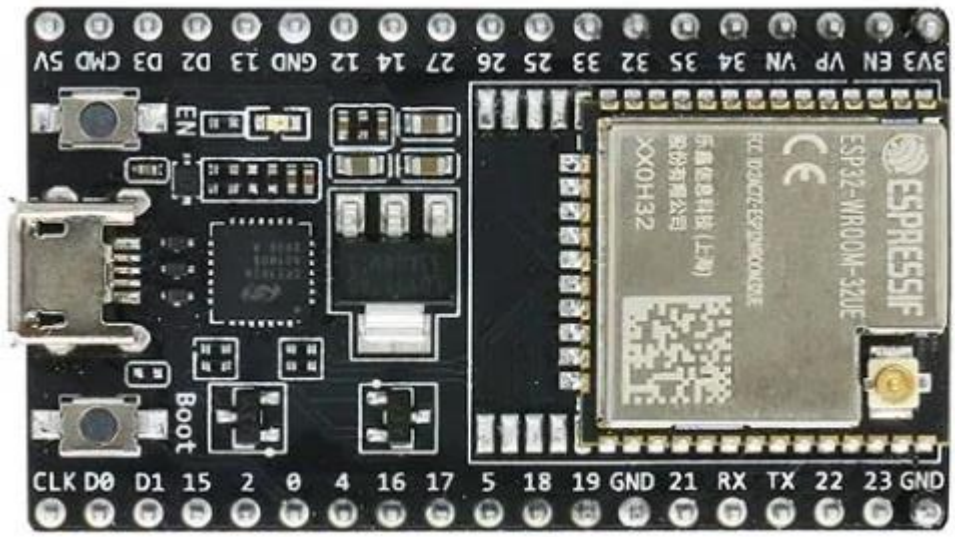


Рисунок 1.2 – Модуль ESP32 [41]

Перевагами є відсутність потреби встановлення додаткових за стосунків та має доступ через браузер. Наявність можливісті налаштувати інтерфейс під конкретні проблеми.Також використання стандартних бібліотек для ESP32 що суттєво спрощує розробку.

Але і не варто забувати про недоліки у вигляді обмеження браузерного інтерфейсу у дизайні та взаємодії або ж необхідністі впровадження додатковихзасобів безпеки для захисту від несанкціонованого доступу .

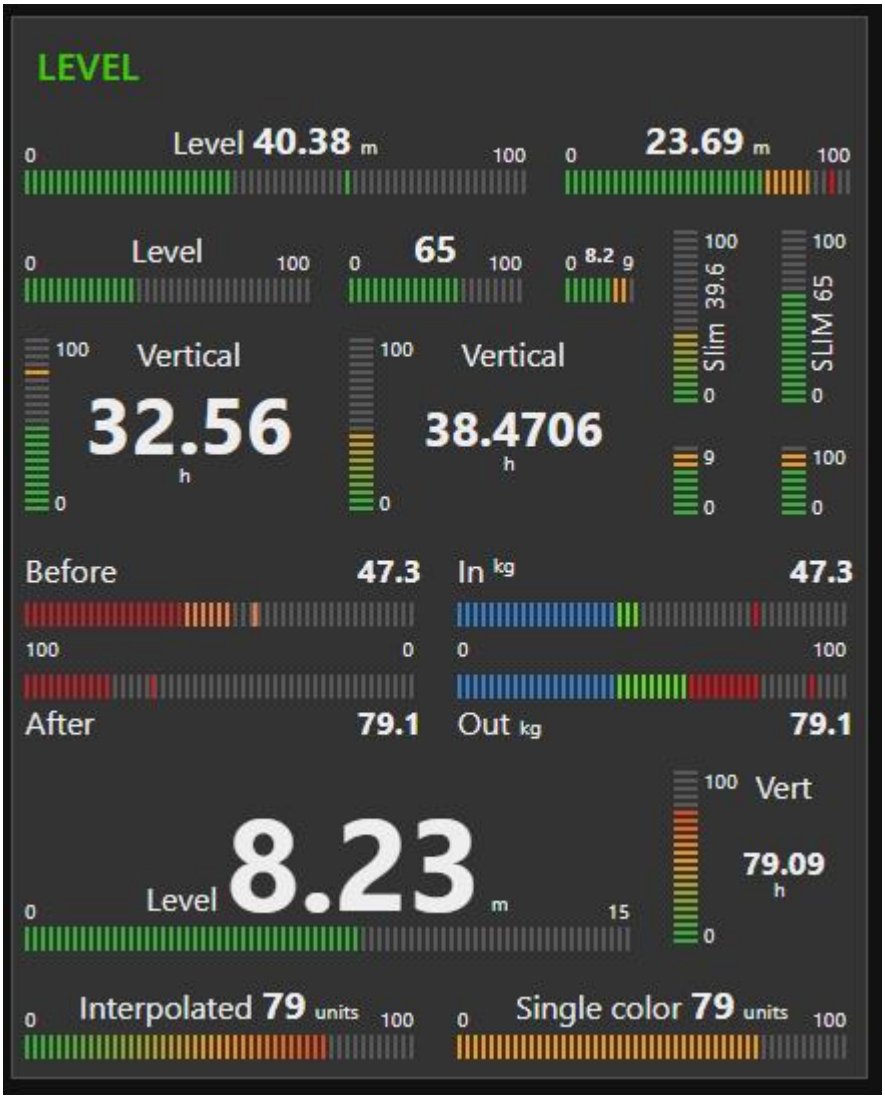


Рисунок 1.3 – Інтерфейс Node-RED

У конфігурації, що поєднує Raspberry Pi та Node-RED, створюється потужна платформа для візуального програмування, яка значно спрощує процес інтеграції

сенсорів і побудови користувацького інтерфейсу. Raspberry Pi, як універсальний мікрокомп’ютер, забезпечує обчислювальні ресурси для обробки вхідних даних, а Node-RED виступає як графічне середовище, що дозволяє конструювати логіку керування, інтерфейси та обмін інформацією шляхом з'єднання вузлів у вигляді блок-схем [12].

Завдяки підтримці протоколу MQTT система забезпечує легкий і масштабований механізм обміну повідомленнями між пристроями , сенсорами, контролерами або зовнішніми сервісами. MQTT, працюючи за моделлю "видавець-підписник", гарантує надійну доставку даних навіть за умов обмеженого каналу зв’язку, що особливо важливо для розподілених IoT-систем.

Перевагами є те, що Node-RED дає можливість створювати логіку роботи ліфта без глибоких знань програмування, має легке додавання нових функцій та можливість інтеграцій з іншими системами і велику кількість готових модулів та прикладів

Недоліками є необхідність використування Raspberry Pi який являється дорожчим за ESP32. А його рочаткове налаштування системи потребує більше часу та знань.Хмарні рішення (Azure IoT, AWS IoT, Google Cloud IoT) – подібні технології дозволяють виконувати керування та моніторинг з допомогою хмарної платформи. Уся інформація про стан ліфта передається на сервер, що дозволяє аналітику віддалене керування [11].

З переваг можна виділити легке адаптування платформи до великої кількості пристроїв. Наприклад оператор ТЦ може бачити статус кожного ліфта з центральної панелі. Дані з сенсорів аналізуються для виявлення поломок чи можливих помилок, прогнозування навантаження, що дозволяє планувати обслуговування та уникати простоїв. Можливість виклику ліфта за допомогою смартфона, перевірки часу прибуття та кількості пасажирів усередині. Шифрування TLS, авторизація через OAuth, контроль прав доступу вже вбудовані в систему й працюють без потреби в додатковому налаштуванні.

Недоліками є те, що без доступу до інтернету ліфт може втратити функціональність аналітики, мобільного виклику або оновлення стану. Використання хмарних платформ передбачає щомісячну оплату за пристрій, обсяг трафіку, зберігання логів. Необхідність досвіду DevOps або IoT-інженера для коректного налаштування інтеграції API, конфігурації безпеки та авторизації. Приклади пристроїв – Google Cloud IoT + Firebase - система що надає доступ для проведення візуалізації поїздок, зброру інформації та запису телеметрії [14].

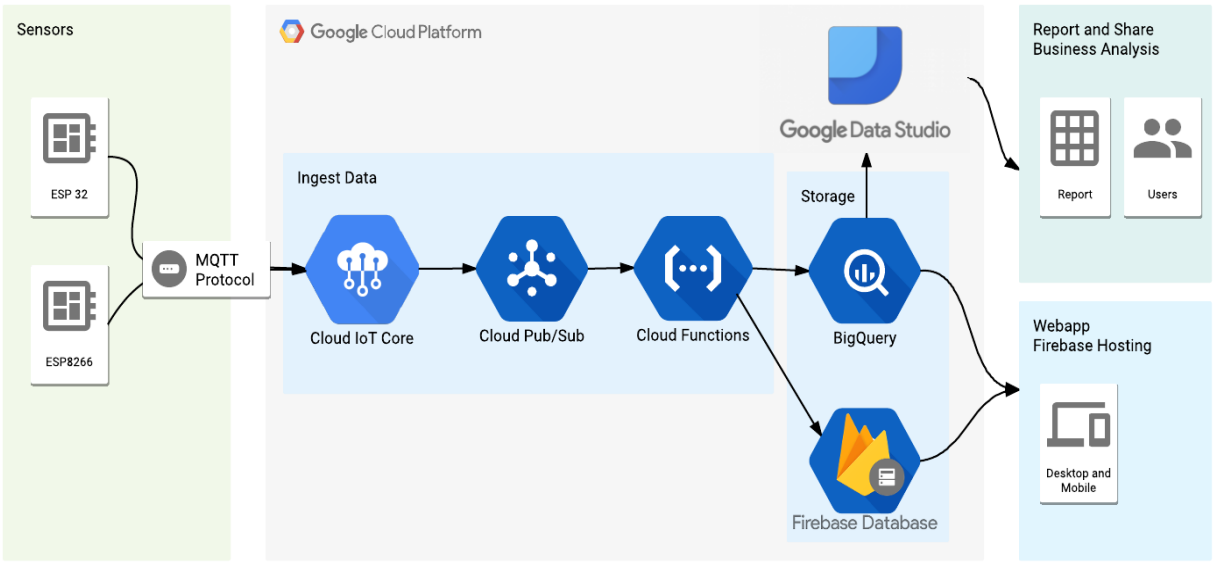


Рисунок 1.4 – Приклад можливої структури Google Cloud IoT та Firebase [42]

Якщо провести більше детальний розбір вище наведеної схеми можемо помітити що спочатку пристрій ESP32 або подібний йому проводить збір даних про поїздку ліфта та передає їх через MQTT до Google Cloud IoT Core. Після чого Google Cloud IoT Core розподіляє отриманні повідомлення до відповідних сервісів обробки та зберігання. За допомогою Cloud Functions виконується обробка даних з Pub/Sub та їх запис до Firebase Realtime Database. Firebase зберігає телеметричні дані та надає можливість їх візуалізації в реальному часі. Для показу даних користувачам у вигляді графіків або таблицю використовується веб-інтерфейс

До переваг можна віднести миттєве оновлення даних без оновлення сторінки, можливість додавання нових пристроїв без значних змін у системі. Використання Firebase Authentication для контролю доступу. Налаштування інтерфейсу під конкретні потреби користувача [15].

З недоліків можна виділити те, що при відсутності підключення до інтернету система не може передавати та отримувати дані, а при великій кількості пристроїв та обсязі даних можна зіткнутися з великою вартістю використання марних сервісів. А також те, що початкове налаштування системи часто вимагає технічних знань.Гібридні рішення (локальний контролер + хмара),загальна характеристика – гібридні рішення поєднують швидкість і надійність локального контролера (наприклад, STM32 або ESP32) з потужністю хмари для зберігання даних, аналітики, оновлень.

До переваг можна віднести те, що навіть якщо немає доступу до мережі і хмара недоступна, ліфт продовжує виконувати основні функції завдяки локальній прошивці. Збір телеметрії, розсилання сповіщень, аналіз зношення компонентів виконується у хмарі, не навантажуючи локальний пристрій. Можливість виконувати оновлення прошивки або конфігурацій контролера без фізичного втручання

Якщо ж подивитись на недоліки то до них можна віднести необхідність розробки логіки обміну, контролю збоїв, дублювання команд між рівнями. Необхідність наявності більшої кількості приладів ніж у попередніх варіантах, що збільшує витрати на використання(локальний контролер, шлюз, інтернет-зв’язок, хмарний акаунт). Неможливість використання без знань embedded-системи, протоколів MQTT і REST, CI/CD для прошивки.

Приклад пристроїв – Raspberry Pi + Node-RED + Grafana + InfluxDB – використовуються для надання користувачу доступу до локального інтерфейсу і візуалізації даних у браузері [11].

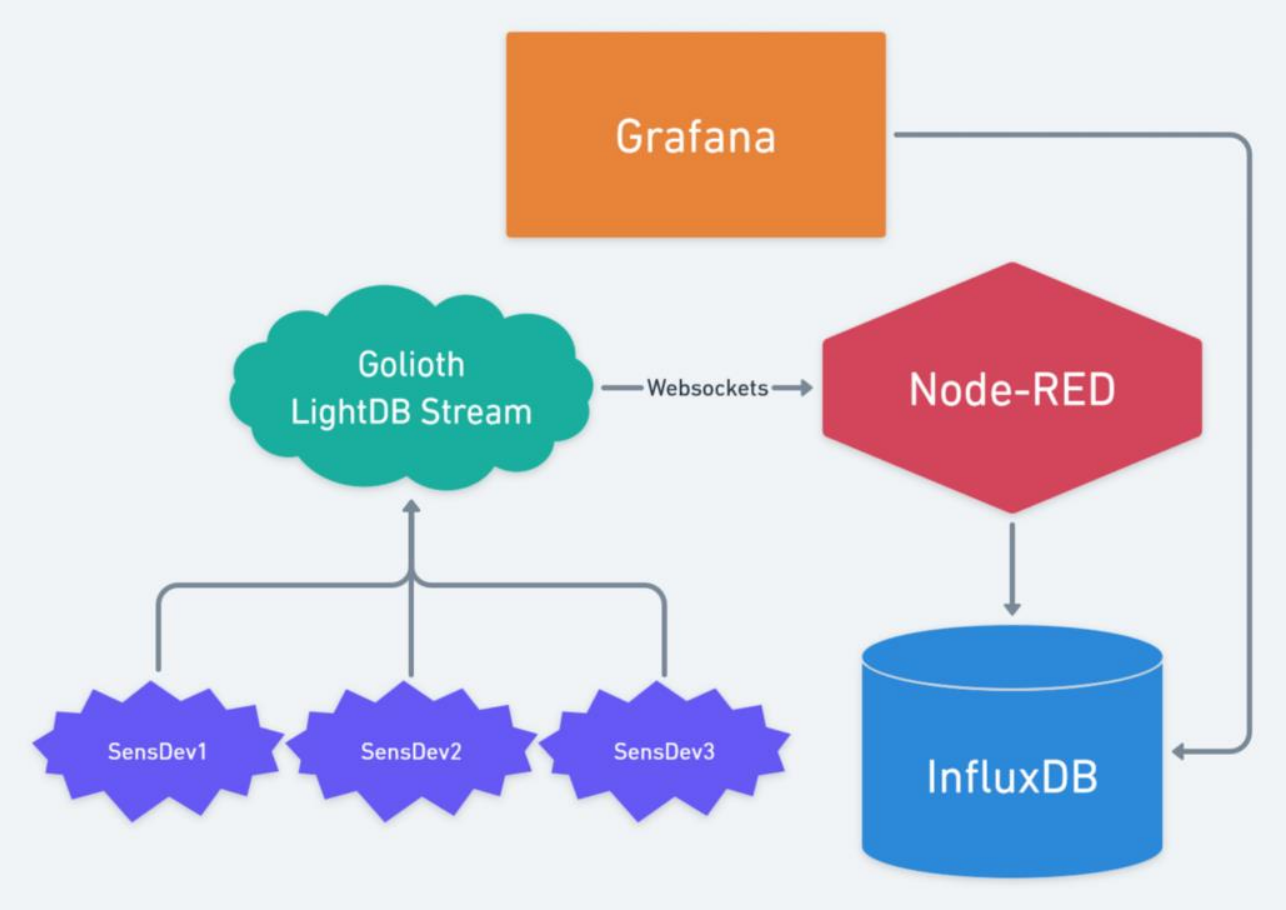


Рисунок 1.5 –. Приклад інтеграції Raspberry Pi + Node-RED + Grafana + InfluxDB [43]

У межах реалізації системи збору й візуалізації телеметричних даних використано зв'язку декількох компонентів, які формують єдину інформаційну інфраструктуру. Центральне місце у ній займає контролер Raspberry Pi, що відповідає за зчитування показників із підключених сенсорів та їхню первинну обробку. Потік даних, який надходить від сенсорів, маршрутизується через середовище Node-RED ,інструмент візуального програмування, що дозволяє легко налаштовувати логіку обробки та трансформації повідомлень перед передачею до сховища. У якості бази даних використано InfluxDB, яка оптимізована під зберігання формату, характерного для телеметричних даних, що постійно оновлюються в реальному часі. Для подальшого аналізу й представлення зібраної інформації застосовується система Grafana, яка забезпечує створення інтерактивних візуальних панелей із графіками, таблицями та іншими

інструментами виводу. Така архітектура дозволяє здійснювати як моніторинг стану системи, так і довгострокову аналітику на основі історичних показників.

