Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____перший (бакалаврський)_____
_____
_____Система управління освітленням в середовищі IoT_____
(тема)

Виконав:
здобувач IV року навчання,
групи __КІУКІ-21-8_____
_____Самодзін Я.В._____
(прізвище, ініціали)

Спеціальність *123 _ Комп'ютерна інженерія*
(код і повна назва спеціальності)
Тип програми __освітньо-професійна_____
_____*Комп'ютерна інженерія*_____
(повна назва освітньої програми)

Керівник _доц. Адамов О.С._
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри АПОТ _____ _Чумаченко С.В
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____перший (бакалаврський)_____

Спеціальність_____123_Комп'ютерна інженерія _____
(код і повна назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Комп'ютерна інженерія _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_____Самодзіну Ярославу Віталійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи ___Система управління освітленням в середовищі IoT _____
_____
_____

затверджена наказом університету від __21__ ____05_____2025 р. № _403Ст_____

2. Термін подання здобувачем роботи до екзаменаційної комісії __16__ ___06_____ 2025 р.

3. Вихідні дані до роботи _____
_____
Програмно-апаратний модуль
_____
_____
_____

4. Перелік питань, що потрібно опрацювати в роботі _____
Аналіз та огляд існуючих систем.
Постановка задачі.
Розробка структурної схеми пристрою.
Розробка функціональної схеми програми.
Розробка алгоритму роботи пристрою.
Тестування
_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
__15 слайдів_____
_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача теми проєкту, узгодження і затвердження теми	08.05.2025 – 09.05.2025	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.05.2025 – 14.05.2025	
3	Розробка структурної схеми пристрою, вибір апаратної платформи	14.05.2025 – 16.05.2025	
4	Розробка фукнціональної схеми програми	16.05.2025 – 17.05.2025	
5	Розробка програмних модулів. Проведення тестування	17.05.2025 – 23.05.2025	
6	Оформлення пояснювальної записки	23.05.2025 – 25.05.2025	
7	Перевірка виконаного проєкту керівником, допуск до захисту	25.05.2025 – 10.06.2025	

Дата видачі завдання _____08.05.2025 р.

Студент _____ _____
(підпис)

Керівник роботи (проекту) _____ _____ доц. Адамов О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 50 сторінки, 16 рисунків, 14 джерел за переліком посилань.

РОЗУМНИЙ ДІМ, МІКРОКОНТРОЛЕР, ІНТЕРНЕТ РЕЧЕЙ, БЕЗДРОТОВИЙ ЗВ'ЯЗОК, РОЗУМНЕ ОСВІТЛЕННЯ

Метою кваліфікаційної роботи є розробка системи розумного освітлення. Система базується на інтеграції апаратних компонентів, таких як мікроконтролери ESP32 та STM32F100, датчиків та виконавчіх пристроїв. Для забезпечення надійного зв’язку між пристроями використано технологію LoRa. Система враховує вплив колірної температури світла на циркадні ритми, сприяючи покращенню сну, концентрації та загального самопочуття.

Програмна частина системи включає веб-додаток для логування даних, розгорнутий на хмарній платформі Amazon Web Services (AWS). Дані з IoT-мережі передаються через LoRa до шлюзу, який надсилає їх до AWS IoT Core для обробки та зберігання. Для реалізації веб-додатку використано сучасні веб-технології, такі як React для фронтенду та Node.js для серверної частини.

ABSTRACT

The explanatory note contains 50 pages, 16 figures, and 14 references in the bibliography.

SMART HOME, MICROCONTROLLER, INTERNET OF THINGS, WIRELESS COMMUNICATION, AUTOMATIC CONTROL, SMART HEATING, SMART LIGHTING, SECURITY SYSTEM, REMOTE CONTROL, WEB INTERFACE, REACT, NODEJS, ESP32, WI-FI, MULTITHREADING, REAL-TIME OPERATING SYSTEMS, AWS, MariaDB, React JS, NodeJS.

The aim of this qualification project is the development of a smart lighting system. The system is based on the integration of hardware components such as ESP32 and STM32F100 microcontrollers, sensors, and actuators. To ensure reliable communication between devices, LoRa technology is used. The system takes into account the effect of light color temperature on circadian rhythms, contributing to improved sleep, concentration, and overall well-being.

The software component includes a web application for data logging, deployed on the Amazon Web Services (AWS) cloud platform. Data from the IoT network is transmitted via LoRa to a gateway, which forwards it to AWS IoT Core for processing and storage. Modern web technologies were used to implement the web application, including React for the frontend and Node.js for the backend.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЦП – аналогово-цифровий перетворювач;

ЕОМ – електронно-обчислювальна машина;

ЕРС – електрорушійна сила;

ІМС – інтегральна мікросхема;

МК – мікроконтролер;

ПЗ – програмне забезпечення;

ПЗП – постійний запам'ятовуючий пристрій;

CRC – cyclic redundancy check (циклічний надлишковий код);

EEPROM – electrically erasable programmable read-only memory (постійний запам'ятовуючий пристрій з електричним стиранням);

GPIO – general purpose input-output (входи-виходи загального призначення);

SWD – serial wire debug (послідовний інтерфейс відлагодження).

ВСТУП

У сучасному світі, де ритм життя постійно прискорюється, а технології стрімко розвиваються, питання підвищення комфорту, здоров’я та енергоефективності в побуті набувають особливої актуальності. Одним із ключових напрямів, що відповідає цим викликам, є створення систем «розумного» будинку, зокрема інтелектуальних систем освітлення. Такі системи, побудовані на основі технологій Інтернету речей (IoT), дозволяють не лише автоматизувати керування освітленням, але й адаптувати його до індивідуальних потреб людини, сприяючи покращенню її фізичного та психоемоційного стану.

Особливу увагу в контексті сучасного способу життя привертає проблема стресу, який є одним із основних факторів, що негативно впливають на здоров’я людини. Постійний вплив стресових ситуацій, викликаних напруженим графіком роботи, інформаційним перевантаженням та недостатньою кількістю відпочинку, призводить до порушення циркадних ритмів, погіршення сну та зниження продуктивності. Освітлення відіграє ключову роль у регулюванні цих ритмів, оскільки колірна температура та інтенсивність світла безпосередньо впливають на вироблення мелатоніну – гормону, відповідального за сон і відпочинок. Неправильно підібране освітлення, особливо в домашніх умовах, може посилювати стрес, викликаючи втому, дратівливість і зниження концентрації.

Використання IoT-технологій у системах розумного освітлення дозволяє створити гнучке та адаптивне середовище, яке враховує як зовнішні фактори (наприклад, рівень природного освітлення), так і поведінку людини (рух, активність). Завдяки інтеграції датчиків освітленості, руху та зміни колірної температури в єдину IoT-мережу, такі системи здатні динамічно налаштовувати параметри світла для створення комфортних умов, що сприяють зниженню стресу та підтримці біологічного ритму. Наприклад,

тепле світло ввечері сприяє релаксації, тоді як холодне світло вранці допомагає активізувати організм.

Крім того, розумне освітлення на основі IoT має значний потенціал для підвищення енергоефективності. Автоматизація дозволяє зменшити споживання електроенергії шляхом вимкнення світла в порожніх приміщеннях або регулювання його інтенсивності залежно від потреб. У поєднанні з можливостями віддаленого керування та аналізу даних через хмарні платформи, такі системи стають не лише зручними, але й економічно вигідними.

Таким чином, розробка програмно-апаратного комплексу для розумного освітлення, що базується на IoT, є актуальним завданням, яке відповідає сучасним викликам. Така система не лише сприяє покращенню якості життя шляхом зниження стресу та оптимізації біоритмів, але й відкриває нові перспективи для впровадження інноваційних технологій у повсякденне життя.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ

Сфера розумного освітлення на основі технологій Інтернету речей (IoT) є однією з найдинамічніших галузей сучасного ринку технологій. Зростання популярності IoT-систем зумовлене їх здатністю підвищувати енергоефективність, комфорт та безпеку, а також сприяти автоматизації побутових і комерційних процесів. За даними аналітичної компанії Mordor Intelligence, обсяг світового ринку IoT у 2024 році оцінюється в 1,17 трлн доларів США, а до 2029 року прогнозується його зростання до 2,37 трлн доларів із середньорічним темпом зростання 15,12%. Сегмент розумного будинку, зокрема освітлення, займає значну частку цього ринку, оскільки попит на автоматизовані системи зростає як серед індивідуальних споживачів, так і в комерційному секторі.

1.1 Сучасні рішення на ринку розумного освітлення

На ринку представлено широкий спектр рішень для розумного освітлення, які можна умовно поділити на три категорії: побутові системи, комерційні системи та системи для розумних міст. Основні гравці на ринку включають такі компанії, як Philips Hue, LIFX, Osram, Xiaomi, TP-Link, а також технологічні гіганти, такі як Google, Amazon і Apple, які інтегрують освітлення в екосистеми розумного будинку.

Найпоширенішими продуктами є «розумні» лампи та світильники, які керуються через мобільні додатки, голосові помічники (Amazon Alexa, Google Assistant, Apple Siri) або автоматизовані сценарії. Наприклад, Philips Hue пропонує лампи з можливістю зміни колірної температури та інтенсивності, які синхронізуються з датчиками руху та освітленості. Система LIFX дозволяє створювати складні сценарії освітлення без додаткового хабу, використовуючи Wi-Fi. Такі системи зазвичай працюють

через Wi-Fi або Bluetooth, що забезпечує простоту встановлення, але може обмежувати масштабування в великих приміщеннях.

Рисунок 1.1 – «Розумні» лампи

У комерційному секторі розумне освітлення застосовується в офісах, торгових центрах і промислових об’єктах. Компанії, такі як Osram і Signify, пропонують системи, що інтегруються з IoT-платформами для моніторингу енергоспоживання та оптимізації освітлення залежно від присутності людей. Наприклад, система Signify Interact Office використовує датчики руху та освітленості для автоматичного регулювання світла, що дозволяє скоротити витрати на електроенергію до 50–80%. Такі рішення часто базуються на протоколах ZigBee або LoRa, які забезпечують надійний зв’язок у великих мережах.

