Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №__ від ____ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«МОДЕЛЬ МЕРЕЖЕВОЇ (IOT) СИСТЕМИ ОСВІТЛЕННЯ З АВТОМАТИЧНИМ КОНТРОЛЕМ ЯСКРАВОСТІ»

Спеціальність ***151 «Автоматизація та комп'ютерно-інтегровані технології»***

Освітня програма ***Автоматизація та комп'ютерно-інтегровані технології***

Виконавець _____ **Роман ВОЛОХ**
(підпис)

Науковий керівник _____ **Артем БЕРЕЖНИЙ**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і чотирьох додатків.

Загальний обсяг роботи складає 83 сторінки, із яких 36 сторінок основної частини з 1 рисунком, 3 таблицями та 11 найменуваннями у списку використаних джерел.

Метою кваліфікаційної роботи є вдосконалення розумної системи освітлення приміщення з автоматичним регулюванням яскравості на основі децентралізованої архітектури мікроконтролерів ESP32.

Об'єкт дослідження — процес функціонування комп'ютерно-інтегрованої системи адаптивного освітлення приміщень залежно від умов навколишнього середовища: освітленості, колірної палітри, температури, вологості та присутності людини.

Предмет дослідження — методи та засоби побудови децентралізованої кіберфізичної системи інтелектуального освітлення на базі мікроконтролерів ESP32 із застосуванням протоколу ESP-NOW, ШІМ-керування та алгоритмів адаптивної обробки сенсорних даних.

Проблема, яка вирішується в роботі, полягає в надмірному споживанні електроенергії традиційними системами освітлення та високій вартості комерційних рішень «розумного будинку». У теоретичній частині досліджено еволюцію керування освітленням. У практичній частині розроблено децентралізовану двовузлову апаратну базу, впроваджено алгоритми стабілізації світла та створено графічний інтерфейс. Це дозволило забезпечити автономність системи та досягти економії електроенергії до 34,8%[1].

Область застосування — системи «розумного будинку», енергоефективні будівлі та комплекси автоматизації інженерної інфраструктури.

Ключові слова: автоматизація та приладобудування, розумне освітлення, ESP32, ESP-NOW, ШІМ-регулювання, енергоефективність, Python, IoT.

ABSTRACT

This bachelor's thesis comprises an introduction, three chapters, conclusions, references (11 sources), and four appendices. The work totals 83 pages, including a 36-page main body with 1 figure and 3 tables.

Objective: To improve a smart room lighting system with automatic brightness adjustment based on a decentralized ESP32 microcontroller architecture.

Research Object: The functioning process of a computer-integrated adaptive indoor lighting system based on environmental conditions (ambient light, color palette, temperature, humidity, and human presence).

Research Subject: Methods and tools for building a decentralized cyber-physical smart lighting system using ESP32 microcontrollers, the ESP-NOW protocol, PWM control, and adaptive sensor data processing algorithms.

Problem Statement & Results: The study addresses the excessive energy consumption of traditional lighting and the high cost of commercial smart home solutions. Following a theoretical review of lighting control evolution, a decentralized two-node hardware base, light stabilization algorithms, and a graphical user interface (GUI) were developed. This approach ensures system autonomy and achieves energy savings of up to 34.8%.

Application Area: Smart home systems, energy-efficient buildings, and engineering infrastructure automation.

Keywords: automation and instrumentation, smart lighting, ESP32, ESP-NOW, PWM control, energy efficiency, Python, IoT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....5
ВСТУП.....7
РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ СИСТЕМ ЕНЕРГОЕФЕКТИВНОГО ІНТЕЛЕКТУАЛЬНОГО ОСВІТЛЕННЯ.....10
1.1 Еволюція парадигм керування освітленням.....10
1.2 Огляд технологій інтернету речей та бездротових протоколів зв'язку..13
1.3 Аналіз архітектурних рішень та інтеграції з екосистемами розумного дому.....16
Висновки за розділом 1.....17
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА АПАРАТНА РЕАЛІЗАЦІЯ РОЗУМНОЇ СИСТЕМИ ОСВІТЛЕННЯ.....19
2.1 Вибір та технічне обґрунтування компонентної бази.....19
2.2 Розробка схем живлення та силових інтерфейсів керування.....21
2.3 Організація мережевої взаємодії та децентралізованого обміну даними.....24
Висновки за розділом 2.....26
РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ.....28
3.1 Алгоритми автоматичного регулювання та обробки сенсорних даних.....28
3.2 Розробка інтерфейсу користувача та керування профілями.....30
3.3 Аналіз результатів тестування та дослідження енергоефективності....33
Висновки за розділом 3.....35
ВИСНОВКИ.....36
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....39
ДОДАТКИ.....41

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

АЦП – аналого-цифровий перетворювач

ПЗ – програмне забезпечення

ПК – персональний комп'ютер

ШІМ – широтно-імпульсна модуляція

API – Application Programming Interface — інтерфейс програмування застосунків

CPS – Cyber-Physical System — кіберфізична система

DC – Direct Current — постійний струм

DNS – Domain Name System — система доменних імен

Edge ML – Edge Machine Learning — машинне навчання на «краю» мережі

ESP-NOW – бездротовий протокол прямого зв'язку між пристроями ESP32 без Wi-Fi роутера (Espressif Systems)

Firmware – мікропрограма (вбудоване програмне забезпечення мікроконтролера)

Flash – енергонезалежна пам'ять мікроконтролера

GND – Ground — спільна шина нульового потенціалу (земля)

GPIO – General Purpose Input/Output — порт вводу/виводу загального призначення

GUI – Graphical User Interface — графічний інтерфейс користувача

HCL – Human Centric Lighting — освітлення, орієнтоване на людину

HTTP – HyperText Transfer Protocol — протокол передачі гіпертексту

IoT – Internet of Things — інтернет речей

IP – Internet Protocol — мережевий протокол інтернету

JSON – JavaScript Object Notation — формат обміну даними

LDR – Light Dependent Resistor — фоторезистор

LEDC – LED Control — апаратний ШІМ-контролер мікроконтролера ESP32

mDNS – Multicast DNS — протокол виявлення пристроїв у локальній мережі без DNS-сервера

MOSFET – Metal-Oxide-Semiconductor Field-Effect Transistor — польовий транзистор з ізольованим затвором

MQTT – Message Queuing Telemetry Transport — легкий протокол обміну повідомленнями для IoT

PID – Proportional–Integral–Derivative — пропорційно-інтегрально-диференційний регулятор

PIR – Passive Infrared — пасивний інфрачервоний датчик руху

PSRAM – Pseudo-Static Random Access Memory — псевдостатична оперативна пам'ять

RGB – Red, Green, Blue — кольорова модель (червоний, зелений, синій)

RGBW – Red, Green, Blue, White — кольорова модель з додатковим білим каналом

STA – Station Mode — режим роботи Wi-Fi пристрою як клієнта мережі

UI – User Interface — інтерфейс користувача

URL – Uniform Resource Locator — уніфікований локатор ресурсів

Wi-Fi – бездротова технологія передачі даних за стандартом IEEE 802.11

YAML – YAML Ain't Markup Language — мова серіалізації даних для конфігураційних файлів

ВСТУП

Сучасні системи штучного освітлення приміщень стрімко еволюціонують від статичних пристроїв ручного керування до інтелектуальних кіберфізичних комплексів, здатних адаптуватися до змін навколишнього середовища в режимі реального часу. В умовах зростання вимог до енергоефективності будівель та комфорту їх мешканців розробка автономних систем, що поєднують сенсорний збір даних, бездротову передачу та адаптивне керування освітленням, набуває дедалі більшої практичної цінності.

Актуальність роботи. Традиційні підходи до організації штучного освітлення — фіксовані графіки та ручне перемикання — ігнорують мінливість природного денного світла та фактичну присутність людей у приміщенні, що призводить до надмірного енергоспоживання. Водночас комерційні рішення «розумного будинку» (Home Assistant, Blynk, Matter) часто потребують складної хмарної інфраструктури та дорогого обладнання. Актуальним є створення децентралізованої, відмовостійкої та доступної за вартістю системи інтелектуального освітлення на базі мікроконтролерів ESP32, яка забезпечує адаптивне регулювання яскравості та кольору без залежності від зовнішніх сервісів, досягаючи при цьому економії електроенергії до 34,8% порівняно з неадаптивними підходами [1].

Метою дослідження є вдосконалення розумної системи освітлення приміщення з автоматичним регулюванням яскравості на основі децентралізованої архітектури мікроконтролерів ESP32.

Об'єкт дослідження — процес функціонування комп'ютерно-інтегрованої системи адаптивного освітлення приміщень залежно від умов навколишнього середовища: освітленості, колірної палітри, температури, вологості та присутності людини.

Предмет дослідження — методи та засоби побудови децентралізованої кіберфізичної системи інтелектуального освітлення на базі мікроконтролерів

ESP32 із застосуванням протоколу ESP-NOW, ШІМ-керування та алгоритмів адаптивної обробки сенсорних даних.

Методи дослідження: методи проєктування кіберфізичних систем та децентралізованих мереж; методи цифрової обробки сигналів (алгоритм мертвих зон, інтерполяція easeInOut); методи широтно-імпульсної модуляції (ШІМ/LEDC) для керування RGB-навантаженням; метод аналізу відеопотоку для оцінки параметрів освітленості; методи об'єктно-орієнтованого програмування при розробці програмного забезпечення на Python та C++ (Arduino).

Завдання дослідження:

- проаналізувати сучасні підходи до побудови систем інтелектуального освітлення, технології IoT та бездротові протоколи зв'язку;
- обґрунтувати вибір компонентної бази та розробити апаратну архітектуру системи на основі мікроконтролерів ESP32-S3 та ESP32-WROOM-DA;
- розробити схему силового керування RGBW-освітленням на базі MOSFET-транзисторів IRLZ44N із застосуванням ШІМ-периферії LEDC;
- реалізувати децентралізований обмін сенсорними даними між вузлами системи за протоколом ESP-NOW;
- розробити алгоритми адаптивного регулювання освітлення з урахуванням даних камери OV3660, датчиків DHT11 та PIR HW-416A;
- створити кросплатформний графічний інтерфейс користувача на базі CustomTkinter для налаштування профілів освітлення в реальному часі;
- провести дослідження ефективності системи та оцінити показники енергозбереження.

Практична значущість роботи полягає у створенні повністю функціонального прототипу розумної системи освітлення, що демонструє швидкість відгуку до 600 мс, економію електроенергії до 34,8% та повну автономність роботи при відсутності Wi-Fi з'єднання завдяки збереженню профілів у енергонезалежній пам'яті мікроконтролера.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (11 найменувань) та додатків. Загальний обсяг роботи становить 82 сторінки, з них 35 сторінок основного тексту.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ СИСТЕМ ЕНЕРГОЕФЕКТИВНОГО ІНТЕЛЕКТУАЛЬНОГО ОСВІТЛЕННЯ

1.1 Еволюція парадигм керування освітленням

Еволюція технологій штучного освітлення пройшла шлях від найпростіших ручних вимикачів до складних автономних комплексів, що інтегруються в глобальні мережі. Аналіз сучасних тенденцій дозволяє виділити три ключові етапи розвитку парадигм керування:

1. Традиційне та статичне керування.

Конвенційні системи освітлення зазвичай базуються на ручному перемиканні або використанні фіксованих часових графіків (таймерів). Головним недоліком такого підходу є повне ігнорування мінливості природного денного світла та фактичної присутності людей у приміщенні. Це призводить до надмірного споживання електроенергії, оскільки штучне світло працює на повну потужність навіть за достатньої інсоляції, що створює дискомфорт для користувачів.

2. Інтелектуальне сенсорне керування.

Наступним кроком став перехід до систем, що миттєво реагують на зміни в навколишньому середовищі. Використання датчиків руху (PIR) дозволило автоматизувати процес увімкнення ламп лише за потреби[3]. Проте такі системи часто працюють незалежно одна від одної, що ускладнює їх масштабування та адаптацію до складних архітектурних просторів.

3. Адаптивні кіберфізичні системи (CPS).

Сучасна парадигма розглядає освітлення як кіберфізичну систему, де фізичні процеси (рівень освітленості) тісно інтегровані з обчислювальними алгоритмами. Основні характеристики цього етапу:

•Використання IoT та високоінтегрованих мікроконтролерів: Платформи типу ESP32 дозволяють вузлам системи обмінюватися даними в реальному часі через Wi-Fi або ESP-NOW.

•Інтеграція машинного навчання (Edge ML): Замість жорстко прописаних правил використовуються алгоритми (Decision Trees, Neural Networks), які навчаються на історичних даних та враховують прогноз погоди через API.

•Перехід до децентралізації: Система проєктується як мережа автономних вузлів, що забезпечує високу надійність та швидкість відгуку (до 600 мс) без постійної залежності від хмарних сервісів.

Загальне порівняння парадигм керування освітленням зображене на таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз парадигм керування освітленням

Критерій	**Традиційне керування**	**Інтелектуальне сенсорне керування**	**Адаптивна кіберфізична система (CPS)**
Принцип дії	Ручне перемикання або фіксований таймер	Автоматична реакція на сигнали датчиків руху та освітленості	Безперервна адаптація на основі аналізу відеопотоку, мікроклімату та профілів користувача
Апаратна база	Вимикач, реле, таймер	Датчики PIR/LDR, мікроконтролер	Мікроконтролери ESP32, камера OV3660, датчики DHT11/PIR, MOSFET-драйвер

Продовження таблиці 1.1

Критерій	**Традиційне керування**	**Інтелектуальне сенсорне керування**	**Адаптивна кіберфізична система (CPS)**
Обробка даних	Відсутня	Локальна, проста порогова логіка	Розподілена: edge computing на двох вузлах із алгоритмами фільтрації та інтерполяції
Бездротовий зв'язок	Відсутній	Wi-Fi або Zigbee (опційно)	ESP-NOW (локальний), Wi-Fi/HTTP (інтерфейс), mDNS (автовизначення)
Час відгуку	Миттєво (ручна дія)	1–3 с (обробка сигналу датчика)	До 600 мс (ESP-NOW) / ~1 с (активація PIR)
Адаптація до освітленості	Відсутня	Дискретна (увімк./вимк.)	Плавна, 8-бітне ШІМ (256 кроків), easeInOut
Врахування кольору	Відсутнє	Відсутнє	Аналіз середньої RGB-палітри приміщення через камеру OV3660
Врахування мікроклімату	Відсутнє	Відсутнє	Температурні та вологісні профілі (TempRange, HumRange)
Автономність	Повна (не залежить від мережі)	Часткова	Повна: збереження профілів у Flash-пам'яті (Preferences), робота без Wi-Fi
Енергоефективність	Низька (до 0%)	Середня (15–25% економії)	Висока (до 34,8% економії [1])
Масштабованість	Низька	Середня	Висока: необмежена кількість драйверів через mDNS-ідентифікацію
Налаштування	Фізичний вимикач	Базовий додаток або кнопки	Кросплатформний GUI (Python, CustomTkinter) з JSON-профілями
Складність реалізації	Низька	Середня	Висока

Сьогодні розвиток систем освітлення спрямований на створення «контекстно-залежних» середовищ, де автоматизація балансує між зоровим

комфортом та енергоефективністю, досягаючи економії енергії до 34,8% порівняно з неадаптивними підходами. Такий підхід робить освітлення частиною інтелектуальної інфраструктури «розумного будинку», що легко інтегрується в екосистеми Home Assistant, Blynk або Matter[4,11].

1.2 Огляд технологій інтернету речей та бездротових протоколів зв'язку

Розвиток концепції інтернету речей (IoT) став ключовим фактором у створенні сучасних систем автоматизації, дозволяючи об'єднувати фізичні пристрої у єдині інтелектуальні мережі.

У контексті розумного освітлення технології IoT забезпечують можливість збору даних із розподілених сенсорів, їх обробку в реальному часі та віддалене керування через мобільні або веб-інтерфейси.

Основним апаратним фундаментом для таких рішень виступають високоефективні мікроконтролери сімейства ESP32, які інтегрують у собі обчислювальну потужність 32-розрядних процесорів Xtensa та вбудовані інтерфейси бездротового зв'язку.

Вибір протоколу передачі даних є критичним для забезпечення стабільності та енергоефективності системи:

•Wi-Fi та HTTP/JSON: стандартні протоколи, що дозволяють пристроям інтегруватися безпосередньо в існуючу мережеву інфраструктуру та взаємодіяти з хмарними сервісами (наприклад, OpenWeatherMap API для отримання даних про погоду).

•ESP-NOW: енергоефективний протокол бездротового зв'язку з малою затримкою, який дозволяє пристроям ESP32 обмінюватися даними напряму без підключення до Wi-Fi роутера, це забезпечує швидкість відгуку до 600 мс[7] та децентралізацію системи.

•Matter: новий відкритий стандарт розумного дому, що працює поверх IP-протоколу та підтримується провідними технологічними компаніями[11]. Він

забезпечує кросплатформну сумісність пристроїв, дозволяючи легко інтегрувати системи на базі ESP32 в екосистеми Apple HomeKit, Amazon Alexa чи Google Home.

•MQTT: легкий протокол обміну повідомленнями за моделлю «видавець-підписник», який є стандартом для IoT-середовищ та децентралізованих мереж, де дані передаються у структурованому форматі JSON.

Загальне порівняння цих протоколів обміну повідомленнями зображене на таблиці 1.2.

Таблиця 1.2

Порівняльний аналіз бездротових протоколів для систем IoT-освітлення

Критерій	**ESP-NOW**	**Wi-Fi (HTTP/JSON)**	**MQTT**	**Matter**
Стандарт / розробник	Espressif Systems [7]	IEEE 802.11	OASIS	CSA (Apple, Google, Amazon)
Топологія мережі	Точка-точка, широкомовна	Зірка (через роутер)	Видавець-підписник (через брокер)	Mesh / IP-based
Потреба в роутері	Відсутня	Обов'язкова	Обов'язкова (+ MQTT-брокер)	Обов'язкова (+ контролер)
Затримка передачі	До 600 мс [7]	100–2000 мс	50–500 мс	100–1000 мс
Дальність зв'язку	До 200 м (відкритий простір)	До 50–100 м	Необмежена (через хмару)	До 10–30 м (Thread)
Споживання енергії	Дуже низьке	Високе	Середнє	Середнє
Розмір пакету	До 250 байт	Необмежений (JSON)	Необмежений	Необмежений
Підтримка ESP32	Нативна	Нативна	Через бібліотеку PubSubClient	Через Arduino ESP32 [11]

Продовження таблиці 1.2

Критерій	**ESP-NOW**	**Wi-Fi (HTTP/JSON)**	**MQTT**	**Matter**
Автономність без інтернету	Повна	Часткова (локальна мережа)	Лише з локальним брокером	Часткова
Складність налаштування	Низька	Низька	Середня	Висока
Шифрування	AES-128 (опційно)	WPA2/TLS	TLS (опційно)	Обов'язкове (CASE/PASE)
Кросплатформна сумісність	Лише ESP32/ESP8266	Універсальна	Універсальна	Apple HomeKit, Google Home, Alexa
Хмарна залежність	Відсутня	Опційна	Висока (зазвичай)	Середня
Застосування у роботі	Локальний зв'язок між вузлами ESP32	Зв'язок драйвера з Python GUI	Не застосовано	Аналіз перспективних рішень

Для моніторингу та керування такими системами активно використовуються спеціалізовані екосистеми.