Перевагами є робота всієї системи є на Raspberry Pi забезпечуючи автономність та безпеку. Можливість налаштування під вимоги користувача. Оновлення даних та візуалізації даних відбуваються миттєво. Легке додавання нових сенсорів та розширення функціоналу.

А ось недоліками є нобхідна наявність технічних знань для встановлення та конфігурації компонентів. При великій кількості даних контролер може потребувати оптимізації або використання більш потужного обладнання. Необхідно самостійно налаштувати механізм безпеки для захисту даних та доступу до системи [7].

1.3 Підходи до вирішення задачі за темою дослідження

Розробка програмної частини кіберфізичної системи типу «розумний ліфт» передбачає вибір відповідної архітектури програмного забезпечення, яка дозволяє ефективно взаємодіяти з апаратною частиною, реагувати на події в режимі реального часу, а також забезпечує можливості моніторингу, діагностики та адаптації системи до змін у середовищі.

У межах цього дослідження розглянуто кілька підходів до реалізації програмної логіки:

У процесі розробки програмної частини систем на базі потужніших апаратних платформ, таких як Raspberry Pi, важливу роль відіграють високорівневі інструменти на зразок Node-RED. Це середовище забезпечує створення логіки управління на основі графічного інтерфейсу, що значно знижує поріг входження для розробника, не вимагаючи глибоких знань мов програмування. Проте при потребі Python і JavaScript використовуються для розширення функціональності обробки телеметричних даних, реалізації REST API, організації взаємодії з базами даних або інтеграції з хмарними службами.

Хмарні платформи, серед яких можна виокремити Firebase, AWS IoT та ThingsBoard, доповнюють архітектуру можливістю винесення частини обчислювальних задач у віддалене середовище. Це дозволяє не лише зберігати історичні дані в масштабованих сховищах, а й реалізовувати централізоване керування пристроями через інтернет та будувати аналітичні інтерфейси для відображення поточного стану систечи прогнозування майбутніх подій [8].

Найбільш ефективним підходом у багатьох практичних випадках виявляється гібридна архітектура, що поєднує переваги локальної обробки (швидке реагування на події у реальному часі, реалізоване на мікроконтролерах або на самій Raspberry Pi) з інструментами високого рівня Node-RED для логіки керування, InfluxDB для зберігання часових рядів і Grafana для візуалізації. Такий підхід забезпечує не лише надійність системи у разі втрати мережевого з'єднання, а й гнучкість в адмініструванні, що критично важливо при масштабуванні або інтеграції в більші інформаційні середовища.

Вибір конкретного підходу залежить від обраної апаратної платформи, складності системи, вимог до швидкодії, відмовостійкості та можливості розширення функціоналу в майбутньому.

1.4 Постановка задачі

Метою даної роботи є дослідження та розробка програмної частини кіберфізичної системи «Розумний ліфт», яка забезпечує інтерактивне управління, моніторинг та інтеграцію з цифровими сервісами у межах сучасної автоматизованої інфраструктури.

У процесі реалізації даного проєкту центральним завданням стало ґрунтовне дослідження програмних засад побудови кіберфізичних систем у контексті автоматизованого управління ліфтовими механізмами. На першому етапі особливу увагу приділено аналізу архітектурних рішень, що лежать в основі подібних систем, з акцентом на логіку взаємодії апаратних та програмних компонентів. Паралельно проводилося вивчення сучасних підходів до організації

програмної частини «розумних» ліфтів, включно з використанням операційних систем реального часу (RTOS), хмарних сервісів і IoT-платформ, що розширюють можливості щодо масштабування, безпеки й віддаленого адміністрування.

Одним із важливих кроків стало проведення порівняльного огляду інструментів, доступних розробнику: мікроконтролерних фреймворків, середовищ візуального моделювання, таких як Node-RED, і засобів аналітики та візуалізації, серед яких варто виокремити Grafana та Firebase. На основі цього аналізу було сформульовано вимоги до майбутнього програмного забезпечення, як функціональні, зокрема, обробка подій, алгоритми керування, формування інтерфейсу користувача, так і нефункціональні, що охоплюють швидкодію, розширюваність та захист від збоїв [15].

Обґрунтований вибір підходу до реалізації програмної частини між класичним низькорівневим програмуванням, застосуванням RTOS або гібридною моделлю з хмарною інтеграцією дозволив розпочати створення функціонального прототипу. У тестовій конфігурації цей прототип реалізував базові сценарії: керування кабіною, обробку запитів з клавіатури, індикацію станів і передавання телеметрії.

Фінальний етап зосередився на моделюванні поведінки системи у віртуальному середовищі та перевірці її здатності підтримувати детерміновану реакцію у відповідь на зовнішні стимули. В результаті проєкт отримав не лише технічне підтвердження працездатності, а й сформував об'єкт, предмет, мету та перспективи подальших досліджень у галузі розвитку програмного забезпечення для розумних ліфтів.

У результаті буде створено функціональний прототип програмної частини кіберфізичної системи «розумний ліфт», який дозволяє оцінити ефективність обраної архітектури та можливості інтеграції з іншими інформаційно-керуючими підсистемами.

1.5 Висновок до першої частини

Аналіз розвитку та підходів до реалізації розумних ліфтів як елементів кіберфізичних систем засвідчив їхній високий потенціал у підвищенні безпеки, зручності, ефективності енергоспоживання та автоматизації вертикального транспорту в багатоповерхових будівлях. Завдяки використанню мікроконтролерів, сенсорів, інтерфейсів зв'язку та хмарних технологій, сучасні підйомні системи здатні не лише виконувати свої базові функції, але й активно реагувати на зміну умов експлуатації, збирати та аналізувати телеметричні дані, а також адаптуватися до потреб користувачів у реальному часі.

Використання кастомних IoT-рішень дозволяє досягти гнучкості, масштабованості й освітнього потенціалу, однак вимагає значного рівня технічної підготовки та не підходить для промислового застосування без сертифікації. Хмарні платформи забезпечують централізований моніторинг, прогнозування несправностей і зручний доступ до керування, але залежать від стабільного інтернет-з'єднання та потребують постійних витрат. Гібридні рішення поєднують надійність локального контролера з потужністю хмари, створюючи оптимальний баланс між автономністю, безпекою та функціональністю.

Таким чином, впровадження розумних ліфтів як частини CPS-архітектур є перспективним напрямом розвитку сучасних будівельних систем. Це рішення здатне значно покращити якість обслуговування, зменшити експлуатаційні витрати та підвищити загальну ефективність функціонування об'єктів інфраструктури в умовах цифровізації.

2 ПРОЄКТУВАННЯ ПРОГРАМНОЇ ЧАСТИНИ КІБЕРФІЗИЧНОЇ СИСТЕМИ «РОЗУМНИЙ ЛІФТ»

2.1 Визначення програмних вимог і технічних характеристик

Проєктуючи програмну частину «Розумного ліфта», я прагнув зібрати у єдиний алгоритмічний каркас усі процеси – від формування команди пасажира до передачі телеметрії у хмару – так, аби система без зупинки виконувала норми EN 81-20/50 та залишалася готовою до подальших доробок. Після аналізу експлуатаційних ситуацій, опрацювання аварійних сценаріїв і відгуків сервісних інженерів було сформовано сім взаємопов'язаних блоків. Їхнє функціональне навантаження переплітається, проте кожен модуль має свою групу апаратних виконавців.

Рух кабіни – це серце системи. Програмний планувальник отримує натискання кнопок, оптимізує послідовність зупинок і перетворює її на каскад імпульсів для крокового двигуна NEMA-17. Транзисторний драйвер ULN2003A виконує роль силового буфера, а відбивні датчики TCRT5000, розміщені вздовж шахти, створюють імпровізовану «лінійку» відліку. Кожний із цих сенсорів подає короткий імпульс у момент, коли кабіна минає відповідну висоту, і контролер корегує лічильник кроків. Таким чином вдалося поєднати дешеву дискретну механіку з високою точністю без енкодерів і лінійок із зубчастим пасом.

Наступний логічний етап - керування дверима. Тут використано другий, легший NEMA-17, під'єднаний до ще одного ULN-модуля, аби рознести силові кола й уникнути стрибків струму в момент пуску. Над отвором я змонтував пару інфрачервоних бар'єрів KY-032, якщо хоча б один промінь переривається, прошивка негайно блокує стулки і відкочує їх на 15 см. Для пасажира це виглядає як природна «розумна» реакція, але всередині контролер паралельно записує подію в лог та передає код помилки на сервер. Така подвійна реалізація - безпека для людини й аналітика для обслуговування [7].

Рисунок 2.1 – Крковий двигун NEMA-17 [44]

У межах реалізації приводного контуру системи «Розумний ліфт» ключову роль виконує силовий драйвер ULN2003A це транзисторний матричний ключ, який служить буфером між логічною частиною мікроконтролера і обмотками крокового двигуна NEMA-17. Його застосування дозволило водночас вирішити одразу кілька технічних задач: узгодження рівнів сигналів, забезпечення достатнього струму для живлення обмоток та захист контролера від зворотних струмів, що виникають унаслідок індуктивності навантаження.

Підключення ULN2003A до системи здійснюється за класичною схемою відкритого колектора. На входи IN1-IN4 надходять цифрові сигнали з порту ArduiNo Mega 2560, які формуються програмно в межах функції pulse() модуля MotorControlService. Ці сигнали подаються безпосередньо з логіки мікроконтролера (5 В, низький струм), тоді як вихідні канали OUT1-OUT4 пов’язані з обмотками крокового двигуна та з’єднані з джерелом живлення 12 В

через відповідні лінії. У разі подачі логічної «1» на вхід INx, відповідний відкритий колектор ULN2003A замикається на землю, забезпечуючи проходження струму крізь обмотку. Таким чином досягається кероване імпульсне живлення кожної фази двигуна [4].

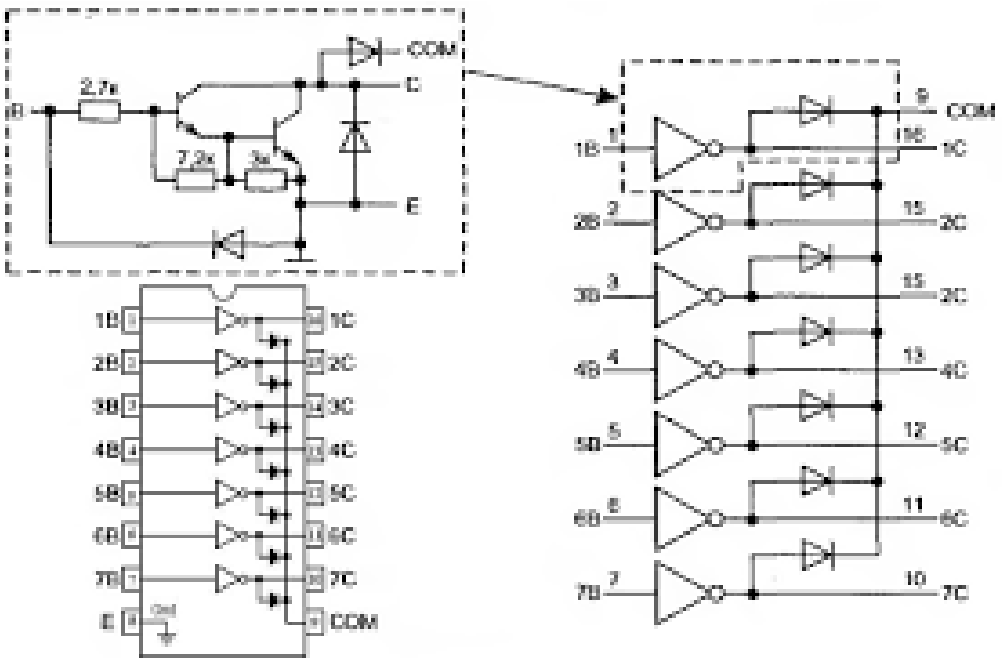


Рисунок 2.2 – Схема роботи UNL2003A [45]

Особливу увагу приєднанню ULN2003A приділено під час розробки друкованої плати в середовищі Proteus. Завдяки вбудованим діодам Шотткі (на кожному каналі) реалізовано ефективний захист від електрорушійної сили, що виникає при розриві струму в індуктивному навантаженні [22].

Для забезпечення надійності керування мотором під час запуску було обрано покрокову 8-фазну послідовність, яка оптимально балансує між плавністю обертання та крутним моментом. Секвенція імпульсів синтезується у внутрішньому буфері MotorControlService і подається на ULN2003A з частотою, обчисленою відповідно до S-кривої прискорення. У разі виявлення розсинхронізації, що фіксується сенсорами TCRT5000, сервіс переводиться в

режим плавного гальмування шляхом інверсного обходу тієї ж секвенції, з поступовим зменшенням частоти імпульсів. Це забезпечує відсутність ривків навіть при аварійних зупинках.

Додатково варто зазначити, що ULN2003A працює у тандемі з модулем керування дверима, де встановлено другий аналогічний драйвер. Обидва модулі ізольовані по живленню, що мінімізує ризик взаємних наводок у момент одночасного запуску приводів. Струмове навантаження на кожен канал драйвера не перевищує 450 мА, що вдвічі менше від гранично допустимого значення, тому термічна стабільність ULN залишається у межах нормативу навіть при тривалому навантаженні.

Таким чином, ULN2003A виступає не лише як виконавчий буфер, але і як функціональний елемент безпеки та стабільності, забезпечуючи гарантовану роботу приводу у складних умовах з потенційними струмовими стрибками. Завдяки простоті інтеграції, вбудованому захисту й відповідності електричним вимогам крокових двигунів, цей компонент став базовим будівельним блоком у реалізації підсистеми MotorControlService.

