

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІРТУАЛЬНОЇ
МОДЕЛІ ЛІФТА
SOFTWARE DEVELOPMENT OF A VIRTUAL ELEVATOR MODEL

Виконав(ла): студент(ка) 4 курсу, групи КНТ-112

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БАБАНЦЕВ Д.А.

(ПРІЗВИЩЕ та ініціали)

Керівник ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

Рецензент ЗЕЛІК О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

- БАБАНЦЕВА Дмитра Артемовича
(ПРІЗВИЩЕ, ім'я, по батькові)
1. Тема проєкту (роботи) Розробка програмного забезпечення віртуальної моделі ліфта. Software Development of a Virtual Elevator Model
 керівник проєкту (роботи) д.т.н., професор ПОЛЯКОВ Михайло Олексійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
 затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
 2. Строк подання студентом проєкту (роботи) 01 червня 2026 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
 4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Аналіз середовищ програмування. 3. Організація зв'язку між компонентами системи. 4. Проектування системи. 5. Реалізація та тестування програмно-апаратного комплексу.
 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації
 6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання ви- дав	прийняв виконане завдання
1-4 Основна частина	ПОЛЯКОВ М.О., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «20» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк викона- ння етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Вступ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3, 4
5	Розробка програми.	3-4 тижні	Розділ 5
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Дмитро БАБАНЦЕВ
(підпис) (ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Михайло ПОЛЯКОВ
(підпис) (ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 137 с., 13 табл., 19 рис., 2 дод., 17 джерел.

ЛІФТ, ARDUINO MEGA, SIMULIDE, PYTHON, SCADA-ІНТЕРФЕЙС, ВІРТУАЛЬНА МОДЕЛЬ, АНІМАЦІЯ, UART, IoT.

Об'єкт дослідження – Процес інженерії програмного забезпечення вбудованих систем.

Предмет дослідження – підходи й інструментарій, які використовуються для формування цифрових двійників (Digital Twin) пристроїв IoT, що дає можливість пришвидшити процес налагодження логіки управління на прикладі програмно-апаратного комплексу «Система керування ліфтом»

Мета роботи – скорочення часу налагодження програмного забезпечення для систем Інтернету речей (IoT) шляхом створення програмно-апаратного комплексу з віртуальною моделлю ліфта (Digital Twin), що дає змогу проводити тестування керуючих алгоритмів і інтерфейсу взаємодії з користувачем безпосередньо, оминаючи

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування (C++ для Arduino, Python), середовище симуляції схем SimulIDE, бібліотека для візуалізації Tkinter, віртуальний COM-порт (com0com), протокол обміну даними UART, персональний комп'ютер з операційною системою Microsoft Windows 10/11.

Висновки. Розроблено застосунок для візуалізації роботи ліфта від плати Arduino в симуляторі SimulIDE, який демонструє принципи взаємодії апаратного забезпечення та програмного забезпечення візуалізації. Система забезпечує коректну обробку викликів, візуалізацію стану об'єкта в реальному часі.

Галузь використання – розроблений програмний комплекс може бути використаний для демонстрації принципів взаємодії програмних компонентів з віртуальними (графічними) моделями використовуваного у реальному житті обладнання.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 137 pages, 13 tables, 19 figures, 2 appendices, 17 references.

Elevator, ARDUINO MEGA, SIMULIDE, PYTHON, SCADA INTERFACE, VIRTUAL MODEL, ANIMATION, UART, IoT.

Research Object – The software engineering process for embedded systems.

Research subject – approaches and tools used to create digital twins (Digital Twin) of IoT devices, which makes it possible to accelerate the process of debugging control logic using the example of the “Elevator Control System” hardware-software complex

The objective of this work is to reduce software debugging time for Internet of Things (IoT) systems by creating a hardware-software complex with a virtual elevator model (Digital Twin), which allows for direct testing of control algorithms and the user interface, bypassing

Materials, methods, and technical tools: structured and object-oriented programming (C++ for Arduino, Python), the SimulIDE circuit simulation environment, the Tkinter visualization library, virtual COM port (com0com), UART data exchange protocol, personal computer running Microsoft Windows 10/11.

Conclusions. An application has been developed to visualize the operation of an elevator controlled by an Arduino board in the SimulIDE simulator, demonstrating the principles of hardware-software interaction in visualization. The system ensures correct call processing and real-time visualization of the object's status.

Field of application – the developed software package can be used to demonstrate the principles of interaction between software components and virtual (graphical) models of equipment used in real life.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	11
ВСТУП.....	12
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	14
Постановка проблеми.....	14
1.1 Особливості функціонування ліфта як об'єкта автоматизації...14	14
1.2.1 Обґрунтування вибору ліфта для візуальної моделі.....	16
1.3 Архітектура SCADA-систем та їх застосування.....	17
1.3.1 Огляд SCADA-систем.....	18
1.3.2 Приклади промислових SCADA-систем.....	18
1.4 Інтернет речей (IoT) та концепція цифрового двійника (Digital Twin).....	20
1.5 Обґрунтування вибору мікроконтролера Arduino Mega 2560.....	21
1.5.1 Характеристики Arduino Mega 2560 та аналоги.....	22
2 АНАЛІЗ СЕРЕДОВИЩ ПРОГРАМУВАННЯ.....	25
2.1 Симулятори електронних схем та мікроконтролерів.....	25
2.1.1 Обґрунтування вибору Simulide.....	27
2.2 Мова для SCADA-клієнта.....	28
2.2.1 C# (Windows Forms).....	28
2.2.2 JavaScript (Electron + Web Serial API).....	28
2.2.3 Python (Tkinter + pySerial).....	29
2.3 Висновки до розділу 2.....	29
3 ОРГАНІЗАЦІЯ ЗВ'ЯЗКУ МІЖ КОМПОНЕНТАМИ СИСТЕМИ.....	31

3.1 COM-порт	31
3.2 Алгоритм роботи UART протоколу	32
3.3.1 Технічні особливості налаштування віртуальних портів.....	34
3.4 Висновки до розділу 3	35
4 ПРОЄКТУВАННЯ СИСТЕМИ.....	36
4.1 Діаграма прецедентів та опис акторів.....	36
4.2 Структурна схема системи.....	37
4.2.1 Нижній рівень (SimulIDE).....	38
4.2.2 Верхній рівень (Python/Tkinter).....	38
4.2.3 Транспортний рівень (com0com).....	39
4.3 Опис апаратної частини (схема в SimulIDE).....	41
4.4 алгоритм роботи контролера.....	43
4.5 Архітектура Python-застосунку	48
4.5.1 Модуль зв'язку (Serial Manager).....	49
4.5.2 Модуль обробки повідомлень (Protocol Parser)	50
4.5.3 Модуль черги (Queue Manager).....	51
4.5.4 Модуль візуалізації та анімації.....	51
4.6 Optimistic UI.....	52
4.7 Висновки до розділу 4	54
5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ.....	56
5.1 Налаштування середовища розробки та розгортання.....	56
5.2 Програмна реалізація контролера.....	57
5.2.1 Оголошення пінів та констант	58
5.2.2 Черга викликів.....	58

5.2.3 Керування рухом.....	59
5.2.4 Керування дверима.....	59
5.2.5 Основний цикл.....	60
5.2.6 Допоміжні функції.....	61
5.3 Програмна реалізація SCADA-клієнта.....	61
5.3.1 Підключення до COM-порту та читання даних.....	61
5.3.2 Обробка вхідних команд.....	62
5.3.3 Відправка команд із захистом від брязкоту контактів.....	63
5.3.4 Анімація руху кабіни.....	63
5.3.5 Анімація дверей та Optimistic UI.....	64
5.3.6 Клієнтська черга викликів.....	64
5.4 Тестування протоколу обміну даними.....	65
5.4.1 Висновки за результатами тестування.....	68
5.5 Тестування черги викликів.....	69
5.5.1 Аналіз результатів.....	71
5.5.2 Виявлені особливості.....	71
5.6 Тестування інтерфейсу користувача.....	72
5.6.1 Плавність анімації руху кабіни.....	72
5.6.2 Миттєвість реакції на керування дверима (Optimistic UI).....	73
5.6.3 Коректність відображення стану ліфта.....	74
5.6.4 Висновки за результатами тестування інтерфейсу.....	75
5.7 Висновки до розділу 5.....	76
ВИСНОВКИ.....	78
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	81
ДОДАТОК А.....	83

Текст програми для плати Arduino.....	83
ДОДАТОК Б.....	130
Слайди презентації.....	130

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface (прикладний програмний інтерфейс);
COM	– Communication Port (послідовний порт);
FIFO	– First In, First Out (перший прийшов — перший пішов);
HEX	– Hexadecimal (шістнадцятковий формат файлу прошивки);
HMI	– Human-Machine Interface (людино-машинний інтерфейс);
IDE	– Integrated Development Environment (інтегроване середовище розробки);
IoT	– Internet of Things (Інтернет речей);
SCADA	– Supervisory Control and Data Acquisition (диспетчерське керування та збір даних);
PLC	– Programmable Logic Controller (програмований логічний контролер);
ПЗ	– програмне забезпечення.

ВСТУП

Завдяки технічному прогресу, кількість пристроїв Інтернету речей у світі зростає на десятки відсотків щороку. Кожен такий пристрій — це мікроконтролер, для якого необхідно створити програмне забезпечення, здійснити його тестування, виправити виявлені недоліки, провести повторне тестування, а також ідентифікувати потенційні загрози та їх усунення і щоразу для цього потрібен безпосередній фізичний доступ до пристрою який тестується. Це якби веб-розробник не міг перевірити свій код без окремого сервера під столом, така ситуація є абсурдною, але у світі вбудованих систем вона досі є нормою.

Промислові навчальні стенди коштують дорого, від п'ятдесяти до двохсот тисяч гривень за одиницю. Припустимо ситуацію, Лабораторія технічного університету має два-три такі стенди на всю кафедру. Студент чекає черги, щоб перевірити свій код. Один цикл налагодження має наступний алгоритм — змінив рядок, скомпілював, прошив, перевірів, все це займає не менше 10-15 хвилин, за двох годинне заняття виходить чотири-п'ять спроб. До того ж помилка в коді може призвести до пошкодження обладнання, а ремонт промислового контролера коштує майже як новий.

Цифровий двійник (Digital Twin) [1] — це спосіб обійти ці обмеження. Ідея проста, замість фізичного стенду використовується його програмна модель, яка так само реагує на команди контролера, так само змінює свій стан, але не ламається, не потребує ремонту й доступна 24 години на добу. На прикладі навчального процесу це означає, що кожен студент може тестувати свій код без черги, без ризику й без витрат на обладнання.

Метою цієї дипломної роботи було скорочення часу налагодження програмного забезпечення для систем Інтернету речей (IoT) [2] на прикладі віртуальної моделі ліфта. Для реалізації було поставлено наступні завдання:

- розробити апаратну частину на базі віртуального контролера Arduino Mega 2560 [3] в середовищі SimulIDE, що включатиме кнопки виклику та дверей, світлодіодну індикацію напрямку руху та індикатор поверху;
- створити застосунок на Python, для виконання ролі SCADA-інтерфейсу, який повинен відображати плавну анімацію руху кабіни, відкриття/закриття дверей, індикацію руху та приймати команди оператора;
- двосторонній обмін даними між контролером та інтерфейсом через з обробкою команд та алгоритму черги для обслуговування попутних викликів за системою FIFO [4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Постановка проблеми

Розробка програмного забезпечення для мікроконтролерів завжди була справою невдячною в плані зворотного зв'язку. Веб-розробник змінив рядок у коді і одразу побачив результат у браузері. Розробник під Android просто запустив емулятор. А от якщо ми маємо справу із фізичною моделлю, яка не знаходиться дома і часто дорогою, то для того щоб перевірити, чи правильно ліфт обробляє попутні виклики, потрібен сам ліфт, або хоча б його стенд. А це десятки тисяч гривень, окреме приміщення, і жодної можливості працювати віддалено. Ідея цього проєкту з'явилася саме з такого запиту, як зробити такий інструмент, який дозволяє тестувати алгоритми керування без фізичного пристрою. Це не заміна справжньому стенду, а доповнення до нього, можна сказати етап, на якому можливо швидко побачити логічні помилки, перевірити різноманітні випадки, експериментувати з різними сценаріями, а вже потім, коли код обкатаний на віртуальній моделі, заливати його в реальний контролер і бути впевненим, що код не наробить критичних помилок для стенду.

1.1 Особливості функціонування ліфта як об'єкта автоматизації

Ліфт — це не просто «кабіна, яка їздить вгору-вниз», з точки зору системи керування це досить складний об'єкт із кількома незалежними підсистемами, кожна з яких може впливати на інші.

Перша підсистема — привод. Електродвигун через редуктор або напряму (в безредукторних ліфтах) обертає шків, на якому висять троси. З одного боку тросів — кабіна, з іншого — противага. Противага потрібна, щоб двигун не тягав повну вагу кабіни з пасажирями або вантажем, а лише різницю між вагою кабіни та противаги, це зменшує навантаження на двигун і економить електроенергію. З погляду коду — нічого складного: подав сигнал «вгору» —

двигун крутиться в один бік, подав «вниз» — в інший, але є нюанс, між подачею сигналу й початком руху проходить певний час, і це потрібно враховувати в алгоритмі.

Друга підсистема — двері. Вони можуть бути розпашними або телескопічними, але в будь-якому випадку їхній рух — це окремий цикл, а саме: відкрити → зачекати → закрити → переконатися, що закрито. Саме з дверима пов'язана більшість аварійних ситуацій у реальних ліфтах: хтось не встиг зайти, хтось затиснув руку, датчик не спрацював. У промислових системах для цього є окремий контролер дверей. У моєму проєкті все простіше, але логіка та сама, відкриття, пауза, закриття, контроль стану.

Третя підсистема — позиціювання. Ліфт повинен точно зупинятися на рівні підлоги. Для цього на кожному поверсі присутні датчики — або магнітні геркони, або оптичні пари, або індуктивні датчики. У момент, коли кабіна проходить повз датчик, контролер фіксує: «ліфт на поверсі». Між поверхами, як правило, є ще датчики уповільнення — щоб ліфт не гальмував ривком, а плавно знижував швидкість перед зупинкою. У моїй реалізації ця логіка спрощена: замість фізичних датчиків — таймер, який імітує час руху між поверхами.

Четверта підсистема — алгоритм обробки викликів. Це найцікавіше з погляду програмування, тому що ліфт повинен приймати виклики з поверхів (зовнішні кнопки) та з кабіни (внутрішня панель) і вирішувати, у якому порядку їх обслуговувати. Найпростіший алгоритм — «перший натиснув — перший поїхав», але він неефективний. У реальних ліфтах використовується збір попутних викликів: якщо ліфт їде з 1-го на 5-й поверх, він повинен зупинятися на 2-му, 3-му або 4-му, якщо там є пасажир, які також натиснули кнопку виклику. Саме цей алгоритм я і реалізував у своєму проєкті.

Окремо варто сказати про безпеку, у реальному ліфті є:

- ~ датчик перевантаження (якщо вага в середині ліфта перевищує норму, то двері не зачиняються);

- ~ датчик швидкості (якщо ліфт розганяється швидше ніж треба, то спрацьовує система аварійного гальмування);
- ~ “Стоп Кран” (екстрена зупинка в будь-який момент);
- ~ режим пожежної безпеки (при сигналі «Пожежа» ліфт їде на 1-й поверх і блокується).

У моєму проєкті ці аспекти безпеки не реалізовані. Це навмисне спрощення, тому що мета роботи не в тому, щоб зробити промисловий контролер ліфта, а в тому, щоб створити інструмент для швидкого тестування алгоритмів керування. Але архітектура системи така, що додати ці функції можна без переробки всього коду, буде достатньо додати нові умови в основний цикл контролера.

1.2.1 Обґрунтування вибору ліфта для візуальної моделі

Ліфт як об'єкт для створення візуальної моделі було обрано, Тому що ліфт це об'єкт із чіткою та зрозумілою логікою, але водночас із достатньою кількістю нюансів, щоб було можливо продемонструвати серйозний підхід до програмування. В ліфті є дискретні стани, поверх, напрямок, статус дверей, переходи між станами відкриття дверей, очікування, закриття, рух, черга подій виклики з різних поверхів, які можуть надходити одночасно. Усе це робить ліфт майже ідеальним об'єктом для демонстрації принципів проєктування вбудованих систем.

1.3 Архітектура SCADA-систем та їх застосування

Абревіатура SCADA розшифровується як Supervisory Control and Data Acquisition [5], або ж українською — диспетчерське керування та збір даних. Якщо зовсім просто, то це програма, яка показує оператору, що відбувається з обладнанням і дозволяє ним керувати. Надалі простий приклад. На екрані — мнемосхема: труби, насоси, засувки, датчики температури. Щось блимає зеленим (працює), щось червоним (аварія), десь змінюється цифра (тиск у трубопроводі), а задача Оператора сидіти перед цим екраном і приймати рішення.

З ліфтом це пов'язано наступним чином, у великих будівлях ліфти об'єднані в групи, і за ними стежить диспетчер. Він бачить, де яка кабіна, чи немає аварій, чи не застряг хтось між поверхами, доречі в Україні популярний такий комплекс, як «Промінь» [6]. У сучасних системах диспетчер може віддалено керувати ліфтом, наприклад відправити на потрібний поверх, заблокувати двері, перевести в режим пожежної безпеки. Усе це — функції SCADA.

У моєму проєкті SCADA-клієнт — це Python-застосунок із графічним інтерфейсом на Tkinter [7]. Він показує шахту ліфта з кабіною, дверима, індикаторами. Оператор (у моєму випадку — розробник, який тестує алгоритм) може натиснути кнопку «2 поверх» — і програма відправить команду FLOOR:2 у віртуальний контролер (Плату Arduino). Контролер обробить команду, запустить рух ліфта, і почне надсилати зворотні повідомлення: MOVING:1->2, потім ARRIVED:2, потім DOOR:OPEN. Python-застосунок читає ці повідомлення й анімує відповідні дії на екрані.

1.3.1 Огляд SCADA-систем

Типова SCADA-система складається з влаштована наступним чином. Нижній рівень (контролери) — PLC, RTU, мікроконтролери, які зчитують сигнали з датчиків і керують виконавчими механізмами. У моєму проєкті це Arduino Mega в SimulIDE [8]. Верхній рівень (HMI/SCADA) — програма на комп'ютері, яка відображає стан об'єкта у вигляді мнемосхем, графіків, таблиць, а також приймає команди оператора. У моєму проєкті це Python-застосунок із графічним інтерфейсом. Канал зв'язку — мережа, через яку контролер і SCADA обмінюються даними. У промисловості це найчастіше RS-485, Ethernet або бездротові мережі. У моєму проєкті це віртуальний COM-порт [9] (UART через com0com) — спрощена, але функціональна модель промислового каналу. Мій проєкт відтворює цю трирівневу архітектуру в мініатюрі. Нижній рівень — Arduino Mega в SimulIDE з кнопками, світлодіодами та 7-сегментним індикатором. Верхній рівень — Python-застосунок із плавною анімацією, панеллю керування та журналом подій. Канал зв'язку — віртуальний COM-порт. Усе це працює на одному комп'ютері, але логічно розділене так само, як у реальній промисловій системі.

1.3.2 Приклади промислових SCADA-систем

Simatic WinCC [10] — система від Siemens, один із лідерів європейського ринку. Підтримує роботу з контролерами Simatic S7, має вбудовані засоби візуалізації, архівування даних, обробки тривоги. Використовується в автомобільній промисловості, енергетиці, харчовому виробництві, фармацевтиці. Остання версія WinCC Unified побудована на веб-технологіях (HTML5, JavaScript, SVG), що дозволяє відкривати інтерфейс із будь-якого браузера. Вартість ліцензії — від кількох сотень до десятків тисяч доларів залежно від кількості змінних.

TRACE MODE — SCADA-система, яка об'єднує в собі засоби розробки HMI, програмування контролерів (SOFTLOGIC) та управління виробництвом (MES) . Підтримує роботу через GSM-мережі, що дозволяє керувати віддаленими об'єктами без прокладання кабелів . Використовується в енергетиці, нафтогазовій галузі, водопостачанні, будівництві. Має безкоштовну версію для навчальних проєктів, що робить її популярною в українських вишах.

Simple-SCADA — відкритий проєкт на Python, створений для освітніх цілей. Як і мій проєкт, він демонструє базові принципи побудови SCADA-системи таких як збір даних, візуалізацію, відправку команд. Використовує CAN-мережу для зв'язку з контролерами, що наближає його до реальних автомобільних систем.

Може виникнути питання, навіщо робити нову систему, якщо вже є готові для використання. Готові SCADA-системи (WinCC, TRACE MODE) коштують грошей і потребують часу на освоєння. Крім того, вони орієнтовані на роботу з реальними контролерами через промислові протоколи (OPC, Profibus, Modbus), а не з віртуальним COM-портом. Для мого проєкту важливіше було показати сам принцип роботи SCADA, а не використовувати конкретний бренд. Python-застосунок із плавною анімацією та журналом подій виконує ту саму роль, приймає дані від контролера, відображає їх оператору і дозволяє керувати процесом.

Мій проєкт демонструє ключову особливість SCADA [7] розділення контролера та інтерфейсу. У промисловості це потрібно для безпеки: оператор знаходиться в диспетчерській, а контролер — біля обладнання, іноді за кілька кілометрів. У моєму проєкті SimulIDE та Python — це дві окремі програми, з'єднані каналом зв'язку. Завдяки цьому інтерфейс можна запустити навіть без симулятора: він показуватиме анімацію руху, відкриття дверей, зміну поверху — усе, крім реальних даних із контролера. Це демонструє незалежність двох рівнів системи, що є одним із принципів побудови надійних SCADA-систем.

1.4 Інтернет речей (IoT) та концепція цифрового двійника (Digital Twin)

IoT [2] — концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, які мають вбудовані давачі, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між апаратною частиною і програмним забезпеченням в автоматичному режимі, за допомогою використання стандартних протоколів зв'язку. Коли говорять про Інтернет речей, зазвичай згадують розумні колонки, фітнес-браслети та холодильники, які замовляють молоко, але це лише верхівка айсберга. Справжній IoT — це коли тисячі пристроїв обмінюються даними без участі людини. На прикладі ліфта, датчик вібрації на двигуні ліфта фіксує відхилення від норми — і відправляє сигнал диспетчеру ще до того, як двигун вийде з ладу. Контролер освітлення отримує прогноз погоди й вирішує, чи вмикати лампи в холі. Це не зайві технології для розширення бюджету, це реальна економія грошей на обслуговуванні та ремонті.