У міських середовищах розумне освітлення використовується для оптимізації вуличного освітлення. Наприклад, проекти в Кардіффі (Велика Британія) та Гуанчжоу (Китай) демонструють використання IoT для керування вуличними ліхтарями, які автоматично регулюють яскравість залежно від часу доби або присутності пішоходів. Такі системи інтегруються з хмарними платформами, що дозволяють централізовано аналізувати дані та прогнозувати потреби в освітленні за допомогою алгоритмів штучного інтелекту.

1.2 Технологічні особливості та протоколи зв’язку

Більшість сучасних систем розумного освітлення використовують бездротові протоколи зв’язку, такі як Wi-Fi, Bluetooth, ZigBee, Z-Wave і LoRa. Wi-Fi є найпоширенішим у побутових системах завдяки простоті інтеграції зі смартфонами, але він споживає більше енергії та має обмеження за кількістю підключених пристроїв. ZigBee та Z-Wave популярні в комерційних системах через низьке енергоспоживання та можливість створення mesh-мереж, що забезпечують стабільний зв’язок на великих відстанях. LoRa, як у запропонованій системі даної роботи, використовується для передачі даних на великі відстані з мінімальними витратами енергії, що робить її ідеальною для розподілених IoT-мереж у розумних будинках або містах.

Розглянемо переваги та недоліки існуючих рішень.

До переваг можна віднести наступне:

– енергоефективність, так як ці датчики дозволяють значно скоротити енергоспоживання (до 50–80% у комерційних системах);

– гнучкість таких систем дзволяє робити налаштування сценаріїв освітлення для різних потреб (робота, відпочинок, сон);

інтеграція, а саме сумісність із голосовими помічниками та іншими системами розумного будинку.

Використання датчиків руху підвищує рівень безпеки в будинку чи офісі.

До недоліків можна віднести таке, як:

– висока вартість, такі як Philips Hue, можуть бути дорогими для масового споживача;

– складність масштабування, тому що Wi-Fi та Bluetooth мережі обмежені за кількістю пристроїв і відстанню;

–IoT-пристрої вразливі до кібератак, якщо не використовується належне шифрування тому проблеми безпекиє також актуальними.

– сумісність систем, так як різні бренди часто використовують власні протоколи, що ускладнює інтеграцію в єдину систему.

1.3 Аналіз потреб ринку

Попит на системи розумного освітлення зростає завдяки підвищенню обізнаності про енергоефективність і вплив світла на здоров’я. За даними Statista, до 2025 року споживчі витрати на розумні будинки перевищать 170 млрд доларів, а кількість домогосподарств із IoT-системами сягне 400 млн. Особливо актуальними є системи, що враховують біоритми людини, оскільки сучасний спосіб життя, пов’язаний зі стресом і порушенням сну, вимагає адаптивного освітлення для підтримки здоров’я. Крім того, зростання популярності хмарних платформ, таких як AWS, сприяє розвитку систем із віддаленим керуванням і аналізом даних, що відповідає потребам як побутових, так і комерційних користувачів.

1.4 Актуальність розробки

Ринок розумного освітлення на основі IoT характеризується високим потенціалом зростання, різноманітністю рішень і технологій. Однак більшість наявних систем орієнтовані на побутовий або комерційний сегмент і мають обмеження щодо масштабування, вартості та безпеки. Запропонована в роботі система, що базується на мікроконтролерах ESP32 і STM32F100, технології LoRa та хмарній платформі AWS, має потенціал подолати ці недоліки, пропонуючи енергоефективне, масштабоване та безпечне рішення для оптимізації біоритмів людини в домашніх умовах. Подальший аналіз ринку підтверджує актуальність розробки таких систем, особливо в контексті зростання попиту на персоналізовані та енергоефективні IoT-рішення.

Назва проекту: Розробка програмно-апаратного комплексу для системи розумного освітлення на основі IoT для оптимізації біоритмів людини.

Мета проекту є створення системи розумного освітлення, яка автоматично регулює інтенсивність, колірну температуру (теплий і холодний світ) та увімкнення/вимкнення освітлення залежно від даних датчиків освітленості, руху та температури, об’єднаних в IoT-мережу за допомогою технології LoRa. Система спрямована на оптимізацію біоритмів людини шляхом вибору правильного відтінку світла для підтримки циркадних ритмів, зниження стресу та підвищення комфорту в домашніх умовах.

Об’єкт розробки – програмно-апаратний комплекс, що включає мікроконтролери ESP32 і STM32F100, датчики, виконавчі пристрої, мережу LoRa та веб-додаток на базі AWS для логування та керування даною системою.

2 ВИМОГИ ДО СИСТЕМИ ОСВІТЛЕННЯ

Підбір елементної бази для апаратної реалізації функціоналу пристрою є ключовим етапом у процесі його розробки. Вибрані компоненти мають забезпечувати належну швидкодію, стабільність у роботі та відповідати вимогам безпеки, при цьому залишаючись максимально економічно вигідними. Крім того, бажано, щоб вони мали певний запас функціональності, який можна використати у майбутньому для розширення або модернізації пристрою. Початковим кроком до створення ефективного рішення є аналіз існуючих аналогів, що дозволяє уникнути типових помилок і врахувати найкращі інженерні практики.

2.1 Функціональні вимоги

Функціональні вимоги передбачають організацію збору даних за допомогою різних типів датчиків. Для визначення рівня як природного, так і штучного освітлення в приміщенні використовуються датчики освітленості. Для автоматичного регулювання освітленням та виявлення присутності людини та її активності в зоні роботи системи необхідно використовувати датчиків руху.

Моніторинг кольорової температури освітлення, що дозволяє визначати поточний відтінок світла в межах від теплого (2700–3500 K) до холодного (5000–6500 K), може бути забезпечений спеціальними датчиками колірної температури. Усі ці сенсори повинні бути підключені до єдиної автоматизованої системи під керуванням мікроконтролера STM32F100, який здійснює обробку зібраної інформації.

2.2 Керування освітленням

Система що розробляється повинна автоматично регулювати яскравість освітлення відповідно до рівня природного світла в приміщенні. Для підтримки біоритмів повинна змінюватися колірна температура: у вечірній час застосовується тепле освітлення в межах 2700–3500 K для створення сприятливої атмосфери відпочинку, а вранці та вдень – холодне світло (5000–6500 K) для стимулювання активності та зосередженості.

Освітлення повинно вмикатися або вимикається автоматично, орієнтуючись на показники датчиків руху. Усі виконавчі пристрої, зокрема світлодіодні лампи або стрічки, повинні бути підключені до мікроконтролера STM32F100, який буде керувати їхньою роботою.

Алгоритми системи повинні автоматично підбирати температуру світла відповідно до часу доби та рівня активності користувача, орієнтуючись на підтримку природних циркадних ритмів людини в приміщені.

Аналізуючи наявність людини та рівень природного освітлення, система повинна створювати комфортні умови, що зменшують рівень стресу й сприяють якісному сну.

2.3 Передача даних

Передача даних повинна здійснюватись наступним чином. Мікроконтролер STM32F100 передає зібрані та оброблені дані до модуля ESP32 за допомогою локального інтерфейсу, наприклад UART або I2C. Далі ESP32 здійснює агрегацію інформації та надсилає її на серверну частину через LoRa-шлюз до хмарного сервісу AWS IoT Core. Технологія LoRa забезпечує стабільний зв’язок на відстанях до 1–2 км у міських умовах із мінімальним енергоспоживанням, тому її доцільно використовувати в таких проєктах.

Для моніторингу й аналізу даних використовується веб-застосунок, побудований на базі AWS. Веб-додаток на базі AWS забезпечує:

– моніторинг даних із датчиків у реальному часі (освітленість, рух, колірна температура);

– зберігання логів у Amazon S3;

– аналіз даних за допомогою AWS Lambda для автоматичного налаштування сценаріїв освітлення;

– інтерфейс для ручного керування параметрами світла (інтенсивність, відтінок).

У реальному часі можна спостерігати за показниками освітленості, руху та колірної температури. Журнали подій зберігаються в Amazon S3, а функції AWS Lambda аналізують інформацію та коригують сценарії освітлення. Крім того, доступний інтерфейс для ручного регулювання інтенсивності та відтінку світла, а саме система автоматично обирає відтінок світла (теплий або холодний) залежно від часу доби та активності людини, щоб підтримувати циркадні ритми.

Структуровані дані про функціонування системи зберігаються в Amazon DynamoDB.

3 РОЗРОБКА ПРОГРАМНО-АПАРАТНОЇ ЧАСТИНИ ПРОЕКТУ

3.1 Розробка блок схеми програмно апаратної частини системи керування розумним освітленням на базі мережі IoT.

Для розробки системи розумного освітлення була створена блок схема, що відображає всі вузли данного пристрою, она наведена на рисунку 3.1 нижче.