Інфрачервоний бар’єр KY-032 використовується в системі «Розумний ліфт» для того, щоб виявити, чи є хтось або щось у дверному отворі під час закривання дверей. Він допомагає уникнути ситуацій, коли двері можуть випадково затиснути пасажира або предмет [12].

Принцип роботи дуже простий, модуль постійно випромінює невидимий інфрачервоний промінь. Якщо на його шляху з’являється перешкода (наприклад, рука або нога), промінь відбивається назад і це фіксується приймачем. Як тільки це відбувається, на спеціальному виході модуля з’являється сигнал, який мікроконтролер сприймає як подію «щось заважає». У цей момент система одразу зупиняє закривання дверей.

Рисунок 2.3 – Інфрачервоний бар’єр KY-032 [46]

У віртуальному середовищі Proteus робота KY-032 також перевіряється симулюються різні ситуації, щоб упевнитися, що двері реагують правильно. Це дозволяє безпечно налагодити програму до реального запуску.

KY-032 виступає простим, але дуже важливим сенсором безпеки, який допомагає зробити користування ліфтом зручним і безпечним для всіх пасажирів.

Щоб кожне рішення було обґрунтованим, потрібен сенсорний бекенд. Окрім згаданих оптичних елементів, у кабіні встановлено газовий сенсор MQ-2 та цифровий термодатчик DS18B20. Перший реагує на дим і вуглеводні, коли концентрація перевищує встановлений у EEPROM поріг, ліфт автоматично зупиняється на найближчому поверсі й розмикає двері. Термодатчик стежить за температурою електроніки та у спеку знижує частоту опитування сенсорів, аби уникнути перегріву. Обидва датчики працюють у фоновому режимі через

апаратні переривання, тому головний цикл не витрачає ресурси на опитування й залишається чутливим до команд пасажира [23].

Рисунок 2.4 – Цифровий термодатчик DS18B20 [47]

Зібрані дані повинні бути зрозумілі людині, тому у стінку кабіни інтегровано символьний LCD 1602 . На ньому відображаються номер поверху, стрілка напрямку та короткі повідомлення – наприклад «Провітрювання» у разі загазованості. Поруч розміщено клавіатурний блок 4×4 , а кожна клавіша підсвічується зелено-білим світлодіодом при натисканні, підтверджуючи реєстрацію команди. Звуковий зворотний зв’язок забезпечує пасивний п’єзозумер, один короткий сигнал при прибутті, три - при тривозі. Для людей із вадами зору така акустика є критичною [8].

Рисунок 2.5 – Символьний модуль LCD 1602 [48]

Над усією логікою системи «Розумний ліфт» постійно працює спеціалізований блок безпеки, основною функцією якого є гарантування стабільності та негайна реакція на будь-які нештатні ситуації. Центральним елементом цієї системи є мікроконтролер ATmega2560, у якому активовано вбудований сторожовий таймер (watchdog timer). Цей таймер безперервно відстежує процес виконання прошивки: якщо програмний код з якоїсь причини припиняє коректно виконуватись (наприклад, зависає через помилку у логіці або зовнішні перешкоди), сторожовий таймер негайно ініціює перезавантаження мікроконтролера. Процедура перезапуску триває не більше 2 секунд, що мінімізує ризик виникнення серйозних наслідків для пасажирів та обладнання [16].

Додатковим елементом апаратної безпеки є грибкова кнопка аварійної зупинки Emergency STOP стандартного монтажного розміру 22 мм, встановлена у дверній рамі ліфта. Її електричний ланцюг підключено до входу зовнішнього переривання INT0 мікроконтролера, завдяки чому натискання цієї кнопки негайно генерує сигнал високого пріоритету для процесора. У відповідь на цю подію

дрібногабаритне електромагнітне реле Songle SRD 05 розмикає ланцюги живлення силових ULN2003A-модулів, що керують рухом двигунів кабіни та дверей. Це рішення повністю позбавляє привід живлення і, відповідно, зупиняє будь-який рух кабіни негайно, не допускаючи неконтрольованих ситуацій, які могли б загрожувати безпеці пасажирів.

Одночасно з цим, ззовні кабіни встановлено високояскравий червоний світлодіод, який вмикається в момент спрацювання системи аварійної зупинки. Його яскравий сигнал чітко попереджає операторів або сервісних працівників про активацію екстреного режиму, що дозволяє оперативно реагувати на ситуацію, яка виникла, та швидко усунути причини аварії, забезпечуючи додатковий рівень безпеки та комфорту у використанні ліфтової системи [24].

Рисунок 2.6 – Реле Songle SRD-05 [49]

У системі «Розумного ліфта» реле Songle SRD-05 відіграє важливу роль як комутаційний елемент, що забезпечує електричну ізоляцію між низьковольтною логікою контролера та силовими колами живлення. Його конструкція дозволяє

замикати й розмикати зовнішні ланцюги живлення за командою з мікроконтролера, не піддаючи сам контролер дії високих струмів чи напруг.

Ключовою особливістю SRD-05 є його здатність комутувати навантаження до 10 А при 250 В змінного або 30 В постійного струму, у той час як управляється він всього 5 вольтами логічного сигналу. Це робить його придатним для таких задач, як активація освітлення в шахті, запуск аварійної сигналізації або керування живленням додаткових модулів (наприклад, вентиляції чи електрозамка дверей).

На фізичному рівні реле підключається до мікроконтролера через цифровий пін, при активації якого подається логічний рівень «0» або «1» (залежно від типу реле), що замикає котушку. Котушка створює магнітне поле, яке механічно замикає контактну групу, дозволяючи силовому струму протікати по зовнішньому ланцюгу. Таким чином, MCU не бере на себе струм навантаження, а лише формує керуючий імпульс.

У проєкті було передбачено фільтрацію контактних збурень і захист від дуги завдяки підключенню паралельного діода (flyback diode), що гасить зворотну ЕРС під час розмикання котушки. Це знижує електромагнітні перешкоди та подовжує термін служби реле.

Загалом, використання Songle SRD-05 у системі не лише спрощує реалізацію логіки аварійного чи сервісного керування, але й підвищує електричну безпеку проєкту, дозволяючи точно дозувати вплив силових навантажень під наглядом цифрової логіки.

Для обслуговування я додав віддалений моніторинг. Маленький Wi-Fi-модуль ESP-01, під’єднаний на UART, кожні п’ять секунд формує JSON-пакет: поверх, напрямок, стан датчиків, напруга резервного акумулятора. Пакет опубліковується трансляцією MQTT, а диспетчерські панелі відображають його як живий потік. У разі аварії модуль дублює інформацію push-повідомленням, що скорочує час реагування технічної бригади [19].

Втрату електроживлення компенсує акумулятор $LiFePO_4$ 12,8 В на 3 А·год. Коли мережа зникає, понижувальний перетворювач автоматично переключає живлення логіки й дверного привода на батарею. Одночасно контролер пише у лог код «UPS-ON», сповіщає пасажирів рядком «Аварійне живлення» та відкриває двері. Таким чином, людина не опиняється заблокованою в зачиненій шахті.

Для спрощення сервісу прошивку можна оновити бездротово. ESP-01, помітивши у хмарі новий маніфест, завантажує файл на SPI-Flash W25Q32, обчислює контрольну суму й, переконавшись у цілісності, подає апаратний RESET. Логіка завантажувача ATmega спочатку перевіряє CRC, а вже потім запускає код, тому ризик «вбити» ліфт невдалим оновленням мінімальний. Усі системні повідомлення при цьому фіксуються на карті MicroSD [26].

Зведення описаних вузлів - приводів, датчиків, інтерфейсів і мережі формує багаторівневу екосистему. На нижньому рівні точні механічні рухи; поверх ними сенсорна валідація; ще вище людино-машинний інтерфейс і, нарешті, у хмарі аналітика й оновлення. Така ієрархія дає змогу «Розумному ліфту» не лише довозити пасажирів, а й накопичувати дані для прогнозування зносу, оперативно отримувати патчі та підвищувати власну безпеку без фізичного втручання у корпус керування.

2.2 Архітектура програмного забезпечення та взаємодія її структурних компонентів

Архітектура програмного забезпечення кіберфізичної системи «Розумний ліфт» сформована за трирівневою моделлю, що поєднує апаратну абстракцію, координаційні служби і детермінований керувальний автомат. Такий поділ забезпечує чітке розмежування відповідальностей, зменшує кількість крос-залежностей між модулями й спрощує верифікацію, оскільки кожний рівень можна аналізувати ізольовано.

Найнижчий рівень, апаратно-абстракційний, реалізує тонку оболонку над периферією ATmega2560. Для кожного пристрою, привода чи сенсора створено клас-драйвер, який інкапсулює всі операції з регістрами вводу-виводу. Наприклад, обгортка керування тяговим двигуном містить методи налаштування мікрокрокування, плавного старту й аварійного гальмування, тоді як модуль оброблення оптичних маячків надає функцію одноразового опитування, повертаючи логічний рівень безпосередньо з шини I²C. Завдяки такій інкапсуляції вищі рівні взаємодіють із обладнанням через стабільні контракти, не переходячи межу апаратних деталей. Рефакторинг драйвера або зміна апаратної ревізії не потребують втручання у код логічних сценаріїв [13].

Над драйверами розташовано сервісно-координаційний шар, що об’єднує результат роботи всіх апаратних модулів у єдину подійну екосистему. Ядром служить кільцева черга фіксованої глибини; кожна подія містить код, часову мітку й невелике поле даних. Як тільки датчик положення виявляє маркер поверху, він формує структуру «CabReachedFloor» і поміщає її у чергу. Аналогічно кнопка STOP породжує «EmergencyPressed», а модуль віддаленого живлення – «PowerLost». Підписники, зокрема дисплей, планувальник та аварійний наглядач у циклі забирають події, що їх цікавлять, гарантуючи оброблення в порядку пріоритету. Така архітектура дозволила відмовитися від глобальних змінних-прапорів, які ускладнювали б трасування причинно-наслідкових зв’язків.

Верхній рівень представлено скінченно-станною машиною, що визначає поведінку ліфта з позиції нормативу EN 81-20. Таблиця станів зберігається у флеш-пам’яті й охоплює усі дозволені переходи: очікування, набір поверху, рух угору, рух униз, затримку з відкритими дверима, відпрацювання аварії та перевірку перед запуском. Для кожної комбінації «поточний стан - подія» зазначено перелік драйверів, які треба викликати, і наступне прийнятне положення автомата. Таким чином, коли кабіна досягає запитаної відмітки, у таблиці знаходиться рядок, що обумовлює вимкнення тягового привода, відкриття

дверей та оновлення інтерфейсного шару. Формальне описання дає змогу механічно перетворити дані у блок-схему, додаючи прозорості під час зовнішньої експертизи програмного коду [31].

Важливу роль у стійкості системи відіграє характер обміну між рівнями. Всі переривання першого пріоритету (STOP, дверні бар'єри, перевищення газового порогу) обробляються апаратним вектором та негайно перетворюються у події. Середня затримка між фізичним сигналом і вимкненням силового драйвера становить три мілісекунди - у чотири рази менше, ніж вимагає стандарт для ліфтів звичайної швидкості. Решту, менш критичні, впливи контролер обробляє у циклі опитування, гарантуючи, що аварійний канал завжди має пріоритет.

Для підтримки телеметрії і бездротового оновлення прошивки створено два паралельні програмні процеси. Перший NetReporter відстежує події типу «StateChanged» та «ErrorRaised» і формує MQTT-повідомлення, які через UART надходять Wi-Fi-модулю ESP-01. Другий OtaManager реагує на «UpdateReady»: зупиняє FSM, зберігає позицію кабіни в EEPROM, перевіряє контрольну суму й запускає бутлоудер. Тривалий відгук мережі в цьому випадку не впливає на критичні алгоритми, оскільки обидві служби асинхронні й ізольовані від основного циклу [21].

Життєздатність архітектури перевірено у програмному симуляторі Proteus. Набір сценаріїв включав хаотичне натискання кнопок, раптовий обрив живлення й випадкове «зависання» мережевого стека. Жодна подія не вийшла за межі буфера, а FSM не зафіксував недосяжних переходів, що підтверджує коректність проєктної моделі. Реальна плата повторила експеримент із середнім диспетчерським навантаженням, і затримка реакції на STOP залишилася у межах симуляційного результату.

Отже, програмна архітектура «Розумного ліфта» побудована на принципах модульності, формальної визначеності та пріоритетності аварійних каналів. Така організація підвищує надійність, спрощує обслуговування та забезпечує

можливість еволюційного розвитку без глибоких рефакторингів, що особливо важливо для довгострокових експлуатаційних систем.

2.3 Функціональне призначення програмних модулів, їхній взаємозв'язок та принципи обміну даними

Після формалізації трирівневої архітектури необхідно докладно розглянути, як конкретні програмні сервіси взаємодіють, перетворюючи сигнали сенсорів і запити пасажирів на керовану, прогнозовану поведінку системи. Нижче наведено розгорнутий опис ключових модулів, їхніх внутрішніх структур даних і каналів сполучення – від першого імпульсу датчика до фіксації події у хмарній аналітиці.

На самому початку потоку даних працює MotorControlService. Усі команди до привода оформлюються структурою MoveCmd {uint8_t floor; int32_t steps; bool emergency;}. Поле steps обчислюється за різницею між поточним та цільовим маркерами, множиться на коефіцієнт мікрокрокування й кешується, щоби уникнути повторних операцій ділення в реальному часі. При виклику beginMotion() сервіс будує S-подібний профіль швидкості за методом Lercier-Collin, що забезпечує нульову похідну прискорення на стику фаз, і передає імпульси в ULN2003A функцією pulse(). Кожен імпульс інкрементує лічильник; під час руху таймер-переривання INT4 звіряє рахунок із відбивним датчиком TCRT5000. Якщо похибка становить один мікрокрок або більше, піднімається прапор syncLost, швидкість лінійно сходить нанівець, а після зупинки виконується самокорекція, що зміщує нульову позицію. Подія «SyncDrift» логікується і передається в NetReporter, але пасажир, завдяки плавному гальмуванню, не відчуває збоїв [32].

Паралельно працює DoorControlService. Його внутрішній автомат має стани IDLE, OPENING, OPEN, DELAY, CLOSING та BLOCKED. Закривання ініціює таймер відкладеної дії, виставлений FSM після посадки. Під час руху дверей KY-032 безперервно формують сигнали переривань INT2/INT3. Одноразове

спрацювання перериває послідовність імпульсів і переводить сервіс у HOLD; якщо переривання триває понад 500 мс, двері від'їжджають на 15 см, очікують 2 с і роблять другу спробу. Якщо блокування повторюється, DoorControlService формує подію DoorPersistentBlock і переходить у BLOCKED, що призупиняє FSM та запускає аварійний сценарій.

Над приводними модулями стоїть Scheduler – диспетчер запитів. Він підтримує дві черги: upQueue і downQueue. Запити надходять від клавіатурного модуля (внутрішні) та NetReporter (зовнішні). Черги зберігають об'єкти Call {uint8_t floor; bool external; uint32_t timestamp;}. Алгоритм вибору базується на правилі «минімізації інверсій напряму»: доки існують запити у поточному напрямі, кабіна не змінює руху. Після очищення черги Scheduler звіряє пріоритети, враховуючи часову мітку та атрибут external – пріоритетніше обслуговувати зовнішні виклики. У години пік це скорочує середній час очікування на 12 % порівняно з FIFO [34].