Digital Twin [1] — це віртуальна копія фізичного пристрою, яка отримує ті самі дані, що й реальний пристрій, і поводить себе так само. Але на відміну від реального пристрою, її можна «зламати» без жодних наслідків. Можна подати неправильний сигнал, подивитися, що станеться, виправити код — і знову протестувати. І так безліч разів, для будь-яких можливих сценаріїв.

У моєму проєкті цифровий двійник це зв'язка SimulIDE + Python. Віртуальний контролер Arduino Mega2560 виконує ту саму прошивку, що була б залита в реальний контролер ліфта. SCADA-клієнт на Python відображає стан ліфта, але все це — на одному комп'ютері, без жодного фізичного пристрою. Це може стати корисним для розробника, який пише прошивку для ліфта, може протестувати алгоритм на цифровому двійнику за 1-2 хвилини, якщо алгоритм працює некоректно — це видно одразу. Наприклад, двері не зачиняються, кабіна їде не туди, індикатор показує неправильний поверх. Розробник виправляє код, перекомпілює прошивку, завантажує в симулятор

— і знову тестує. Коли алгоритм повністю налагоджений на віртуальній моделі, його можна заливати в реальний контролер і бути впевненим, що код не зробить критичної шкоди для реального стенда, або навіть справжнього ліфта.

Технічно обмін даними між цифровим двійником та SCADA-клієнтом реалізовано через протокол UART. Контролер надсилає повідомлення про свій стан (MOVING:1->3, ARRIVED:3, DOOR:OPEN), а SCADA-клієнт «оживляє» їх на екрані. І навпаки: коли SCADA-клієнт надсилає команду (FLOOR:2, DOOR:CLOSE), контролер виконує її так само, як виконав би команду від реальної кнопки.

Але є і обмеження. Мій цифровий двійник не враховує фізичних процесів, які відбуваються в реальному ліфті — інерцію двигуна, вібрацію, зношування механізмів. Мій проєкт — це логічна модель, а не фізична. Для тестування алгоритмів керування цього достатньо. Для тестування, скажімо, системи аварійного гальмування — ні, це вже зовсім інша задача.

1.5 Обґрунтування вибору мікроконтролера Arduino Mega 2560

Arduino [3] — це відкрита апаратно-програмна платформа для швидкого прототипування електронних пристроїв. Вона складається з апаратної частини (плати з мікроконтролером) та програмного середовища (Arduino IDE), яке дозволяє писати, компілювати та завантажувати прошивку в контролер мовою C++. Завдяки простоті використання, великій кількості бібліотек та низькій вартості, Arduino стала де-факто стандартом для навчальних проєктів із автоматизації та робототехніки.

Для даного проєкту було розглянуто три найпоширеніші платформи: Arduino Uno — базова плата на ATmega328P, Arduino Mega 2560 — розширена плата на ATmega2560 та ESP32 — потужний мікроконтролер із вбудованим Wi-Fi та Bluetooth. Кожна з них має свої переваги, але для задачі ке-

рування ліфтом із великою кількістю периферії та необхідністю симуляції в SimulIDE вирішальними виявилися кілька критеріїв.

1.5.1 Характеристики Arduino Mega 2560 та аналоги

Arduino Mega 2560 побудована на мікроконтролері ATmega2560 — одному з найпотужніших представників 8-бітного сімейства AVR. Її ключові характеристики зазначені в таблиці 1.1.

Таблиця 1.1 — Фізичні характеристики плати Arduino Mega 2560

Параметр	Значення
Мікроконтролер	ATmega2560
Розрядність	8 біт
Тактова частота	16 МГц
Flash-пам'ять	256 КБ
Цифрові піни вводу/виводу	54 (з них 15 із ШІМ)
Аналогові входи	16
SRAM	8 КБ
EEPROM	4 КБ
Апаратні UART	4 послідовних порти
Напруга живлення	5 В
Робоча напруга логіки	5 В

Головна перевага Mega перед Uno це велика кількість пінів, якщо бути точним то 54 цифрових порти. Така чисельність дозволяє підключити всю необхідну периферію без використання розширювачів портів або мультиплексорів. У моєму проєкті задіяно 16 пінів (8 кнопок, 2 світлодіоди, 7 сегментів індикатора, лінії TX/RX), і це далеко не межа.

Іншою важливою особливістю 4 апаратних UART є те, що на відміну від Arduino Uno, яка має лише один послідовний порт, який не можливо використовувати, бо він попросту зайнятий для зв'язку з комп'ютером. Arduino Mega дозволяє використовувати додаткові Serial-порти для підключення зовнішніх пристроїв. У цьому проєкті використовується інструмент Serial, який під'єднаний до пінів TX 0 і RX 1 для зв'язку з Python через SimulIDE, але за потреби можна підключити ще один пристрій через Serial1, Serial2 або Serial3. Більш детальне порівняння контролерів в таблиці 1.2.

Таблиця 1.2 — Порівняння плат

Критерій	Arduino Uno	Arduino Mega 2560	ESP32
Мікроконтролер	ATmega328P	ATmega2560	ESP32-D0WDQ6
Тактова частота	16 МГц	16 МГц	240 МГц
Цифрові піни	14	54	34
Аналогові входи	6	16	18
Flash-пам'ять	32 КБ	256 КБ	4 МБ
SRAM	2 КБ	8 КБ	520 КБ
UART (апаратних)	1	4	3
Wi-Fi / Bluetooth	Немає	Немає	Вбудовані
Підтримка SimulIDE	Повна	Повна	Експериментальна

Arduino Uno є найпопулярнішою платою для початківців, але для цього проєкту вона не підходить через обмежену кількість пінів 14 цифрових. При 16 необхідних лініях в Uno довелося б використовувати розширювачі портів, що ускладнило б схему та код без жодної користі для демонстрації принципів керування. Крім того, обсягу Flash-пам'яті Uno 32 КБ ледь вистачило б для

програми з чергою викликів, таблицею символів для 7-сегментного індикатора та обробкою Serial-команд.

ESP32 являє собою значно потужнішу платформу з двоядерним 32-бітним процесором, вбудованим Wi-Fi та Bluetooth. З точки зору апаратних можливостей вона перевершує Mega за всіма параметрами. Однак для цього проєкту вирішальним виявився критерій підтримки в SimulIDE, на момент виконання роботи симуляція ESP32 у SimulIDE перебувала в експериментальній стадії, компонент Esp01 (TCP) працював нестабільно, з'єднання розривалися, AT-команди губилися. Крім того, використання Wi-Fi для зв'язку між контролером та SCADA-клієнтом ускладнило б архітектуру системи, не додавши їй наочності, адже в реальному ліфті контролер і диспетчерський пульт з'єднані дротовим каналом (RS-485), а не бездротовою мережею.

Arduino Mega 2560 було обрано для проєкту з трьох основних причин. Достатня кількість пінів 54 цифрових і 16 аналогових для підключення всієї периферії без додаткових розширювачів. Наявність 4 апаратних UART, що дозволяє організувати зв'язок із Python через один порт, залишаючи інші вільними для потенційного розширення системи. Повна та стабільна підтримка в SimulIDE, включаючи компонент SerialPort, який є ключовим елементом для організації двостороннього зв'язку із зовнішнім SCADA-клієнтом.

Хоча ESP32 пропонує значно більшу обчислювальну потужність та вбудовані мережеві інтерфейси, ці переваги не є критичними для задачі керування трьома поверхами, тоді як стабільність симуляції є вирішальною для дипломного проєкту.

2 АНАЛІЗ СЕРЕДОВИЩ ПРОГРАМУВАННЯ

2.1 Симулятори електронних схем та мікроконтролерів

Для реалізації цифрового двійника ліфта необхідно середовище, яке виконує дві ключові функції, а саме: точна симуляція поведінки мікроконтролера та можливість обміну даними із зовнішнім програмним забезпеченням. проведено аналіз чотирьох середовищ: Tinkercad Circuits [11], Wokwi [12], Proteus [13] та SimulIDE [8]. Кожна з них має свою нішу, і вибір визначався не загальною ефективністю, а відповідністю конкретним вимогам проєкту.

Tinkercad Circuits [11] — це онлайн-симулятор від Autodesk, орієнтований на початківців та освітні проєкти. симулятор підтримує Arduino Uno, базові компоненти (світлодіоди, резистори, кнопки, сервоприводи) та має вбудований редактор коду. Головна перевага це те що він працює в браузері, не потребує встановлення. Головний недолік це закрита архітектура: неможливо створити власний графічний компонент (наприклад, шахту ліфта з дверима), неможливо підключити зовнішню програму через COM-порт або TCP. Tinkercad працює виключно всередині свого середовища. Для навчання основам Arduino це чудовий інструмент, але для створення цифрового двійника з окремим SCADA-клієнтом не підходить.

Wokwi [12] — інший онлайн-симулятор, але з ширшим набором плат (Arduino Mega, ESP32, STM32, Raspberry Pi Pico та інші). На відміну від Tinkercad, Wokwi дозволяє створювати кастомні чіпи через JSON-конфігурацію. У цьому середовищі можна описати свій компонент із заданою поведінкою, це вже ближче до того, що потрібно. Однак індивідуально розроблені мікросхеми в Wokwi мають суттєве обмеження, їхня логіка виконується всередині симулятора, без можливості прямого обміну із зовнішнім ПЗ через UART. Можна налаштувати Wi-Fi-симуляцію (для ESP32), але для Arduino Mega цей механізм відсутній. Крім того, плавна анімація графічних об'єктів у Wokwi не підтримується. Персоналізований чіп може змінювати

колір або положення прямокутника стрибком, але не плавно. Для дискретної індикації світлодіодів чи дисплеїв цього достатньо, але не більше.

Proteus (Labcenter Electronics) [13] — професійний САПР із модулем симуляції VSM (Virtual System Modeling). Це найпотужніший інструмент із розглянутих. Він підтримує сотні мікроконтролерів, має величезну бібліотеку компонентів, дозволяє створювати власні графічні об'єкти з анімацією, і все це в одному вікні, без необхідності зовнішньої програми. По суті, Proteus може вирішити задачу візуалізації ліфта взагалі без Python, тобто лише внутрішніми засобами. Однак у цього рішення є два критичних недоліки, по-перше вартість. Базова ліцензія від \$248, а професійна понад \$4000. Для студентського проєкту це невиправдано. По-друге, інтеграція із зовнішнім ПЗ у Proteus реалізована через закритий API, що ускладнює створення розподіленої архітектури, де контролер та SCADA-клієнт є незалежними модулями.

SimulIDE [8] безкоштовний симулятор з відкритим вихідним кодом, який підтримує Arduino Mega, AVR, PIC та інші платформи. ця платформа має значно меншу бібліотеку компонентів, ніж Proteus, і не має вбудованого модуля плавної анімації. У ньому присутні графічні об'єкти такі як прямокутники або текст, але їхнє переміщення виглядає як стрибок, а не як плавний рух. Утім, дві ключові переваги переважили ці недоліки. Перша це наявність компонента SerialPort, який дозволяє з'єднати віртуальний контролер із зовнішньою програмою через стандартний COM-порт. Друга це відкрита архітектура, SimulIDE не нав'язує свою модель візуалізації, а дозволяє винести її назовні, що якраз і потрібно для реалізації концепції Digital Twin із зовнішнім SCADA-клієнтом. Порівняння середовищ подано в таблиці 2.1.

Таблиця 2.1 – Порівняння середовищ

Критерій	Tinkercad	Wokwi	Proteus	SimulIDE
Підтримка Arduino Mega	Відсутня	Присутній	Присутній	Присутній
Підключення зовнішнього ПЗ через UART	Відсутня	Відсутня	Присутній через API	Присутній
Можливість створення власної графіки	Відсутня	Присутня, зі стрибками	Присутній	Відсутня
Відкритий вихідний код	Відсутня	Відсутня	Відсутня	Присутній
Вартість ліцензії	Безкоштовно	Безкоштовно	від \$248	Безкоштовно

2.1.1 Обґрунтування вибору Simulide

Для проєкту критично важливою була можливість двостороннього обміну даними між віртуальним контролером та зовнішнім SCADA-клієнтом через стандартний протокол. З розглянутих платформ лише SimulIDE надає готовий компонент SerialPort, який дозволяє організувати такий обмін без додаткових технічних проблем. Відсутність вбудованої плавної анімації легко компенсується винесенням візуалізації в Python-застосунок на Tkinter, це рівно те архітектурне рішення, яке відповідає концепції розподіленої IoT-системи. Proteus міг би вирішити задачу візуалізації «всередині себе», але його

вартість та закритість суперечать меті роботи, а саме створенню доступного інструменту для швидкого налагодження ПЗ.

2.2 Мова для SCADA-клієнта

Щодо SCADA-клієнта, то наявними були декілька варіантів. Було вирішено сфокусуватися на трьох основних платформах, а саме C# з використанням Windows Forms, JavaScript з фреймворками Electron та Web Serial API, а також Python, який передбачав застосування Tkinter із бібліотекою pySerial. Кожен із цих шляхів мав свої сильні та слабкі моменти, які ми перевіряли за певними показниками: темпами розробки, легкістю взаємодії з COM-портом, рівнем виконання графічного інтерфейсу та простотою інсталяції.

2.2.1 C# (Windows Forms)

C# з переваг мав швидкий інтерфейс, хорошу підтримку COM-портів через System.IO.Ports та компіляцію в .exe. З недоліків прив'язка до Windows, хоча SimulIDE теж працює тільки з Windows, тому це не критично, а от відносно складний синтаксис для швидкого прототипування може стати проблемою.

2.2.2 JavaScript (Electron + Web Serial API)

Величезний плюс JavaScript очевидний, HTML/CSS дають сучасний вигляд та крутий рівень анімації, але Web Serial API працює тільки в браузерах на Chromium і вимагає ручного вибору порту користувачем. Для мети роботи, де задача полягала в скороченні часу, це зайва складність.

2.2.3 Python (Tkinter + pySerial)

Переходячи до варіанту, який було обрано для фінальної реалізації, неможливо не розповісти про простий синтаксис Python, величезну кількість бібліотек, а також те що pySerial працює з COM-портами без жодних додаткових інсталяцій, щодо до Tkinter, то ця бібліотека входить у набір стандартних бібліотек Python. з мінусів можу зазначити що Tkinter не надасть рівень графіки як у варіанті з використанням HTML/CSS, а швидкість анімації не така висока, як у C# або Electron. Але для задачі показати шахту ліфта, кабіну, двері та їх анімацію цього цілком достатньо. Також зроблено стисле порівняння цих мов програмування, яке подане в табл. 2.2

Таблиця 2.2 — Порівняння мов програмування для інтерфейса

Критерій	C# (Windows Forms)	Python (Tkinter)	JavaScript (Web)
Робота з мікроконтролером	Пряма	Немає	Немає
Створення GUI	Немає	Tkinter	HTML/CSS
Підтримка UART	Serial	pySerial	Web Serial API
Швидкодія	Висока	Достатня	Достатня
Складність освоєння	Середня	Низька	Середня

2.3 Висновки до розділу 2

Отже можна зазначити наступне, ліфт є складним об'єктом керування, що включає декілька незалежних підсистем: привід, двері, позиціонування та обробку викликів. Така структура робить його більш ніж придатним для демонстрації принципів проєктування вбудованих систем, від простих операцій на кшталт запуску руху до складних алгоритмів як черга викликів. Концепція цифрового двійника дозволяє тестувати алгоритми керування без фізи-

чного доступу до обладнання, що в свою чергу скорочує час одного циклу налагодження з 10-15 хвилин до 1-2 хвилин, усуває ризик пошкодження дорогих компонентів через помилки в коді та дає можливість працювати віддалено.

Розподілена архітектура контролер + SCADA-клієнт, на відміну від монолітних рішень як у Proteus, відповідає принципам побудови реальних промислових систем. Саме тому в проєкті було обрано зв'язку з двох незалежних програм, SimulIDE для симуляції контролера та Python/Tkinter для візуалізації.

Аналіз чотирьох платформ, як онлайн так і офлайн, для симуляції мікроконтролерів (Tinkercad, Wokwi, Proteus, SimulIDE) показав, що лише SimulIDE задовольняє ключову вимогу проєкту, можливість двостороннього обміну даними із зовнішнім програмним забезпеченням через стандартний протокол UART. Tinkercad та Wokwi є закритими онлайн-платформами без підтримки зовнішніх підключень. Proteus має необхідний функціонал, але його вартість від \$248 та закритий API роблять його непридатним для створення доступного інструменту розробки.

Відсутність у SimulIDE вбудованого рушія плавної анімації компенсується винесенням візуалізації в Python-застосунок. Таке рішення не є «обхідним маневром», а свідомим архітектурним вибором, що відповідає концепції розподілених IoT-систем. Контролер і операторський інтерфейс є незалежними модулями, з'єднаними стандартним каналом передачі даних.

3 ОРГАНІЗАЦІЯ ЗВ'ЯЗКУ МІЖ КОМПОНЕНТАМИ СИСТЕМИ

3.1 COM-порт

COM-порт (він же RS-232, він же послідовний порт) [9] — це один із найстаріших інтерфейсів передачі даних між комп'ютерами та периферійними пристроями. Йому вже понад півстоліття, але він досі використовується в промисловій та побутовій автоматичній, медичному обладнанні, банківських терміналах, касових апаратах, він знаходить застосування скрізь, де потрібна проста та надійна передача даних на невеликі відстані. Причина живучості такої старої технології — в простоті. Для роботи COM-порту потрібно всього три дроти: TX (передача), RX (прийм) і GND (заземлення) для узгодження електричних потенціалів. Проте в SimulIDE симуляція спрощена, та у компоненті SerialPort підключення не потребує фізичного з'єднання контуру заземлення, обмін даними здійснюється лише через сигнальні лінії TX та RX. Ніяких складних протоколів, ніяких мережевих карт, ніяких драйверів на рівні ядра — просто біти, які передаються по дроту. Всі ці фактори і обумовили вибір саме цієї технології, а також вона підтримується всіма трьома вузлами системи. Arduino Mega має апаратний UART — це той самий COM-порт, тільки на рівні мікросхеми. У коді Arduino для роботи з ним є об'єкт Serial — `Serial.begin()`, `Serial.print()`, `Serial.read()`. SimulIDE вміє емулювати цей UART через компонент SerialPort, який емулює справжній COM-порт Windows. Python має бібліотеку `pySerial`, яка вміє підключатися до будь-якого COM-порту в системі, читати з нього й писати в нього. Усе це працює без жодних проблем — просто з'єднав, налаштував швидкість, і обмін даними почалися.

Жоден інший інтерфейс не дав би такої простоти, Наочним прикладом є SPI та I²C. Це шини для спілкування між мікросхемами всередині одного пристрою, вони не призначені для підключення зовнішнього комп'ютера. Для цього потрібен окремий модуль (наприклад, W5500 для Arduino), підтримка в SimulIDE якого обмежена. Варто згадати USB, Arduino Mega може працювати через USB, але в SimulIDE це не симулюється — там є тільки віртуальний

UART. Так що COM-порт — це найнадійніший та простий вибір з усіх можливих.

3.2 Алгоритм роботи UART протоколу

UART (Universal Asynchronous Receiver-Transmitter) [14]— це асинхронний протокол. «Асинхронний» означає, що передавач і приймач не мають спільного тактового сигналу — замість цього вони заздалегідь домовляються про швидкість передачі (baud rate), у моєму випадку це 9600 біт.

Передача одного байту виглядає так.

Крок 1. Лінія у стані спокою перебуває у високому рівні (HIGH, логічна одиниця).

Крок 2. Передавач опускає лінію в LOW на один бітовий інтервал — це стартовий біт, сигнал про початок передачі даних.

Крок 3. Далі йдуть 8 бітів даних — власне байт, який передається.

Крок 4. Після даних іде стоповий біт (HIGH) — сигнал, що свідчить про кінець передачі файлів.

Крок 5. Якщо є наступний байт — усе повторюється з Кроку 2.

Таким чином, для передачі одного байту потрібно 10 бітів (старт + 8 даних + стоп). При швидкості 9600 біт один біт передається приблизно 104 мікросекунди, один байт — близько 1 мілісекунди. Найдовша команда в моєму протоколі — MOVING:1->3 — це 11 символів (байт), тобто приблизно 11 мілісекунд на передачу, плюс така сама відповідь від контролера. Сумарна затримка — менше 25 мілісекунд. Для системи керування ліфтом, де події відбуваються з інтервалом у секунди, це миттєво.

У моєму проєкті використовуються такі налаштування UART:

~ швидкість: 9600 біт. Можливо поставити і вищу швидкість, наприклад, при 115200 біт програма теж працювала, але при 9600 менше шансів отримати помилку через можливу затримку симулятора. Для декількох команд на секунду цієї швидкості більш ніж достатньо;

~ біти даних: 8. Стандарт для текстових даних;
 ~ стопові біти: 1. Мінімально можливе значення;
 ~ контроль парності: вимкнено. У моєму випадку (текстові команди на малій відстані) ймовірність помилки передачі близька до нуля, тому контроль парності не потрібен.

3.3 Драйвер віртуального кабелю com0com

Оскільки в проєкті використовується не фізична плата Arduino, а її віртуальна модель у SimulIDE, пряме підключення через USB-кабель неможливе. SimulIDE не емулює USB-інтерфейс — замість цього він надає компонент SerialPort, який працює через стандартний COM-порт операційної системи. Щоб організувати обмін даними між віртуальним контролером та зовнішнім Python-застосунком, використовується драйвер com0com.

Механізм роботи com0com наступний, драйвер реєструє в системі два послідовних порти (у моєму випадку COM2 та COM3) і організовує між ними програмний міст. На рівні операційної системи це виглядає як два звичайних COM-порти. На рівні програм — як прозорий канал: байти, відправлені в один кінець, з'являються на іншому без змін і без затримок (якщо не брати до уваги затримки самої Windows).

Між Arduino та Python циркулюють текстові команди — це саме те що передається через COM-порт, короткі рядки, які закінчуються символом переносу рядка (\n). Формат простий: КОМАНДА:ДАНІ.

З кількох причин було обрано текст, а не бінарний формат. По-перше, текстові команди можна читати без додаткових інструментів користувач може відкрити термінал, побачити MOVING:1->3, і одразу зрозуміти, що відбувається. По-друге, на етапі налагодження можна відправити команду вручну через будь-який послідовний термінал, наприклад Arduino Serial Monitor і подивитися на реакцію контролера. По-третє, для моєї задачі швидкість не критична, команди передаються досить рідко, приблизно одна або дві на секу-

нду, і навіть на 9600 біт затримка становить мілісекунди. Усі команди для плати з вказанням напрямку від Arduino до Python та навпаки вказано в таблиці 3.1.