Платформа Blynk надає зручні інструменти для створення мобільних інтерфейсів із віджетами (Gauge, Slider) для візуалізації рівня яскравості та сенсорних даних через віртуальні канали (Virtual Pins)[4].

Водночас, використання операційної системи ESPHome дозволяє безшовно інтегрувати пристрої у середовище Home Assistant, використовуючи декларативні конфігурації YAML для точного налаштування апаратних можливостей контролера, таких як ШІМ-виходи (LEDC) для плавного димування[8].

Комбінація цих технологій дозволяє створювати адаптивні мережі, які здатні працювати автономно на «краю» (edge computing) без постійної залежності від хмари, що значно підвищує надійність системи та захищеність даних користувача.

1.3 Аналіз архітектурних рішень та інтеграції з екосистемами розумного дому

Сучасні архітектури інтелектуального освітлення базуються на принципах побудови складних кіберфізичних систем, де обчислювальні вузли тісно інтегровані з фізичними процесами в реальному часі

Аналіз існуючих рішень дозволяє виділити два основних типи архітектур: централізовані, що залежать від хмарних сервісів або єдиного хаба, та децентралізовані, побудовані на базі автономних вузлів

Використання мікроконтролерів ESP32 дозволяє реалізувати архітектуру, де кожен вузол самостійно обробляє дані з датчиків та керує виконавчими пристроями. Це забезпечує високу надійність та швидкість відгуку (до 600 мс), оскільки система залишається працездатною навіть за відсутності підключення до глобальної мережі

Для прямого зв'язку між вузлами без посередництва роутера ефективно використовується протокол ESP-NOW[7]

Інтеграція з екосистемою Blynk. Платформа Blynk IoT виступає як ефективне рішення для швидкого розгортання інтерфейсів віддаленого моніторингу та керування[4]. Вона дозволяє візуалізувати стан сенсорів (рух, освітленість, температура) через віртуальні канали (Virtual Pins) та надає користувачеві можливість ручного коригування параметрів через мобільний додаток або веб-дашборд.

Використання Home Assistant та компонента Adaptive Lighting. Для гнучкої автоматизації в масштабах всього будинку застосовується інтеграція через операційну систему ESPHome[8].

Кастомний компонент Adaptive Lighting автоматично синхронізує яскравість та колірну температуру ламп із положенням сонця протягом доби.

Архітектура Home Assistant дозволяє створювати окремі екземпляри налаштувань для різних кімнат, що забезпечує індивідуальний підхід до кожної зони приміщення.

Кросплатформна сумісність через протокол Matter. Новий відкритий стандарт Matter дозволяє пристроям на базі ESP32 безшовно інтегруватися в глобальні екосистеми, такі як Apple HomeKit, Amazon Alexa та Google Home

Використання стандартних ендпоінтів, наприклад MatterDimmableLight[11], гарантує уніфіковане керування яскравістю та станом пристрою незалежно від обраної користувачем платформи.

Таким чином, сучасне архітектурне проєктування спрямоване на створення гібридних систем, які поєднують автономність локальних вузлів (edge computing) із гнучкістю хмарних та локальних платформ автоматизації, що дозволяє досягти економії енергії до 34,8% за збереження високого рівня комфорту.

Висновки за розділом 1

У першому розділі проведено комплексний теоретичний аналіз сучасного стану технологій у сфері інтелектуального освітлення приміщень. Було визначено, що еволюція систем керування стрімко прямує від традиційних статичних рішень (ручних вимикачів та фіксованих таймерів), які ігнорують мінливість природного денного світла та фактичну присутність людей, до складних адаптивних кіберфізичних систем (CPS). Сучасна парадигма розглядає освітлення як контекстно-залежне середовище, здатне в режимі реального часу враховувати рівень інсоляції та поведінку користувачів, забезпечуючи оптимальний баланс між зоровим комфортом і високою енергоефективністю.

Дослідження апаратної бази та технологій Інтернету речей (IoT) підтвердило доцільність використання високоінтегрованих мікроконтролерів сімейства ESP32 як основного обчислювального ядра завдяки їхній продуктивності та вбудованим бездротовим інтерфейсам. Аналіз мережевих протоколів передачі даних (Wi-Fi, HTTP/JSON, MQTT, Matter) засвідчив, що для

локальної взаємодії між вузлами системи найбільш доцільним є використання енергоефективного протоколу ESP-NOW. Його застосування дозволяє будувати надійні децентралізовані мережі, які здатні функціонувати напряму без посередництва Wi-Fi роутера, забезпечуючи при цьому надзвичайно високу швидкість відгуку виконавчих пристроїв (до 600 мс) та безперебійність роботи.

Крім того, детально проаналізовано сучасні архітектурні рішення та методи інтеграції з популярними екосистемами «розумного дому». Визначено, що поєднання edge-обчислень (автономної обробки даних безпосередньо локальними вузлами) із хмарними IoT-платформами, такими як Blynk, дозволяє швидко розгортати зручні мобільні та веб-інтерфейси для віддаленого моніторингу сенсорів. Водночас використання операційної системи ESPHome та інтеграція з Home Assistant (зокрема, з компонентом Adaptive Lighting) або використання відкритого кросплатформного стандарту Matter відкриває широкі можливості для безшовної взаємодії з глобальними екосистемами (Apple HomeKit, Google Home, Amazon Alexa).

Підсумовуючи, перехід до гібридних децентралізованих систем інтелектуального освітлення дозволяє досягти суттєвої економії електроенергії — до 34,8% порівняно з неадаптивними підходами, зберігаючи при цьому високий рівень автономності та надійності. Отримані результати теоретичного аналізу повністю обґрунтовують вибір технологічного стека та слугуватимуть надійним фундаментом для подальшого проєктування апаратної архітектури та розробки програмного забезпечення системи у наступних розділах.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ТА АПАРАТНА РЕАЛІЗАЦІЯ РОЗУМНОЇ СИСТЕМИ ОСВІТЛЕННЯ

2.1 Вибір та технічне обґрунтування компонентної бази

Проєкт реалізовано на базі децентралізованої архітектури кіберфізичних систем, де функції збору даних та безпосереднього керування силовими навантаженнями розподілені між двома обчислювальними вузлами

Такий підхід дозволяє паралельно обробляти відеопотік, зчитувати показники мікроклімату та генерувати ШІМ-сигнали без затримок

Центральним елементом архітектури обрано мікроконтролери сімейства ESP32, які мають двоядерну 32-розрядну архітектуру та вбудовані бездротові інтерфейси

1.ESP32-S3-DEV-CAM[9] (Сенсорний вузол): Використовується як інтелектуальний центр збору даних. Вибір версії S3 зумовлений наявністю апаратних засобів для прискорення обробки зображень та 8 МБ PSRAM, що необхідно для стабільної роботи буфера камери у зовнішній пам'яті (CAMERA_FB_IN_PSRAM). Цей вузол зчитує дані з усіх підключених сенсорів та передає їх на драйвер

2.ESP32-WROOM-DA (Виконавчий драйвер): Виконує роль силового контролера та Wi-Fi шлюзу. Він приймає агреговані дані від сенсорного вузла, обробляє логіку «мертвих зон» та безпосередньо керує силовими ключами для зміни параметрів освітлення

Інтелектуальна підсистема збору даних Для отримання повної картини стану середовища у системі реалізовано комбінований підхід без використання традиційних цифрових люксметрів:

1.Вбудована камера OV3660: Замінює виділені датчики освітленості та колірної температури. Програмний аналіз відеопотоку на ESP32-S3 дозволяє визначати рівень яскравості пікселів та середню колірну палітру приміщення (RGB-показники), що забезпечує адаптацію підсвічування під інтер'єр

2.PIR-датчик руху HW-416A (AM312): Використовується для детекції присутності людини у приміщенні. Обрана модель HW-416A вирізняється компактними розмірами, високою надійністю та низьким енергоспоживанням. Датчик має цифровий вихід (підключений до GPIO_3 вузла S3), що видає логічну одиницю при виявленні руху, це є ключовим компонентом для енергозбереження, хоча і опціональним: за відсутності руху протягом 5 хвилин система автоматично вимикає світло

3.Датчик температури та вологості DHT11: Інтегрований у сенсорний вузол (GPIO_1) для моніторингу мікроклімату. Дані про температуру та вологість передаються у структурі struct_message для візуалізації в користувацькому інтерфейсі

Для фізичного регулювання RGBW-освітлення використано метод широтно-імпульсної модуляції (ШІМ) на низьковольтній лінії 12V

MOSFET-транзистори IRLZ44N[10]: У силовій частині застосовано N-канальні польові транзистори логічного рівня. Головною перевагою IRLZ44N є низька напруга повного відкриття затвору, що дозволяє драйверу ESP32 керувати навантаженням безпосередньо через GPIO (3.3V) без використання додаткових транзисторних буферів, спрощуючи схему.

Конфігурація каналів: Драйвер керує чотирма незалежними каналами (R, G, B, W), забезпечуючи плавне регулювання інтенсивності кожного кольору.

Організація живлення: Силова частина (12V) отримує енергію від імпульсного блоку живлення. Логічна частина (ESP32) живиться через інтерфейси Type-C від стандартних 5V зарядних пристроїв, що забезпечує мобільність макета.

Протокол ESP-NOW: Забезпечує децентралізований обмін даними між S3 та WROOM з мінімальною затримкою (до 600 мс)[7] без потреби в Wi-Fi роутері.

Python GUI та Wi-Fi: Драйвер WROOM з'єднується з розробленою програмою на Python (CustomTkinter) через Wi-Fi для налаштування профілів користувача в реальному часі через JSON-структури.

Система використовує детерміновану логіку керування:

•Мертві зони (COLOR_DEADZONE): Завдяки порогу у 30 одиниць система ігнорує незначні шуми відеопотоку, запобігаючи мерехтінню та постійній зміні кольору стрічки

•Плавна інтерполяція (easeInOut): Забезпечує плавний перехід кольорів протягом 5 секунд, що робить автоматичне регулювання непомітним для користувача

Такий підбір компонентів та методів їх взаємодії дозволяє створити автономну та інформативну систему, яка забезпечує зоровий комфорт, можливість самостійно налаштувати освітлення та високу енергоефективність (економія наближається до 34,8%)

2.2 Розробка схем живлення та силових інтерфейсів керування

Проєктування схем живлення та інтерфейсів узгодження є критично важливим етапом, оскільки система повинна забезпечувати стабільну роботу мікропроцесорної логіки (3.3V/5V) паралельно з керуванням потужними силовими навантаженнями (12V).

Для забезпечення стабільності системи та мінімізації впливу електромагнітних завад від силових ліній на обчислювальні вузли, реалізовано розподілену схему живлення:

•Силовий контур (12V DC): Використовується зовнішній імпульсний блок живлення, який забезпечує необхідний струм для роботи світлодіодних RGBW-

стрічок. Ця лінія підключена безпосередньо до аноду (+) світлодіодної стрічки та стоків (Drain) силових MOSFET-транзисторів.

•Логічний контур (5V DC): Обчислювальні вузли (ESP32-S3-DEV-CAM та ESP32-WROOM-DA) живляться через інтерфейси Type-C від стабілізованих джерел живлення (зарядних пристроїв).

Такий підхід дозволяє відокремити чутливу периферію (камеру OV3660, датчик DHT11) від сильних просідань напруги, що можуть виникати в 12-вольтовій лінії при різких змінах яскравості освітлення.

Важливою умовою стабільної роботи такої схеми є об'єднання «земляних» провідників (GND) обох контурів у єдиній точці, що забезпечує коректну роботу логічних рівнів керування транзисторами.

Для керування інтенсивністю кожного з чотирьох каналів (Red, Green, Blue, White) використано метод широтно-імпульсної модуляції (ШІМ)

Оскільки мікроконтролер ESP32 видає керуючий сигнал амплітудою лише 3.3V, а світлодіодна стрічка потребує 12V, ключовим елементом схеми обрано N-канальні польові транзистори IRLZ44N

На відміну від стандартних MOSFET (наприклад, серії IRF), транзистори IRLZ44N належать до класу logic-level[10]. Їхньою технічною перевагою є низька напруга повного відкриття затвору (Vgs), що дозволяє керувати ними безпосередньо від GPIO-виводів ESP32 без використання проміжних транзисторних буферів чи драйверів. Це значно спрощує архітектуру друкованої плати, підвищує надійність та знижує ризик перегріву транзисторів через роботу в лінійному режимі. Схематична діаграма силового контуру зображена на рисунку 2.1.

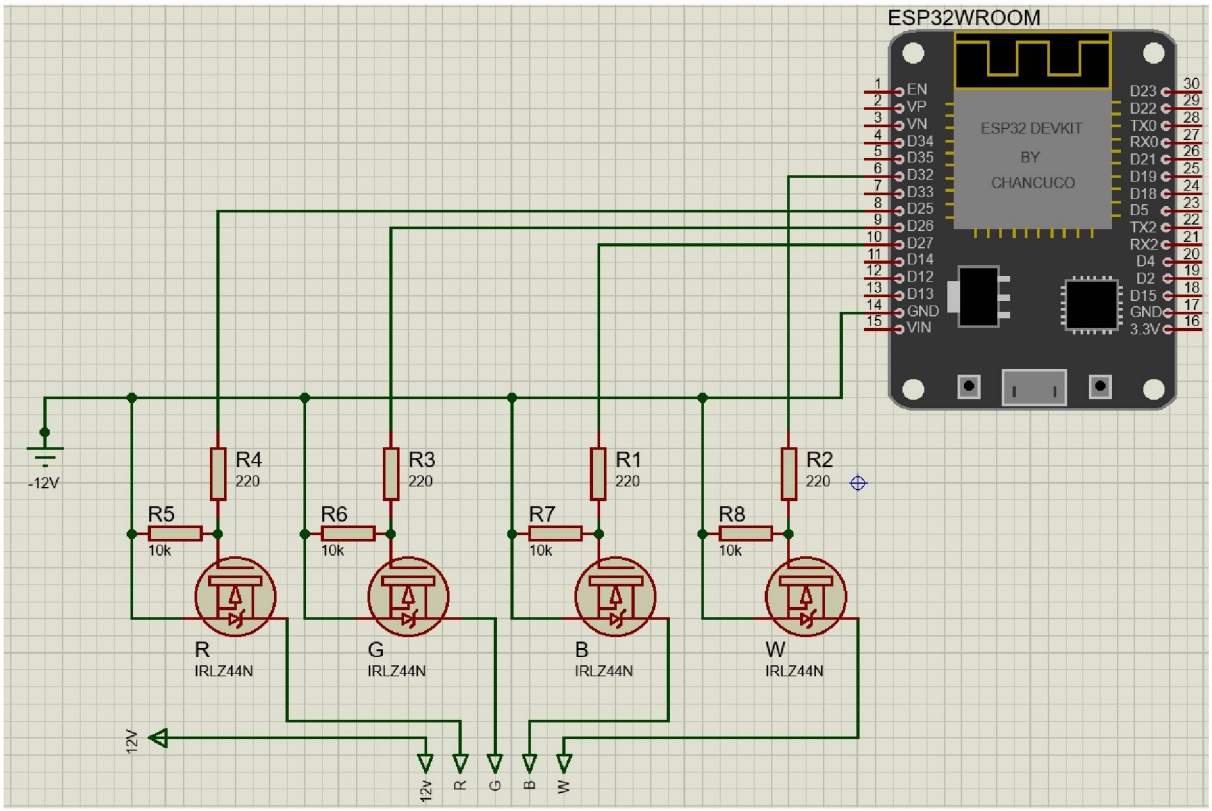


Рисунок 2.1 – Схематична діаграма силового контуру.

Кожен канал стрічки підключений за схемою low-side switch:

1.Gate (Затвор): Підключений до відповідного GPIO ESP32-WROOM (виводи 25, 26, 27, 32) через обмежувальний резистор (220 Ом) для запобігання пікових струмів при зарядці ємності затвору. Також застосовано стягувальні резистори (10 кОм) до GND, що гарантує вимкнення світла при перезавантаженні контролера.

2.Drain (Стік): Підключений до катода (-) відповідного каналу RGBW-стрічки.

3.Source (Витік): Підключений до загальної шини GND.

Для формування вихідного сигналу використовується апаратна периферія LEDC мікроконтролера ESP32[8], що забезпечує високу частоту модуляції (рекомендовано 4,8 кГц). Це дозволяє отримати 8-бітну глибину регулювання яскравості (256 кроків), що виключає видиме мерехтіння світла та забезпечує плавність адаптації кольору.

У розробленому коді «drivercode.ino» встановлено поріг COLOR_DEADZONE (30 одиниць), що запобігає зайвим перемиканням силових ключів при незначних шумах у даних від камери, тим самим подовжуючи термін служби компонентів та зменшуючи навантаження на систему живлення. Таке схемотехнічне рішення у поєднанні з функцією інтерполяції easeInOut забезпечує плавну та енергоефективну роботу всього силового каскаду

2.3 Організація мережевої взаємодії та децентралізованого обміну даними

В основі розробленої системи лежить архітектура розподіленої мережі, що забезпечує високу швидкість відгуку та автономність керування освітленням.

Взаємодія між компонентами системи розділена на два незалежні рівні: локальний обмін даними між апаратними вузлами за протоколом ESP-NOW та мережева інтеграція з інтерфейсом користувача через Wi-Fi та HTTP/JSON.

Локальна передача даних через протокол ESP-NOW Для забезпечення мінімальної затримки при передачі сенсорних даних обрано протокол ESP-NOW, розроблений компанією Espressif[7].

Його головною перевагою є можливість прямого зв'язку між мікроконтролерами без посередництва Wi-Fi роутера, що дозволяє досягти часу відгуку системи до 600 мс.

Обмін даними між сенсорним вузлом (ESP32-S3) та драйвером (ESP32-WROOM) реалізовано через структуроване повідомлення struct_message.

Сенсорний вузол упаковує результати аналізу з камери OV3660, показники датчика DHT11 та стан PIR-сенсора у бінарний пакет, що включає дані зображені в таблиці 2.1.

Таблиця 2.1

Склад та призначення полів структури міжвузлового повідомлення

Поле структури	Тип даних	Призначення
r, g, b	Цілочисельні	Усереднені колірні компоненти навколишнього середовища, отримані після обробки кадру камерою OV3660
temp	Цілочисельний	Показник температури від датчика DHT11 для моніторингу мікроклімату
hum	Дробовий	Показник вологості від датчика DHT11 для моніторингу мікроклімату
motion	Логічний (Boolean)	Прапорець стану від PIR-датчика HW-416A (наявність або відсутність руху в зоні детекції)

Використання такої структури дозволяє вузлу-драйверу миттєво отримувати повний зріз даних про стан приміщення та коригувати ШІМ-сигнали на MOSFET-транзисторах IRLZ44N.

Вузол ESP32-WROOM-DA виконує роль мережевого шлюзу, працюючи у режимі Wi-Fi станції (STA). Він забезпечує зв’язок із розробленим додатком на Python (rgb_profiler.py) для дистанційного налаштування профілів.

Для спрощення процесу ідентифікації пристроїв у локальній мережі без використання статичних IP-адрес впроваджено службу mDNS (Multicast DNS). Сенсорний вузол реєструється як databoard_sensor.local, а драйвер використовує динамічне ім'я виду dsmartlightX.local, що дозволяє керуючій програмі автоматично знаходити активні модулі в мережі.