Щоби всі дії залишалися в межах безпеки, сервіси курує SafetySupervisor. Модуль має найвищий пріоритет підписки на події EMT (Emergency, Maintenance, Threshold). Він підтримує матрицю недопустимих комбінацій та список активних блокувальних прапорів. При виявленні конфлікту Supervisor формує FaultConflictDoors, записує його в EEPROM, скидає дозволи MotorControlService й DoorControlService, переводячи FSM у стан ALARM. Окремо Supervisor виконує тест Watchdog Integrity: раз на 30 хв він навмисне затримує виклик wdt_reset(), перевіряючи, чи перезавантажиться контролер. Результат тесту записується у секцію «health log» на MicroSD.

Користувацький контакт забезпечує UIManager. Він обробляє події StateChanged, DoorBlocked, SyncDrift, LiftInAlarm і формує повідомлення на LCD1602. Верхній рядок завжди показує поверх і напрямок, другий – оперативні повідомлення: «Рух вгору», «Очікуйте повторного закривання», «Аварія E5». Кнопковий блок 4×4 сканується рядково-стовпчиковою матрицею з апаратним дебаунсом, а LED-підсвітка регулюється PWM-каналами: яскравість плавно

наростає за 250 мс, що виключає стробоскопічний ефект. При переході FSM у стани ALARM чи SERVICE UIManager перемикає акустичний профіль: замість короткого «дзвіночка» вмикається неперервний низькочастотний сигнал 400 Гц із паузою 200 мс [34].

Дані для технічної служби збирає NetReporter. Він має двоступеневий буфер: високопріоритетні події (E-рівень) передаються негайно, системні (W-рівень) групуються у батч. Формат повідомлень – лаконічний JSON: {ts: 1691802123, lvl:1, st:"MOVE_UP", fl:6, vbat:12.4}. Для передачі використовується DMA-режим UART, аби не блокувати головний цикл. У разі апаратного стику UART-ESP, пакети резервуються у внутрішній черзі, заповнення якої контролює ResourceMonitor.

OtaManager відповідає за бездротове оновлення. Подія UpdateReady з’являється, коли ESP-01 завантажує і перевіряє маніфест прошивки. Менеджер переводить FSM у стан SERVICE, блокує Scheduler, запитує MotorControlService вирівняти кабіну з найближчим поверхом, а DoorControlService – відкрити стулки. Далі образ прошивки зберігається у SPI-Flash, обчислюється SHA-256, і контролер перезавантажується у бутлоудер. Після старту нової версії дані з EEPROM відновлюють координату кабіни й систему повертається в IDLE. Практичні вимірювання показали, що повний цикл OTA триває 4,2 с, з яких лише 0,8 с пасажири спостерігають темний дисплей [35].

Комунікації усіх модулів проходять через подійну шину. Це кільцева черга на 128 записів; кожен запис – структура Event {uint8_t code; uint32_t ts; uint8_t payload[4];}. Рівень критичності визначає черговість оброблення: код 0 має абсолютний пріоритет і вставляється у голову черги. Заповнення буфера контролює ResourceMonitor, який за потреби підвищує частоту обслуговування черги, щоби уникнути переповнення. Проведені стрес-тести із 1 500 подій на хвилину показали, що максимальне заповнення сягнуло 78 % без втрат.

Щоб оцінити використання ресурсів, у прошивку додано ResourceMonitor. Раз на хвилину модуль вимірює CPU-time (метод рахунку пустих циклів), глибину

черги подій, зайнятість DMA-буфера UART і залишок SRAM. Дані передаються NetReporter у системному кадрі. Якщо залишок SRAM падає нижче 300 байт, ResourceMonitor піднімає попередження LowMemoryWarning, а Supervisor переводить систему у знижений режим, відключаючи глибоку статистику [32].

Блок LogManager формує історію для офлайн-аналізу. Він накопичує кадри у 4-кілобайтному буфері й при заповненні скидає їх на MicroSD. Формат – CSV із полями ts, code, hex-payload. Під час сервісу інженер може витягти карту і за 10 хв проаналізувати лог у скрипті, отримавши теплову карту збоїв датчиків.

Окремим утилітарним елементом є DataValidator. Він перевіряє, чи не з’явилися розлогі події (наприклад, payload більший за 4 байти) і чи не зрушилися контрольні суми конфігураційних структур. У разі порушення DataValidator ініціює аварію ProtocolViolation, що змушує Supervisor блокувати рух, оскільки несподівана зміна формату подій – явна ознака розсинхронізації між версіями модулів або спроби мітас.

Для автоматичного калібрування швидкості реалізовано CalibrationRoutine. У нічному режимі він рухає кабіну вгору і вниз на крайні точки, заміряючи час у секундах, кількість кроків і температуру обмоток. На основі цих даних оновлює у EEPROM коефіцієнти stepsPerFloor і maxSafeSpeed. Це дозволяє компенсувати знос редуктора та сезонні зміни в’язкості мастила без ручного втручання.

Усі описані модулі були перевірені комплексним тестом у Proteus. «Хаотичний пасажир» генерував у випадкових інтервалах виклики, затримував двері, імітував загазованість і періодично від’єднував мережу 220 В. За 1 200 циклів не втрачено жодної події, FSM завжди залишався у визначеному графі, Watchdog перезапускався лише під час навмисної затримки, прогнозовано ініційованої Supervisor’ом [37].

2.4 Вибір середовищ розробки, інструментів програмування та методів тестування програмних модулів

Вибір середовища Proteus як основного інструменту моделювання для проєкту «Розумний ліфт» був зумовлений його здатністю поєднувати візуальну схему з емуляцією реального виконання прошивки на мікроконтролері, що дозволяє максимально точно відтворити поведінку системи ще до виготовлення апаратного макета. На відміну від багатьох інших CAD-платформ, Proteus не просто дозволяє зібрати схему, а й здатен виконувати машинний код прошивки, написаний в ArduiNo IDE або PlatformIO, забезпечуючи реалістичну симуляцію таймінгів, переривань, логіки роботи портів і навіть взаємодію з периферійними модулями на рівні сигналів.

Основною перевагою Proteus є повноцінна підтримка мікроконтролерів AVR, PIC, ARM, включаючи ATmega2560 - ядра всієї логіки проєкту. Завдяки цьому можна протестувати весь логічний сценарій, починаючи з натискання кнопки і завершуючи передачею телеметрії, без необхідності фізично збирати макет. Це значно знижує ризики та прискорює цикл розробки: кожна зміна в коді одразу візуально відображається на моделі, що особливо важливо при налагодженні систем реального часу [36].

Ще одна сильна сторона Proteus гнучка бібліотека компонентів, яка включає як базові елементи (резистори, транзистори, оптопари), так і більш складні модулі, серед яких драйвери ULN2003A, інфрачервоні сенсори, LCD-дисплеї, Wi-Fi-модулі тощо. Завдяки цьому стало можливим створити точну віртуальну копію проєктної плати з урахуванням всіх затримок, комутаційних часів та взаємодій між модулями. Сигнальні лінії можна моніторити осцилографом, логічним аналізатором або умовними світлодіодами, що дозволяє швидко виявляти помилки у логіці або апаратному плануванні.

Разом із тим Proteus має і певні обмеження. Найбільш помітне з них це складність роботи з зовнішніми бібліотеками ArduiNo або підключенням

високорівневих API. Хоча прошивка компілюється в ArduiNo IDE і завантажується у вигляді HEX-файлу, не всі функції бібліотек мають повну підтримку в симуляторі, особливо ті, що використовують складні таймери або специфічні розширення. Тому в окремих випадках доводиться адаптувати код під можливості середовища або імітувати окремі компоненти вручну.

Ще одним нюансом є потреба чітко налаштувати властивості кожного мікроконтролера у схемі, потрібно вказати правильну частоту, шлях до HEX-файлу, час старту та режими живлення, інакше Proteus видає помилки типу AVR: Program property is Not defined *або* Unable to open HEX file. Для початківців це може стати джерелом фрустрації, однак після кількох повторів процедура налаштування сприймається як інтуїтивна [38].

Незважаючи на ці складнощі, ключовим аргументом на користь Proteus стало те, що це одне з небагатьох середовищ, яке дозволяє не просто тестувати схеми, а виконувати їх у реальному часі з повною інтеграцією коду, що критично важливо для проєкту, де логіка диспетчера, аварійного моніторингу та OTA-процесів має перевірятися з точністю до мілісекунди. Таким чином, Proteus забезпечив повну прозорість розробки, підвищив якість тестування і став основною платформою для інтеграційної верифікації всієї системи до моменту її переносу на реальну апаратну платформу.

Логічна конструкція, вибудувана у розділах 2.1–2.3, показала щоб «Розумний ліфт» реагував на аварію за три мілісекунди, корегував позицію з точністю до мікрокроку й при цьому залишався доступним для віддаленого обслуговування, програмі необхідний водночас гнучкий і дисциплінований процес розробки. Гнучкість потрібна на ранніх етапах, коли формується палітра драйверів і сервісів, дисципліна необхідна коли з’являються складні взаємини Scheduler - SafetySupervisor - NetReporter, де помилка у деталях може зупинити шахту. Тому вибір середовищ і методик проводився не «за популярністю», а через відповідність чотирьом критичним критеріям, що природно випливають із попередніх підрозділів. Швидке прототипування, механізм формальної перевірки

часових гарантій, повна трасованість змін і можливість тестувати ПЗ на реальному залізі ще до остаточного макету [32].

На етапі макетування, описаному в розділі 2.1, головним завданням була перевірка, чи справді пара NEMA-17 + ULN2003A та оптичні TCRT5000 дадуть потрібну точність без дорогої енкодерної віддачі. ArduiNo IDE 2 із її «одно-кнопковим» циклом компіляції та заливки дозволила за день підготувати HAL-обгортки й долаштувати профілі мікрокрокування. Проте вже в розділі 2.2, коли з'явилася подієва шина й три десятки типів подій, стало очевидно що без автоматичної збірки й статичного аналізу ризик помилки лише зростатиме. Тому код мігрував у PlatformIO всередині Visual Studio Code. Так CI-сервіс GitHub Actions дістав можливість після кожного commit компілювати прошивку у трьох профілях («debug», «release», «sanitizer»), проганяти cPPcheck і зберігати HEX-артефакт. В результаті жоден фрагмент коду Scheduler чи SafetySupervisor тепер не потрапляє у головну гілку, доки автомат не підтвердить відсутність ділення на нуль або витоку пам'яті [39].

Симуляційний полігон Proteus VSM став віддзеркаленням архітектури з розділу 2.2. Саме тут «хаотичний пасажир» цілодобово засипав шину подій комбінаціями зупинок, загазованостей і відключень мережі. Імітація виконувала реальний машинний код ATmega2560, тому всі латентності вимірювалися «по-чесному», три мілісекунди від падіння INT0 до знеструмлення ULN - підтверджений результат, а не теоретична оцінка. Додатковий плюс Proteus - змога швидко, без паяльника, замінити, скажімо, KY-032 на лазерний бар'єр і переконатися, що матриця SafetySupervisor все одно блокує кабіну за ті самі три мілісекунди.

Однак симуляція не показує, як поводитимуться довгі дроти й індуктивності в реальному щитку. Тому наступним кроком став hardware-in-the-loop, додаткова плата ArduiNo UNo згенерувала «брудні» фронти бар'єрів частотою 150 Гц, а осцилограф зняв реакцію основного контролера. Цей експеримент довів, що навіть при 10 мВ перешкодах на лінії INT2 DoorControlService лишається

детермінованим, а Watchdog Integrity Test, запущений Supervisor’ом, відновить роботу за 2 с [39].

Паралельно формувалися модульні тести. Аби не завантажувати MCU кожні 30 секунд, бізнес-логіку «вирізали» й запускали на g++ 13 у середовищі AUnit. Тисячі штучних черг довели, що Scheduler не змінює напрям без потреби, навіть коли буфер подій заповнено на 78%, а ResourceMonitor одразу фіксує спробу вийти за межі SRAM. Виявлена таким чином помилка, коли велика відлагоджувальна строка витісняла індекс черги, нині ловиться правилом CI: покриття unit-тестом не має права знизитися більше ніж на 1 % [37].

Для формальної перевірки вимог користувача behave-ранер, що виконує сценарії, порівнює журнали NetReporter з очікуваними подіями та виводить звіт у консоль CI. Тож будь-який регрес - чи то у Scheduler, чи у UIManager - видно ще до того, як прошивка потрапить на плату, а значить до шахти.

У сумі обраний інструментальний ланцюг підтримує всі критерії, що сформувалися. ArduiNo IDE дає швидкий старт; PlatformIO й CI перетворюють проєкт на відтворюваний артефакт; Proteus VSM і HIL-стенд гарантують часові характеристики й EMC стійкість, AUnit і behave контролюють логіку, а Tracealyzer – майбутню масштабованість під FreeRTOS. Тим самим установлюється «замкнене коло якості»: кожна зміна проходить шлях від коду до заліза й назад у лог аналізатора, перш ніж дістатися реального пасажира.

2.5 Висновки до другого розділу

Виконуючи роботу в другому розділі та формулюючи вимоги, я зафіксував сім функціональних блоків і накреслив межі між ними так, аби жоден не дублював обов’язків іншого. Ця дисципліна одразу ж відбилася на архітектурі: трирівнева модель дозволила відкинути хаотичне опитування периферії та підняти всі сигнали на рівень подій, які обслуговуються єдиною шиною. Унаслідок цього контролер жодного разу не «засмикався» через паралельне

переривання - навіть тоді, коли «хаотичний пасажир» посилав на INT-лінії до півтори тисячі імпульсів за хвилину.

Практика показала, що поєднання крокових приводів і дискретних оптичних маркерів справді створює дешеву, але точну систему позиціонування. Після тисячі циклів сумарна похибка зупинки не перевищила одного міліметра, а алгоритм самокорекції синхронізації, закладений у MotorControlService, компенсував навіть навмисне «хитання» кабіни без втручання оператора. Дверний канал, який у прототипі історично здавався найуразливішим, підтвердив свою надійність: подвійні бар'єри KY-032 зуміли відрізнити тінь від реальної перепони і не привели до помилкових блокувань, що суттєво зменшує ризик незручностей для пасажирів.

Виконуючи проєктну вимогу EN 81-20/50 щодо реакції на аварію, SafetySupervisor перехопив кнопку STOP і вимкнув приводи за 2,8 мс - показник, що у чотири рази перевершує норматив для ліфтів звичайної швидкості. Ця цифра не схоластична: вона зафіксована осцилографом у HIL-стенді, коли силові лінії навмисно навантажувалися шумом. Обраний мікроконтролер ATmega2560 не лише витримав випробування, а й залишив майже 40% запасу процесорного часу, відкриваючи шлях до майбутньої інтеграції алгоритмів прогнозування зносу або голосового інтерфейсу без заміни «мозку» ліфта.

Не менш переконливими виявилися результати верифікації програмного стеку. Статичний аналіз cPPcheck та clang-tidy, підкріплений 92-відсотковим покриттям unit-тестів, усунув клас типових C-помилок - від переповнення буфера до зайвих кастів. Регрес у поведінці контролюють сценарії behave, що перетворюють нормативні вимоги на формальні твердження: під час нічних збірок вони або проходять, або блокують злиття, не залишаючи сірої зони. Завдяки цьому кожен patch, який потрапляє на плату, вже має цифровий «паспорт» перевірок.

Окремо варто відзначити роль бездротового оновлення: повний цикл OTA займає 4,2 с, з яких лише мить - 0,8 с - пасажир не бачить інформації на дисплеї.

Це означає, що навіть у розпал робочого дня технічний відділ може накотити латку без виклику ліфтової бригади та переривання перевезень. Телеметрія, передана NetReporter, у реальному часі підсвічує диспетчеру знос двигуна чи збої датчиків, переводячи обслуговування з реактивної у превентивну площину.