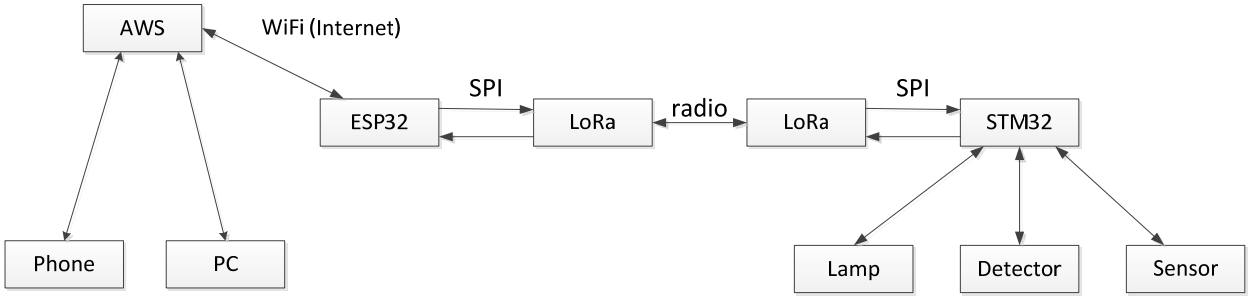

Рисунок – 3.1 Схема запропонованої системи

Система складається з трьох основних модулів, що розділені за їх логікою роботи, а саме:

– головний концентратор та модуль зв'язку, що приймає рішення стосовно логіки керування освітленням, він має постійне з'єднання з мережею сенсорів та виконуючих пристроїв, що являє собою мережу IoT пристроїв об'єднаних за допомогою LoRa, також він має з'єднання з хмарним сервісом за допомогою WiFi, що допомагає дистанційно керувати системою за допомогою WEB додатку;

– мережеві вузли, сенсори та пристрої керування світлом, кожен з них має з'єднання з концентратором за допомогою мережі LoRa та приймає конфігурацію власної логіки;

– хмарний сервіс що приймає дистанційні команди від користувача та передає їх на концентратор за допомогою Internet. Також він отримує статус системи та передає його на пристрій користувача за допомогою WEB додатку

Для побудови даної системи потрібно вибрати елементну базу що відповідає вимогам описаного вище технічного завдання. А саме, модуль концентратора повинен мати можливість під'єднання до мережі Internet за допомогою WiFi та можливість під'єднання до мережі LoRa також він має бути компактним та мати високу енергоефективність.

Для виконавчих пристроїв це повинні бути контролери що мають можливість під'єднання до мережі LoRa, низьке споживання енергії та можуть виконувати аналіз стану сенсорів та керувати перемикання світла

Для забезпечення видаленого доступу та зберігання інформації потрібно побудувати веб сервер з можливістю обміну даними з концентратором. Такий сервіс повинен мат можливість запуску коду сервера на та зберігання даних у базі даних.

3.2 Концентратор на базі ESP32

В якості концентратора було вирішено обрати радіомодуль на базі ESP-32 WRoom. Він має на борту вбудований двухдіапазоний WiFi модуль, що працює на частоті 2.4 ГГц і двуядерний процесор відкритої архітектури Xteensa LX6 з тактовою частотою 240 МГц. Також він має послідовні інтерфейси зв'язку такі як SPI, що необхідні для комунікації з модулем радіозв’язку LoRa, який буде детально описаний нижче. На рисунку 3.2 приведені разпіновка даного модуля. Також важливим є той факт що в ньому можна легко встановити операційну систему FreeRTOS, за допомогою якої набагато легше створити та підтримувати зв'язок з хмарним сервісом.

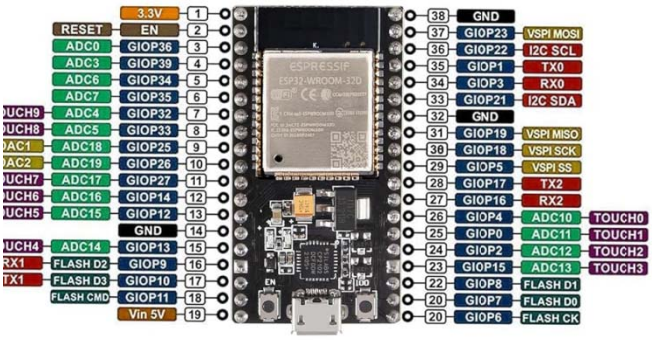

Рисунок – 3.2 Схема призначення виводів ESP32 DevKit v1

Модуль ESP-32 має на борту ще багато додаткових функціональних модулів, що не будуть розглядатися в даній роботі за відсутності вимог, але нище наведена повна блок-діаграма внутрішньої побудови цього модуля для кращої демонстрації його можливостей на випадок розширення функціональних можливостей проекту. Схема наведена на рисунку 3.3.

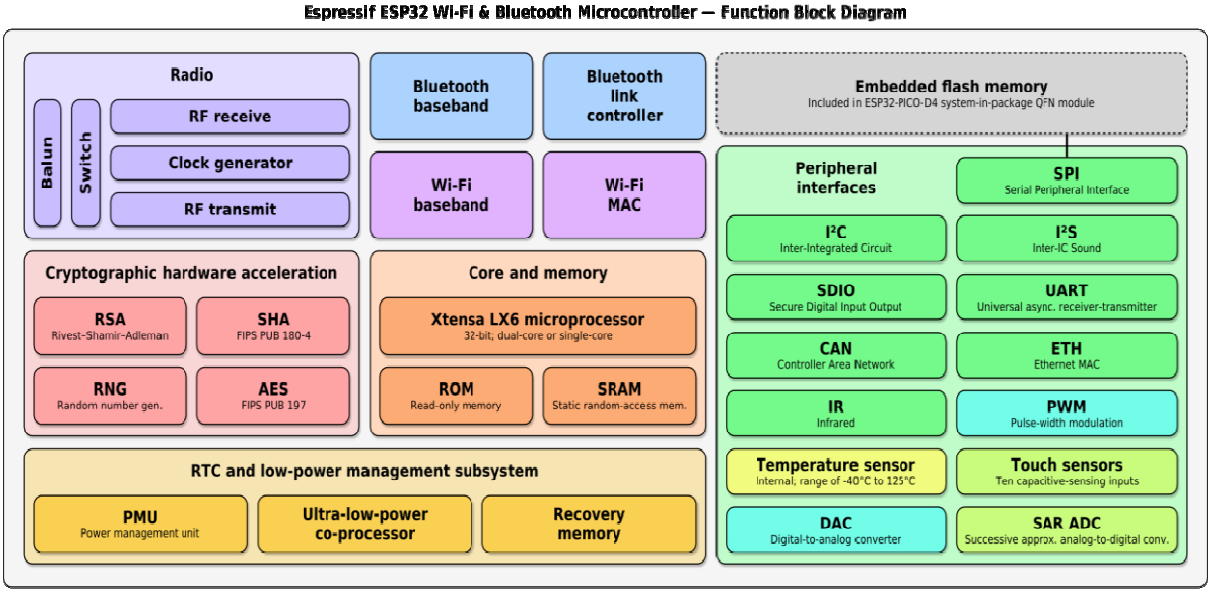

Рисунок 3.3 – Функціональна схема

Також можна відмітити що даний модуль став майже стандартом при розробці вбудованих систем у сучасному світі завдяки тому що він має можливість зв'язку з WiFI + Bluetooth, має невелику ціну та дуже важливим є

ще той факт що він має сертифікацію по сучасних стандартах зв'язку та випромінювання, що дозволяє його використання без додаткових затраних випробувань.

3.3 Радіомодуль зв'язку LoRa

LoRa (Long Range) – це технологія бездротового зв’язку, яка дозволяє передавати дані на великі відстані з мінімальним енергоспоживанням. Вона працює на низьких частотах (зазвичай 433 або 868 МГц) і використовується в мережах Інтернету речей (IoT), де важливо забезпечити стабільний зв’язок між пристроями, навіть у важкодоступних або розподілених локаціях.

LoRa ідеально підходить для застосувань, де потрібна передача невеликих обсягів даних, але на великі відстані – наприклад, у «розумному» місті, агросекторі, моніторингу довкілля або системах автоматизації будинків що саме і підходить під вимоги роботи. Основна ідея передачі даних в LoRa базується на методі широкосмугової частотної модуляції з розтягуванням спектра (Chirp Spread Spectrum, CSS). Вона забезпечує низькошвидкісну передачу даних (від 0.3 до 50 кбіт/с) на великі відстані — до 15 км у сільській місцевості та 2–5 км у міських умовах.

LoRa працює на неліцензованих частотах ISM-діапазону:

– 433 МГц (регіонально, зокрема в Азії);

– 868 МГц (Європа);

– 915 МГц (Америка).

Основні технічні характеристики:

– мала потужність передавача (звичайно 14–20 dBm), що дозволяє працювати на батарейках до 10 років;

– висока завадостійкість завдяки CSS-модуляції;

– типова пропускна здатність: залежить від фактору розширення (spreading factor, SF7–SF12).

Також модулі LoRa мають зручний для підключення до ESP-32 інтерфейс SPI що робить вибір дуже оптимальним в умовах даної роботи. Нижче на рисунку 3.4 приведене зображення даного модуля.

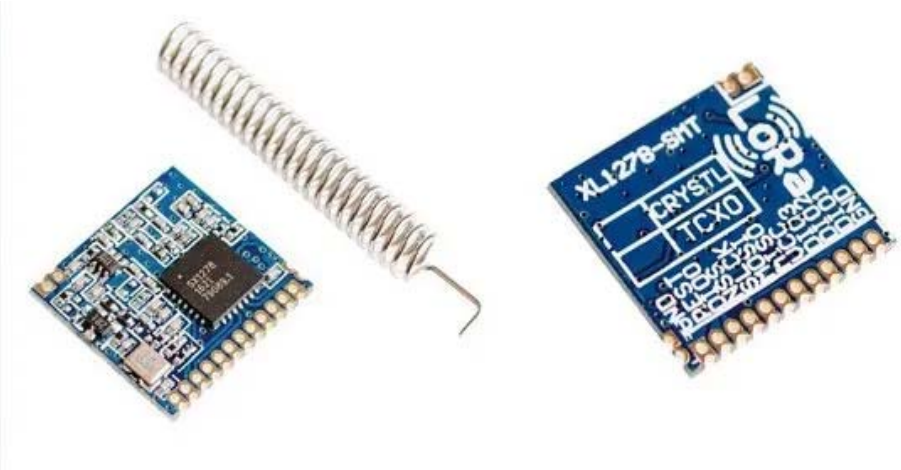

Рисунок 3.4 – Модуль зв'язку LoRa

3.4 Виконавчі пристрої (мережеві вузли)

У якості вузлових модулів, що будуть виконувати функцію керування світлом на місцях було прийнято рішення використати мікроконтролер STM-32 сотої серії.

STM32F100 – це бюджетна серія мікроконтролерів від STMicroelectronics на основі ядра ARM Cortex-M3, яка працює на частоті до 24 МГц, має до 128 КБ флеш-пам’яті та до 8 КБ оперативної пам’яті.

Вона підтримує базові периферійні інтерфейси, такі як USART, SPI, I2C, ADC, таймери, а також забезпечує стабільну роботу при живленні 2.0–3.6 В. Завдяки своїй простоті, енергоефективності та низькій вартості, ця серія ідеально підходить для недорогих вбудованих систем, побутової електроніки та базових IoT-рішень. Для спрощення монтажних робіт був обраний модуль на базі цього контроллеру Blue-Pill що наведений на рисунку 3.5.

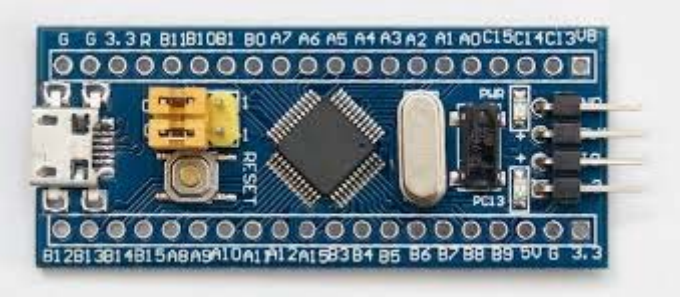

Рисунок 3.5 – STM32F100

3.5 Сенсор руху

Датчик руху в системі розумного освітлення застосовується для виявлення присутності людини в приміщенні, що дозволяє автоматично вмикати або вимикати світло, економити енергію та підвищувати комфорт користувача. Датчик руху (PIR HC-SR501), підключений до STM32F100, працює за принципом виявлення інфрачервоного випромінювання від рухомих об’єктів (людини). При зміні ІЧ-сигналу датчик генерує сигнал, який STM32F100 обробляє, активуючи освітлення або передаючи дані через до ESP-32 у мережу LoRa для подальшого керування системою.

Рисунок 3.6 – Інфрачервоний датчик руху

3.6 Сенсор освітлення

У якості детектору освітлення використовується фоторезистор. Фоторезистор виготовлений із напівпровідникового матеріалу (наприклад,

сульфіду кадмію), який реагує на світло. При високому рівні освітленості (наприклад, денне світло) опір фоторезистора зменшується (до кількох десятків Ом), а в темряві або при низькому освітленні опір значно зростає (до кількох МОм). Це дозволяє використовувати його для вимірювання інтенсивності світла.

На рисунку 3.7 наведено фотографію даного пристрою. У запропонованій системі фоторезистор підключений до STM32F100, який зчитує аналоговий сигнал (через АЦП), перетворює його в цифровий формат і передає через ESP32 у мережу LoRa. На основі цих даних система визначає, чи потрібно вмикати світло, регулювати його яскравість або змінювати колірну температуру для підтримки біоритмів.

Рисунок 3.7 – Датчик освітлення

3.7 Хмарний сервіс дистанційного керування

Для створення хмарного сервісу запропоновано використати популярний сервіс від компанії Amazon AWS EC2. Amazon EC2 (Elastic Compute Cloud) – це хмарний сервіс від AWS, який надає масштабовані віртуальні сервери (інстанси) для запуску додатків. Користувачі можуть вибирати конфігурацію процесора, пам’яті, сховища та мережі, платити лише за використані ресурси, а також автоматично масштабувати потужності залежно від потреб. Для даної роботи було обрано базову модель під ОС Linux.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

4.1 Реалізація базового модуля (Концентратора)

4.1.1 Підключення LoRa до ESP-32

Для забезпечення зв'язку концентратора з вузлами мережі у даній роботи була використана мережа LoRa, модулі LoRa мають власний фізичний інтерфейс зв'язку для взаємодії з контролерами за допомогою шини послідовної передачі даних SPI. На рисунку 3.1.1 наведена схема підключення модуля до контролера ESP-32.

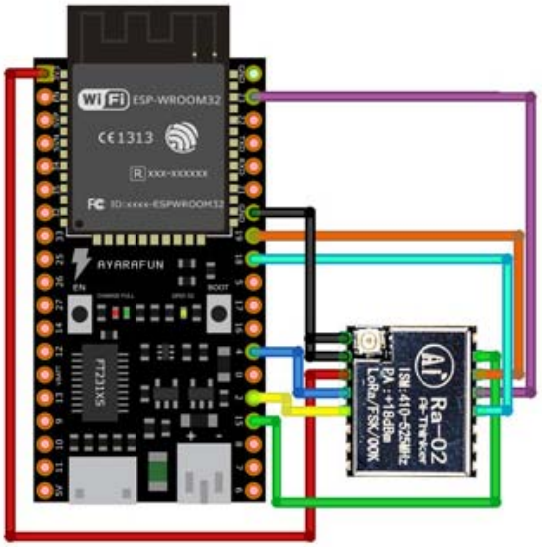

Рисунок 4.1 – Схема підключення LoRa до ESP-32

Ця шина використовується для налаштування модуля на потрібну частоту, вибору режимів роботи та прийому і передачі даних. Запис та зчитування відбувається за допомогою доступу до внутрішніх регістрів чипу всередині модуля LoRa. Протокол виглядає наступним чином: спочатку йде байт адреси реєстрації у якому старший біт вказує на операцію запису або

зчитування а потім в залежності від обраного режиму проходить або запис або читання даних. Часова діаграма вказана на рисунку 4.2.

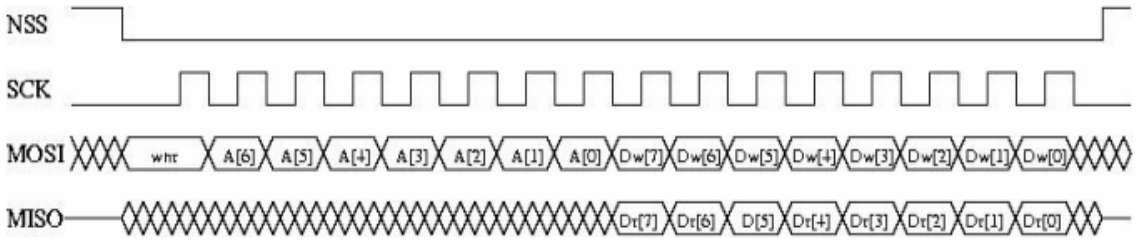

Рисунок 4.2 – Часова діаграма обміну даних по SPI

Для того щоб налаштувати модуль та забезпечити коректну роботу з даними був створений драйвер на мові С. Нижче йде його фрагмент з детальним описом.

```
typedef struct{
   uint8_t *frequency;
   uint8_t txPower;
   uint8_t bandWidth;
   uint8_t codingRate;
   uint8_t spreadFactor;
   uint8_t crcEnable;
   uint8_t headerType;
}loraConfigStruct;
```

Вказаний вище код, це структура для налаштування модуля на потрібний користувачу режим роботи, кожній байт відповідає за окрему функцію, їх назва англійською відповідає призначенню даного байта. Для комфортної взаємодії була створена таблиця значень з яких можна вибирати режими. Запис до модулю здійснюється за допомогою передачі цієї структури у модуль, за допомогою високорівневої функції, що наведена нижче з потрібними нам налаштуваннями. Для зручності код винесено в окрему функцію для кращого формування коду.

```
#
void setupLoRa() {
 loraConfigStruct loraConfig;
```

```
 loraConfig.frequency = sx1276_7_8FreqTbl;
 loraConfig.txPower = PWR_11DBM;
 loraConfig.bandWidth = BW_500KHZ;
 loraConfig.codingRate = CR_4_5;
 loraConfig.headerType = IMP_HEADER;
 loraConfig.spreadFactor = SF_RATE_12;
 loraConfig.crcEnable = CRC_EN;

 sx1276_7_8_Config(&loraConfig);
}
```

Для комунікаційних потреб було створено наступні функції, що служать для відправки пакета даних і прийому пакета. Функції для прийому є дві, одна з очікуванням без ліміту часу інша з таймером, по закінченню інтервалу повертає код помилки за очікуванням і не блокує код. Також помічні функції що переводять модуль в режим прийому або передачі. Перелік їх наведено нижче.

```
#
uint8_t sx1276_7_8_LoRaEntryRx(uint8_t packLenth, uint32_t
timeOut);
uint8_t sx1276_7_8_LoRaRxWaintPacket(uint8_t *rxData);
uint8_t sx1276_7_8_LoRaRxPacketTimeOut(uint8_t *rxData,
uint32_t timeOut);

uint8_t sx1276_7_8_LoRaEntryTx(uint8_t packLen, uint32_t
timeOut);
uint8_t sx1276_7_8_LoRaTxPacket(uint8_t* txBuffer, uint8_t
packLen,   uint32_t timeOut);
#
```

4.1.2 Головна частина коду концентратору

Код написано з використанням фреймворку Arduino що допомагає зменшити витрати часу на низькорівневі налаштування апаратної частини. Код починає виконуватися з функції setup що проводе базову ініціалізацію системи. Код приведено нижче:

```
void setup() {
   Serial.begin(115200);
   setupWiFi();
   setupLoRa();
   setupWebSocket();
```

```
    setupNTP();
}
```

Для того щоб тримати всі налаштування в одному місці були створені початкові налаштування у вигляді констант:

```
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* serverUrl = "http://your-server.com/api";
const char* wsHost = "your-server.com";
const uint16_t wsPort = 80;
const char* wsPath = "/ws";
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 7200; // UTC+2 for Ukraine
const int daylightOffset_sec = 0;
```

Функція Serial.begin(115200) ініціалізує послідовний порт для виведення налагоджувальної інформації зі швидкістю 115200 бод.

Функція setupWiFi() підключає ESP32 до Wi-Fi мережі, використовуючи зада Burger заданий SSID і пароль, забезпечуючи доступ до Інтернету для REST API та WebSocket. Код приведено нижче.

```
void setupWiFi() {
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
    }
}
```

Функція setupLoRa() налаштовує модуль LoRa (SX1276/7/8) із заданими параметрами, такими як частота, потужність, ширина смуги, для роботи в мережі LoRa. Детально розглядається у розділі про підключення LoRa.

Функція setupWebSocket() встановлює WebSocket-з’єднання з сервером за вказаною адресою, портом і шляхом, налаштовуючи обробник подій і повторне підключення. Функція webSocketEvent є обробником подій

WebSocket у програмі для ESP32, яка керує розумним освітленням. Вона відповідає за обробку вхідних повідомлень і подій, пов’язаних із WebSocket-з’єднанням.

```
void setupWebSocket() {
   webSocket.begin(wsHost, wsPort, wsPath);
   webSocket.onEvent(webSocketEvent);
   webSocket.setReconnectInterval(5000);
}
```

Функція setupNTP() синхронізує час із NTP-сервером для точного визначення часу, необхідного для розкладу зміни колірної температури світла.

```
void setupNTP() {
   configTime(gmtOffset_sec, daylightOffset_sec,
ntpServer);
}
```

Далі йде головне коло де проходе вся логіка, а саме, опитування усіх вузлів, передача їх стану на хмарный сервер та очікування комод від нього. Код розміщено у функції loop, що є головним тілом програм на Arduino.

```
void loop()
{
   webSocket.loop();
   checkLightSchedule();
   if (millis() - lastPoll >= POLL_INTERVAL)
   {
       for (int devId = 0; i < DEVICE_COUNT; devId++)
       {
           lastPoll = millis();
           uint8_t txBuffer[8];
           txBuffer[0] = (SYSTEM_ID >> 24) & 0xFF;
           txBuffer[1] = (SYSTEM_ID >> 16) & 0xFF;
           txBuffer[2] = (SYSTEM_ID >> 8) & 0xFF;
           txBuffer[3] = SYSTEM_ID & 0xFF;
           txBuffer[4] = devId;
           txBuffer[5] = 'S';
           txBuffer[6] = 0x00;
           txBuffer[7] = 0x00;
           sx1276_7_8_LoRaEntryTx(8, 1000);
```

```
                sx1276_7_8_LoRaTxPacket(txBuffer, 8, 1000);
                uint8_t rxData[2];
                sx1276_7_8_LoRaEntryRx(2, 5000);
                if (sx1276_7_8_LoRaRxPacketTimeOut(rxData, 5000)
== 0)
                {
                    bool state = rxData[0] == 1;
                    sendSensorData(state);
                }
            }
        }
    }
```

В цьому коді формується пакет для відправки на простої, перші чотири байта це унікальний ідентифікатор мережі, щоб не було конфліктів з іншими пристроями якщо вони є поряд, потім ідентифікатор вузла що опитують, потім команда ‘S’, що означає статус. По отриманню даних вони передаються на сервер. Код передачі статусу приведено нижче.

```
void sendSensorData(uint8_t state)
{
   if (WiFi.status() == WL_CONNECTED)
   {
       HTTPClient http;
       http.begin(serverUrl);
       http.addHeader("Content-Type", "application/json");
       StaticJsonDocument<200> doc;
       doc["sensorId"] = DEVICE_ID;
       doc["state"] = state;
       String payload;
       serializeJson(doc, payload);
       int httpCode = http.POST(payload);
       http.end();
   }
}
```

Функція sendSensorData відправляє дані про стан освітлення на віддалений сервер через REST API. Вона приймає параметр state (тип uint8_t), який вказує, увімкнене чи вимкнене світло (1 = увімкнено, 0 = вимкнено). Спочатку перевіряється статус Wi-Fi з’єднання за допомогою WiFi.status(). Якщо з’єднання активне, створюється об’єкт HTTPClient для

виконання HTTP-запиту. Функція ініціалізує з’єднання з сервером, вказаним у serverUrl, і встановлює заголовок Content-Type: application/json. Далі формується JSON-об’єкт за допомогою StaticJsonDocument, куди записуються sensorId (ідентифікатор пристрою, константа DEVICE_ID) і state. JSON серіалізується в рядок payload, який відправляється на сервер методом POST. Після отримання HTTP-коду відповіді (зберігається в httpCode), з’єднання завершується викликом http.end(). Функція забезпечує надійну передачу даних про стан системи до сервера для логування або подальшої обробки.

При отриманні команди від користувача через хмарний сервіс команда надходить через Інтернет завдяки websocket, потрапляючи у функцію обробник. Що надсилає команду вузлам з діями котрі вони повинні виконати, або з розкладом, що має перемикати режим холодного або теплого освітлення.

```
     void webSocketEvent(WStype_t type, uint8_t *payload, size_t
length)
     {
        switch (type)
        {
        case WStype_DISCONNECTED:
            break;
        case WStype_CONNECTED:
            break;
        case WStype_TEXT:
            StaticJsonDocument<200> doc;
            deserializeJson(doc, payload);
            if (doc.containsKey("command"))
            {
                String command = doc["command"];
                if (command == "setLight")
                {
                    uint8_t targetId = doc["sensorId"];
                    bool state = doc["state"];
                    uint8_t lightMode = doc["mode"];

                    uint8_t txBuffer[8];
                    txBuffer[0] = (SYSTEM_ID >> 24) & 0xFF;
                    txBuffer[1] = (SYSTEM_ID >> 16) & 0xFF;
                    txBuffer[2] = (SYSTEM_ID >> 8) & 0xFF;
                    txBuffer[3] = SYSTEM_ID & 0xFF;
```

```
                    txBuffer[4] = targetId;
                    txBuffer[5] = 'C';
                    txBuffer[6] = state ? 1 : 0;
                    txBuffer[7] = lightMode;

                    do{
                        sx1276_7_8_LoRaEntryTx(3, 1000);
                        sx1276_7_8_LoRaTxPacket(txBuffer, 3,
1000);
                    while (sx1276_8_8_LoRaWaitResp(1000));

                }
                else if (command == "setSchedule")
                {
                    schedule.warmHour = doc["warmHour"];
                    schedule.warmMinute = doc["warmMinute"];
                    schedule.coldHour = doc["coldHour"];
                    schedule.coldMinute = doc["coldMinute"];
                    schedule.scheduled = true;
                }
            }
            break;
        }
    }
```

Формат пакету схожий на отримання статуса але команда надається ‘C’ що вказує на те що, це зміна режиму, після цього йдуть два байта, перший – стан, другий – кольорова температура освітлення.

Функція checkLightSchedule відповідає за перевірку розкладу автоматичного переходу на теплий або холодний світ на основі поточного часу. Якщо прапор schedule.scheduled встановлено в false, функція завершується, не виконуючи жодних дій. У разі активного розкладу вона отримує поточний час через виклик time(&now) і конвертує його в локальний час за допомогою localtime_r, витягуючи години та хвилини.

Далі порівнюється поточний час із заданими в структурі schedule значеннями warmHour:warmMinute (для теплого світла) та coldHour:coldMinute (для холодного світла). Якщо поточний час збігається з часом для теплого світла, викликається broadcastLightMode(0), щоб надіслати команду всім вузлам через LoRa для перемикання на теплий світ. Аналогічно, якщо час відповідає холодному світлу, викликається

broadcastLightMode(1) для перемикання на холодний світ. Функція забезпечує автоматичну зміну колірної температури відповідно до розкладу, отриманого з сервера через WebSocket.

```
void checkLightSchedule()
{
   if (!schedule.scheduled)
       return;

   time_t now;
   struct tm timeinfo;
   time(&now);
   localtime_r(&now, &timeinfo);

   int currentHour = timeinfo.tm_hour;
   int currentMinute = timeinfo.tm_min;

   if (currentHour == schedule.warmHour && currentMinute ==
schedule.warmMinute)
   {
       broadcastLightMode(0); // Warm light
   }
   else if (currentHour == schedule.coldHour &&
currentMinute == schedule.coldMinute)
   {
       broadcastLightMode(1); // Cold light
   }
}
```

Це є базовий функціонал концентратора, що дозволяє обробляти всі команди від користувача. У майбутньому можна додати систему діагностики вузлів, та додати додатковий шар безпеки. У даному прототипи ці частини свідомо спрощені.

4.2 Вузловий модуль

Вузловий модуль являє собою систему з контролера STM32f100, сенсору руху, модуля LoRa та кнопки примусового керування світлом. Код написано мовою С. Конфігурація зроблена у конфігураторі ST CUBEMX що поставляється з контролерами фірмою виробником. На рисунку 3.2.1

показана схема підключення LoRa до STM32 за допомогою SPI шини. Принцип той самий що і вище розглянутому приклада на ESP-32, Тому детально розглядатися не буде.

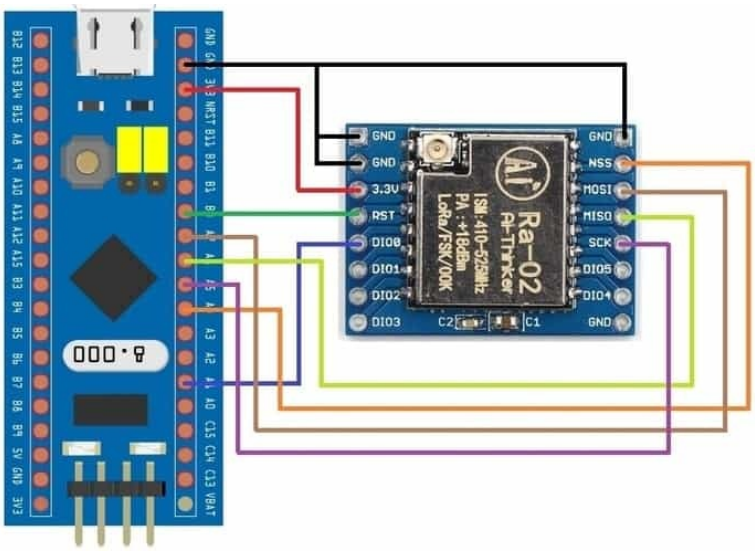

Рисунок 4.3 – Підключення LoRa до Stm32

Програма складається з блоку ініціалізації модуля зв'язку, налаштування портів вводу виводу для приєднання сенсору освітлення, сенсору руху та керування двома типами освітлення, теплий та холодний, що фізично перемикаються модулем керування. Також кнопки, що вмикає або вимикає світло за бажанням користувача, не зважаючі на зовнішні інструкції.

Функція MX_GPIO_Init налаштовує піни GPIO на STM32 для роботи з датчиком руху, кнопкою та світлодіодом/реле. Вона активує тактування порту GPIOA, ініціалізує пін PA2 (світлодіод) як вихід із низькою швидкістю та без підтягування, встановлюючи його початково вимкненим. Піни PA0 (датчик руху) та PA1 (кнопка) налаштовуються як входи з перериванням по зростанню сигналу без підтягування. Функція також активує переривання для пінів PA0 і PA1 через NVIC із високим пріоритетом, забезпечуючи реакцію на події від датчика руху та кнопки.

```
void MX_GPIO_Init(void)
```

```
{
   GPIO_InitTypeDef GPIO_InitStruct = {0};

   __HAL_RCC_GPIOA_CLK_ENABLE();

   HAL_GPIO_WritePin(GPIOA, LED_PIN, GPIO_PIN_RESET);

   GPIO_InitStruct.Pin = MOTION_PIN | BUTTON_PIN;
   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
   GPIO_InitStruct.Pull = GPIO_NOPULL;
   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

   GPIO_InitStruct.Pin = LED_PIN;
   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
   GPIO_InitStruct.Pull = GPIO_NOPULL;
   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

   HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
   HAL_NVIC_EnableIRQ(EXTI0_IRQn);
   HAL_NVIC_SetPriority(EXTI1_IRQn, 0, 0);
   HAL_NVIC_EnableIRQ(EXTI1_IRQn);
}
```

Функція handleLoRaPacket обробляє вхідний LoRa-пакет розміром 8 байтів, що містить SYSTEM_ID, devId, символ 'S' та два нульові байти. Вона отримує пакет через sx1276_7_8_LoRaRxPacketTimeOut із таймаутом 5000 мс, перевіряє відповідність SYSTEM_ID, devId і ‘S’ або ‘C’. Якщо пакет валідний, формує відповідь із 2 байтів: поточний стан світла lightState і режим (за замовчуванням 0), та надсилає її через sx1276_7_8_LoRaTxPacket із таймаутом 1000 мс. Якщо надійшла команда то обробляє її налаштовуючи режим світла та вмикаючи або вимикаючи світло дистанційно.

```
void handleLoRaPacket(void)
{
   uint8_t rxBuffer[8];
   sx1276_7_8_LoRaEntryRx(8, 5000);
   if (sx1276_7_8_LoRaRxPacketTimeOut(rxBuffer, 5000) == 0)
   {
       uint32_t receivedSystemId = ((uint32_t)rxBuffer[0]
<< 24) | ((uint32_t)rxBuffer[1] << 16) | ((uint32_t)rxBuffer[2]
<< 8) | rxBuffer[3];
       uint8_t receivedDevId = rxBuffer[4];
       if (receivedSystemId == SYSTEM_ID && receivedDevId
== DEVICE_ID && rxBuffer[5] == 'S')
```

```
            {
                uint8_t txBuffer[2];
                txBuffer[0] = lightState;
                txBuffer[1] = 0;
                sx1276_7_8_LoRaEntryTx(2, 1000);
                sx1276_7_8_LoRaTxPacket(txBuffer, 2, 1000);
            } else {
                lightState = rxData[0];
                HAL_GPIO_WritePin(GPIOA, LED_PIN, lightState ?
GPIO_PIN_SET : GPIO_PIN_RESET);
            }
        }
    }
```

Для обробки сигналу від сенсору руху и кнопки були створені обробники подій HAL_GPIO_EXTI_Callback, що реагують на зміну логічного рівня на пінах куди вони приєднані. Код цього фрагмента наведено нижче.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == MOTION_PIN)
  {
     handleMotionSensor();
  }
  else if (GPIO_Pin == BUTTON_PIN)
  {
     handleButtonPress();
  }
}
```

MOTION_PIN відповідає за спрацьовування сенсору руху, що вмикає світло і чекає дві хвилини потім вимикає світло, якщо знову не було зафіксовано новий рух. BUTTON_PIN примусово вмикає або вимикає світло як звичайний перемикач.

4.3 Серверна частина дистанційного керування

Серверна частина системи дистанційного керування розумним освітленням забезпечує централізовану обробку запитів від клієнтів,

управління станом пристроїв і передачу команд до ESP32 через WebSocket та REST API. Вона реалізована з використанням сучасних веб-технологій, що гарантують високу продуктивність, масштабованість і зручність інтеграції з апаратною частиною.

Фронтенд-частина розроблена на основі фреймворку React, який забезпечує створення реактивного та інтерактивного користувацького інтерфейсу. React дозволяє організувати компонентну архітектуру, що полегшує розробку та підтримку веб-додатку. Користувач через веб-інтерфейс може вмикати/вимикати світло, змінювати колірну температуру (теплий/холодний) та налаштовувати розклад автоматичних переходів. Для стилізації використовується Tailwind CSS, що забезпечує швидке створення адаптивного дизайну. Запити до серверної частини надсилаються через REST API (наприклад, для передачі стану датчиків) і WebSocket (для команд у реальному часі, таких як увімкнення світла чи зміна розкладу).

Бекенд-частина побудована на фреймворку Express.js, який працює на платформі Node.js. Express забезпечує легку маршрутизацію HTTP-запитів і обробку WebSocket-з’єднань через бібліотеку ws. Сервер обробляє REST-запити від ESP32, що містять дані про стан датчиків (наприклад, JSON { "sensorId": 1, "state": true }), і надсилає команди до пристроїв через WebSocket (наприклад, { "command": "setLight", "sensorId": 1, "state": true, "mode": 0 }). Завдяки асинхронній природі Node.js, сервер ефективно обробляє численні одночасні з’єднання, що критично для IoT-систем із багатьма пристроями.

Для зберігання стану системи використовується Redis — високопродуктивна in-memory база даних. Redis зберігає поточний стан кожного пристрою (ввімкнено/вимкнено, режим світла) і розклад переходів на теплий/холодний світ (наприклад, { warmHour: 20, warmMinute: 0, coldHour: 7, coldMinute: 0 }). Використання Redis забезпечує швидкий доступ до даних і їхню консистентність навіть при частих оновленнях. Дані в Redis оновлюються при кожній зміні стану (через REST або WebSocket) і синхронізуються з апаратною частиною через ESP32.

Інтеграція фронтенду, бекенду та апаратної частини здійснюється через чітко визначений API. REST API обробляє періодичні запити від ESP32, що надсилаються раз на хвилину з даними датчиків. WebSocket забезпечує двосторонній зв’язок у реальному часі, дозволяючи серверу надсилати команди до ESP32 (наприклад, увімкнення світла за ID) і отримувати підтвердження. Такий підхід мінімізує затримки та забезпечує миттєве оновлення інтерфейсу користувача при зміні стану пристроїв.

Безпека системи реалізована через використання HTTPS для REST API та захисту WebSocket-з’єднань. Аутентифікація користувачів на фронтенді може бути додана через токени (наприклад, JWT), що зберігаються в Redis, хоча в базовій версії це не реалізовано для спрощення. Сервер розгортається на хмарній платформі, такій як AWS EC2, що забезпечує масштабованість і доступність системи.

Таким чином, серверна частина поєднує React для зручного інтерфейсу, Express.js для ефективної обробки запитів і Redis для швидкого зберігання стану, створюючи надійну та масштабовану систему дистанційного керування розумним освітленням.

4.4 Програмна реалізація серверу

Серверна частина системи дистанційного керування розумним освітленням розроблена для забезпечення централізованого управління пристроями, обробки даних від апаратних компонентів (ESP32) та надання зручного інтерфейсу для користувача. Код реалізований на Node.js із використанням фреймворку Express.js для створення REST API, бібліотеки ws для WebSocket-з’єднань і Redis для зберігання стану пристроїв та розкладу. Основна мета – забезпечити швидку, надійну та масштабовану взаємодію між фронтендом, апаратною частиною та сервером.

Код є базовою реалізацією, яка може бути розширена додатковими функціями, такими як аутентифікація, логування чи розгортання на хмарних

платформах (наприклад, AWS EC2). Для запуску потрібні встановлені Node.js, Redis і залежності (express, ws, redis). Нижче розглянемо реалізацію.

Підключення всіх модулів на мові JavaScript необхідних для покриття функціоналу. Під'єднання до бази даних Redis, створення сутності сервера та обробника JSON формату.

```
const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const redis = require('redis');

const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
const redisClient = redis.createClient({ url:
'redis://localhost:6379' });

app.use(express.json());

redisClient.connect().catch(console.error);
```

Обробники REST запитів та короткий опис їх функціональності:

– POST /api отримує дані від ESP32 ({ sensorId, state }), зберігає стан у Redis (у хеші devices) і повертає статус 200;

– GET /api/schedule повертає поточний розклад із Redis;

– POST /api/schedule зберігає новий розклад у Redis і надсилає його всім підключеним клієнтам через WebSocket;

– POST /api/light оновлює стан і режим пристрою в Redis, надсилає команду { command: "setLight", sensorId, state, mode } через WebSocket.

Формат JSON даних наведено нижче:

– команда на встановлення розкладу часу зміни режимів світла:

```
{ "command": "setSchedule", "warmHour": 20, "warmMinute":
0, "coldHour": 7, "coldMinute": 0 };
```

– команда на вмикання світла для пристрою з номером один:

```
{ "command": "setLight", "sensorId": 1, "state": true,
"mode": 0 }.
```

Код обробки вищеописаних запитів мовою JavaScript з використанням бібліотеки expresJS.

```
     app.post('/api', async (req, res) => {
        const { sensorId, state } = req.body;
        await redisClient.hSet('devices', sensorId,
JSON.stringify({ state }));
        res.status(200).send({ status: 'ok' });
     });

     app.get('/api/schedule', async (req, res) => {
        const schedule = await redisClient.get('schedule');
        res.status(200).send(schedule ? JSON.parse(schedule) :
{});
     });

     app.post('/api/schedule', async (req, res) => {
        const schedule = req.body;
        await redisClient.set('schedule',
JSON.stringify(schedule));
        wss.clients.forEach((client) => {
            if (client.readyState === WebSocket.OPEN) {
                client.send(JSON.stringify({ command:
'setSchedule', ...schedule }));
            }
        });
        res.status(200).send({ status: 'ok' });
     });

     app.post('/api/light', async (req, res) => {
        const { sensorId, state, mode } = req.body;
        await redisClient.hSet('devices', sensorId,
JSON.stringify({ state, mode }));
        wss.clients.forEach((client) => {
            if (client.readyState === WebSocket.OPEN) {
                client.send(JSON.stringify({ command:
'setLight', sensorId, state, mode }));
            }
        });
        res.status(200).send({ status: 'ok' });
     });
```

Обробник WebSocket обробляє підключення клієнтів (ESP32). При отриманні повідомлення з командою getState повертає стан пристрою з Redis. Розсилає команди (setLight, setSchedule) підключеним клієнтам при змінах через REST API.

```
    wss.on('connection', (ws) => {
       ws.on('message', async (message) => {
           const data = JSON.parse(message);
           if (data.command === 'getState') {
               const deviceState = await
redisClient.hGet('devices', data.sensorId);
               ws.send(JSON.stringify({ sensorId:
data.sensorId, ...JSON.parse(deviceState || '{}') }));
           }
       });
    });
```

5 ТЕСТУВАННЯ ПРИСТРОЮ

Процес тестування пристрою проходить у декілька етапів. Першим етапом тестування була перевірка підключення головного базового модуля до мережі Wi-Fi. Для цього в на самому пристрої повинен загорітися індикаторний світлодіод, а у терміналі, що приєднаний на етапі розробки до комп'ютера повинна з'явитися строчка про успішний коннект. Процес під'єднання показано на рисунку 5.1.

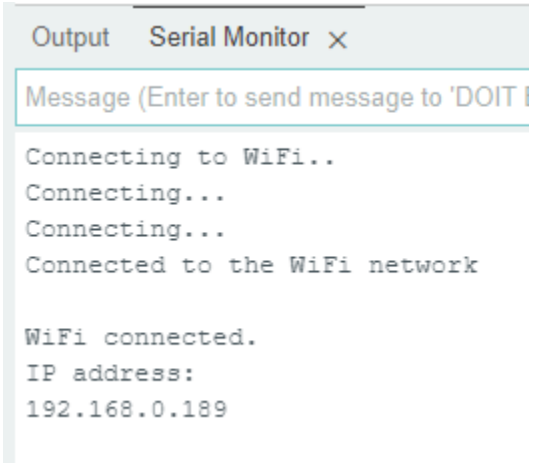

Рисунок 5.1 – Процес під'єднання

Для достовірної перевірки що базовий модуль приєднано до хмарного сервісу потрібно перевірити логи на сервері у хмарному сервісі. Для цього потрібно приєднатися до консолі AWS та зайти на віртуальний сервер, що був там розгорнутий. На рисунку 5.2 можна побачити панель керування хмарним сервісом.

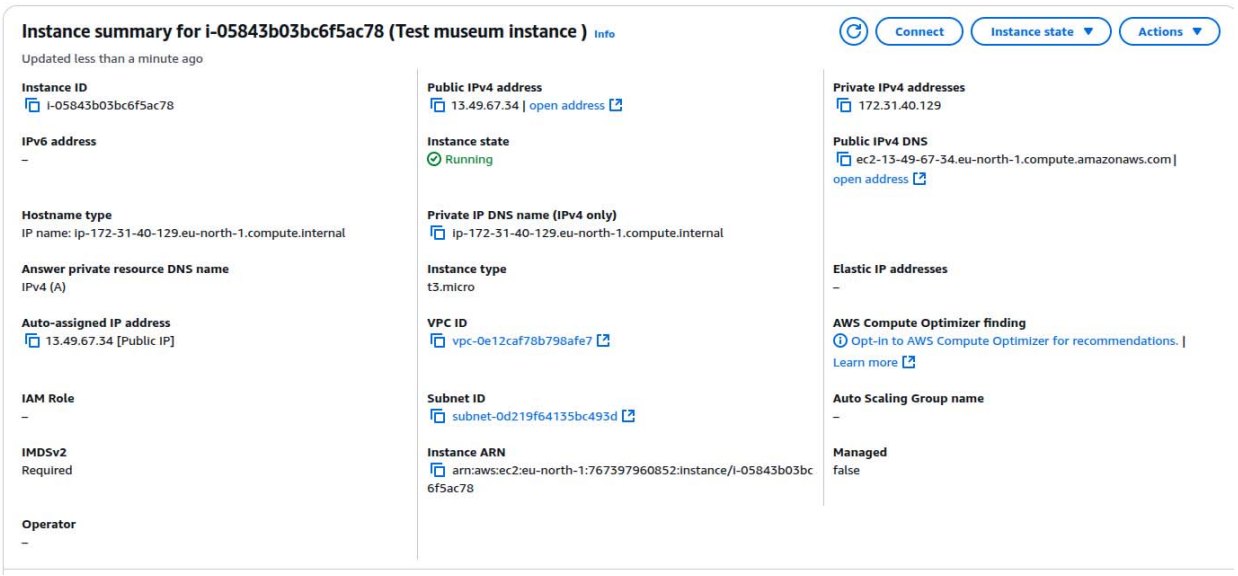

Рисунок 5.2 – Панель керування

Після цього необхідно зайти на веб інтерфейс додатку та налаштувати розклад переключень світла у денний та ночний режими за часом. Також там є можливість увімкнути світло дистанційно за допомогою кнопок керування. Інтерфейс додатку можна побаціти на рисунку 5.3

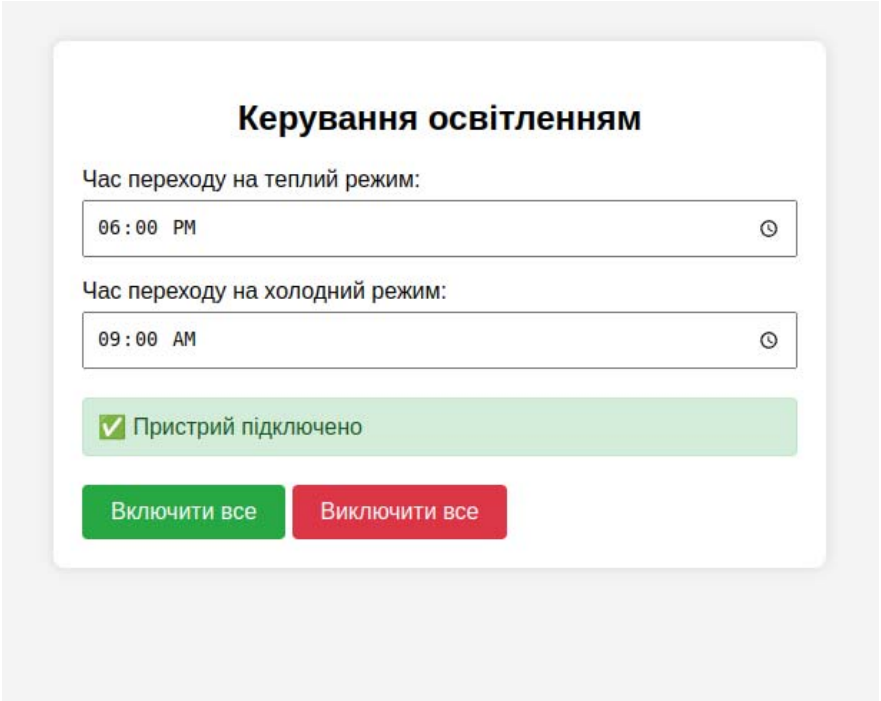

Рисунок 5.3 – Інтерфейс додатку керування світлом

Фінальний етап перевірки, проводимо рукою поміж сенсору руху і дивимось чи призведе це до вмикання світла. На рисунку нижче наведено фотографію виконуючого вузлового пристрою що має сенсор руху, радіомодуль та коннектор для приєднання до лампи освітлення. Для безпеки вони обгорнуті у ізоляційну термоусадку.

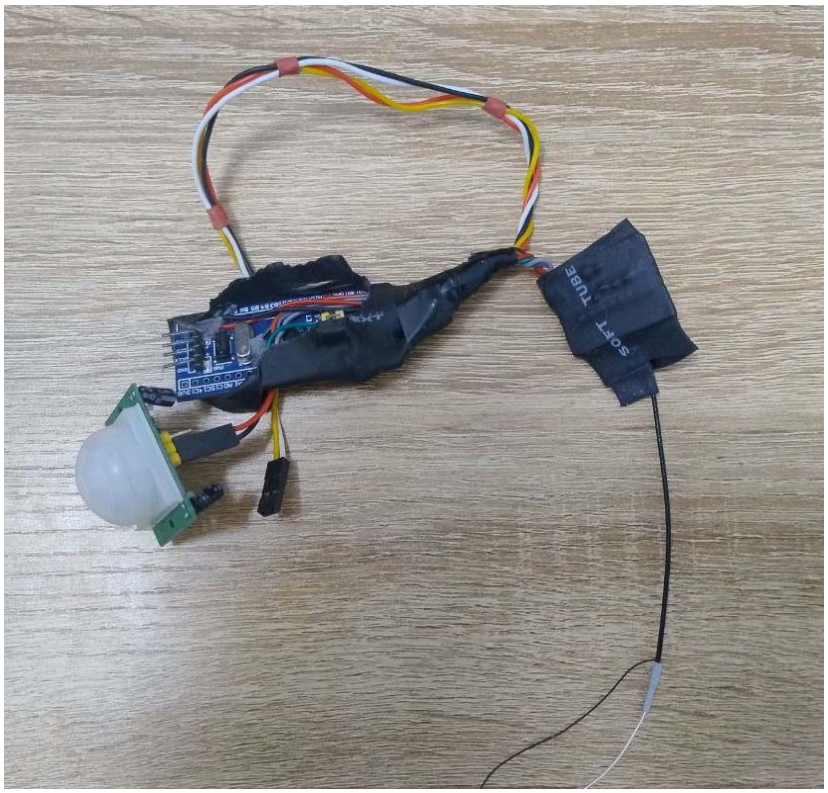

Рисунок 5.4 – Вузловий пристрій

Головний модуль можна побачити на рисунку 5.5, він складається з радіо модуля LoRa, ESP32 Wroom та допоміжного блоку для імітації другого вузлового пристрою.

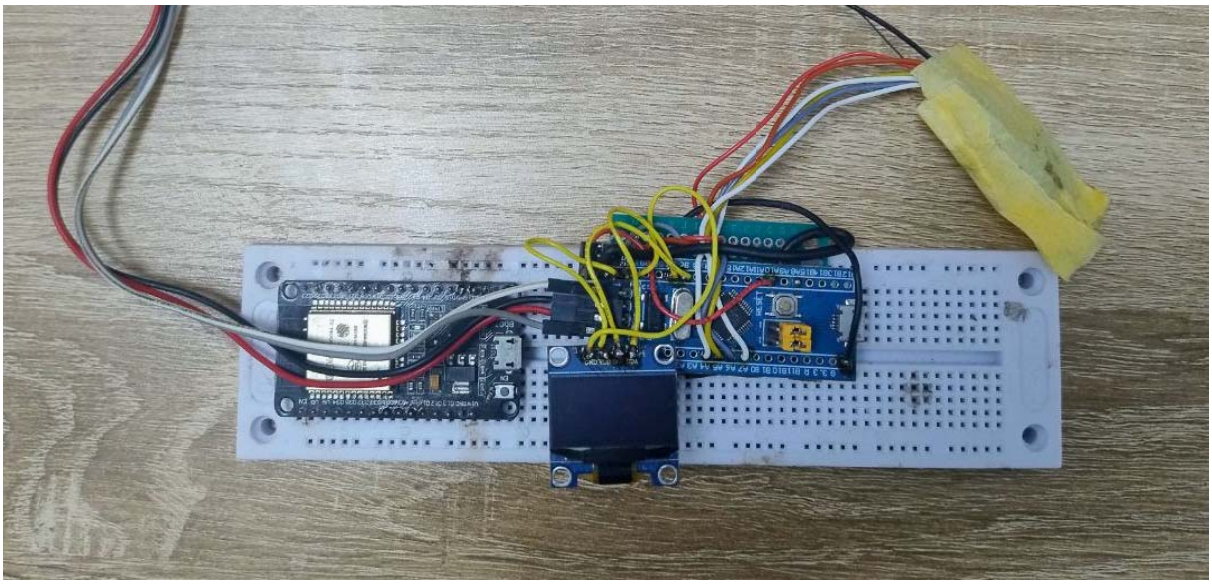

Рисунок 5.5 – Головний модуль

Після проведення рукою світло вмикається, та відповідає тому режиму що ми зазначили у налаштуваннях, також за таймером воно змінює тон. При натисканні на панелі керування веб інтерфейсі кнопок керування світлом, воно дистанційно вмикати та вимикати. Це свідчить про те що всі поставлені завдання роботи виконані і працюють коректно.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено програмно-апаратний комплекс для системи розумного освітлення на основі технологій Інтернету речей (IoT), який забезпечує оптимізацію біоритмів людини в домашніх умовах. Основні досягнення та висновки роботи такі:

Було створено функціонуючий комплекс, що включає мікроконтролери ESP32 і STM32F100, датчики освітленості, руху та колірної температури, а також виконавчі пристрої для керування світлодіодним освітленням. Система використовує мережу LoRa для надійної передачі даних на великі відстані з низьким енергоспоживанням, що робить її ефективною для використання в домашніх умовах.

Розроблені алгоритми автоматичного вибору відтінку світла (теплого, 2700–3500 K, для релаксації, та холодного, 5000–6500 K, для бадьорості) залежно від часу доби та активності людини. Це дозволяє підтримувати циркадні ритми, знижувати рівень стресу та покращувати якість сну і концентрацію.

Реалізовано веб-додаток на базі Amazon Web Services (AWS), який забезпечує моніторинг даних у реальному часі, логування (з використанням Amazon S3), аналіз (через AWS Lambda) та зберігання структурованих даних (у Amazon DynamoDB). Додаток надає зручний інтерфейс для керування системою та аналізу її роботи.

Також в роботі завдяки автоматизації на основі датчиків руху та освітленості та використаню енергоефективних протоколів LoRa, дало можливість системі знизити енергоспоживання щонайменше на 30% порівняно з традиційними системами освітлення.

Забезпечено захист даних у мережі LoRa та хмарній інфраструктурі AWS за допомогою шифрування та управління доступом (AWS IAM,

Cognito). Архітектура системи дозволяє легко масштабувати її шляхом додавання нових датчиків і вузлів мережі.

Проведений аналіз сучасних рішень у сфері розумного освітлення показав, що більшість комерційних систем мають обмеження щодо вартості, масштабування або безпеки. Запропонована система є конкурентоспроможною завдяки використанню доступних компонентів, відкритих платформ і технології LoRa, що робить її привабливою для побутового використання.

Розроблена система може бути використана в домашніх умовах для створення комфортного та здорового середовища, а також адаптована для комерційних або громадських приміщень. Вона сприяє популяризації IoT-технологій, підвищенню енергоефективності та покращенню якості життя користувачів.

Таким чином, розроблений програмно-апаратний комплекс є ефективним рішенням для автоматизації освітлення з урахуванням біологічних потреб людини. Робота підтверджує актуальність і перспективність використання IoT-технологій у створенні розумних систем, які поєднують комфорт, енергоефективність і турботу про здоров’я.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tanenbaum A. S., Wetherall D. J. Computer Networks. – 5th ed. – Pearson, 2021. – 960 p.
2. Philips Hue. Official Documentation and API Reference [Електронний ресурс]. – Режим доступу: https://www.philips-hue.com/en-us/support/developers. – Дата доступу: 15.04.2025.
3. LoRa Alliance. LoRaWAN Specification v1.1 [Електронний ресурс]. – Режим доступу: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/. – Дата доступу: 10.04.2025.
4. Amazon Web Services. AWS IoT Core Developer Guide [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html. – Дата доступу: 12.04.2025.
5. Rea M. S., Figueiro M. G. Light and Human Health: An Overview of Circadian Lighting Design. – Lighting Research & Technology, 2022. – Vol. 54, Issue 3. – P. 245–262.
6. Mordor Intelligence. Internet of Things (IoT) Market – Growth, Trends, and Forecasts (2024–2029) [Електронний ресурс]. – Режим доступу: https://www.mordorintelligence.com/industry-reports/internet-of-things-iot-market. – Дата доступу: 08.04.2025.
7. ESP32 Technical Reference Manual [Електронний ресурс]. – Espressif Systems, 2024. – Режим доступу: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. – Дата доступу: 10.04.2025.
8. STMicroelectronics. STM32F100 Datasheet [Електронний ресурс]. – Режим доступу: https://www.st.com/resource/en/datasheet/stm32f100.pdf. – Дата доступу: 12.04.2025.
9. Statista. Smart Home Market – Global Revenue Forecast 2020–2025 [Електронний ресурс]. – Режим доступу:

https://www.statista.com/statistics/1234567/smart-home-market-revenue-worldwide/. – Дата доступу: 14.04.2025.

10. Semtech Corporation. LoRa and LoRaWAN: A Technical Overview [Електронний ресурс]. – Режим доступу: https://www.semtech.com/lora/lora-overview. – Дата доступу: 11.04.2025.
11. Boyes H., Hallaq B. Cybersecurity for IoT Devices. – IEEE Internet of Things Journal, 2023. – Vol. 10, Issue 5. – P. 1234–1245.
12. Analysis of the implementation efficiency of digital signal processing systems on the technological platform SoC Zynq 7000 / Olexander Shkil, Oleh Filippenko, Dariia Rakhlis, Inna Filippenko, Valentyn Kornienko // Radioelectronic and Computer Systems. – 2024. – No. 4 (112). – P. 168-177.
13. http://colinord.blogspot.com/2015/01/arduino-oled-module-with-3d-demo.html/ 11.05.2025 – Загол. з екрану
14. Evaluation of the performance of a computing cluster based on single board raspberry pi 3b+ computer / O. Barkovska, V Korniienko, I Filippenko et al. // 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 02-06 October, 2023: proceedings. – P. 1–6.