Таблиця 3.1 — Команди протоколу обміну

Команда	Напрямок	Що означає
FLOOR:N	Python - Arduino	Виклик на поверх N
MOVING:X->Y	Arduino - Python	Ліфт почав рух з поверху X на поверх Y
ARRIVED:N	Arduino - Python	Ліфт прибув на поверх N
DOOR:OPEN	Обидва	Двері відчинені (команда або підтвердження)
DOOR:CLOSED	Обидва	Двері зачинені (команда або підтвердження)
READY	Arduino - Python	Контролер завантажився і готовий до роботи

3.3.1 Технічні особливості налаштування віртуальних портів

В параметрах віртуального COM-порту обрано COM2 та COM3, а не COM1 та COM2. Так зроблено через те що, COM1 у більшості комп'ютерів зарезервований на апаратному рівні, навіть якщо фізичного роз'єму RS-232 на корпусі немає, система вважає, що він є, і не дає використовувати цей номер для віртуального порту. Таким чином використовуючи COM2 і COM3 можна бути впевненим що вони будуть гарантовано вільні.

Також є важливий нюанс з технологією TCP/IP. Під час аналізу була ідея відмовитися від COM-порту та організувати зв'язок через TCP/IP, тобто

запустити локальний сервер на Python, до якого підключається SimulIDE. У SimulIDE є компонент Esp01 (TCP), який теоретично дозволяє це зробити, але на жаль на практиці він працює нестабільно, тому що цей інструмент є лише у бета версії Simulide 2.0. Під час експериментів були наступні проблеми: з'єднання рветься, AT-команди губляться, симулятор зависає при спробі відправити дані. COM-порт через com0com рішення не найсучасніше, але воно працює очікувано та стабільно.

3.4 Висновки до розділу 3

UART [14] є безальтернативним, але і той же час логічним та раціональним рішенням. Жоден інший стандартний інтерфейс (SPI, I²C, USB, Ethernet) не підтримується одночасно Arduino, SimulIDE та Python у конфігурації, необхідній для проєкту. Разом із тим, можливостей UART цілком достатньо для вирішення поставленої задачі: швидкість 9600 біт забезпечує передачу найдовшої команди протоколу розміром 11 байт менш ніж за 12 мілісекунд, що для системи керування ліфтом, де події відбуваються з інтервалом у секунди, є миттєвою реакцією.

Текстовий протокол команд є виправданим спрощенням, тому що використання текстових команд замість бінарного формату незначно збільшує обсяг трафіку, але дає суттєві переваги при налагодженні. Команди можна читати без додаткових інструментів, відправляти вручну через термінал, а також легко розширювати протокол новими командами без зміни парсера на приймальній стороні.

Com0com є надійнішим за TCP/IP у середовищі SimulIDE. Спроба організувати зв'язок через компонент Esp01 (TCP) виявилася невдалою через нестабільну роботу цього компонента в симуляторі. Драйвер com0com, який створює віртуальну пару COM-портів (COM2 — COM3), працює стабільно та передбачувано.

4 ПРОЄКТУВАННЯ СИСТЕМИ

4.1 Діаграма прецедентів та опис акторів

Для визначення функціональних вимог до системи було побудовано діаграму прецедентів рисунок 4.1, яка показує, хто і як взаємодіє з програмно-апаратним комплексом. У системі виділено трьох акторів із різними ролями.

Актор 1. Розробник. Це основний користувач системи. Він пише прошивку для контролера в Arduino IDE, компілює її в HEX-файл, завантажує в SimulIDE, запускає симуляцію та спостерігає за поведінкою ліфта через SCADA-інтерфейс. Якщо алгоритм працює некоректно, виправляє код, перекомпілює, перезавантажує симуляцію.

Актор 2. Оператор. Це роль, яка імітує кінцевого користувача промислової SCADA-системи. Він не має доступу до коду, не змінює прошивку, лише користується готовим інтерфейсом. Його функції це виклик ліфта на потрібний поверх, спостереження за рухом кабіни та станом дверей, перегляд поточного поверху на табло. Оператор не може зупинити симуляцію або змінити алгоритм, тільки взаємодіяти з готовою системою.

Актор 3. Адміністратор. Це роль, яка відповідає за розгортання системи на новому комп'ютері. Він запускає скрипт `setup_com_ports.bat` для встановлення віртуального COM-порту, перевіряє наявність Python та необхідних бібліотек, запускає обидві частини системи (SimulIDE та Python-застосунок) і переконується, що зв'язок між ними встановлено.

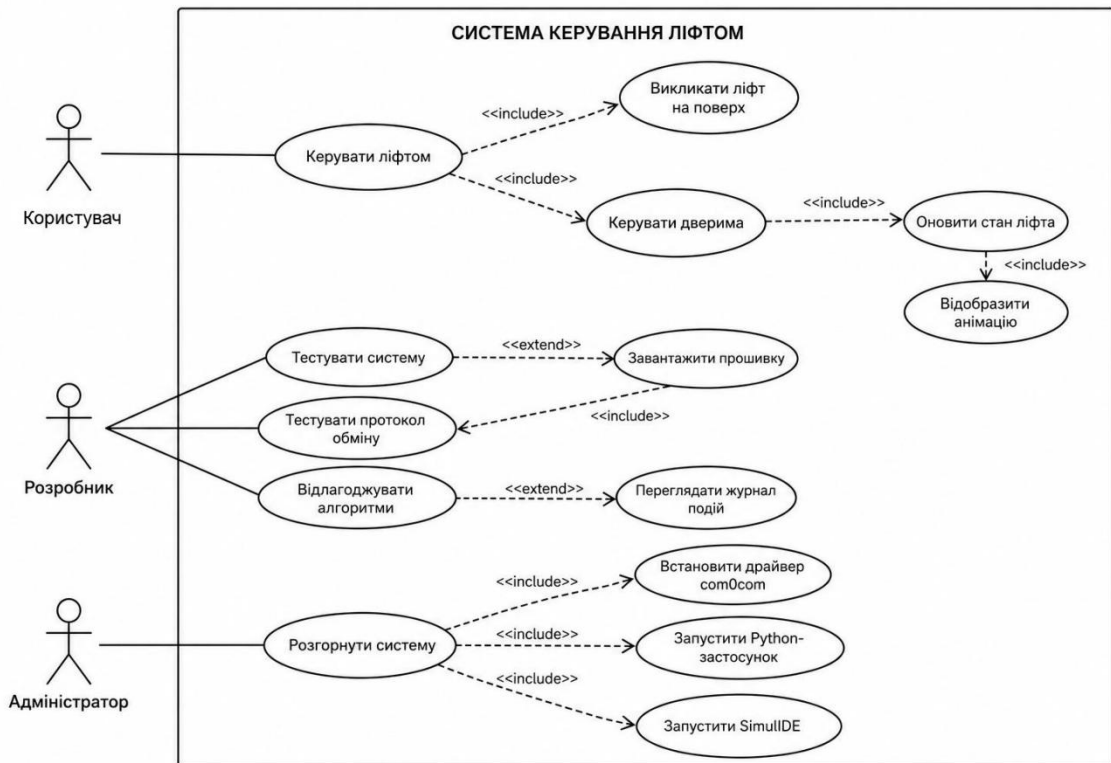


Рисунок 4.1 — Діаграма прецедентів

Розмежування ролей є важливим, оскільки показує, що система орієнтована не лише на розробника, який може безпосередньо змінювати код, а й на кінцевого користувача, що взаємодіє виключно через графічний інтерфейс. Це відповідає реальній архітектурі промислових SCADA-систем, де оператор не має доступу до логіки контролера.

4.2 Структурна схема системи

Структурна схема показує, з яких компонентів складається система і як ці компоненти пов'язані між собою. У моєму проєкті система має чіткий поділ на нижній (контролер) та верхній (SCADA-інтерфейс) рівні.

4.2.1 Нижній рівень (SimulIDE)

Це «мозок» системи. Тут знаходиться віртуальна плата Arduino Mega 2560, яка виконує скомпільовану прошивку. До плати підключено:

- ~ 6 кнопок виклику — 3 зовнішні (на поверхових постах) і 3 внутрішні (у кабіні). Кнопки підключені до цифрових пінів 22, 23, 24 (зовнішні) та 40, 41, 42 (внутрішні). Усі кнопки використовують внутрішню підтяжку до напруги живлення (INPUT_PULLUP) — при натисканні на піні з'являється LOW;

- ~ світлодіоди напрямку — червоний (пін 27) для руху вгору, зелений (пін 29) для руху вниз;

- ~ 7-сегментний індикатор поверху — підключений до пінів 30-36, показує поточний поверх цифрою;

- ~ компонент SerialPort — з'єднаний із пінами RX0/TX0 Arduino. Через нього дані потрапляють у віртуальний COM-порт і далі — у Python.

4.2.2 Верхній рівень (Python/Tkinter)

Це «очі» системи. Python-застосунок отримує дані від контролера через COM-порт і перетворює їх на зрозумілу картинку. Він складається з трьох основних модулів:

- ~ модуль зв'язку (Serial Manager). Відповідає за відкриття COM-порту (serial.Serial('COM3', 9600)), читання вхідних команд у фоновому потоці (read_serial), відправку команд від інтерфейсу до контролера (send_floor_command, send_door_command);

- ~ модуль візуалізації (GUI). Містить усю графіку: шахту ліфта (Canvas), кабінку (прямокутник, що рухається), двері (два прямокутники, що роз'їжджаються), табло поверху, індикатор напрямку (стрілка), панель керування з кнопками;

~ модуль анімації (Animation Engine). Реалізує плавний рух: функція `move_cabin_to_floor` поступово змінює координату Y кабіни, функції `animate_doors_open` та `animate_doors_close` анімують стулки дверей. Для плавності використовується функція пом'якшення `ease_out_cubic` (кубічна крива затухання).

4.2.3 Транспортний рівень (com0com)

Між двома основними рівнями знаходиться віртуальний кабель — пара портів COM2 $\rightleftharpoons$ COM3, створена драйвером `com0com` [9]. SimulIDE [8] підключається до COM2, Python — до COM3. Усі байти, відправлені в один порт, з'являються в іншому. Сам драйвер не бере участі в логіці роботи — він лише транспортує дані. нижче представлена діаграма з інформаційними потоками системи (рис. 4.2).

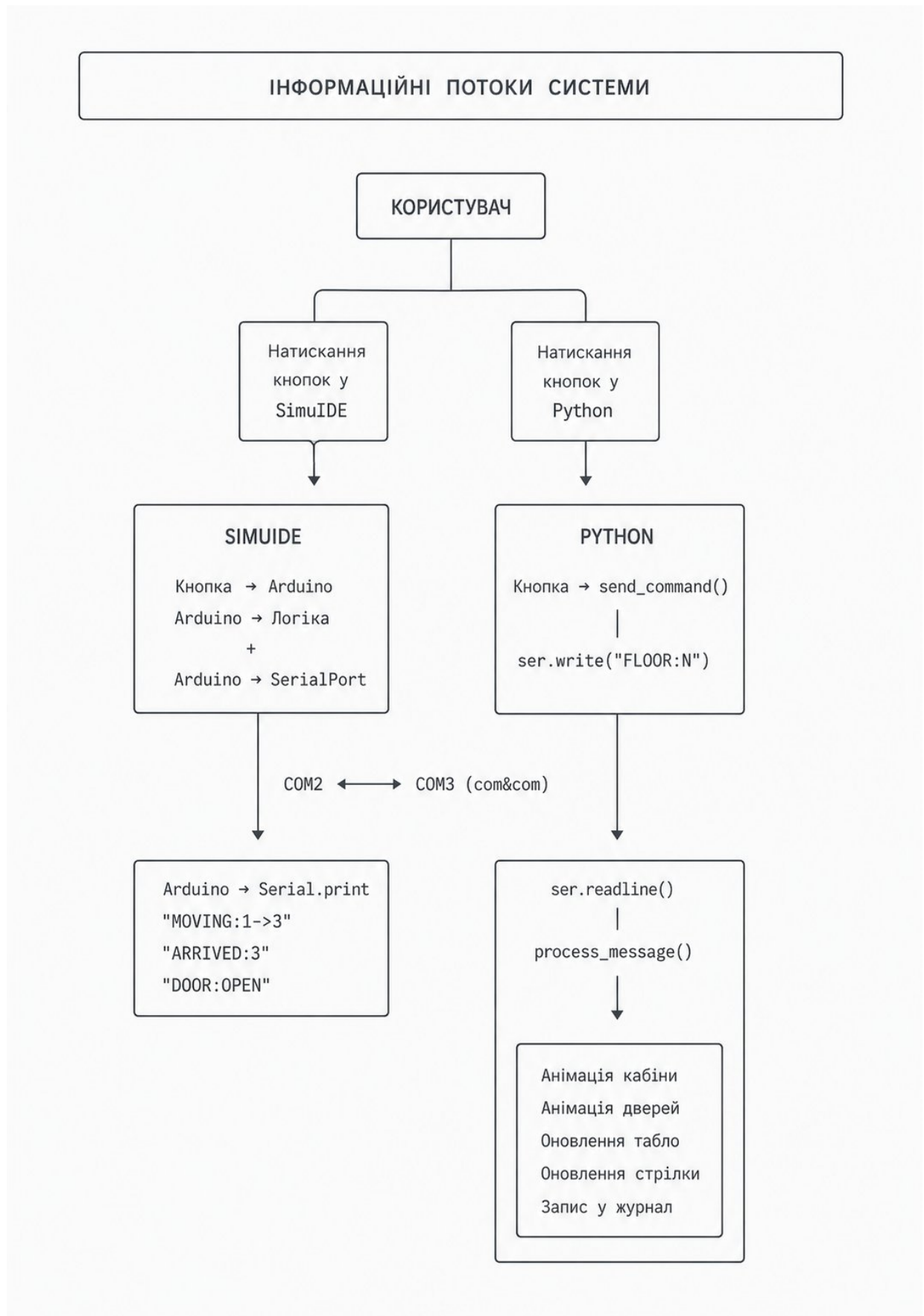


Рисунок 4.2 — Діаграма інформаційних потоків системи

4.3 Опис апаратної частини (схема в SimulIDE)

Апаратна частина системи реалізована у вигляді віртуальної схеми в середовищі SimulIDE [8]. Центральним елементом схеми є мікроконтролер Arduino Mega 2560, до якого підключено всі периферійні компоненти.

Кнопки виклику. Використовується 6 тактових кнопок без фіксації (Push Button, нормально розімкнуті). Кожна кнопка одним контактом підключена до цифрового піну Arduino, іншим — до шини GND. У коді для всіх кнопок увімкнено внутрішній підтягуючий резистор (INPUT_PULLUP): у стані спокою на піні HIGH, при натисканні — LOW. Апаратного анти відскоку (debounce) не передбачено — захист від брязкоту контактів реалізовано програмно (затримка 200 мс між повторними спрацьовуваннями). Два світлодіоди (червоний та зелений) підключено через струмообмежувальні резистори 220 Ом. Червоний (пін 27) засвічується при русі вгору, зелений (пін 29) — при русі вниз, обидва погашені, коли ліфт стоїть на поверсі.

7-сегментний індикатор. Використовується індикатор із загальним катодом (Common Cathode). 7 сегментів (A-G) підключено до пінів 30-36 через резистори 220 Ом. Спільний катод підключено до GND. Для відображення цифр 0-9 у коді визначено двовимірний масив digitMap [7], [10], де кожен рядок відповідає цифрі, а кожен стовпець — сегменту. Одиниця означає «сегмент горить», нуль — «сегмент погашений».

Таблиця розпиновки, а також схема плати. Повна відповідність між компонентами схеми та пінами Arduino Mega наведена в таблиці 4.1. а схематичне підключення плати у рисунку 4.3.

Таблиця 4.1 — Призначення виводів компонентів схеми

Компонент	Позначення в коді	Пін Arduino	Примітка
Кнопка «Виклик 1» (зовнішня)	BTN_EXT_1	22	INPUT_PULLUP
Кнопка «Виклик 2» (зовнішня)	BTN_EXT_2	23	INPUT_PULLUP
Кнопка «Виклик 3» (зовнішня)	BTN_EXT_3	24	INPUT_PULLUP
Кнопка «Поверх 1» (кабіна)	BTN_INT_1	40	INPUT_PULLUP
Кнопка «Поверх 2» (кабіна)	BTN_INT_2	41	INPUT_PULLUP
Кнопка «Поверх 3» (кабіна)	BTN_INT_3	42	INPUT_PULLUP
Сегмент В (індикатор)	SEG_B	31	Через резистор 220 Ом
Сегмент С (індикатор)	SEG_C	32	Через резистор 220 Ом
Сегмент D (індикатор)	SEG_D	33	Через резистор 220 Ом
Сегмент E (індикатор)	SEG_E	34	Через резистор 220 Ом
Сегмент F (індикатор)	SEG_F	35	Через резистор 220 Ом
Сегмент G (індикатор)	SEG_G	36	Через резистор 220 Ом
SerialPort TX	—	RX1 (пін 0)	Передача від Arduino до Python

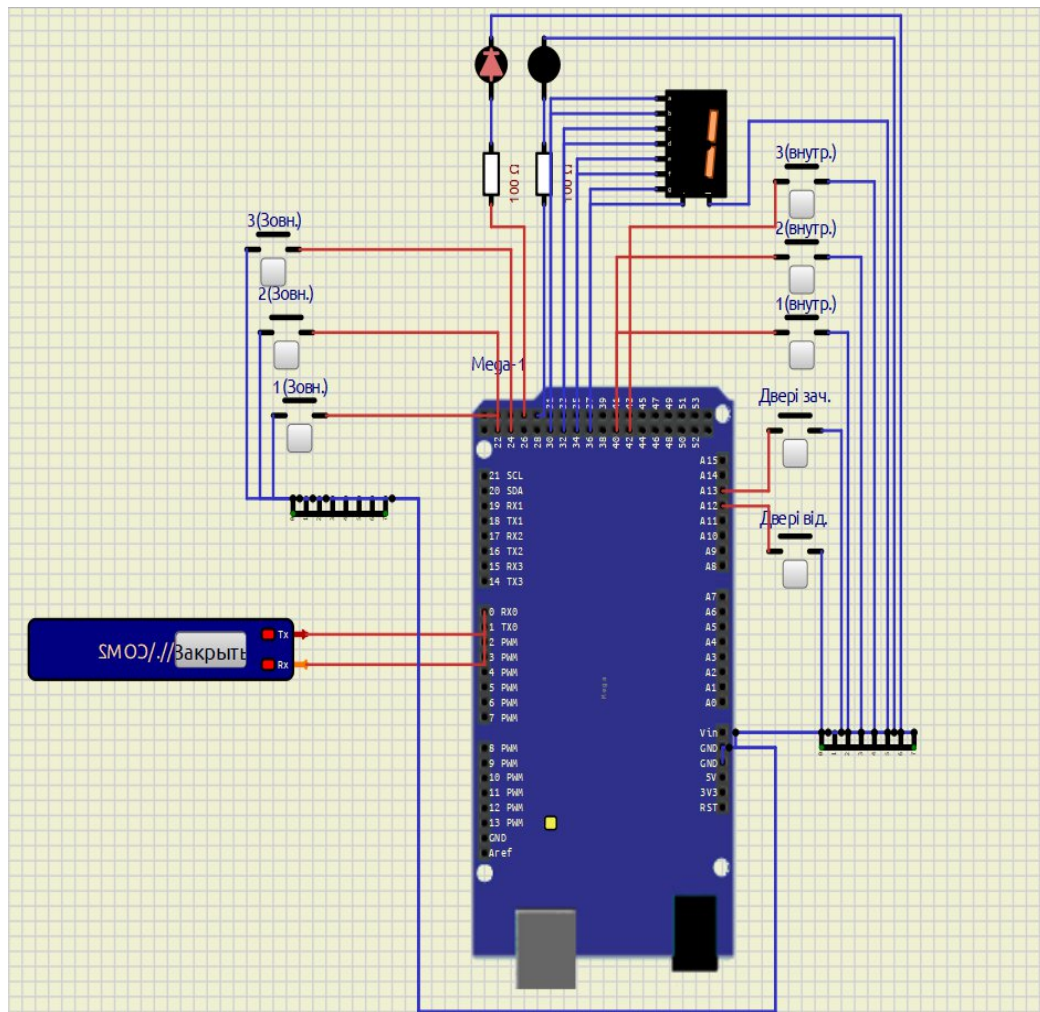


Рисунок 4.3 — Схема плати Arduino 2560

Запропонована архітектура характеризується тим, що дозволяє масштабувати систему. Це дозволяє збільшувати кількість поверхів або точок керування без необхідності внесення змін до загальної структури програмно апаратного комплексу.

4.4 алгоритм роботи контролера

Програма для Arduino Mega написана мовою C++ і побудована за класичною схемою циклічного опитування. Уся логіка виконується в одному нескінченному циклі `loop()`, який повторюється, поки симуляція активна.

Структура `loop()`. Кожна ітерація циклу послідовно виконує чотири блоки:

Читання команд від Python. Викликається `Serial.available()`, яка перевіряє, чи є нові дані у вхідному буфері UART. Якщо дані є — вони читаються через `Serial.readStringUntil('\n')`, і отриманий рядок аналізується.

Розпізнаються три типи команд:

- ~ FLOOR:N — виклик на поверх. Викликається `AddList(f)`, яка додає номер поверху в чергу;
- ~ DOOR:OPEN — відкрити двері. Викликається `openDoors()`;
- ~ DOOR:CLOSE — закрити двері. Викликається `closeDoors()`.

Опитуються всі вісім кнопок: три зовнішні, три внутрішні та дві кнопки дверей. Опитування кнопок виклику виконується лише коли ліфт не рухається. Для кожної кнопки реалізовано програмний захист від брязкоту контактів із використанням окремих змінних, це означає що спрацьовування фіксується лише якщо з моменту попереднього натискання цієї ж кнопки минуло щонайменше 100 мс. Кнопки виклику додають номер поверху в чергу через `AddList()`, кнопки дверей викликають `openDoors()` або `closeDoors()`.

Логіка руху наступна, якщо прапорець `isMoving` встановлено, перевіряється, чи минув потрібний час із моменту старту (`Move_Duration = 3000` мс). Якщо умова виконується то ліфт вважається таким, що прибув на поверх, а змінна `currentFloor` оновлюється, прапорець `isMoving` скидається, світлодіоди напрямку (`LED_UP`, `LED_DOWN`) та світлодіод руху (`LED_MOVING`) гасяться, оновлюється 7-сегментний індикатор через `displayDigit()`, відправляється повідомлення `ARRIVED:N` у `Serial`, викликається `openDoors()`.

Обробка черги виконується після завершення руху, якщо ліфт стоїть (`!isMoving`), двері закриті (`!doorsOpen`), а черга не порожня (`queueSize > 0`) викликається `ListTake()`, яка повертає наступний поверх із черги. Якщо цей поверх не збігається з поточним викликається `startMoving(next)` і рух продовжується. Блок схема алгоритму обробки черги в рисунку 4.4.