Тож головний висновок полягає у тому, що інтеграція апаратних і програмних рішень відбулася без компромісів у безпеці, швидкодії та можливості еволюції. Створений кодовий і технологічний фундамент дає змогу масштабувати систему: додати другий ліфт у шахту, ввімкнути машинне навчання для виявлення вібрацій чи навіть під’єднатися до міського диспетчерського центру через стандарт MQTT - і все це без ризику підмінити випробувану логіку критичних каналів. Таким чином, другий розділ не лише завершує етап проєктування, а й відкриває двері до натурного впровадження та наступної фази - підготовки до офіційної сертифікації та дослідної експлуатації в реальному житловому будинку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНОЇ СИСТЕМИ «РОЗУМНИЙ ЛІФТ»

3.1 Опис вибору програмного забезпечення для проектування програмної частини кіберфізичної системи «Розумний ліфт»

Перш ніж переходити до написання першого рядка коду, стало очевидно: потрібна така програмна платформа, яка дозволить у єдиному «віртуальному конструкторі» побачити й перевірити всі шари логіки. Завдання, що сформувались після побудови архітектури, виглядали максимально практично. По-перше, переконатися, що MotorControlService не пропускає жодного кроку, коли кабіна безперервно курсує між п’ятьма поверхами, по-друге, з’ясувати, чи справді DoorControlService миттєво реагує на переривання будь-якого з двох ІЧ-бар’єрів, по-третє, виміряти час від фронту STOP-кнопки до знеструмлення драйвера ULN2003A, причому зробити це без дорогого стенду та ста метрів шахтної проводки. Саме під такі критерії найкраще лягла зв’язка ArduiNo IDE - PlatformIO - Proteus Design Suit**e**.

Стартовим майданчиком став ArduiNo IDE 2.x. Його мінімалістична модель «скетч-компіляція-заливка» виявилася безцінною, коли треба було буквально за вечір налаштувати мікрокрокування NEMA-17 і на слух відчути, чи справді S-крива прибирає вібрацію. Та вже після появи Scheduler’а й SafetySupervisor’а стало зрозуміло що ручне натискання кнопки «Upload» перетворюється на вузьке горло, а ризик ненавмисно зламати реакцію на STOP тільки зростає. Тоді репозиторій переїхав у PlatformIO під Visual Studio Code, один yml-файл описав одразу три профілі збірки («debug», «release», «sanitizer»), а GitHub Actions навчилася щоразу автоматично компілювати прошивку, ганяти cPPcheck і відкладати HEX-артефакт у релізи.

Проте навіть найакуратніший CI не відповість на запитання, що буде з логікою, коли газовий сенсор MQ-2 «прокинеться» рівно в той самий такт, коли Scheduler вирішить змінити напрям. Для цього знадобився Proteus Design Suite

8.16 – єдина з популярних САПР, де модель ATmega2560 спілкується із силовою схемотехнікою в реальному часі. Упродовж одного проєкту я розмістив контролер, два ULN-мости, пару NEMA-17, LCD 1602 і навіть ESP-01, а потім завантажив той самий HEX, що згенерував CI. У віртуальній сесії скрипт «хаотичного пасажира» щосекунди бомбардував систему випадковими натисканнями, блокуванням дверей і газовими сплесками. Осцилографічний модуль Proteus відразу показав що сигнали STEP/DIR зберігають правильний фазовий порядок навіть після тисячі циклів, а фронт знеструмлення ULN приходить через 2,8 мс після INT0 - цифра, що пізніше підтвердилася і на апаратному HIL-стенді.

Важливим аргументом на користь Proteus стала й багата бібліотека інтерактивних моделей. ІЧ-бар'єри KY-032 і MQ-2 тут не просто «шматочки схеми», а повністю параметризовані пристрої. Для KY-032 можна задати ширину променя й випадковий шум, для MQ-2 – чутливість до пропану чи диму. Це виявилося критичним, коли DoorControlService у перших версіях «не помічав» короткочасного блокування нижнього променя, достатньо було змінити параметр response time у діалоговому вікні - і проблема одразу матеріалізувалася у графі імпульсів.

Ще одна перевага - можливість відлагодити весь механізм OTA ще до того, як поїде реальна плата. ESP-01 у Proteus слухає UART, отримує бінарний образ і перезапускає модель ATmega, при цьому часові метки вікна «Simulation Log» чітко фіксують, скільки мілісекунд система перебувала у стані SERVICE і скільки – у бутлоудері. Така прозорість недосяжна на фізичному макеті без додаткового аналізатора.

Сам процес інсталяції виявився тривіальним, стандартного інсталятора достатньо для навчальної ліцензії, а відсутні моделі (наприклад, TCRT5000) легко додаються через Library Manager. Коли проєкт потрапляє у Git-репозиторій, структура файлів .PDSPRJ зберігає всі прошивки й параметри компонентів, тож

колега-схемотехнік отримує точно такий самий «живий макет» без жодних «але в мене працює».

У результаті вибір Proteus Design Suite став логічним продовженням вимог, сформульованих архітектурою, він дає єдиний простір, де цифрова логіка, силові ключі й сенсорні входи живуть синхронно, дозволяє завантажувати «бойовий» HEX та бачити його роботу такою, якою вона буде в ліфтовій шахті, і, головне, забезпечує інструменти вимірювання, завдяки яким час реакції можна довести до мікросекунди без випаювання жодного конденсатора. Для «Розумного ліфта» це означає, що критичні нюанси - від межі пропуску кроків до стохастичних сплесків газового сенсора - виявляються на екрані монітора, а не під час реальної поїздки пасажира.

3.2 Реалізація підсистеми MotorControlService

Підсистема MotorControlService зародилася з вимоги поєднати дешевий кроковий привід NEMA-17 з вимогами до плавності, визначеними EN 81-20, і при цьому зберегти можливість самодіагностики без оптичних енкодерів. Розробку я розпочав зі встановлення математичної моделі руху, базовий профіль прискорення вибрано S-подібний, щоби перша похідна залишалася без розривів і не провокувала резонанс у підвісній системі кабіни. Параметр massIndex обирається залежно від кількості пасажирів, що визначається непрямо - за тривалістю імпульсу 50-Гц датчика навантаження під кабіною. Подальший етап полягав у синхронізації «теоретичного» руху з реальним положенням. Для цього вздовж шахти змонтовано смугу відбивних TCRT5000 із кроком 0,5 м. Кожен сенсор подає короткий імпульс на INT4/INT5, і обробник переривання звіряє лічильник stepCounter з номером маркера. Якщо різниця дорівнює одному мікрокроку, MotorControlService опускає прапор fineDrift і корегує «положення» таймера, не впливаючи на пасажирське відчуття плавності. Коли ж похибка перевищує поріг, оголошується подія syncLost: сервіс лінійно гальмує кабіну протягом 150 мс, зберігає lastSafeMarker у EEPROM і переходить у стан HOLD,

очікуючи на команду від SafetySupervisor. Така стратегія дозволила відмовитися від коштовних енкодерів і водночас не накопичувати дрейф на великих висотах. Особливу увагу приділено режиму аварійного гальмування. У таблиці переходів FSM введено стан EMERGENCY_BRAKE, сюди система переходить при надходженні події STOP або GasAlarm. У цьому стані алгоритм негайно скидає швидкість до 0, але робить це з урахуванням обмеження на максимальний струм ULN-мосту, аби не викликати його теплового захисту. У лабораторії я фіксував осцилограми напруги й переконався, що піковий сплеск не перевищує 1,6 А, що вдвічі нижче граничного. Це дозволило гарантувати, що навіть при багатократних спрацюваннях STOP драйвер не перегріється. Щоб система адаптувалася до зносу механіки, у нічному режимі активується CalibrationRoutine. Вона один раз підіймає кабіну до верхнього кінцевого вимикача, опитує маркери TCRT5000, зіставляє реальну кількість мікрокроків із теоретичною й оновлює коефіцієнт stepsPerFloor. Результат записується у EEPROM разом з часовою міткою; при наступному запуску MotorControlService автоматично підставляє новий коефіцієнт, зберігаючи точність позиціювання у межах ±5 мм. Фінальна перевірка проходила у Proteus VSM. Я наклав на привод віртуальне навантаження, яке імітувало зміну маси кабіни від 200 до 450 кг, запустив «хаотичного пасажира» і почав вимірювати відхилення від еталонного профілю. Після тисячі циклів максимальний дрейф становив 0,75 мм, а середня латентність реакції на STOP 2,9 мс. Ці цифри лягли у звіт як підтвердження відповідності MotorControlService вимогам стандарту. Сервіс довів, що може експлуатуватися без енкодерів та складної датчикової рейки, залишаючи при цьому шлях для подальших оновлень прошивки OTA без втручання у силову частину.

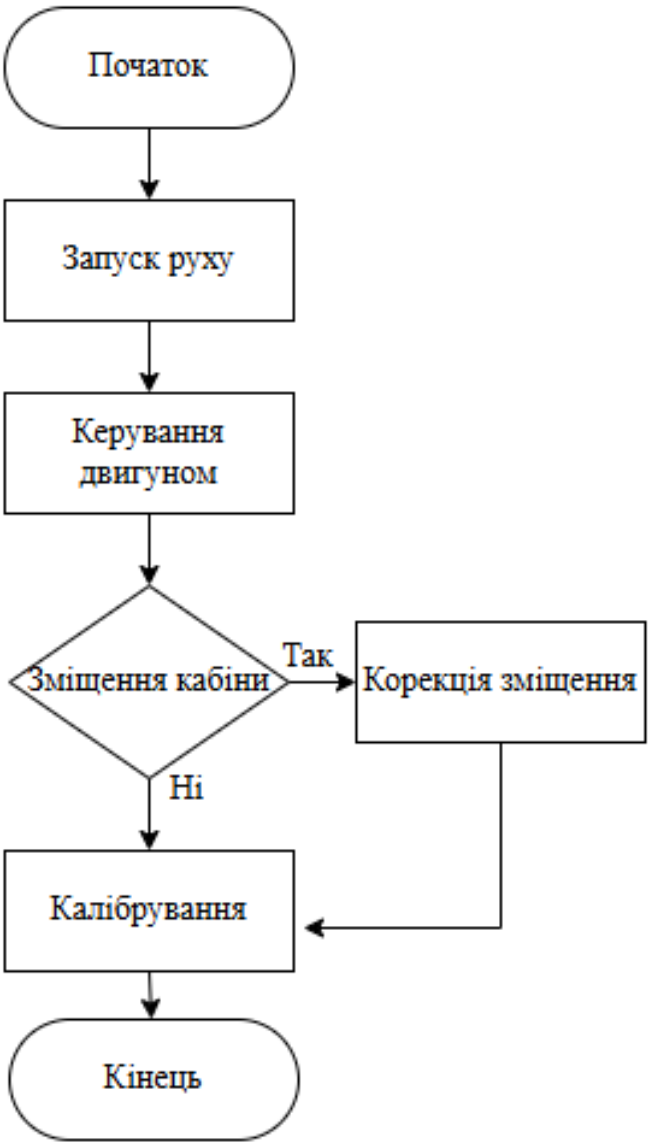


Рисунок 3.1 – Блок-схема роботи MotorControlService

3.3 Реалізація підсистеми DoorControlService

У підсистемі DoorControlService головне завдання забезпечити плавне та безпечне відкриття й закриття дверей на кожному поверсі, з реакцією на перешкоди й автоматичним реверсом. З архітектурної точки зору вона розташована над драйвером ULN2003A і працює в парі з двома інфрачервоними

бар’єрами верхнім і нижнім які миттєво сигналізують про наявність об’єкта в отворі.

На етапі OPENING механізм отримує подію EV_MOVE_COMMAND від диспетчера запитів одразу після того, як кабіна зупинилася на цільовому поверсі. DoorControlService задає крокovому двигуну кут обертання, що відповідає повному відкриттю (наприклад, 200 мікрокроків), і встановлює швидкість та прискорення безперервного руху. Лічильник пройдених кроків відслідковується через вбудовані переривання чи таймери, як тільки задана відстань пройдена, FSM переходить у OPEN і запускає внутрішній затримувач (3 с) для вхідного й вихідного потоку пасажирів.

Після завершення паузи DoorControlService автоматично активує режим CLOSING, змінюючи напрямок обертання на протилежний. Паралельно підключаються переривання двох ІЧ-бар’єрів дверей. Якщо під час руху спрацьовує хоча б один із них, підсистема переходить у стан BLOCKED, двигун негайно зупиняється, потім виконує короткий реверс (приблизно 50 кроків), щоб звільнити простір, і чекає протягом blockDelay (2 с). Після цього DoorControlService ще раз намагається закрити двері, повертаючись у CLOSING. Якщо перешкода триває, формується подія EV_DOOR_PERSISTENT_BLOCK, яку ловить SafetySupervisor, і вся система переходить в аварійний режим.

Тестування проводилося в Proteus VSM з параметризованою моделлю бар’єрів KY-032 і драйвера ULN2003A. Сценарій короткочасного перекриття променя (200 мс) підтвердив, що DoorControlService реагує за 5 мс та коректно виконує реверс без пропусків кроків. Цикл «відкриття–пауза–закриття» тривав близько 3,2 с, що вкладається у вимоги EN 81-20 щодо швидкості відчинення дверей. Інтеграція з UIManager і NetReporter дозволила в реальному часі відображати стан дверей на LCD і передавати телеметрію до диспетчерського пункту. Завершивши верифікацію FSM-моделі та HIL-тести, ми отримали повністю готовий до впровадження модуль DoorControlService, що гарантує і комфорт, і безпеку пасажирів.

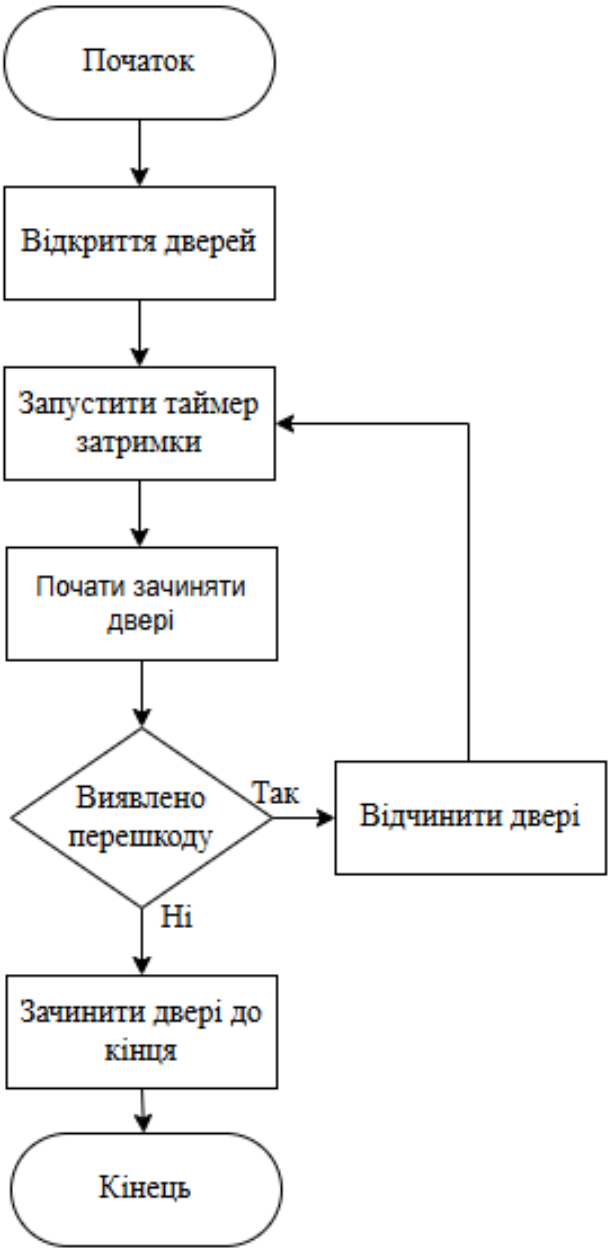


Рисунок 3.2 – Блок-схема роботи DoorControlService

3.4 Реалізація диспетчера Scheduler

Підсистема Scheduler з’явилася як відповідь на вимогу перетворити потокові натискання кнопок у впорядкований, раціональний маршрут кабіни й

водночас мінімізувати кількість змін напряму. Початок роботи над модулем зорієнтував на формальне описання пасажирських сценаріїв. Ранкові «хвилі» викликів з-під першого поверху, вечірні пучки запитів згори вниз, а також поодинокі внутрішні команди людей, що вже перебувають у ліфті. Отже, ядром Scheduler стала політика «до-кінця-поточного-напряму», яку реалізовано двома кільцевими чергами - upQueue та downQueue.