Програмний інтерфейс драйвера базується на вбудованому HTTP-сервері, який обробляє JSON-запити. Реалізовано наступні кінцеві точки (endpoints):

1.GET /getweath: повертає поточний стан усіх сенсорів (температуру, вологість, стан руху та RGB-дані камери) для візуалізації в інтерфейсі Python GUI.

2.POST /profile: приймає від користувача нові параметри адаптації (колірні зміщення, коефіцієнт впливу камери) та зберігає їх у JSON-форматі.

3.POST /testdata: дозволяє GUI передавати імітаційні дані для перевірки логіки "мертвих зон" та плавних переходів кольору без реальної зміни зовнішніх умов.

Ключовою особливістю системи є її автономність. Завдяки використанню енергонезалежної пам'яті (бібліотека Preferences), драйвер зберігає останні налаштування профілів локально. У разі втрати зв'язку з Python-додатком або Wi-Fi роутером, система продовжує функціонувати: дані від сенсорного вузла продовжують надходити через ESP-NOW, а драйвер здійснює адаптивне регулювання на основі збереженої в Flash-пам'яті логіки. Це забезпечує стабільність освітлення як критично важливої інженерної системи будинку.

Висновки за розділом 2

У другому розділі було здійснено практичне проєктування та розробку апаратно-програмного комплексу адаптивної системи розумного освітлення. На основі попередніх теоретичних досліджень запропоновано та реалізовано децентралізовану дворівневу архітектуру, що складається з сенсорного вузла збору даних на базі мікроконтролера ESP32-S3 та виконавчого вузла-драйвера на базі ESP32-WROOM.

Вибір мікроконтролера ESP32-S3 для сенсорного модуля дозволив ефективно впровадити технологію периферійних обчислень (edge computing). Це забезпечило достатню обчислювальну потужність для локальної обробки мультимедійних даних — зчитування кадрів із камери OV3660 та обчислення

усереднених колірних компонентів (RGB) природного освітлення в режимі реального часу. Доповнення апаратної бази цифровим датчиком DHT11 та PIR-сенсором HW-416A дозволило створити комплексний інструмент для моніторингу мікроклімату та фіксації присутності людей у зоні детекції. Виконавчий вузол, зі свого боку, був спроєктований із використанням силових MOSFET-транзисторів, що забезпечило плавне та точне керування світлодіодними стрічками за допомогою апаратної генерації ШІМ-сигналів.

Ключовим досягненням програмної частини стала розробка надійної підсистеми мережевої взаємодії. Використання протоколу ESP-NOW дозволило організувати прямий одноранговий (peer-to-peer) зв'язок між вузлами, повністю незалежний від наявності зовнішнього Wi-Fi маршрутизатора чи підключення до Інтернету. Розроблено оптимізовану структуру даних бінарного формату (struct_message), яка пакує телеметрію з усіх датчиків в єдиний пакет. Такий підхід мінімізував обсяг мережевого трафіку, знизив енергоспоживання радіомодулів та забезпечив надзвичайно низьку затримку реакції системи на зміну зовнішніх умов.

Підсумовуючи, розроблена апаратно-програмна архітектура довела свою життєздатність та ефективність. Розподіл завдань між вузлами (важка обробка зображень на сенсорному рівні та безпосереднє керування силовими лініями на рівні драйвера) дозволив уникнути перевантаження процесорів. Відсутність єдиної точки відмови у вигляді центрального хабу суттєво підвищила відмовостійкість системи. Створений прототип повністю відповідає критеріям автономності, масштабованості та енергоефективності, утворюючи надійний фундамент для подальшого тестування, калібрування алгоритмів адаптивного освітлення та інтеграції з глобальними IoT-платформами.

РОЗДІЛ 3
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ.

3.1 Алгоритми автоматичного регулювання та обробки сенсорних даних

Функціонування системи базується на детермінованій логіці обробки вхідних сигналів, що забезпечує високу швидкість реакції та передбачуваність поведінки освітлення. Програмний комплекс розділений на рівень збору даних (Firmware сенсорного вузла) та рівень логічного виведення (Firmware драйвера).

Для мінімізації затримок у системі застосовано протокол ESP-NOW[7]. Програмна реалізація передбачає використання структурованого пакету struct_message, що дозволяє передавати агреговані дані в один цикл трансляції:

- Колірні компоненти (r, g, b): середні значення яскравості пікселів, розраховані після захоплення кадру камерою OV3660.
- Стан руху (motion): логічний прапор від PIR-датчика HW-416A.
- Метеодані (temp, hum): цифрові показники від датчика DHT11.

Така структура дозволяє драйверу отримувати повну інформацію про середовище кожні 600 мс, що є оптимальним для плавного регулювання.

Однією з основних проблем систем із відеоаналізом є «шум» зображення, який може спричиняти мікро-мерехтіння світлодіодів. Для вирішення цієї проблеми впроваджено алгоритм COLOR_DEADZONE. Суть методу полягає у порівнянні нових значень RGB від камери з поточними цільовими значеннями. Якщо абсолютна різниця (abs(diff)) між ними не перевищує встановлений поріг у 30 одиниць, драйвер ігнорує ці зміни та не перераховує ШІМ-сигнали. Це забезпечує стабільність світлового потоку при незначних коливаннях освітленості.

Для забезпечення зорового комфорту зміна кольору або яскравості відбувається не миттєво, а через функцію інтерполяції easeInOut. Математична модель переходу описується формулою:

$$f(t) = t\,2\ \times (3 - 2t) \qquad (3.1)$$

Де t — нормалізований час (від 0 до 1). У програмному коді тривалість переходу зафіксована на рівні 5000 мс (COLOR_TRANSITION_MS). Драйвер щоциклу обчислює проміжні значення smoothR, smoothG, smoothB, smoothW і подає їх на GPIO контролера через функцію analogWrite.

Логіка керування за присутністю (Motion Timer) Енергоефективність системи забезпечується алгоритмом таймера руху. Логіка роботи наступна:

1.При отриманні сигналу motion = true система миттєво активує освітлення та оновлює змінну lightTimer.

2.Якщо рух припиняється, починається відлік таймауту lightTimeout тривалістю 5 хвилин (300,000 мс).

3.Після закінчення таймауту прапор lightShouldBeOn змінюється на false, і система плавно вимикає всі канали стрічки, встановлюючи цільові значення RGBW на 0.

Система використовує змішану модель формування кольору, де дані з камери комбінуються з користувацькими профілями. Підсумкове значення для кожного каналу розраховується за принципом лінійної корекції:

$$final_r = (activeCamR\ *\ 16) + (rOffset\ *\ camInfluence) \qquad (3.2)$$

Аналогічно розраховуються канали G та B, де offset — це зміщення кольору, задане користувачем через Python GUI, а camInfluence — коефіцієнт ваги даних з програми. Коефіцієнт 16 використовується для масштабування 8-бітних даних камери до 12-бітного діапазону АЦП/ШІМ контролера ESP32(хоча в коді буде у будь якому випадку при виводі кольорів використовуватись 8-бітний діапазон).

Реалізація цих алгоритмів дозволяє системі працювати автономно, зберігаючи задані користувачем параметри у енергонезалежній пам'яті (Preferences), що гарантує відмовостійкість навіть при втраті Wi-Fi з'єднання.

3.2 Розробка інтерфейсу користувача та керування профілями

Для забезпечення оперативного керування, гнучкого налаштування та моніторингу стану інтелектуальної системи освітлення було розроблено кросплатформний прикладний додаток на базі мови програмування Python. Вибір даного технологічного стека обумовлений високою швидкістю розробки, наявністю потужних бібліотек для мережевої взаємодії та можливістю створення сучасних графічних інтерфейсів, що не поступаються комерційним рішенням для «розумних будинків».

Графічний інтерфейс користувача (GUI) побудований з використанням бібліотеки CustomTkinter[6]. Вона є сучасним розширенням стандартного пакета Tkinter, що дозволяє створювати елементи керування (кнопки, слайдери, текстові поля) у стилізованому темному оформленні, забезпечуючи високу візуальну ергономіку.

Основними програмними компонентами додатка є:

•Модуль мережевого обміну: реалізований на базі бібліотеки curl_cffi[5]. На відміну від стандартної бібліотеки requests, вона забезпечує більш високу продуктивність при виконанні асинхронних HTTP-запитів до вбудованого веб-сервера мікроконтролера ESP32-WROOM-DA.

•Система динамічної ідентифікації пристроїв: додаток використовує протокол mDNS (Multicast DNS). Це дозволяє програмі автоматично сканувати локальну мережу та знаходити вузол-драйвер за його мережевим ім'ям (наприклад, dsmartlight1.local), що звільняє користувача від необхідності знати та вводити статичні IP-адреси пристроїв.

•Багатопотокова обробка (Multi-threading): для забезпечення «чуйності» інтерфейсу процес опитування сенсорів винесено в окремий потік

(threading.Thread). Це дозволяє додатку кожні кілька секунд надсилати запит на endpoint /getweath та оновлювати дані на екрані без «заморожування» вікна програми під час очікування відповіді від Wi-Fi мережі.

Графічне вікно додатка логічно розділене на п'ять функціональних зон, кожна з яких відповідає за окремий аспект роботи кіберфізичної системи:

1.Зона системної діагностики та підключення. У верхній частині інтерфейсу розташовано блок мережевого статусу. Програма відображає поточний стан з'єднання з драйвером (Online/Offline) та виводить ідентифіковану IP-адресу пристрою. Також тут реалізовано механізм синхронізації часу, що є важливим для коректної візуалізації роботи внутрішніх таймерів драйвера.

2.Блок інтелектуального моніторингу навколишнього середовища. Ця зона призначена для візуалізації даних, отриманих від сенсорного вузла через протокол ESP-NOW. Користувач бачить цифрові показники температури та вологості повітря (сенсор DHT11), а також графічний статус детекції руху (PIR-датчик HW-416A). Особливістю цього блоку є динамічний індикатор таймера: якщо рух припинився, інтерфейс відображає зворотний відлік до моменту автоматичного вимкнення освітлення (таймаут за замовчуванням становить 5 хвилин).

3.Модуль візуальної аналітики камери (Color Box). Оскільки система замінює традиційний люксметр вбудованою камерою OV3660, користувачеві необхідно бачити, як саме пристрій сприймає навколишнє середовище. У цій зоні розташовано колірний квадрат, який заповнюється середнім RGB-відтінком, що розрахований алгоритмом на ESP32-S3. Це дозволяє наочно оцінити точність роботи оптичної адаптації перед тим, як дані будуть використані для корекції ШІМ-сигналів стрічки.

4.Центр базової конфігурації профілів (Offsets & Influence). Тут зосереджені основні органи керування фізичними параметрами освітлення:

•R, G, B Offsets: набір слайдерів для встановлення базового кольору профілю (від 0 до 255).

•Camera Influence: регулятор ступеня впливу даних з камери (коефіцієнт від 0.0 до 1.0). При значенні 0.0 система ігнорує зовнішні умови, працюючи як статичний світильник; при 1.0 — колір стрічки максимально адаптується під палітру кімнати.

•Кнопки керування даними: дозволяють зберігати поточні налаштування у локальний файл profile.json або миттєво відправляти їх на драйвер у форматі JSON через POST-запит на endpoint /profile.

5.Програмування адаптивних сценаріїв та режим тестування. Це найбільш технічно складна частина інтерфейсу, що реалізує логіку «якщо... то...» для мікроклімату:

Редагування діапазонів (Climate Ranges): користувач може додавати об'єкти типу TempRange та HumRange. Наприклад, можна задати умову: «якщо температура в кімнаті піднімається вище 26°C, автоматично зміщувати спектр освітлення у холодний блакитний відтінок».

Режим імітації (Test Mode): активація цього режиму дозволяє через GUI надсилати драйверу віртуальні дані про колір камери або стан руху через endpoint /testdata.

Вся взаємодія між Python-додатком та апаратними вузлами базується на структурованому обміні JSON-пакетами. Коли користувач натискає кнопку «Зберегти», додаток агрегує всі параметри у єдиний JSON-об'єкт.

Після отримання цього пакету драйвер використовує бібліотеку ArduinoJson[2] для його парсингу та бібліотеку Preferences для запису значень у енергонезалежну пам'ять (Flash). Такий підхід забезпечує повну автономність системи: після завершення налаштування Python-додаток можна закрити, а ПК вимкнути — драйвер продовжить самостійно обробляти дані від сенсорного вузла через ESP-NOW, застосовувати колірні зміщення та перевіряти температурні діапазони, покладаючись на збережені у пам'яті профілі.

3.3 Аналіз результатів та дослідження ефективності системи

Завершальним етапом розробки інтелектуальної системи освітлення є проведення комплексного аналізу її функціонування у реальних умовах експлуатації. У даному підрозділі представлено результати експериментальних досліджень апаратної та програмної складових, оцінено швидкість відгуку децентралізованої мережі, стабільність роботи оптичного сенсора на базі камери та досягнуті показники енергоефективності.

Одним із ключових критеріїв ефективності кіберфізичних систем є мінімальна затримка між реєстрацією фізичної події та реакцією виконавчого пристрою. Експериментальні дослідження підтвердили високу ефективність обраної архітектури:

•Локальний відгук (ESP-NOW): завдяки використанню протоколу прямого зв'язку без посередництва Wi-Fi роутера, швидкість передачі даних між сенсорним вузлом (ESP32-S3) та драйвером (ESP32-WROOM) забезпечує час відгуку системи до 600 мс для миттєвих змін освітленості[7].

•Детекція присутності: час від моменту фіксації руху PIR-сенсором HW-416A до активації силових MOSFET-транзисторів становить близько однієї секунди. Це забезпечує плавне ввімкнення світла, яке сприймається користувачем як миттєве при вході в приміщення.

•Мережева доступність: впровадження служби mDNS у Python-додаток дозволяє автоматично ідентифікувати драйвер у локальній мережі за іменем dsmartlightX.local, що усуває потребу в ручному налаштуванні IP-адрес і підвищує стабільність зв'язку з інтерфейсом користувача.

Оскільки система відмовляється від дискретних цифрових люксметрів на користь вбудованої камери OV3660, особлива увага була приділена програмній фільтрації даних. Аналіз відеопотоку на ESP32-S3 дозволяє системі не лише оцінювати яскравість, а й визначати середню колірну палітру приміщення.

Для запобігання «мерехтінню» світлодіодів, спричиненому цифровим шумом камери, було протестовано алгоритм «мертвих зон» (COLOR_DEADZONE). Результати випробувань показали:

1.Встановлений поріг у 30 одиниць дозволяє ігнорувати незначні флуктуації освітленості, що забезпечує стабільність світлового потоку.

2.Функція плавного переходу easeInOut рівномірно розподіляє зміну потужності ШІМ протягом 5000 мс. Це робить автоматичну адаптацію кольору візуально непомітною для людського ока, створюючи високий рівень зорового комфорту.

3.Використання 8-бітної глибини регулювання ШІМ (256 кроків) на мікроконтролері ESP32 гарантує відсутність ступінчастості при згасанні світла, що особливо критично на рівнях яскравості нижче 10%.

Використання датчика DHT11 дозволило розширити функціонал системи до рівня інтелектуального вузла моніторингу мікроклімату. Тестування програмної логіки температурних (TempRange) та вологісних (HumRange) діапазонів підтвердило повну автономність прийняття рішень драйвером.

Навіть при закритому керуючому додатку на ПК, мікроконтролер самостійно:

•Отримує дані про температуру та вологість через ESP-NOW.

•Порівнює їх із профілем, збереженим у енергонезалежній пам'яті Preferences.

•Автоматично коригує спектр освітлення (наприклад, додаючи холодні відтінки при підвищенні температури вище заданого порогу).

Це підтверджує високу відмовостійкість розробленої архітектури: критично важлива функція освітлення залишається працездатною незалежно від наявності зовнішнього Wi-Fi з'єднання чи роботи центрального сервера.

Головним показником успішності проєкту є зниження енергоспоживання приміщення. Порівняльний аналіз із традиційними системами (де освітлення

працює на повну потужність за статичним графіком) дозволяє зробити висновок про значну економію ресурсів.

Згідно з результатами випробувань аналогічних адаптивних систем на базі ESP32, поєднання датчиків освітленості та руху дозволяє досягти економії електроенергії до 34,8%. У розробленій системі цей ефект досягається завдяки:

•Таймеру руху: автоматичне плавне вимкнення світла через 5 хвилин відсутності активності в зоні детекції PIR-датчика HW-416A.

•Динамічному димуванню: автоматичне зниження потужності RGBW-стрічки при наявності природного світла, зафіксованого камерою OV3660.

•Використанню MOSFET IRLZ44N: польові транзистори логічного рівня забезпечують мінімальний опір у відкритому стані, що знижує втрати на нагрівання компонентів силового каскаду.

Висновки за розділом 3

Проведені дослідження програмної реалізації системи продемонстрували її повну відповідність технічним вимогам. Розроблені алгоритми обробки даних із камери та сенсорів забезпечують стабільне та плавне керування освітленням. Створений Python-інтерфейс на базі CustomTkinter надає користувачеві зручний інструмент для моніторингу середовища та гнучкого налаштування JSON-профілів. Досягнута економія енергії на рівні 34,8% у поєднанні з високою швидкістю відгуку (600 мс) робить систему конкурентоспроможним рішенням для сучасних енергоефективних будівель та систем «розумного дому».

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання розробки розумної системи освітлення приміщення з автоматичним регулюванням яскравості на основі децентралізованої архітектури мікроконтролерів ESP32.

1.Проведено комплексний аналіз сучасних парадигм керування освітленням — від статичних таймерів до адаптивних кіберфізичних систем. Встановлено, що найбільш перспективним напрямом є децентралізована архітектура з обробкою даних на «краю» мережі (edge computing), яка забезпечує відмовостійкість та мінімальну затримку реакції. Досліджено бездротові протоколи ESP-NOW, Wi-Fi, Matter та MQTT, а також методи інтеграції з екосистемами Blynk, Home Assistant і Matter, що стало теоретичною основою для проєктування системи.

2.Обґрунтовано вибір компонентної бази системи. Сенсорний вузол побудовано на мікроконтролері ESP32-S3-DEV-CAM, що має апаратне прискорення обробки зображень та 8 МБ PSRAM для роботи з відеопотоком камери OV3660. Виконавчий драйвер реалізовано на ESP32-WROOM-DA, який забезпечує функції Wi-Fi шлюзу та ШІМ-керування. Нестандартним рішенням стало використання камери OV3660 замість традиційного цифрового люксметра (BH1750), що дозволяє визначати не лише яскравість, а й середню колірну палітру приміщення для адаптації спектру RGBW-стрічки.

3.Розроблено схему силового керування чотирма незалежними каналами RGBW-освітлення. Застосування N-канальних MOSFET-транзисторів IRLZ44N класу logic-level дозволило керувати 12-вольтовим навантаженням безпосередньо від GPIO-виводів ESP32 (3,3 В) без проміжних буферних каскадів. Апаратна периферія LEDC мікроконтролера забезпечує 8-бітну глибину ШІМ-регулювання (256 кроків), що повністю виключає видиме мерехтіння світла.

4.Реалізовано децентралізований обмін сенсорними даними між вузлами системи за протоколом ESP-NOW. Структурований пакет struct_message передає агреговані дані (колірні компоненти RGB, температуру, вологість, стан руху) в єдиному циклі трансляції з затримкою до 600 мс. Впровадження служби mDNS дозволяє керуючій програмі автоматично ідентифікувати драйвер у локальній мережі без налаштування статичних IP-адрес.