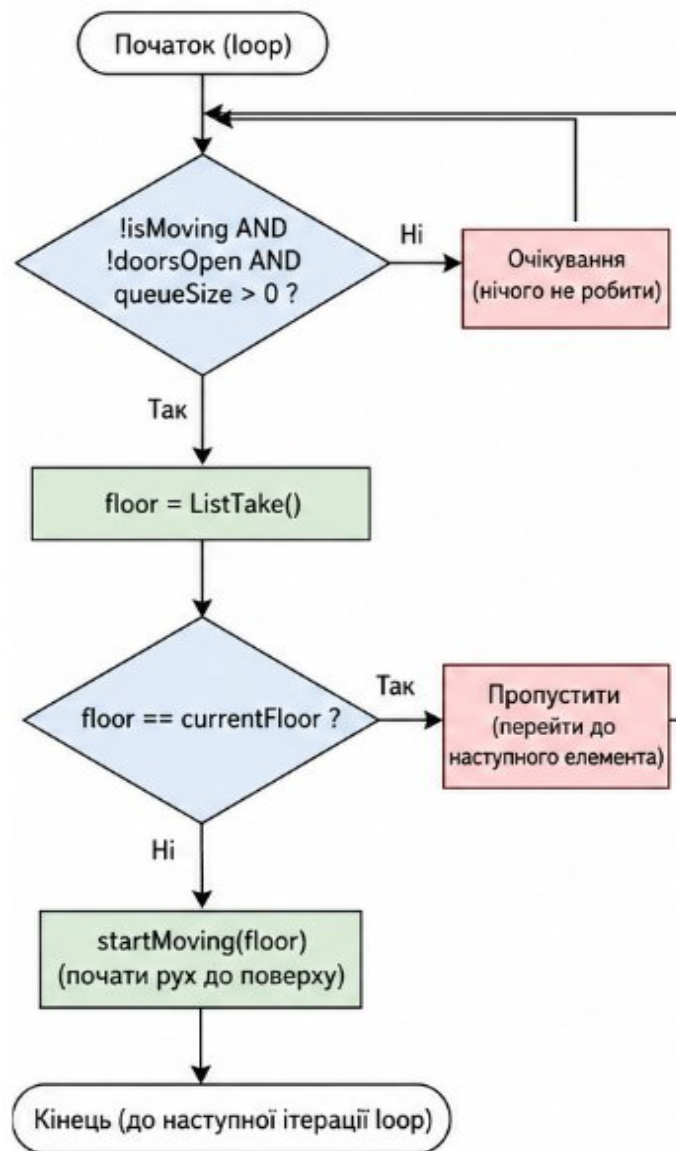


Рисунок 4.4 — Блок схема алгоритму обробки черги

Функція `AddList(int floor)`, додає виклик у чергу. Виконує три перевірки: якщо викликаний поверх збігається з поточним, то ігнорує виклик, якщо такий поверх уже є в черзі — ігнорує дублікат, якщо черга складає більше 10 елементів ігнорує новий виклик. Після успішного додавання відправляє в `Serial` підтвердження `QUEUE ADD: N`. Блок схема алгоритму додавання у чергу в рисунку 4.5.

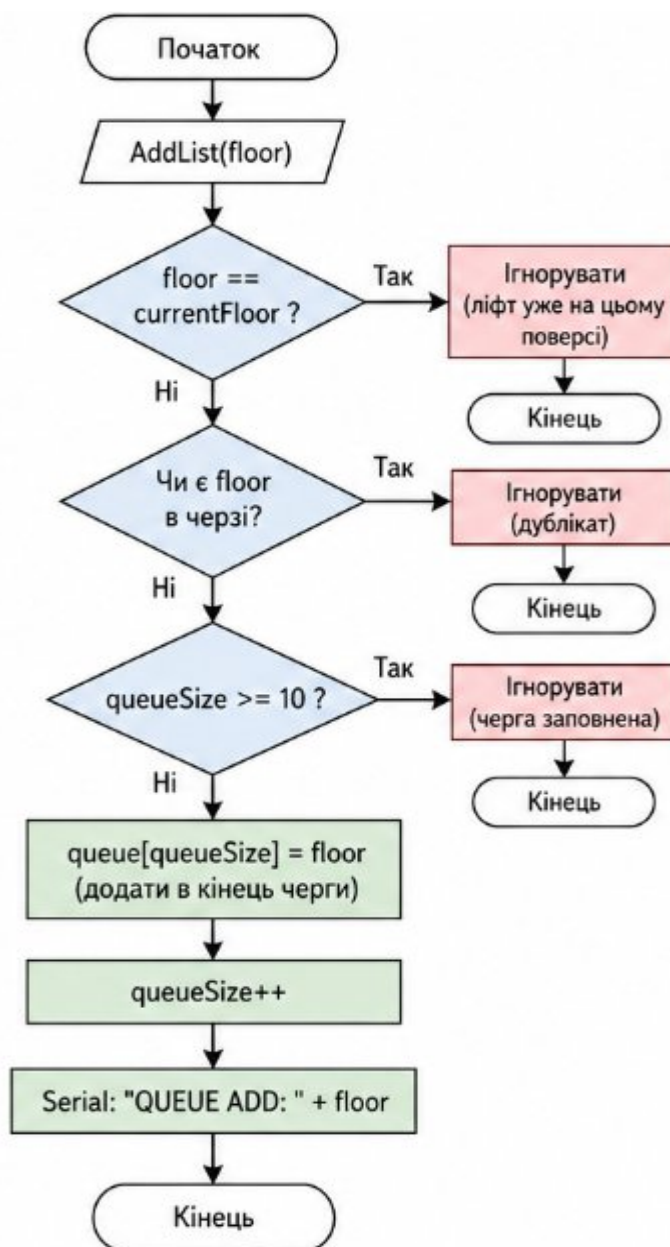


Рисунок 4.5 — Блок схема алгоритму додавання у чергу

Функція ListTake(). Повертає перший елемент черги, зсуває всі наступні елементи на одну позицію вліво, зменшує queueSize. Якщо черга порожня повертає -1. Блок схема алгоритму вилучення з черги рисунку 4.6.

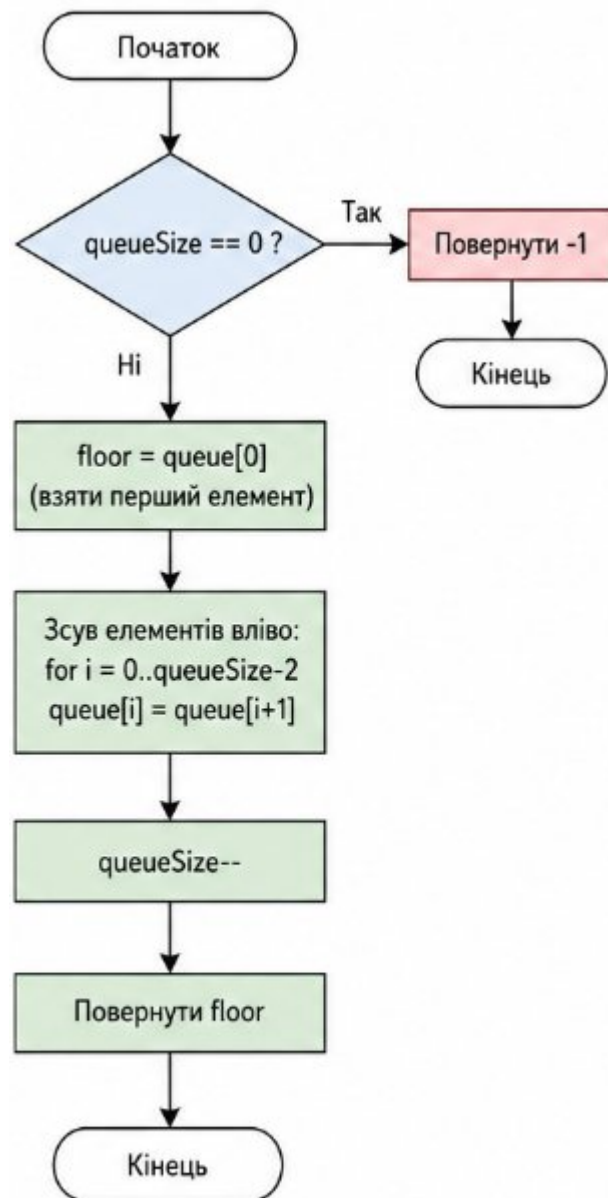


Рисунок 4.6 — Блок схема алгоритму вилучення з черги

Функція `startMoving(int floor)`. Виконує додаткову перевірку: якщо цільовий поверх збігається з поточним негайно виходить. Інакше встановлює `targetFloor = floor`, `isMoving = true`, запам'ятовує поточний час (`moveStartTime = millis()`), вмикає світлодіод руху та один зі світлодіодів напрямку, відправляє в `Serial` повідомлення `MOVING:X->Y`.

Функції дверей перевіряє, що двері ще не відкриті й ліфт не рухається, після чого встановлює `doorsOpen = true`, відправляє `DOOR:OPEN` і запускає

таймер автоматичного закриття (`doorTimer = millis()`, `doorTimerActive = true`). `closeDoors()` перевіряє, що двері відкриті, скидає прапорець і відправляє `DOOR:CLOSED`.

Автоматичне закриття дверей присутнє у кожній ітерації `loop()` перевіряється, якщо таймер активний, ліфт не рухається, і з моменту відкриття минуло 3 секунди викликається `closeDoors()` і таймер деактивується.

захист від брязкоту контактів кнопок, був зроблений для кожної з восьми кнопок використовується окрема змінна для відстеження часу останнього спрацьовування (`lastBtnTime1`, `lastBtnTime2` тощо). Якщо між двома натисканнями однієї кнопки минуло менше 100 мс друге натискання ігнорується. Це програмний метод захисту від брязкоту контактів, який не потребує додаткових апаратних компонентів.

Відображення поверху виконується завдяки функції `displayDigit(int number)` приймає число від 0 до 9 і запалює відповідні сегменти 7-сегментного індикатора згідно з таблицею `digitMap`. Якщо число виходить за межі діапазону функція нічого не робить.

4.5 Архітектура Python-застосунку

Python-застосунок виконує роль SCADA-клієнта: приймає дані від контролера через COM-порт, відображає стан ліфта в реальному часі, дозволяє оператору керувати системою та веде журнал подій. Архітектурно застосунок поділено на чотири модулі, кожен із яких відповідає за свій аспект роботи. Інтерфейс зображено на рисунку 4.7.

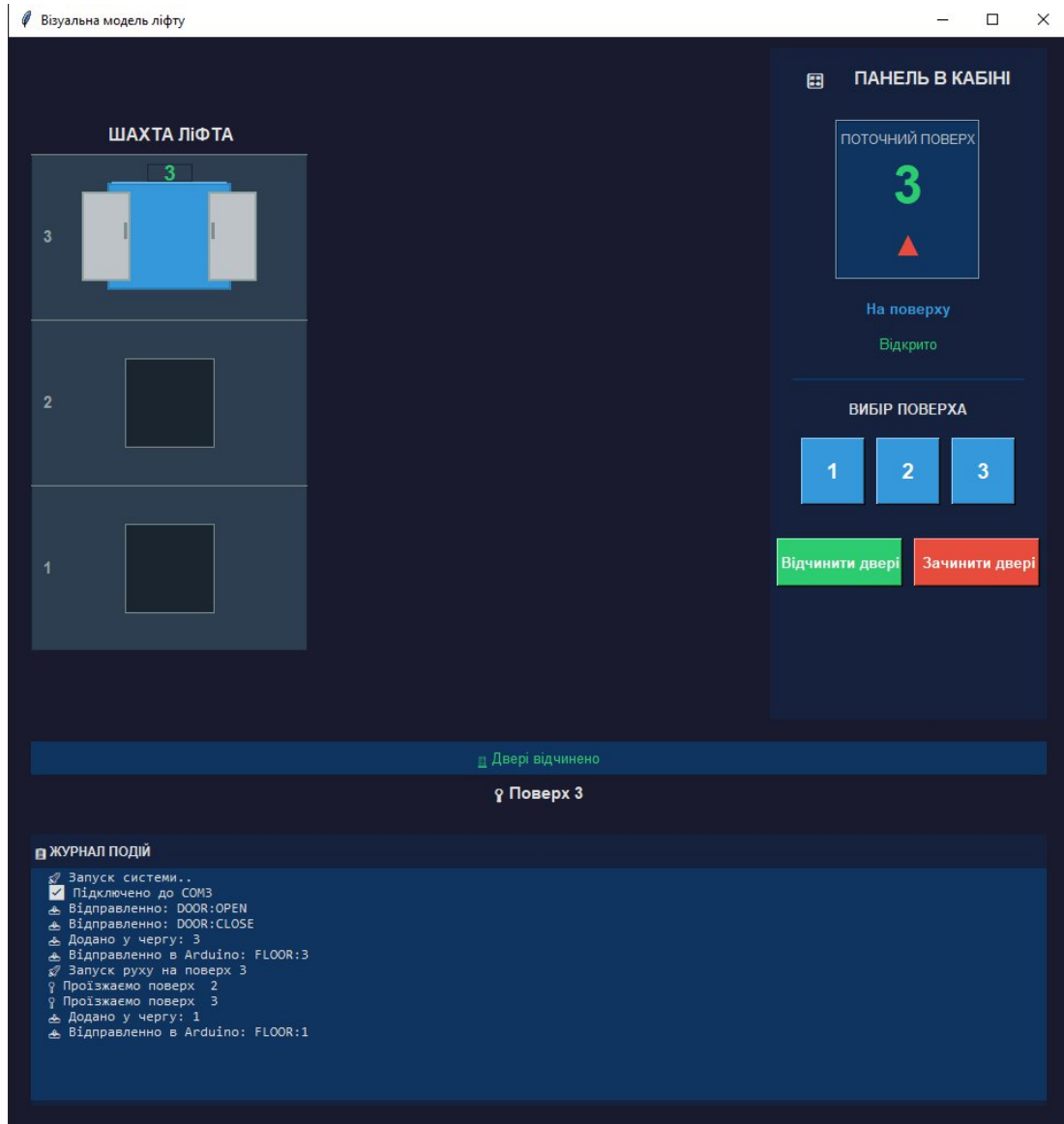


Рисунок 4.7 — SCADA-інтерфейс в Python застосунку

4.5.1 Модуль зв'язку (Serial Manager)

Цей модуль відповідає за весь обмін даними з віртуальним контролером через COM-порт, він реалізований у функціях:

~ `connect_serial()` — відкриває порт COM3 на швидкості 9600 бод через бібліотеку `pySerial`. Перевіряє наявність порту в системі (через `serial.tools.list_ports`), виводить статус підключення в інтерфейс. У разі помилки (порт не знайдено, порт зайнятий, недостатньо прав) показує користувачеві зрозуміле повідомлення;

~ `read_serial()` — фоновий потік, який безперервно читає дані з порту через `ser.readline()`. Отриманий рядок передається в головний потік через `window.after(0, process_message, line)`, оскільки Tkinter не є потокобезпечним і не дозволяє змінювати інтерфейс із побічних потоків;

~ `send_floor_command(floor)` — відправляє команду `FLOOR:N` у порт та містить трирівневий захист, якщо викликаний поверх збігається з поточним — команда не відправляється, а якщо з моменту попередньої команди минуло менше `COMMAND_PAUSE` (0,5 с) — команда ігнорується (захист від випадкових подвійних кліків). Якщо такий поверх уже є в черзі — дублікат не додається. Після успішної відправки викликається `process_queue()` для негайної обробки черги;

~ `send_door_command(action)` — відправляє `DOOR:OPEN` або `DOOR:CLOSE`. Особливість: анімація дверей запускається миттєво, не чекаючи відповіді від контролера це механізм Optimistic UI, детальніше описаний у підрозділі 4.7. Команда контролеру відправляється паралельно у фоновому режимі.

4.5.2 Модуль обробки повідомлень (Protocol Parser)

Цей модуль реалізований у функції `process_message(msg)` і відповідає за аналіз вхідних команд від Arduino. `MOVING:X->Y` — перевіряє, чи не є рух холостим ($X = Y$, наприклад, `3->3`). Якщо ні визначає напрямок, оновлює стрілку ($\blacktriangle/\blacktriangledown$), закриває двері, запускає анімацію руху кабіни через `move_cabin_to_floor(Y)`.

`ARRIVED:N` — оновлює поточний поверх на всіх елементах інтерфейсу (табло в шахті, табло на панелі, статус-бар), встановлює статус «НА ПОВЕРСІ», запускає анімацію відкриття дверей, через 1,5 секунди гасить стрілку напрямку.

`DOOR:OPEN / DOOR:CLOSED` — запускає відповідну анімацію та оновлює текстовий статус дверей на панелі керування.

READY — сигналізує, що контролер завантажився, переводить інтерфейс у робочий стан.

FLOOR:N (від Arduino) — отримується, коли користувач натискає кнопку виклику в SimulIDE. Додає поверх у чергу, якщо його там ще немає.

4.5.3 Модуль черги (Queue Manager)

Оскільки Arduino має власну чергу, Python-застосунок також підтримує клієнтську чергу для оптимістичного керування. Реалізовано з використанням стандартної бібліотеки `collections.deque`:

~ floor_queue — двостороння черга (deque), у якій зберігаються викликані поверхи;

~ queue_worker() — фоновий потік, який послідовно обробляє чергу: витягує наступний поверх через `popleft()`, пропускає його, якщо ліфт уже на ньому, запускає рух кабіни через `move_cabin_to_floor()`, чекає завершення руху та відкриття/закриття дверей;

~ process_queue() — запускає `queue_worker()` у новому потоці, якщо той ще не активний (`processing_queue == False`). Використовується блокування `queue_lock` для уникнення стану гонитви (race condition) при одночасному додаванні викликів із різних джерел.

4.5.4 Модуль візуалізації та анімації

Графічний інтерфейс побудований на бібліотеці Tkinter, яка входить до стандартної бібліотеки Python. Інтерфейс складається з трьох панелей:

~ ліва панель — шахта ліфта. Реалізована на компоненті Canvas (250×450 пікселів). Містить статичні елементи (фон шахти, лінії поверхів, дверні прорізи) та рухомі елементи: кабіну (`cabin_body`), двері (`door_left`, `door_right`), ручки дверей (`door_handle`), табло над кабіною (`cabin_number`). Усі рухомі елементи позначені тегом `cabin_part` для синхронного переміщення;

~ права панель — панель керування в кабіні. Містить табло поточно-го поверху (цифра 36-м шрифтом), індикатор напрямку (стрілка ▲/▼/■), статус дверей (ВІДЧИНЕНО/ЗАЧИНЕНО), кнопки вибору поверху (1, 2, 3), кнопки ручного керування дверима;

~ нижня панель — статус і журнал подій. Містить рядок стану (підключення до COM-порту) та журнал подій (Text із моноширинним шрифтом Consolas), у якому відображаються всі команди протоколу обміну.

Анімація руху. Функція `move_cabin_to_floor(floor)` обчислює цільову координату Y кабіни за формулою $\text{SHAFT_HEIGHT} - \text{floor} * \text{FLOOR_HEIGHT}$. Рух розбивається на 10-20 проміжних кадрів (залежно від відстані). Для плавності використовується функція пом'якшення `slow_animation(t)` (кубічна крива затухання: $1 - (1-t)^3$), завдяки якій кабіна швидко набирає швидкість і плавно гальмує перед зупинкою. Функції `animate_doors_open()` та `animate_doors_close()` змінюють координати X лівої та правої стулок на 35 пікселів за 15 кадрів із тим самим пом'якшенням `slow_animation(t)`. Після відкриття автоматично запускається таймер на 3 секунди, після чого двері закриваються (якщо не були закриті вручну).

Функція `update_direction(direction)` змінює символ стрілки на панелі керування: ▲ (червоний, рух вгору), ▼ (зелений, рух вниз), ■ (сірий, стоїть), тим самим створює індикацію напрямку.

4.6 Optimistic UI

Optimistic UI (Оптимістичний користувальницький інтерфейс) [15] — це підхід у фронтенд-розробці, при якому інтерфейс реагує на дії користувача негайно, не очікуючи підтвердження від сервера. У класичній схемі взаємодії «запит — відповідь» інтерфейс відправляє команду контролеру й чекає підтвердження, перш ніж показати результат, користувач натискає кнопку і нічого не відбувається, поки команда не дійде до Arduino через COM-порт, не обробиться там, і відповідь не повернеться назад. Цей шлях займає 20-30 мілі-

секунд у найкращому випадку, а з урахуванням особливостей симуляції в SimulIDE — іноді й більше. Користувач бачить мікрозатримку, інтерфейс здається повільним.

У веб-розробці цю проблему вирішили давно. Коли ви ставите лайк у соцмережі, сердечко загоряється миттєво, хоча насправді запит до сервера ще навіть не дійшов. Якщо сервер згодом відповідає помилкою, лайк забирається, але 99% часу все працює і користувач не помічає затримок. Цей підхід називається Optimistic UI — «оптимістичний інтерфейс», тому що він оптимістично припускає, що операція виконується успішно.

У моєму проєкті Optimistic UI застосовано для керування дверима — це та частина системи, де затримка найпомітніша, оскільки двері реагують на пряму команду оператора, а не на внутрішню логіку контролера. Класичний підхід виглядав би так. Надіслати DOOR:OPEN у COM-порт, дочекатися, поки Arduino прочитає команду, дочекатися, поки Arduino надішле відповідь DOOR:OPEN, запустити анімацію. Між натисканням кнопки й початком руху дверей пройшло б щонайменше 40-50 мілісекунд — цілком достатньо, щоб людське око помітило затримку. Натомість у моїй реалізації анімація запускається миттєво, не чекаючи жодної відповіді. Команда контролеру відправляється паралельно, у фоновому режимі. Користувач натискає кнопку і двері миттєво починають рухатися. Arduino в цей час отримує команду, перевіряє, чи справді можна відкрити двері (ліфт не рухається), і або виконує її, або ігнорує. Але користувач уже бачить результат — і для нього система працює миттєво.

Випадок, коли Arduino проігнорує команду, можливий лише в одній ситуації: якщо ліфт у русі. Але в цьому випадку Python-застосунок сам блокує кнопки дверей — перевірка `if is_moving: return` не дасть відправити команду взагалі. Тобто колізія неможлива, якщо команда пішла, то Arduino її виконає. Тому оптимістичний підхід тут повністю безпечний, жодного разу не доведеться відкочувати анімацію назад. У поточній версії проєкту Optimistic UI реалізовано тільки для дверей, але цей самий принцип можна поширити й

на інші частини інтерфейсу. Наприклад, при натисканні кнопки виклику поверху можна одразу показувати підсвічений поверх і додавати його в чергу, не чекаючи підтвердження від Arduino. У веб-розробці цей патерн використовується повсюдно, від лайків до відправки повідомлень у месенджерах. Мій проєкт показує, що він чудово працює і в десктопних SCADA-інтерфейсах, де проблема затримок стоїть так само гостро.