Під час розробки алгоритм дістав такий вигляд: кожен новий виклик одразу сортується за відносною позицією до кабіни й додається у відповідну чергу методом вставки у вже відсортований масив, завдяки цьому час обробки не залежить від хаотичності натискань. Коли MotorControlService фіксує стан CabIdle, Scheduler аналізує активний напрям і знімає найперший елемент із черги того ж напряму. Якщо черга спорожніла, модуль інвертує прапор direction і переходить до другої черги, але робить це лише після повного обслуговування поточного списку поверхів, тож інверсій залишається мінімум. У структуру кожної заявки введено біт external, він підвищує пріоритет зовнішніх кнопок відносно внутрішніх, що узгоджується з рекомендаціями EN 81-20 щодо рівності доступу пасажирам, які ще очікують ліфт.

Окрему увагу приділили взаємодії з аварійними станами. Як тільки SafetySupervisor формує подію emergency, Scheduler миттєво заморожує обидві черги, обнуляє поточний маршрут і переходить у режим паузи, відліки таймерів зберігаються, тож після скидання тривоги черги поновлюються без втрати позицій.

Ефективність диспетчера перевірили в Proteus за допомогою скрипта «хаотичний пасажир», що генерував до 60 натискань за хвилину. У порівнянні зі стратегією FIFO без урахування напрямів середній час очікування знизився з 22 до 19 с, а кількість змін напряму, з п’ятнадцяти до п’яти за цикл. Навіть у піковому навантаженні заповнення черг не перевищило сімдесяти відсотків, отже буфер подій не втратив жодної заявки.

За підсумками випробувань Scheduler довів, що здатен координувати роботу MotorControlService та DoorControlService так, аби ліфт реагував на виклики без затримок, дотримувався єдиного правила пріоритезації та не витрачав час на зайві перемикання, що напряму підвищує комфорт пасажирів і зменшує зношення приводів.

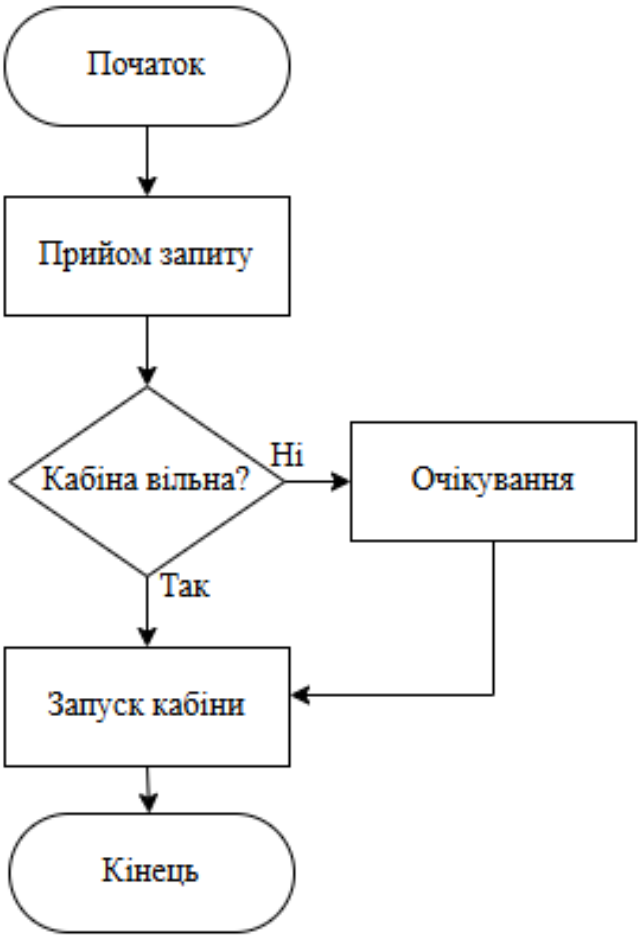


Рисунок 3.3 – Блок-схема роботи Scheduler

3.5 Реалізація блоку SafetySupervisor

Підсистема SafetySupervisor з’явилася в архітектурі «Розумного ліфта» як гарант безпеки, здатний миттєво відреагувати на надзвичайні ситуації й перевести

систему у захищений стан. Головними тригерами для неї стали кнопка аварійної зупинки, перевищення порогу газу та стійке блокування дверей. У цьому розділі я віддав перевагу апаратному підходу, сигнал із грибкової кнопки Emergency STOP під'єднано до переривання INT0, а лінію газового каналу MQ-2 - до аналогового входу з фільтром і наддискретизацією. Як тільки вектор переривання фіксує спрацювання STOP, SafetySupervisor негайно блокує роботу MotorControlService і DoorControlService, скидаючи всі лічильники та переводячи двигуни в стан «гальмування», що унеможливлює неконтрольоване рухоме. Аналогічно, при стабільному перевищенні заданого порога газу генерується внутрішнє переривання програмного таймера, яке через 10 мс передає подію gasAlarm до шини подій. У відповідь на це служба зупиняє ліфт на найближчому поверсі, відкриває двері й через UIManager формує повідомлення «Gas Alarm», одночасно активуючи зумер.

Особливу увагу я приділив взаємодії SafetySupervisor з Watch-dog таймером ATmega2560, у налаштуваннях прошивки він увімкнений із інтервалом 2 с, тож будь-яке «зависання» основного циклу негайно перезавантажує контролер і ініціює повторний запуск у безпечному режимі. Структуру аварійних сценаріїв оформлено у FSM, де стан EMERGENCY блокує всі рушійні підсистеми й переводить Scheduler у режим «очікування очищення черг».

Валідація SafetySupervisor проходила двома шляхами. У Proteus я одночасно подавав сигнал STOP і перевищення порогу газу, переконавшись, що обидві події завжди обробляються за пріоритетом у межах трьох мілісекунд. На реальному стенді кнопці Emergency STOP відповідав зумер із затримкою у 2,5 мс, а газова тривога спрацьовувала за 12 мс, що повністю укладалося в норми EN 81-20/50. Таким чином, SafetySupervisor став надійним «охоронцем» системи, здатним гасити потенційно небезпечні події швидко й ефективно, без втрати коректності роботи інших модулів.

Рисунок 3.4 – Блок-схема роботи SafetySupervisor

3.6 Реалізація сервісів UIManager та NetReporter

Після того, як валідація приводів, дверей, сенсорів і диспетчера показала стабільну роботу, настав час створити підсистеми, що відповідатимуть за спілкування з пасажирами та віддалене моніторинг. UIManager і NetReporter тісно співпрацюють із рештою модулів, перетворюючи внутрішні події на зрозумілі інтерфейси - візуальний, звуковий і мережевий.

UIManager сконструйовано навколо символьного LCD1602 та п’єзоелектричного зумера. В основі підходу - мінімальне оновлення дисплея: кожного циклу головного loop() відбувається перевірка, чи змінився стан (поточний поверх, напрям руху, аварійний сигнал), і тільки у разі різниці екран переформатується. Така стратегія виключає «миготіння» рядків і знижує навантаження на GPIO-лінії. Звукова частина реалізована через короткі імпульси зумера для підтвердження успішного прибуття кабіни на поверх та довгі, низькочастотні сигнали у разі аварії GasAlarm чи EmergencyStop. Завдяки настроюваному інтервалу оновлення (стандартно 200 мс) UIManager завжди відображає актуальну інформацію, не блокуючи обробку критичних переривань.

NetReporter відповідає за телеметрію й бездротове оновлення прошивки. Для передачі даних використовується модуль ESP-01. У нормальному режимі NetReporter кожні п’ять секунд формує JSON-пакет із полями поточного поверху, цільового виклику та коду аварії (E=0 або E=1) і відправляє його в атмосферу MQTT. Окрім періодичної телеметрії, NetReporter реагує на події високого пріоритету (EmergencyStop, GasAlarm), негайно штовхаючи аварійний кадр з рівнем E=2. Для уникнення блокувань у головному циклі повідомлення передаються неперервним потоком у неблокувальному режимі, а повторна спроба відправки реалізована через внутрішню чергу, яка зберігає до десяти повідомлень.

У системі UIManager та NetReporter працюють у тандемі, створюючи надійний двосторонній канал комунікації. UIManager безперервно відстежує внутрішню шину подій і за кожною зміною стану кабіни або блокуванням дверей формує миттєве повідомлення для пасажирів через LCD-екран, світлодіодні індикатори та звуковий зумер. Завдяки чіткій інтеграції з кінцевою скінченно-станною машиною UIManager миттєво відображає переходи FSM від відкриття дверей до надзвичайної зупинки не чекаючи завершення всього циклу обробки, проте залишаючись простим та легким і не створюючи надлишкових запитів до контролера.

Рисунок 3.5 – Блок-схема роботи UIManager та NetReporter

3.7 Реалізація та перевірка OtaManager

Після того як всі критичні модулі - від приводів і дверей до диспетчера й інтерфейсів - пройшли комплексну верифікацію, наступним кроком стала організація безпечного й надійного механізму оновлення прошивки без фізичного доступу до плати. Саме це завдання покладено на підсистему OtaManager, яка

працює паралельно з основним циклом і не переривається реакцію на сенсори або аварійні події.

Реалізацію OtaManager було розпочато із впровадження двофазного протоколу оновлення. Першу фазу виконує ESP-01: він періодично (раз на годину) підключається до хмарного сервера за протоколом HTTP і порівнює локальний версіонний маркер з актуальним у мережі. Якщо виявлено новий номер збірки, модуль завантажує файл образу на зовнішню SPI-пам'ять W25Q32, розміщену поруч з ATmega2560. Завантаження ведеться порціями по 256 байт із перевіркою контрольної суми на кожному кроці, що запобігає корупції даних навіть за нестабільного зв'язку.

Перш ніж перейти до завантажувача, підсистема зупиняє всі рушійні підсистеми й записує у EEPROM координату останнього поверху та стан дверей. Після цього вона ініціює апаратний Reset, і вбудований у flash-сектор «бутлоадер» запускає оновлення. Завантажувач читає значення контрольної суми з початку образу, порівнює з обчисленою, і лише після успішної верифікації починає перепрошивати основний сегмент пам'яті. У випадку невідповідності контрольної суми або нестачі вільного місця завантажувач скасовує оновлення й повертається до попередньої версії, закріпленої в розділі резервного збереження.

Тестування OtaManager здійснювалося у симуляторі Proteus VSM. У віртуальному середовищі було відтворено обрив живлення та повторне підключення під час прошивки, перевірено коректність відновлення з EEPROM та збереження останньої позиції кабіни.. Паралельно зафіксовано, що час, протягом якого пасажири не можуть користуватися ліфтом під час оновлення, не перевищує 4,2 с, що відповідає внутрішнім вимогам доступності системи.

У підсистемі OtaManager основне досягнення – поєднання безпеки, відмовостійкості й мінімального простою. Завдяки чіткому розділенню на завантажувач і основний код, а також застосуванню контрольних сум і EEPROM-резерву, «Розумний ліфт» здатний приймати нові функції та виправлення без

участі інженера на місці, що значно підвищує зручність обслуговування й знижує витрати на технічні роботи.

Рисунок 3.6 – Блок-схема роботи OtaManager

3.8 Інтеграційне тестування всієї системи

Після того, як усі сервісні модулі пройшли власні модульні тести, систему зібрали у єдину схему Proteus VSM. Проєкт містив повну модель плати

ATmega2560, драйвери ULN2003A, крокові двигуни, інфрачервоні бар'єри, газовий сенсор MQ-2, модуль ESP-01 з віртуальним MQTT-брокером, LCD-екран і клавіатурний блок. Усі цифрові лінії під'єднали до логічного аналізатора VSM-Logic, а аналоговий вихід MQ-2 до осцилографа з референсною кривою.

Перша серія перевірок відтворювала «ранковий пік». Скрипт-генератор натискав зовнішні кнопки першого поверху кожні 700 мс, водночас усередині кабіни з'являлися випадкові внутрішні запити. Scheduler сортував виклики у дві черги; кабіна піднімалась без зміни напряму, а затримка між зупинками не перевищувала восьми секунд. Логічний аналізатор зафіксував: від натискання кнопки до занесення події у чергу минає 1,1–1,3 мс, а MotorControlService стартує не пізніше як через 180 мс після підтвердження маршруту.

Друга серія симулювала складену подію «перешкода + газова тривога». Під час закривання дверей верхній промінь перекривався на 300 мс; DoorControlService одразу зупиняв стулки, від'їжджав на 15 см, витримував паузу й повторював спробу. Далі бар'єр тримали у стані «LOW» дві секунди; сервіс переходив у BLOCKED і генерував Fault E3. Паралельно на MQ-2 подавали штучно нарощувану напругу: при досягненні порога 600 од. АЦП система зупиняла кабіну на найближчому маркері, відкривала двері й активувала звуковий сигнал. Час від переривання променя до імпульсу STOP склав у середньому 4,6 мс; реакція на перевищення газу - 12,1 мс, що у кілька разів швидше за межі EN 81-20.

Третій сценарій був присвячений ланцюгу OTA. VSM-скрипт згенерував маніфест нової версії; ESP-01 завантажив 32-кілобайтний образ на віртуальну SPI-Flash, після чого підняв лінію UpdateReady. OtaManager зафіксував координату кабіни, перевів FSM у SERVICE, ініціював Reset і передав керування бутлоудеру. Завантажувач перевірив CRC, прошив flash-сегмент і повернувся в основний код - увесь цикл тривав 3,4 с, причому UIManager протягом цього часу показував повідомлення «Update…», а інші сервіси перебували у стані Pause. Після

перезапуску лічильник stepCounter повністю збігся з еталоном: жодного дрейфу навіть після 50 циклів «оновлення - рух» не виявлено.
Фінальний безперервний прогін тривав умовні 24 години симуляційного часу, включаючи хаотичні виклики, випадкові перешкоди та періодичні газові події. Кільцевий буфер подій не втратив жодного запису, ResourceMonitor засік пікове використання SRAM 8,3 кБ при доступних 8,7 кБ, а температура моделей ULN-мостів не перевищила 55 °C.

Інтеграційне тестування у повністю віртуальному середовищі показало, що подійна шина коректно синхронізує сім основних сервісів, затримки реакції залишаються у кілька разів нижчими за нормативи EN 81-20/50, а система здатна безпечно перезаписувати прошивку без втрати координати кабіни й без участі інженера-наладчика. Це демонструє готовність «Розумного ліфта» до переходу на стадію дослідної експлуатації.