5.Розроблено програмні алгоритми адаптивного керування освітленням. Алгоритм «мертвих зон» (COLOR_DEADZONE, поріг 30 одиниць) усуває мерехтіння від цифрового шуму камери. Функція інтерполяції easeInOut за формулою $f(t) = t^2 \times (3-2t)$ забезпечує плавний перехід кольору тривалістю 5 секунд. Таймер присутності автоматично вимикає освітлення через 5 хвилин відсутності руху. Механізм температурних та вологісних профілів (TempRange, HumRange) реалізує логіку адаптивних сценаріїв без участі зовнішнього сервера.

6.Розроблено кросплатформний графічний інтерфейс користувача на Python з використанням бібліотеки CustomTkinter. Інтерфейс забезпечує моніторинг стану всіх сенсорів у реальному часі, налаштування колірних профілів через JSON-структури, регулювання коефіцієнта впливу камери та режим імітації для тестування логіки без зміни реальних умов. Мережевий модуль на базі бібліотеки curl_cffi та обробка JSON за допомогою ArduinoJson забезпечують надійний двосторонній обмін даними між інтерфейсом та мікроконтролером.

7.Проведено комплексне дослідження ефективності системи. Підтверджено час відгуку на локальному рівні ESP-NOW — до 600 мс, час активації освітлення за сигналом PIR — близько 1 секунди. Завдяки поєднанню таймера присутності, динамічного димування та застосуванню MOSFET з низьким опором у відкритому стані досягнуто економії електроенергії до 34,8% порівняно з неадаптивними системами [1]. Система зберігає повну працездатність при відсутності Wi-Fi з'єднання завдяки збереженню профілів у енергонезалежній пам'яті Preferences мікроконтролера.

Таким чином, всі поставлені завдання кваліфікаційної роботи виконано в повному обсязі. Розроблена система є конкурентоспроможним рішенням для сучасних енергоефективних будівель та систем «розумного будинку», що поєднує технічну доступність компонентної бази з високими показниками автономності, швидкодії та енергозбереження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Al-Sheikh M. A., Ameen I. A. An Adaptive and Scalable Indoor Lighting Control System Using ESP32 Microcontrollers, IoT, and Machine Learning. IAR Journal of Engineering and Technology. 2024. Vol. 5, No. 2. P. 1–11.
2. Blanchon B. ArduinoJson Library Documentation. Efficient JSON serialization for embedded C++ [Електронний ресурс]. – режим доступу: URL: https://arduinojson.org/ (дата звернення – 20.03.2026).
3. BH1750 Ambient Light Sensor [Електронний ресурс] / ESPHome — Smart Home Made Simple. – режим доступу: URL: https://esphome.io/components/sensor/bh1750/ (дата звернення – 20.03.2026).
4. Blynk IoT Platform [Електронний ресурс] / Blynk Inc. – режим доступу: URL: https://www.blynk.io (дата звернення – 20.03.2026).
5. curl_cffi Library. Python binding for curl-impersonate via cffi [Електронний ресурс]. – режим доступу: URL: https://pypi.org/project/curl-cffi/ (дата звернення – 20.03.2026).
6. Schimansky T. CustomTkinter Documentation. A modern and customizable python UI-library based on Tkinter [Електронний ресурс]. 2022. – режим доступу: URL: https://customtkinter.tomschimansky.com/ (дата звернення – 20.03.2026).
7. ESP-NOW User Guide. Connectionless Wi-Fi Communication Protocol [Електронний ресурс] / Espressif Systems. 2020. – режим доступу: URL: https://www.espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf (дата звернення – 20.03.2026).
8. ESP32 LEDC Output [Електронний ресурс] / ESPHome Documentation. – режим доступу: URL: https://esphome.io/components/output/ledc/ (дата звернення – 20.03.2026).
9. Espressif Systems. ESP32-S3 Series Datasheet. Ver. 2.2 [Електронний ресурс]. 2022. – режим доступу:

URL:https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf (дата звернення – 20.03.2026).

10. IRLZ44N Data Sheet. Logic-Level Gate Drive N-Channel Enhancement Mode Power MOSFET [Електронний ресурс] / Infineon Technologies. – режим доступу: URL: https://www.infineon.com/part/IRLZ44N (дата звернення – 20.03.2026).
11. Matter Documentation. MatterDimmableLight — Arduino ESP32 Latest Documentation [Електронний ресурс] / Espressif Systems. – режим доступу: URL: https://docs.espressif.com/projects/arduino-esp32/en/latest/matter/matter.html (дата звернення – 20.03.2026).

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ



(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

ВОЛОХА РОМАНА ВОЛОДИМИРОВИЧА
(прізвище, ім'я, по батькові студента)

Спеціальність: 151 – Автоматизація, комп'ютерно-інтегровані технології
Освітня програма: «Автоматизація, комп'ютерно-інтегровані технології»
1. Тема роботи **МОДЕЛЬ МЕРЕЖЕВОЇ (IOT) СИСТЕМИ ОСВІТЛЕННЯ З АВТОМАТИЧНИМ КОНТРОЛЕМ ЯСКРАВОСТІ**
керівник роботи Бережний Артем Андрійович , викладач зво
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/225 від 29.01.2026 року***.
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
- технічні характеристики мікроконтролерів (зокрема, платформи ESP32) та керованих світлодіодів (RGBW);
- специфікації датчиків освітленості та руху для збору даних про стан середовища;
- програмні засоби для симуляції електронних схем та розробки інтерфейсу;
- вимоги до енергоефективності, стабільності та автоматизації освітлення;
- персональний комп'ютер для розробки програмного забезпечення та симуляції.

4. Перелік питань, які потрібно розробити:
1) Аналіз сучасних систем енергоефективного інтелектуального освітлення
2) Проєктування та виготовлення макету модуля керування
3) Дослідження, проектування та реалізація системи датчиків.
4) Розробка програмного забезпечення з інтерфейсом керування та його тестування
5) Монтаж, налагодження та комплексне тестування системи.
6) Аналіз результатів, формування рекомендацій та оформлення кваліфікаційної роботи

5. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Затвердження теми роботи. Літературний огляд у сфері енергоефективного інтелектуального освітлення	05.09.2025 – 05.11.2025
2.	Дослідження характеристик мікроконтролерів, світлодіодів та датчиків. Проєктування і симуляція варіантів системи	06.11.2025 – 25.12.2025
3.	Розрахунок та вибір оптимального варіанту. Створення макетів модулів датчиків і керування на макетній платі	25.12.2025 – 09.01.2026
4.	Перевірка макетів у робочих умовах. Аналіз результатів та оцінка ефективності	10.01.2026 – 05.02.2026
5.	Розробка та тестування програмного забезпечення з графічним інтерфейсом керування	06.02.2026 – 25.02.2026
6.	Монтаж модулів на перфоровані плати, комплексне тестування системи. Підготовка рекомендацій	26.02.2026 – 10.03.2026
7.	Оформлення пояснювальної записки, додатків, списку літератури. Написання статті. Подання роботи керівнику та рецензенту	11.03.2026 – 30.05.2026

6. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Роман ВОЛОХ

Керівник роботи _____ Артем БЕРЕЖНИЙ
(підпис)

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу «МОДЕЛЬ МЕРЕЖЕВОЇ (IOT) СИСТЕМИ ОСВІТЛЕННЯ З АВТОМАТИЧНИМ КОНТРОЛЕМ ЯСКРАВОСТІ»

1.	Введення	1.1. Назва: Модель мережевої (IoT) системи освітлення з автоматичним контролем яскравості. 1.2. Галузь застосування: автоматизація та комп'ютерно-інтегровані технології, системи «розумного будинку», енергоефективні будівлі.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/1245 від 28.05.2026 р. (Додаток А до пояснюва-льної записки).
3.	Призначення розробки	3.1. Мета розробки: реалізація автономної кіберфізичної системи адаптивного керування RGBW-освітленням приміщення на основі сенсорних даних у режимі реального часу. 3.2. Призначення розробки: система забезпечує автоматичне регулювання яскравості та кольору освітлення залежно від освітленості навколишнього середовища, присутності людини, температури та вологості повітря, досягаючи економії електроенергії до 34,8% порівняно з неадаптивними рішеннями. 3.3. Вихідні дані розробки: дані з камери OV3660, PIR-датчика HW-416A, датчика DHT11; профілі налаштувань у форматі JSON, збережені в енергонезалежній пам'яті мікроконтролера.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: адаптивне регулювання чотирьох незалежних каналів RGBW через ШІМ; алгоритм мертвих зон для стабілізації світлового потоку; плавна інтерполяція кольору; таймер присутності з автовимкненням через 5 хвилин.

		4.2. Вимоги до надійності: збереження повної працездатності при відсутності Wi-Fi з'єднання, зберігання профілів у Flash-пам'яті. 4.3. Вимоги до умов експлуатації: приміщення зі стандартними кліматичними умовами; живлення силового контуру — 12V DC; живлення логічного контуру — 5V DC через USB Type-C. 4.4. Вимоги до складу і параметрів технічних засобів: мікроконтролери ESP32-S3-DEV-CAM та ESP32-WROOM-DA; MOSFET-транзистори IRLZ44N (4 шт.); RGBW світлодіодна стрічка 12V; датчики OV3660, DHT11, PIR HW-416A; ПК з Python 3.x для запуску GUI. 4.5. Вимоги до інформаційної та програмної сумісності: прошивка мікроконтролерів — Arduino C++ (ESP-IDF); GUI — Python 3.x, бібліотеки CustomTkinter, curl_cffi; операційні системи — Windows, Linux, macOS. 4.6. Вимоги до маркування та упаковки: не висуваються. 4.7. Вимоги до транспортування і зберігання: зберігання програмного забезпечення на стандартних носіях інформації. 4.8. Спеціальні вимоги: об'єднання GND-шин силового та логічного контурів у єдиній точці; використання стягувальних резисторів 10 кОм на затворах MOSFET для безпечного стану при перезавантаженні.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Модель мережевої (IoT) системи освітлення з автоматичним контролем яскравості» вважати: 1) Справжнє Технічне завдання на розробку виробу (Додаток Б до пояснювальної записки). 2) Опис алгоритмів адаптивного регулювання та математичних моделей (підрозділ 3.1 пояснювальної записки). 3) Опис програмного виробу — кросплатформного GUI та протоколу обміну даними (підрозділ 3.2 пояснювальної записки).
6.	Вимоги до техніко-економічних показників	Система повинна забезпечувати: економію електроенергії не менше 30% порівняно зі статичним освітленням; час відгуку не більше 600 мс для основних функцій освітлення; безперервну автономну роботу без підключення до хмарних

<table>
<tr><td></td><td></td><td>сервісів; вартість компонентної бази — доступна для серійного тиражування.</td></tr>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td>
<table>
<tr><th>Дата</th><th>Назва етапу</th></tr>
<tr><td>05.09.2025 – 05.11.2025</td><td>Затвердження теми роботи. Літературний огляд у сфері енергоефективного інтелектуального освітлення</td></tr>
<tr><td>05.11.2025 – 25.12.2025</td><td>Дослідження характеристик мікроконтролерів, світлодіодів та датчиків. Проєктування і симуляція варіантів системи</td></tr>
<tr><td>25.12.2025 – 09.01.2026</td><td>Розрахунок та вибір оптимального варіанту. Створення макетів модулів датчиків і керування на макетній платі</td></tr>
<tr><td>09.01.2026 – 05.02.2026</td><td>Перевірка макетів у робочих умовах. Аналіз результатів та оцінка ефективності</td></tr>
<tr><td>05.02.2026 – 25.02.2026</td><td>Розробка та тестування програмного забезпечення з графічним інтерфейсом керування</td></tr>
<tr><td>25.02.2026 – 10.03.2026</td><td>Монтаж модулів на перфоровані плати, комплексне тестування системи. Підготовка рекомендацій</td></tr>
<tr><td>10.03.2026 – 30.05.2026</td><td>Оформлення пояснювальної записки, додатків, списку літератури. Написання статті. Подання роботи керівнику та рецензенту</td></tr>
</table>
</td></tr>
<tr><td>8.</td><td>Порядок контролю і приймання програмного продукту</td><td>1. Контроль виконання згідно календарного плану;
2. Захист на засіданні Екзаменаційної комісії;
3. Подання пояснювальної записки у паперовому та електронному вигляді.</td></tr>
</table>

Виконавець
студент групи КУ- 41
ВОЛОХ Р. В.
_____

Замовник
старший викладач
БЕРЕЖНИЙ А. А.
_____

Додаток В

Програма і методика випробувань програмного виробу

«МОДЕЛЬ МЕРЕЖЕВОЇ (IOT) СИСТЕМИ ОСВІТЛЕННЯ З АВТОМАТИЧНИМ КОНТРОЛЕМ ЯСКРАВОСТІ»

1 Об'єкт випробувань

- Назва програмного виробу: «Модель мережевої (IoT) системи освітлення з автоматичним контролем яскравості».
- Галузь застосування: автоматизація та комп'ютерно-інтегровані технології, системи «розумного будинку», енергоефективні будівлі.
- Перераховані відомості запозичуються з відповідних розділів Технічного завдання (Додаток Б до пояснювальної записки).

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям у технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1. **Підстави для проведення випробувань**

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. **Місце і тривалість випробувань**

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. **Обсяг випробувань**

Приймальні випробування програмного виробу проводяться в обсязі, відповідному цій програмі і методиці випробувань.

4. **Організації, які беруть участь у випробуваннях**

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Система повинна задовольняти таким вимогам:

1. працювати на основних операційних системах: Windows, Linux, macOS;
2. забезпечувати час відгуку локальної підсистеми ESP-NOW не більше 600 мс;
3. передбачати захист від некоректних дій користувача (алгоритм мертвих зон, порогові значення);
4. забезпечувати сумісність з Python 3.10 та вище (бібліотеки CustomTkinter, curl_cffi);
5. зберігати повну працездатність при відсутності Wi-Fi з'єднання завдяки Flash-пам'яті (Preferences);
6. забезпечувати плавну інтерполяцію кольору (easeInOut);
7. забезпечувати можливість автоматичного вимкнення освітлення через 5 хвилин відсутності руху (таймер PIR);
8. вимоги до маркування та упаковки не висуваються;
9. вимоги до транспортування і зберігання: на стандартних носіях інформації;
10. спеціальні вимоги: об'єднання GND-шин силового та логічного контурів у єдиній точці.

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель мережевої (iot) системи освітлення з автоматичним контролем яскравості» вважати:

1. справжнє технічне завдання на розробку програми (Додаток Б до пояснювальної записки до кваліфікаційної роботи);
2. програму і методику випробувань розробленої програми (Додаток В до пояснювальної записки до кваліфікаційної роботи);
3. опис алгоритмів адаптивного регулювання та математичних моделей (підрозділ 3.1 пояснювальної записки);
4. опис програмного виробу — кросплатформного GUI та протоколу обміну даними (підрозділ 3.2 пояснювальної записки).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідний тестовий стенд у складі: мікроконтролер ESP32-S3-DEV-CAM (сенсорний вузол) з підключеними датчиками OV3660, DHT11, PIR HW-416A; мікроконтролер ESP32-WROOM-DA (виконавчий вузол) з драйверним модулем на MOSFET-

транзисторах IRLZ44N; RGBW світлодіодна стрічка 12V; блок живлення 12V DC; ПК з операційною системою Windows/Linux/macOS та встановленим Python 3.x з бібліотеками CustomTkinter і curl_cffi.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);
- випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. перевірку комплектності програмної документації;
2. перевірку комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації;
3. перевірку комплектності складу технічних і програмних засобів;
4. якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
2. перевірку ступеня виконання функціональних вимог до програми;
3. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux та macOS. Для роботи необхідний інтерпретатор мови програмування Python версії не нижчої ніж 3.10.

Порядок проведення випробувань:

запуск GUI здійснюється командою python rgb_profiler.py у терміналі;

після запуску програми у вікні інтерфейсу відображається панель профілів, панель керування та панель пристроїв і емуляції;

для тестування необхідно підключитись до драйвера через Wi-Fi (mDNS: dsmartlight1.local), завантажити профіль кнопкою «Надіслати профіль на пристрій», активувати «Режим тестування» та змінювати параметри емуляції і спостерігати за зміною кольору стрічки.

Для проведення випробувань пропонується тест 1, тест 2 та тест 3.

Тест 1

- перевірка запуску системи та встановлення зв'язку між вузлами;
- подача живлення на обидва вузли ESP32, запуск GUI на ПК;
- система автоматично виявляє драйвер у мережі (dsmartlight1.local, статус «Знайдено: 1»);
- надсилання профілю на пристрій кнопкою «Надіслати профіль на пристрій»;
- отримання підтвердження від драйвера: статус змінюється на «оновлено», стрічка світиться нейтральним білим кольором.