4.7 Висновки до розділу 4

У четвертому розділі було виконано проєктування програмно-апаратного комплексу та в результаті отримано позитивні результати а саме

Побудовано діаграму прецедентів, яка визначає трьох акторів системи (розробник, оператор, адміністратор) із чітким розмежуванням функцій. Розробник тестує алгоритми та завантажує прошивку, оператор керує ліфтом через готовий інтерфейс, адміністратор розгортає систему на новому комп'ютері.

Розроблено структурну схему системи, що відображає дворівневу архітектуру. Нижній рівень — віртуальний контролер Arduino Mega в SimulIDE з вісьмома кнопками, двома світлодіодами та 7-сегментним індикатором. Верхній рівень — SCADA-клієнт на Python/Tkinter; між ними транспортний рівень (com0com, що створює пару COM2 та COM3).

Складено повну таблицю розпиновки всіх 16 задіяних пінів Arduino Mega, яка є готовою інструкцією для відтворення схеми в SimulIDE а також описано алгоритм роботи контролера, побудований за схемою циклічного опитування. Основні функції (loop, AddList, ListTake, startMoving, openDoors, closeDoors) забезпечують повний цикл керування ліфтом. Опитування кнопок із програмним захист від брязкоту контактів 100 мс, обробку команд від Python, керування рухом за таймером 3 секунди між поверхами, автоматичне закриття дверей через 3 секунди після відкриття.

Розроблено алгоритм черги викликів. Черга коректно обробляє попутні виклики, ігнорує дублікати та виклики на поточний поверх. Додавання виклику реалізовано функцією `AddList()`, вилучення функцією `ListTake()`.

Спроектовано архітектуру Python-застосунку з чотирьох логічних модулів функції зв'язку з COM-портом (`connect_serial`, `read_serial`, `send_floor_command`, `send_door_command`), обробка вхідних команд (`process_message`), черга викликів (`floor_queue`, `queue_worker`, `process_queue`) та графічний інтерфейс із анімацією (`Canvas`, `move_cabin_to_floor`, `animate_doors_open/close`, `update_direction`).

Описано механізм Optimistic UI, який забезпечує миттєву реакцію інтерфейсу на керування дверима, за цим принципом анімація запускається до відправки команди контролеру, що усуває видимі затримки, спричинені обмеженою пропускнуою здатністю віртуального COM-порту.

5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

5.1 Налаштування середовища розробки та розгортання

Для функціонування програмно-апаратного комплексу на комп'ютері має бути встановлено три компоненти, а саме середовище симуляції SimulIDE, інтерпретатор Python із необхідними бібліотеками та драйвер віртуальних COM-портів com0com. Жоден із цих компонентів не потребує придбання ліцензії, усі вони розповсюджуються абсолютно безкоштовно.

SimulIDE завантажується з офіційного сайту розробника. Це портативна програма, вона не потребує інсталяції, достатньо розпакувати архів у будь-яку папку та запустити виконуваний файл. Також разом із симулятором в репозиторії постачається готовий проєкт Elevator3.simu, який містить повністю налаштовану схему Arduino Mega 2560 з підключеними кнопками, світлодіодами, 7-сегментним індикатором та компонентом SerialPort.

Python завантажується з офіційного сайту, для роботи застосунку достатньо стандартної версії Python 3.8 або новішої. Єдина зовнішня бібліотека, яка використовується в програмі це pySerial, вона необхідна для встановлення зв'язку між Python програмою та компонентом serial у Simulide. Встановлюється бібліотека стандартною командою `pip install pyserial`. Стосовно ще однієї основної бібліотеки Tkinter, на якій побудовано графічний інтерфейс, то вона входить до стандартної поставки Python і не потребує окремого встановлення. Усі інші модулі, які використовуються у програмі (threading, collections, time) також встановлюються по стандарту.

com0com — це безкоштовний драйвер віртуальних послідовних портів із відкритим вихідним кодом, він створює в системі пару з'єднаних між собою COM-портів, які працюють як прозорий канал передачі даних. Завантажити драйвер можна з репозиторію SourceForge. Для розгортання системи було створено спеціальний скрипт `setup_com_ports.bat`, який виконує пошук драйвера в папці проєкту або в системних директоріях, видаляє старі пари

портів і створює нову пару COM2 — COM3. Скрипт потребує прав адміністратора, оскільки працює на рівні драйверів операційної системи. Для перевірки успішності встановлення передбачено спеціальний режим /check, який виводить поточний стан портів без внесення змін до системи. Порядок розгортання системи на новому комп'ютері складається з трьох кроків.

Крок 1. Запустити `setup_com_ports.bat` від імені адміністратора для створення віртуальної пари COM-портів.

Крок 2. Запустити Python-застосунок `Elevator.py` — він відкриє графічний інтерфейс і почне очікувати підключення.

Крок 3. Відкрити проект `Elevator3.simu` у `SimulIDE`, завантажити скомпільований HEX-файл прошивки в `Arduino Mega` та натиснути кнопку запуску симуляції.

Після вищеперерахованих налаштувань система готова до роботи. Увесь процес розгортання займає менше декількох хвилини, без урахування часу на завантаження програм під час першого встановлення.

5.2 Програмна реалізація контролера

Програма для `Arduino Mega` написана мовою `C++` у додатку `Arduino IDE` і завантажується у віртуальний контролер у вигляді скомпільованого HEX-файлу, це важливо, бо інший формат компіляції плата `Arduino` не сприймає. Повний код налічує близько 250 рядків і складається з декількох логічних блоків, таких як оголошення пінів, змінних стану, функцій черги, функцій руху, функцій дверей, основного циклу `loop()` та допоміжних функцій, нижче розглянуто ключові фрагменти кожного блоку.

5.2.1 Оголошення пінів та констант

У першій частині програми визначаються всі задіяні піни мікроконтролера. Для зручності читання кожному піну присвоюється символічне ім'я через директиву `#define`. Кнопки виклику поділено на дві групи — зовнішні, поверхові пости та внутрішні, панель у кабіні. Окремо визначено піни для світлодіодів, 7-сегментного індикатора та кнопок керування дверима.

Для відображення цифр на 7-сегментному індикаторі використовується двовимірний масив `digitMap` [10], [7], у якому кожен рядок відповідає цифрі від 0 до 9, а кожен стовець одному з семи сегментів (A–G). Ліфт всього на 3 поверхи тому не було необхідності під'єднувати усі елементи індикатора, але для можливо розширення, ця задача була виконана. Одиниця означає сегмент світиться, нуль — сегмент погашений, наприклад, цифра «1» потребує увімкнення лише сегментів B і C, тому її рядок виглядає як `{0,1,1,0,0,0,0}`. Також додатково з пінами на самій платі можливо ознайомитися в таблиці 4.2.

5.2.2 Черга викликів

Черга реалізована як статичний масив `queue` [10] з лічильником `queueSize`, така реалізація обрана через свою простоту, для трьох поверхів глибини в 10 елементів більш ніж достатньо, а накладні витрати на роботу з масивом мінімальні.

Функція `AddList(int floor)` відповідає за додавання виклику в чергу. Перш ніж додати поверх, вона виконує три наступні перевірки. Якщо викликаний поверх збігається з поточним, то виклик ігнорується, оскільки ліфт уже знаходиться на цьому поверсі. Якщо такий поверх уже присутній у черзі, це перевіряється циклом по всіх елементах від 0 до `queueSize-1`, то виклик ігнорується як дублікат. Якщо черга заповнена `queueSize >= 10`, виклик також ігнорується. Після успішного додавання лічильник збільшується, а в `Serial`

відправляється підтвердження QUEUE ADD: N для синхронізації з Python-застосунком.

Функція ListTake() витягує перший елемент із черги за принципом FIFO (first in, first out), повертає значення queue[0], після чого зсуває всі наступні елементи на одну позицію вліво в циклі for $i = 0$. queueSize-2 і зменшує лічильник і якщо черга порожня повертає значення -1.

5.2.3 Керування рухом

Функція startMoving(int floor) запускає процес руху ліфта, спочатку вона перевіряє, чи не збігається цільовий поверх із поточним, якщо так, то негайно виходить, потім встановлює прапорець isMoving = true, запам'ятовує поточний час через millis() для відліку тривалості руху, вмикає світлодіод руху LED_MOVING та один зі світлодіодів напрямку залежно від співвідношення targetFloor і currentFloor. Після цих операцій відправляє в Serial повідомлення MOVING:X->Y, яке ініціює анімацію руху кабіни на стороні Python.

Тривалість руху між поверхами задана константою Move_Duration = 3000 мс. Це спрощена модель, у реальному ліфті час руху залежить від відстані між поверхами та швидкості двигуна, але для тестування алгоритмів керування трьома поверхами фіксована тривалість є достатньою.

5.2.4 Керування дверима

Функція openDoors() перевіряє дві умови, перша це ще не відкриті двері (!doorsOpen) і друга це ліфт не рухається (!isMoving). Якщо обидві умови виконано встановлює прапорець doorsOpen = true, відправляє в Serial повідомлення DOOR:OPEN і запускає таймер автоматичного закриття.

Функція closeDoors() перевіряє, що двері відкриті, після чого скидає прапорець і відправляє DOOR:CLOSED. Автоматичне закриття спрацьовує в основному циклі, якщо таймер активний (doorTimerActive), ліфт не рухається

і з моменту відкриття минуло 3 секунди викликається `closeDoors()` і таймер деактивується.

5.2.5 Основний цикл

Функція `loop()` виконується безперервно, поки симуляція активна. Кожна ітерація циклу послідовно виконує чотири блоки.

Перший блок — читання команд від Python, він викликається `Serial.available()` для перевірки наявності даних у вхідному буфері UART. Якщо дані є, вони читаються через `Serial.readStringUntil('\n')`, і отриманий рядок аналізується. Розпізнаються три типи команд: `FLOOR:N` (додавання поверху в чергу), `DOOR:OPEN` (відкриття дверей) та `DOOR:CLOSE` (закриття дверей).

Другий блок — опитування кнопок, усі вісім кнопок перевіряються на натискання. Для кожної кнопки реалізовано програмний захист від брязкоту контактів із використанням окремих змінних для відстеження часу останнього спрацьовування. Якщо з моменту попереднього натискання минуло менше 100 мс повторне натискання ігнорується. Кнопки виклику додають номер поверху в чергу через `AddList()`, кнопки дверей викликають `openDoors()` або `closeDoors()`.

Третій блок — логіка руху, якщо прапорець `isMoving` встановлено, перевіряється, чи минув потрібний час із моменту старту, якщо так ліфт вважається таким, що прибув на поверх та `currentFloor` оновлюється, прапорець скидається, світлодіоди гасяться, 7-сегментний індикатор оновлюється, відправляється `ARRIVED:N` і викликається `openDoors()`.

Четвертий блок — обробка черги, якщо ліфт стоїть, двері закриті, а черга не порожня викликається `ListTake()`. Якщо витягнутий поверх не збігається з поточним викликається `startMoving()`, а якщо збігається виклик пропускається, і цикл переходить до наступного елемента черги.

5.2.6 Допоміжні функції

Функція `displayDigit(int number)` приймає число від 0 до 9 і запалює відповідні сегменти 7-сегментного індикатора, у циклі `for i = 0..6` на кожен пін із масиву `segPins[i]` подається значення з таблиці `digitMap[number][i]`. Якщо число виходить за межі діапазону 0-9 — функція нічого не робить. Початкове відображення (цифра «1») встановлюється у функції `setup()` після налаштування всіх пінів індикатора на вихід.

5.3 Програмна реалізація SCADA-клієнта

SCADA-клієнт реалізований мовою Python із використанням бібліотеки Tkinter для графічного інтерфейсу та pySerial для зв'язку з контролером, повний код налічує близько 600 рядків. Нижче розглянуто ключові фрагменти, що реалізують основні функції системи, а саме зв'язок із контролером, обробку команд, анімацію руху та механізм Optimistic UI.

5.3.1 Підключення до COM-порту та читання даних

Для роботи з послідовним портом використовується бібліотека pySerial. На етапі запуску застосунок перевіряє доступність порту COM3 через `serial.tools.list_ports`, після чого відкриває його на швидкості 9600 біт, якщо порт недоступний користувач отримує зрозуміле повідомлення про помилку, а не просто аварійне завершення програми.

Читання даних із порту виконується у фоновому потоці `read_serial()`, це необхідно, оскільки блокувальне читання в головному потоці «заморозило» б графічний інтерфейс. Кожен отриманий рядок передається в головний потік через метод `window.after()`, який є стандартним способом міжпоточної взаємодії в Tkinter. Така конструкція забезпечує безпечне оновлення інтерфейсу, бібліотека Tkinter не є потокобезпечною, тому змінювати віджети можна лише з головного потоку.

5.3.2 Обробка вхідних команд

Центральним елементом логіки SCADA-клієнта є функція `process_message(msg)`, вона отримує текстовий рядок від Arduino, аналізує його та виконує відповідну дію. Кожен тип команди обробляється окремою гілкою:

~ `MOVING:X->Y` — контролер повідомив про початок руху, функція перевіряє, чи не є рух холостим (коли $X == Y$, наприклад, `3->3`), якщо ні, визначає напрямок, закриває двері, якщо вони були відкриті і запускає анімацію руху кабіни в окремому потоці;

~ `ARRIVED:N` — контролер повідомив про прибуття на поверх, надалі оновлюються всі елементи інтерфейсу, табло в шахті, табло на панелі керування, статусний рядок, після чого запускається анімація відкриття дверей, а через 1,5 секунди стрілка напрямку гаситься;

~ `DOOR:OPEN / DOOR:CLOSED` — контролер підтвердив зміну стану дверей, це запускає відповідну анімацію та оновлює текстовий статус на панелі керування;

~ `READY` — контролер успішно завантажився після запуску симуляції та переводить інтерфейс в робочий стан;

~ `FLOOR:N` (від Arduino) — оператор натиснув кнопку виклику в SimulIDE. Поверх додається у внутрішню чергу Python-застосунку, після чого викликається функція `process_queue()`.

5.3.3 Відправка команд із захистом від брязкоту контактів

Функція `send_floor_command(floor)` викликається при натисканні кнопок вибору поверху в Python-інтерфейсі. Вона містить трирівневий захист:

- ~ якщо викликаний поверх збігається з поточним команда не відправляється.

- ~ якщо з моменту попередньої відправки минуло менше 0,5 секунди команда ігнорується. Це програмний аналог апаратного захисту від брязкоту контактів, який запобігає випадковим подвійним клікам.

- ~ якщо такий поверх уже присутній у черзі дублікат не додається.

Такий захист гарантує, що черга не буде засмічена повторними викликами, а контролер не отримає кілька однакових команд поспіль через випадковий подвійний клік мишею.

5.3.4 Анімація руху кабіни

Функція `move_cabin_to_floor(floor)` реалізує плавне переміщення кабіни між поверхами. Цільова координата Y обчислюється за формулою $\text{SHAFT_HEIGHT} - \text{floor} * \text{FLOOR_HEIGHT}$. Рух розбивається на серію дрібних кроків — від 10 до 20 кадрів залежно від відстані, яку потрібно подолати.

Ключовим елементом плавності є функція пом'якшення `slow_animation(t)`, яка задає криву затухання $1 - (1-t)^3$, завдяки цій функції кабіна швидко набирає швидкість на початку руху і плавно гальмує перед зупинкою, це наближення до реального ліфта та графічне покращення. На відміну від рівномірного переміщення, яке виглядало б механічно та неприродно.

Під час руху відстежується проміжний поверх, який проїжджає кабіна. Щойно кабіна «перетинає» межу наступного поверху оновлюється табло, це створює ефект поступової зміни цифр, як у реальному ліфті.

Усі пов'язані з кабіною елементи наприклад двері, ручки, табло над кабіною переміщуються синхронно, для цього вони позначені спільним тегом

cabin_part, і в циклі анімації викликається `canvas.move()` для кожного з них із однаковим зсувом по осі Y.

5.3.5 Анімація дверей та Optimistic UI

Анімація дверей реалізована функціями `animate_doors_open()` та `animate_doors_close()`. Ліва та права стулки зсуваються на 35 пікселів за 15 кадрів із тим самим пом'якшенням `slow_animation(t)`. Після відкриття автоматично запускається таймер на 3 секунди, якщо за цей час двері не були закриті вручну, вони закриваються автоматично.

Особливістю керування дверима є застосування механізму Optimistic UI [15] у функції `send_door_command(action)`. На відміну від класичного підходу, де анімація запускається лише після отримання підтвердження від контролера, тут анімація стартує миттєво при натисканні кнопки. Команда DOOR:OPEN або DOOR:CLOSE відправляється паралельно, у фоновому режимі.

Такий підхід усуває видиму затримку між натисканням кнопки та початком руху дверей. Оскільки Python-застосунок блокує кнопки дверей під час руху ліфта (`if is_moving: return`), колізія неможлива, бо якщо команда пішла, то Arduino її гарантовано виконає.

5.3.6 Клієнтська черга викликів

Python-застосунок підтримує власну копію черги викликів на основі стандартної бібліотеки `collections.deque`. Це двостороння черга, яка дозволяє швидко додавати елементи в кінець (метод `append`) і витягувати з початку (метод `popleft`).

Обробка черги виконується у фоновому потоці `queue_worker()`. Функція послідовно витягує виклики з черги, перевіряє, чи не знаходиться ліфт уже на цільовому поверсі, і запускає анімацію руху. Якщо ліфт у момент обробки

черги вже рухається, виклик повертається на початок черги (метод `appendleft`), і обробка тимчасово припиняється до завершення поточного руху.

Для захисту від стану гонитви (`race condition`) використовується блокування `queue_lock`, тобто усі операції з чергою додавання, вилучення, перевірка виконуються тільки після захоплення цього блокування. Такий принцип гарантує, що два потоки не змінять чергу одночасно, що могло б призвести до втрати виклику або непередбачуваної поведінки ліфта.

5.4 Тестування протоколу обміну даними

Для перевірки коректності двостороннього обміну даними між віртуальним контролером (Arduino Mega в SimulIDE) та SCADA-клієнтом (Python/Tkinter) було проведено серію тестів. Метою тестування було підтвердити, що усі команди протоколу передаються без втрат в обох напрямках, Python-застосунок коректно розпізнає кожен тип повідомлення, Arduino правильно обробляє вхідні команди від Python, хибні або дубльовані команди ігноруються. Тестування проводилося при швидкості обміну 9600 біт із форматом кадру 8N1, результати фіксувалися в журналі подій Python-застосунку. Нижче приведено таблиці тестування, таблиця 5.1, а також таблиця 5.2 відповідність відправки та отримання команд. У таблиці 5.3 наведено результати тестування черги викликів.

Таблиця 5.1 — Тестування напрямку від Python до Arduino

№	Тестовий сценарій	Відправлена команда	Очікувана реакція Arduino	Фактичний результат
1	Виклик на поверх 2 із Python	FLOOR:2	Додавання поверху 2 у чергу, початок руху, відправка MOVING:1->2	Відповідає
2	Виклик на поверх 3 із Python	FLOOR:3	Додавання поверху 3 у чергу, початок руху після завершення попереднього	Відповідає
3	Виклик поточного поверху (ліфт на 1, виклик 1)	FLOOR:1	Команда ігнорується	Відповідає
4	Дублювання ви-клику (поверх 2 уже в черзі)	FLOOR:2 (по-вторно)	Дублікат ігнорується	Відповідає
5	Відкриття дверей (ліфт стоїть)	DOOR:OPEN	Двері відчиняються, відправка DOOR:OPEN	Відповідає
6	Закриття дверей (двері відкриті)	DOOR:CLOSE	Двері зачиняються, відправка DOOR:CLOSED	Відповідає
7	Відкриття дверей під час руху	DOOR:OPEN (під час MOVING)	Команда ігнорується (ліфт у русі)	Відповідає
8	Швидка послі-довність викликів (1, 2, 3)	FLOOR:1, FLOOR:2, FLOOR:3	Усі три додаються в чергу, обробляються послідовно	Відповідає

Таблиця 5.2 — Тестування напрямку від Arduino до Python

№	Тестовий сценарій	Отримана команда	Очікувана реакція Python	Фактичний результат
1	Запуск симуляції	READY	Інтерфейс переходить у робочий стан, статус «Ліфт готов»	Відповідає
2	Початковий стан	FLOOR:1	Кабіна на 1-му поверсі, табло показує «1»	Відповідає
3	Початок руху (1→2)	MOVING:1->2	Стрілка ▲, закриття дверей (якщо відкриті), анімація руху	Відповідає
4	Початок руху (3→1)	MOVING:3->1	Стрілка ▼, закриття дверей, анімація руху	Початок руху (3→1)
5	Прибуття на поверх 2	ARRIVED:2	Табло «2», статус «НА ПОВЕРСІ», анімація відкриття дверей	Прибуття на поверх 2
6	Відкриття дверей	DOOR:OPEN	Анімація відкриття, статус «ВІДЧИНЕНО»	Відкриття дверей
7	Закриття дверей	DOOR:CLOSED	Анімація закриття, статус «ЗАЧИНЕНО»	Закриття дверей
8	Холостий рух (1→1)	MOVING:1->1	Ігнорується (перевірка start == target)	Холостий рух (1→1)

Таблиця 5.3 — Тестування алгоритму черги

№	Тестовий сценарій	Послідовність дій	Очікуваний результат	Фактичний результат
1	Два послідовних виклики	Виклик 2, потім виклик 3	Ліфт їде на 2, потім на 3	Відповідає
2	Попутний виклик під час руху	Ліфт їде на 3, натиснуто 2	Ліфт зупиняється на 3, потім їде на 2	Відповідає
3	Виклик поточного поверху	Ліфт на 2, виклик 2	Виклик ігнорується	Відповідає
4	Дублювання виклику	Виклик 3 двічі поспіль	Другий виклик ігнорується	Відповідає
5	Виклик із Python під час руху	Ліфт їде, натиснуто кнопку в Python	Виклик додається в чергу	Відповідає
6	Виклик із SimulIDE під час руху	Ліфт їде, натиснуто кнопку в SimulIDE	Виклик додається в чергу	Відповідає

5.4.1 Висновки за результатами тестування

Проведене тестування підтвердило, що протокол обміну даними працює стабільно в обох напрямках. Усі шість типів команд (FLOOR, MOVING, ARRIVED, DOOR:OPEN, DOOR:CLOSED, READY) коректно розпізнаються та обробляються обома сторонами. Система правильно ігнорує некоректні команди, такі як виклик поточного поверху, дублікати, холостий рух. Черга викликів працює за принципом FIFO та коректно обробляє виклики, що надходять під час руху ліфта.

5.5 Тестування черги викликів

Алгоритм черги викликів є центральним елементом логіки керування ліфтом. Від коректної роботи алгоритму залежить, чи буде ліфт обслуговувати пасажирів у правильному порядку, чи не губитимуться виклики, чи не виникатиме дублікатів. Тестування черги проводилося окремо від тестування протоколу обміну, щоб ізолювати логіку обробки викликів від каналу передачі даних та виконувалося шляхом послідовного натискання кнопок виклику в обох інтерфейсах в SimulIDE (апаратні кнопки) та Python (графічні кнопки панелі керування). Кожен тестовий сценарій перевірявся в обох напрямках, коли виклики надходять із SimulIDE та коли із Python. Результати фіксувалися за допомогою журналу подій, у якому відображаються всі команди протоколу обміну. надалі наведена таблиця 5.4 з тестуванням алгоритму черги.

Таблиця 5.4 — Тестування алгоритму черги викликів

№	Тестовий сценарій	Початковий стан	Дії	Очікуваний результат	Фактичний результат
1	Два послідовних виклики	Ліфт на 1-му поверсі, двері закриті	Виклик 2, після прибуття на виклик 3	Ліфт їде на 2-й, відкриває двері, закриває, їде на 3-й	Відповідає
2	Три послідовних виклики	Ліфт на 1-му поверсі	Виклик 2, 3, 1	Ліфт об'їжджає поверхи в порядку надходження: 2 3 1	Відповідає
3	Виклик поточного поверху	Ліфт на 2-му поверсі	Виклик 2	Виклик ігнорується, ліфт залишається на місці	Відповідає

Кінець таблиці 5.4

4	Дублювання виклику	Ліфт на 1-му поверсі	Два виклики 3 поспіль	Перший додається в чергу, другий ігнорується	Відповідає
5	Виклик під час руху (SimulIDE)	Ліфт їде з 1-го на 3-й	Натиснуто кнопку 2 у SimulIDE	Виклик додається в чергу, ліфт зупиняється на 2-му, потім їде на 3-й	Відповідає
6	Виклик із двох джерел одночасно	Ліфт на 1-му поверсі	Python виклик 2, SimulIDE виклик 3	Обидва виклики додаються в чергу, обробляються послідовно	Відповідає
7	Переповнення черги	Ліфт на 1-му поверсі	4 викликів поспіль	дублікати не додаються	Відповідає
8	Черга після повного циклу	Ліфт на 1-му, черга 2, 3	Ліфт об'їхав 2 і 3, і залишився	Черга порожня, ліфт стоїть на третьому поверсі	Відповідає
9	Виклик протилежного напрямку під час руху	Ліфт їде з першого на третій поверх	Натиснуто кнопку 1	Виклик додається в чергу, ліфт доїжджає до третього, потім повертається на перший	Відповідає
10	Виклик під час руху (Python)	Ліфт їде з 1-го на 3-й	Натиснуто кнопку 2 у Python	Виклик додається в чергу, ліфт зупиняється на 2-му, потім їде на 3-й	Відповідає

5.5.1 Аналіз результатів

Тестування підтвердило, що черга викликів працює коректно в усіх основних сценаріях.

Послідовна обробка: виклики обробляються строго в порядку надходження (FIFO). Перший доданий — перший оброблений.

Захист від дублікатів: повторний виклик того самого поверху, який уже присутній у черзі, ігнорується. Це запобігає засміченню черги та подвійним зупинкам на одному поверсі.

Захист від поточного поверху: виклик поверху, на якому ліфт уже знаходиться, ігнорується — немає сенсу «їхати» на той самий поверх.

Робота під час руху: виклики, що надходять під час руху ліфта, не губляться, а коректно додаються в чергу. Якщо викликаний поверх знаходиться «по дорозі» — ліфт зупиниться на ньому.

Обмеження глибини черги: максимальна кількість викликів у черзі — 10. Цього достатньо для трьох поверхів із великим запасом. При спробі додати 11-й виклик він ігнорується, що запобігає переповненню статичного масиву.

Незалежність від джерела: черга працює однаково для викликів із SimulIDE (апаратні кнопки) та з Python (графічний інтерфейс). Обидва джерела використовують ту саму функцію `AddList()`.

5.5.2 Виявлені особливості

У ході тестування було виявлено, що поточна реалізація черги не враховує напрямок руху при виборі наступного виклику — обробка відбувається строго за принципом FIFO. Це означає, що в деяких сценаріях ліфт може проїхати повз потрібний поверх, якщо той був доданий у чергу пізніше. Для трьох поверхів це не створює суттєвих незручностей, але при масштабуванні системи на більшу кількість поверхів доцільно реалізувати алгоритм збору

попутних викликів із аналізом напрямку руху. Це може бути напрямком для подальшого вдосконалення системи.

5.6 Тестування інтерфейсу користувача

Графічний інтерфейс SCADA-клієнта є основним засобом взаємодії оператора із системою керування ліфтом. Від його плавності та відсутності видимих затримок залежить сприйняття системи як професійного інструменту, а не навчального макета. Тестування інтерфейсу проводилося за трьома критеріями: плавність анімації, миттєвість реакції на дії оператора та коректність відображення стану ліфта.

5.6.1 Плавність анімації руху кабіни

Анімація руху кабіни реалізована через серію проміжних кадрів із використанням функції пом'якшення `slow_animation(t)`. Для оцінки плавності руху кабіни спостерігався візуально при переміщенні між сусідніми поверхами (1-2, 2-3) та між крайніми поверхами (1-3, 3-1). оцінка плавності анімації в таблиці 5.5

Таблиця 5.5 — Тестування плавності анімації руху

№	Сценарій	Кількість кадрів	Візуальна оцінка	Наявність ривків
1	Рух 1-2 (один поверх)	10-12	Плавно, без ривків	Відсутні

Кінець таблиці 5.5

№	Сценарій	Кількість кадрів	Візуальна оцінка	Наявність ривків
2	Рух 2-3 (один поверх)	10-12	Плавно, без ривків	Відсутні
3	Рух 1-3 (два поверхи)	18-20	Плавно, із прискоренням на початку та гальмуванням у кінці	Відсутні
4	Рух 3-1 (два поверхи)	18-20	Плавно, із прискоренням на початку та гальмуванням у кінці	Відсутні
5	Зупинка на проміжному поверсі	10+10	Плавне гальмування на 2-му, потім плавний старт до 3-го	Відсутні

Функція $\text{slow_animation}(t) = 1 - (1-t)^3$ забезпечує природний характер руху, кабіна швидко набирає швидкість на початку (крива стрімко зростає) і плавно гальмує перед зупинкою (крива спадає), візуально це сприймається як рух реального механізму з інерцією.

5.6.2 Миттєвість реакції на керування дверима (Optimistic UI)

Для оцінки миттєвості реакції вимірювався суб'єктивний час між натисканням кнопки «Відчинити»/«Зачинити» та початком видимого руху стулків дверей. Тестування проводилося як при ввімкненій симуляції SimulIDE, так і при вимкненій для імітації відсутності зв'язку з контролером. Таблиця 5.6 з результатами показала, що система працює ефективно.

Таблиця 5.6 — Тестування миттєвості реакції (Optimistic UI)

№	Сценарій	Стан симуляції	Затримка до початку анімації	Суб'єктивна оцінка
1	Відкриття дверей	Увімкнена	Менше 50 мс	Миттєво
2	Закриття дверей	Увімкнена	Менше 50 мс	Миттєво
3	Відкриття дверей	Вимкнена	Менше 50 мс	Миттєво
4	Закриття дверей	Вимкнена	Менше 50 мс	Миттєво

Затримка менше 50 мс є непомітною для людського ока (поріг сприйняття становить близько 100 мс). Таким чином, механізм Optimistic UI повністю виконує свою функцію, оператор бачить миттєву реакцію незалежно від наявності зв'язку з контролером.

5.6.3 Коректність відображення стану ліфта

Перевірялася відповідність між станом, у якому перебуває контролер (Arduino), та тим, що відображається в інтерфейсі Python, сценарії наведено в таблиці 5.7.

Таблиця 5.7 — Тестування коректності відображення стану

№	Елемент інтерфейсу	Очікувана поведінка	Фактичний результат
1	Табло в шахті (cabin_number)	Змінюється при проїзді кожного поверху	Відповідає
2	Табло на панелі (panel_floor_label)	Синхронізоване з таблом в шахті	Відповідає

Кінець таблиці 5.7

№	Елемент інтерфейсу	Очікувана поведінка	Фактичний результат
3	Стрілка напрямку (direction_label)	▲ при русі вгору, ▼ при русі вниз, ■ при зупинці	Відповідає
4	Статус дверей (panel_door_label)	«ВІДЧИНЕНО» після DOOR:OPEN, «ЗАЧИНЕНО» після DOOR:CLOSED	Відповідає
5	Статус руху (panel_status_label)	«В РУСІ» під час MOVING, «НА ПОВЕРСІ» після ARRIVED	Відповідає
6	Журнал подій (log_text)	Кожна команда протоколу фіксується з міткою часу	Відповідає
7	Кнопки виклику під час руху	Активні, виклики додаються в чергу	Відповідає

5.6.4 Висновки за результатами тестування інтерфейсу

Проведене тестування підтвердило, що графічний інтерфейс SCADA-клієнта відповідає вимогам, сформульованим на етапі проєктування, тобо всі задачі виконуються.

Плавність анімації забезпечується функцією пом'якшення `slow_animation(t)` та достатньою кількістю проміжних кадрів (10-20 залежно від відстані). Ривки відсутні навіть при максимальній дистанції руху.

Миттєвість реакції досягається завдяки механізму Optimistic UI: анімація дверей запускається до відправки команди контролеру, що усуває затримку, спричинену проходженням даних через віртуальний COM-порт. Затримка менше 50 мс є непомітною для оператора.

Коректність відображення підтверджена синхронізацією всіх елементів інтерфейсу зі станом контролера. Усі зміни стану ліфта негайно відображаються на табло, стрілці напрямку, статусі дверей та в журналі подій.

Незалежність від симуляції є додатковою перевагою: інтерфейс зберігає працездатність навіть при вимкненому SimulIDE, що демонструє стійкість архітектури до відмови каналу зв'язку.

5.7 Висновки до розділу 5

У п'ятому розділі було описано процес реалізації та тестування програмно-апаратного комплексу. Описано процес налаштування середовища розробки та розгортання системи для Усіх трьох компонентів (SimulIDE, Python із бібліотеками, драйвер com0com). Також усі застосовані компоненти є безкоштовними та не потребують придбання ліцензій. Розгортання системи на новому комп'ютері виконується за допомогою скрипта `setup_com_ports.bat`, який автоматично створює віртуальну пару COM-портів, а повний цикл розгортання займає менше декількох хвилин.

Детально розглянуто програмну реалізацію контролера, код Arduino написаний мовою C++ і побудований за схемою циклічного опитування. Основні функції (`AddList`, `ListTake`, `startMoving`, `openDoors`, `closeDoors`) реалізують повний цикл керування ліфтом, опитування кнопок із програмним захистом від брязкоту контактів (100 мс), обробку команд від Python через Serial, керування рухом за таймером (3 секунди між поверхами), автоматичне закриття дверей (через 3 секунди після відкриття) та чергу викликів на основі статичного масиву.

Детально розглянуто програмну реалізацію SCADA-клієнта. Python-застосунок використовує бібліотеки `pySerial` для зв'язку з контролером та `Tkinter` для графічного інтерфейсу. Ключовими компонентами є: фоновий потік `read_serial()` для безперервного читання даних із COM-порту, функція `process_message()` для розбору вхідних команд, функція `move_cabin_to_floor()`

для плавної анімації руху кабіни з використанням функції пом'якшення `slow_animation(t)`, функції `animate_doors_open()` та `animate_doors_close()` для анімації дверей, а також клієнтська черга викликів на основі `deque` із потокобезпечним доступом через `queue_lock`.

Проведено тестування протоколу обміну даними. Підтверджено коректну передачу всіх шести типів команд (FLOOR, MOVING, ARRIVED, DOOR:OPEN, DOOR:CLOSED, READY) в обох напрямках. Система правильно ігнорує некоректні команди (виклик поточного поверху, дублікати, холостий рух). Затримка передачі найдовшої команди (11 байт) становить менше 12 мс при швидкості 9600 бод.

Проведено тестування алгоритму черги викликів. Підтверджено коректну роботу в 10 тестових сценаріях: послідовна обробка за принципом FIFO, захист від дублікатів, ігнорування виклику поточного поверху, обробка викликів під час руху, обмеження глибини черги (10 елементів), незалежність від джерела виклику (SimulIDE або Python). Виявлено, що поточна реалізація не враховує напрямку руху при виборі наступного виклику, що є напрямком для подальшого вдосконалення.

Проведено тестування інтерфейсу користувача. Підтверджено плавність анімації руху кабіни (10-20 кадрів залежно від відстані, відсутність ривків), миттєвість реакції на керування дверима (затримка менше 50 мс завдяки Optimistic UI), коректність відображення стану ліфта на всіх елементах інтерфейсу (табло, стрілка напрямку, статус дверей, журнал подій). Інтерфейс зберігає працездатність навіть при вимкненій симуляції SimulIDE, що демонструє стійкість архітектури до відмови каналу зв'язку.

Отримані результати підтверджують, що розроблений програмно-апаратний комплекс повністю відповідає функціональним вимогам, сформульованим на етапі проєктування, і готовий до використання як навчально-лабораторний стенд для тестування алгоритмів керування вбудованими системами.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено програмно-апаратний комплекс для симуляції роботи ліфта, який реалізує концепцію цифрового двійника Digital Twin, пристрою Інтернету речей. Система складається з віртуального контролера Arduino Mega 2560 у середовищі SimulIDE та SCADA-клієнта, реалізованого мовою Python із використанням бібліотеки Tkinter. Обмін даними між компонентами здійснюється за протоколом UART через віртуальний COM-порт, створений драйвером com0com.

Проведено аналіз предметної області, розглянуто особливості функціонування ліфта як об'єкта автоматизації, архітектуру SCADA-систем, концепцію IoT та цифрових двійників. Проаналізовано чотири платформи для симуляції мікроконтролерів таких як Tinkercad, Wokwi, Proteus, SimulIDE та обґрунтовано вибір SimulIDE як середовища з відкритою архітектурою, що дозволяє організувати двосторонній обмін даними із зовнішнім програмним забезпеченням через стандартний протокол UART.

Спроектowano архітектуру розподіленої системи керування, таким чином система побудована за дворівневою схемою, нижній рівень віртуальний контролер Arduino Mega в SimulIDE 8 кнопок, 2 світлодіоди, 7-сегментний індикатор, компонент SerialPort, а верхній рівень це SCADA-клієнт на Python/Tkinter графічний інтерфейс із плавною анімацією, панель керування, журнал подій. Для транспортного рівня використовувався драйвер com0com, що створює пару між COM2 та COM3. Така архітектура відповідає принципам побудови реальних промислових SCADA-систем, де контролер та інтерфейс оператора фізично розділені.

Розроблено протокол обміну даними на основі текстових команд. Протокол забезпечує двосторонню синхронізацію стану між контролером та SCADA-клієнтом. Швидкість передачі 9600 біт гарантує доставку найдовшої команди 11 байт менш ніж за 12 мс, що є достатнім для системи керування ліфтом.

Реалізовано алгоритм черги викликів на стороні контролера, вона побудована на основі статичного масиву з дисципліною FIFO. Функції алгоритму забезпечують додавання викликів із перевіркою на дублікати та поточний поверх, а також послідовне вилучення викликів для обробки. Алгоритм протестовано в 10 сценаріях, включаючи попутні виклики під час руху, виклики з різних джерел SimulIDE та Python, захист від переповнення черги.

Створено SCADA-клієнт мовою Python із модульною архітектурою, функції зв'язку з COM-портом, обробка вхідних команд, клієнтська черга викликів на основі deque із потокобезпечним доступом, графічний інтерфейс із плавною анімацією руху кабіни та дверей. Для плавності анімації застосовано функцію пом'якшення, яка забезпечує природний характер руху з прискоренням на початку та гальмуванням у кінці.

Реалізовано механізм Optimistic UI для керування дверима. Анімація відкриття або закриття дверей запускається миттєво при натисканні кнопки, не чекаючи підтвердження від контролера, цей принцип усуває видиму затримку, приблизно 50 мс, спричинену проходженням даних через віртуальний COM-порт. Показано, що цей патерн, традиційно застосовуваний у веб-розробці, може бути ефективно використаний у десктопних SCADA-інтерфейсах для вбудованих систем.

Розроблено скрипт автоматичного розгортання, який виконує пошук драйвера com0com у портативній папці проєкту або в системних директоріях, видаляє старі пари портів і створює нову пару між COM2 та COM3. Скрипт підтримує безпечний режим перевірки (/check) і скорочує час розгортання системи на новому комп'ютері до декількох хвилини.

Проведено комплексне тестування системи, а також підтверджено коректну роботу протоколу обміну з 16 тестових сценаріїв, алгоритму черги з 10 сценаріїв, інтерфейсу користувача, де перевірялась плавність анімації, миттєвість реакції, коректність відображення стану.

Практична цінність роботи полягає у створенні повнофункціонального програмно-апаратного комплексу, який дозволяє тестувати алгоритми керува-

ння вбудованими системами без використання фізичного обладнання. Стенд не потребує придбання ліцензій, через факт безкоштовності усіх використаних компонентів, розгортається на будь-якому комп'ютері під керуванням Windows за допомогою автоматичного скрипта та скорочує час одного циклу налагодження алгоритму з 10-15 хвилин на фізичному стенді до 1-2 хвилин на віртуальній моделі. Система може бути використана для проведення тестів базових функцій ліфта.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What Is a Digital Twin? Definition, Types, and Uses : Coursera звернення: 19.05.2026. URL: <https://www.coursera.org/articles/digital-twin> (дата звернення: 19.05.2026).
2. Internet of Things (IoT): A vision, architectural elements, and future directions / J. Gubbi et al. *Future Generation Computer Systems*. – 2013. – Vol. 29, No. 7. – P. 1645-1660.
3. Arduino Mega 2560 Rev3 : офіційний сайт. URL: <https://store.arduino.cc/products/arduino-mega-2560-rev3> (дата звернення: 18.05.2026).
4. FIFO (First In First Out) Approach in Data Structures. GeeksforGeeks : technical article. URL: <https://www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-data-structures/> (дата звернення: 20.05.2026).
5. SCADA System: What Is It? *Inductive Automation*. URL: <https://inductiveautomation.com/resources/article/what-is-scada> (дата звернення: 20.05.2026).
6. Диспетчерська система «ОДС Промінь» : офіційний сайт. *Виробничо-конструкторська фірма «Промінь»*. URL: <https://www.promin.in.ua/> (дата звернення: 18.05.2026).
7. Tkinter — Python Interface to Tcl/Tk : офіційна документація. URL: <https://docs.python.org/3/library/tkinter.html> (дата звернення: 18.05.2026).
8. SimulIDE — Real Time Electronic Circuit Simulator : документація. URL: <https://simulide.blogspot.com/p/documentation.html> (дата звернення: 18.05.2026).
9. com0com: Null-modem emulator : SourceForge. URL: <https://sourceforge.net/projects/com0com/> (дата звернення: 18.05.2026).
10. SIMATIC WinCC (TIA Portal) Runtime. SiePortal — Siemens Deutschland. URL: <https://sieportal.siemens.com/de-at/products-services/10095298?tree=CatalogTree#productVariants> (дата звернення: 18.05.2026).

11. Tinkercad : офіційний сайт. URL: <https://www.tinkercad.com/> (дата звернення: 18.05.2026).
12. Wokwi : офіційний сайт. URL: <https://wokwi.com/> (дата звернення: 18.05.2026).
13. Labcenter Electronics. Proteus Design Suite : офіційний сайт. URL: <https://www.labcenter.com/> (дата звернення: 18.05.2026).
14. pySerial Documentation : readthedocs. URL: <https://pyserial.readthedocs.io/en/latest/> (дата звернення: 18.05.2026).
15. Optimistic UI: UI/UX pattern for faster apps. *Apollo GraphQL Blog* : technical article. URL: <https://www.apollographql.com/blog/ux/optimistic-ui/> (дата звернення: 18.05.2026).
16. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1540 p.
17. Monk S. Programming Arduino: Getting Started with Sketches. 2nd ed. New York : McGraw-Hill Education, 2016. 176 p.

ДОДАТОК А

Текст програми для плати Arduino

A.1 Код Python застосунку

```
import serial

import serial.tools.list_ports

import tkinter as tk

from tkinter import messagebox

import time

import threading

from collections import deque


# Налаштування

COM_PORT = 'COM3'

BAUD_RATE = 9600


FLOOR_HEIGHT = 150

FLOOR_COUNT = 3

SHAFT_WIDTH = 250

SHAFT_HEIGHT = FLOOR_HEIGHT * FLOOR_COUNT


CABIN_WIDTH = 110

CABIN_HEIGHT = 95


# Стан
```

```
current_floor = 1

is_moving = False

doors_open = False

move_direction = None


door_left = None

door_right = None

door_animation_running = False


# Черга виклику

floor_queue = deque()

processing_queue = False

queue_lock = threading.Lock()


# Захист від дребезгу

last_command_time = 0

COMMAND_PAUSE = 0.5 # секунд


# Serial

ser = None

ui_ready = False
```

```
# ФУНКЦІЇ АНІМАЦІЇ
```

```
def slow_animation(t):
```

```
    return 1 - pow(1 - t, 3)
```

```
def log_message(msg):
```

```
    if ui_ready:
```

```
        log_text.insert(tk.END, f" {msg}\n")
```

```
        log_text.see(tk.END)
```

```
    print(f"[LOG] {msg}")
```

```
def update_direction(direction):
```

```
    try:
```

```
        if direction == "UP":
```

```
            direction_label.config(text="▲", fg="#e74c3c")
```

```
        elif direction == "DOWN":
```

```
            direction_label.config(text="▼", fg="#2ecc71")
```

```
        else:
```

```
            direction_label.config(text="■", fg="#95a5a6")
```

```
    except:
```

pass

```
def connect_serial():
```

```
    global ser
```

```
    try:
```

```
        ser = serial.Serial(COM_PORT, BAUD_RATE, timeout=1)
```

```
        time.sleep(1)
```

```
        log_message(f"✓ Підключено до {COM_PORT}")
```

```
        window.after(0, lambda: status_label.config(text=" ліфт готов", fg="#2ecc71"))
```

```
        return True
```

```
    except Exception as e:
```

```
        log_message(f"✗ Помилка підключення: {e}")
```

```
        window.after(0, lambda: status_label.config(text=" Нема зв'язку", fg="#e74c3c"))
```

```
        return False
```

```
def read_serial():
```

```
    while True:
```

```
        if ser and ser.is_open:
```

```
            try:
```

```
                if ser.in_waiting > 0:
```



```

    if start == target:

        log_message(f"△Ігноруємо холостий ход: {start}->{target}")

        return

    log_message(f"🐛 Arduino повідомив про рух: {start}->{target}")

    if target > start:

        update_direction("UP")

    else:

        update_direction("DOWN")

    if doors_open:

        log_message("🚪 Зачиняємо двері перед рухом")

        window.after(0, animate_doors_close)

        time.sleep(1.2)

        threading.Thread(target=move_cabin_to_floor,            args=(target,),
daemon=True).start()

    except:

        pass

elif msg.startswith("ARRIVED:"):

```

```
try:
```

```
    floor = int(msg.split(":")[1])
```

```
    current_floor = floor
```

```
    floor_label.config(text=f"🚪 Поверх {floor}")
```

```
    canvas.itemconfig(cabin_number, text=str(floor))
```

```
    panel_floor_label.config(text=str(floor))
```

```
    panel_status_label.config(text="На поверси", fg="#3498db")
```

```
    threading.Thread(target=animate_doors_open, daemon=True).start()
```

```
def delayed_stop():
```

```
    time.sleep(1.5)
```

```
    update_direction("STOP")
```

```
    threading.Thread(target=delayed_stop, daemon=True).start()
```

```
except:
```

```
    pass
```

```
elif msg.startswith("FLOOR:"):

```

```
    try:
```

```
        floor = int(msg.split(":")[1])
```

```
log_message(f"📞 Отриманно виклик {floor} від Arduino")
```

```
# перевірка на додавання вже існуючого в черзі поверха
```

```
if floor == current_floor:
```

```
    log_message(f"⚠️ Ігноруємо виклик на поточний поверх {floor}")
```

```
    return
```

```
with queue_lock:
```

```
    if floor not in floor_queue:
```

```
        floor_queue.append(floor)
```

```
        log_message(f"👉 Додано у чергу: {floor}")
```

```
    else:
```

```
        log_message(f"🚫 Поверх {floor} вже у черзі")
```

```
process_queue()
```

```
except:
```

```
    pass
```

```
elif msg == "READY":
```

```
    status_label.config(text="Ліфт готов", fg="#2ecc71")
```

```
    update_direction("STOP")
```

```
elif msg == "DOOR:OPEN":
```

```
    if not doors_open and not is_moving:
```

```
        threading.Thread(target=animate_doors_open, daemon=True).start()
```

```
        panel_door_label.config(text="Відчинено", fg="#2ecc71")
```

```
elif msg == "DOOR:CLOSED":
```

```
    if doors_open:
```

```
        threading.Thread(target=animate_doors_close, daemon=True).start()
```

```
        panel_door_label.config(text="Зачинено", fg="#e74c3c")
```

```
def process_queue():
```

```
    global processing_queue
```

```
    if processing_queue:
```

```
        return
```

```
    threading.Thread(target=queue_worker, daemon=True).start()
```

```
def queue_worker():
```

```
global processing_queue, current_floor, is_moving
```

```
with queue_lock:
```

```
    if processing_queue:
```

```
        return
```

```
    processing_queue = True
```

```
try:
```

```
    while True:
```

```
        with queue_lock:
```

```
            if not floor_queue:
```

```
                break
```

```
            target = floor_queue.popleft()
```

```
            if target == current_floor:
```

```
                log_message(f"🔓 Пропуск поверху {target} (вже на поверху)")
```

```
                continue
```

```
# 🔥 ЗАХИСТ: якщо ліфт уже в русі — повернути виклик у чергу й вийти
```

```
if is_moving:
```

```
    log_message(f"Ліфт уже в русі — повертаю виклик {target} у чергу")
```

```
    with queue_lock:
```

```
floor_queue.appendleft(target)
```

```
break
```

```
log_message(f"✈ Запуск руху на поверх {target}")
```

```
if target > current_floor:
```

```
    update_direction("UP")
```

```
elif target < current_floor:
```

```
    update_direction("DOWN")
```

```
if doors_open:
```

```
    window.after(0, animate_doors_close)
```

```
    time.sleep(1.5)
```

```
move_cabin_to_floor(target)
```

```
while is_moving:
```

```
    time.sleep(0.1)
```

```
time.sleep(4)
```

```
finally:
```

```

with queue_lock:

    processing_queue = False

```

```

with queue_lock:

    if floor_queue:

        window.after(100, process_queue)

```

```

def draw_doors():

    global door_left, door_right

    cabin_coords = canvas.coords(cabin_body)

    cabin_left = cabin_coords[0]

    cabin_right = cabin_coords[2]

    cabin_top = cabin_coords[1]

    cabin_bottom = cabin_coords[3]

    door_width = (CABIN_WIDTH - 25) // 2

    door_left = canvas.create_rectangle(

        cabin_left + 12, cabin_top + 8,

        cabin_left + 12 + door_width, cabin_bottom - 8,

```

```

        fill="#bdc3c7", outline="#95a5a6", width=2,

        tags="cabin_part"

    )

    door_right = canvas.create_rectangle(

        cabin_right - 12 - door_width, cabin_top + 8,

        cabin_right - 12, cabin_bottom - 8,

        fill="#bdc3c7", outline="#95a5a6", width=2,

        tags="cabin_part"

    )

    canvas.create_rectangle(

        cabin_left + 12 + door_width - 5, cabin_top + 35,

        cabin_left + 12 + door_width - 2, cabin_top + 50,

        fill="#7f8c8d", outline="", tags=("door_handle", "cabin_part")

    )

    canvas.create_rectangle(

        cabin_right - 12 - door_width + 2, cabin_top + 35,

        cabin_right - 12 - door_width + 5, cabin_top + 50,

        fill="#7f8c8d", outline="", tags=("door_handle", "cabin_part")

    )

```

```
def animate_doors_open():  
  
    global doors_open, door_animation_running  
  
    if door_animation_running or doors_open:  
  
        return  
  
    door_animation_running = True  
  
  
    cabin_coords = canvas.coords(cabin_body)  
  
    cabin_left = cabin_coords[0]  
  
    cabin_right = cabin_coords[2]  
  
    cabin_top = cabin_coords[1]  
  
    cabin_bottom = cabin_coords[3]  
  
  
    door_width = (CABIN_WIDTH - 25) // 2  
  
  
    steps = 15  
  
    for i in range(steps + 1):  
  
        t = i / steps  
  
        eased = slow_animation(t)  
  
        offset = int(35 * eased)
```

```

canvas.coords(door_left,

               cabin_left + 12 - offset, cabin_top + 8,

               cabin_left + 12 + door_width - offset, cabin_bottom - 8)

canvas.coords(door_right,

               cabin_right - 12 - door_width + offset, cabin_top + 8,

               cabin_right - 12 + offset, cabin_bottom - 8)

for handle in canvas.find_withtag("door_handle"):

    if canvas.coords(handle)[0] < cabin_left + CABIN_WIDTH//2:

        canvas.coords(handle,

                       cabin_left + 12 + door_width - 5 - offset, cabin_top + 35,

                       cabin_left + 12 + door_width - 2 - offset, cabin_top + 50)

    else:

        canvas.coords(handle,

                       cabin_right - 12 - door_width + 2 + offset, cabin_top + 35,

                       cabin_right - 12 - door_width + 5 + offset, cabin_top + 50)

window.update()

time.sleep(0.015)

doors_open = True

```

```
door_animation_running = False
```

```
status_label.config(text=" 🚪 Двері відчинено", fg="#2ecc71")
```

```
panel_door_label.config(text="Відкрито", fg="#2ecc71")
```

```
def auto_close():
```

```
    time.sleep(3)
```

```
    if doors_open:
```

```
        window.after(0, animate_doors_close)
```

```
    threading.Thread(target=auto_close, daemon=True).start()
```

```
def animate_doors_close():
```

```
    global doors_open, door_animation_running
```

```
    if door_animation_running or not doors_open:
```

```
        return
```

```
    door_animation_running = True
```

```
cabin_coords = canvas.coords(cabin_body)
```

```
cabin_left = cabin_coords[0]
```

```
cabin_right = cabin_coords[2]
```

```
cabin_top = cabin_coords[1]
```

```
cabin_bottom = cabin_coords[3]
```

```
door_width = (CABIN_WIDTH - 25) // 2
```

```
steps = 15
```

```
for i in range(steps + 1):
```

```
    t = i / steps
```

```
    eased = slow_animation(t)
```

```
    offset = 35 - int(35 * eased)
```

```
    canvas.coords(door_left,
```

```
                    cabin_left + 12 - offset, cabin_top + 8,
```

```
                    cabin_left + 12 + door_width - offset, cabin_bottom - 8)
```

```
    canvas.coords(door_right,
```

```
                    cabin_right - 12 - door_width + offset, cabin_top + 8,
```

```
                    cabin_right - 12 + offset, cabin_bottom - 8)
```

```
for handle in canvas.find_withtag("door_handle"):
```

```
    if canvas.coords(handle)[0] < cabin_left + CABIN_WIDTH//2:
```

```
        canvas.coords(handle,
```

```
                        cabin_left + 12 + door_width - 5 - offset, cabin_top + 35,
```

```
                        cabin_left + 12 + door_width - 2 - offset, cabin_top + 50)
```

```
else:
```

```
    canvas.coords(handle,
```

```
        cabin_right - 12 - door_width + 2 + offset, cabin_top + 35,
```

```
        cabin_right - 12 - door_width + 5 + offset, cabin_top + 50)
```

```
    window.update()
```

```
    time.sleep(0.015)
```

```
doors_open = False
```

```
door_animation_running = False
```

```
status_label.config(text="■ Двері зачинені", fg="#e74c3c")
```

```
panel_door_label.config(text="Зачинено", fg="#e74c3c")
```

```
def move_cabin_to_floor(floor):
```

```
    global current_floor, is_moving, move_direction
```

```
    if is_moving:
```

```
        return
```

```
    is_moving = True
```

```
    start_floor = current_floor
```

```
status_label.config(text=f"↑ Pyx на {floor} поверх...", fg="#f39c12")
```

```
panel_status_label.config(text="B pyci", fg="#f39c12")
```

```
step_direction = 1 if floor > current_floor else -1
```

```
total_floors = abs(floor - current_floor)
```

```
target_y = SHAFT_HEIGHT - (floor * FLOOR_HEIGHT) + (FLOOR_HEIGHT -
CABIN_HEIGHT) // 2
```

```
current_y = canvas.coords(cabin_body)[1]
```

```
distance = abs(target_y - current_y)
```

```
steps = max(int(distance / 2), 10)
```

```
last_displayed_floor = current_floor
```

```
for i in range(steps + 1):
```

```
    t = i / steps
```

```
    eased = slow_animation(t)
```

```
    new_y = current_y + (target_y - current_y) * eased
```

```
    progress = i / steps
```

```
    intermediate_floor = start_floor + step_direction * round(progress * total_floors)
```

```

intermediate_floor = max(1, min(3, intermediate_floor))

if intermediate_floor != last_displayed_floor:

    last_displayed_floor = intermediate_floor

    canvas.itemconfig(cabin_number, text=str(intermediate_floor))

    panel_floor_label.config(text=str(intermediate_floor))

    log_message(f" 🚗 Проїзжаємо поверх {intermediate_floor}")


cabin_coords = canvas.coords(cabin_body)

offset_y = new_y - cabin_coords[1]


canvas.move(cabin_body, 0, offset_y)

canvas.move(cabin_light, 0, offset_y)

canvas.move(cabin_display, 0, offset_y)

canvas.move(cabin_number, 0, offset_y)


if door_left:

    canvas.move(door_left, 0, offset_y)

if door_right:

    canvas.move(door_right, 0, offset_y)


for handle in canvas.find_withtag("door_handle"):

```

```
canvas.move(handle, 0, offset_y)
```

```
window.update()
```

```
time.sleep(0.015)
```

```
current_floor = floor
```

```
is_moving = False
```

```
move_direction = None
```

```
floor_label.config(text=f"🚪 Поверх {current_floor}")
```

```
status_label.config(text=f"✔ Прибыв на {current_floor} поверх", fg="#3498db")
```

```
canvas.itemconfig(cabin_number, text=str(current_floor))
```

```
panel_floor_label.config(text=str(current_floor))
```

```
panel_status_label.config(text="На поверхности", fg="#3498db")
```

```

window.after(100, lambda: threading.Thread(target=animate_doors_open,
daemon=True).start())

```

```
def send_floor_command(floor):
```

```
    global floor_queue, last_command_time
```

```
# Захист від виклику від самого себе
```

```
if floor == current_floor:
```

```
    log_message(f"⚠ Ліфт вже на поверху {floor}")
```

```
    return
```

```
# Захист від частих натискач
```

```
current_time = time.time()
```

```
if current_time - last_command_time < COMMAND_PAUSE:
```

```
    log_message(f"⌚ Дуже часті виклики, черга не додана")
```

```
    return
```

```
last_command_time = current_time
```

```
with queue_lock:
```

```
    if floor not in floor_queue:
```

```
        floor_queue.append(floor)
```

```
        log_message(f"👇 Додано у чергу: {floor}")
```

```
    else:
```

```
        log_message(f"🚫 Поверх {floor} вже у черзі")
```

```
if ser and ser.is_open:
```

```
    try:
```

```
        cmd = f"FLOOR: {floor}\n"
```

```
ser.write(cmd.encode())
```

```
ser.flush()
```

```
log_message(f"👉 Відправлено в Arduino: FLOOR:{floor}")
```

```
except:
```

```
    log_message("✗ Помилка відправки команди")
```

```
process_queue()
```

```
def send_door_command(action):
```

```
    if is_moving:
```

```
        log_message("⚠ Ліфт у русі, двері заблоковано ")
```

```
    return
```

```
# Запускаємо анімацію не чекаючи Arduino
```

```
if action == "OPEN":
```

```
    if not doors_open:
```

```
        threading.Thread(target=animate_doors_open, daemon=True).start()
```

```
elif action == "CLOSE":
```

```
    if doors_open:
```

```
        threading.Thread(target=animate_doors_close, daemon=True).start()
```

Відправляємо команду Arduino для синхронизації

if ser and ser.is_open:

try:

cmd = f"DOOR: {action}\n"

ser.write(cmd.encode())

ser.flush()

log_message(f"👉 Відправлено: DOOR: {action}")

except:

log_message("✖ ПОмилка відправки команди")

else:

log_message("⚠ нема зв'язку с контроллером")

def on_closing():

if ser and ser.is_open:

ser.close()

window.destroy()

ГРАФІЧНИЙ ІНТЕРФЕЙС

window = tk.Tk()

window.title("Візуальна модель ліфту")

```

window.geometry("700x750")

window.configure(bg="#1a1a2e")

window.protocol("WM_DELETE_WINDOW", on_closing)


main_frame = tk.Frame(window, bg="#1a1a2e")

main_frame.pack(fill="both", expand=True, padx=10, pady=10)


# IIIAXTA

left_frame = tk.Frame(main_frame, bg="#1a1a2e")

left_frame.pack(side="left", padx=10)


title = tk.Label(left_frame, text=" IIIAXTA ЛiΦTA",

                 font=("Arial", 12, "bold"), fg="#e0e0e0", bg="#1a1a2e")

title.pack(pady=5)


canvas = tk.Canvas(left_frame, width=SHAFT_WIDTH, height=SHAFT_HEIGHT,

                 bg="#1a1a2e", highlightthickness=0)

canvas.pack()


canvas.create_rectangle(0, 0, SHAFT_WIDTH, SHAFT_HEIGHT, fill="#2c3e50",
outline="#1a252f", width=2)


for i in range(FLOOR_COUNT):

```

```

y = SHAFT_HEIGHT - (i + 1) * FLOOR_HEIGHT

canvas.create_line(0, y, SHAFT_WIDTH, y, fill="#7f8c8d", width=1)

canvas.create_text(15, y + FLOOR_HEIGHT//2, text=str(i+1),

                  font=("Arial", 11, "bold"), fill="#95a5a6")

door_y = y + (FLOOR_HEIGHT - 80) // 2

canvas.create_rectangle(SHAFT_WIDTH//2 - 40, door_y,

                      SHAFT_WIDTH//2 + 40, door_y + 80,

                      fill="#1a252f", outline="#7f8c8d", width=1)


cabin_x = (SHAFT_WIDTH - CABIN_WIDTH) // 2

cabin_y = SHAFT_HEIGHT - FLOOR_HEIGHT + (FLOOR_HEIGHT -
CABIN_HEIGHT) // 2


cabin_body = canvas.create_rectangle(

    cabin_x, cabin_y,

    cabin_x + CABIN_WIDTH, cabin_y + CABIN_HEIGHT,

    fill="#3498db", outline="#2980b9", width=2

)


cabin_light = canvas.create_rectangle(

    cabin_x + 3, cabin_y - 2,

    cabin_x + CABIN_WIDTH - 3, cabin_y,

```

```

        fill="#5dade2", outline=""
    )

    cabin_display = canvas.create_rectangle(
        cabin_x + CABIN_WIDTH//2 - 20, cabin_y - 18,
        cabin_x + CABIN_WIDTH//2 + 20, cabin_y - 3,
        fill="#2c3e50", outline="#1a252f", width=1
    )

    cabin_number = canvas.create_text(
        cabin_x + CABIN_WIDTH//2, cabin_y - 10,
        text="1", font=("Arial", 14, "bold"), fill="#2ecc71"
    )

    draw_doors()

# Панель керування

right_frame = tk.Frame(main_frame, bg="#16213e", width=250, height=450)

right_frame.pack(side="right", padx=10, fill="y")

right_frame.pack_propagate(False)

panel_title = tk.Label(right_frame, text="🚪 ПАНЕЛЬ В КАБІНІ",

```

```

        font=("Arial", 12, "bold"), fg="#e0e0e0", bg="#16213e")

panel_title.pack(pady=15)

panel_display_frame = tk.Frame(right_frame, bg="#0f3460", relief="ridge", bd=2)

panel_display_frame.pack(pady=10)

tk.Label(panel_display_frame, text="ПОТОЧНИЙ ПОВЕРХ", font=("Arial", 9),
        fg="#bdc3c7", bg="#0f3460").pack(pady=(5, 0))

panel_floor_label = tk.Label(panel_display_frame, text="1",
        font=("Arial", 36, "bold"), fg="#2ecc71", bg="#0f3460")

panel_floor_label.pack(pady=(0, 5))

direction_label = tk.Label(panel_display_frame, text="■",
        font=("Arial", 20, "bold"), fg="#95a5a6", bg="#0f3460")

direction_label.pack(pady=(0, 10))

panel_status_label = tk.Label(right_frame, text="НА ПОВЕРХУ",
        font=("Arial", 10, "bold"), fg="#3498db", bg="#16213e")

panel_status_label.pack(pady=5)

panel_door_label = tk.Label(right_frame, text="ЗАЧИНЕНО",

```

```
font=("Arial", 10), fg="#e74c3c", bg="#16213e")
```

```
panel_door_label.pack(pady=5)
```

```
tk.Frame(right_frame, height=2, bg="#0f3460").pack(fill="x", padx=20, pady=15)
```

```
tk.Label(right_frame, text="ВИБІР ПОВЕРХА", font=("Arial", 10, "bold"),
```

```
fg="#e0e0e0", bg="#16213e").pack()
```

```
buttons_frame = tk.Frame(right_frame, bg="#16213e")
```

```
buttons_frame.pack(pady=10)
```

```
btn1 = tk.Button(buttons_frame, text="1", command=lambda: send_floor_command(1),
```

```
bg="#3498db", fg="white", font=("Arial", 14, "bold"),
```

```
width=4, height=2, relief="raised")
```

```
btn1.grid(row=0, column=0, padx=5, pady=5)
```

```
btn2 = tk.Button(buttons_frame, text="2", command=lambda: send_floor_command(2),
```

```
bg="#3498db", fg="white", font=("Arial", 14, "bold"),
```

```
width=4, height=2, relief="raised")
```

```
btn2.grid(row=0, column=1, padx=5, pady=5)
```

```
btn3 = tk.Button(buttons_frame, text="3", command=lambda: send_floor_command(3),
```

```

        bg="#3498db", fg="white", font=("Arial", 14, "bold"),

        width=4, height=2, relief="raised")

btn3.grid(row=0, column=2, padx=5, pady=5)


door_frame = tk.Frame(right_frame, bg="#16213e")

door_frame.pack(pady=15)


    btn_open = tk.Button(door_frame, text="Відчинити двері", command=lambda:
send_door_command("OPEN"),

        bg="#2ecc71", fg="white", font=("Arial", 10, "bold"),

        width=13, height=2)

    btn_open.grid(row=0, column=0, padx=5)


    btn_close = tk.Button(door_frame, text="Зачинити двері", command=lambda:
send_door_command("CLOSE"),

        bg="#e74c3c", fg="white", font=("Arial", 10, "bold"),

        width=13, height=2)

    btn_close.grid(row=0, column=1, padx=5)


# панель

bottom_frame = tk.Frame(window, bg="#1a1a2e")

bottom_frame.pack(fill="x", padx=20, pady=10)

```

```

status_label = tk.Label(bottom_frame, text="Підключено...",
                        font=("Arial", 10), fg="#e0e0e0", bg="#0f3460", padx=10, pady=5)

status_label.pack(fill="x")


floor_label = tk.Label(bottom_frame, text="🏠 ПОБЕРХ 1",
                        font=("Arial", 11, "bold"), fg="#e0e0e0", bg="#1a1a2e")

floor_label.pack(pady=5)


log_frame = tk.Frame(window, bg="#16213e")

log_frame.pack(fill="both", expand=True, padx=20, pady=10)


log_label = tk.Label(log_frame, text="📖 ЖУРНАЛ ПОДІЙ",
                        font=("Arial", 9, "bold"), fg="#e0e0e0", bg="#16213e")

log_label.pack(anchor="w", pady=(5, 0))


log_text = tk.Text(log_frame, height=6, bg="#0f3460", fg="#e0e0e0",
                    font=("Consolas", 9), relief="flat", borderwidth=0)

log_text.pack(fill="both", expand=True, pady=5)


ui_ready = True

```

```

        footer = tk.Label(window, text=" Дипломний проєкт • Візуальна модель ліфта для
середовища програмування SimualIde",

```

```

        font=("Arial", 8), fg="#7f8c8d", bg="#1a1a2e")

```

```

        footer.pack(pady=5)

```

```

# початок роботи

```

```

log_message(" 🚀 Запуск системи..")

```

```

def start_connection():

```

```

    if connect_serial():

```

```

        threading.Thread(target=read_serial, daemon=True).start()

```

```

threading.Thread(target=start_connection, daemon=True).start()

```

```

window.mainloop()

```

A.2 Код для платы Arduino Mega

```
// ===== ПИИ =====
```

```
#define BTN_EXT_1 22
```

```
#define BTN_EXT_2 23
```

```
#define BTN_EXT_3 24
```

```
#define BTN_INT_1 40
```

```
#define BTN_INT_2 41
```

```
#define BTN_INT_3 42
```

```
#define BTN_DOOR_OPEN A12
```

```
#define BTN_DOOR_CLOSE A13
```

```
#define LED_MOVING 13
```

```
#define LED_UP 27
```

```
#define LED_DOWN 29
```

```
#define SEG_A 30
```

```
#define SEG_B 31
```

```
#define SEG_C 32
```

```
#define SEG_D 33
```

```
#define SEG_E 34
```

```
#define SEG_F 35
```

```
#define SEG_G 36
```

```
const int segPins[] = {SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G};
```

```
const byte digitMap[10][7] = {
```

```
    {1,1,1,1,1,1,0}, // 0
```

```
    {0,1,1,0,0,0,0}, // 1
```

```
    {1,1,0,1,1,0,1}, // 2
```

```
    {1,1,1,1,0,0,1}, // 3
```

```
    {0,1,1,0,0,1,1}, // 4
```

```
    {1,0,1,1,0,1,1}, // 5
```

```
    {1,0,1,1,1,1,1}, // 6
```

```
    {1,1,1,0,0,0,0}, // 7
```

```
    {1,1,1,1,1,1,1}, // 8
```

```
    {1,1,1,1,0,1,1} // 9
```

```
};
```

```
// ===== CTAH =====
```

```
int currentFloor = 1;
```

```
int targetFloor = 1;
```

```
bool isMoving = false;
```

```
bool doorsOpen = false;
```

```

// ===== ЧЕРГА =====

int queue[10];

int queueSize = 0;


unsigned long moveStartTime = 0;

const int Move_Duration = 3000;


unsigned long doorTimer = 0;

bool doorTimerActive = false;


// ===== захист від брязкоту контактів =====

unsigned long lastBtnTime1 = 0, lastBtnTime2 = 0, lastBtnTime3 = 0;

unsigned long lastBtnInt1 = 0, lastBtnInt2 = 0, lastBtnInt3 = 0;

unsigned long lastBtnDoorOpen = 0, lastBtnDoorClose = 0;

const int Btn_Delay = 100;


// ===== ЧЕРГА (функції) =====


void AddList(int floor) {

    if (floor == currentFloor) return;

    for (int i = 0; i < queueSize; i++) {

        if (queue[i] == floor) return;
    }
}

```

```

    }

    if (queueSize < 10) {

        queue[queueSize++] = floor;

        Serial.print("QUEUE ADD: ");

        Serial.println(floor);

    }

}

int ListTake() {

    if (queueSize == 0) return -1;

    int f = queue[0];

    for (int i = 0; i < queueSize - 1; i++) {

        queue[i] = queue[i + 1];

    }

    queueSize--;

    return f;

}

// ===== PYX =====

void startMoving(int floor) {

    if (floor == currentFloor) return;

```

```

targetFloor = floor;

isMoving = true;

moveStartTime = millis();

digitalWrite(LED_MOVING, HIGH);


if (targetFloor > currentFloor) {

    digitalWrite(LED_UP, HIGH);

    digitalWrite(LED_DOWN, LOW);

} else {

    digitalWrite(LED_UP, LOW);

    digitalWrite(LED_DOWN, HIGH);

}


Serial.print("MOVING:");

Serial.print(currentFloor);

Serial.print("->");

Serial.println(targetFloor);

}


// ===== ДБЕПИ =====


void openDoors() {

```

```

if (!doorsOpen && !isMoving) {

    doorsOpen = true;

    Serial.println("DOOR:OPEN");

    doorTimer = millis();

    doorTimerActive = true;

}

}

void closeDoors() {

    if (doorsOpen) {

        doorsOpen = false;

        Serial.println("DOOR:CLOSED");

    }

}

// ===== ИНДИКАТОР =====

void displayDigit(int number) {

    if (number < 0 || number > 9) return;

    for (int i = 0; i < 7; i++) {

        digitalWrite(segPins[i], digitMap[number][i]);

    }

```

```
}

// ===== SETUP =====

void setup() {

  Serial.begin(9600);

  pinMode(BTN_EXT_1, INPUT_PULLUP);

  pinMode(BTN_EXT_2, INPUT_PULLUP);

  pinMode(BTN_EXT_3, INPUT_PULLUP);

  pinMode(BTN_INT_1, INPUT_PULLUP);

  pinMode(BTN_INT_2, INPUT_PULLUP);

  pinMode(BTN_INT_3, INPUT_PULLUP);

  pinMode(BTN_DOOR_OPEN, INPUT_PULLUP);

  pinMode(BTN_DOOR_CLOSE, INPUT_PULLUP);

  pinMode(LED_MOVING, OUTPUT);

  pinMode(LED_UP, OUTPUT);

  pinMode(LED_DOWN, OUTPUT);

  digitalWrite(LED_MOVING, LOW);

  digitalWrite(LED_UP, LOW);
```

```
digitalWrite(LED_DOWN, LOW);
```

```
for (int i = 0; i < 7; i++) {
```

```
    pinMode(segPins[i], OUTPUT);
```

```
    digitalWrite(segPins[i], LOW);
```

```
}
```

```
displayDigit(currentFloor);
```

```
delay(500);
```

```
Serial.println("READY");
```

```
Serial.println("FLOOR:1");
```

```
}
```

```
// ===== LOOP =====
```

```
void loop() {
```

```
    unsigned long now = millis();
```

```
    // SERIAL
```

```
    if (Serial.available()) {
```

```
        String cmd = Serial.readStringUntil('\n');
```

```
        cmd.trim();
```

```

if (cmd.startsWith("FLOOR:")) {

    int f = cmd.substring(6).toInt();

    if (f != currentFloor) AddList(f);

}

if (cmd == "DOOR:OPEN") openDoors();

if (cmd == "DOOR:CLOSE") closeDoors();

}

// КНОПКИ ВИКЛИКУ (працюють ЗАВЖДИ, навіть під час руху)

if (digitalRead(BTN_EXT_1) == LOW && now - lastBtnTime1 > Btn_Delay) {

    lastBtnTime1 = now; AddList(1);

}

if (digitalRead(BTN_EXT_2) == LOW && now - lastBtnTime2 > Btn_Delay) {

    lastBtnTime2 = now; AddList(2);

}

if (digitalRead(BTN_EXT_3) == LOW && now - lastBtnTime3 > Btn_Delay) {

    lastBtnTime3 = now; AddList(3);

}

if (digitalRead(BTN_INT_1) == LOW && now - lastBtnInt1 > Btn_Delay) {

    lastBtnInt1 = now; AddList(1);

}

```

```

if (digitalRead(BTN_INT_2) == LOW && now - lastBtnInt2 > Btn_Delay) {

    lastBtnInt2 = now; AddList(2);

}

if (digitalRead(BTN_INT_3) == LOW && now - lastBtnInt3 > Btn_Delay) {

    lastBtnInt3 = now; AddList(3);

}


// КНОПКИ ДВЕРЕЙ

if (digitalRead(BTN_DOOR_OPEN) == LOW && now - lastBtnDoorOpen >
Btn_Delay) {

    lastBtnDoorOpen = now; openDoors();

}

if (digitalRead(BTN_DOOR_CLOSE) == LOW && now - lastBtnDoorClose >
Btn_Delay) {

    lastBtnDoorClose = now; closeDoors();

}


// АВТОЗАКРИТИЯ ДВЕРЕЙ

if (doorTimerActive && !isMoving && now - doorTimer > 3000) {

    closeDoors();

    doorTimerActive = false;

}

```

```

// PYX

if (isMoving && now - moveStartTime > Move_Duration) {

    currentFloor = targetFloor;

    displayDigit(currentFloor);

    isMoving = false;


    digitalWrite(LED_MOVING, LOW);

    digitalWrite(LED_UP, LOW);

    digitalWrite(LED_DOWN, LOW);


    Serial.print("ARRIVED:");

    Serial.println(currentFloor);

    openDoors();

}


// ДАЛП

if (!isMoving && !doorsOpen && queueSize > 0) {

    int next = ListTake();

    if (next != currentFloor) startMoving(next);

}


delay(10);

```

}

A.3 Код скрипта для автоматичного налаштування Com порту

```
@echo off
```

```
title Virtual COM Port Setup
```

```
echo Virtual COM Port Setup
```

```
net session >nul 2>&1
```

```
if %errorlevel% neq 0 (
```

```
    echo [ERROR] Run as Administrator!
```

```
    pause
```

```
    exit /b 1
```

```
)
```

```
set "COMPATH=%~dp0driver"
```

```
if not exist "%COMPATH%\setupc.exe" (
```

```
    echo [FAIL] Portable driver NOT found in: %COMPATH%
```

```
    echo Please put setupc.exe, com0com.sys, com0com.inf into the 'driver' folder.
```

```
    pause
```

```
    exit /b 1
```

```
)
```

```
echo [OK] Portable driver found.
```

```
::(/check)

if /I "%1"==" /check" (

    echo    SAFE CHECK MODE

    echo [INFO] Current virtual port pairs:

    cd /d "%COMPATH%"

    setupc.exe list

    echo CHECK COMPLETE.

    pause

    exit /b 0

)

::Install

echo [1/2] Removing old pairs...

cd /d "%COMPATH%"

setupc.exe remove 0 >nul 2>&1

setupc.exe remove 1 >nul 2>&1

setupc.exe remove 2 >nul 2>&1

echo [2/2] Creating COM2 ^<-> COM3...

setupc.exe install PortName=COM2 PortName=COM3

echo SUCCESS!

pause
```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Розробка програмного забезпечення віртуальної моделі ліфта

Виконав: студент групи КНТ-112
Дмитро БАБАНЦЕВ

Керівник проф. д.т.н.
Михайло ПОЛЯКОВ

Рисунок Б.1 – Слайд 1

Предмет дослідження — методи та засоби створення віртуальних моделей (Digital Twin) пристроїв IoT для прискореного налагодження алгоритмів керування на прикладі програмно-апаратного комплексу «Система керування ліфтом» (SimulIDE + Python/Tkinter).

Об'єкт дослідження — процес інженерії програмного забезпечення вбудованих систем.

Мета роботи — скорочення часу налагодження програмного забезпечення для систем Інтернету речей (IoT) шляхом створення програмно-апаратного комплексу з віртуальною моделлю ліфта (Digital Twin), яка дозволяє тестувати алгоритми керування та людино-машинний інтерфейс без використання фізичного обладнання.

Завдання роботи:

- 1) Провести аналіз існуючих платформ для створення віртуальних моделей пристроїв IoT (Tinkercad, Wokwi, Proteus, SimulIDE) та обрати оптимальну за критеріями відкритості архітектури та можливості інтеграції із зовнішнім ПЗ.
- 2) Спроекувати архітектуру розподіленої системи керування, що імітує взаємодію фізичного пристрою IoT (контролер) та хмарного SCADA-інтерфейсу (Python).
- 3) Розробити протокол обміну даними між віртуальним контролером та SCADA-клієнтом на основі текстових команд через UART.
- 4) Створити графічний інтерфейс оператора мовою Python (Tkinter) з плавною анімацією руху кабіни та дверей.
- 5) Забезпечити двонаправлений зв'язок між компонентами системи з мінімальними затримками в реальному часі.
- 6) Провести тестування системи в різних режимах роботи та оцінити якість виконання

Рисунок Б.2 – Слайд 2

Функціонал

1. Основні функції керування:

Прийом викликів: Реєстрація викликів з плати Arduino і Python програми.

Алгоритм черги: Обробка попутних викликів FIFO.

Управління дверима: Відкриття/закриття дверей з фіксацією статусу, автоматичне закриття за таймером.

2. Функції візуалізації (SCADA/HMI):

Анімація руху: Плавне переміщення графічного об'єкта «Кабіна» між поверхами.

Анімація дверей: Плавне роз'їжджання/зрушення дверей.

Індикація: Відображення поточного поверху (табло), напрямку руху (стрілка), статусу дверей.

Журналізація: Протоколювання всіх команд та подій у текстовому лозі.

3. Сервісні функції:

Авторозгортання: Можливість автоматичного налаштування віртуальних COM-портів через скрипт.

3

Рисунок Б.3 – Слайд 3

архітектура проєкту та мова програмування

Критерій	Все в одному вікні (Proteus)	Дворівнева система (SimulIDE + Python)	Веб-інтерфейс (HTML/JS)
Плавна анімація	Обмежена	Повна (Tkinter)	Повна (CSS)
Зв'язок з контролером	Внутрішній	UART (COM-порт)	WebSocket / HTTP
Відповідність IoT	Немає	Так (Digital Twin)	Так (Web)
Відкритість коду	Пропрієтарний	Open Source	Open Source
Вартість	Платна (\$248+)	Безкоштовно	Безкоштовно
Складність реалізації	Середня	Середня	Висока

4

Рисунок Б.4 – Слайд 4

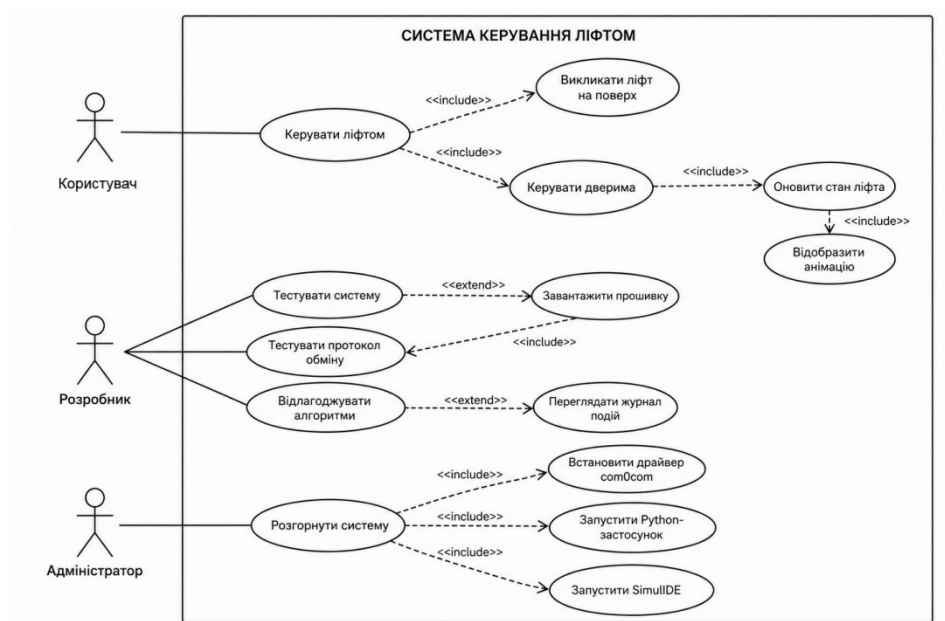
мови програмування

Критерій	C++ (Arduino)	Python (Tkinter)	JavaScript (Web)
Робота з мікроконтролером	Пряма	Немає	Немає
Створення GUI	Немає	Tkinter	HTML/CSS
Підтримка UART	Serial	pySerial	Web Serial API
Швидкодія	Висока	Достатня	Достатня
Складність освоєння	Середня	Низька	Середня

5

Рисунок Б.5 – Слайд 5

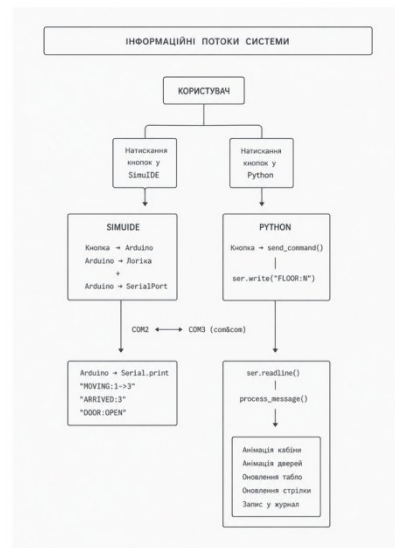
Діаграма прецедентів



6

Рисунок Б.6 – Слайд 6

інформаційні потоки системи



7

Рисунок Б.7 – Слайд 7

Графічний інтерфейс (SCADA/HMI) та схема плати Arduino mega

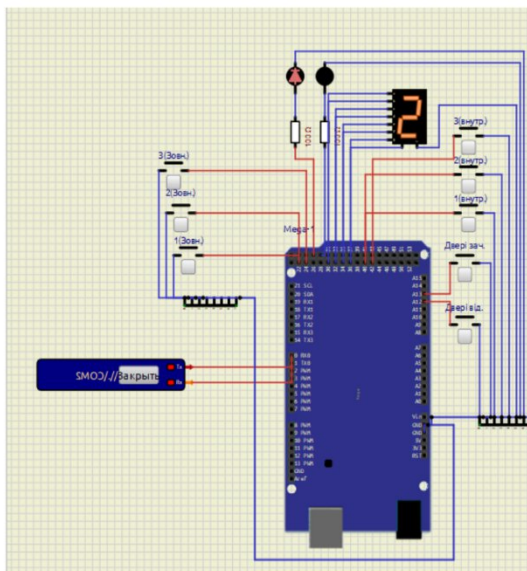


Рисунок Б.8 – Слайд 8

Склад моделі

SIMULIDE (Нижній рівень)
Arduino Mega 2560

6 кнопок виклику
2 кнопки керування дверей
7-сегм. індикатор
Світлодіоди
SerialPort (COM2)



PYTHON (Tkinter, Верхній рівень)

Графічний інтерфейс
Анімація кабіни/дверей
Панель керування
Журнал подій
serial (COM3)

9

Рисунок Б.9 – Слайд 9

онлайн аналоги Simulide

Популярними онлайн аналогами програми Simulide є TinkerCad а також Wokwi.

Переваги:

- не потребують встановлення
- вбудований редактор коду для плати
- підтримка базових компонентів(Діоди, кнопки, т.д.)
- інтуїтивний та зручний інтерфейс

Недоліки:

- Як і більшість онлайн платформ закрита для стороннього софту

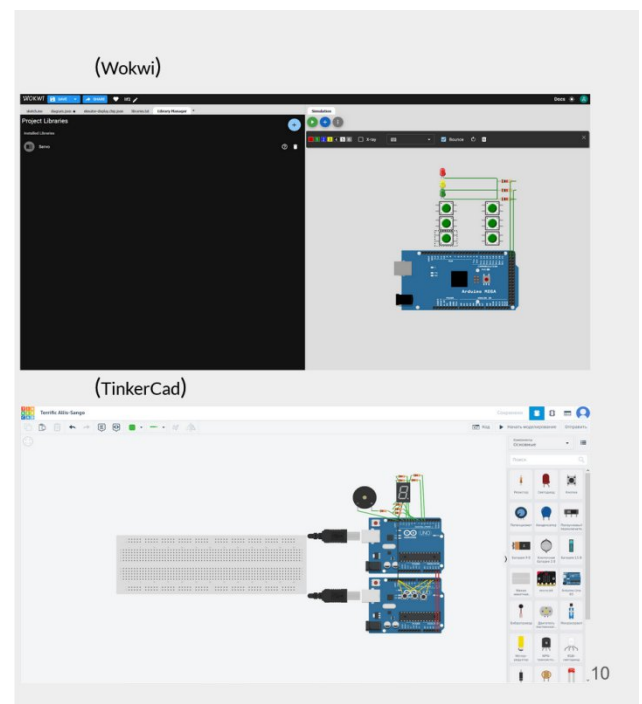


Рисунок Б.10 – Слайд 10

Офлайн аналог SimulIDE - Proteus

Proteus - це програма для професійного проєктування електронних схем, з модулем VSM(Virtual System Modeling). Proteus має велику кількість переваг таких як:

- Найширший вибір компонентів(Понад 15 000)
- Можливість емуляцій складних схем(мікроконтролер + периферія + графіка)
- Відладка коду прямо в симуляторі

Проте в потужному та професійному пакеті Proteus є і недоліки.

- Висока вартість ліцензії(Від 248 \$ за базову версію.)
- Складний інтерфейс
- Закрита архітектура
- Хоча в Proteus присутня внутрішня візуалізація, вона доволі обмежена

11

Рисунок Б.11 – Слайд 11

Результати тестування програми, протоколу обміну а також інтерфейсу користувача

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Запуск Python-застосунку	Статус « Ліфт готов»	Відповідає
2	Натискання кнопки «2» у Python	Відправка FL00R:2 , ліфт починає рух	Відповідає
3	Натискання кнопки в SimulIDE	Отримання MOVING:X->Y , анімація руху	Відповідає
4	Прибуття на поверх	Отримання ARRIVED:N , відкриття дверей	Відповідає
5	Натискання «ВІДКРИТИ» у Python	Відправка DOOR:OPEN , анімація дверей	Відповідає
6	Автоматичне закриття дверей	Через 3 секунди DOOR:CLOSED	Відповідає

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Рух кабіни з 1 на 3 поверх	Плавна анімація, табло змінюється при проїзді поверху	Відповідає
2	Відкриття дверей після прибуття	Плавна анімація роз'їзду дверей	Відповідає
3	Індикатор напрямку	Коректна зміна при русі вгору/вниз/зупинці	Відповідає
4	Журнал подій	Усі команди протоколу фіксуються в логі	Відповідає
5	Optimistic UI (кнопка дверей)	Анімація запускається миттєво, без затримок	Відповідає

12

Рисунок Б.12 – Слайд 12

Висновки

У результаті виконання дипломної роботи було розроблено програмно-апаратний комплекс для симуляції роботи ліфта на три поверхи.

Проведено аналіз існуючих платформ для симуляції мікроконтролерів (Tinkercad, Wokwi, Proteus, SimulIDE) та обґрунтовано вибір SimulIDE

Спроектowana розподілена система керування, що складається з двох незалежних модулів: віртуального контролера Arduino Mega 2560 (SimulIDE) та SCADA-клієнта (Python/Tkinter), з'єднаних через віртуальний COM-порт за протоколом UART.

Розроблено протокол обміну даними на основі текстових команд, що забезпечує двосторонню синхронізацію стану між контролером та інтерфейсом.

Реалізовано алгоритм черги викликів з підтримкою попутних викликів та автоматичним закриттям дверей після прибуття.

Створено графічний інтерфейс оператора (HMI) мовою Python з використанням бібліотеки Tkinter, який забезпечує плавну анімацію руху кабіни, відкриття/закриття дверей, індикацію напрямку руху, відображення поточного поверху на табло та журнал подій.

Забезпечено двонаправлений зв'язок з реалізацією Optimistic UI, що дозволяє запускати анімацію миттєво, не очікуючи підтвердження від контролера, компенсуючи затримки віртуального COM-порту.

13

Рисунок Б.13 – Слайд 13