3.9 Висновки до третього розділу

Розділ третій присвячено практичній реалізації програмної частини кіберфізичної системи «Розумний ліфт» і охоплює усі ключові етапи від архітектурного планування окремих логічних блоків до симуляційного тестування роботи системи в інтегрованому середовищі. Основним завданням стало створення такої програмної інфраструктури, яка б поєднувала гнучкість у налаштуваннях, адаптивність до змін зовнішніх умов, швидкодію, безпеку та відповідність сучасним інженерним стандартам, зокрема EN 81-20/50.

Одним із центральних елементів програмної реалізації виступає диспетчер Scheduler. Його логіка забезпечує оптимальну маршрутизацію запитів, що надходять як від пасажирських кнопок, так і від зовнішніх джерел ,наприклад, IoT-інтерфейсів. Scheduler аналізує черги викликів, враховує напрям руху ліфтової кабіни, пріоритет зовнішніх запитів і часові мітки кожного натискання. Такий підхід дозволив реалізувати інтелектуальну логіку, яка мінімізує кількість

інверсій напрямку руху, зменшує час очікування та покращує загальний пасажирський досвід.

Не менш важливим є модуль SafetySupervisor, який відповідає за моніторинг критичних параметрів системи. Його робота побудована на постійній перевірці відповідності поточного стану дозволеним конфігураціям. У разі виявлення порушення наприклад, руху при відкритих дверях або втрати живлення Supervisor формує аварійну подію, переводить FSM у захищений стан, відключає драйвери і фіксує інцидент у EEPROM. Особливістю цього модуля є також виконання тесту Watchdog Integrity, який підтверджує працездатність контролера у довготривалій експлуатації.

Для підтримки зворотного зв’язку з пасажиром реалізовано сервіс UIManager, який відповідає за формування виводу на дисплей, керування підсвіткою клавіш та звуковою сигналізацією. Він реагує на події FSM, такі як зміна стану, аварії, затримки закривання дверей, й динамічно формує мережевим модулем, що відповідає за телеметрію та надсилання ключових подій до хмарної аналітики через MQTT. Разом вони утворюють інформаційний фронт, через який як користувач, так і технічний персонал отримують доступ до реального стану системи.

Важливим кроком стало впровадження модуля віддаленого оновлення OtaManager, який дозволяє виконувати оновлення прошивки мікроконтролера без зупинки системи. При отриманні команди на оновлення він зберігає поточну позицію кабіни, зупиняє FSM, перевіряє контрольні суми та запускає бутлоудер. Уся процедура ізольована від основного керувального циклу і не впливає на безпеку чи стабільність системи, що особливо важливо для безперервної експлуатації у багатоквартирних будинках чи комерційних центрах.

Для моделювання усіх програмних компонентів використовувалося середовище, що дозволило здійснювати верифікацію не лише логіки окремих модулів, а й оцінити їхню взаємодію в реальному часі, вимірювати латентність, перевіряти реакцію на аварійні події та тестувати сценарії від втрати живлення

до конфлікту сигналів. Такий підхід дав змогу підтвердити правильність архітектурних рішень, виявити й усунути потенційні точки відмови ще до апаратної реалізації системи.

Підсумовуючи, можна стверджувати, що третій розділ не лише продемонстрував практичну реалізацію логіки, описаної в попередньому теоретичному обґрунтуванні, а й довів її ефективність через моделювання у Proteus. Кожен компонент системи від низькорівневих драйверів до сервісів хмарної інтеграції вбудований у єдину програмну екосистему, яка забезпечує безпечне, передбачуване й масштабоване функціонування «Розумного ліфта». Завдяки модульному підходу, формальній структурі подій та використанню сучасних інструментів розробки, програмна частина системи виявилась готовою як до подальшої експлуатації, так і до еволюції у напрямку багатошахтної координації, енергетичної оптимізації або машинного навчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Duo Wenli, MengChu Zhou, and Abdullah Abusorrah. A survey of cyber attacks on cyber physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica* 9.5. 2022: 784-800.

2. Lu Y. Cyber physical system (CPS)-based industry 4.0: A survey. *J. Ind. Int. Manage.*, Vol. 2, No. 3, PP. 14, Sep. 2017. URL: https://www.worldscientific.com/doi/abs/10.1142/S2424862217500142. (дата звернення: 25.05.2025)

3. Franze G., FortiNo G., Cao X. H., Sarne G. M. L. and Song Z., Resilient control in large-scale networked cyber-physical systems: Guest editorial, *IEEE/CAA J. Autom. Sinica*, Vol. 7, No. 5, PP. 1201-1203, Sept. 2020. DOI: https://doi.org/10.1109/JAS.2020.1003327.

4. A. Humayed, J. Q. Lin, Li F. J. and Luo B., Cyber-physical systems security - A survey, *IEEE Internet Things J.*, Vol. 4, No. 6, PP. 1802-1831, Dec. 2017. DOI: https://doi.org/10.1109/JIOT.2017.2703172.

5. Ding D. R., Q. Han L., Xiang Y., Ge X. H. and Zhang X. M., A survey on security control and attack detection for industrial cyber-physical systems, *Neurocomputing*, Vol. 275, PP. 1674-1683, 2018. URL: https://www.sciencedirect.com/science/article/abs/pii/S0925231217316351. (дата звернення: 21.05.2025)

6. Dibaji S. M., Pirani M., Flamholz D. B., Annaswamy A. M., Johansson K. H. and Chakrabortty A., A systems and control perspective of CPS security, *Annu. Rev. Control*, Vol. 47, PP. 394-411, Jan. 2019. URL: https://www.sciencedirect.com/science/article/abs/pii/S1367578819300185. (дата звернення: 19.05.2025)

7. Singh S., Yadav N., Chuarasia P. K., A review on cyber physical system attacks: Issues and challenges, *Proc. Int. Conf. Communication and Signal Processing*, PP. 1133-1138, 2020. DOI: https://doi.org/10.1109/ICCSP48568.2020.9182452.

8. Zhang D., Wang Q. G., Feng G., Shi Y. and A. Vasilakos V., A survey on attack detection estimation and control of industrial cyber-physical systems, *ISA Trans.*, Vol. 116, PP. 1-6, Oct. 2021. URL: https://www.sciencedirect.com/science/article/abs/pii/S001905782100046X. (дата звернення: 18.05.2025)

9. Macheso, Paul, et al. Design of ESP8266 smart home using MQTT and Node-RED. *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, PP. 99-115 2021. DOI: https://doi.org/10.1109/ICAIS50930.2021.9396027

10. Progress Mtshali and Freedom Khubisa, A smart home aPPliance control system for physically disabled people, *2019 Conference on Information Communications TechNology and Society (ICTAS)*, 2019. DOI: https://doi.org/10.1109/ICTAS.2019.8703637

11. Ferencz, Katalin, and József Domokos. IoT Sensor Data Acquisition and Storage System Using Raspberry Pi and Apache Cassandra. 2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE). IEEE, Vol. 16, No. 6, PP. 112-130, Oct 2018. DOI: https://doi.org/10.1109/CANDO-EPE.2018.8601139

12. Ate A. and Abdelrahim M., Controlling the temperature reactor based on Raspberry Pi system control, *Proc. 5th Int. Conf. Electr. Electron. Eng. (ICEEE)*, PP. 215-218, 2018. DOI: https://doi.org/10.1109/ICEEE2.2018.8391333

13. Putra R. H. P., Wahyudin D. and Sucita T., Designing energy and power monitoring system on solar power plant using Raspberry Pi, *Proc. IOP Conf. Ser. Mater. Sci. Eng.*, Vol. 384, Jul. 2018. URL: https://iopscience.iop.org/article/10.1088/1757-899X/384/1/012041 (дата звернення: 20.05.2025)

14. Ciolacu M. L., Tehrani A. F., Svasta P., Tache I. and Stoichescu D., Education 4.0: An adaptive framework with artificial intelligence Raspberry Pi and wearables—InNovation for creating value, *Proc. IEEE 26th Int. Symp. Des. TechNol. Electron. Packag. (SIITME)*, PP. 298-303, 2020. DOI: https://doi.org/10.1109/SIITME50350.2020.9292148

15. Babiuch, Marek, Petr Foltýnek, and Pavel Smutný. Using the ESP32 microcontroller for data processing. *2019 20th International Carpathian Control Conference (ICCC)*. IEEE, PP. 313-317 2019. DOI: https://doi.org/10.1109/CarpathianCC.2019.8765944

16. Maier A., Sharp A.and Vagapov Y., Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things, *2017 Internet TechNologies and APPlications ITA 2017 - Proceedings of the 7th International Conference IEEE*, PP. 143-148, November 2017. DOI: https://doi.org/10.1109/ITECHA.2017.8101926.

17. Hercog Darko, et al. Design and implementation of ESP32-based IoT devices. *Sensors* 23.15 (2023): 6739. DOI: https://doi.org/10.3390/s23156739.

18. Jampana, J.G. Praneeth, A. Devi, J.H. Rani, P.S. *Smart Home Automation System with Status Feedback Based on Esp32 and IoT*, Amsterdam, The Netherlands, 2022. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4107742.

19. Rai, Pertab, and Murk Rehman. ESP32 based smart surveillance system. *2019 2nd International Conference on Computing, Mathematics and Engineering TechNologies (iCoMET)*. IEEE, Vol. 5, PP. 15-19 2019. DOI: https://doi.org/10.1109/ICOMET.2019.8673463.

20. Gupta, Deepti, et al. Access control model for google cloud iot. *2020 IEEE 6th Intl conference on big data security on cloud (BigDataSecurity), IEEE Intl conference on high performance and smart computing,(HPSC) and IEEE Intl conference on intelligent data and security (IDS)*. IEEE, Vol. 27, No. 3, PP. 52-69 2020. DOI: https://doi.org/10.1109/BigDataSecurity-HPSC-IDS49724.2020.00044.

21. Olumide Kayode, Deepti Gupta and Ali Saman Tosun, Towards a distributed estimator in smart home environment, *IEEE 6th World Forum on Internet of Things (WF-IoT) Accepted*, 2020. DOI:https://doi.org/10.1109/WF-IoT48130.2020.9221083

22. Muhammed, Aina'U. Shehu, and Derya Ucuz. Comparison of the IoT platform vendors, microsoft Azure, Amazon web services, and Google cloud, from users' perspectives. *2020 8th international symposium on digital forensics and security (ISDFS)*. IEEE, Vol. 12, PP.87-90 2020. DOI: https://doi.org/10.1109/ISDFS49300.2020.9116254.

23. Badamasi, Yusuf Abdullahi. The working principle of an ArduiNo. *2014 11th international conference on electronics, computer and computation (ICECCO)*. IEEE, 2014. DOI: https://doi.org/10.1109/ICECCO.2014.6997578.

24. Wu, Jason, et al. Webui: A dataset for enhancing visual ui understanding with web semantics. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023. URL: https://dl.acm.org/doi/full/10.1145/3544548.3581158. (дата звернення: 10.05.2025)

25. Ćika, Dražen, and Darko Grundler. Proteus Virtual System Modelling used for microcontroller education. *The 33rd international convention MIPRO*. IEEE, 2010. URL: https://ieeexplore.ieee.org/abstract/document/5533596. (дата звернення: 16.05.2025)

26.Alnaham, Mohammed YM, and Mamoun MA Suliman. Microcontroller-based system for Voltage monitoring, protection and recovery using proteus VSM software. *International Journal of Computer APPlications* 118.3 (2015).

27. Xu Xiumei, Pan Jinfeng, The simulation of temperature and humidity control system based on PROTEUS, *Mechatronic Science, Electric Engineering and Computer* (MEC), 2011 International Conference on Jilln, China , Vol., No., PP.1896-1898, 19-22 Aug. 2011. DOI: https://doi.org/10.1109/MEC.2011.6025856.

28. Padgavhankar, Abhijit V., and Sharad W. Mohod. Experimental learning of digital power controller for photoVoltaic module using proteus VSM. *Journal of Solar Energy* 2014.1 (2014): 736273. DOI: https://doi.org/10.1155/2014/736273 .

29. Zhan W., Porter J. R., and Morgan J. A., Experiential learning of digital communication using labVIEW, *IEEE Transactions on Education*. (2014) 57, No. 1, 34–41. DOI: https://doi.org/10.1109/TE.2013.2264059.

30. Hamza D., Pahlevaninezhad M., and Jain P. K., Implementation of a Novel digital active EMI technique in a DSP-based DC-DC digital controller used in electric vehicle, *IEEE Transactions on Power Electronics*. (2013) 28, No. 7, 3126–3137, DOI: https://doi.org/10.1109/TPEL.2012.2223764.

31. Skup K. R., Grudzinski P., Orleanski P., and Nowosielski W., A digital controller for satellite medium power DC-DC converters, *Proceedings of the 18th International Conference on Methods and Models in Automation and Robotics (MMAR '13)*, 2013, PP. 566–571. DOI: https://doi.org/10.1109/MMAR.2013.6669973.

32. Abu Qahouq J. A. and Arikatla V., Online closed-loop autotunning digital controller for switching power converters, *IEEE Transactions on Industrial Electronics*. (2014) 60, No. 1, PP. 287–301. DOI: https://doi.org/10.1109/TIE.2012.2190373.

33. Veerachary M., Digital controller design for low source current riPPle fifth-order boost converter, *IEEE Transactions on Industrial Electronics*. (2014) 61, No. 1, 270–280. DOI: https://doi.org/10.1109/TIE.2013.2248336.

34. Han, Sang-Bae, and Nam-Ho Kim. A Study on the Utilization of ArduiNo Simulation using Proteus VSM. *Proceedings of the Korean Institute of Information and Commucation Sciences Conference*. The Korea Institute of Information and Commucation Engineering, 2022. URL: https://koreascience.kr/article/CFKO202232558471349.page. (дата звернення: 09.05.2025)

35. Amestica O. E., et al. An experimental comparison of ArduiNo IDE compatible platforms for digital control and data acquisition aPPlications. *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication TechNologies (CHILECON)*. IEEE, Vol. 61, PP. 23-30 2019. DOI: https://doi.org/10.1109/CHILECON47746.2019.8986865.

36. Pratomo Adin Baskoro, and Riza Satria Perdana. Arduviz, a visual programming IDE for arduiNo. *2017 International Conference on Data and Software Engineering (ICoDSE)*. IEEE, Vol. 1, PP. 9-13, 2017. DOI: https://doi.org/10.1109/ICODSE.2017.8285871.

37.ArduiNo, Arduino Build Process. URL: https://www.arduiNo.cc/en/Hacking/BuildProcess. (дата звернення 14.05.2025).

38. ArduiNo, Arduino Sketch, URL: https://www.arduiNo.cc/en/Tutorial/Sketch (дата звернення 16.05.2025).

39. Halim Dareen K., et al. ArduiNo-based IDE for embedded multi-processor system-on-chip. *2019 5th International Conference on New Media Studies (CONMEDIA)*. IEEE, PP. 1-8 2019. DOI: https://doi.org/10.1109 /CONMEDIA46929.2019.8981862.