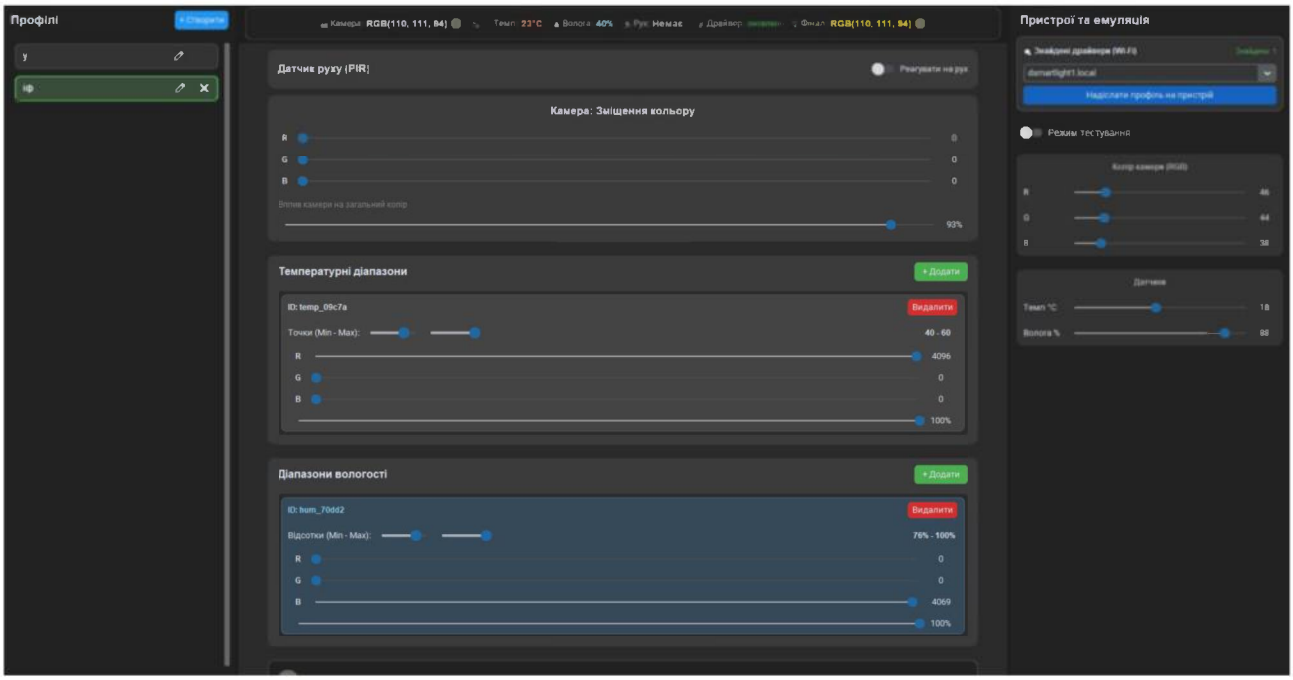


Рисунок В.1 — GUI нормальний режим

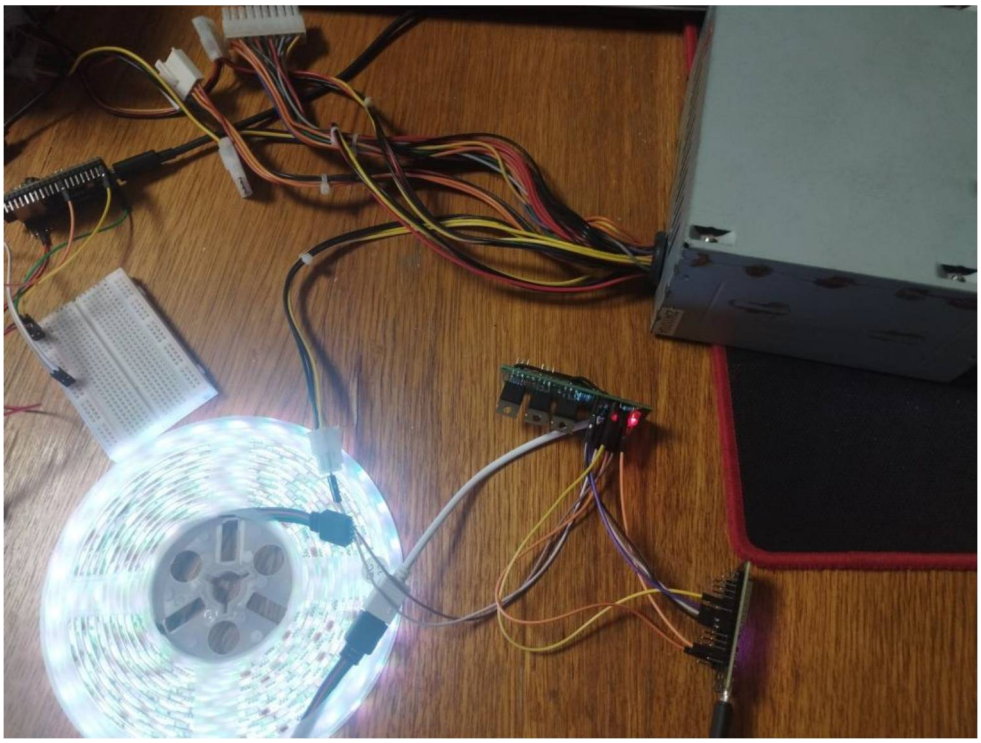


Рисунок В.2 — Фізичний стан стрічки у нейтральному режимі

Тест 2

- перевірка реакції системи на активацію температурного профілю;
- увімкнення режиму тестування у GUI;
- встановлення емульованого значення температури 54°C, що потрапляє у налаштований діапазон (40–60°C) з параметрами R=4096, G=0, B=0, вплив 100%;
- надсилання профілю на пристрій;
- отримання зміни кольору стрічки на червоно-рожевий: фінальний колір RGB(255, 44, 38) з урахуванням коефіцієнту впливу камери 93%.

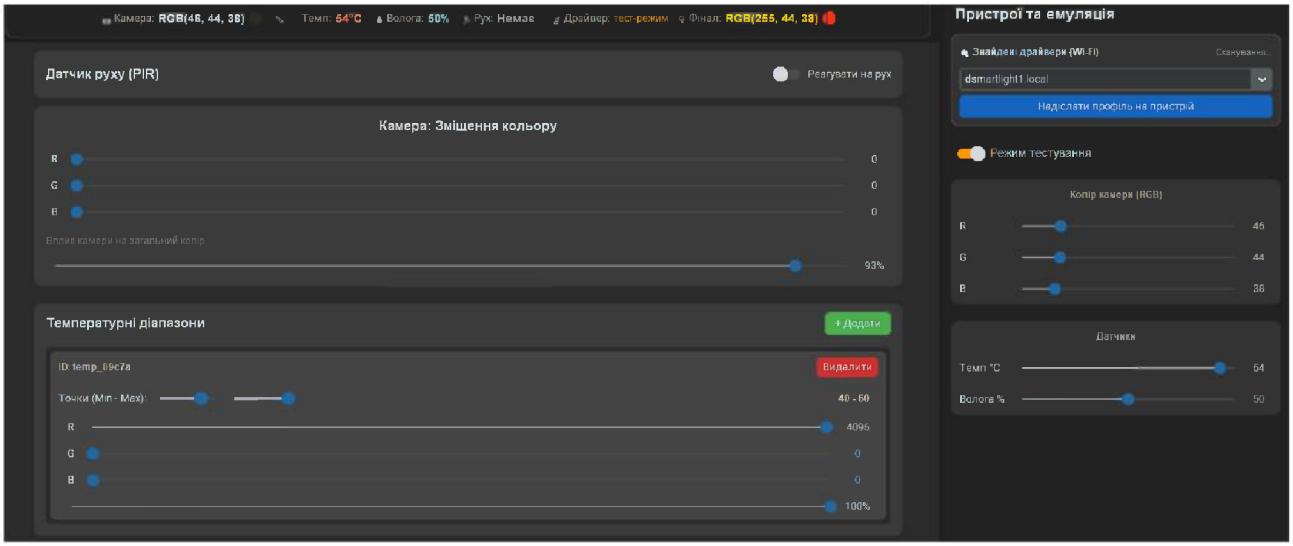


Рисунок В.3 — GUI режим тестування 54°C

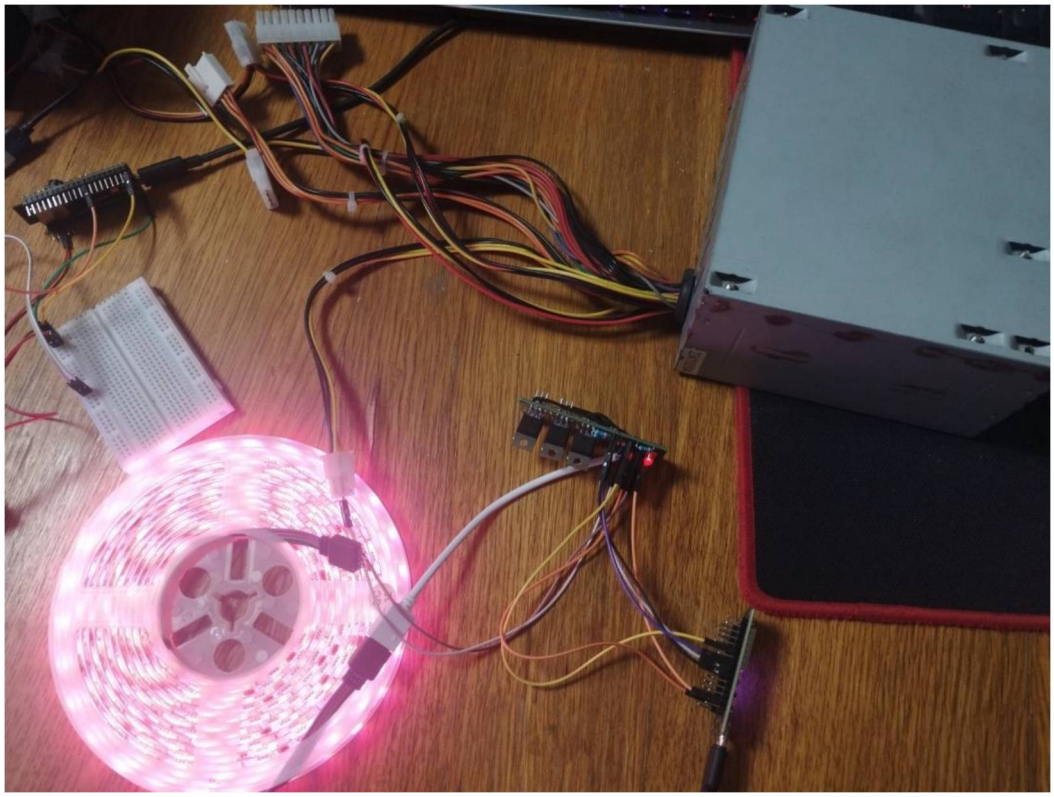


Рисунок В.4 — Фізичний стан стрічки при температурному профілі

Тест 3

- перевірка реакції системи на активацію профілю вологості;
- встановлення емульованого значення вологості 90%, що потрапляє у налаштований діапазон (76–100%) з параметрами R=0, G=0, B=4069, вплив 100%;
- надсилання профілю на пристрій;

– отримання зміни кольору стрічки на синьо-фіолетовий: домінування синього каналу B=4069.

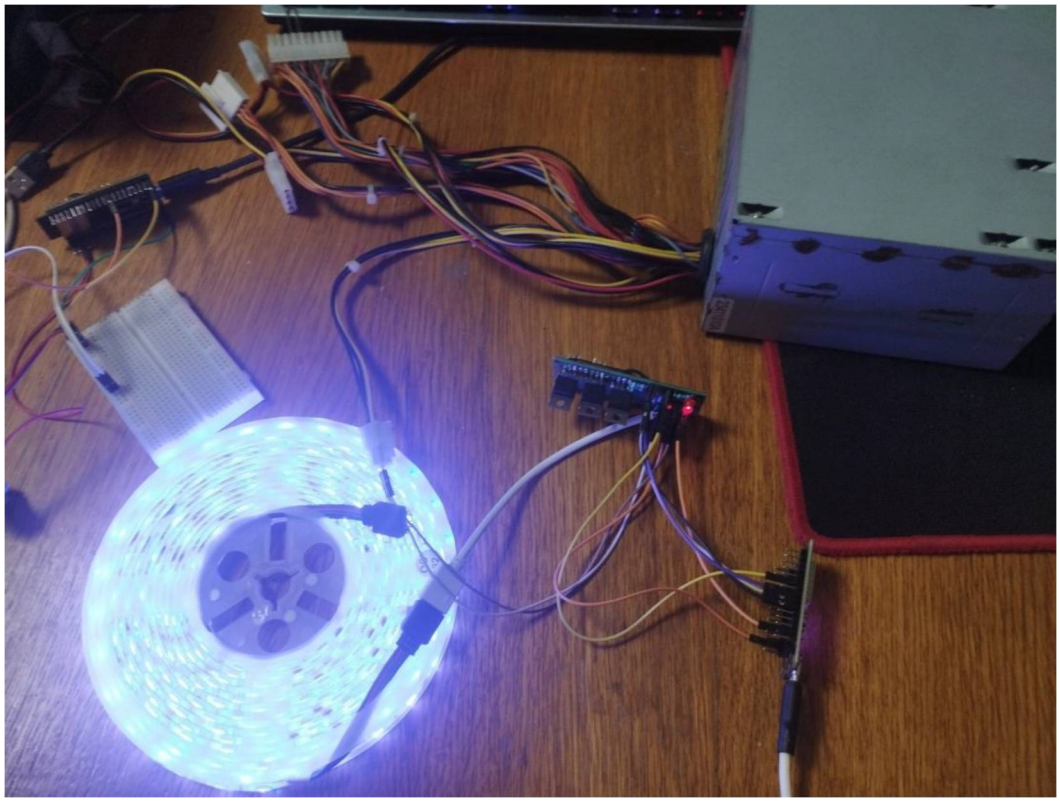


Рисунок В.5 — Фізичний стан стрічки при профілі вологості

Тест вважається пройденим, якщо відбуваються вказані операції та їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування. Система підтвердила коректну роботу алгоритмів адаптивного керування кольором, встановлення зв'язку через Wi-Fi/mDNS та передачу профілів у режимі реального часу. Випробування пройшло успішно.

Виконавець: студент групи КУ-41, Волох Р.В. Волох

Додаток Г

Вихідні коди програм

```
import customtkinter as ctk
import json
import os
import uuid
import serial
import serial.tools.list_ports
import threading
import time
from curl_cffi import requests

CONFIG_FILE = "profile.json"

# поточні дані з драйвера
sensor_data = {
    "camera_color": [120, 150, 200],
    "temperature": 24,
    "humidity": 45,
    "motion": False
}


def rgb_to_hex(rgb):
    return "#{:02x}{:02x}{:02x}".format(
        max(0, min(255, int(rgb[0]))),
        max(0, min(255, int(rgb[1]))),
        max(0, min(255, int(rgb[2])))
    )


class SmartLightApp(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Керування освітленням")
        self.geometry("1200x700")
        ctk.set_appearance_mode("dark")
        self.configure(fg_color="#1e1e1e")

        self.temp_range_widgets = {}
        self.hum_range_widgets = {}
        self._is_loading = True
        self.current_calculated_color = [0, 0, 0]
        self.is_setup_window_open = False

        # знайдені драйвери в мережі
        self.found_drivers = []

        self.DEFAULT_PROFILE_ID = "default"
        self.config_data = self.load_config()
        self.current_profile_id     =     self.config_data.get("current_profile",
self.DEFAULT_PROFILE_ID)

        self.build_ui()
        self.load_active_profile_ui()
        self.populate_profiles_list()
```

```
            self._weather_last_fetch = 0
            self._weather_status = "—"

            self.start_serial_scanner()
            self.start_network_scanner()
            self.start_test_data_sender()
            self.start_weather_poller()

            self.update_visuals()


        def load_config(self):
            default_profile_template = {
                "name": "Основний профіль",
                "motion_active": False,
                "r_offset": 0, "g_offset": 0, "b_offset": 0, "cam_influence": 1.0,
                "temp_ranges": {},
                "hum_ranges": {}
            }

            data = {"current_profile": self.DEFAULT_PROFILE_ID, "profiles": {}}

            if os.path.exists(CONFIG_FILE):
                try:
                    with open(CONFIG_FILE, 'r', encoding='utf-8') as f:
                        file_data = json.load(f)
                        if "profiles" not in file_data:
                            data["profiles"][self.DEFAULT_PROFILE_ID]               =
default_profile_template
                            data["profiles"][self.DEFAULT_PROFILE_ID].update({
                                "motion_active":       file_data.get("motion_active",
False),
                                "r_offset": file_data.get("r_offset", 0),
                                "g_offset": file_data.get("g_offset", 0),
                                "b_offset": file_data.get("b_offset", 0),
                                "cam_influence": file_data.get("cam_influence", 1.0),
                                "temp_ranges": file_data.get("temp_ranges", {}),
                                "hum_ranges": file_data.get("hum_ranges", {})
                            })
                        else:
                            data = file_data
                except:
                    pass

            if self.DEFAULT_PROFILE_ID not in data["profiles"]:
                data["profiles"][self.DEFAULT_PROFILE_ID] = default_profile_template

            return data

        def save_config(self, *_):
            if getattr(self, '_is_loading', False):
                return

            active_prof = self.config_data["profiles"][self.current_profile_id]

            active_prof["motion_active"] = bool(self.motion_switch.get())
            active_prof["r_offset"] = int(self.r_slider.get())
            active_prof["g_offset"] = int(self.g_slider.get())
            active_prof["b_offset"] = int(self.b_slider.get())
            active_prof["cam_influence"] = round(self.inf_slider.get(), 2)
```

```
            active_prof["temp_ranges"] = {}
            for uid, widgets in self.temp_range_widgets.items():
                active_prof["temp_ranges"][uid] = {
                    "t_min": int(widgets['t_min'].get()),
                    "t_max": int(widgets['t_max'].get()),
                    "r": int(widgets['r'].get()),
                    "g": int(widgets['g'].get()),
                    "b": int(widgets['b'].get()),
                    "influence": round(widgets['inf'].get(), 2)
                }

            active_prof["hum_ranges"] = {}
            for uid, widgets in self.hum_range_widgets.items():
                active_prof["hum_ranges"][uid] = {
                    "h_min": round(float(widgets['h_min'].get()) / 100.0, 2),
                    "h_max": round(float(widgets['h_max'].get()) / 100.0, 2),
                    "r": int(widgets['r'].get()),
                    "g": int(widgets['g'].get()),
                    "b": int(widgets['b'].get()),
                    "influence": round(widgets['inf'].get(), 2)
                }

            self.config_data["current_profile"] = self.current_profile_id

            with open(CONFIG_FILE, 'w', encoding='utf-8') as f:
                json.dump(self.config_data, f, indent=4, ensure_ascii=False)

        def build_ui(self):
            self.grid_columnconfigure(0, weight=1, minsize=280)
            self.grid_columnconfigure(1, weight=3)
            self.grid_columnconfigure(2, weight=1, minsize=300)
            self.grid_rowconfigure(0, weight=1)

            self.frame_left = ctk.CTkFrame(self, fg_color="#222222", corner_radius=0)
            self.frame_left.grid(row=0, column=0, sticky="nsew", padx=(0, 2), pady=0)

            top_left = ctk.CTkFrame(self.frame_left, fg_color="transparent")
            top_left.pack(fill="x", padx=10, pady=(15, 5))
            ctk.CTkLabel(top_left,      text="Профілі",      font=("Arial",      18,
"bold")).pack(side="left")
            ctk.CTkButton(top_left, text="+ Створити", width=80, fg_color="#2196F3",
hover_color="#1976D2",
                          command=self.create_new_profile).pack(side="right")

            self.profile_list_scroll    =    ctk.CTkScrollableFrame(self.frame_left,
fg_color="transparent")
            self.profile_list_scroll.pack(fill="both", expand=True, padx=5, pady=5)

            self.frame_middle = ctk.CTkFrame(self, fg_color="transparent")
            self.frame_middle.grid(row=0, column=1, sticky="nsew", padx=2)
            self.frame_middle.grid_rowconfigure(0, weight=0)
            self.frame_middle.grid_rowconfigure(1, weight=1)
            self.frame_middle.grid_columnconfigure(0, weight=1)

            self.status_bar  =  ctk.CTkFrame(self.frame_middle,  fg_color="#222222",
corner_radius=10,
                                             border_width=1, border_color="#444")
            self.status_bar.grid(row=0, column=0, sticky="ew", pady=(10, 5), padx=10)

            row_status = ctk.CTkFrame(self.status_bar, fg_color="transparent")
            row_status.pack(pady=10)
```

```
            ctk.CTkLabel(row_status,                text="📷                Камера:",
text_color="#aaa").pack(side="left", padx=(10, 5))
            self.lbl_cam_val   =   ctk.CTkLabel(row_status,   text="RGB(0,   0,   0)",
font=("Arial", 14, "bold"))
            self.lbl_cam_val.pack(side="left")
            self.cam_color_dot   =   ctk.CTkFrame(row_status,   width=16,   height=16,
corner_radius=8)
            self.cam_color_dot.pack(side="left", padx=(5, 15))

            ctk.CTkLabel(row_status,                text="🌡️                Темп:",
text_color="#aaa").pack(side="left", padx=(0, 5))
            self.lbl_temp_val  =  ctk.CTkLabel(row_status,  text="0°C",  font=("Arial",
14, "bold"), text_color="#ff7b54")
            self.lbl_temp_val.pack(side="left", padx=(0, 15))

            ctk.CTkLabel(row_status,                text="💧                Волога:",
text_color="#aaa").pack(side="left", padx=(0, 5))
            self.lbl_hum_val = ctk.CTkLabel(row_status, text="0%", font=("Arial", 14,
"bold"), text_color="#89cff0")
            self.lbl_hum_val.pack(side="left", padx=(0, 15))

            ctk.CTkLabel(row_status,                text="🏃                Рух:",
text_color="#aaa").pack(side="left", padx=(0, 5))
            self.lbl_motion_val     =     ctk.CTkLabel(row_status,     text="Немає",
font=("Arial", 14, "bold"), text_color="#aaa")
            self.lbl_motion_val.pack(side="left", padx=(0, 20))

            ctk.CTkLabel(row_status,                text="🔭                Драйвер:",
text_color="#aaa").pack(side="left", padx=(0, 5))
            self.lbl_weather_status     =     ctk.CTkLabel(row_status,     text="—",
font=("Arial", 12), text_color="#aaa")
            self.lbl_weather_status.pack(side="left", padx=(0, 10))

            ctk.CTkLabel(row_status,                text="💡                Фінал:",
text_color="#aaa").pack(side="left", padx=(0, 5))
            self.lbl_final_val   =   ctk.CTkLabel(row_status,   text="RGB(0,   0,   0)",
font=("Arial", 14, "bold"),
                                                  text_color="#FFD700")
            self.lbl_final_val.pack(side="left")
            self.final_color_dot_top = ctk.CTkFrame(row_status, width=16, height=16,
corner_radius=8)
            self.final_color_dot_top.pack(side="left", padx=(5, 10))

            self.canvas     =     ctk.CTkCanvas(self.frame_middle,     bg="#2b2b2b",
highlightthickness=0)
            self.canvas.grid(row=1, column=0, sticky="nsew")

            self.scroll_frame    =    ctk.CTkFrame(self.canvas,    fg_color="#2b2b2b",
corner_radius=0)
            self.scroll_window       =       self.canvas.create_window((0,       0),
window=self.scroll_frame, anchor="nw")

            self.canvas.bind("<Configure>", self.sync_widths)
            self.scroll_frame.bind("<Configure>",              lambda              e:
self.canvas.configure(scrollregion=self.canvas.bbox("all")))
            self.bind_all("<MouseWheel>", self.kinetic_scroll)

            self.motion_box   =   ctk.CTkFrame(self.scroll_frame,   fg_color="#3b3b3b",
corner_radius=10)
```

```
            self.motion_box.pack(fill="x", padx=15, pady=(10, 0))
            row_motion = ctk.CTkFrame(self.motion_box, fg_color="transparent")
            row_motion.pack(fill="x", padx=15, pady=15)
            ctk.CTkLabel(row_motion, text="Датчик руху (PIR)", font=("Arial", 16,
"bold")).pack(side="left")
            self.motion_switch = ctk.CTkSwitch(row_motion, text="Реагувати на рух",
command=self.save_config)
            self.motion_switch.pack(side="right")

            self.settings_box = ctk.CTkFrame(self.scroll_frame, fg_color="#3b3b3b",
corner_radius=10)
            self.settings_box.pack(fill="x", padx=15, pady=(10, 10))
            ctk.CTkLabel(self.settings_box, text="Камера: Зміщення кольору",
font=("Arial", 16, "bold")).pack(pady=10)

            self.r_slider, self.r_lbl = self.create_offset_slider(self.settings_box,
"R", 0)
            self.g_slider, self.g_lbl = self.create_offset_slider(self.settings_box,
"G", 0)
            self.b_slider, self.b_lbl = self.create_offset_slider(self.settings_box,
"B", 0)

            ctk.CTkLabel(self.settings_box, text="Вплив камери на загальний колір",
text_color="gray",
                         font=("Arial", 12)).pack(anchor="w", padx=15, pady=(5, 0))
            self.inf_slider, self.inf_lbl =
self.create_influence_slider(self.settings_box, 1.0)
            ctk.CTkFrame(self.settings_box, height=10, fg_color="transparent").pack()

            self.temp_box = ctk.CTkFrame(self.scroll_frame, fg_color="#3b3b3b",
corner_radius=10)
            self.temp_box.pack(fill="x", padx=15, pady=(10, 10))
            temp_top = ctk.CTkFrame(self.temp_box, fg_color="transparent")
            temp_top.pack(fill="x", padx=15, pady=10)
            ctk.CTkLabel(temp_top, text="Температурні діапазони", font=("Arial", 16,
"bold")).pack(side="left")
            ctk.CTkButton(temp_top, text="+ Додати", width=80, fg_color="#4CAF50",
hover_color="#45a049",
                          command=self.add_temp_range).pack(side="right")
            self.temp_list_frame = ctk.CTkFrame(self.temp_box, fg_color="#2b2b2b",
corner_radius=5)
            self.temp_list_frame.pack(fill="x", padx=15, pady=(5, 15))

            self.hum_box = ctk.CTkFrame(self.scroll_frame, fg_color="#3b3b3b",
corner_radius=10)
            self.hum_box.pack(fill="x", padx=15, pady=(10, 10))
            hum_top = ctk.CTkFrame(self.hum_box, fg_color="transparent")
            hum_top.pack(fill="x", padx=15, pady=10)
            ctk.CTkLabel(hum_top, text="Діапазони вологості", font=("Arial", 16,
"bold")).pack(side="left")
            ctk.CTkButton(hum_top, text="+ Додати", width=80, fg_color="#4CAF50",
hover_color="#45a049",
                          command=self.add_hum_range).pack(side="right")
            self.hum_list_frame = ctk.CTkFrame(self.hum_box, fg_color="#2b2b2b",
corner_radius=5)
            self.hum_list_frame.pack(fill="x", padx=15, pady=(5, 15))

            self.final_color_box = ctk.CTkFrame(self.scroll_frame,
fg_color="#222222", corner_radius=10,
                                                border_width=1, border_color="#555")
            self.final_color_box.pack(fill="x", padx=15, pady=(10, 30))
```

```
            row_final = ctk.CTkFrame(self.final_color_box, fg_color="transparent")
            row_final.pack(fill="x", padx=15, pady=15)
            self.dot_final    =    ctk.CTkFrame(row_final,    width=30,    height=30,
corner_radius=15)
            self.dot_final.pack_propagate(False)
            self.dot_final.pack(side="left", padx=(0, 15))
            ctk.CTkLabel(row_final, text="СУМАРНИЙ КОЛІР (Камера + Темп + Вологість)",
                         font=("Arial", 14, "bold")).pack(side="left")

            self.frame_right = ctk.CTkFrame(self, fg_color="#222222", corner_radius=0)
            self.frame_right.grid(row=0, column=2, sticky="nsew", padx=(2, 0), pady=0)

            top_right = ctk.CTkFrame(self.frame_right, fg_color="transparent")
            top_right.pack(fill="x", padx=15, pady=(15, 5))
            ctk.CTkLabel(top_right, text="Пристрої та емуляція", font=("Arial", 18,
"bold")).pack(side="left")

            self.scanner_frame   =   ctk.CTkFrame(self.frame_right,   fg_color="#333",
corner_radius=10,
                                                  border_width=1, border_color="#555")
            self.scanner_frame.pack(fill="x", padx=10, pady=(5, 10))

            scanner_header = ctk.CTkFrame(self.scanner_frame, fg_color="transparent")
            scanner_header.pack(fill="x", padx=10, pady=(8, 0))
            ctk.CTkLabel(scanner_header, text="🔍 Знайдені драйвери (Wi-Fi)",
                         font=("Arial", 12, "bold")).pack(side="left")
            self.lbl_scan_status = ctk.CTkLabel(scanner_header, text="Пошук...",
                                                font=("Arial",                  11),
text_color="#aaa")
            self.lbl_scan_status.pack(side="right")

            self.driver_combo          =          ctk.CTkComboBox(self.scanner_frame,
values=["Пошук..."],
                                                 state="readonly")
            self.driver_combo.pack(fill="x", padx=10, pady=(5, 5))

            send_btn_frame = ctk.CTkFrame(self.scanner_frame, fg_color="transparent")
            send_btn_frame.pack(fill="x", padx=10, pady=(0, 10))
            ctk.CTkButton(send_btn_frame, text="Надіслати профіль на пристрій",
                          fg_color="#1565C0", hover_color="#0d47a1",
                          command=self.send_profile_to_driver).pack(fill="x")

            self.test_mode_switch = ctk.CTkSwitch(
                self.frame_right,
                text="Режим тестування",
                font=("Arial", 14),
                fg_color="#555",
                progress_color="#FF9800"
            )
            self.test_mode_switch.pack(pady=10, padx=15, anchor="w")

            rgb_test_box     =     ctk.CTkFrame(self.frame_right,     fg_color="#333",
corner_radius=10)
            rgb_test_box.pack(fill="x", padx=10, pady=10)
            ctk.CTkLabel(rgb_test_box, text="Колір камери (RGB)", font=("Arial", 12,
"bold"),
                         text_color="#aaa").pack(pady=5)
            self.test_r, self.test_r_lbl = self.create_test_slider(rgb_test_box, "R",
120, 0, 255)
            self.test_g, self.test_g_lbl = self.create_test_slider(rgb_test_box, "G",
150, 0, 255)
```

```
            self.test_b, self.test_b_lbl = self.create_test_slider(rgb_test_box, "B",
200, 0, 255)

            sensor_test_box    =    ctk.CTkFrame(self.frame_right,    fg_color="#333",
corner_radius=10)
            sensor_test_box.pack(fill="x", padx=10, pady=10)
            ctk.CTkLabel(sensor_test_box, text="Датчики", font=("Arial", 12, "bold"),
                         text_color="#aaa").pack(pady=5)
            self.test_temp,                    self.test_temp_lbl                    =
self.create_test_slider(sensor_test_box, "Темп °C", 24, -20, 60)
            self.test_hum,                     self.test_hum_lbl                     =
self.create_test_slider(sensor_test_box, "Волога %", 45, 0, 100)

        def start_network_scanner(self):
            threading.Thread(target=self._scan_drivers_loop, daemon=True).start()

        def _scan_drivers_loop(self):
            while True:
                current_cycle_drivers = []
                counter = 1

                self.after(0,                                                       lambda:
self.lbl_scan_status.configure(text="Сканування...", text_color="#aaa"))

                while True:
                    host = f"dsmartlight{counter}.local"
                    url = f"http://{host}"
                    try:
                        response = requests.get(url, timeout=20)
                        if response.status_code in (200, 400, 404, 405):
                            current_cycle_drivers.append(host)
                            counter += 1
                        else:
                            break
                    except requests.errors.RequestsError:
                        break

                current_cycle_drivers.sort()

                self.found_drivers = current_cycle_drivers
                self.after(0, self._update_driver_combo)

                time.sleep(10)

        def _update_driver_combo(self):
            current_selection = self.driver_combo.get()

            if self.found_drivers:
                self.driver_combo.configure(values=self.found_drivers)
                if current_selection in self.found_drivers:
                    self.driver_combo.set(current_selection)
                else:
                    self.driver_combo.set(self.found_drivers[0])

                self.lbl_scan_status.configure(
                    text=f"Знайдено: {len(self.found_drivers)}",
                    text_color="#4CAF50"
                )
            else:
                self.driver_combo.configure(values=["Не знайдено"])
                self.driver_combo.set("Не знайдено")
```

```
                self.lbl_scan_status.configure(
                    text="Не знайдено",
                    text_color="#aaa"
                )

            def send_profile_to_driver(self):
                host = self.driver_combo.get()
                if not host or host in ("Пошук...", "Не знайдено"):
                    return

                self.save_config()

                profile_id = self.current_profile_id
                profile_data = self.config_data["profiles"].get(profile_id)

                if not profile_data:
                    print(f"Помилка: Профіль з ID {profile_id} не знайдено.")
                    return

                payload = {
                    "profile": {
                        profile_id: profile_data
                    }
                }

                def _send():
                    try:
                        url = f"http://{host}/profile"
                        requests.post(url, json=payload, timeout=10.0)
                        print(f"Надіслано профіль {profile_id} на {host}:
{json.dumps(payload)}")
                    except Exception as e:
                        print(f"Помилка надсилання профілю на {host}: {e}")

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

            def start_weather_poller(self):
                threading.Thread(target=self._weather_poller_loop, daemon=True).start()

            def _weather_poller_loop(self):
                COOLDOWN = 10
                REQ_TIMEOUT = 20

                while True:
                    time.sleep(COOLDOWN)

                    if self.test_mode_switch.get():
                        self.after(0, lambda: self.lbl_weather_status.configure(
                            text="тест-режим", text_color="#FF9800"))
                        continue

                    host = self.driver_combo.get()
                    if not host or host in ("Пошук...", "Не знайдено", ""):
                        self.after(0, lambda: self.lbl_weather_status.configure(
                            text="немає пристрою", text_color="#aaa"))
                        continue

                    url = f"http://{host}/getweath"
                    try:
                        response = requests.get(url, timeout=REQ_TIMEOUT)
                        if response.status_code == 200:
```

```
                        data = response.json()

                        sensor_data["camera_color"]     =     data.get("camera_color",
sensor_data["camera_color"])
                        sensor_data["temperature"]      =      data.get("temperature",
sensor_data["temperature"])
                        hum_raw   =   data.get("humidity",   sensor_data["humidity"]   /
100.0)
                        sensor_data["humidity"] = int(round(hum_raw * 100))
                        sensor_data["motion"]           =           data.get("motion",
sensor_data["motion"])

                        self._weather_last_fetch = time.time()

                        self.after(0, lambda: self.lbl_weather_status.configure(
                            text="оновлено", text_color="#4CAF50"))
                    else:
                        self.after(0,          lambda          s=response.status_code:
self.lbl_weather_status.configure(
                            text=f"HTTP {s}", text_color="#f44336"))

                except Exception as e:
                    err_short = str(e)[:30]
                    self.after(0,                  lambda                 m=err_short:
self.lbl_weather_status.configure(
                        text=f"помилка", text_color="#f44336"))

        def start_test_data_sender(self):
            threading.Thread(target=self._test_data_sender_loop, daemon=True).start()

        def _test_data_sender_loop(self):
            while True:
                time.sleep(2)

                if not self.test_mode_switch.get():
                    continue

                host = self.driver_combo.get()
                if not host or host in ("Пошук...", "Не знайдено"):
                    continue

                test_data = {
                    "camera_color": [
                        int(self.test_r.get()),
                        int(self.test_g.get()),
                        int(self.test_b.get())
                    ],
                    "temperature": int(self.test_temp.get()),
                    "humidity": int(self.test_hum.get()),
                    "motion": sensor_data["motion"]
                }

                payload = {"testdata": test_data}

                try:
                    url = f"http://{host}/testdata"
                    requests.post(url, json=payload, timeout=20)
                    print(f"Надіслано тестові дані на {host}: {json.dumps(payload)}")
                except Exception as e:
                    print(f"Помилка надсилання тестових даних на {host}: {e}")
```

```
        def create_test_slider(self, parent, label_text, default_val, min_val,
max_val):
            frame = ctk.CTkFrame(parent, fg_color="transparent")
            frame.pack(fill="x", padx=10, pady=5)

            ctk.CTkLabel(frame, text=label_text, width=65,
anchor="w").pack(side="left")
            val_lbl = ctk.CTkLabel(frame, text=str(default_val), width=35)

            def on_change(val):
                val_lbl.configure(text=str(int(val)))

            slider = ctk.CTkSlider(frame, from_=min_val, to=max_val,
command=on_change, height=14)
            slider.set(default_val)
            slider.pack(side="left", fill="x", expand=True, padx=5)
            val_lbl.pack(side="right")
            return slider, val_lbl

        def populate_profiles_list(self):
            for widget in self.profile_list_scroll.winfo_children():
                widget.destroy()

            for pid, pdata in self.config_data["profiles"].items():
                is_active = (pid == self.current_profile_id)
                bg_color = "#3a4d3a" if is_active else "#333333"
                border_col = "#4CAF50" if is_active else "#444444"

                p_frame = ctk.CTkFrame(self.profile_list_scroll, fg_color=bg_color,
corner_radius=8,
                                       border_width=2, border_color=border_col)
                p_frame.pack(fill="x", pady=5, padx=5)

                name_btn = ctk.CTkButton(p_frame, text=pdata["name"],
fg_color="transparent",
                                         hover_color="#445544", font=("Arial", 14,
"bold"), anchor="w",
                                         command=lambda id=pid:
self.switch_profile(id))
                name_btn.pack(side="left", fill="x", expand=True, padx=5, pady=5)

                edit_btn = ctk.CTkButton(p_frame, text="🖍", width=30,
fg_color="transparent",
                                         hover_color="#555",
                                         command=lambda id=pid, name=pdata["name"]:
self.rename_profile(id, name))
                edit_btn.pack(side="left", padx=2)

                if pid != self.DEFAULT_PROFILE_ID:
                    del_btn = ctk.CTkButton(p_frame, text="✕", width=30,
fg_color="transparent",
                                            hover_color="#d32f2f",
                                            command=lambda id=pid:
self.delete_profile(id))
                    del_btn.pack(side="left", padx=(2, 5))
                else:
                    ctk.CTkFrame(p_frame, width=34, height=30,
fg_color="transparent").pack(side="left", padx=(2, 5))

        def switch_profile(self, profile_id):
            if profile_id not in self.config_data["profiles"]:
```

```
                return
            self.save_config()
            self.current_profile_id = profile_id
            self.load_active_profile_ui()
            self.populate_profiles_list()
            self.save_config()

        def create_new_profile(self):
            new_id = f"prof_{uuid.uuid4().hex[:6]}"
            self.config_data["profiles"][new_id] = {
                "name": "Новий профіль",
                "motion_active": False,
                "r_offset": 0, "g_offset": 0, "b_offset": 0, "cam_influence": 1.0,
                "temp_ranges": {}, "hum_ranges": {}
            }
            self.switch_profile(new_id)

        def rename_profile(self, profile_id, old_name):
            dialog = ctk.CTkInputDialog(text="Введіть нову назву для профілю:",
title="Перейменування")
            new_name = dialog.get_input()
            if new_name and new_name.strip():
                self.config_data["profiles"][profile_id]["name"] = new_name.strip()
                self.save_config()
                self.populate_profiles_list()

        def delete_profile(self, profile_id):
            if profile_id == self.DEFAULT_PROFILE_ID:
                return
            if profile_id in self.config_data["profiles"]:
                del self.config_data["profiles"][profile_id]
            if self.current_profile_id == profile_id:
                self.current_profile_id = self.DEFAULT_PROFILE_ID
                self.load_active_profile_ui()
            self.save_config()
            self.populate_profiles_list()

        def load_active_profile_ui(self):
            self._is_loading = True

            for uid, widgets in self.temp_range_widgets.items():
                widgets['frame'].destroy()
            self.temp_range_widgets.clear()

            for uid, widgets in self.hum_range_widgets.items():
                widgets['frame'].destroy()
            self.hum_range_widgets.clear()

            prof = self.config_data["profiles"][self.current_profile_id]

            if prof.get("motion_active", False):
                self.motion_switch.select()
            else:
                self.motion_switch.deselect()

            self.r_slider.set(prof.get("r_offset", 0))
            self.r_lbl.configure(text=str(prof.get("r_offset", 0)))

            self.g_slider.set(prof.get("g_offset", 0))
            self.g_lbl.configure(text=str(prof.get("g_offset", 0)))
```

```
            self.b_slider.set(prof.get("b_offset", 0))
            self.b_lbl.configure(text=str(prof.get("b_offset", 0)))

            inf_val = prof.get("cam_influence", 1.0)
            self.inf_slider.set(inf_val)
            self.inf_lbl.configure(text=f"{int(inf_val * 100)}%")

            for uid, data in prof.get("temp_ranges", {}).items():
                self.add_temp_range(uid, data)
            for uid, data in prof.get("hum_ranges", {}).items():
                self.add_hum_range(uid, data)

            self._is_loading = False

        def sync_widths(self, event):
            canvas_width = event.width
            target_width = int(canvas_width * 0.95)
            x_offset = (canvas_width - target_width) // 2
            self.canvas.coords(self.scroll_window, x_offset, 0)
            self.canvas.itemconfig(self.scroll_window, width=target_width)

        def kinetic_scroll(self, event):
            widget_under_cursor = self.winfo_containing(event.x_root, event.y_root)
            if widget_under_cursor:
                parent_name = widget_under_cursor.winfo_parent()
                if str(self.profile_list_scroll) in parent_name or widget_under_cursor
== self.profile_list_scroll:
                    return
            self.canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")

        def add_temp_range(self, uid=None, data=None):
            if uid is None:
                uid = f"temp_{uuid.uuid4().hex[:5]}"
            if data is None:
                data = {"t_min": 10, "t_max": 30, "r": 0, "g": 0, "b": 0, "influence":
0.5}

            range_frame   =   ctk.CTkFrame(self.temp_list_frame,   fg_color="#444444",
corner_radius=8,
                                           border_width=1, border_color="#555")
            range_frame.pack(fill="x", pady=5, padx=5)

            header = ctk.CTkFrame(range_frame, fg_color="transparent")
            header.pack(fill="x", padx=10, pady=5)
            ctk.CTkLabel(header,     text=f"ID:     {uid}",     font=("Arial",     12,
"bold")).pack(side="left")
            ctk.CTkButton(header,   text="Видалити",   width=60,   fg_color="#d32f2f",
hover_color="#b71c1c",
                          height=24,
                          command=lambda:     self.delete_range(uid,     range_frame,
self.temp_range_widgets)).pack(side="right")

            temp_row = ctk.CTkFrame(range_frame, fg_color="transparent")
            temp_row.pack(fill="x", padx=10, pady=5)
            ctk.CTkLabel(temp_row, text="Точки (Min - Max):").pack(side="left")

            val_lbl     =     ctk.CTkLabel(temp_row,     text=f"{data['t_min']}     -
{data['t_max']}", width=60,
                                    font=("Arial", 12, "bold"))
            val_lbl.pack(side="right")
```

```
            def update_lbl(*_):
                val_lbl.configure(text=f"{int(slider_min.get())}                    -
{int(slider_max.get())}")
                self.save_config()

            slider_min   =   ctk.CTkSlider(temp_row,   from_=-20,   to=60,   width=80,
command=update_lbl)
            slider_max   =   ctk.CTkSlider(temp_row,   from_=-20,   to=60,   width=80,
command=update_lbl)
            slider_min.set(data['t_min'])
            slider_max.set(data['t_max'])
            slider_min.pack(side="left", padx=(10, 5))
            slider_max.pack(side="left", padx=5)

            r_sl, _ = self.create_offset_slider(range_frame, "R", data['r'])
            g_sl, _ = self.create_offset_slider(range_frame, "G", data['g'])
            b_sl, _ = self.create_offset_slider(range_frame, "B", data['b'])
            inf_sl, _ = self.create_influence_slider(range_frame, data['influence'])

            self.temp_range_widgets[uid] = {
                "frame": range_frame,
                "t_min": slider_min, "t_max": slider_max,
                "r": r_sl, "g": g_sl, "b": b_sl, "inf": inf_sl
            }
            self.save_config()

        def add_hum_range(self, uid=None, data=None):
            if uid is None:
                uid = f"hum_{uuid.uuid4().hex[:5]}"
            if data is None:
                data = {"h_min": 0.2, "h_max": 0.6, "r": 0, "g": 0, "b": 0, "influence":
0.5}

            range_frame   =   ctk.CTkFrame(self.hum_list_frame,   fg_color="#364958",
corner_radius=8,
                                           border_width=1, border_color="#4f6b82")
            range_frame.pack(fill="x", pady=5, padx=5)

            header = ctk.CTkFrame(range_frame, fg_color="transparent")
            header.pack(fill="x", padx=10, pady=5)
            ctk.CTkLabel(header, text=f"ID: {uid}", font=("Arial", 12, "bold"),
                         text_color="#89cff0").pack(side="left")
            ctk.CTkButton(header,   text="Видалити",   width=60,   fg_color="#d32f2f",
hover_color="#b71c1c",
                          height=24,
                          command=lambda:     self.delete_range(uid,     range_frame,
self.hum_range_widgets)).pack(side="right")

            hum_row = ctk.CTkFrame(range_frame, fg_color="transparent")
            hum_row.pack(fill="x", padx=10, pady=5)
            ctk.CTkLabel(hum_row, text="Відсотки (Min - Max):").pack(side="left")

            val_min = int(data['h_min'] * 100)
            val_max = int(data['h_max'] * 100)

            val_lbl = ctk.CTkLabel(hum_row, text=f"{val_min}% - {val_max}%", width=70,
                                   font=("Arial", 12, "bold"))
            val_lbl.pack(side="right")

            def update_lbl(*_):
```

```
                val_lbl.configure(text=f"{int(slider_min.get())}%                    -
{int(slider_max.get())}%")
                self.save_config()

            slider_min   =   ctk.CTkSlider(hum_row,   from_=0,   to=100,   width=80,
command=update_lbl)
            slider_max   =   ctk.CTkSlider(hum_row,   from_=0,   to=100,   width=80,
command=update_lbl)
            slider_min.set(val_min)
            slider_max.set(val_max)
            slider_min.pack(side="left", padx=(10, 5))
            slider_max.pack(side="left", padx=5)

            r_sl, _ = self.create_offset_slider(range_frame, "R", data['r'])
            g_sl, _ = self.create_offset_slider(range_frame, "G", data['g'])
            b_sl, _ = self.create_offset_slider(range_frame, "B", data['b'])
            inf_sl, _ = self.create_influence_slider(range_frame, data['influence'])

            self.hum_range_widgets[uid] = {
                "frame": range_frame,
                "h_min": slider_min, "h_max": slider_max,
                "r": r_sl, "g": g_sl, "b": b_sl, "inf": inf_sl
            }
            self.save_config()

        def delete_range(self, uid, frame, target_dict):
            if uid in target_dict:
                del target_dict[uid]
            frame.destroy()
            self.save_config()

        def create_offset_slider(self, parent, label_text, default_val):
            frame = ctk.CTkFrame(parent, fg_color="transparent")
            frame.pack(fill="x", padx=15, pady=2)

            ctk.CTkLabel(frame,   text=label_text,   width=20,   font=("Arial",   12,
"bold")).pack(side="left")
            val_lbl = ctk.CTkLabel(frame, text=str(default_val), width=40)

            def on_change(val):
                val_lbl.configure(text=str(int(val)))
                self.save_config()

            slider   =   ctk.CTkSlider(frame,   from_=0,   to=4096,   command=on_change,
height=14)
            slider.set(default_val)
            slider.pack(side="left", fill="x", expand=True, padx=10)
            val_lbl.pack(side="right")
            return slider, val_lbl

        def create_influence_slider(self, parent, default_val):
            frame = ctk.CTkFrame(parent, fg_color="transparent")
            frame.pack(fill="x", padx=15, pady=2)

            val_lbl = ctk.CTkLabel(frame, text=f"{int(default_val * 100)}%", width=40)

            def on_change(val):
                val_lbl.configure(text=f"{int(val * 100)}%")
                self.save_config()
```

```
            slider = ctk.CTkSlider(frame, from_=0.0, to=1.0, command=on_change,
height=14)
            slider.set(default_val)
            slider.pack(side="left", fill="x", expand=True, padx=5)
            val_lbl.pack(side="right")
            return slider, val_lbl

        def start_serial_scanner(self):
            scan_thread = threading.Thread(target=self._scan_ports_task, daemon=True)
            scan_thread.start()

        def _scan_ports_task(self):
            while True:
                if self.is_setup_window_open:
                    time.sleep(1)
                    continue

                ports = serial.tools.list_ports.comports()
                found_in_this_cycle = False

                for port_info in ports:
                    if self.is_setup_window_open:
                        break

                    try:
                        ser = serial.Serial(port_info.device, 115200, timeout=1)
                        time.sleep(1.5)

                        end_time = time.time() + 3.0
                        found = False

                        while time.time() < end_time:
                            if ser.in_waiting:
                                line = ser.readline().decode('utf-8',
errors='ignore').strip()
                                if "rewifi" in line:
                                    ser.write(b"rewifying\n")
                                    found = True
                                    break

                        if found:
                            self.is_setup_window_open = True
                            found_in_this_cycle = True
                            self.after(0, self.show_wifi_setup_window, ser,
port_info.device)
                            break
                        else:
                            ser.close()

                    except serial.SerialException:
                        pass

                if not found_in_this_cycle:
                    time.sleep(3)

        def show_wifi_setup_window(self, ser_conn, port_name):
            wifi_win = ctk.CTkToplevel(self)
            wifi_win.title("Налаштування мережі: Розумне освітлення")
            wifi_win.geometry("400x300")
            wifi_win.attributes("-topmost", True)
            wifi_win.resizable(False, False)
```

```
            wifi_win.protocol("WM_DELETE_WINDOW",                                lambda:
self._on_wifi_window_close(wifi_win, ser_conn))

            ctk.CTkLabel(
                wifi_win,
                text=f"Знайдено  пристрій  на  порту  {port_name}\nВведіть  дані  для
підключення",
                font=("Arial", 14, "bold")
            ).pack(pady=(20, 15))

            ctk.CTkLabel(wifi_win, text="Wi-Fi SSID (Назва мережі):", font=("Arial",
12)).pack(anchor="w", padx=40)
            ssid_entry          =          ctk.CTkEntry(wifi_win,          width=320,
placeholder_text="Моя_Мережа")
            ssid_entry.pack(pady=(0, 15), padx=40)

            ctk.CTkLabel(wifi_win,           text="Пароль:",           font=("Arial",
12)).pack(anchor="w", padx=40)
            pass_entry = ctk.CTkEntry(wifi_win, width=320, placeholder_text="Пароль",
show="*")
            pass_entry.pack(pady=(0, 20), padx=40)

            send_btn = ctk.CTkButton(
                wifi_win,
                text="Надіслати в пристрій",
                fg_color="#4CAF50",
                hover_color="#45a049",
                command=lambda:    self._send_wifi_credentials(wifi_win,    ser_conn,
ssid_entry.get(), pass_entry.get())
            )
            send_btn.pack(pady=10)

        def _send_wifi_credentials(self, window, ser_conn, ssid, password):
            if not ssid:
                return

            payload = json.dumps({"ssid": ssid, "password": password}) + "\n"

            try:
                ser_conn.write(payload.encode('utf-8'))
                print(f"Відправлено на ESP32: {payload.strip()}")
            except Exception as e:
                print(f"Помилка відправки через Serial: {e}")
            finally:
                if ser_conn and ser_conn.is_open:
                    ser_conn.close()
                self.is_setup_window_open = False
                window.destroy()

        def _on_wifi_window_close(self, window, ser_conn):
            if ser_conn and ser_conn.is_open:
                ser_conn.close()
            self.is_setup_window_open = False
            window.destroy()

        def update_visuals(self):
            if self.test_mode_switch.get():
                cam_r = int(self.test_r.get())
                cam_g = int(self.test_g.get())
                cam_b = int(self.test_b.get())
```

```
                cur_temp = int(self.test_temp.get())
                cur_hum = int(self.test_hum.get())
                cur_motion = sensor_data["motion"]
            else:
                cam_r, cam_g, cam_b = sensor_data["camera_color"]
                cur_temp = sensor_data["temperature"]
                cur_hum = sensor_data["humidity"]
                cur_motion = sensor_data["motion"]

            self.lbl_cam_val.configure(text=f"RGB({cam_r}, {cam_g}, {cam_b})")
            self.cam_color_dot.configure(fg_color=rgb_to_hex([cam_r, cam_g, cam_b]))
            self.lbl_temp_val.configure(text=f"{cur_temp}°C")
            self.lbl_hum_val.configure(text=f"{cur_hum}%")

            if cur_motion:
                self.lbl_motion_val.configure(text="Є", text_color="#4CAF50")
            else:
                self.lbl_motion_val.configure(text="Немає", text_color="#aaa")

            r_off = self.r_slider.get()
            g_off = self.g_slider.get()
            b_off = self.b_slider.get()
            cam_inf = self.inf_slider.get()

            final_r_12 = (cam_r * 16) + (r_off * cam_inf)
            final_g_12 = (cam_g * 16) + (g_off * cam_inf)
            final_b_12 = (cam_b * 16) + (b_off * cam_inf)

            for uid, widgets in self.temp_range_widgets.items():
                t_min = int(widgets['t_min'].get())
                t_max = int(widgets['t_max'].get())
                if t_min <= cur_temp <= t_max:
                    final_r_12 += widgets['r'].get() * widgets['inf'].get()
                    final_g_12 += widgets['g'].get() * widgets['inf'].get()
                    final_b_12 += widgets['b'].get() * widgets['inf'].get()

            for uid, widgets in self.hum_range_widgets.items():
                h_min = int(widgets['h_min'].get())
                h_max = int(widgets['h_max'].get())
                if h_min <= cur_hum <= h_max:
                    final_r_12 += widgets['r'].get() * widgets['inf'].get()
                    final_g_12 += widgets['g'].get() * widgets['inf'].get()
                    final_b_12 += widgets['b'].get() * widgets['inf'].get()

            final_r_12 = min(4095, max(0, final_r_12))
            final_g_12 = min(4095, max(0, final_g_12))
            final_b_12 = min(4095, max(0, final_b_12))

            final_r_8bit = int(final_r_12 / 16)
            final_g_8bit = int(final_g_12 / 16)
            final_b_8bit = int(final_b_12 / 16)

            self.current_calculated_color = [final_r_8bit, final_g_8bit, final_b_8bit]
            hex_color = rgb_to_hex(self.current_calculated_color)

            self.dot_final.configure(fg_color=hex_color)
            self.lbl_final_val.configure(text=f"RGB({final_r_8bit},   {final_g_8bit},
{final_b_8bit})")
            self.final_color_dot_top.configure(fg_color=hex_color)

            self.after(100, self.update_visuals)
```

```
if __name__ == "__main__":
    app = SmartLightApp()
    app.mainloop()
```

Лістинг Г.1 — код програми управління профілями

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WebServer.h>
#include <Preferences.h>
#include <ArduinoJson.h>
#include <esp_now.h>
#include <vector>
#include <algorithm>

WebServer server(80);
Preferences preferences;

String myHostname = "";
String currentSSID = "";
String currentPassword = "";

String activeProfileName = "";
bool motionActive = false;
int rOffset = 0;
int gOffset = 0;
int bOffset = 0;
float camInfluence = 1.0;
String currentProfileJson = "{}";

struct TempRange {
  int t_min;
  int t_max;
  int r;
  int g;
  int b;
  float influence;
};

struct HumRange {
  float h_min;
  float h_max;
  int r;
  int g;
  int b;
  float influence;
};

std::vector<TempRange> activeTempRanges;
std::vector<HumRange> activeHumRanges;

int finalR = 0;
int finalG = 0;
int finalB = 0;
int finalW = 0;

#define COLOR_TRANSITION_MS  5000
#define COLOR_DEADZONE       30

float transStartR = 0, transStartG = 0, transStartB = 0, transStartW = 0;
```

```
      float transTargetR = 0, transTargetG = 0, transTargetB = 0, transTargetW = 0;
      float smoothR = 0, smoothG = 0, smoothB = 0, smoothW = 0;
      unsigned long transStartTime = 0;
      bool transitionActive = false;

      // Функція інтерполяції Ease in-out
      float easeInOut(float t) {
        return t * t * (3.0f - 2.0f * t);
      }

      // Встановлення нової цілі для зміни кольору
      void setColorTarget(float r, float g, float b, float w) {
        float diffR = abs(r - transTargetR);
        float diffG = abs(g - transTargetG);
        float diffB = abs(b - transTargetB);

        if (diffR <= COLOR_DEADZONE && diffG <= COLOR_DEADZONE && diffB <=
COLOR_DEADZONE) {
          return;
        }

        transStartR = smoothR;
        transStartG = smoothG;
        transStartB = smoothB;
        transStartW = smoothW;

        transTargetR = r;
        transTargetG = g;
        transTargetB = b;
        transTargetW = w;

        transStartTime = millis();
        transitionActive = true;
      }

      // Оновлення поточних значень кольору
      void updateColorTransition() {
        if (!transitionActive) return;
        unsigned long elapsed = millis() - transStartTime;
        if (elapsed >= COLOR_TRANSITION_MS) {
          smoothR = transTargetR;
          smoothG = transTargetG;
          smoothB = transTargetB;
          smoothW = transTargetW;
          transitionActive = false;
          return;
        }

        float t = easeInOut((float)elapsed / (float)COLOR_TRANSITION_MS);
        smoothR = transStartR + (transTargetR - transStartR) * t;
        smoothG = transStartG + (transTargetG - transStartG) * t;
        smoothB = transStartB + (transTargetB - transStartB) * t;
        smoothW = transStartW + (transTargetW - transStartW) * t;
      }

      int activeCamR = 0, activeCamG = 0, activeCamB = 0;
      int activeTemp = 0;
      int activeHum = 0;
      bool activeMotion = false;

      int realCamR = 0, realCamG = 0, realCamB = 0;
```

```
int realTemp = 24;
int realHum = 45;
bool realMotion = false;

int testCamR = 0, testCamG = 0, testCamB = 0;
int testTemp = 0;
int testHum = 0;
bool testMotion = false;

bool testmode = false;
unsigned long testModeTimer = 0;
unsigned long lastDebugPrint = 0;

unsigned long lightTimer = 0;
const unsigned long lightTimeout = 5 * 60 * 1000;
bool lightShouldBeOn = false;

typedef struct struct_message {
  uint8_t msg_type;
  char payload[12];
  int r;
  int g;
  int b;
  int temp;
  float hum;
  bool motion;
} struct_message;

// Обробка отриманих даних через ESP-NOW
void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t *incomingData, int
len) {
  if (len == sizeof(struct_message)) {
    struct_message recMsg;
    memcpy(&recMsg, incomingData, sizeof(recMsg));

    if (recMsg.msg_type == 0 && strcmp(recMsg.payload, "weathdata") == 0) {
      struct_message replyMsg;
      replyMsg.msg_type = 0;
      strcpy(replyMsg.payload, "pong");

      if (!esp_now_is_peer_exist(info->src_addr)) {
        esp_now_peer_info_t peerInfo = {};
        memcpy(peerInfo.peer_addr, info->src_addr, 6);
        peerInfo.channel = 0;
        peerInfo.encrypt = false;
        esp_now_add_peer(&peerInfo);
      }
      esp_now_send(info->src_addr, (uint8_t *) &replyMsg, sizeof(replyMsg));
    }
    else if (recMsg.msg_type == 1) {
      realCamR = recMsg.r;
      realCamG = recMsg.g;
      realCamB = recMsg.b;
      realTemp = recMsg.temp;
      realHum  = (int)(recMsg.hum * 100);
      realMotion = recMsg.motion;
    }
  }
}

void OnDataSent(const wifi_tx_info_t *info, esp_now_send_status_t status) {}
```

```
// Завантаження профілю освітлення з пам'яті
void loadProfile() {
  Preferences profPrefs;
  profPrefs.begin("light_prof", true);
  currentProfileJson = profPrefs.getString("json", "{}");
  profPrefs.end();

  activeTempRanges.clear();
  activeHumRanges.clear();

  Serial.println("Loading profile...");
  if (currentProfileJson == "{}") {
    Serial.println("Profile empty. Default used.");
  } else {
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, currentProfileJson);
    if (!error && doc.containsKey("profile")) {
      JsonObject profMap = doc["profile"].as<JsonObject>();
      for (JsonPair kv : profMap) {
        JsonObject pData = kv.value().as<JsonObject>();
        activeProfileName = pData["name"].as<String>();
        motionActive = pData["motion_active"].as<bool>();
        rOffset = pData["r_offset"].as<int>();
        gOffset = pData["g_offset"].as<int>();
        bOffset = pData["b_offset"].as<int>();
        camInfluence = pData["cam_influence"].as<float>();

        if (pData.containsKey("temp_ranges")) {
          JsonObject tRanges = pData["temp_ranges"].as<JsonObject>();
          for (JsonPair tkv : tRanges) {
            JsonObject rData = tkv.value().as<JsonObject>();
            TempRange tr;
            tr.t_min = rData["t_min"].as<int>();
            tr.t_max = rData["t_max"].as<int>();
            tr.r = rData["r"].as<int>();
            tr.g = rData["g"].as<int>();
            tr.b = rData["b"].as<int>();
            tr.influence = rData["influence"].as<float>();
            activeTempRanges.push_back(tr);
          }
        }

        if (pData.containsKey("hum_ranges")) {
          JsonObject hRanges = pData["hum_ranges"].as<JsonObject>();
          for (JsonPair hkv : hRanges) {
            JsonObject rData = hkv.value().as<JsonObject>();
            HumRange hr;
            hr.h_min = rData["h_min"].as<float>();
            hr.h_max = rData["h_max"].as<float>();
            hr.r = rData["r"].as<int>();
            hr.g = rData["g"].as<int>();
            hr.b = rData["b"].as<int>();
            hr.influence = rData["influence"].as<float>();
            activeHumRanges.push_back(hr);
          }
        }

        Serial.println("Profile imported: " + activeProfileName);
        break;
      }
```

```
      } else {
        Serial.println("Error parsing profile!");
      }
    }
  }

  // Обробка HTTP запиту на збереження профілю
  void handleProfileData() {
    if (server.hasArg("plain")) {
      String jsonPayload = server.arg("plain");
      JsonDocument doc;
      DeserializationError error = deserializeJson(doc, jsonPayload);
      if (!error && doc.containsKey("profile")) {
        Preferences profPrefs;
        profPrefs.begin("light_prof", false);
        profPrefs.clear();
        profPrefs.putString("json", jsonPayload);
        profPrefs.end();

        loadProfile();
        server.send(200, "application/json", "{\"status\":\"profile_saved\"}");
      } else {
        server.send(400,                                          "application/json",
"{\"status\":\"error\",\"message\":\"Invalid profile structure\"}");
      }
    }
  }

  // Обробка HTTP запиту з тестовими даними
  void handleTestData() {
    if (server.hasArg("plain")) {
      String jsonPayload = server.arg("plain");
      JsonDocument doc;
      DeserializationError error = deserializeJson(doc, jsonPayload);

      if (!error && doc.containsKey("testdata")) {
        JsonObject tData = doc["testdata"];
        testCamR = tData["camera_color"][0];
        testCamG = tData["camera_color"][1];
        testCamB = tData["camera_color"][2];
        testTemp = tData["temperature"];
        testHum = tData["humidity"];
        testMotion = tData["motion"];
        testmode = true;
        testModeTimer = millis();

        server.send(200, "application/json", "{\"status\":\"testdata_received\"}");
      }
    }
  }

  // Обробка HTTP запиту на отримання поточних сенсорів
  void handleGetWeather() {
    JsonDocument doc;
    doc["temperature"] = realTemp;
    doc["humidity"]    = realHum / 100.0;
    doc["motion"]      = realMotion;
    doc["camera_color"][0] = realCamR;
    doc["camera_color"][1] = realCamG;
    doc["camera_color"][2] = realCamB;
```

```
        String response;
        serializeJson(doc, response);
        server.send(200, "application/json", response);
      }

      void      handlePostData()      {      server.send(200,      "application/json",
"{\"status\":\"ok\"}"); }
      void      handleGetStatus()      {      server.send(200,      "application/json",
"{\"status\":\"online\"}"); }

      void setup() {
        Serial.begin(115200);
        preferences.begin("wifi_config", false);
        currentSSID = preferences.getString("ssid", "YOUR_SSID");
        currentPassword = preferences.getString("password", "YOUR_PASSWORD");

        delay(1000);
        Serial.println("rewifi");

        unsigned long startTime = millis();
        bool configMode = false;
        while (millis() - startTime < 3000) {
          if (Serial.available()) {
            String response = Serial.readStringUntil('\n');
            response.trim();
            if (response == "rewifying") { configMode = true; break; }
          }
        }

        if (configMode) {
          unsigned long waitDataTime = millis();
          while (millis() - waitDataTime < 45000) {
            if (Serial.available()) {
              String jsonData = Serial.readStringUntil('\n');
              jsonData.trim();
              JsonDocument doc;
              DeserializationError error = deserializeJson(doc, jsonData);
              if (!error && doc.containsKey("ssid") && doc.containsKey("password")) {
                currentSSID = doc["ssid"].as<String>();
                currentPassword = doc["password"].as<String>();
                preferences.putString("ssid", currentSSID);
                preferences.putString("password", currentPassword);
                break;
              }
            }
          }
        }

        WiFi.mode(WIFI_STA);
        WiFi.begin(currentSSID.c_str(), currentPassword.c_str());
        while (WiFi.status() != WL_CONNECTED) { delay(500); }

        if (esp_now_init() != ESP_OK) {
          Serial.println("ESP-NOW Error");
        } else {
          esp_now_register_recv_cb(OnDataRecv);
          esp_now_register_send_cb(OnDataSent);
        }

        if (!MDNS.begin("esp32_probe")) { Serial.println("mDNS Error"); }
        int index = 1;
```

```
  while (true) {
    String checkHost = "dsmartlight" + String(index);
    IPAddress remoteIP = MDNS.queryHost(checkHost.c_str(), 1500);
    if (remoteIP == IPAddress(0, 0, 0, 0)) { myHostname = checkHost; break; }
    else { index++; }
  }
  MDNS.end();
  MDNS.begin(myHostname.c_str());

  server.on("/", HTTP_GET, handleGetStatus);
  server.on("/", HTTP_POST, handlePostData);
  server.on("/profile", HTTP_POST, handleProfileData);
  server.on("/testdata", HTTP_POST, handleTestData);
  server.on("/getweath", HTTP_GET, handleGetWeather);

  server.begin();
  MDNS.addService("http", "tcp", 80);

  loadProfile();
}

void loop() {
  server.handleClient();

  // Перевірка таймауту тестового режиму
  if (testmode && (millis() - testModeTimer >= 20000)) {
    testmode = false;
    Serial.println("Timeout: Test mode off.");
  }

  // Вибір джерела даних (тест або реал)
  if (testmode) {
    activeCamR   = testCamR;   activeCamG   = testCamG;   activeCamB   = testCamB;
    activeTemp   = testTemp;   activeHum    = testHum;    activeMotion = testMotion;
  } else {
    activeCamR   = realCamR;   activeCamG   = realCamG;   activeCamB   = realCamB;
    activeTemp   = realTemp;   activeHum    = realHum;    activeMotion = realMotion;
  }

  // Логіка таймера датчика руху
  if (motionActive) {
    if (activeMotion) {
      lightShouldBeOn = true;
      lightTimer = millis();
    } else {
      if (lightShouldBeOn && (millis() - lightTimer >= lightTimeout)) {
        lightShouldBeOn = false;
        Serial.println("Timeout: Motion inactive. Light off.");
      }
    }
  } else {
    lightShouldBeOn = true;
  }

  // Розрахунок кольору RGBW
  if (lightShouldBeOn) {
    float final_r_12 = (activeCamR * 16) + (rOffset * camInfluence);
    float final_g_12 = (activeCamG * 16) + (gOffset * camInfluence);
    float final_b_12 = (activeCamB * 16) + (bOffset * camInfluence);

    // Вплив температурних діапазонів
```

```
          for (const auto& tr : activeTempRanges) {
            if (activeTemp >= tr.t_min && activeTemp <= tr.t_max) {
              final_r_12 += tr.r * tr.influence;
              final_g_12 += tr.g * tr.influence;
              final_b_12 += tr.b * tr.influence;
            }
          }

          // Вплив діапазонів вологості
          float cur_hum_float = activeHum / 100.0;
          for (const auto& hr : activeHumRanges) {
            if (cur_hum_float >= hr.h_min && cur_hum_float <= hr.h_max) {
              final_r_12 += hr.r * hr.influence;
              final_g_12 += hr.g * hr.influence;
              final_b_12 += hr.b * hr.influence;
            }
          }

          int r_12 = constrain((int)final_r_12, 0, 4095);
          int g_12 = constrain((int)final_g_12, 0, 4095);
          int b_12 = constrain((int)final_b_12, 0, 4095);

          finalR = r_12 / 16;
          finalG = g_12 / 16;
          finalB = b_12 / 16;

          // Виділення білого каналу
          finalW = std::min({finalR, finalG, finalB});

          setColorTarget((float)finalR, (float)finalG, (float)finalB, (float)finalW);

          if (millis() - lastDebugPrint >= 2000) {
            Serial.printf("Target: R:%d G:%d B:%d W:%d | Smooth: R:%.1f G:%.1f B:%.1f
W:%.1f\n",
                          finalR, finalG, finalB, finalW, smoothR, smoothG, smoothB,
smoothW);
            lastDebugPrint = millis();
          }
        } else {
          setColorTarget(0, 0, 0, 0);
        }

        updateColorTransition();

        // ШІМ керування стрічкою
        analogWrite(25, (int)smoothR);
        analogWrite(26, (int)smoothG);
        analogWrite(27, (int)smoothB);
        analogWrite(32, (int)smoothW);

        delay(2);
      }
```

Лістинг Г.2 — Програма мікроконтролера ESP32-WROOM-DA (виконавчий вузол)

```
#include "esp_camera.h"
#include <DHT.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WebServer.h>
#include <ArduinoJson.h>
#include <Preferences.h>
#include <esp_now.h>

WebServer server(80);
Preferences preferences;
String currentSSID = "";
String currentPassword = "";

#define DHTPIN 1
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define PIR_PIN 3

unsigned long lastSendTime = 0;
const unsigned long sendInterval = 500;

// Розпіновка камери для ESP32-S3 N16R8
#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     15
#define SIOD_GPIO_NUM      4
#define SIOC_GPIO_NUM      5
#define Y9_GPIO_NUM       16
#define Y8_GPIO_NUM       17
#define Y7_GPIO_NUM       18
#define Y6_GPIO_NUM       12
#define Y5_GPIO_NUM       10
#define Y4_GPIO_NUM        8
#define Y3_GPIO_NUM        9
#define Y2_GPIO_NUM       11
#define VSYNC_GPIO_NUM     6
#define HREF_GPIO_NUM      7
#define PCLK_GPIO_NUM     13

typedef struct struct_message {
  uint8_t msg_type;
  char payload[12];
  int r;
  int g;
  int b;
  int temp;
  float hum;
  bool motion;
} struct_message;

uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
uint8_t discoveredMacs[10][6];
int discoveredCount = 0;

// Колбек прийому відповідей Pong через ESP-NOW
void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t *incomingData, int
len) {
  if (len == sizeof(struct_message)) {
    struct_message recMsg;
```

```
    memcpy(&recMsg, incomingData, sizeof(recMsg));

    if (recMsg.msg_type == 0 && strcmp(recMsg.payload, "pong") == 0) {
      bool alreadyExists = false;
      for (int i = 0; i < discoveredCount; i++) {
        if (memcmp(discoveredMacs[i], info->src_addr, 6) == 0) {
          alreadyExists = true;
          break;
        }
      }

      if (!alreadyExists && discoveredCount < 10) {
        memcpy(discoveredMacs[discoveredCount], info->src_addr, 6);
        discoveredCount++;
        Serial.printf("Driver found! MAC: %02X:%02X:%02X:%02X:%02X:%02X\n",
                      info->src_addr[0], info->src_addr[1], info->src_addr[2],
                      info->src_addr[3], info->src_addr[4], info->src_addr[5]);
      }
    }
  }
}

void OnDataSent(const wifi_tx_info_t *info, esp_now_send_status_t status) {}

void handleRoot() {
  server.send(200, "text/plain", "Hello from databoard_sensor!");
}

void setup() {
  Serial.begin(115200);
  pinMode(PIR_PIN, INPUT);

  preferences.begin("wifi_config", false);
  currentSSID = preferences.getString("ssid", "YOUR_SSID");
  currentPassword = preferences.getString("password", "YOUR_PASSWORD");

  delay(1000);
  Serial.println("rewifi");
  unsigned long startTime = millis();
  bool configMode = false;
  while (millis() - startTime < 3000) {
    if (Serial.available()) {
      String response = Serial.readStringUntil('\n');
      response.trim();
      if (response == "rewifying") {
        configMode = true;
        break;
      }
    }
  }

  if (configMode) {
    Serial.println("Waiting for WiFi JSON...");
    unsigned long waitDataTime = millis();
    bool dataReceived = false;

    while (millis() - waitDataTime < 45000) {
      if (Serial.available()) {
        String jsonData = Serial.readStringUntil('\n');
        jsonData.trim();
```

```
      JsonDocument doc;
      DeserializationError error = deserializeJson(doc, jsonData);

      if (!error && doc.containsKey("ssid") && doc.containsKey("password")) {
        currentSSID = doc["ssid"].as<String>();
        currentPassword = doc["password"].as<String>();
        preferences.putString("ssid", currentSSID);
        preferences.putString("password", currentPassword);
        Serial.println("WiFi config saved.");
        dataReceived = true;
        break;
      } else {
        Serial.println("JSON parse error.");
      }
    }
  }

  if (!dataReceived) {
    Serial.println("Timeout. Using stored config.");
  }
}

WiFi.mode(WIFI_STA);
WiFi.begin(currentSSID.c_str(), currentPassword.c_str());
WiFi.setSleep(false);

Serial.println("Connecting to WiFi: " + currentSSID);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("\nWiFi connected. IP: " + WiFi.localIP().toString());

if (!MDNS.begin("databoard_sensor")) {
  Serial.println("mDNS Error");
} else {
  Serial.println("mDNS started: databoard_sensor.local");
  MDNS.addService("http", "tcp", 80);
}

server.on("/", HTTP_GET, handleRoot);
server.begin();
Serial.println("HTTP server started.");

dht.begin();

// Ініціалізація камери
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
```

```
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_RGB565;
  config.frame_size = FRAMESIZE_QQVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
  config.fb_location = CAMERA_FB_IN_PSRAM;

  if (esp_camera_init(&config) != ESP_OK) {
    Serial.println("Cam Error");
    while (true) { delay(1000); }
  }

  sensor_t * s = esp_camera_sensor_get();
  if (s) {
    s->set_vflip(s, 0);
    s->set_hmirror(s, 0);
  }

  if (esp_now_init() != ESP_OK) {
    Serial.println("ESP-NOW Error");
  } else {
    Serial.println("ESP-NOW active.");
    esp_now_register_recv_cb(OnDataRecv);
    esp_now_register_send_cb(OnDataSent);

    esp_now_peer_info_t peerInfo = {};
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    esp_now_add_peer(&peerInfo);
  }

  Serial.println("System Ready");
}

void loop() {
  server.handleClient();

  if (millis() - lastSendTime >= sendInterval) {
    lastSendTime = millis();
    if (WiFi.status() != WL_CONNECTED) return;

    // Збір даних з датчиків DHT11 та PIR
    int temperature_int = 0;
    float humidity_float = 0.0;

    float h = dht.readHumidity();
    float t = dht.readTemperature();
    if (!isnan(h) && !isnan(t)) {
      temperature_int = (int)round(t);
      humidity_float = h / 100.0f;
    }

    bool isMotionDetected = (digitalRead(PIR_PIN) == HIGH);

    // Обробка зображення з камери (середній колір)
```

```
          int avgR = 0, avgG = 0, avgB = 0;
          camera_fb_t * fb = esp_camera_fb_get();
          if (fb) {
            uint32_t totalR = 0, totalG = 0, totalB = 0;
            uint32_t pixelCount = 0;

            for (size_t i = 0; i < fb->len; i += 2) {
              if (i + 1 >= fb->len) break;
              uint16_t pixel = (fb->buf[i] << 8) | fb->buf[i+1];
              totalR += ((pixel >> 11) & 0x1F) * 255 / 31;
              totalG += ((pixel >> 5) & 0x3F) * 255 / 63;
              totalB += (pixel & 0x1F) * 255 / 31;
              pixelCount++;
            }

            if (pixelCount > 0) {
              avgR = totalR / pixelCount;
              avgG = totalG / pixelCount;
              avgB = totalB / pixelCount;
            }
            esp_camera_fb_return(fb);
          }

          // Широкомовний пошук драйверів (Ping)
          discoveredCount = 0;
          struct_message pingMsg;
          pingMsg.msg_type = 0;
          strcpy(pingMsg.payload, "weathdata");

          esp_now_send(broadcastAddress, (uint8_t *) &pingMsg, sizeof(pingMsg));
          Serial.println("Broadcast Ping sent. Waiting for responses...");

          // Вікно очікування відповідей
          unsigned long waitStart = millis();
          while (millis() - waitStart < 600) {
            server.handleClient();
            yield();
          }

          // Надсилання зібраних даних на всі знайдені драйвери
          if (discoveredCount > 0) {
            Serial.printf("Search closed. Discovered devices: %d. Sending data...\n",
discoveredCount);

            struct_message dataMsg;
            dataMsg.msg_type = 1;
            dataMsg.r = avgR;
            dataMsg.g = avgG;
            dataMsg.b = avgB;
            dataMsg.temp = temperature_int;
            dataMsg.hum = humidity_float;
            dataMsg.motion = isMotionDetected;

            Serial.printf("Sending: RGB=(%d,%d,%d) | T=%d°C | H=%d%% | Motion=%s\n",
                          dataMsg.r,      dataMsg.g,      dataMsg.b,      dataMsg.temp,
(int)(dataMsg.hum * 100),
                          dataMsg.motion ? "YES" : "NO");

            for (int i = 0; i < discoveredCount; i++) {
              if (!esp_now_is_peer_exist(discoveredMacs[i])) {
                esp_now_peer_info_t peerInfo = {};
```

```
                memcpy(peerInfo.peer_addr, discoveredMacs[i], 6);
                peerInfo.channel = 0;
                peerInfo.encrypt = false;
                esp_now_add_peer(&peerInfo);
              }

              esp_err_t result = esp_now_send(discoveredMacs[i], (uint8_t *) &dataMsg,
sizeof(dataMsg));
              if (result == ESP_OK) {
                Serial.printf("Data sent to MAC: %02X:%02X:%02X:%02X:%02X:%02X\n",
                              discoveredMacs[i][0],              discoveredMacs[i][1],
discoveredMacs[i][2],
                              discoveredMacs[i][3],              discoveredMacs[i][4],
discoveredMacs[i][5]);
              } else {
                Serial.println("ESP-NOW send error");
              }
            }
          } else {
            Serial.println("Search closed. No drivers found.");
          }
        }
      }
```

Лістинг Г.3 — Програма мікроконтролера ESP32-S3-DEV-CAM (сенсорного вузла)