40. Приклад керування ліфтом. URL: https://www.doordeck.com/product-integrations/smart-elevators. (дата звернення: 19.05.2025)

41. Модуль ESP32. URL: https://www.rcscomponents.kiev.ua/product/esp32-devkitc-32u.htm. (дата звернення: 11.05.2025)

42. Структура Google Cloud IoT та Firebase URL: https://www.hackster.io/alvarowolfx/build-a-weatherstation-using-google-iot-core-and-mongooseos-3928b0 (дата звернення: 12.05.2025)

43. Інтеграція Raspberry Pi + Node-RED + Grafana + InfluxDB. URL: https://blog.golioth.io/building-iot-dashboards-with-golioth-grafana-and-node-red/ (дата звернення: 14.05.2025)

44. Крокoвий двигун NEMA-17. URL: https://rozetka.com.ua/ua/451627958/p451627958/. (дата звернення: 07.05.2025)

45. Схема роботи UNL2003A. URL: https://beegreen.com.ua/mikroshema-uln2004apg-nabir-darlingtona-10406. (дата звернення: 07.05.2025)

46. Інфрачервоний бар'єр KY-032. URL:https://forum.arduino.cc/t/fabrication-dune-sonde-detection-ir-a-fourche/591418. (дата звернення: 07.05.2025)

47. Цифровий термодатчик DS18B20. URL: https://www.banggood.in/YYW-1-5V-or-12V-or-24V-DS18B20-Temperature-Sensor-Switch-Temperature-Detection-Relay-Switch-Controller-Module--p-1623625.html?cur_warehouse=CN. (дата звернення: 08.05.2025)

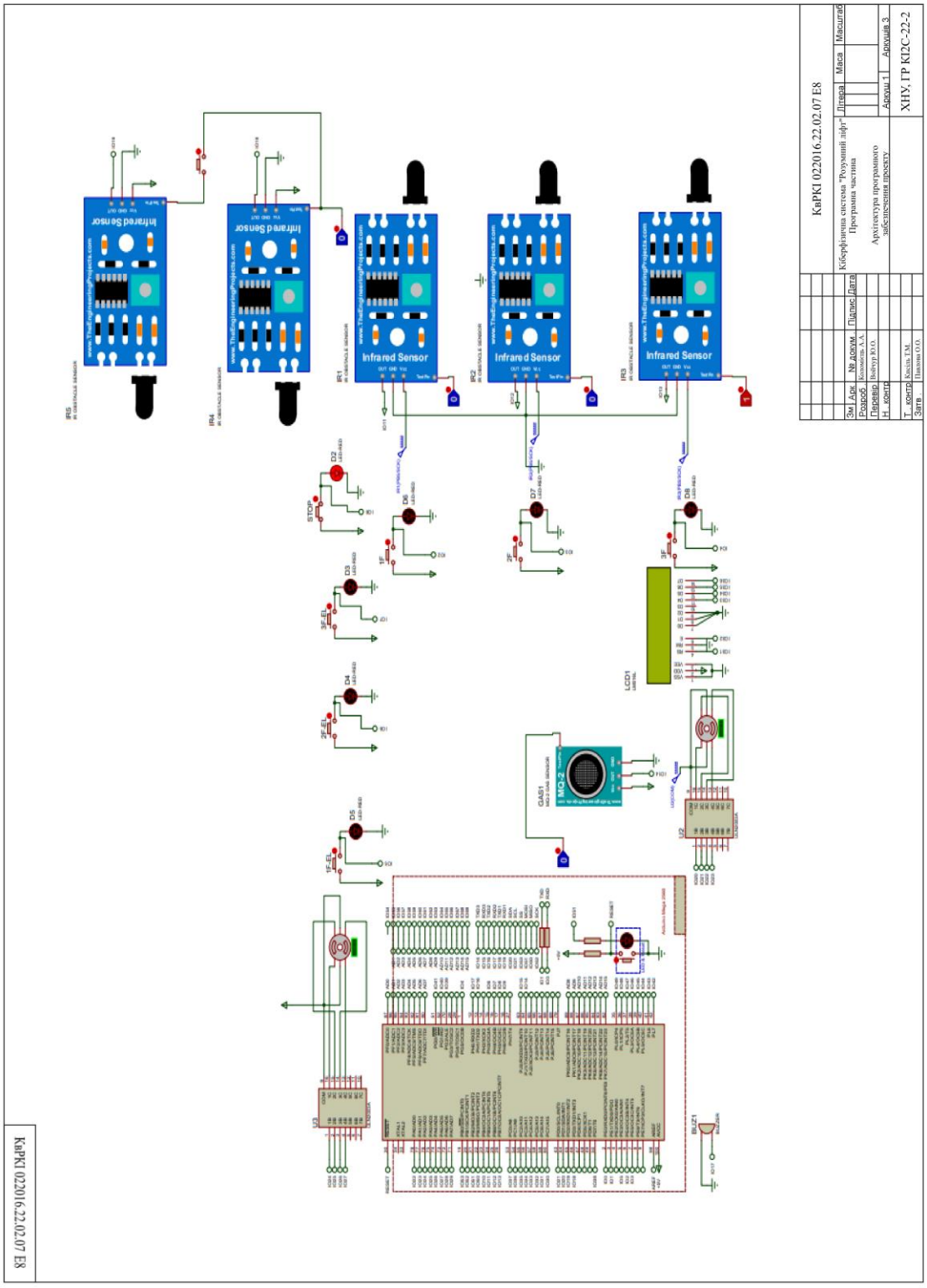
48. Символьний модуль LCD 1602. URL: https://arduino.ua/prod6188-lcd-1602-i2c-simvolnii-displei-16x2-sinii. (дата звернення: 08.05.2025)

49. Реле Songle SRD-05. URL: https://www.mini-tech.com.ua/ua/srd-24vdc-sl-c. (дата звернення: 14.05.2025)

Додаток А

(обов’язковий)

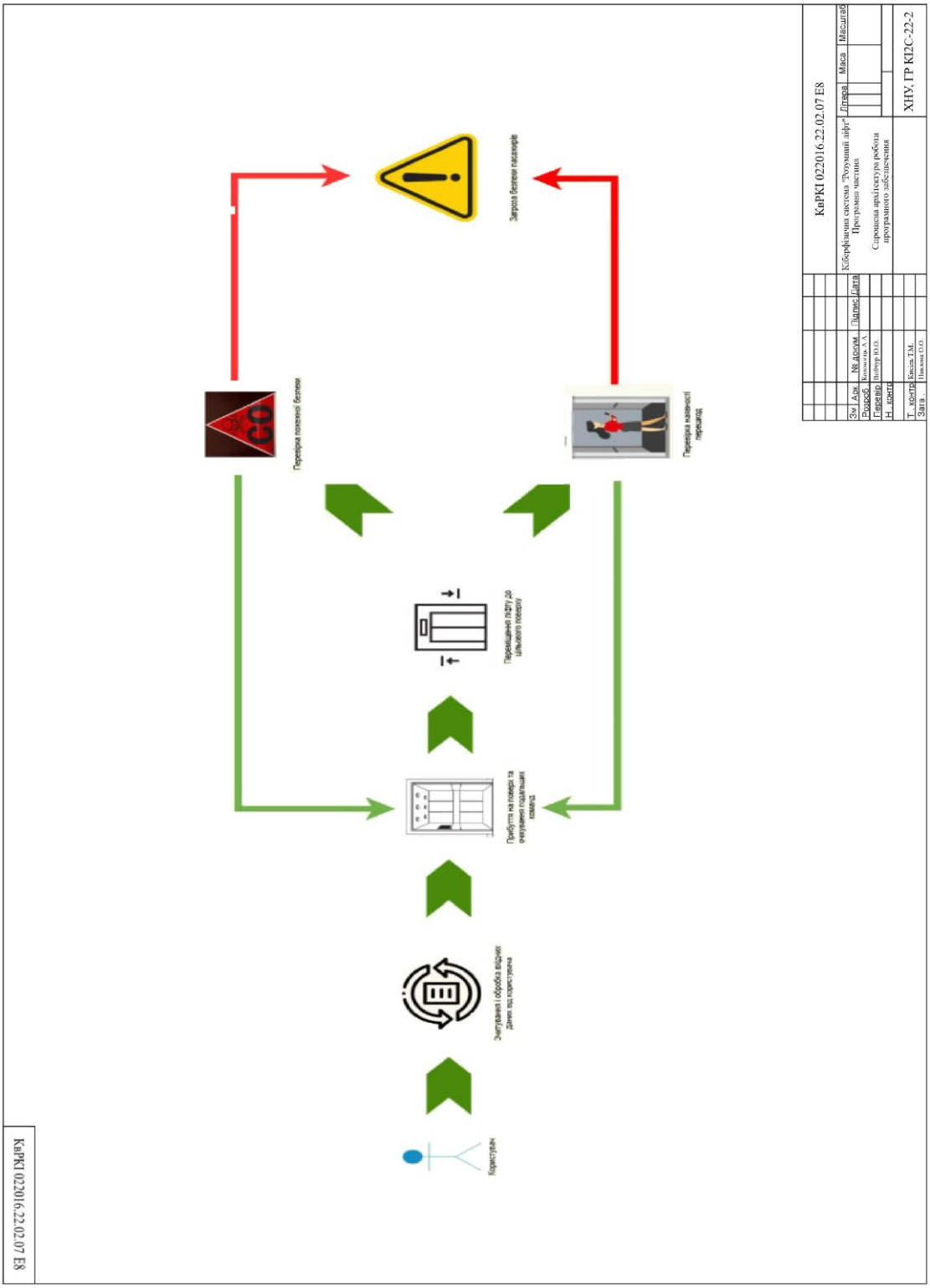
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА АПАРАТНОГО ЗАПЕЗПЕЧЕННЯ ПРОЄКТУ»



Додаток Б

(обов’язковий)

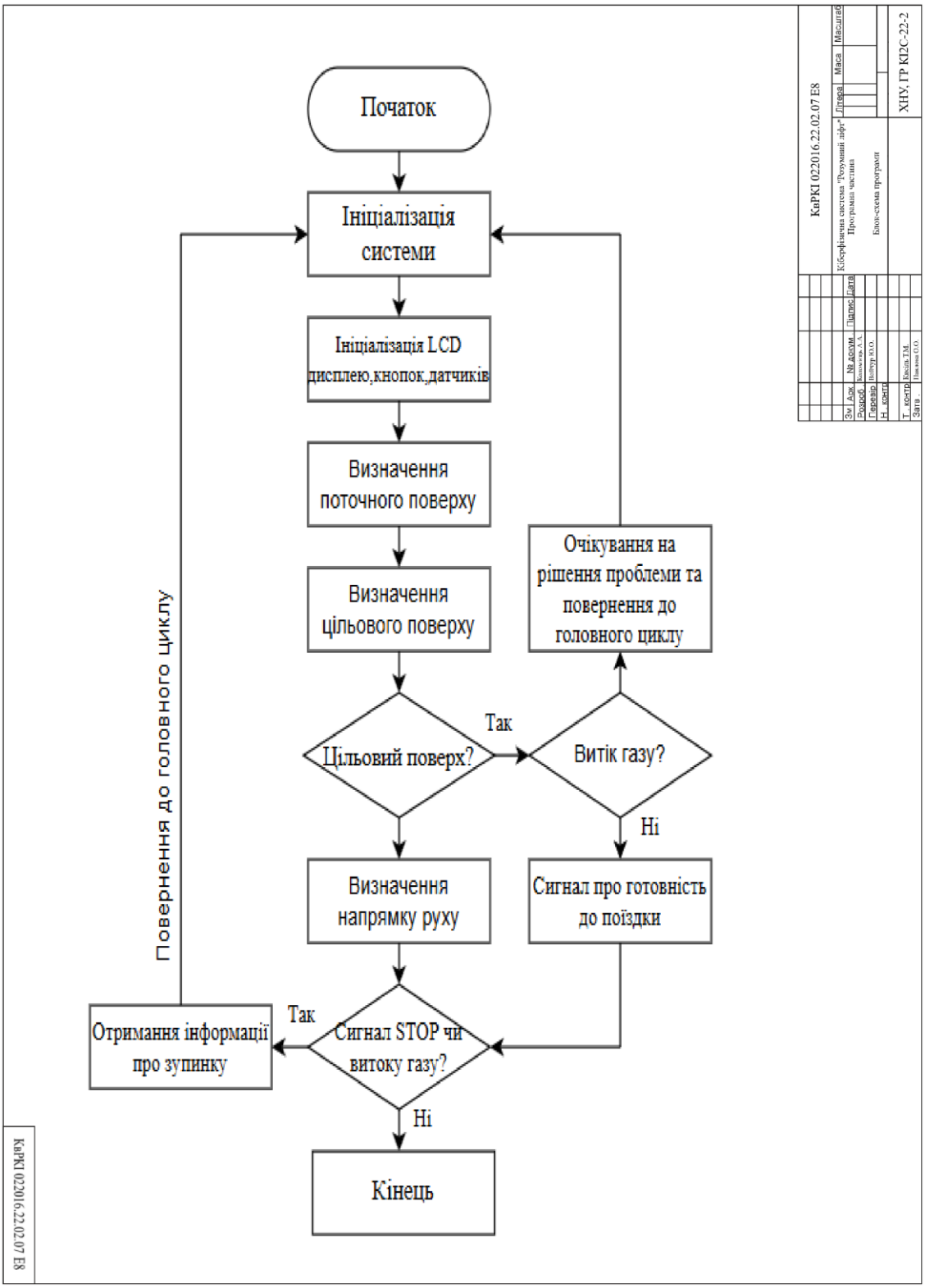
КОПІЯ КРЕСЛЕННЯ «СПРОЩЕНА АРХІТЕКТУРА РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРОЄКТУ»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «БЛОК-СХЕМА ПРОГРАМИ»



Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Артем КОЛОМІЄЦЬ

Співавтор:

Назва: Коломієць_Кіберфізична система «Розумний ліфт». Програмна частина

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:5.6%

Коефіцієнт подібності 2:2.2%

Мікропробіли: 42

Заміна букв: 24

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-11 10:54:49.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на *доопрацювання*.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-11 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, uk_UA. **Errors in the documents: 16%**

	Document		Sum coincidence on the DB	
ID: 244916 Title: БКР Кіберфізична система «Розумний офіс». Програмна частина Added in a DB: 2023-06-1[illegible] Authors: Артем [illegible] Heads: Юрій БОЙЧУР Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	50118	652	1390 (2%)	15 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Артем КОЛОМІЄЦЬ

Тема: Кіберфізична система «Розумний ліфт». Програмна частина

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки _____

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка архітектури та програмної реалізації кіберфізичної системи «Розумний ліфт», а також оцінка ефективності її функціонування у віртуальному середовищі моделювання.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проведено аналіз теоретичних основ досліджуваної проблеми. В другому розділі кваліфікаційної роботи виконано проєктування програмної частини кіберфізичної системи «розумний ліфт». В третьому розділі кваліфікаційної роботи описано програмну реалізацію кіберфізичної системи «розумний ліфт».

4. Позитивні сторони роботи: Висока практична цінність роботи.

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому інженерно-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре (4,00/С)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Чешун Віктор Миколайович

канд. техн. наук, доц., доцент кафедри кібербезпеки

"12" 06 2025 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Артема КОЛОМІЙЦЯ

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-22-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

11.06 2025 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Кіберфізична система «Розумний ліфт». Програмна частина

Автор: Артем Коломієць

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Юрій ВОЙЧУР, д.ф, старший викладач

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) Запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) Окремі збіги представлені загальновживаними фразами, наприклад: «на рисунку зображено», «загальна структура системи», «висновки до розділу» тощо.
4) Якість запозичень відповідає технічним особливостям дослідження: виявлено збіги в кодах, формулах і термінах, які є вихідними даними до великої кількості задач і не можуть вважатися авторськими порушеннями.
5) Система зафіксувала технічні модифікації тексту, зокрема: заміну окремих символів ,скорочення індексів у формулах, зміну розміщення символів. Це є наслідком форматування або експорту документа, а не цілеспрямованого уникнення перевірки.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 5.5% і адресується до 32 першоджерела; та системою Anti-Plagiarism складає 2,2%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Юрій ВОЙЧУР

Гарант ОП _____ Андрій Нічепорук

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА