Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО
"_____" _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний засіб для збирання та зберігання даних з розподілених сенсорних модулів

Виконавець: Микола ВОЙТЕХ

Керівник: Ольга КРАВЧЕНКО

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій

Кафедра комп’ютеризованих систем правління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

«_____»_____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Войтеха Миколи Юрійовича

1. Тема кваліфікаційної роботи «Програмний засіб для збирання та зберігання даних з розподілених сенсорних модулів»

затверджена наказом ректора від «28» 08 2023 р. №1494/ ст

2. Термін виконання роботи (проєкту): з 02.10.2023 по 31.12.2023

3. Вихідні дані до роботи (проєкту): мікроконтролер *ESP32*, сенсорні модулі *HDC1080*, *CCS811*, *SI1145*, мови програмування *C/C++*, бібліотеки *Qt* та *ESP-IDF*

4. Зміст пояснювальної записки: вступ, огляд існуючих засобів, вибір засобів розробки, розробка програмного засобу для збирання та зберігання даних з розподілених сенсорних модулів, висновки

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Схематичне зображення мережі розподілених сенсорних модулів

2) Вибір кореневого вузла

3) Процес підключення пристроїв до мережі

4) Процес відновлення мережі при втраті пристрою

5) Діаграма послідовності обміну пакетами між сервером та вузлом

6) Фото зібраних налагоджувальних плат(пристроїв)

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Отримати тему роботи	02.10.2023	
2	Зібрати і ознайомитися з інформацією згідно з темою роботи	03.10.2023 –06.10.2023	
3	Обрати середовище розробки для виконання поставленої задачі	09.10.2023 –11.10.2023	
4	Ознайомитися з принципами побудови розподілених мереж	12.10.2023 –17.10.2023	
5	Сконструювати та зібрати електронні модулі розподіленої сенсорної мереж	18.10.2023 –23.10.2023	
6	Розробити необхідні компоненти програмного продукту	24.10.2023 –17.11.2023	
7	Написати функціональну частину програми	18.11.2023 –01.12.2023	
8	Виконати тестування розробленого програмного продукту	04.12.2023 –08.12.2023	
9	Оформити пояснювальну записку	11.12.2023 –15.12.2023	

7. Дата видачі завдання: «02» 10. 2023 р.

Керівник кваліфікаційної роботи _____ Ольга КРАВЧЕНКО

Завдання прийняв до виконання _____ Микола ВОЙТЕХ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний засіб для збирання та зберігання даних з розподілених сенсорних модулів.»: 114 с., 65 рис., 1 таблиця, 38 літературних джерел, 1 додаток.

Ключові слова: РОЗПОДІЛЕНА СЕНСОРНА СИСТЕМА, СЕНСОРНИЙ МОДУЛЬ*, ESP32*, *ESP-MESH*, *WIFI*.

Об’єкт дослідження – процес збирання та зберігання даних із розподілених сенсорних модулів.

Предмет дослідження – програмний засіб для збирання та зберігання даних із розподілених сенсорних модулів.

Мета кваліфікаційної роботи – розробка програмного засобу для збирання та зберігання даних із розподілених сенсорних модулів.

Методи дослідження: аналіз існуючих програмних засобів для збирання та зберігання даних із розподілених сенсорних модулів, розробка та відлагодження системи на реальних пристроях.

Технічні та програмні засоби: мікроконтролери *ESP32*; датчики *HDC1080*, *CCS811* та *SI1145*; мови програмування *C/C++,* фреймворки *Qt* та *ESP-IDF*.

Отримані результати: програмний засіб для керування розподіленою сенсорною системою, отримані дані та графіки із сенсорних модулів.

Наукова новизна: застосування технології *ESP-WIFI-MESH* для створення розподіленої сенсорної системи, створення механізму та засобів для легкого додавання нових сенсорних модулів до пристрою, створення розподіленої сенсорної системи з простим додаванням нових пристроїв(вузлів), створення віртуальних сенсорних модулів, що дозволяють виконувати додатковий аналіз та обчислення даних на пристрої.

Результати кваліфікаційної роботи рекомендується використовувати в реальних проектах, що потребують збирання та зберігання даних з сенсорних модулів.

ЗМІСТ

ВСТУП

Актуальність теми. Актуальність обраної теми, а саме "Програмний засіб для збирання та зберігання даних з розподілених сенсорних модулів", визначається комплексом факторів, які виокремлюють її як науково та технічно важливу проблему. Ці фактори, що визначають ключові аспекти обраної теми дослідження для магістерської роботи, включають:

- Зростаюча важливість сенсорних технологій. Зараз наш світ стає все більше залежним від сенсорних модулів, які забезпечують надходження важливих даних. Ця залежність охоплює не лише науково-дослідний сектор, а й промисловість, медицину, сільське господарство, транспорт і інші галузі.
- Потреба у розподіленому зборі даних. Оскільки сенсорні модулі можуть бути розташовані в різних місцях, їх взаємодія та об'єднання в одну систему вимагає спеціалізованого програмного інструменту для збору та аналізу даних.
- Важливість зберігання та аналізу даних. Зібрані сенсорні дані мають значний потенціал для отримання цінної інформації та виявлення паттернів. Відтак, існує потреба у створенні програмного засобу, який забезпечує надійне зберігання та аналіз цих важливих даних.
- Практичний внесок. Результатом вивчення цієї теми є розробка функціонального програмного засобу, який може здійснювати корисні функції для різних галузей або вирішувати конкретні завдання, що може визначити новий етап впровадження та розвитку технологій.
- Спрощення процесу збирання даних. Розробка системи для збору даних з розподілених сенсорних модулів прискорює процес розробки програмного продукту, що призводить до економії ресурсів і часу розробників, сприяючи впровадженню системи збору даних.

Отже, обрана тема визначається необхідністю вирішення актуальних завдань, пов'язаних із зростаючою важливістю сенсорних технологій та необхідністю у

розподіленому зборі та аналізі даних з метою розробки практично застосовних програмних засобів.

Головною метою даної кваліфікаційної роботи є розробка програмного засобу, спрямованого на збір та зберігання даних з розподіленої сенсорної системи. Призначення цього програмного продукту полягає в тому, щоб спростити процес створення таких сенсорних систем, а також надати користувачам зручні інструменти для роботи з ними.

Один із визначальних викликів у розробці розподілених сенсорних систем - це інтеграція нових пристроїв. Цей процес вимагає значних зусиль і часу, оскільки потрібно забезпечити взаємодію між різними пристроями кожного разу при додаванні нових елементів у систему.

План роботи передбачає ретельне розглядання різних аспектів, починаючи від теоретичних основ і закінчуючи конкретною реалізацією програмного засобу. Особливий акцент буде зроблено на визначенні структури розподіленої системи, оскільки це визначальний фактор для забезпечення її надійності та легкості управління. Структура системи гратиме ключову роль у забезпеченні працездатності та сприятиме зручній взаємодії з нею.

Об’єкт дослідження – процес збирання та зберігання даних із розподілених сенсорних модулів.

Предмет дослідження – програмний засіб для збирання та зберігання даних із розподілених сенсорних модулів.

Мета кваліфікаційної роботи – розробка програмного засобу для збирання та зберігання даних із розподілених сенсорних модулів.

З метою досягнення визначеної цілі, передбачається вирішення низки завдань, спрямованих на дослідження та оптимізацію функціонування розподілених сенсорних систем та процесу збирання та зберігання даних у них. Необхідно вирішити наступні задачі:

- Провести детальний аналіз теоретичних аспектів розподілених сенсорних систем, а також процесу збирання та зберігання даних в таких системах.

Визначити основні тенденції та визначити ключові питання, пов'язані з цими аспектами.

- Провести аналіз і вибір інструментів для реалізації визначених завдань, зокрема для збору та зберігання даних у розподілених сенсорних системах.
- Створити структуру розподіленої сенсорної системи, враховуючи взаємодію між її складовими частинами та забезпечуючи оптимальну функціональність.
- Розробити структуру програмного засобу для вузла системи, забезпечуючи збір та передачу даних.
- Розробити структуру програмного засобу для центрального блоку системи, що керує та координує роботу вузлів.
- Визначити послідовність дій у комунікації між центральним блоком та вузлами системи для забезпечення обміну інформацією.
- Використовуючи імперичний метод, отримати та оцінити сенсорні дані з розподіленої сенсорної системи, враховуючи практичні виміри та характеристики системи у реальних умовах.

Наукова новизна отриманих результатів – визначено структуру розподіленої сенсорної системи для збирання та зберігання даних, що базується на протоколі *ESP-WIFI-MESH*. Цей підхід вирізняється від існуючих методів збору інформації із розподілених сенсорних модулів тим, що надає можливість динамічної побудови та самовідновлення мережі у випадку виникнення проблем у одному з вузлів системи.

У порівнянні з існуючими підходами, дана структура забезпечує унікальну можливість вузлам системи змінювати своє місцеположення у просторі без втрати зв’язку з центральним модулем. Це розширює можливості системи, дозволяючи адаптуватися до змін у середовищі та оптимізувати роботу сенсорної системи в реальному часі. Такий інноваційний підхід сприяє вдосконаленню надійності та відмовостійкості розподіленої сенсорної системи, роблячи його значущим внеском у сучасні технології збирання та зберігання даних.

Практичне значення отриманих результатів – отримані результати мають значуще практичне застосування та можуть знайти широке впровадження в

різноманітних сферах. Від простих метеостанцій до збирання даних на заводах та фабриках, в медичній сфері та агрономії, ці результати відкривають можливості для покращення та оптимізації різноманітних систем.

Завдяки розробленому програмному засобу, час, необхідний для створення подібних систем, значно зменшиться, що сприятиме простішому впровадженню технологій у відповідних галузях. Крім того, використання цього програмного інструменту дозволить економити фінансові ресурси, що є важливим фактором в сучасних умовах. Таким чином, практичне застосування отриманих результатів розширює можливості в сфері розробки та впровадження систем збирання та зберігання даних, забезпечуючи позитивний вплив на широкий спектр відомостей та галузей використання.

Особистий внесок випускника. Всі результати, представлені у кваліфікаційній роботі, отримані випускником особисто.

Апробація отриманих результатів. Практичні та теоретичні аспекти, результат яких отримано в кваліфікаційній роботі, проходили апробацію на міжнародній науково-технічній конференції «Інтелектуальні технології лінгвістичного аналізу», науковому круглому столі «Розвиток інформаційних технологій авіаційної галузі», науково-технічній конференції «Сучасні тенденції розвитку системного програмування».

Публікації. Войтех М.Ю., Кравченко О.П. Програмний засіб для збирання та зберігання даних з розподілених сенсорних модулів: міжнар. науково-техн. конф. «Інтелектуальні технології лінгвістичного аналізу», 24-25 жовтня 2023р.:, С. 84.

У ході виконання кваліфікаційної роботи був проведений аналіз теоретичних аспектів, пов'язаних із розподіленими сенсорними системами. Виявлено та розглянуто ключові аспекти цієї тематики, визначено необхідний інструментарій для розробки програмного засобу. Окреслено структуру розподіленої сенсорної системи та розроблено архітектуру програмного засобу для сервера та вузла системи. Додатково визначено алгоритм обміну інформацією в рамках даної системи.

Окрім того, отримано емпіричні дані з сенсорної системи, використовуючи імперичний метод. Цей підхід дозволив отримати практичні результати та здійснити

експериментальне вивчення роботи системи в реальних умовах. Такий комплексний підхід до дослідження та розробки дозволяє отримати вичерпні дані та глибше зрозуміти функціонування розподілених сенсорних систем.

Розроблений програмний модуль відзначається надзвичайною легкістю впровадження. Для належної роботи системи потрібно лише визначити внутрішню реалізацію сенсорного модулю, тобто метод отримання даних з нього. Усі інші аспекти, включаючи збір, зберігання даних та управління пристроєм, автоматизовані та відбуваються автоматично. Цей простий у використанні модуль спрощує впровадження системи, дозволяючи зосередитися на налаштуванні сенсорного модулю та отриманні корисних даних, забезпечуючи високий рівень зручності та ефективності в процесі експлуатації.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ

1.1. Поняття «розподілена система»

У літературі можна знайти різні визначення розподілених систем, причому іноді вони не узгоджуються між собою.

Сьогодні всі типи обчислювальних завдань — від керування базами даних до відеоігор — використовують розподілені обчислення. Фактично, багато типів програмного забезпечення, як-от системи криптовалют, наукове моделювання, технології блокчейну та платформи штучного інтелекту, були б взагалі неможливі без цих платформ.

Розподілена система (*Distributed System*) – це набір незалежних комп'ютерів, що представляється їх користувачам єдиною об'єднаною системою [5].

Розподілена система (*Distributed System*) – це будь-яке середовище, в якому декілька комп'ютерів або пристроїв спільно виконують різноманітні завдання та функції, і всі вони з'єднані в мережу [6]. Компоненти в розподілених системах розподіляють навантаження, координуючи свою роботу для більш ефективного виконання завдань, ніж це можливо для одного окремого пристрою.

Розподілені сенсорні системи (*Distributed Sensor Systems*) – це комплексні системи, які складаються з великої кількості розташованих на великій площині або у великому просторі сенсорів і датчиків, які призначені для збору інформації з навколишнього середовища та передачі її для подальшої обробки і аналізу [7]. Основна ідея полягає в тому, щоб забезпечити можливість моніторингу або вимірювання фізичних параметрів в різних точках простору або на великих відстанях.

Серед основних характеристик розподілених систем варто зазначити наступні[9]:

1. Забезпечення з'єднання між користувачами та ресурсами[8]: Основною метою розподіленої системи є спрощення доступу користувачів до

віддалених ресурсів і надання їм можливості керувати доступом інших користувачів до цих ресурсів. Під поняттям "ресурси" можна розуміти практично будь-що, зазвичай це охоплює принтери, пристрої зберігання, дані, файли, веб-сторінки і мережі. Існує багато причин для спільного використання цих ресурсів, однією з таких причин є економічна вигода.

2. Неоднорідність[8]. Неоднорідність відноситься до здатності системи працювати з різними апаратними та програмними компонентами. Проміжне програмне забезпечення на рівні програмного забезпечення допомагає досягти неоднорідності. Метою проміжного програмного забезпечення є інтерпретація викликів програмування таким чином, щоб розподілена обробка була завершена.
3. Прозорість[8]: Важлива мета розподіленої системи — приховати той факт, що її процеси та ресурси фізично розподілені між кількома комп’ютерами. Розподілена система, яка здатна представити себе користувачам і програмам так, що це лише одна комп’ютерна система, називається прозорою. Тобто цей програмний засіб чи система є єдиним цілим, об’єкти системи мають доступ до всіх ресурсів системи. В такій системі відмінність між комп’ютерами та способи зв’язку між ними.
4. Відкритість[8]: Ще однією важливою метою розподілених систем є відкритість. Відкрита розподілена система — це система, яка пропонує послуги в стандартах, які описують синтаксис і семантику цих екземплярів служб; стандартні правила в комп’ютерних мережах контролюють формат, вміст і значення повідомлень, що надсилаються та отримуються. Такі правила оформляються протоколами. У розподілених системах служби зазвичай визначаються через інтерфейси, які часто називають мовами визначення інтерфейсу (*IDL*). Визначення інтерфейсу, написані на *IDL*, майже завжди охоплюють лише синтаксис служб. Вони точно вказують назви доступних функцій із типами параметрів, значеннями, що повертаються, можливими винятками, які можна викликати, тощо.

5. Масштабованість[9]: У розподілених системах спостерігається невизначений тренд у напрямку збільшення розміру систем. Це спостереження має важливі наслідки для проектування розподілених файлових систем. Алгоритми, які ефективно працюють у системах із 100 комп'ютерами, можуть працювати у системах із 1000 комп'ютерів, але вже не застосовні для систем із 10 000 комп'ютерів. Перш за все, централізований алгоритм погано масштабується. Якщо для відкриття файлу потрібно звертатися до одного централізованого сервера для фіксації факту відкриття файлу, то з часом сервер стане обмежувальним фактором, коли система росте.
6. Надійність[9]: Головною метою побудови розподілених систем було зробити їх більш надійними, ніж однопроцесорні системи. Ідея полягає в тому, що якщо якась машина виходить з ладу, інша машина звикає до цього. Іншими словами, теоретично надійність системи в цілому може бути логічним АБО надійності компонентів. Наприклад, з чотирма файловими серверами, кожен з яких має ймовірність роботи 0,95 у будь-який момент, ймовірність того, що всі чотири одночасно будуть недоступні, становить 0,000006, тому ймовірність того, що принаймні один буде доступним, становить 0,999994, далеко краще, ніж будь-який окремий сервер.
7. Відмовостійкість[10]. Розподілена система, швидше за все, буде схильна до системних збоїв. Це пов’язано з тим, що кілька комп’ютерів мають апаратне забезпечення різного віку. Здатність системи справлятися з цими збоями називається стійкістю до відмов. Відмовостійкість досягається за рахунок:
   – Відновлення: системи та процеси матимуть збережену резервну копію. Він бере на себе, коли система виходить з ладу.
   – Надмірність: Коли компонент діє передбачувано та контрольовано, це називається надлишковістю.
8. Продуктивність[10]: Побудова прозорої, гнучкої, надійної розподіленої системи марна, якщо вона повільна, як патока. Зокрема, застосування в

розподіленій системі не повинно погіршуватися краще, ніж запуск деякої програми на одному процесорі. Можна використовувати різні показники ефективності. Час відгуку один, але пропускна здатність, використання системи та кількість споживаної ємності мережі також. Крім того, результати будь-якого еталонного тесту часто сильно залежать від характеру еталонного тесту. Порівняльний тест передбачає велику кількість незалежних обчислень із сильною навантаженням на ЦП, які дають радикально інші результати, ніж порівняльний тест, який складається зі сканування одного великого файлу на той самий шаблон.

Згідно із характеристиками системи, можна зробити висновок, що розподілені системи повинні також відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика є прямим наслідком наявності незалежних комп'ютерів, але в той же час не указує, яким чином ці комп'ютери насправді об'єднуються в єдину систему. Розподілені системи зазвичай існують постійно, проте деякі їх частини можуть тимчасово виходити з ладу або вимикатися. Користувачі і додатки не повинні повідомлятися про те, що ці частини замінені або полагоджені або що додані нові частини для підтримки додаткових користувачів або додатків.

Для того, щоб підтримати представлення різних комп'ютерів і мереж у вигляді єдиної системи, організація розподілених систем часто включає додатковий рівень програмного забезпечення, що знаходиться між верхнім рівнем, на якому знаходяться користувачі і додатки, і нижнім рівнем, що складається з операційних систем, як показано на рис. 1.1. Відповідно, така розподілена система зазвичай називається системою проміжного рівня (*middleware*).

Рис. 1.1. Ієрархія розподіленої системи, як служби проміжного рівня

1.2. Типи розподілених систем

Розподілені системи можна поділити на декілька типів:

– Системи клієнт/сервер(рис. 1.2): це найпростіші форми серверів. Клієнт передає вхідні дані серверу, а сервер відповідає обробленими даними клієнта. Клієнт хоче виконати завдання на сервері, а сервер виділяє та виконує завдання та надсилає результат як відповідь. Ці типи серверів можна застосовувати до багатьох інших серверів [11].

– Однорангові системи(*Peer-to-Peer*): у цій системі кожен вузол виконує своє завдання у своїй локально виділеній пам’яті та ділиться даними через допоміжне середовище. Програми комп’ютерної мережі використовують однорангову систему для керування процесорами, які спілкуються один з одним, але підтримують незалежні бази пам’яті [11]. Схему такої системи зображено на рис. 1.3.

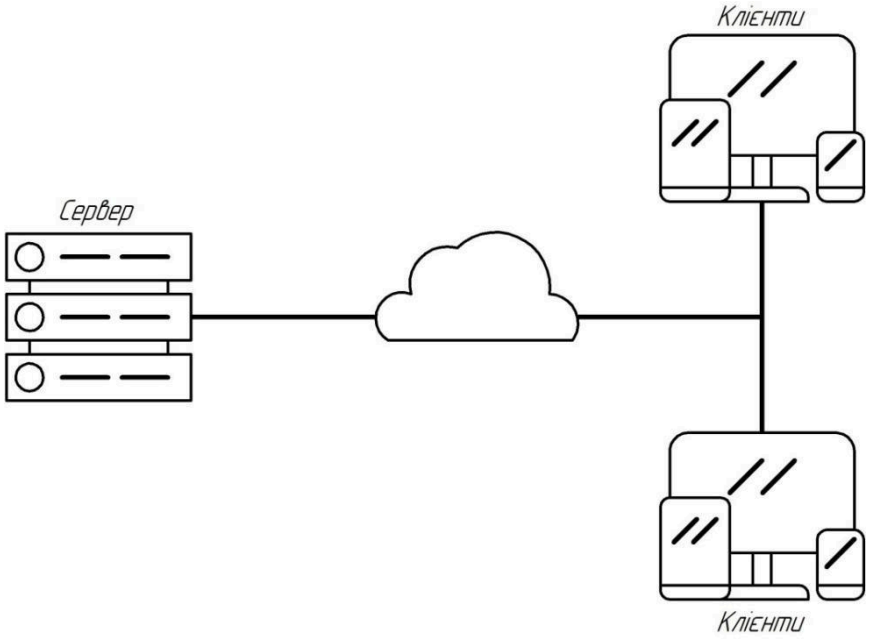


Рис. 1.2. Клієнт-серверна архітектура

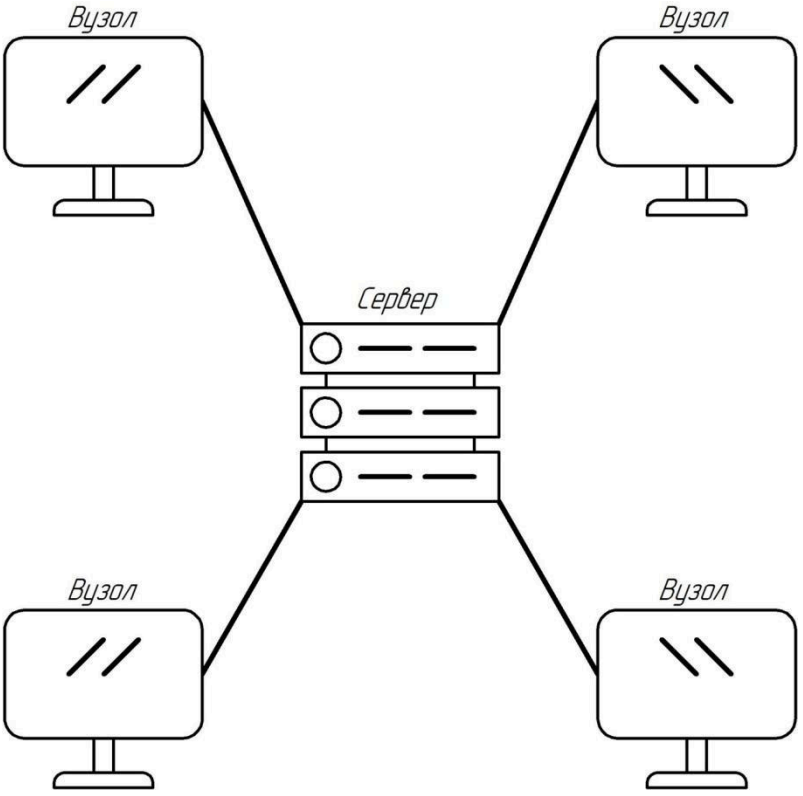


Рис. 1.3. *Peer-to-Peer* архітектура

– Проміжне програмне забезпечення(рис. 1.4): це програма, яка знаходиться між двома різними програмами та надає послуги та переваги для обох [11].

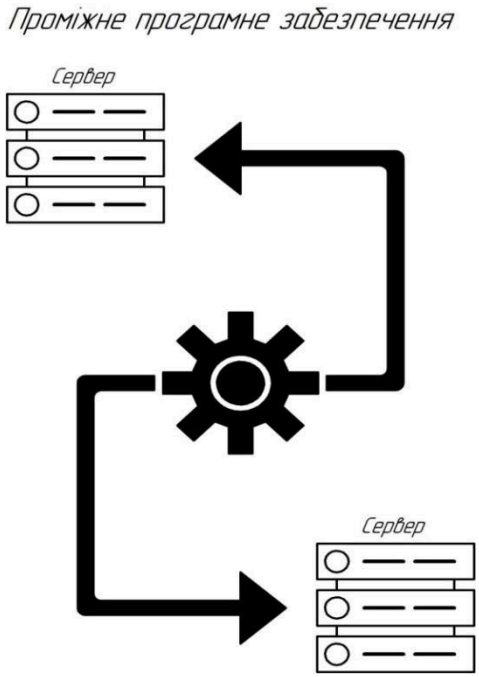


Рис. 1.4. Схематичне зображення проміжного програмного забезпечення

– Трирівнева(рис. 1.5): трирівнева система використовує окремий рівень і сервер для кожної функції програми. Дані клієнта зберігаються на середньому рівні. Він містить прикладний рівень, рівень даних і рівень презентації. Ця трирівнева система найчастіше використовується в веб-або онлайн-додатках [11].

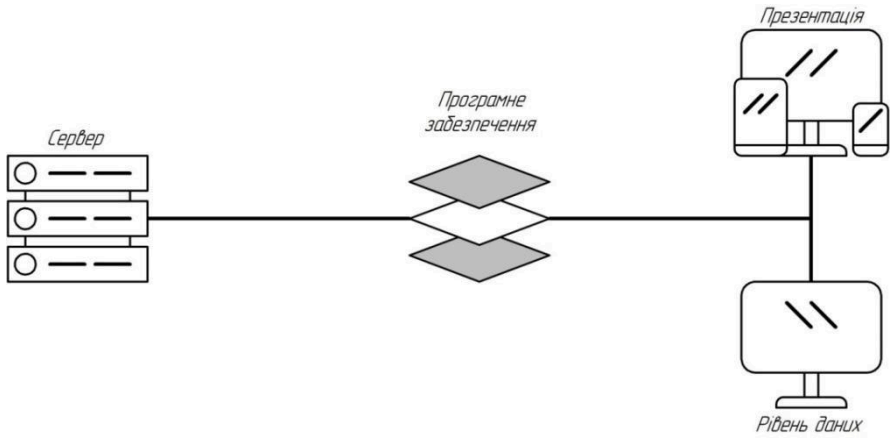


Рис. 1.5. Схематичне зображення трирівневої системи

– *N*-рівневий: також відомий як багаторівнева розподілена система. Як випливає з назви, ця система може містити будь-яку кількість функцій, подібних до трирівневої системи. Ця *N*-рівнева система частіше використовується у веб-додатках і системах даних [11].

За минулі роки було запропоновано безліч різних схем класифікації комп'ютерних систем з декількома процесорами, але жодна з них не стала дійсно популярною і широко поширеною. Нас цікавлять виключно системи, побудовані з набору незалежних комп'ютерів. Системи, в яких комп'ютери використовують пам'ять спільно, зазвичай називаються мультипроцесорами (*multiprocessors*), а що працюють кожен з своєю пам'яттю — мультикомп'ютерами (*multicomputer*). Основна різниця між ними полягає в тому, що мультипроцесори мають єдиний адресний простір, спільно використовуваний всіма процесорами.

1.3. Структури розподілених систем

У великій корпоративній мережі повністю централізована система управління, побудована на базі єдиного менеджера, навряд чи працюватиме добре з кількох причин[12]. По-перше, такий варіант не забезпечує необхідної масштабованості по продуктивності, оскільки єдиний менеджер змушений буде обробляти весь потік повідомлень від усіх агентів, що при кількох тисячах керованих об'єктів зажадає дуже високопродуктивної платформи для роботи менеджера і перевантажить службовою інформацією керуючої канали передачі даних в тій мережі, де буде розташований менеджер. По-друге, таке рішення не забезпечить необхідного рівня надійності, оскільки при відмові єдиного менеджера буде втрачено керування мережею. По-третє, у великій розподіленій мережі доцільно розташовувати у кожному географічному пункті окремим оператором чи адміністратором, керуючим своєю частиною мережі, але це зручніше реалізувати з допомогою окремих менеджерів кожному за оператора.

Схема «менеджер – агент» дозволяє будувати досить складні у структурному відношенні розподілені системи управління.

Зазвичай розподілена система управління включає велику кількість зв'язок менеджер – агент, які доповнюються робочими станціями операторів мережі, за допомогою яких вони отримують доступ до менеджерів (рис. 6).

Кожен агент збирає дані та керує певним елементом мережі. Менеджери, іноді також звані серверами системи управління, збирають дані від своїх агентів, узагальнюють їх та зберігають у базі даних. Оператори, що працюють за робочими станціями, можуть з'єднатися з будь-яким з менеджерів та за допомогою графічного інтерфейсу переглянути дані про керовану мережу, а також видати менеджеру деякі директиви з управління мережею або її елементами.

Наявність кількох менеджерів дозволяє розподілити з-поміж них навантаження з обробки даних управління, забезпечуючи масштабованість системи.

Як правило, зв'язки між агентами та менеджерами мають більш упорядкований характер, ніж той, який показаний на рис. 1.6.

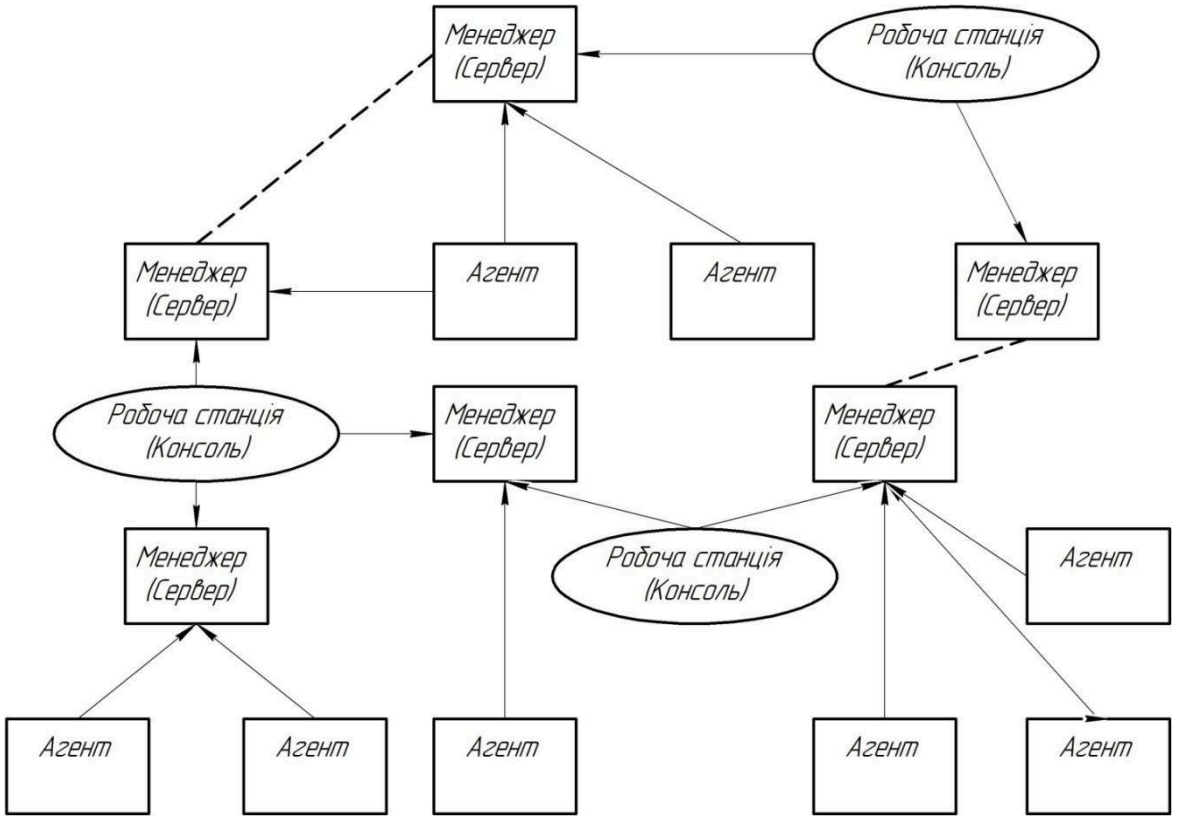


Рис. 1.6. Схематичне зображення розподіленої системи управління на основі кількох менеджерів та робочих станцій

Найчастіше використовуються два підходи до їх з'єднання – одноранговий (рис. 1.7) та ієрархічний (рис. 1.8).

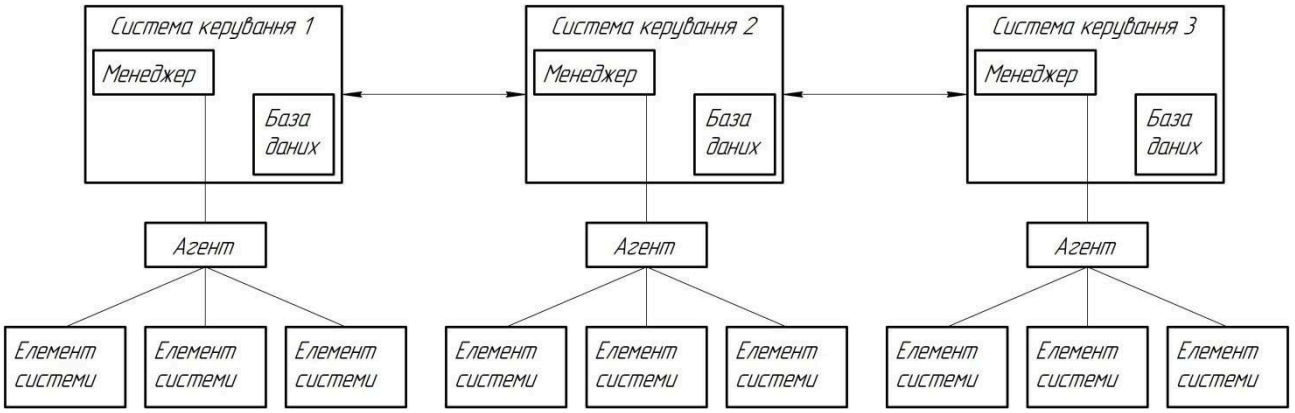


Рис. 1.7. Однорангові зв’язки між менеджерами

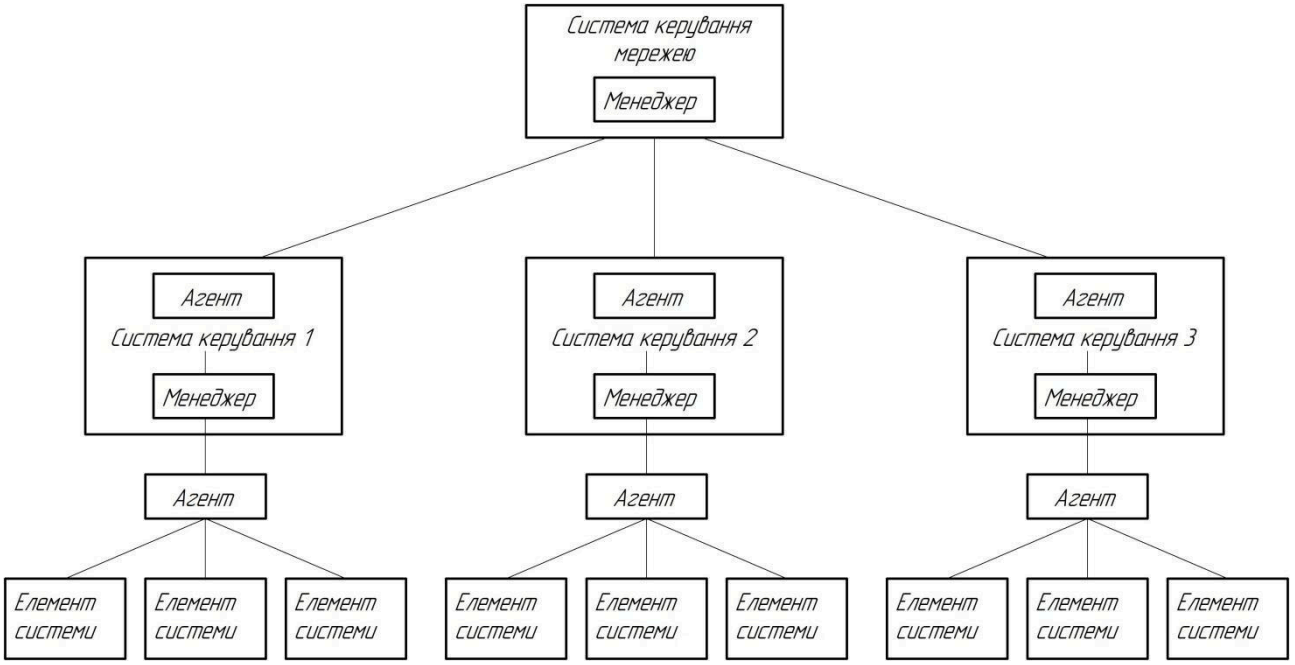


Рис. 1.8. Ієрархічні зв’язки між менеджерами

У разі однорангових зв'язків кожен менеджер керує своєю частиною мережі на основі інформації, що отримується від агентів, що знаходяться нижче. Центрального

менеджера немає. Координація роботи менеджерів досягається з допомогою обміну інформацією між базами даних кожного менеджера.

Однорангова побудова системи управління сьогодні вважається неефективною та застарілою. Зазвичай воно викликано тією обставиною, що елементарні системи управління побудовані як монолітні системи, які спочатку не були орієнтовані на модульність системи (наприклад, багато систем управління, розроблені виробниками обладнання, не підтримують стандартні інтерфейси для взаємодії з іншими системами управління). Потім ці менеджери нижнього рівня стали об'єднуватись для створення інтегрованої системи управління мережею, але зв'язки між ними виявилося можливим створювати лише на рівні обміну між базами даних, що досить повільно. Крім того, в базах даних таких менеджерів накопичується занадто детальна інформація про керовані елементи мережі (оскільки спочатку ці менеджери розроблялися як менеджери нижнього рівня), внаслідок чого така інформація є малопридатною для координації роботи всієї мережі в цілому. Такий підхід до побудови системи управління називається підходом знизу вгору.

Набагато гнучкішим є ієрархічна побудова зв'язків між менеджерами. Кожен менеджер нижнього рівня виконує функції агента для менеджера верхнього рівня. Такий агент працює вже з більш укрупненою моделлю (MIB) своєї частини мережі, в якій збирається саме та інформація, яка потрібна менеджеру верхнього рівня для управління мережею в цілому. Зазвичай розробки моделей мережі різних рівнях проектування починають з верхнього рівня, де визначається склад інформації, необхідної від менеджерів-агентів нижчого рівня, тому такий підхід названий підходом «згори донизу». Він скорочує обсяги інформації, що циркулює між рівнями системи управління, і призводить до більш ефективної системи управління.

1.4. Огляд існуючих розробок

Існують різні розподілені сенсорні системи, які сприймають інформацію з різних джерел та здатні обробляти дані. Декілька прикладів:

– Системи відеоспостереження

– *IoT*(інтернет речей)
– Системи моніторингу здоров’я
– Системи контролю за дорожнім рухом
– Системи моніторингу навколишнього здоров’я
– Системи віртуальної реальності(*VR*)

Існує більше напрямків в яких використовуються розподілені сенсорні системи. Проте розглянемо деякі з перелічених систем. Для обміну даними в таких системах можуть використовуватись різні типи комунікації.

Зокрема для систем відеоспостереження(рис. 1.9) та контролю за дорожнім рухом доцільно використовувати провідний спосіб комунікації або ж бездротовий із високою пропускною здатністю(наприклад *Wi-Fi*). Дані отримані з камер можуть записуватись в локальному сховищі чи оброблятись на локальних комп’ютерах або ж транслюватись в хмарне сховище [13].

Рис. 1.9. Системи відеоспостереження

В системах *IoT* як правило дані відправляються в інтернет-сервіс для моніторингу даних отриманих із датчиків(рис. 1.10). Ці дані потім можуть бути оброблені іншими додатками. Для таких мереж необхідний доступ до мережі інтернет. Ці пристрої отримують доступ до мережі інтернет із використанням протоколів бездротового зв'язку. Це пов'язано із зручністю використання, оскільки відсутні кабелі, які можуть заважати вдома та у таких пристроїв з'являється деяка мобільність. Доступ в інтернет може бути наданий для кожного пристрою(кожен пристрій має *SIM*-карту на борту) або ж отримувати доступ до інтернету через шлюз локальної мережі.

Рис. 1.10. Схематичне зображення обміну даними в системах *IoT*

В якості системи моніторингу здоров'я можна розглянути *Apple Watch*(рис. 1.11). Ці пристрої здатні відслідковувати різноманітні дані про здоров'я(наприклад пульс, рівень кисню в крові, тривалість та якість сну тощо). Ці дані збираються і можуть відображені на самому *Apple Watch* або ж вони передаються на телефон через *Bluetooth*. У таких системах відстань між елементами системи є дуже малою і технологія *Bluetooth* тут буде дуже доречна через низьке споживання електроенергії а відстані на якій працює *Bluetooth* є достатньою [14].

Системи віртуальної реальності за критеріями до вибору способу комунікації схожі на системи моніторингу здоров'я. У цих системах між елементами системи є невелика відстань, проте у таких системах виконується передача відеоданих, що вимагає високої пропускної здатності між пристроями. *Wi-Fi* може чудово

задовольнити такі потреби, проте досить часто використовуються і провідні з’єднання [15]. Приклад таких пристроїв показано на рис. 1.12.

Із інформації отриманої вище, для кожної розподіленої сенсорної системи використовуються різні способи комунікації. Проте фаворитом у таких системах є бездротові системи, провідні системи використовуються тільки у тому випадку, коли використання бездротових є неможливим, як правило через занадто довгий час відповіді чи малу пропускну здатність бездротового зв’язку.

Рис. 1.11. *Apple Watch*

Всі ці способи забезпечення комунікації описані вище є різними і специфічними для кожного пристрою, керувати такими пристроями є досить складним завданням. Для полегшення комунікації між пристроями та спрощення їх керування є доцільним створити спосіб для збирання та моніторингу розподіленої сенсорної системи на основі бездротових технологій. Додавання нових компонентів до системи має бути максимально простим, щоб розробники сенсорної системи

могли більше концентруватись на самих сенсорних пристроях, ніж на способі доставки цих даних до центрального комп’ютера.

Рис. 1.12. *VR*-шолом та контролери

1.5. Способи взаємодії між компонентами розподілених сенсорних систем

У розподілених сенсорних системах можна забезпечити взаємодію між їх складовими за допомогою провідних з'єднань (кабелі, шини даних, такі як *I2C* та *SPI*) і бездротових з'єднань (*Bluetooth, Wi-Fi*, мобільні мережі та інше).

Зв'язок між елементами системи може бути провідним (з використанням фізичних кабелів та провідників) або бездротовим (за допомогою радіохвиль або інших безпровідних технологій). Нижче надано короткий опис кожного типу з'єднання:

Провідні з'єднання:

– Кабелі - це з'єднувальні пристрої, які можуть включати в себе кілька провідників для передачі даних, енергії тощо. Наприклад, кабелі *USB* і *HDMI*.

– Шини даних - це спеціальні комунікаційні протоколи, які дозволяють різним пристроям на шині спілкуватися між собою. *I2C* і *SPI* - це приклади таких шин, які часто використовуються в електроніці.

Бездротові з'єднання:

– *Bluetooth* - це бездротовий протокол короткого діапазону, який використовується для підключення різних пристроїв, таких як гарнітури, клавіатури, миші і смартфони.
– *Wi-Fi* - це бездротовий стандарт для локальних мереж, який дозволяє підключати комп'ютери, смартфони і інші пристрої до Інтернету та локальних мереж через бездротовий сигнал.
– Мобільні мережі, такі як *3G*, *4G* і *5G*, надають з'єднання для мобільних пристроїв і дозволяють доступ до Інтернету і голосового зв'язку, навіть у руху.

Провідні з'єднання мають свої переваги і недоліки, які варто враховувати при їх використанні. Кабель чи провідник може бути вигідним в одних ситуаціях і менш доцільним в інших. Нижче розглянуто деякі з основних переваг і недоліків провідних з'єднань:

Переваги провідних з'єднань:

1. Стабільність сигналу. Провідні з'єднання зазвичай мають стабільну і надійну передачу даних. Вони менше вразливі до перешкод і інтерференції, які можуть виникнути в бездротових мережах.
2. Висока швидкість передачі даних. Провідні з'єднання дозволяють досягти високих швидкостей передачі даних, особливо в разі використання волоконно-оптичних кабелів.
3. Захист від зовнішніх вторгнень. Фізична природа провідних з'єднань може забезпечувати певний рівень захисту від несанкціонованого доступу, оскільки зловмисники повинні мати фізичний доступ до кабелів.
4. Стійкість до втрати сигналу. Провідні з'єднання не страждають від проблем, які властиві бездротовим з'єднанням, таким як втрата сигналу через велику відстань або перешкоди.

Недоліки провідних з'єднань:

1. Обмеженість в мобільності. Провідні з'єднання потребують фізичного підключення до мережі, що обмежує мобільність пристроїв.
2. Складність монтажу. Установка провідних з'єднань може бути важкою та вимагати значних зусиль, особливо в тих випадках, коли потрібно прокладати кабелі через важкодоступні місця або великі відстані.
3. Витрати на інфраструктуру. Введення інфраструктури для провідних з'єднань, такої як прокладка кабелів та розгалужувачів, може бути вартісним завданням.
4. Залежність від фізичного стану кабелів. Кабелі можуть вийти з ладу через фізичний знос, пошкодження або навіть вандалізм.
5. Пошук несправностей та ремонт. Пошкодження в кабелі важко знайти на досить довгій відстані, оскільки потрібно переглянути в найгіршому випадку весь кабель щоб знайти пошкодження. Ремонт у деяких випадках є складним або навіть не можливим, і тому можлива заміна всього кабелю.

Бездротові з'єднання мають свої переваги і недоліки. Нижче перераховані деякі з основних переваг і недоліків бездротових з'єднань:

Переваги бездротових з'єднань:

1. Мобільність. Бездротові з'єднання дозволяють пристроям бути мобільними і не обмежувати їх фізичним підключенням до мережі. Це особливо важливо для пристроїв, які використовуються в рухливих об'єктах або для мобільних пристроїв, таких як смартфони і планшети.
2. Зручність і простота встановлення. Встановлення бездротових з'єднань зазвичай менше часом і зусиллями, оскільки не потрібно прокладати фізичні кабелі або проводити складну інсталяцію.
3. Висока робоча відстань. Бездротові з'єднання можуть працювати на значних відстанях від базової станції або точки доступу, що робить їх ідеальними для зовнішніх застосувань або для підключення віддалених пристроїв.

4. Зручний доступ до Інтернету. Бездротові технології, такі як *Wi-Fi* і мобільні мережі, дозволяють забезпечувати зручний доступ до Інтернету в будь-якому місці з покриттям мережі.

Недоліки бездротових з'єднань:

1. Нестабільність сигналу: Бездротові з'єднання вразливі до перешкод, інтерференції та втрати сигналу через велику відстань або стіни. Це може вплинути на надійність з'єднання.
2. Обмежена швидкість передачі даних. У порівнянні з провідними з'єднаннями, бездротові мережі можуть мати обмежену пропускну здатність і швидкість передачі даних.
3. Безпека. Бездротові мережі можуть бути менш безпечними, оскільки сигнал може бути зловмисно перехоплений або схильний до атак. Вони потребують додаткових заходів забезпечення безпеки.
4. Залежність від живлення. Бездротові пристрої потребують живлення, і це може вимагати більшої кількості батарей або зарядних пристроїв.
5. Витрати на обладнання. Побудова і підтримка бездротових мереж може бути вартісною, зокрема при необхідності створення великих покриттів або вдосконалення існуючого обладнання.

Елементи розподіленої сенсорної системи можуть знаходитись на різній відстані один від одного, зокрема ці відстані можуть змінюватись або бути досить великими. Як правило такі системи отримують дані від датчиків, зберігають та надсилають їх до серверів чи інших комп’ютерів для подальшої обробки. Ці дані можуть мати досить малий обсяг, відповідно для таких систем може підійти будь який бездротовий спосіб передачі. Проте у випадку додавання якогось сенсора, що працює із аудіо, відео чи фото, то швидкість передачі даних може бути значно вищою ніж передача звичайних текстових даних.

Оцінивши недоліки та переваги кожного з способів взаємодії, було вирішено що найкращим для таких систем буде все ж технологія бездротового з’єднання. У літературі технології *Wi-Fi*, *Bluetooth* та *ZigBee* часто розглядаються як конкуруючі.

Насправді кожна зі згаданих технологій має свої унікальні характеристики, що зумовлюють їх оптимальні сфери застосування.

1.5.1. Технологія *Bluetooth*

Bluetooth - це бездротова технологія зв'язку, яка дозволяє пристроям обмінюватися даними на коротких відстанях через радіохвилі. Ця технологія була розроблена з метою спростити підключення і обмін інформацією між різними електронними пристроями, такими як смартфони, навушники, клавіатури, миші, акустичні системи та інші пристрої.

Основні характеристики *Bluetooth* [16]:

- Бездротова технологія. *Bluetooth* використовує радіохвилі для передачі даних, що дозволяє підключати пристрої без потреби в фізичних кабелях.
- Низька споживана потужність. *Bluetooth* розроблено з огляду на ефективне використання енергії, що дозволяє працювати пристроям на батарейках протягом тривалого часу.
- Малий радіус дії. Зазвичай *Bluetooth* здатний працювати на відстані близько 10-30 метрів.
- Можливість підключення багатьох пристроїв. *Bluetooth* дозволяє одночасно підключати багато пристроїв до одного джерела. Це дозволяє створювати мережі, в яких різні пристрої можуть взаємодіяти між собою.
- Зручність підключення. Процес підключення пристроїв через *Bluetooth* визнаний за легкий і зручний. Просто увімкніть *Bluetooth* на обох пристроях, встановіть їх у режим парування та виберіть необхідний пристрій для підключення.
- Постійний розвиток. *Bluetooth* постійно оновлюється і вдосконалюється. Нові версії додають нові функції і покращують якість підключення, безпеку та енергоефективність.

Ця технологія має досить багато версій [17]. Розглянемо версії 2 та 4.2.

Bluetooth 2.0:

- Випущено в 2004 році.
- Швидкість передачі даних складала 3 *Мбіт/с*, що було значним покращенням порівняно з попередніми версіями.

– Підтримував зв'язок між пристроями на відстані до 10 *м*.
– Вдосконалено систему автоматичного управління потужністю для ефективного використання енергії.

Bluetooth 4.2:

– Випущено в 2014 році.
– Значно покращено енергоефективність, що дозволило працювати пристроям на батарейках протягом довшого часу.
– Підтримує підключення до Інтернету через *Bluetooth*, що дозволяє прямо з пристроїв відправляти дані в інтернет і використовувати Інтернет-ресурси.
– Покращено безпеку з'єднань *Bluetooth*, зокрема, була введена технологія шифрування інформації.

Розглянемо готовий *Bluetooth* модуль *HC-06*. Цей модуль є досить поширеним і доступним в Україні. Він є сумісний із будь яким мікроконтролером. На основі цього модуля можна розглянути потенційні можливості *Bluetooth*.

Основні характеристики *HC-06*:

– Споживаний струм, до 20 *мА*.
– Живлення модуля: 5 *В*.
– Комунікація через *UART* порт на швидкості 9600 *бод*
– Версія *Bluetooth 2.0*.
– Дальність дії 10-30 *м*.

Оглянутий модуль має досить низькі показники споживаного струму та непогану швидкість обміну. Але низький радіус дії не дозволить працювати на великих відстанях або буде потрібно дуже багато «повторювачів», що призведе до збільшення часу комунікації та вартості системи. Також збільшення часу комунікації призведе до зменшення енергоефективності.

1.5.2. Технологія *ZigBee* та стандарт 802.15.4

Спочатку технологія *ZigBee* розроблялася як передачі невеликих обсягів телеметричної інформації на частоті 2,4 *МГц* з мінімально можливим

енергоспоживанням і невеликою швидкістю передачі і призначалася для переносних пристроїв з батарейним живленням [18].

Норвезька фірма *Radiocrafts AS* представляє на ринку пристроїв стандарту 802.15.4 повністю готові до роботи модулі. У цій лінійці виділяються дві нові серії "*RC2200 ZigBee ready*" (модулі з *DSSS*-маніпуляцією) і "*RC2200AT-SPPIO - ZigBee*" - модулі з підтримкою послідовного інтерфейсу та користувальницьких входів-виходів. Модулі містять прошивку стек протоколів *ZigBee*.

Сам стандарт *IEEE 802.15.4* визначає лише два нижні рівні (табл. 1.1): фізичний рівень (*PHY*) і рівень управління доступом до радіоканалу для трьох неліцензійних діапазонів частот: 2,4 *ГГц*, 868 і 915 *МГц* (*MAC-layer*).

Фізичний рівень протоколів 802.15.4 (*PHY*) визначає методи кодування-декодування та механізми, які використовуються для забезпечення необхідної швидкості передачі в залежності від середовища передачі даних. Іншими словами, на рівні *PHY* визначаються параметри приймачів.

Питання регулювання спільного використання середовища передачі даних визначаються на рівні *MAC* (*Media Access Control*—рівень доступу до середовища передачі даних). Саме на *MAC*-рівні встановлюються базові принципи взаємодії мережі із кількох *ZigBee*-пристроїв.

Безпосередньо у стандарті 802.15.4 обумовлюються діапазон частот, тип модуляції, структура пакетів, правила формування контрольної суми, способи запобігання колізії і т.д.

Найбільш важливі аспекти технології *ZigBee* запатентовано. Вони визначають такі конкретні параметри, як рівень мережі, протоколи безпеки, структури програми. Реально у цьому комплекті документації регламентуються алгоритми та детальне програмне забезпечення, що управляє роботою різних *ZigBee*-пристроїв та їх зв'язком з іншими зовнішніми пристроями.

Стандарт *IEEE 802.15.4* є відкритим та доступним для будь-якого користувача.

На жаль, цього не можна сказати про протоколи *ZigBee Alliance*. Вони закриті та доступні лише членам альянсу, які відраховують до нього щорічні внески. Пересічні користувачі можуть лише купувати необхідну їм інформацію.

Мережевий рівень *ZigBee* відповідає за конфігурацію мережі, також за виявлення пристроїв і протоколи їх взаємодії між собою. На мережному рівні в *ZigBee* підтримуються три варіанти топології мережі: "зірка", "кластерне дерево", "кожен з кожним" (*mesh*).

Таблиця. 1.1

Опис рівнів технології *ZigBee*

Рівень	Регламентуючий орган	Примітка
Application/Profiles	Користувач, Альянс *ZigBee*	Визначає інтерфейси зв'язку між різними пристроями і конкретні параметри користувача. Не має суворої регламентації
Application framework	Альянс *ZigBee*	Визначає конкретне програмне забезпечення, що керує роботою *ZigBee*-пристроїв та їх зв'язку з кінцевими пристроями(стек протоколів *ZigBee*). Закритий комплект документації та ПЗ
Network/Security layers		
MAC(Medium Access Control)	Стандарт *IEEE 802.15.4*	Рівні *PHY* та *MAC* визначені у відкритій документації, що визначає параметри радіо модулів, а також правила спільного використання середовища передачі даних одночасно декількома вузлами
PHY layer		

Слід особливо наголосити, що топологія mesh-мереж є однією з основних переваг *ZigBee* (рис. 1.13).

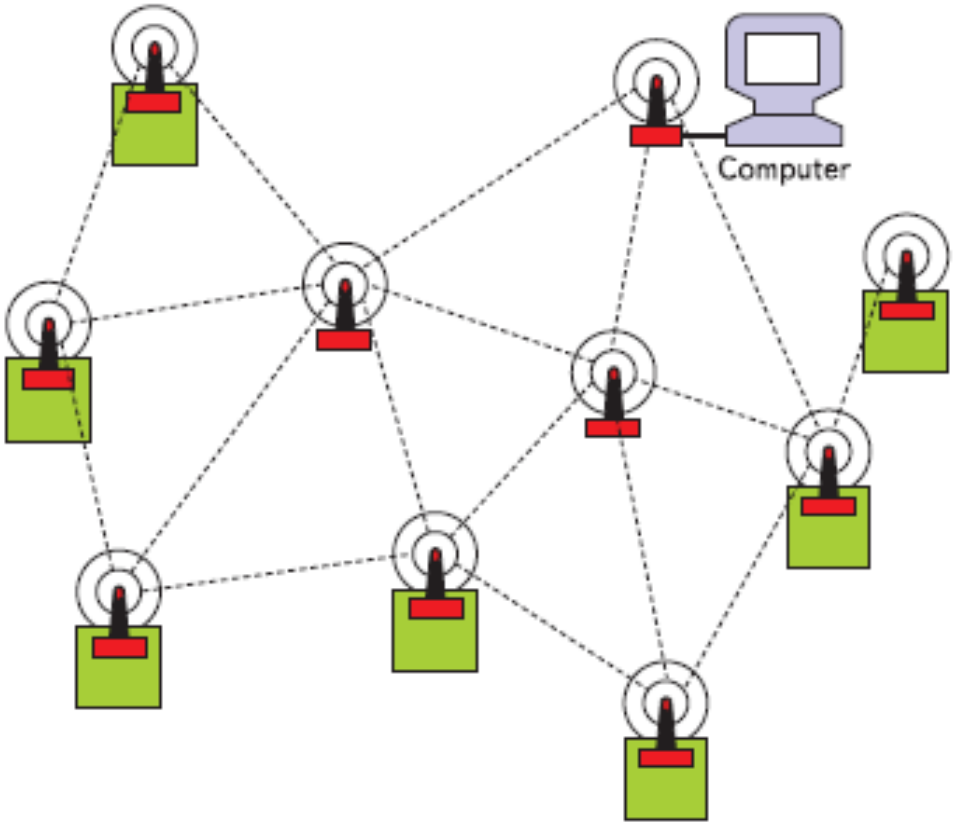


Рис. 1.13. Схематичне зображення *mesh*-мережі

У *mesh*-мережах окремі вузли кооперуються для того, щоб доставити повідомлення за призначенням. У разі несправності одного з вузлів дані надсилаються іншим маршрутом. Ця властивість є виключно важливою у мережах, що функціонують на промислових об'єктах у режимі жорстких умов експлуатації.

Профіль програм користувача підтримує конкретні параметри прикладних пристроїв користувача та вузлів мережі, а також їх індивідуальні адреси та характеристики. Цей рівень є відкритим у технології *ZigBee*. Тому користувач може використовувати як власний профіль, так і профілі, закріплені альянсом *ZigBee*.

У технології *ZigBee* використовуються два типи пристроїв різної складності.

Повністю функціональний пристрій (*FFD - Full Function Device*) здатний приймати та передавати дані, у тому числі і чужі, по ланцюжку. При об'єднанні *FFD*-пристроїв можуть бути реалізовані топології «зірка», «кожен з кожним» та «кластерне дерево».

Пристрій з обмеженим набором функцій (*RFD – Reduced Function Device*) – це найпростіший тип, який може лише переговорюватися з координуючим пристроєм. При об'єднанні в мережу *RFD* може використовуватися лише у топології «Зірка».

Окрім поділу на *FFD* та *RFD* у специфікації *ZigBee* визначено три типи логічних пристроїв – координатор мережі, маршрутизатор та кінцевий пристрій. Координатор ініціалізує мережу, керує мережними вузлами, зберігає інформацію про налаштування кожного вузла мережі, задає номер частотного каналу та ідентифікатор мережі *PAN ID*. Маршрутизатор відповідає за вибір шляху доставки повідомлення, що передається мережею від одного мережного вузла до іншого. Тип логічного пристрою під час побудови мережі визначає сам користувач на етапі вибору певного профілю та програмування.

Кожен вузол мережі має унікальну адресу. Залежно від виду адресації в мережі може бути визначена різна кількість логічних одержувачів та джерел інформації (кінцевих точок). У технології *ZigBee* передбачено використання кількох видів адресації, починаючи від стандартної 16-бітної та закінчуючи розширеною 64-бітною.

Протоколи *ZigBee* розроблені з урахуванням максимального енергозбереження. Якщо пристрій не задіяний, він переходить у режим очікування з мінімальною витратою енергоспоживання. При необхідності координатор активізує лише конкретний пристрій, який потрібний передачі інформації в даний момент. Швидкість передачі між пристроями залежить від кількості зайнятих каналів і у діапазоні від 20 до 256 *кбіт/с*.

Для побудови мереж *ZigBee* потрібно мати модулі, що включають трансівер, зовнішній мікроконтролер і стек протоколів *ZigBee* (набір керуючих програм). Як уже зазначалося вище, робота трансівера регламентована відкритим та повністю доступним міжнародним стандартом *IEEE 802.15.4*. Щодо стеку протоколів, то він доступний лише членам *ZigBee Alliance*. Тому для розробки мереж бездротового зв'язку необхідно купувати додаткове дороге програмне забезпечення та налагоджувальні комплекти.

Фірма *Radiocrafts* пропонує не витрачати гроші та час на програмне забезпечення та розробку, а скористатися її готовими рішеннями. Серія "*RC220x - ZigBee ready*" - це закінчені *ZigBee*-радіомодулі з *DSSS* (*Direct Sequence Spread Spectrum*, технологія розширення спектра методом прямої послідовності).

Модулі серії "*RC220x - ZigBee read*y" працюють на частоті 2,45 *ГГц* неліцензійного *ISM*-діапазону відповідно до стандарту *IEEE 802.15.4*. У базовій конфігурації *RC220x* мають прошивку за допомогою лише двох нижніх рівнів *PHY* і *MAC Chipcon firmware*. Модулі поставляються без прошивки верхніх рівнів стек протоколів *ZigBee*. У цій конфігурації модулі можна використовувати передачі даних лише у топології «точка — точка» і «зірка» через послідовний порт.

Модулі *RC2200* містять вбудований мікроконтролер *Atmega 128*. Користувачі можуть самостійно створювати додатки верхнього рівня та записувати їх поверх стека *ZigBee*-протоколів. Для цього достатньо використовувати відомі засоби програмування мікроконтролерів *Atmel: WinAVR/AVR GCC/Programmer's Notepad* та *Atmel AVR Studio*. Для завантаження програм у вбудований мікроконтролер знадобиться також *Atmel JTAG ICE mkll*.

Технічні характеристики модулів "*RC2200 - ZigBee ready*":

- кількість каналів 16 (2,45 *ГГц*, *ISM*);
- чутливість –94 *дБм*;
- швидкість передачі 250 *кбіт/с*;
- струм 30 *мА* у режимі *RX*;
- вихідна потужність до 0 *дБ/м* (27 *мА*);
- пам'ять: *Flash* - 32/64/128 *кбіт* (*RC2202/RC2204/RC2200*);
- 4 *кбіт EEPROM*, 4 *кбіт SRAM*;
- керування потоком *CTS/RTS*;
- інтерфейси *UART-1, UART-2, SPI, JTAG, boot-loader*;
- 32 цифрових та аналогових входу-виходу;
- 8-канальний 10-бітовий АЦП;
- 6 входів-виходів векторів переривання;
- годинник реального часу 32 *кГц* (*RTC*);

– антенний інтерфейс 50 *Ом*;
– дальність дії: 80 *м* на відкритому повітрі у зоні прямої видимості та 10–30 *м* у приміщеннях (залежно від матеріалів конструкції);
– габаритні розміри 16,5×29,2×3,5 *мм*;
– вбудована антена або роз'єм *MMCX* для зовнішньої антени;
– напруга живлення 2,7-3,6 *В*;
– *SMD*-корпус;
– відповідність вимогам *EN 300 400* (Європа), *CE з R&TTE*.

Оскільки *ZigBee-MAC*-рівень займає об'єм близько 16 кбіт, модуль *RC2202* (*Flash* 32 *кбіт*) може бути використаний тільки в якості *RFD* і кінцевого пристрою.

Модулі *RC2204* (*Flash* 64 *кбіт*) та *RC2200* (*Flash* 128 *кбіт*) можуть бути використані як *FFD* і запрограмовані для роботи як координатор або маршрутизатор.

Модулі *RC220x* дають можливість розробнику вбудовувати власне програмне забезпечення верхніх рівнів. Однак для цього потрібно додатково купувати програмне забезпечення для стека протоколу *ZigBee*. При цьому за допомогою модулів серії можна буде реалізувати мережі з більш складною топологією "кожний з кожним" та "кластерне дерево".

1.5.3. Технологія *Wi-Fi*

Wi-Fi (скорочення в англійській мові «*Wireless Fidelity*») - це технологія, яка дозволяє підключати пристрої до мережі без використання фізичних кабелів. *Wi-Fi* базується на стандартах бездротового зв'язку, розроблених Інститутом електротехніки та електроніки (*IEEE*), зокрема на стандартах 802.11 [19].

Основні характеристики технології *Wi-Fi* включають такі аспекти:

– Бездротовий зв'язок. *Wi-Fi* дозволяє підключати пристрої до мережі через радіохвильовий сигнал, що передається між бездротовим роутером (точкою доступу) і підключеними пристроями. Це вимагає бездротових адаптерів у пристроях, таких як смартфони, ноутбуки, планшети та інші.

– Спектр частот. *Wi-Fi* працює на різних частотах, включаючи 2,4 *ГГц* і 5 *ГГц*. Кожен діапазон має свої переваги і недоліки. Наприклад, 2,4 *ГГц* має більшу зону покриття, але меншу пропускну здатність, тоді як 5 *ГГц* надає вищу швидкість передачі даних, але обмежується коротшою дистанцією.
– Зони покриття. *Wi-Fi* мережа має обмежену зону покриття, зазвичай від кількох метрів до декількох сотень метрів в залежності від потужності роутера та перешкод на шляху сигналу.
– Захист. *Wi-Fi* мережі можуть бути захищені паролями та іншими методами шифрування, щоб уникнути несанкціонованого доступу до мережі.
– Стандарти. Існують різні покоління стандартів *Wi-Fi*, такі як *802.11b, 802.11g, 802.11n, 802.11ac* та *802.11ax* (відомий як *Wi-Fi 6*). Кожен новий стандарт надає покращену швидкість передачі даних та інші покращення у порівнянні з попередніми версіями.
– Висока швидкість передачі даних. *Wi-Fi* мережі можуть підтримувати різні швидкості передачі даних в залежності від стандарту та конкретного обладнання.

Основні покоління стандартів *Wi-Fi* та їх максимальні теоретичні швидкості включають:

– *802.11b*: Максимальна швидкість - 11 *Мбіт/с*.
– *802.11g*: Максимальна швидкість - 54 *Мбіт/с*.
– *802.11n*: Максимальна швидкість - до 600 *Мбіт/с* (з використанням *MIMO* - "*multiple-input and multiple-output*" технології).

Wi-Fi є дуже поширеною технологією, і вона використовується вдома, в офісах, у громадських місцях, у медичних закладах, в готелях і в інших сферах. Ця технологія значно полегшує доступ до Інтернету та обмін даними між різними пристроями без необхідності використання проводів.

Традиційна інфраструктурна мережа *Wi-Fi* – це мережа «точка-багато точок», де один центральний вузол, відомий як точка доступу (*AP*), безпосередньо підключений до всіх інших вузлів, відомих як станції. *AP* відповідає за арбітраж і пересилання передач між станціями. Деякі точки доступу також ретранслюють

передачу до/з зовнішньої *IP*-мережі через маршрутизатор. Традиційна інфраструктура мереж *Wi-Fi* має недолік обмеженої зони покриття через вимогу, щоб кожна станція була в зоні дії для прямого підключення до точки доступу. Крім того, традиційні мережі *Wi-Fi* чутливі до перевантаження, оскільки максимальна кількість станцій, дозволених у мережі, обмежена пропускною здатністю точки доступу (рис. 1.14). Таким чином ця мережа має топологію «зірка».

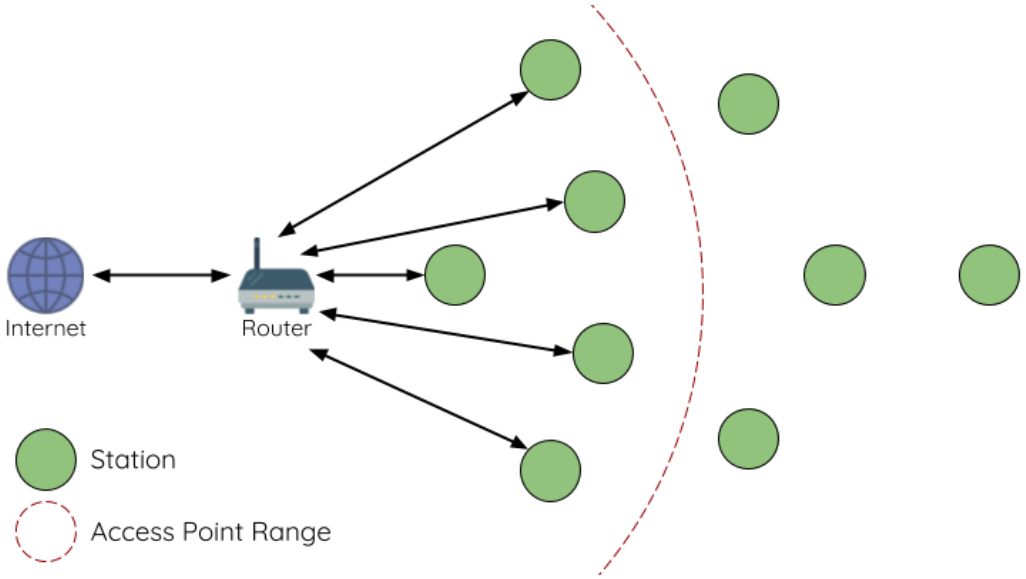


Рис. 1.14. Схематичне зображення мережі *Wi-Fi*

Проте, варто звернути увагу також на рішення від компанії *Espressif* – *ESP-WIFI-MESH*.

ESP-WIFI-MESH – це мережевий протокол, який дозволяє з'єднувати багато пристрої в, що знаходяться на великій відстані одне від одного.

ESP-WIFI-MESH відрізняється від традиційних інфраструктурних мереж *Wi-Fi* тим, що вузли не потребують підключення до центрального вузла. Натомість вузлам дозволено з'єднуватися з сусідніми вузлами(рис. 1.15). Вузли взаємно відповідають за ретрансляцію передач один одному. Це дозволяє мережі *ESP-WIFI-MESH* мати набагато більшу зону покриття, оскільки вузли все ще можуть досягати взаємозв'язку без необхідності перебувати в зоні дії центрального вузла. Подібним чином *ESP-WIFI-MESH* також менш сприйнятливий до перевантаження, оскільки

кількість вузлів, дозволених у мережі, більше не обмежується одним центральним вузлом.

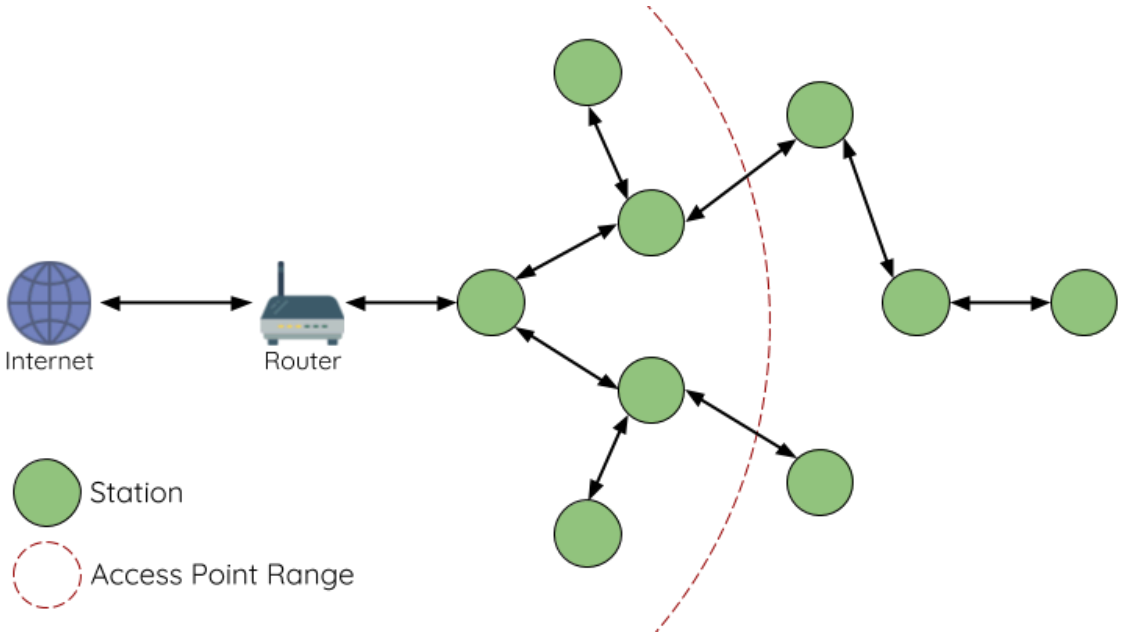


Рис. 1.15. Схематичне зображення мережі *ESP-WIFI-MESH*

Мережа *ESP-WiFi-Mesh* утворює ієрархічну структуру(рис. 1.16). *ESP-WIFI-MESH* побудовано на основі інфраструктурного протоколу *Wi-Fi*, і його можна розглядати як мережевий протокол, який об'єднує багато окремих мереж *Wi-Fi* в одну *WLAN*. У *Wi-Fi* станції обмежені одним з'єднанням із точкою доступу (висхідне з'єднання) у будь-який час, тоді як точка доступу може бути під'єднана одночасно до кількох станцій (низхідні з'єднання). Однак *ESP-WIFI-MESH* дозволяє вузлам одночасно діяти як станція та точка доступу. Тому вузол у *ESP-WIFI-MESH* може мати кілька низхідних з'єднань за допомогою свого інтерфейсу *softAP*, одночасно маючи одне висхідне з'єднання за допомогою свого інтерфейсу станції. Це природним чином призводить до деревовидної топології мережі з ієрархією «батько-нащадок», що складається з кількох рівнів.

ESP-WIFI-MESH — це мережа з декількома проміжками (*multi-hop*), що означає, що вузли можуть передавати пакети іншим вузлам у мережі через один або

кілька бездротових проміжків. Тому вузли в *ESP-WIFI-MESH* не тільки передають власні пакети, але одночасно служать ретрансляторами для інших вузлів. За умови, що існує шлях між будь-якими двома вузлами на фізичному рівні (через один або більше бездротових переходів), будь-яка пара вузлів у мережі *ESP-WIFI-MESH* може спілкуватися [20].

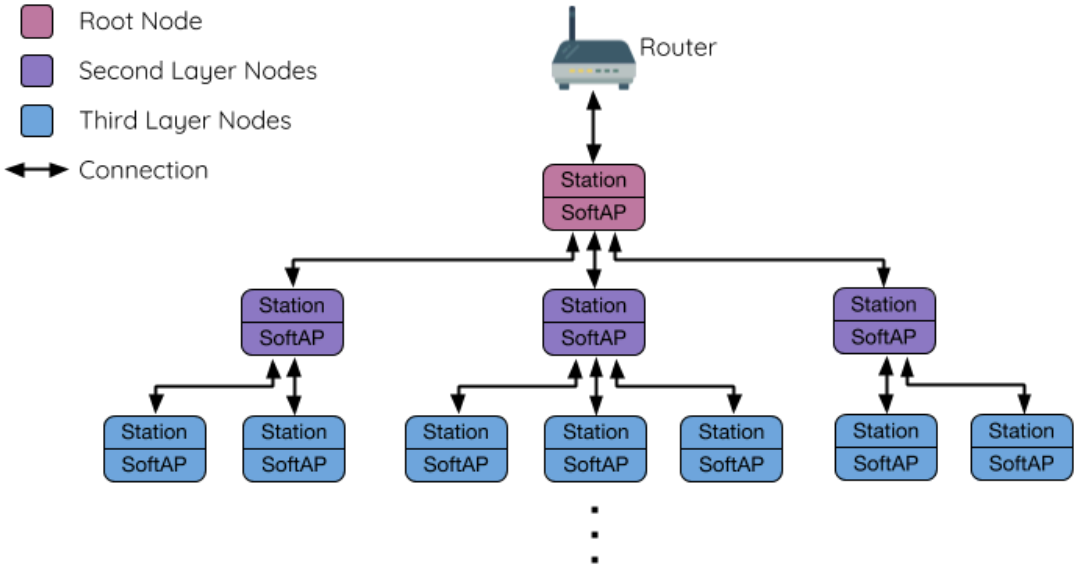


Рис. 1.16. Структурна схема *ESP-WIFI-MESH*

Таким чином технологія *Wi-Fi* є чудовим засобом для комунікації між елементами розподіленої сенсорної системи.

1.5.4. Вибір технології для комунікації

Для комунікації між пристроями було обрано бездротовий спосіб обміну даних. Було розглянуто існуючі рішення, зокрема *технології ZigBee, Bluetooth, Wi-Fi*.

Технологія *Bluetooth* має досить обмежений радіус дії та низьку пропускну здатність, але перевагою такої технології є низьке енергоспоживання. Ця технологія доцільна коли у нас є малі відстані між елементами системи. Використання такої технології на великих відстанях є недоцільним та дорогим.

Технологія *ZigBee* є привабливим способом комунікації, оскільки має більший радіус дії, так само є енергоефективним. Додатковою привабливою рисою є те що в системі присутнє з'єднання «кожен з кожним», що дозволяє просто розширювати систему. Проте швидкість обміну даних вже вимірюється не в мегабайтах, а в кілобайтах. Така швидкість не підходить коли є досить багато даних які потрібно передавати. Ця технологія є неможливою у використанні, коли в системі використовується відео-зйомка, оскільки для відео потрібно велика пропускна здатність, проте ця технологія добре підходитиме у випадку передачі даних, швидкість доставки яких не є критичною. Варто відмітити і той факт що деякі компоненти цієї технології є ліцензованими, то це означає, що потрібно витрачати додаткові кошти на ліцензію.

Технологія *Wi-Fi* має найбільший радіус дії та швидкість обміну даними серед зазначених. Цей фактор дає можливість охопити більше різних типів розподілених сенсорних систем. Технологія *ESP-WiFi-Mesh*, дозволяє легко будувати складні системи та додавати в неї нові пристрої, оскільки пристрої в мережі вже здатні до самоорганізації, без створення додаткового прикладного забезпечення. Фактично можна сказати *ESP-WiFi-Mesh* це надбудова над звичайним протоколом *Wi-Fi*, що дає додаткову можливість для легкого розширення системи.

В якості технології для комунікації в мережі було обрано технологію *Wi-Fi*, а також мережевий протокол *ESP-WIFI-MESH*.

1.6. Висновки до розділу

Було отримано загальні знання про розподілені системи. Розглянуто різні типи розподілених систем. Також визначено існуючі структури розподілених систем. Всі ці знання можна також застовувати до розподілених сенсорних систем.

Було розглянуто існуючі розподілені сенсорні системи та як вони працюють. Зокрема було розглянуті системи відеоспостереження, *IoT*(інтернет речей), системи моніторингу здоров'я, контролю за дорожнім рухом, моніторингу навколишнього здоров'я, віртуальної реальності(*VR*).

Було оцінено способи обміну даних між компонентами системи. Зокрема було визначено, що для розподіленої сенсорної системи провідний зв’язок є менш зручним, оскільки вони є більш дорогими та складнішими в експлуатації, більше того можливість переміщення сенсорних модулів в системі є проблемним процесом.

Застосування безпровідного зв’язку дозволяє пристроям змінювати положення в просторі без втрати зв’язку. Відсутність кабелів в системі полегшує ремонтні та діагностичні роботи. Використання безпровідного зв’язку дозволяє простіше додавати пристрої до системи.

Було розглянуто безпровідні технології *Bluetooth*, *ZigBee* та *Wi-Fi*. Розглянуто переваги та недоліки кожної із технологій. Технології *Bluetooth* та *ZigBee* мають досить низьку пропускну здатність та менший радіус дії, порівняно із технологією *Wi-Fi*. Для комунікації між пристроями в мережі було обрано технологію *Wi-Fi*.

РОЗДІЛ 2
ВИБІР ЗАСОБІВ РОЗРОБКИ

Для створення програмного забезпечення для збирання та зберігання даних із розподілених сенсорних модулів використовуються набір інструментів та технологій.

В цьому розділі виконується аналіз та вибір мови програмування, мікроконтролерів, фрейморків та інших засобів розробки для програмного засобу.

Для порівняння було відібрано популярні та доступні засоби.

Загалом потрібно обрати мову програмування та фреймворки для створення засобу та середовище розробки.

2.1. Порівняння мов програмування

Проаналізувавши існуючі мови програмування що підходять для роботи із мікроконтролерами, було з'ясовано що *C/C++, Assembler, MicroPython* (рис. 2.1) є найбільш популярними в цій сфері.

Рис. 2.1. Логотипи мов програмування *C++, Assembler, MicroPython*

2.1.1. C/C++

Особливості мов *C/C++*:

- Мова *C* забезпечує досить низькорівневий доступ до пам'яті чи регістрів, що робить його високоефективним при роботі із вбудованими системами чи операційними системами.
- Код мовою *C/C++* є ефективним і як правило дуже швидким, що є досить критичним для пристроїв із обмеженими можливостями, такими як мікроконтролерами, оскільки, як правило вони працюють на невисоких частотах і кожна операція має бути максимально ефективною [21].
- Мова *C++* є розширенням мови *C*, а це довзоляє використовувати *C*-елементи в програмах *C++*, що є великим плюсом.
- В мові *C++* є доступними класи, що дозволяє використовувати об'єктно-орієнтовані концепції, такі як поліморфізм, спадкування, абстракція, інкапсуляція.

Переваги:

- Висока продуктивність.
- Ефективне використання ресурсів.
- Широке використання від вбудованих систем до застосувань високого рівня.
- Велика спільнота користувачів та база знань.
- Можливість роботи без операційної системи.
- Підтримка вставок коду Assembler.
- Об'єктно-орієнтоване програмування.

Недоліки:

- Вимагає більше коду для досягнення конкретного результату порівняно із високорівневими мовами.
- Залежно від рівня знань може бути складніше виправлення помилок.
- Відсутність стандартів для вбудованих систем.

2.1.2. Assembler

Особливості мови *Assembler*:

– *Assembler* є низькорівневою мовою програмування, яка використовує команди, що є специфічними для конкретної архітектури чи процесора.
– Мовою *Assembler* забезпечується прямий доступ до апаратури, дозволяючи розробникам взаємодіяти із регістрами, пам’ятю чи периферійними пристроями. Завдяки цій можливості забезпечується найефективніший контроль ресурсами пристрою [22].
– Код мовою *Assembler* має найменший розмір та є найефективнішим, проте він є досить складним для розуміння і потребує великих зусиль при написанні.
– Код мовою *Assembler* є специфічним для кожної архітектури, тому він не є сумісним із іншою архітектурою, а відповідно і для іншого типу мікроконтролерів. Цією мовою доцільно створювати драйвери чи критично важливі ділянки коду.
– *Assembler* надає повний контроль та оптимізацію для специфічних завдань, але в той же час вимагає високого рівня експертизи в архітектурі процесора та не завжди є оптимальним вибором для розробки високорівневого програмного забезпечення.

Переваги:

– Прямий доступ до ресурсів мікроконтролера.
– Найбільша швидкість виконання коду.
– Ефективне використання пам’яті.

Недоліки:

– Складна мова для вивчення і використання.
– Великий обсяг коду для виконання простого завдання.
– Залежність від архітектури.
– Відсутність кросплатформенності.

2.1.3. MicroPython

Особливості мови *MicroPython*:

– *MicroPython* – це реалізація мови програмування *Python* для роботи з мікроконтролерами.
– Мова є високорівневою, що полегшує процес розробки та збільшує зрозумілість коду [23].
– Мова має обмежений доступ до ресурсів пристрою.
– Мова *MicroPython* є простою у вивченні та використанні, що є перевагою для початківців та розробників, що не мають глибоких знань в низькорівневому програмуванні.
– За швидкістю виконання та розміром коду, *MicroPython* є менш ефективним ніж мови *Assembler*, *C/C++* [24].
– *MicroPython* часто використовується для розробки програмного забезпечення для *IoT*-пристроїв, де важлива простота та швидкість розробки.
– Ця мова має підтримку для багатьох мікроконтролерів та архітектур, таких як: *STM32, RISC-V, PIC, ESP32, Microchip, Renesas RA*.

Переваги:

– Простий для вичення та розуміння.
– Підтримка великої кількості мікроконтролерів.
– Висока швидкість розробки.

Недоліки:

– Менша продуктивність ніж в мовах *C/C++, Assembler*.
– Обмежений доступ до ресурсів пристрою.

2.1.4. Вибір мови програмування

Для розробки програмного засобу було обрано мову програмування *C/C++*. Вибір цих мов програмування, дозволяє знайти середину між швидкістю розробки

та ефективним використанням ресурсів. Для мов *C/C++* існує вже багато рішень, що дозволяє частково або повністю використовувати їх у нових проектах.

2.2. Порівняння мікроконтролерів

Кожен елемент системи має свій власний мікроконтролер та набір сенсорних модулів. Існують різні сімейства мікроконтролерів, які можна класифікувати за різними ознаками. Досить важливою класифікацією є розмір шини даних та адрес, оскільки це має великий вплив на швидкодію та кількість доступної пам'яті.

Для порівняння було обрано три мікроконтролери: *ESP32, STM32F407, Arduino Uno Rev3*.

2.2.1. ESP32

Мікроконтролер *ESP32*(рис. 2.2) належить до класу мікроконтролерів на основі архітектури *Xtensa LX6 RISC. ESP32* має 32-бітну шину даних та 32-бітну шину адрес.

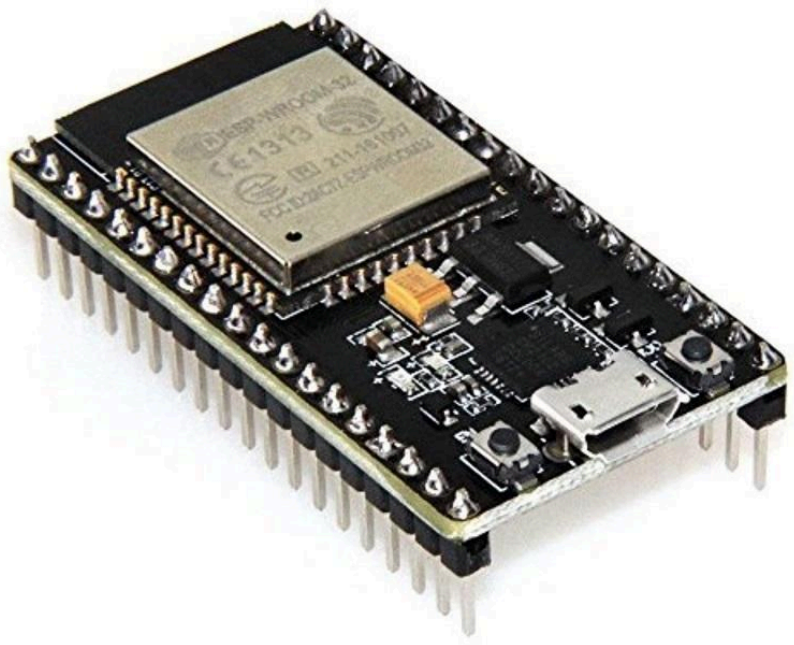


Рис. 2.2. Мікроконтролер *ESP32*

Особливості *ESP32*:

- Висока продуктивність: *ESP32* пропонує двоядерний процесор та широкий функціонал, що забезпечує високу продуктивність обробки даних. Максимальна частота кожного із ядер сягає 240 *МГц* [25].
- Широкі можливості підключення: Вбудований модуль *Wi-Fi* дозволяє зручно взаємодіяти з мережею для передачі даних [26].
- Підтримка мережі *MESH*: *ESP32* надає можливість створення *MESH* мережі, що дозволяє оптимізувати передачу даних між різними модулями.
- Широка спільнота та підтримка: *ESP32* користується великою активною спільнотою, що спрощує вирішення проблем та надає доступ до різноманітних бібліотек [27].
- Простота програмування: Застосування мови програмування *C/C++*, а також підтримка *Arduino IDE*, сприяють швидкій та зручній розробці.

2.2.2. STM32F407

Мікроконтролер *STM32F407*(рис. 2.3) належить до сімейства *STM32*, що базується на архітектурі *ARM Cortex-M4*. *STM32F407* має 32-бітну шину даних та 32-бітну шину адрес.

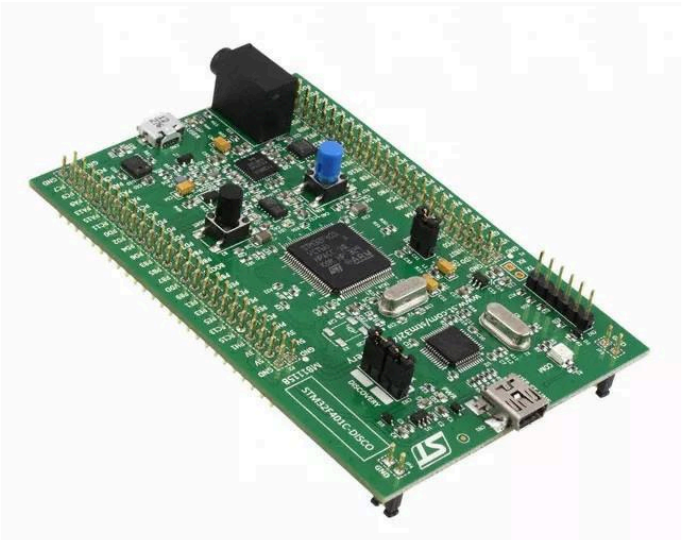


Рис. 2.3. Мікроконтролер *STM32F407 DISCOVERY*

Особливості *STM32F407*:

- Висока продуктивність та архітектура *ARM Cortex-M4*: Мікроконтролер *STM32F407* пропонує ядро *ARM Cortex-M4*, що забезпечує високу продуктивність та ефективну обробку даних. Мікроконтролер здатний працювати на частоті до 168 *МГц* та змінювати її під час роботи із досить великою градацією [28].
- Багатофункціональність та розширена периферія: *STM32F407* володіє розширеною набором периферійних пристроїв, включаючи вбудований *USB*, *Ethernet*, *SPI*, *I2C* та *UART* інтерфейси, що дозволяє ефективно взаємодіяти з різними пристроями та модулями [28].
- Багатошвидкісні порти та великий об'єм пам'яті: Наявність багатошвидкісних портів та значний об'єм вбудованої Flash-пам'яті та оперативної пам'яті робить *STM32F407* відмінним вибором для проектів з великою кількістю даних [28].
- Широкий вибір інтегрованих інструментів розробки: Компанія *STMicroelectronics* надає багато інтегрованих середовищ розробки, таких як *STM32CubeMX* та *STM32CubeIDE*, що спрощує налаштування та розробку програмного забезпечення [29].
- Активна спільнота та доступність ресурсів: *STM32* використовується великою та активною спільнотою, що полегшує обмін знань, вирішення проблем та забезпечує доступ до багатьох ресурсів [29].
- Надійність та промислова відповідність: *STM32* має репутацію надійного та стабільного мікроконтролера, який відповідає вимогам промислових застосувань [29].

2.2.3. Arduino Uno Rev3

Мікроконтролер *Arduino Uno Rev3*(рис. 2.4) входить до сімейства мікроконтролерів *AVR*, яке виготовляється компанією *Atmel* (закуплена корпорацією *Microchip Technology*). В основі мікроконтролерів *AVR* лежить архітектура *RISC*

(Reduced Instruction Set Computing). *ATmega328*, який використовується в *Arduino Uno Rev3*, має 8-бітну шину даних та 16-бітну шину адрес.

Рис. 2.4. Мікроконтролер *Arduino Uno Rev3*

Особливості *Arduino Uno Rev3*:

– Простота використання та доступність: *Arduino Uno Rev3* є ідеальним вибором для початківців та тих, хто шукає простоту у розробці вбудованих систем [30].
– Широкий вибір бібліотек і спільнота: Маючи велику спільноту та широкий вибір бібліотек, *Arduino Uno* спрощує розробку і забезпечує доступ до готових рішень [30].
– Невеликі розміри та енергоефективність: *Arduino Uno Rev3* володіє компактними розмірами та ефективним використанням енергії, що є чудовим рішенням для проектів з обмеженими ресурсами [30].

– Широкий вибір входів/виходів та розширюваність: Маючи достатньо входів/виходів для підключення сенсорів та інших пристроїв, *Arduino Uno* дозволяє розширювати функціонал проекту [30].
– Інтегроване середовище розробки (*IDE*): *Arduino IDE* - це просте та зручне середовище розробки, що полегшує написання та відлагодження коду [30].

2.2.4. Вибір мікроконтролера

Враховуючи вище наведені характеристики мікроконтролерів найбільш привабливими мікроконтролерами для системи є *STM32F407* та *ESP32*, оскільки вони мають 32-бітну шини даних та адрес. Також важливою характеристикою є висока продуктивність в *STM32F407* та *ESP32*. Ці мікроконтролери також здатні до гнучкого керування частотою *CPU* під час роботи.

Мікроконтролер *Arduino Uno Rev3* має лише одну перевагу над іншими мікроконтролерами – це просте використання для початківців.

Було обрано мікроконтролер для проекту *ESP32*. *ESP32* має більшу максимальну частоту *CPU* ніж *STM32F407*. Також *ESP32* має на чіпі 2 ядра, коли в *STM32F407* лише одне.

Ключовою перевагою *ESP32* над іншими є наявність радіомодулів на платі, що дозволяє економити кошти та використовувати вбудовані функції в фреймворк для роботи із цими радіомодулями.

2.3. Вибір фреймворку

Для розробки програмного засобу буде використовуватись 2 *фреймворки: Qt* та *ESP-IDF*.

Фреймворк *Qt* написаний мовою *C++,* а *ESP-IDF* на мові *C*. Проте обидва фреймворки доступні в мові *C++*. В *ESP-IDF* також є часткова підтримка мови *C++*.

2.3.1. *Qt*

Qt є потужним інструментарієм для розробки крос-платформених десктопних та мобільних додатків. Цей фреймворк розроблений компанією *Qt Company* та широко використовується у різних галузях, включаючи програмування інтерфейсів користувача, графічні додатки, а також розробку програмного забезпечення для вбудованих систем [31].

Логотип фреймворку зображено на рис. 2.5.

Рис. 2.5. Логотип *Qt*

Qt має великий набір бібліотек для графічного інтерфейсу, роботи із мережею, файлами і тд. Цей інструмент є кросплатформенним і слугує для розробки програм як для десктопних так і для мобільних пристроїв. Інструмент постійно оновлюється та розширюється.

Особливості *Qt*:

– Крос-платформеність: Однією з ключових переваг *Qt* є можливість розробки додатків, які легко переносити між різними операційними системами, такими як *Windows, macOS, Linux*, а також платформами для мобільних пристроїв [31].
– Доступність на різних мовах програмування: *Qt* базується на мові програмування *C++*. Також підтримується використання в інших мовах, таких як *Python* та *JavaScript*.

– Модульність: Фреймворк складається з різних модулів, що дозволяє розробникам використовувати тільки ті компоненти, які необхідні для їхнього конкретного проекту [31].
– Система подій та сигналів (*Signals and Slots*): Це потужний механізм, який дозволяє об'єктам взаємодіяти між собою, спрощуючи реалізацію взаємодій відповідно до певних подій [31].
– Підтримка баз даних: *Qt* має вбудовану підтримку роботи з різними системами управління базами даних (СУБД), що спрощує роботу з даними в додатках [31].
– Гнучкість та розширюваність: *Qt* надає засоби для створення як невеликих додатків, так і великих, складних програм [31].

Для роботи із *Qt* можна використовувати *Qt Creator*. Це *IDE*, що надає зручний інтерфейс для роботи із фреймворком.

Для збирання проектів в *Qt* використовується *qmake*.

Qt буде корисним для створення серверної частини системи, в якій буде виконуватись керування кожним елементом системи.

2.3.2. *ESP-IDF*

ESP-IDF (Espressif IoT Development Framework) є фреймворком для розробки програмного забезпечення на мікроконтролерах *ESP32* виробництва *Espressif Systems*. Цей фреймворк призначений для створення вбудованих систем для Інтернету речей (*IoT*) та інших застосувань, де потрібна висока продуктивність та підтримка бездротового зв'язку [32].

Логотип фреймворку зображено на рис. 2.6.

Особливості *ESP-IDF*:

– Вбудована підтримка ОС *FreeRTOS: ESP-IDF* використовує *FreeRTOS* як операційну систему в реальному часі, що робить його ідеальним для вбудованих систем з обмеженими ресурсами [33].

Рис. 2.6. Логотип *ESP-IDF*

– Вбудований *Wi-Fi* та *Bluetooth*: *ESP-IDF* має вбудовану підтримку *Wi-Fi* та *Bluetooth*, що робить його ідеальним для розробки *IoT*-проектів та інших додатків, які вимагають бездротового зв'язку [33].
– Висока продуктивність: Заснований на архітектурі *Xtensa LX6 RISC*, *ESP32* надає високий рівень продуктивності та оптимізовану обробку даних [33].
– Мережева система *Mesh*: *ESP-IDF* дозволяє створювати *mesh*-мережі, що полегшує взаємодію між різними пристроями та оптимізує передачу даних в розподілених системах [33].
– Інтерфейси комунікації: *ESP-IDF* підтримує різноманітні інтерфейси введення/виведення, такі як *GPIO*, *I2C*, *SPI*, *UART*, та інші, що дозволяє ефективно взаємодіяти з різними пристроями та сенсорами [33].
– Бібліотеки та компоненти: Фреймворк постачається з багатьма вбудованими бібліотеками та компонентами, що полегшує розробку та розширення функціоналу [33].
– Система відлагодження та профілювання: *ESP-IDF* надає засоби для відладки та профілювання коду, що спрощує виявлення та виправлення помилок [33].
– мікроконтролерів *ESP32* та *ESP8266*. Це дозволяє економити час при розробці проектів [33].

Фреймворк не має офіційної *IDE*, проте має багато плагінів/додатків для інтеграції фреймворку в інші *IDE*, такі як *Eclipse, VS Code* та інші.

2.4. Вибір середовища розробки

Для розробки програмного засобу буде зручно використовувати середовище розробки. Буде використовуватись дві *IDE: Qt Creator* та *Visual Studio Code.*

2.4.1. *Qt Creator*

Qt Creator - це інтегроване середовище розробки (*IDE*), спеціально розроблене для роботи з фреймворком *Qt*. Воно надає розробникам інтуїтивний і ефективний інтерфейс(рис. 2.7) для розробки крос-платформених десктопних та мобільних додатків на основі *Qt Framework*.

Qt Creator обрано через використання фреймворку *Qt*. Саме в *Qt Creator* є зручним використання всіх можливостей *Qt*, оскільки це середовище було створено для нього [34].

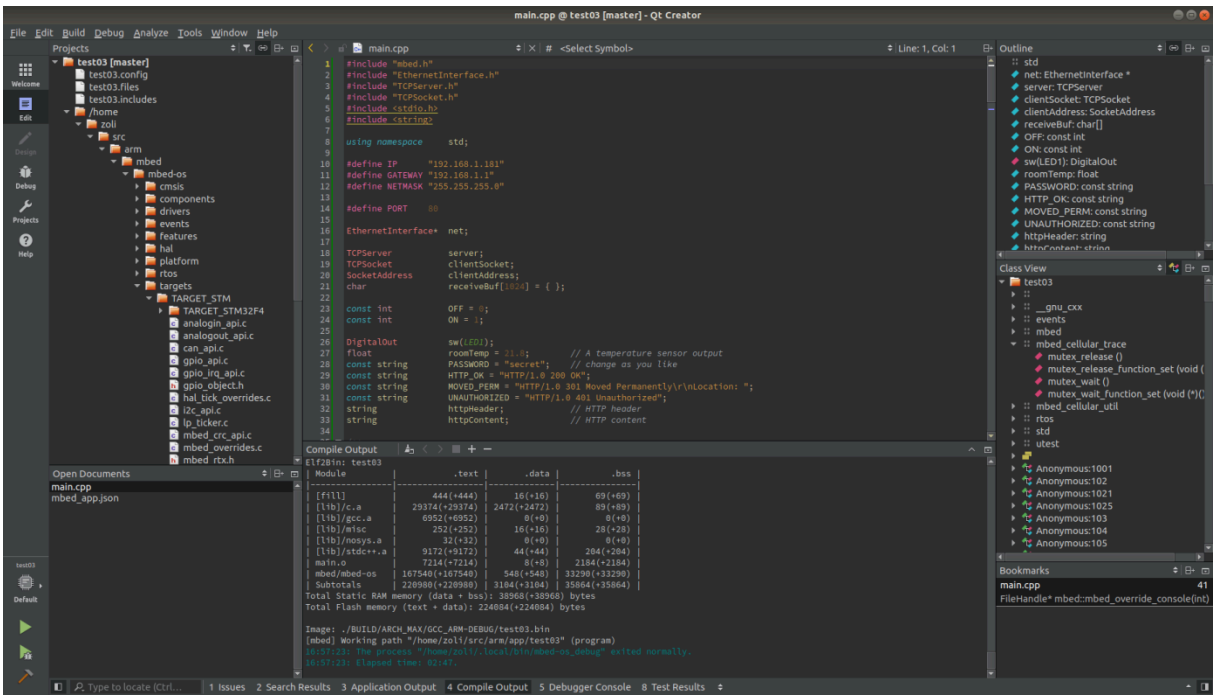


Рис. 2.7. Зовнішній вигляд *IDE Qt Creator*

Особливості *Qt Creator*:

- Підтримка мови програмування *C++: Qt Creator* дозволяє розробникам працювати з мовою програмування *C++*.
- Графічний дизайнер та редактор *QML: IDE* має вбудований графічний дизайнер і редактор для роботи з інтерфейсами користувача, що значно спрощує створення та відлагодження *GUI*-компонентів.
- Система управління проектами: *Qt Creator* підтримує роботу з різними типами проектів, включаючи консольні застосування, десктопні додатки, а також мобільні додатки для платформ, таких як *Android* та *iOS*.
- Інтегрована система відладки: *IDE* надає потужні інструменти для відлагодження коду, включаючи можливість крокування, перегляду значень змінних та відстеження викликів функцій.
- Автоматичне завершення коду та сповіщення про помилки: *IDE* автоматично доповнює код та виводить повідомлення про можливі помилки, що полегшує процес написання коду та виявлення помилок.
- Можливості профілювання: *Qt Creator* надає інструменти для аналізу продуктивності додатків, дозволяючи виявляти та усувати можливі проблеми з ефективністю.

Qt Creator широко використовується для розробки додатків з використанням *Qt Framework*, зокрема для створення крос-платформених додатків з сучасним інтерфейсом користувача.

Qt Creator ставить своєю метою полегшити розробку програмного забезпечення, особливо тих, що базуються на *Qt Framework*, надаючи інтегроване та зручне середовище для всього цього процесу.

2.4.2. *Visual Studio Code*

Visual Studio Code (VS Code) - це легкий та потужний текстовий редактор(рис. 2.8), який використовується для розробки різноманітних типів додатків, включаючи

веб-додатки, мобільні додатки та проекти з вбудованими системами. Він розроблений *Microsoft* та є безкоштовним та відкритим для використання [35].

VS Code має додаткову перевагу в можливості встановлення різних плагінів для підтримки конкретних фреймворків чи інструментів. Для розробки з *ESP-IDF* можна встановити спеціальний плагін, який додасть підтримку для цього фреймворку. Це може включати автоматичне завершення коду, інтегровану систему відладки, аналіз помилок та інші корисні функції, що сприяють ефективній розробці з *ESP-IDF*.

Особливості *Visual Studio Code*:

- Розширюваність та плагіни: *VS Code* надає велику кількість розширень та плагінів, що дозволяє розробникам налаштовувати робоче оточення під свої потреби.
- Інтегрована система відладки: Вбудована система відладки дозволяє розробникам відстежувати та виправляти помилки у своєму коді.
- Система управління версіями: Вбудована підтримка систем керування версіями (наприклад, *Git*) спрощує роботу над проектами в команді.
- Крос-платформеність: *VS Code* підтримує роботу на різних операційних системах, включаючи *Windows, macOS та Linux*.
- Розширена система автодоповнення та підказок: Редактор автоматично доповнює код та надає підказки, що полегшує написання коду та уникнення помилок.
- Швидкість та легкість використання: *VS Code* відомий своєю швидкістю та легкістю використання, що робить його відмінним вибором для широкого кола розробників.

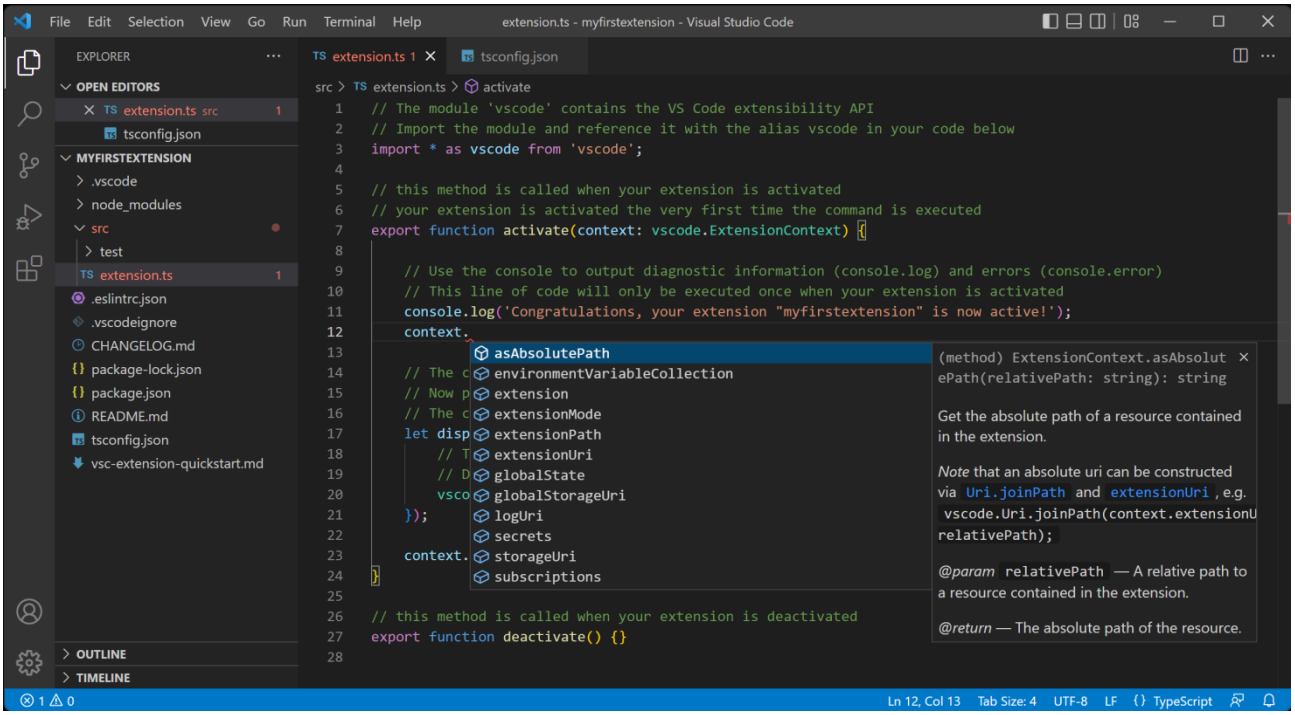


Рис. 2.8. Зовнішній вигляд *IDE Visual Studio Code*

Visual Studio Code (VS Code) використовується для розробки широкого спектру додатків та програмного забезпечення. Завдяки своїй легкості використання та високій розширюваності, редактор є популярним серед розробників на всіх етапах роботи з кодом. Від консольних застосунків до десктопних програм і вбудованих систем, *VS Code* надає зручне середовище для написання, відлагодження та підтримки коду. Можливість встановлення плагінів, також дозволяє розширити функціональність редактора, включаючи підтримку конкретних фреймворків, таких як *ESP-IDF* для вбудованих систем.

2.5. Висновки до розділу

Було розглянуто існуючі мови програмування для вбудованих систем та для десктопних рішень. Обрано мови *C/C++* для створення програмного коду, це дозволяє створити баланс між високою швидкодією коду та швидкістю розробки.

Кожен пристрій системи містить складається із мікроконтролера та сенсорних модулів. Кандидатами на роль мікроконтролера були *STM32F407*, *ESP32, Arduino*

Uno Rev3. Було обрано мікроконтролер *ESP32* через більшу продуктивність та наявність вбудованого радіомодуля, що полегшує роботу з ним.

Створення програмного коду для мікроконтролерів *ESP32* виконується із використанням фреймворку *ESP-IDF*. Цей фреймворк написаний мовою *C*, але він чудово адаптований для використання в мові *C++*. Поєднання *ESP-IDF* та *C++* пришвидшить швидкість розробки програмного коду та надасть нові можливості для створення більш гнучкої програмної архітектури прошивки.

Для створення серверної частини системи було обрано фреймворк *Qt*. Цей фреймворк написаний мовою *C++*. Фреймворк містить багато вже готових класів для створення графічного інтерфейсу та роботи із мережею, що значно полегшує створення програмного коду.

Qt Creator – це *IDE*, що використовується для створення проектів із використанням фреймворку *Qt*. Цю *IDE* було обрано для роботи із *Qt*, оскільки він є інтегрований в *Qt Creator*.

Для створення програмного коду для мікроконтролерів використовується *Visual Studio Code*. Це досить швидкий та зручний засіб для розробки програмного забезпечення. *IDE* має просту систему для підключення додаткових плагінів. Існує спеціальний плагін, що дозволяє легко додати інструменти для роботи з фреймворком *ESP-IDF*.

У цьому розділі було розглянуто, проаналізовано та обрано інструменти для створення програмного коду для мікроконтролера та серверної частини системи.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ЗБИРАННЯ ТА ЗБЕРІГАННЯ ДАНИХ З РОЗПОДІЛЕНИХ СЕНСОРНИХ МОДУЛІВ

Розробку програмного засобу для збирання та зберігання даних з розподілених сенсорних модулів було розділено на кілька етапів, в яких виконується розробка:

1) структури мережі для комунікації між сенсорними модулями;

2) алгоритму функціонування мікроконтролера;

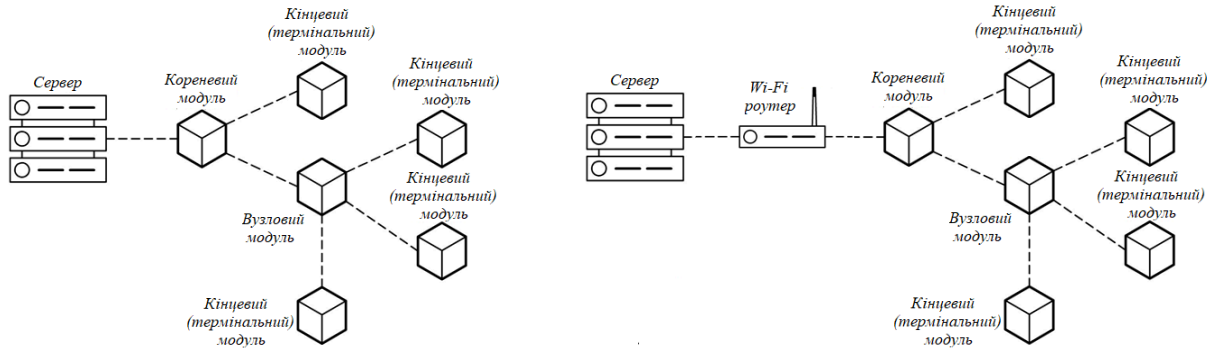
3) структури серверу;

4) забезпечення комунікації між сервером та сенсорними модулями.

Розроблений програмний засіб був протестований для перевірки його належного функціонування.

3.1. Розробка структури мережі розподілених сенсорних модулів

Структура мережі розподілених сенсорних модулів повинна виявляти динамічний характер взаємодій між елементами системи, в якій окремі пристрої можуть, за допомогою технології *mesh*, вільно змінювати комунікативні зв’язки між собою і адаптивно перебудовуватися у внутрішній комунікативній мережі, коли якийсь пристрій починає функціонувати не належним чином або виходить з ладу. Таке рішення дозволяє значно підвищити надійність функціонування системи як одного цілого. Виходячи з цього, а також для того, щоб запобігти формуваввню «хибних» мереж, які не матимуть доступу до сервера, була використана конфігурація мережі з деревоподібною (дендритною) структурою, в якій присутні корінь, вузли та закінчення дендриту і яка реалізується через наявність в мережі кореневого, вузлових та термінальних сенсорних модулів (рис. 3.1а). Така мережа може також містити *Wi-Fi* роутер, який забезпечує можливість для сервера отримувати доступ до кореневого пристрою, проте, у випадку, коли сервер має власний безпровідний мережевий інтерфейс, роутер в мережі є відсутнім (рис. 3.1б).



а) б)

Рис. 3.1. Схематичне зображення мережі розподілених сенсорних модулів: а) без роутера, б) з роутером

Сервер виконує керування системою і створює зв’язок із кореневим пристроєм. Для обміну даними між сервером та кореневим вузлом використовується стек протоколів *TCP/IP*, який забезпечує надійний обмін даними між ними. Відповідно сервер та кореневий пристрій повинні мати *IP* адресу. В мережі, яка розробляється, використовується протокол *IPv4*, проте можливим також є використання протоколу *IPv6*.

З боку сервера запускається відповідний програмний засіб, який починає моніторити вхідні з’єднання до певного порту. Після того як кореневий модуль під’єднається до мережі, він намагатиметься встановити з’єднання із сервером. У випадку, якщо таке з’єднання не відбудеться відразу, тоді ці спроби будуть періодично повторюватись нескінченну кількість разів. Після того, як між сервером та кореневим пристроєм буде встановлено зв’язок, кореневий модуль сповістить всі вузлові та термінальні пристрої про доступність до сервера. Кореневий та кожен інший модуль мережі відсилає повідомлення до сервера про свою доступність у мережі. Ці повідомлення йдуть до кореневого модулю, який, отримуючи ці повідомлення, перенаправляє їх до сервера, причому кореневий модуль чітко розрізняє від якого пристрою і куди направляється пакет.

Обмін даними між вузловими, термінальними та кореневим модулями виконується за допомогою протоколу *ESP-WIFI-MESH*, де кожен пристрій

ідентифікується не *IP*-, а *MAC*-адресою. Тому мережа поділяється на дві частини, де кореневий пристрій є сполучною ланкою між частинами (рис. 3.2).

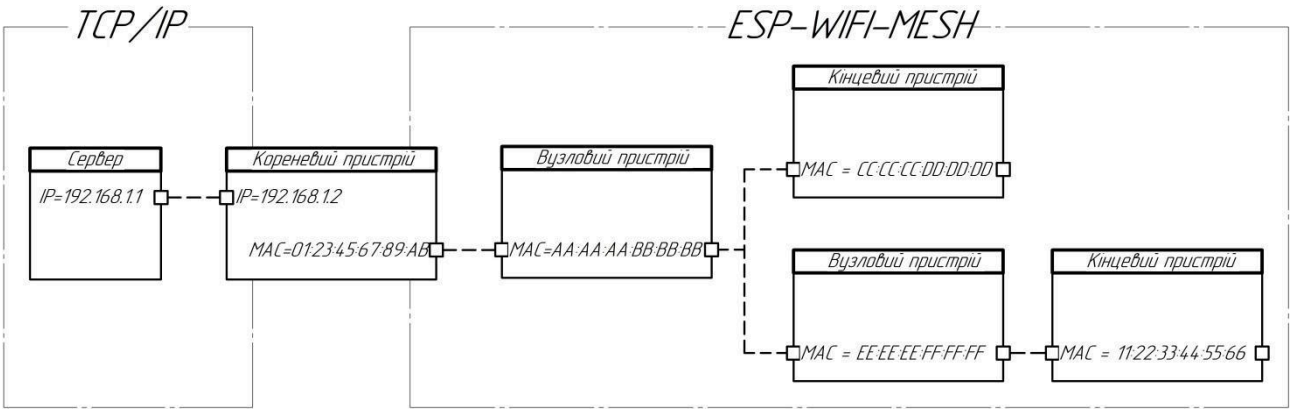


Рис. 3.2. Схематичне зображення мережі розподілених сенсорних модулів, що складається з двох частин

Таким чином, завданнями кореневого вузла є встановлення зв’язку з сервером, встановлення зв’язку з вузловими та термінальними пристроями, пересилання трафіку між сервером та сенсорними модулями.

Кореневий вузол обирається шляхом «голосування» окремих пристроїв, коли мережа ще не створена (рис. 3.3).

Пристрій, який характеризується найсильнішим сигналом від сервера чи роутера виграє таке «голосування». На протязі цього процесу кожен пристрій поширює в ефірі свої *MAC*-адресу та потужність сигналу із сервером. Паралельно кожен пристрій прослуховує *Wi-Fi* ефір і якщо знаходить в ефірі пристрій який має кращий сигнал ніж у нього і найкращий серед всіх, то він тепер буде поширювати *MAC*-адресу того пристрою і силу його сигналу, цей принцип дещо схожий на «сарафанне радіо». Після декількох визначених спроб голосування визначається пристрій з найбільшою кількістю голосів і він стає кореневим. Новий кореневий пристрій створив мережу і тепер він здатний приєднувати до себе інші пристрої. Кожен пристрій що не є в мережі шукає в своєму ефірі пристрої, що вже приєднані до мережі. Приєднання пристроїв є паралельним процесом до всіх інших, тому інші пристрої що вже доєднались до мережі можуть виконувати свої завдання. Через те

що приєднання пристроїв виконується послідовно, то це запобігає «великому вибуху» трафіку в мережі.

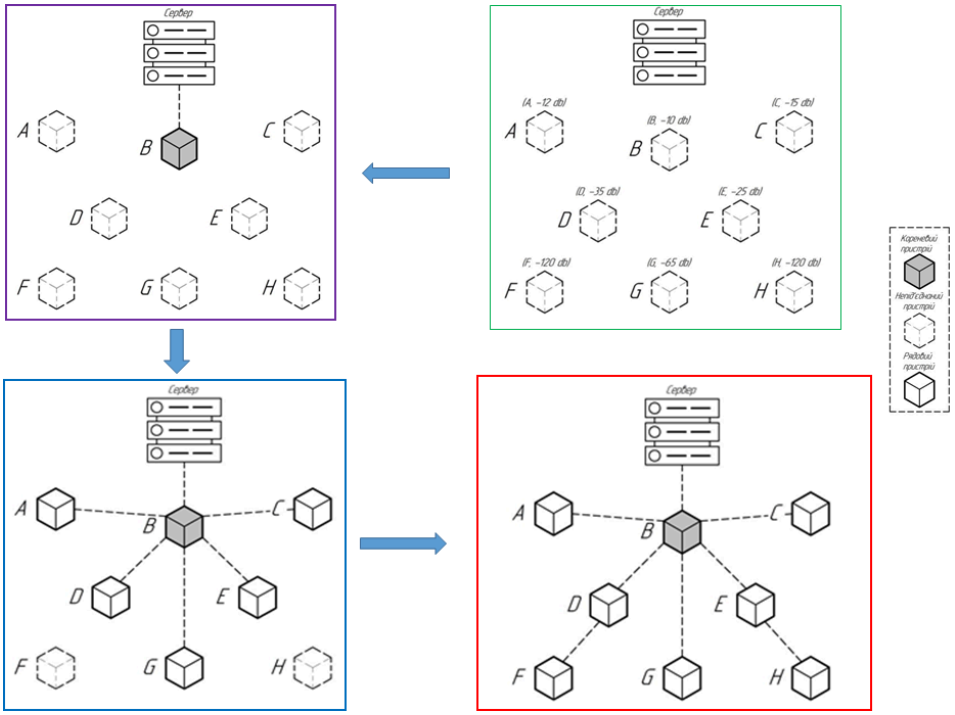


Рис. 3.3. Процес вибору кореневого пристрою та підключення пристроїв до мережі

Варто зазначити, що *mesh* мережа складається з рівнів, починаючи із першого рівня, який належить кореневому вузлу, він збільшується на одиницю (рис. 3.4). Можна сказати, що віддаленість від сервера характеризується поточним рівнем пристрою в мережі.

Система має бути відмовостійкою та надійною. У випадку якщо один із модулів виходить з ладу, це не може бути причиною зупинки функціонування всієї мережі. Якщо термінальний сенсорний модуль помічає відсутність вузлового модулю, він спочатку знаходиться в режимі очікування. Якщо через визначений час вузловий пристрій не повертається в мережу, то термінальний пристрій просто приєднується до сусіднього пристрою з найкращим сигналом доступу до нього. Схему процесу відновлення мережі при пошкодженні кінцевого пристрою показано на рис. 3.5.

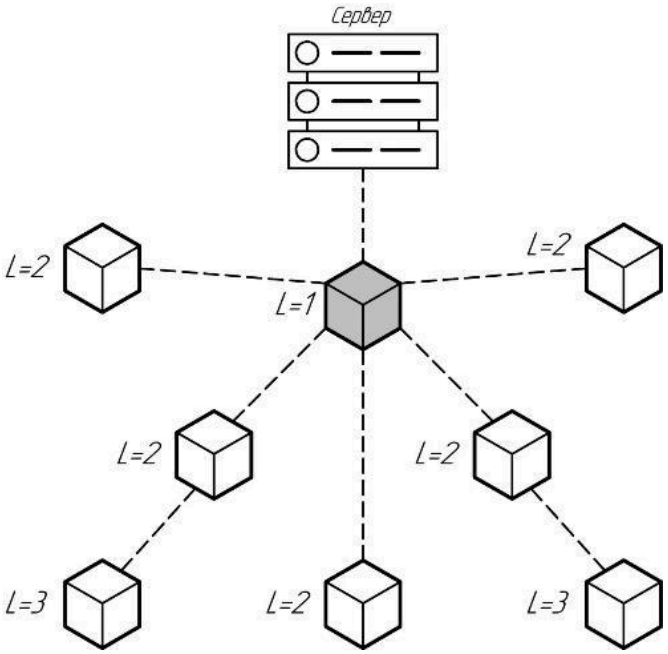


Рис. 3.4. Схематичне зображення з визначеними рівнями пристрою

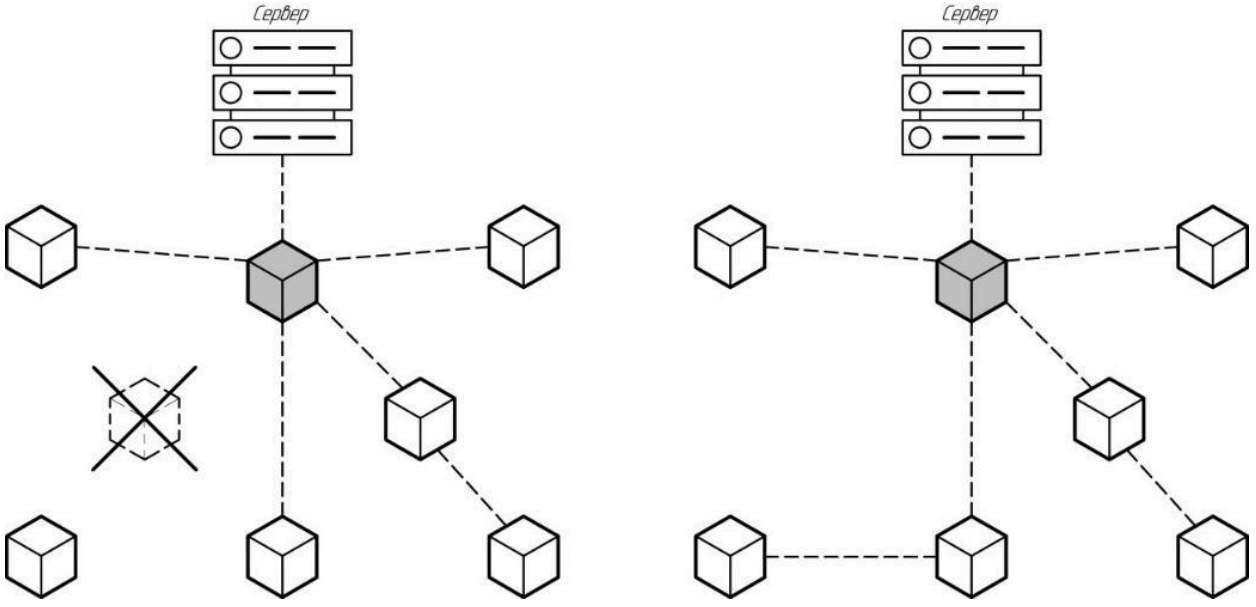


Рис. 3.5. Процес відновлення мережі при пошкодженні одного з пристроїв

Якщо з ладу виходить кореневий пристрій, то всі пристрої, так само, визначений час знаходитимуться в режимі очікування, а потім почнуть перебудовуватися в нову комунікативну мережу. Відсутність кореневого пристрою

означає необхідність повторного голосування, проте в такому випадку голосування буде виконуватись серед рівня *L=2*, оскільки вони мають найкращі сигнали серед інших пристроїв (рис. 3.6).

Після завершення голосування, всі пристрої рівня *L=2* автоматично під'єднаються до нового кореневого пристрою (рис. 3.7).

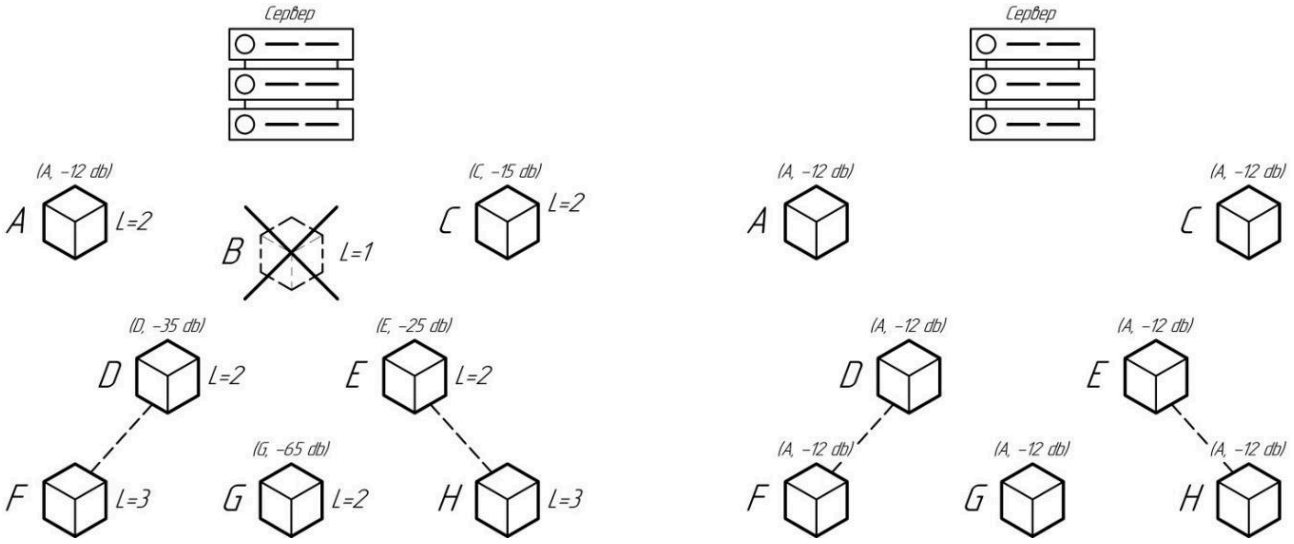


Рис. 3.6. Процес голосування при втраті кореневого пристрою

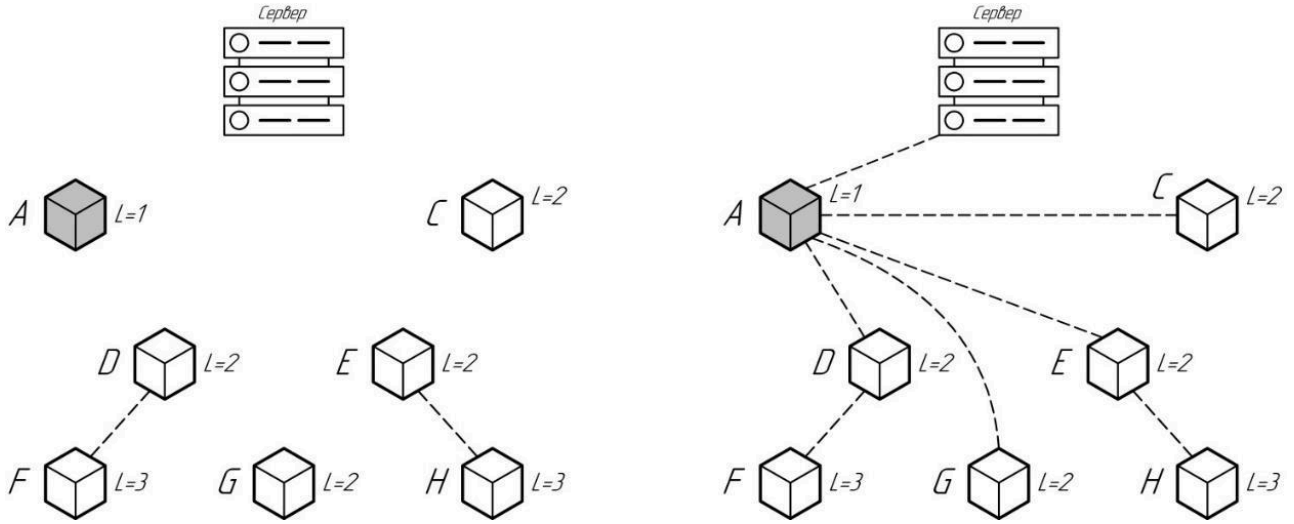


Рис. 3.7. Процес відновлення мережі після втрати кореневого пристрою

Автовідновлення кореневого пристрою не вимагає перевизначення рівня пристрою в мережі. Проте відновлення кореневого пристрою потребує процесу голосування, хоч воно виконується і в більш спрощеному вигляді, на практиці це займає приблизно стільки ж часу, скільки і при першому голосуванні.

3.2. Структура програми для мікроконтролера

Центральною частиною сенсорного модулю є мікроконтролер *ESP32*, який має 2 ядра із максимальною частотою 240 *МГц*, що дає додаткові можливості для виконання паралельних операцій. Під час створення прошивки всі операції, які не пов’язані із роботою в мережі виконуються на першому ядрі(тобто 0), а операції пов’язані із обміном даних через мережу на другому ядрі(тобто 1).

3.2.1. Ініціалізація компонентів мікроконтролера

Сенсорні модулі повинні бути ініціалізовані перш, ніж пристрій з’явиться в мережі. Тому, на першому етапі виконується ініціалізація сенсорних модулів та механізму для роботи із ними. Наступним етапом є ініціалізація флеш-сховища, яка виконується для того, щоб радіомодулі могли зберігати деякі свої налаштування. Також ця ініціалізація дозволяє зберігати власні дані в флеш-пам’яті, наприклад налаштування пристрою, версію прошивки, тощо.

Для роботи в мережі спочатку потрібно виконати ініціалізацію радіомодулю *Wi-Fi*. Під час ініціалізації виконується базове налаштування модулю, виключення *DHCP* серверу та клієнта, створення реакції на події що стаються в мережі. Це не є налаштуванням *MESH*-мережі, а тільки лише самого модулю *Wi-Fi*.

Наступним кроком є ініціалізація *MESH*-мережі. На цьому етапі виконується доналаштування модулю *Wi-Fi*, визначення топології мережі, максимальної кількості пристроїв в мережі та рівнів, *SSID* та пароль до мережі. Також на цьому етапі можна визначити тривалість голосування та час до автоматичного відновлення мережі, якщо відсутній якийсь пристрій. Ініціалізація *MESH*-мережі завершується

визначенням функцій, що оброблюватимуть різні події, які відбуваються в *MESH*-мережі.

Останнім етапом в ініціалізації пристрою є ініціалізація механізму для обробки вхідних пакетів отриманих із мережі. Під час ініціалізації виконується реєстрування функції-обробника для відповідного пакету даних (рис. 3.8)**.**

Наступним кроком є ініціалізація *MESH*-мережі. На цьому етапі виконується доналаштування модулю *Wi-Fi*, визначення топології мережі, максимальної кількості пристроїв в мережі та рівнів, *SSID* та пароль до мережі. Також на цьому етапі можна визначити тривалість голосування та час до автоматичного відновлення мережі, якщо відсутній якийсь пристрій. Ініціалізація *MESH*-мережі завершується визначенням функцій, що оброблюватимуть різні події що відбуваються в *MESH*-мережі.

Останнім етапом в ініціалізації пристрою є ініціалізація механізму для обробки вхідних пакетів отриманих із мережі. Під час ініціалізації виконується реєстрування функції-обробника для відповідного пакету даних.

Рис. 3.8. Схема алгоритму роботи прошивки мікроконтролера

3.2.2. Обробка подій мережі

Після того як пристрій виконав ініціалізацію виконується наступний етап, що вже залежить від подій що стаються в мережі чи отриманих пакетів даних. Прикладами подій є підключення до батьківського пристрою, отримання доступу до сервера, втрата доступу до сервера, відключення дочірнього пристрою, тощо.

Обробка подій *MESH*-мережі завжди супроводжується логуванням в термінал. Спеціальна обробка подій застосовується лише до деяких, оскільки це вимагається для забезпечення комунікації між мережею та пристроєм.

Нижче наведено список всіх подій які логуються в термінал:

- *MeshEventStarted*: виникає при успішному запуску мережі *ESP-MESH*. Це вказує на те, що поточний пристрій став активним членом мережі та готовий до обміну даними з іншими пристроями.
- *MeshEventStopped*: виникає, коли мережу *ESP-MESH* призупинено або зупинено. Це може статися у випадку помилки, відключення або спеціального вимкнення мережі. Після виникнення цієї події, взаємодія в межах мережі *ESP-MESH* призупинена, і поточний пристрій вважається відключеним від мережі.
- *MeshEventChildConnected*: спостерігається після того, як дочірній пристрій успішно приєднався до мережі *ESP-MESH* та встановив поточний пристрій як свого батька. Ця подія свідчить про успішне створення нового вузла в мережі та визначає його підключення до поточного вузла.
- *MeshEventChildDisconnected*: виникає, коли дочірній пристрій від'єднався від поточного пристрою у мережі *ESP-MESH*. Це може статися через втрату зв'язку, вимкнення дочірнього пристрою або інші обставини, що призводять до відключення вузла. Ця подія дозволяє відстежувати стан підключених пристроїв та реагувати на їхнє відключення від мережі.
- *MeshEventRoutingTableAdd*: виникає, коли в мережі *ESP-MESH* додається новий запис до таблиці маршрутизації. Ця подія вказує на успішне встановлення маршруту до конкретного пристрою в мережі. При

виникненні цієї події система отримує інформацію про новий маршрут та його параметри, такі як адреса призначення, шлях до цієї адреси, та інші важливі характеристики.

– *MeshEventRoutingTableRemove*: виникає, коли в мережі *ESP-MESH* видаляється запис з таблиці маршрутизації. Це може статися у випадку втрати зв'язку з конкретним пристроєм або при перестановці мережевої топології. Ця подія вказує на видалення маршруту та надає інформацію про видалений запис.
– *MeshEventParentConnected*: спостерігається після успішного підключення поточного пристрою до нового батьківського пристрою у мережі *ESP-MESH*. Ця подія вказує на зміну статусу батьківського пристрою та підтверджує успішну реконфігурацію мережі.
– *MeshEventParentDisconnected*: виникає, коли батьківський пристрій відключається від поточного пристрою у мережі *ESP-MESH*. Це може статися у випадку втрати зв'язку або інших причин, які призводять до відключення від батьківського вузла. Ця подія важлива для відслідковування стану батьківського вузла та адаптації до змін у топології мережі.
– *MeshEventLayerChange*: виникає, коли змінюється рівень поточного притрою в мережі *ESP-MESH*. Це може відбутися внаслідок оптимізації топології мережі або зміни ролі вузла. Подія надає інформацію про новий рівень вузла та його роль в системі *ESP-MESH*.
– *MeshEventRootAddress*: спостерігається при визначенні або зміні адреси кореневого вузла в мережі *ESP-MESH*. Ця подія надає інформацію про адресу поточного кореневого вузла, яка може бути корисною для відстеження кореневого вузла та його ролі у топології.
– *MeshEventVoteStarted*: виникає, коли в мережі *ESP-MESH* починається процес голосування для визначення оптимального маршруту. Це є частиною механізму вибору оптимального маршруту для передачі даних в мережі.

– *MeshEventVoteStopped*: виникає, коли процес голосування завершується в мережі *ESP-MESH*. Це свідчить про завершення вибору оптимального маршруту для передачі даних.
– *MeshEventRootSwitchReq*: виникає, коли вузол мережі *ESP-MESH* висловлює запит на зміну кореневого вузла. Це може статися, якщо поточний кореневий вузол втратив зв'язок або став недоступним.
– *MeshEventRootSwitchAck*: виникає після успішного підтвердження зміни кореневого вузла відповідно до запиту, висловленого подією *MeshEventRootSwitchReq*. Ця подія вказує на те, що мережа *ESP-MESH* успішно переключилася на новий кореневий вузол, і стало можливим продовжити нормальну роботу мережі.
– *MeshEventToDS*: виникає, коли в мережі змінюється стан доступності до сервера для інформування кожного пристроя *ESP-MESH* мережі.
– *MeshEventNetworkState*: спостерігається при зміні стану мережі *ESP-MESH*. Це може включати такі зміни, як перехід мережі в режим роботи, виявлення чи втрата з'єднання з іншими вузлами, або інші зміни, які впливають на стан та роботу мережі в цілому.

Для забезпечення працездатності системи, було перевизначено обробку подій для наступних подій:

– *MeshEventParentConnected*(рис. 3.9): виконується перевірка чи являється поточний пристрій кореневим для мережі і якщо це так, то запускається *DHCP*-клієнт для отримання *IP*-адреси, яка в подальшому буде використовуватися для комунікації із сервером. Також для кожного пристрою запускається завдання в іншу нитку для очікування пакетів даних і їх обробку з *MESH*-мережі.

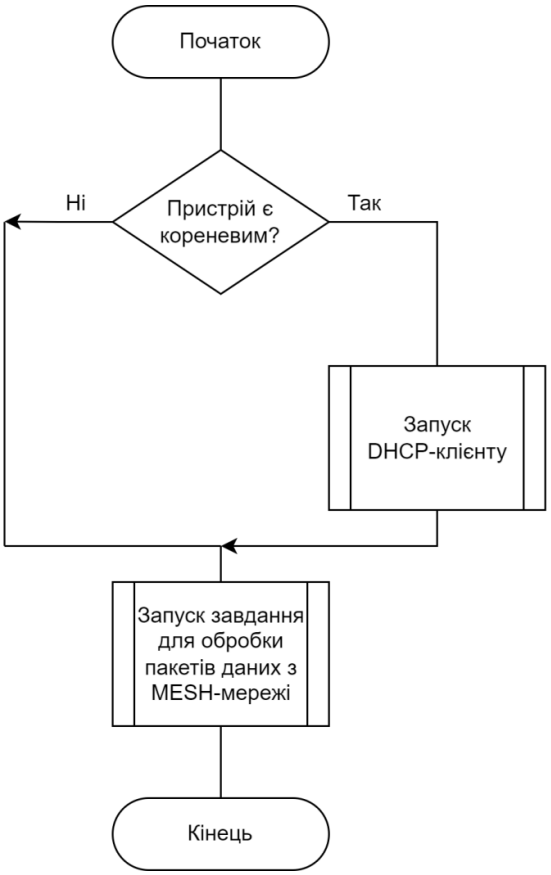


Рис. 3.9. Схема алгоритму обробки події *MeshEventParentConnected*

– *MeshEventParentDisconnected*: виконується зупинка завдання для обробки даних з *MESH*-мережі, що було створено під час обробки події *MeshEventParentConnected*.
– *MeshEventRootAddress*: після отримання події виконується оновлення адреси кореневого пристрою. Ця адреса буде використовуватись для надсилання запиту чи відповіді до сервера.
– *MeshEventToDS*(рис. 3.10): виконується перевірка чи є сервер доступним в мережі і чи є поточний пристрій кореневим. Якщо поточний пристрій є кореневим і сервер стає доступним, тоді відновлюється обробка пакетів даних отриманих від сервера і паралельно виконуються спроби з’єднатись із ним, після встановлення з’єднання із сервером йому надсилається пакет даних про готовність до роботи. Якщо поточний пристрій не є кореневим,

але сервер стає доступним, тоді пристрій посилає сигнал до кореневого пристрою про те що він знаходиться в мережі і готовий до роботи.

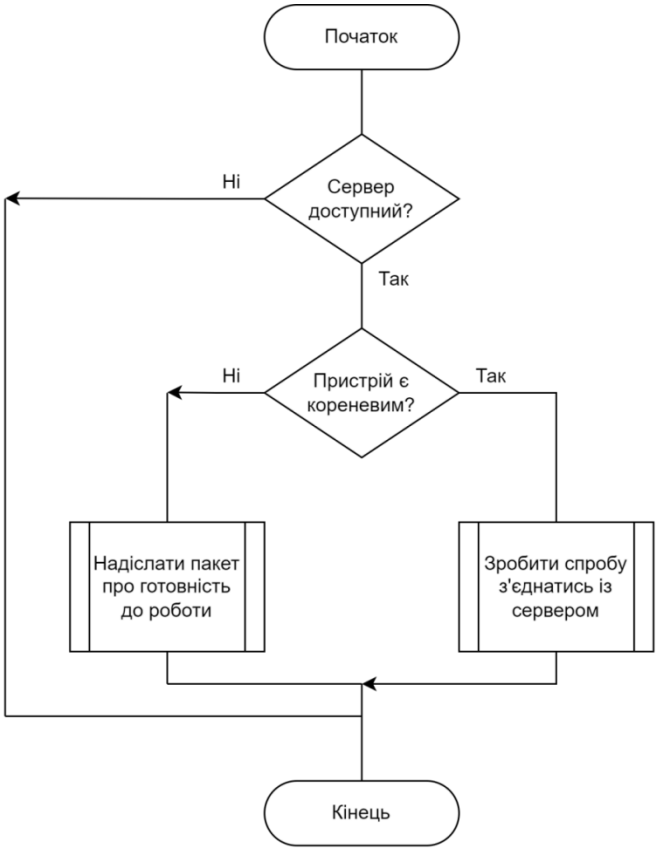


Рис. 3.10. Схема алгоритму обробки події *MeshEventToDS*

3.2.3. Задача контролю з’єднання з сервером

Для підтримки зв’язку кореневого пристрою із сервером, на кореневому пристрої виконується фонова задача, що час від часу перевіряє з’єднання із сервером. Якщо з’єднання з сервером відсутнє, то ця задача намагається встановити з’єднання. Після будь якої за результатом спроби задача переходить в режим сну.

Більшість свого часу ця задача проводить в режимі сну і лише час від часу прокидається за таймером щоб виконати своє завдання.

Ця задача не виконується, якщо пристрій не є кореневим.

Схема алгоритму задачі зображена на рис. 3.11.

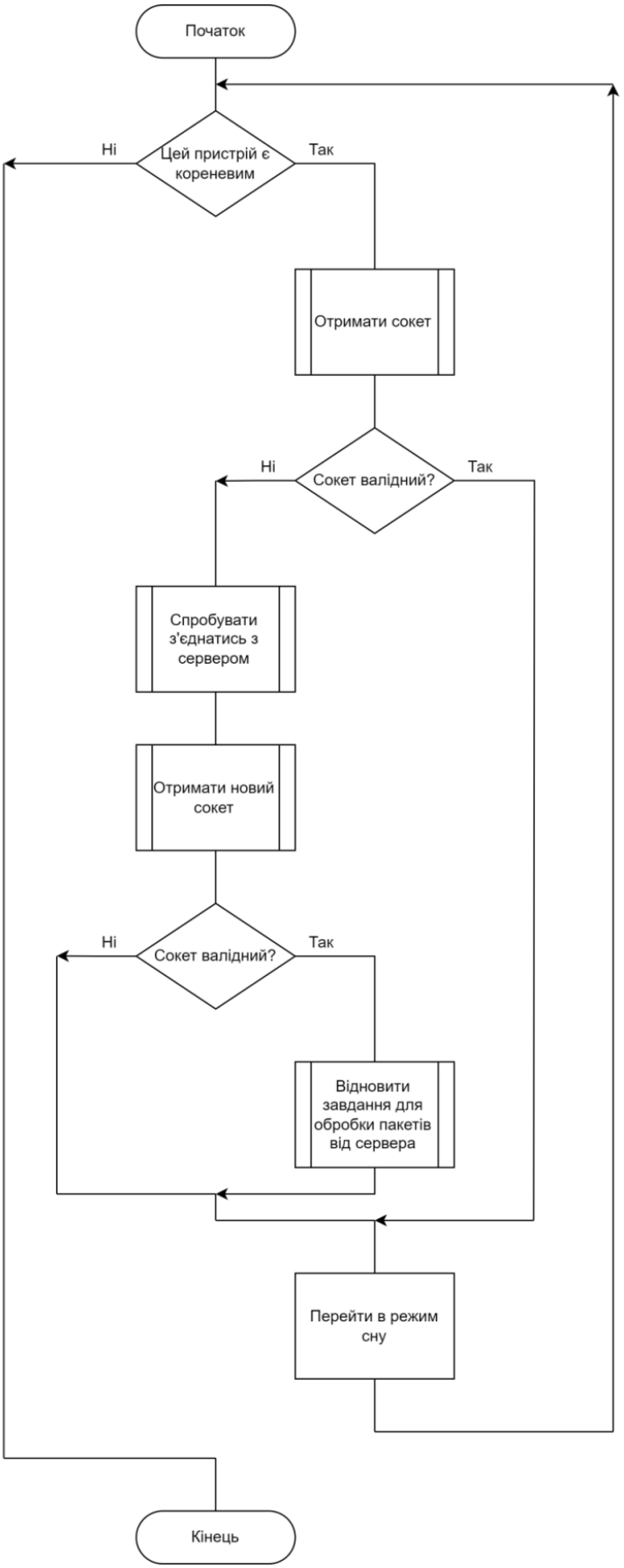


Рис. 3.11. Схема алгоритму задачі для контролю з’єднання з сервером

3.2.4. Задачі отримання пакетів даних з мережі

Для забезпечення ефективного отримання та обробки отриманих даних від сервера і *MESH*-мережі виконуються одночасно дві різні задачі. Ці завдання подібні одне до одного, основна відмінність між ними полягає лише в тому, як реалізований програмний код відповідно до типу мережі. Вони мають назву *receiveIPTask* та *receiveMeshTask*. Обидві функції орієнтовані на отримання та обробку даних, проте вони використовують різні підходи до реалізації цих операцій, забезпечуючи оптимальну роботу із отриманими інформаційними потоками.

Функція *receiveIPTask* спрямована на взаємодію із сервером за допомогою протоколу *TCP*. Ця завдання виконується лише на кореневому пристрої, оскільки саме він має можливість безпосередньо обмінюватись даними із сервером. Для роботи з мережею використовуються функції сокетів. Після аналізу отриманого пакету, цей пакет може бути переданий далі в *MESH*-мережу або відправлений до сервера. Це необхідно для забезпечення ефективної комунікації між сенсорними пристроями та сервером, дозволяючи обмінюватись необхідною інформацією для оптимального функціонування системи. Такий підхід гарантує, що дані збираються та передаються ефективно та точно, сприяючи взаємодії між різними частинами системи.

Функція *receiveMeshTask* призначена для організації комунікації між сенсорними пристроями з використанням протоколу *ESP-WIFI-MESH*. Це завдання виконується на всіх пристроях, включаючи кореневий, за умови, що пристрій підключений до мережі. Для взаємодії з мережею використовуються функції фреймворку *ESP-IDF*, які дозволяють отримувати пакети з *MESH*-мережі. Після аналізу отриманих пакетів вони можуть бути відправлені іншим пристроям у цій мережі або передані до сервера. Ця функція грає ключову роль у забезпеченні взаємодії між сенсорними пристроями та забезпечує оптимальний обмін даними для ефективної роботи системи в цілому. Такий підхід гарантує, що інформація від сенсорів надходить до системи та обробляється ефективно та з точністю, забезпечуючи надійність та швидкість обміну даними в мережі *ESP-WIFI-MESH*.

Схема алгоритму для задачі отримання даних із мережі зображена на рис. 3.12. Кожна із вказаних задач може бути призупинена, якщо пристрій не з'єднаний із мережею. Це рішення зроблено для оптимізації ресурсів. В разі, якщо пристрій активний у мережі, очікується на надходження пакетів. Після отримання даних, вони перевіряються на їхню валідність та виявлення можливих помилок.

У випадку, якщо дані виявляються невалідними, процес повторюється спочатку, а відповідна задача може бути тимчасово призупинена. Якщо отримані дані вважаються валідними, вони піддаються подальшому аналізу та обробці. Після завершення цього процесу, виконується наступна ітерація задачі.

Такий підхід дозволяє ефективно управляти задачами в залежності від доступності мережі та гарантує надійність обробки даних у випадку їхньої валідності, забезпечуючи оптимальне використання ресурсів пристрою.

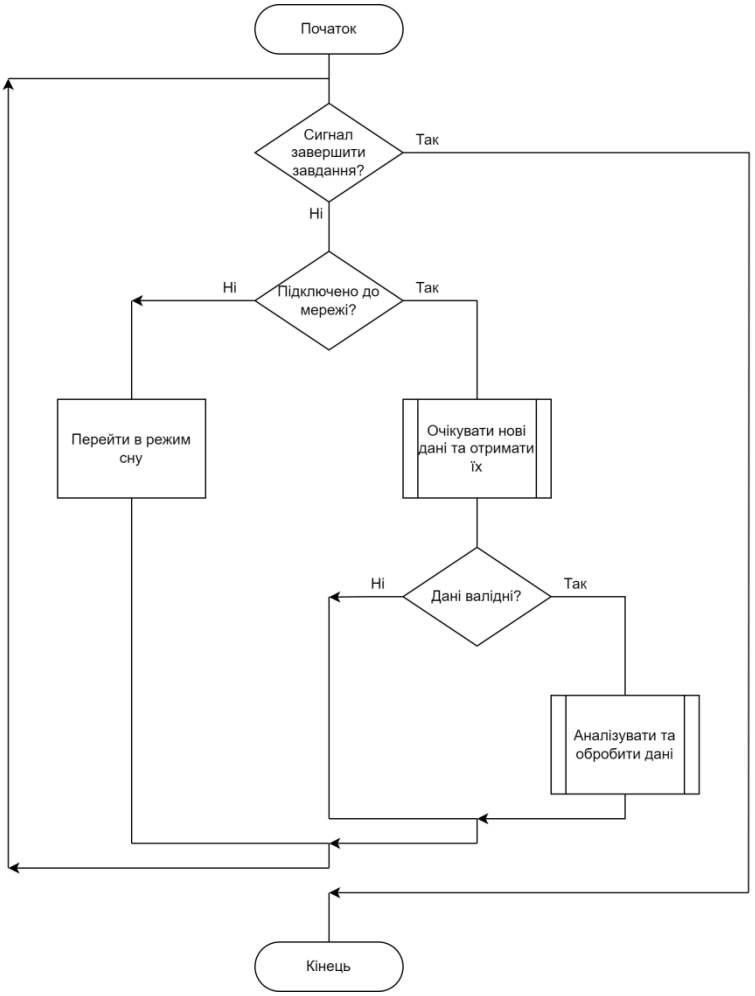


Рис. 3.12. Схема алгоритму задач для отримання і обробки даних із мережі

Процес аналізу даних(рис. 3.13), розпочинається із перетворення бінарних даних у структурований формат. Цей етап дозволяє надати дані зрозумілу та легко оброблювану структуру, щоб подальший аналіз був ефективним та точним.

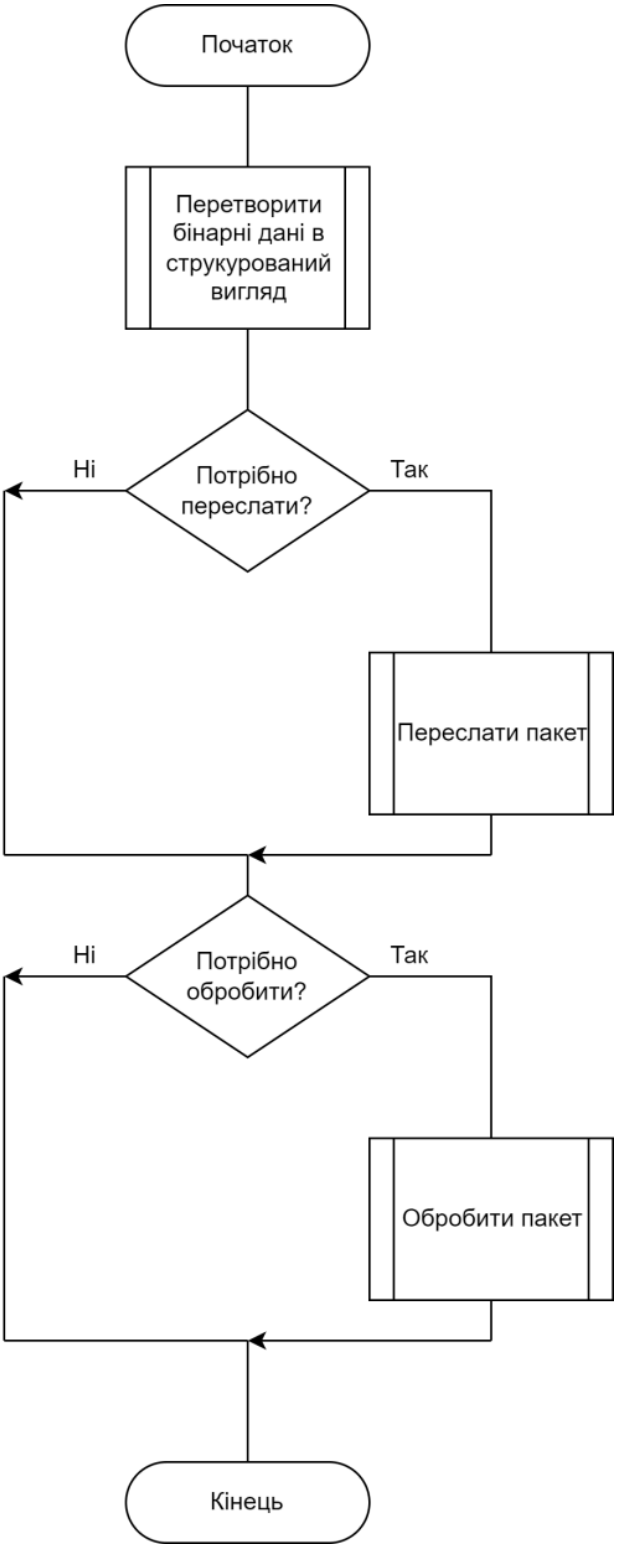


Рис. 3.13. Схема алгоритму аналізу та обробки даних

На наступному етапі проводиться перевірка необхідності подальшого пересилання цих даних(рис. 3.14). Якщо виявляється, що пересилання даних необхідне, вони пересилаються по мережі далі. Цей крок забезпечує оптимізований потік інформації, дозволяючи передавати лише ті дані, які мають значення для подальших обчислень та аналізу.

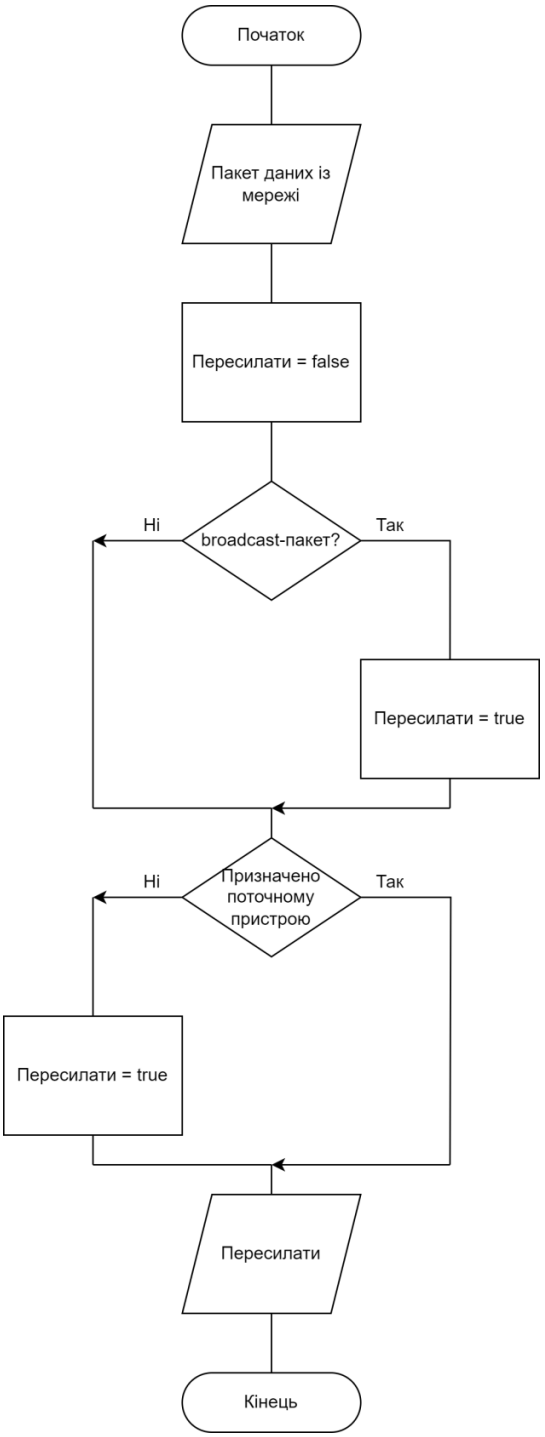


Рис. 3.14. Схема алгоритму перевірки пересилання пакету

Наступним етапом є перевірка необхідності обробки цих даних(рис. 3.15). Якщо виявляється, що пакет вимагає обробки, використовується відповідний механізм для обробки пакетів, що надходять з мережі. Цей процес дозволяє системі ефективно опрацьовувати інформацію, яка вимагає додаткового аналізу чи опрацювання.

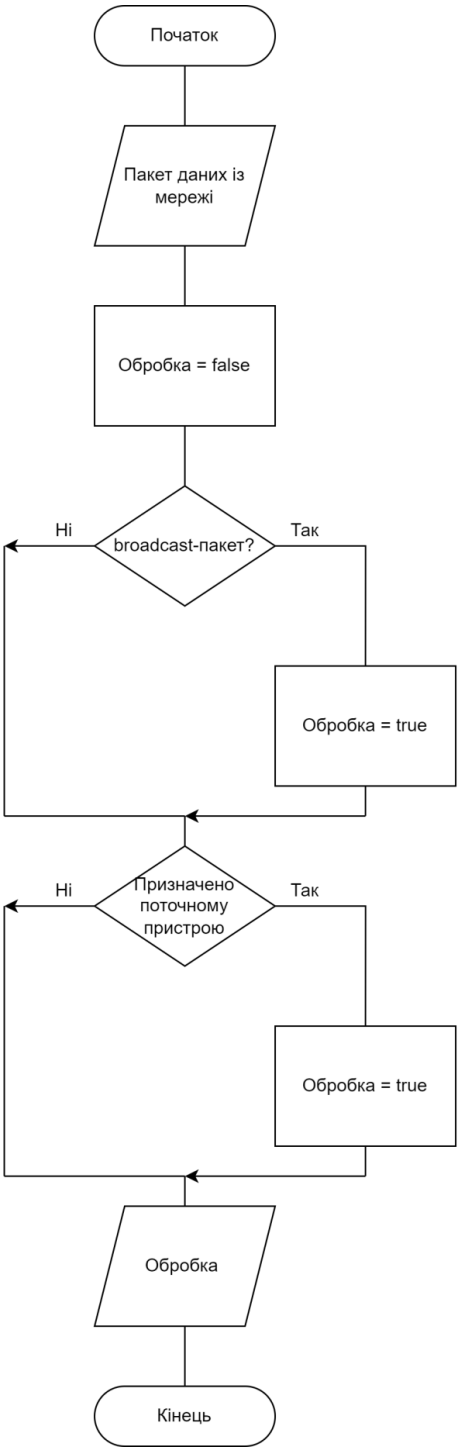


Рис. 3.15. Схема алгоритму перевірки обробки пакету

Після виконання пересилання та/або обробки пакету завершується весь процес аналізу та обробки. Ця фінальна фаза позначає успішне виконання всіх необхідних етапів, спрямованих на отримання, перевірку та подальшу обробку даних, що надходять з мережі. Такий підхід гарантує високу ефективність та точність управління отриманням та опрацюванням інформації в системі.

3.2.5. Механізм обробки вхідних пакетів з мережі

Механізм обробки вхідних пакетів виконує обробку пакетів надісланих з мережі та формує відповідь(якщо потрібно) для відправника. Механізм призначений для простого до впровадження та гнучкої у використанні обробки даних.

На рис. 3.16 зображено схему алгоритму роботи механізму.

Механізм обробки вхідних пакетів з мережі складається із словника, який містить інформацію про тип пакету та відповідну функцію-обробник. Коли виникає необхідність обробити пакет, він передається цьому механізму. При отриманні нового пакету механізм визначає його тип та використовує словник для пошуку відповідної функції-обробника для цього типу пакету.

Після визначення типу пакету механізм перевіряє, чи існує функція-обробник для цього типу в словнику. Якщо знайдено відповідний обробник, викликається відповідна функція для обробки пакету. У випадку, якщо для конкретного типу пакетів не знайдено відповідної функції-обробника, пакет відкидається.

Цей механізм дозволяє ефективно та гнучко обробляти різні типи вхідних пакетів, забезпечуючи можливість визначати та використовувати власні обробники для кожного типу пакету в системі.

На етапі ініціалізації мікроконтролера, словник заповнюється функціями-обробниками для різних типів пакетів. Цей підхід надає можливість гнучкого налаштування системи, де теоретично можливо додавати та видаляти обробники пакетів під час роботи мікроконтролерів. Це реалізовано з метою надання системі спроможності адаптуватися до змінних вимог у процесі експлуатації.

Рис. 3.16. Схема алгоритму роботи механізму обробки пакетів з мережі

Цей механізм створено для того, щоб надавати інженерам і розробникам можливість динамічно взаємодіяти з обробниками пакетів, що може бути особливо корисним під час етапу розробки та підтримки. В подальших розробках такий підхід може бути ключовим, сприяючи гнучкій адаптації системи до різноманітних завдань та вимог, що можуть змінюватися протягом часу.

Вимога до функції-обробника обмежується лише тим, що ця функція повинна приймати масив даних, після чого автоматично створиться структурований пакет для подальшої обробки. Це визначення виражає велику гнучкість у розробці, оскільки дозволяє передавати будь-які дані у функцію-обробник та визначати власну реалізацію обробки, не будучи обмеженим конкретним типом даних.

Такий підхід надає розробникам велику свободу у визначенні функцій-обробників, де вони можуть динамічно адаптувати свої обробники під конкретні завдання чи формати даних. Це є ключовим елементом гнучкої архітектури, яка спрощує розробку та підтримку системи в майбутньому.

3.2.6. Механізм управління сенсорними модулями

Механізм управління сенсорними модулями відзначається як одне з ключових завдань пристрою, спрямованих на ефективне керування усіма сенсорними модулями. Кожен із сенсорних модулів володіє однаковими або подібними операціями для взаємодії із ним. Вигідним вибором було визначити загальні функції абстрактного сенсорного модуля, що включають:

- Ініціалізація
- Налаштування модулю
- Скидання налаштувань
- Увімкнення/вимкнення модулю
- Отримання показників із сенсорів
- Отримання назви модулю
- Отримання списку показників, що здатний генерувати модуль

– Очищення даних
– Зберігання даних

Цей підхід до стандартизації функцій сенсорних модулів дозволяє ефективно управляти їхнім функціоналом, спрощуючи інтеграцію та обслуговування в рамках системи.

Ці базові операції є фундаментом абстрактного класу сенсорного модулю, оскільки вони усі або більшість з них є загальними для будь-якого сенсорного модулю. Механізм керування сенсорними модулями не має відчувати різницю між конкретними модулями, і всі вони повинні виглядати однаково для цього механізму з точки зору внутрішньої реалізації. Таким чином, всі сенсорні модулі повинні успадковувати від цього абстрактного класу, який відображає загальні характеристики та функції, що визначають їхню взаємодію з системою, як показано на рис. 3.17.

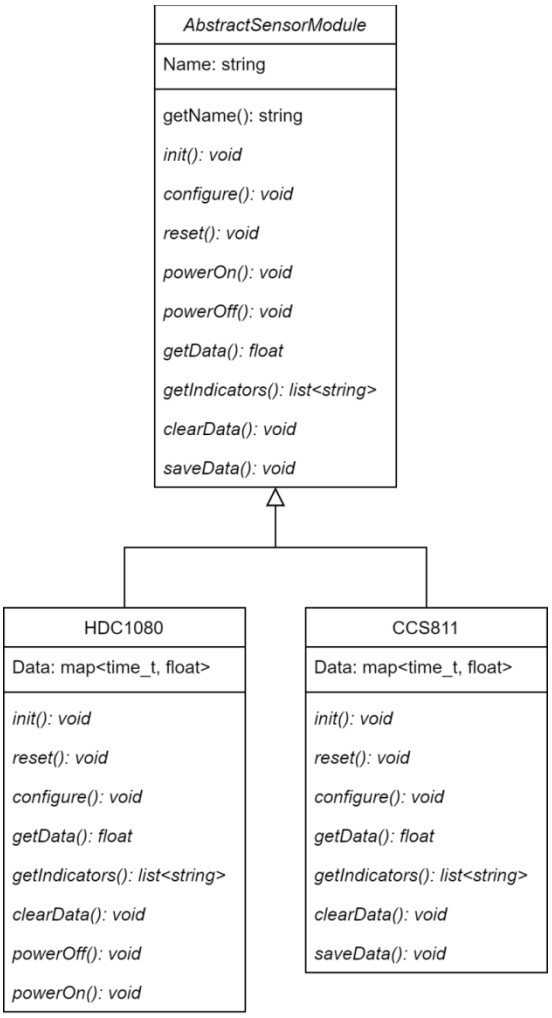


Рис. 3.17. Діаграма класів для сенсорних модулів

Це архітектурне рішення спрощує інтеграцію нових сенсорних модулів у систему та підтримку їхньої єдиної системи управління. Абстрактний клас створює стандартизовану основу, яка визначає очікувану функціональність для кожного модулю, забезпечуючи єдність інтерфейсу для взаємодії з ними.

Кожен новий клас, який успадковує функціональність від абстрактного класу, має здійснювати перевизначення функцій базового класу. Не обов'язково кожен клас повинен перевизначати всі функції, оскільки деякі пристрої можуть бути позбавлені певних можливостей. Наприклад, для мікроконтролера, який не керує живленням сенсора, функції включення або виключення живлення можуть бути непотрібними.

Проте, певні обов'язкові функції, такі як отримання показників із сенсорів та отримання списку доступних показників, завжди повинні бути перевизначені. Це обумовлено тим, що ці функції є прямим призначенням для сенсорних модулів, і їх

перевизначення в нових класах дозволяє забезпечити взаємодію з конкретними пристроями та впроваджувати їхні унікальні особливості.

Під час ініціалізації мікроконтролера, механізм управління сенсорними модулями реєструє всі доступні сенсорні модулі у своєму внутрішньому списку. Надалі, він може звертатись до цього списку для взаємодії як із конкретним сенсорним модулем, так і з усіма модулями одночасно.

Такий підхід визначає механізм як універсальний менеджер для всіх сенсорів, наділяючи його гнучкістю та можливістю ефективно взаємодіяти з будь-яким сенсором у системі. Це забезпечує зручний та централізований спосіб керування та використання різноманітних сенсорів, спрощуючи їхню інтеграцію та управління.

Приклад такого механізму описаний в Додатку А.

3.2.7. Задача періодичного отримання показників з датчиків

Ця завдання існує в рамках механізму управління сенсорними модулями, і його метою є періодичне отримання показників від усіх сенсорних модулів, які зареєстровані в системі.

За замовчуванням завдання запускається після ініціалізації механізму управління сенсорними модулями. Однак воно відразу призупиняється, очікуючи вказівок від сервера щодо тривалості періоду та дозволу на початок виконання. Схему роботи цього завдання можна побачити на рис. 3.18.

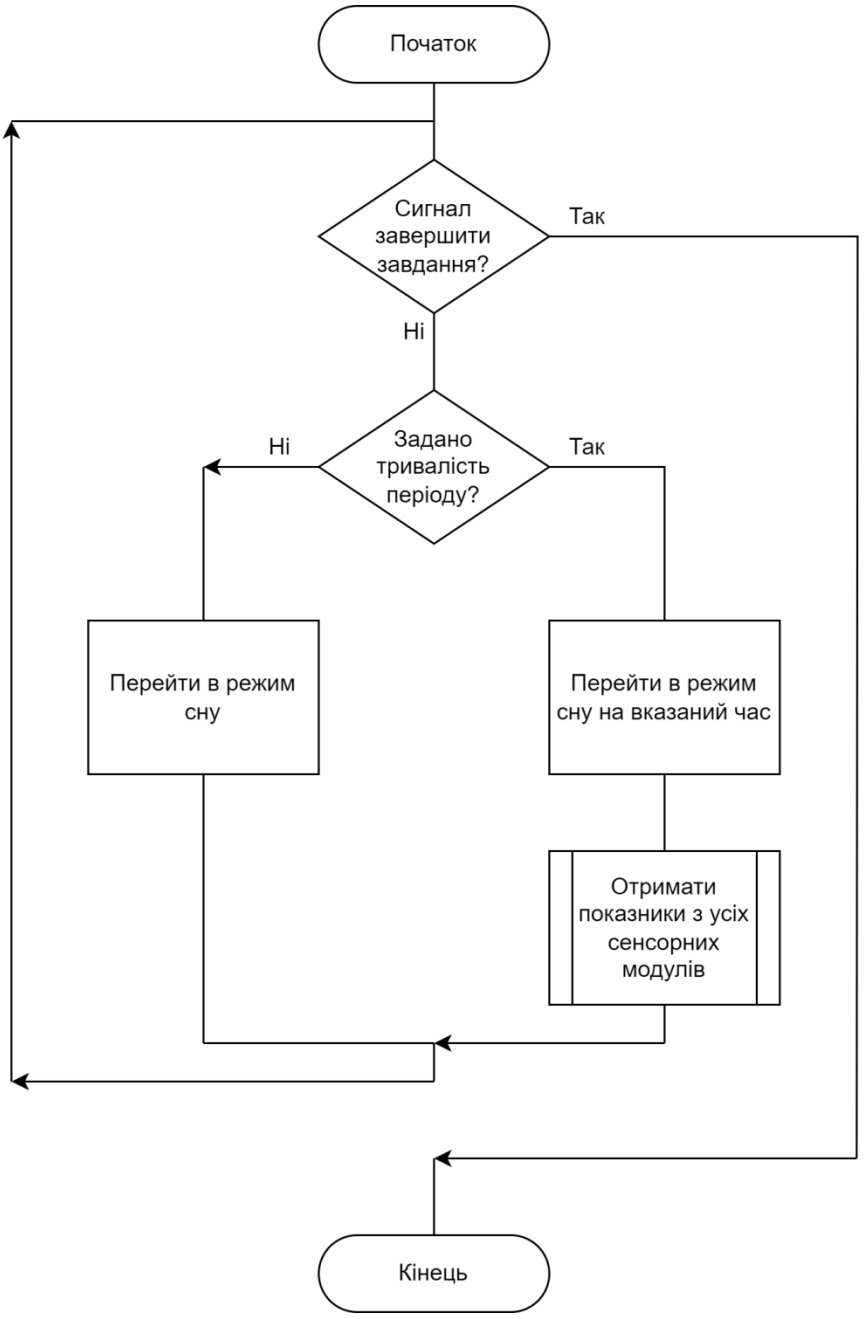


Рис. 3.18. Схема алгоритму задачі отримання даних з сенсорів

Після запуску завдання відбувається перевірка наявності конфігурації зі сторони сервера. Якщо конфігурація відсутня, то завдання переходить в режим очікування (сну), і залишається у цьому стані до моменту, коли сервер надішле відповідні налаштування.

У випадку, якщо була визначена тривалість періоду для отримання показників, тоді завдання переходить в режим очікування на зазначений період, використовуючи

режим сну. Після пробудження викликається відповідна функція механізму управління сенсорами для отримання показників з усіх зареєстрованих сенсорів.

3.3. Структура серверної частини

Опис серверної частини стає трошки складнішим, і його подачу у формі схем, на яких події розгортаються послідовно, ускладнює те, що багато процесів відбувається паралельно чи асинхронно, що є результатом специфіки фреймворку *Qt*.

Програма серверної частини розділена на три ключові компоненти: графічний інтерфейс користувача (*GUI*), *TCP*-сервер та система зберігання даних. Кожна з цих складових виконує свої унікальні функції, взаємодіючи між собою та забезпечуючи високорівневий функціонал серверної частини програми.

Початковим етапом функціонування програми є ініціалізація основного вікна програмного інтерфейсу. Далі відбувається ініціалізація *TCP*-сервера, що включає визначення функцій-обробників для пакетів даних, які надходять з мережі. Для обробки пакетів використовується такий самий механізм, як і в мікроконтролері, що описаний в розділі 3.2.5.

Графічний інтерфейс користувача (*GUI*) був розроблений з використанням засобів *Qt Designer*. У *GUI* реалізовано дерево, яке відображає всі пристрої у мережі, їх датчики та останні зчитані показники. Це вікно надає можливість отримати докладну інформацію про останні дані, зібрані від датчиків. При цьому забезпечується зручний інтерфейс для моніторингу та взаємодії з усіма доступними пристроями та їх даними в мережі.

Для актуалізації інформації *TCP*-сервер регулярно направляє запити до всіх пристроїв у мережі. Принцип роботи такий: пристрої відправляють відповіді на ці запити, і *TCP*-сервер виконує обробку отриманих даних, оновлюючи відповідні віджети у програмі. Детальніша комунікація між сервером та пристроями буде розглянута у розділі 3.4.

Сервер, який отримує дані, накопичує їх в оперативній пам'яті. Регулярно виконується автоматичне збереження отриманих даних у вигляді окремих *CSV*-файлів. Така система зберігання дозволяє уникнути втрати інформації у випадку можливих системних збоїв. Крім того, користувач має можливість зберегти накопичені дані вручну, забезпечуючи додатковий рівень контролю над інформацією.

На рис. 3.19 представлено графічний інтерфейс програми для управління системою. У центральній частині вікна розташований перелік усіх пристроїв, що наразі доступні в системі. Кожен пристрій ідентифікується за його *MAC*-адресою.

Кожен пристрій може містити різні сенсорні модулі, і при розгортанні списку пристрою ви можете переглянути назви цих модулів. Кожен сенсорний модуль обладнаний принаймні одним індикатором з відображеними значеннями, і кількість цих індикаторів може бути різною.

Нижче цієї області розташовані допоміжні кнопки для розгортання та згортання всієї ієрархії. Ці кнопки мають підписи "*Expand all*" та "*Collapse all*".

Як було вказано раніше, автоматичне збереження даних відбувається періодично. Також є можливість вручну зберігати дані, натискаючи кнопку "*Save to CSV*".

Для налаштування періоду збирання даних з сенсорного модулю використовується опція "*Change reading period*". Після натискання цієї кнопки відкривається діалогове вікно (рис. 3.20), де можна вказати бажаний період в секундах та встановити прапорець, що визначає дозвіл на збір даних.

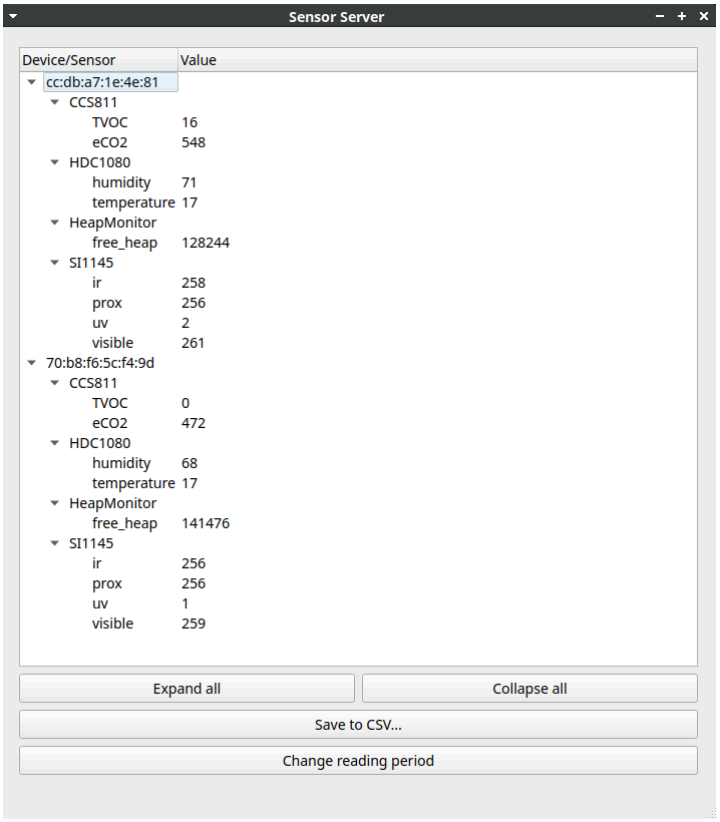


Рис. 3.19. Зовнішній вигляд програми

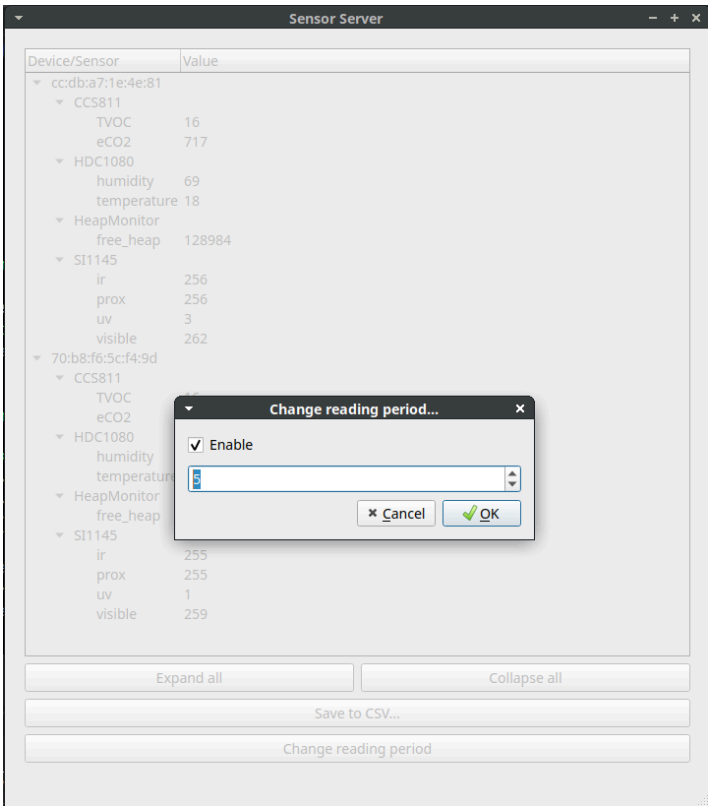


Рис. 3.20. Діалогове вікно для вказання періоду збирання даних з сенсору

Інформація, яка зберігається у *CSV*-файлах, відповідає стандартному формату цього типу файлу. Це текстовий документ, в якому перший рядок містить заголовки для даних, а наступні рядки представляють собою самі дані, розділені комами. Приклад таких даних наведено на рис. 3.21.

Система автоматично збирає дані з пристроїв через *TCP*-сервер, який періодично висилає запити. Мікроконтролер відповідає на ці запити, надсилаючи відповіді, якщо це вимагає сервер. Крім того, *TCP*-сервер обробляє окремі запити від мікроконтролера та надсилає відповіді. Більш докладний опис взаємодії між сервером та пристроями подається у розділі 3.4.

```
time_t, temperature
1701199332,20
1701199333,20
1701199334,20
1701199335,20
1701199336,20
1701199337,20
1701199338,20
1701199339,20
1701199340,20
1701199341,20
1701199342,20
1701199343,20
1701199345,20
1701199346,20
1701199347,20
1701199348,20
1701199349,20
1701199350,20
1701199351,20
1701199352,20
1701199353,20
1701199354,20
1701199355,20
```

Рис. 3.21. Вміст *CSV*-файлу із даними про температуру

3.4. Комунікація між сервером та пристроєм

З метою забезпечення взаємодії між пристроєм та сервером розроблено спеціальні пакети даних та установлено послідовність їх передачі для ефективного обміну інформацією. На рис. 3.22 показано діаграму послідовності обміну пакетів даних між пристроєм та сервером.

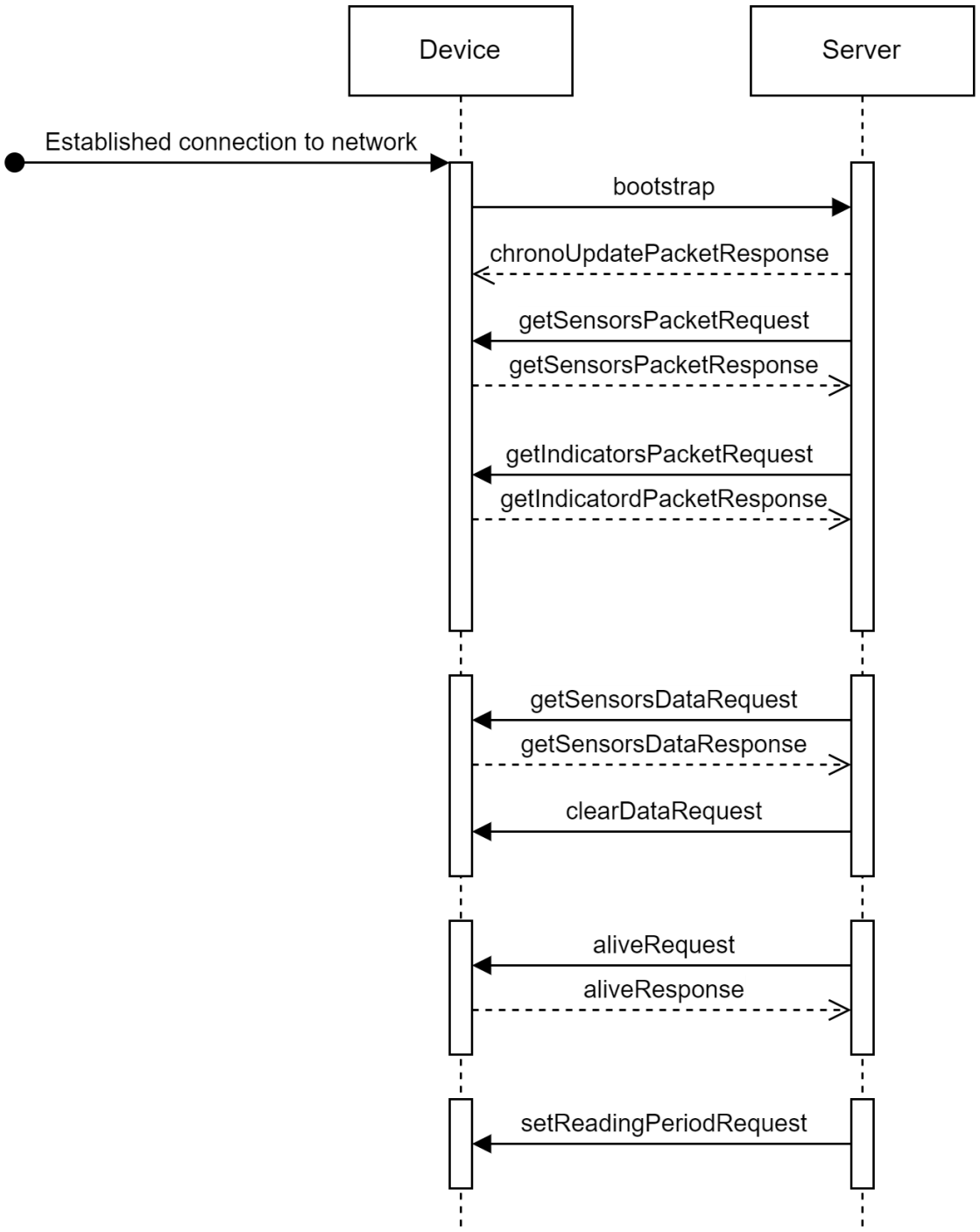


Рис. 3.22. Діаграма послідовності обміну пакетами даних

Після встановлення зв'язку між пристроєм і сервером, пристрій автоматично створює повідомлення "*Bootstrap*" та висилає його на сервер. Цей пакет повідомляє серверу, що даний пристрій присутній в мережі та готовий до подальшої взаємодії.

Сервер, який отримує вказаний пакет, негайно відправляє поточний час пристрою для його уточнення. Передача поточного часу відбувається за допомогою пакету "*ChronoUpdatePacket*". Крім того, сервер розпочинає опитування пристрою для отримання інформації про його сенсорні модулі. Перший запит, який він надсилає, — це "*GetSensorsPacketRequest*". За допомогою цього запиту сервер просить про інформацію щодо назв всіх сенсорних модулів. Пристрій, отримавши цей запит, негайно генерує відповідь, звертаючись до свого менеджера для отримання списку із назвами сенсорів. Відповідь від пристрою потім упаковується в "*GetSensorsPacketResponse*".

Отримавши відповідь із сенсорними модулями, сервер генерує наступний запит, щоб визначити назви індикаторів для кожного конкретного сенсорного модуля. Для цього створюється пакет "*GetIndicatorsPacketRequest*". Наприклад, якщо пристрій має три сенсорні модулі, буде сформовано три пакети "*GetIndicatorsPacketRequest*", кожен із яких міститиме назву відповідного сенсорного модуля. Пристрій, отримавши ці пакети, генерує відповідь "*GetIndicatorsPacketResponse*". Таким чином, якщо пристрій отримує три такі пакети, для сервера буде сформовано три відповіді.

Отримавши відповідь із сенсорними модулями, сервер може додати інформацію про ці модулі до дерева сенсорних модулів. Тепер сервер має достатньо даних для відправки пакету "*GetSensorsDataRequest*", який запитує у пристрою дані, зібрані для певного сенсору та конкретного індикатора. Пристрій намагається упакувати максимально можливу кількість даних, яку він зібрав, і відправляє їх на сервер у пакеті "*GetSensorsDataResponse*". Дані передаються від найстаріших до найновіших у цьому відповіді.

Отримавши дані з пакету "*GetSensorsDataResponse*", сервер включає їх до свого сховища в оперативній пам'яті. Сервер реєструє проміжок часу, під час якого

були зібрані ці дані, і відправляє пристрою новий пакет "*ClearDataRequest*", щоб пристрій міг очистити відповідні дані на своєму боці. Пристрій, отримавши цей пакет, може виконати очищення без відповіді серверу.

У будь-який момент часу сервер може ініціювати відправку двох пакетів до пристрою: "*AliveRequest*" та "*SetReadingPeriodRequest*".

Пакет "*AliveRequest*" періодично висилається до пристрою з метою перевірки його присутності в мережі, що дозволяє визначити активні пристрої. Проте ця взаємодія не завершується, поки пристрій не відповість на запит за допомогою пакету "*AliveResponse*". У випадку, якщо сервер протягом значного часового інтервалу не отримає відповідь у вигляді "*AliveResponse*", він автоматично виключає такий пристрій із списку активних.

Пакет "*SetReadingPeriodRequest*" використовується для встановлення періоду зчитування даних із сенсорів конкретного пристрою. Якщо пристрій не отримує цей пакет після запуску, то зчитування даних відключено, оскільки за замовчуванням регулярне отримання даних вимкнене на мікроконтролері. Важливо відзначити, що пристрій може не відповідати серверу щодо отриманого пакету.

Таким чином, цей механізм автоматично управляє системою для збору та збереження даних із сенсорних модулів. Проте для початку збору даних необхідно встановити період зчитування для кожного пристрою.

3.5. Випробовування програмного засобу

Для оцінки працездатності системи було вирішено провести пробний запуск системи, яка включає два пристрої, сервер та домашній роутер. Схему системи зображено на рис. 3.23.

Кожен пристрій у системі обладнаний трьома сенсорними модулями, які здатні збирати дані щодо різних показників. Серед цих показників можна виділити:

– Викиди *CO2*. Вимірюється в *ppm*(мільйонна частка концентрації) [36].
– Викиди летких органічних сполук(*TVOC*). Вимірюється в *ppb*(мільярдна частка концентрації) [36].

– Температура повітря. Вимірюється в ^oC [37].
– Вологість повітря. Вимірюється у відсотках(%) [37].
– Інтенсивність світла. Вимірюється в люксах(*lux*) [38].
– Ультрафіолетове випромінювання. Вимірюється в $мВт/см^2$ [38].
– Інфрачервоне випромінювання. Вимірюється в $мВт/см^2$ [38].

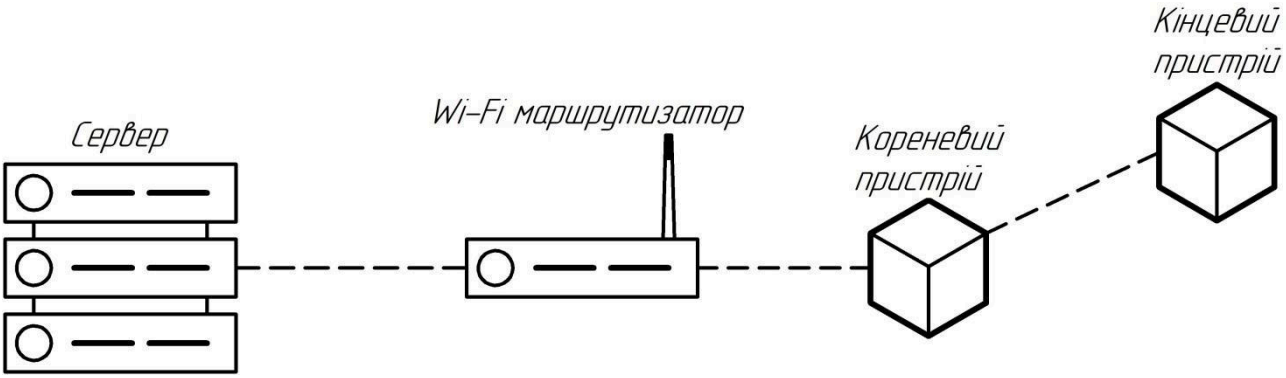


Рис. 3.23. Схематичне зображення випробуваної системи

Також був розроблений додатковий віртуальний сенсорний модуль, призначений для вимірювання кількості вільної пам'яті в кучі. Це відкриває перспективи для створення додаткових віртуальних сенсорних модулів, які можуть визначати стан або системні налаштування мікроконтролера, або виконувати додаткову обробку конкретних даних безпосередньо на пристрої.

У процесі було зібрано два пристрої, кожен обладнаний сенсорними модулями *CCS811*, *HDC1080* та *SI1145*.

Принципова схема налагоджувальної плати відображена на рис. 3.24.

Оскільки розміри налагоджувальної плати перевищують розміри безпаєчних макетних плат, були розроблені та виготовлені власні налагоджувальні плати, які включають слоти для мікроконтролера та сенсорних модулів. Створені налагоджувальні плати представлені на рис. 3.25.

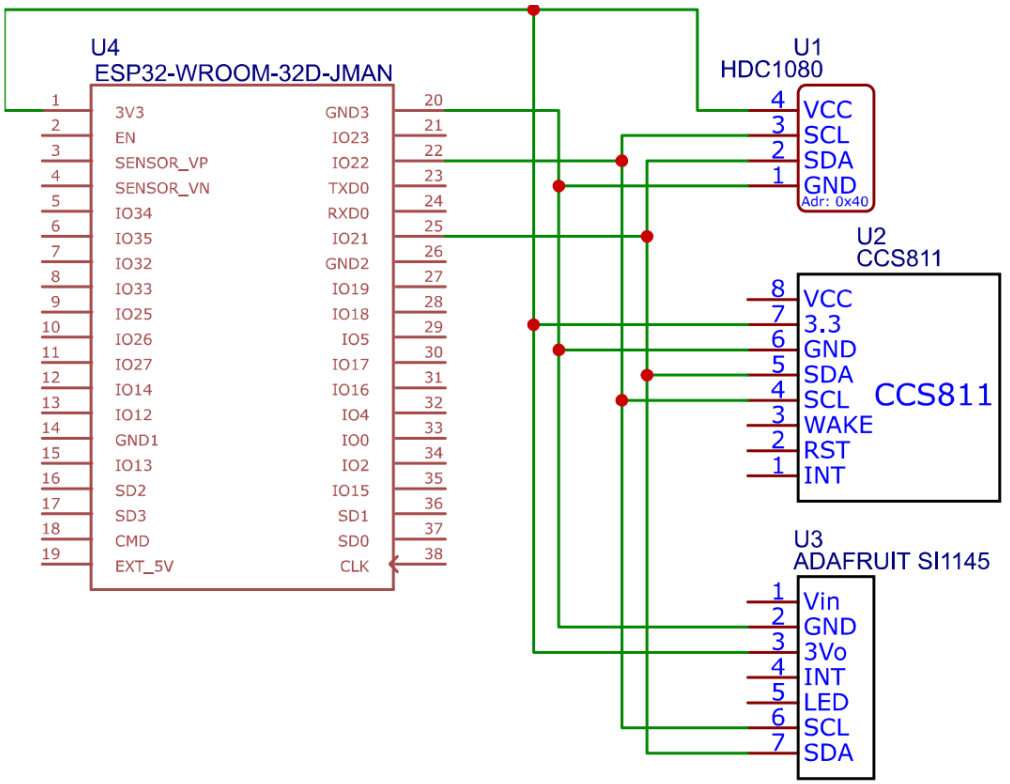


Рис. 3.24. Схема з’єднань пристрою

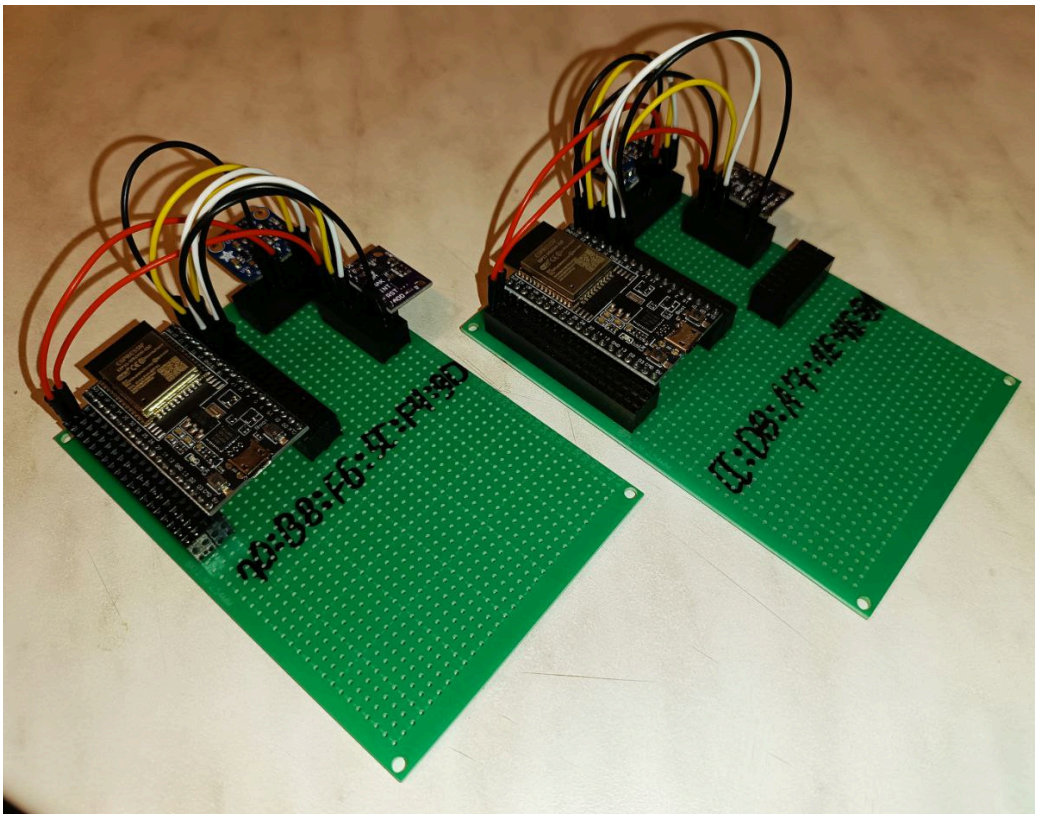


Рис. 3.25. Зовнішній вигляд налагоджувальних плат

Завдяки наявності двох пристроїв можливо провести одночасне вимірювання параметрів у двох різних середовищах. Вирішено здійснити виміри як всередині приміщення, так і за його межами (поза вікном), створюючи два набори даних для порівняння. Заміри виконувались в різні години дня, кожне вимірювання тривало 1 годину, а показники фіксувалися кожні 10 секунд.

У результаті цих вимірювань було отримано графіки, що представлені нижче. Показники отримані всередині приміщення відображені синім кольором, ті що з вулиці – оранжевим.

Спочатку розглянемо графіки показників отриманих всередині дня. Їх зображено на рис. 3.26-3.33.

На рис. 3.26, 3.27 відображено показники пов’язані з якістю повітря. Як видно з графіків показники *CO2* в приміщенні значно більші за вуличні. Показники *TVOC* в приміщенні часто рівні 16, в той час як вуличні рідше стають більшими за 0.

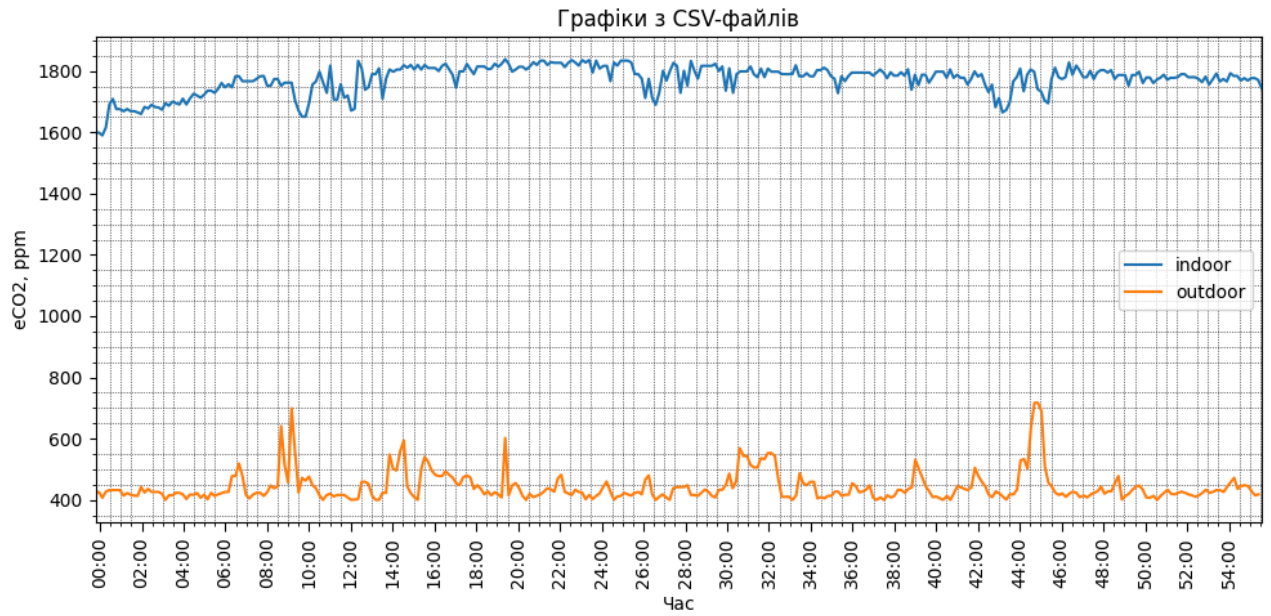


Рис. 3.26. Показники *CO2* в середині дня

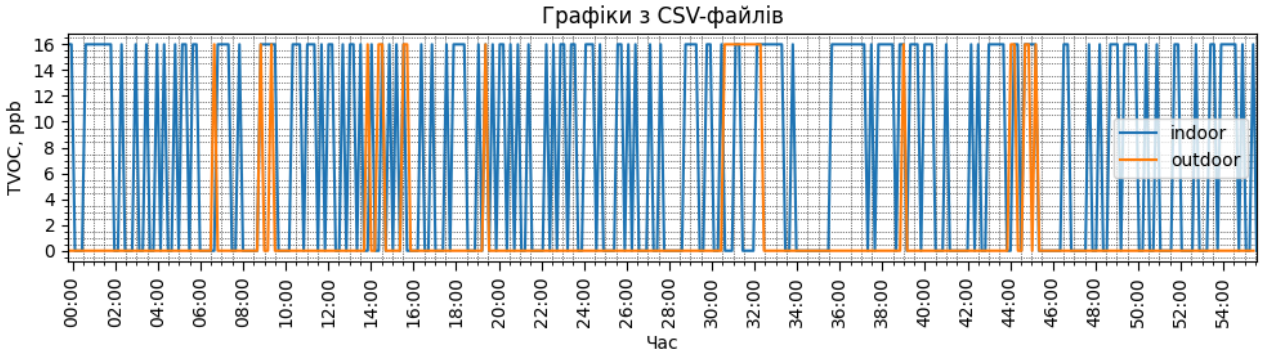


Рис. 3.27. Показники *TVOC* в середині дня

На рис. 3.28, 3.29 відображено показники вологості та температури. Показники вологості між внутрішнім та зовнішнім пристроями відрізняється на 10%. Температура відрізняється набагато сильніше.

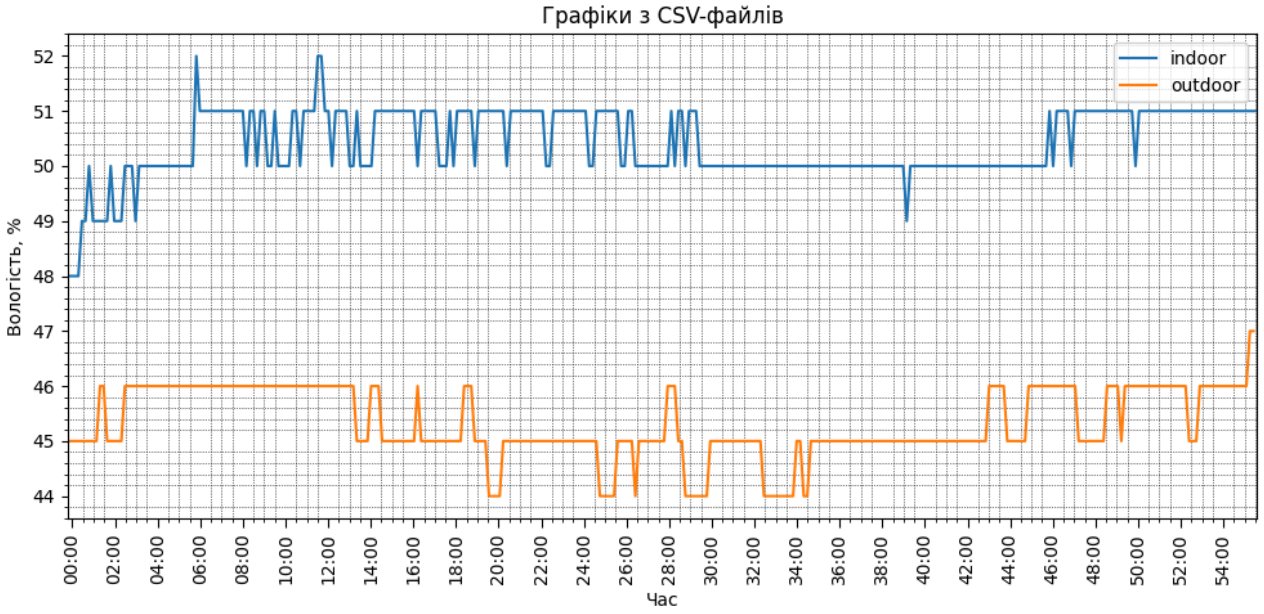


Рис. 3.28. Показники вологості в середині дня

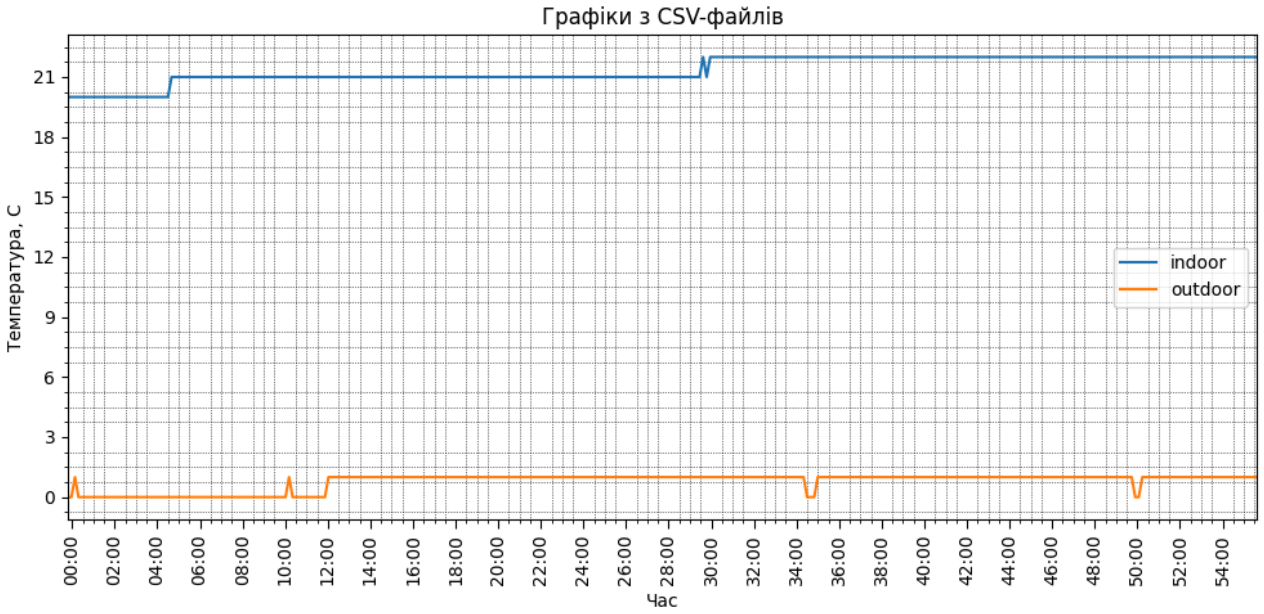


Рис. 3.29. Показники температури в середині дня

На рис. 3.30, 3.31 відображено показники інтенсивності світла та інфрачервоне випромінювання. Різниця між зовнішнім і внутрішнім пристроями присутня, хоч і не висока. Це може бути пов’язане із положенням зовнішнього датчику в тіні і його направлення в сторону вікна.

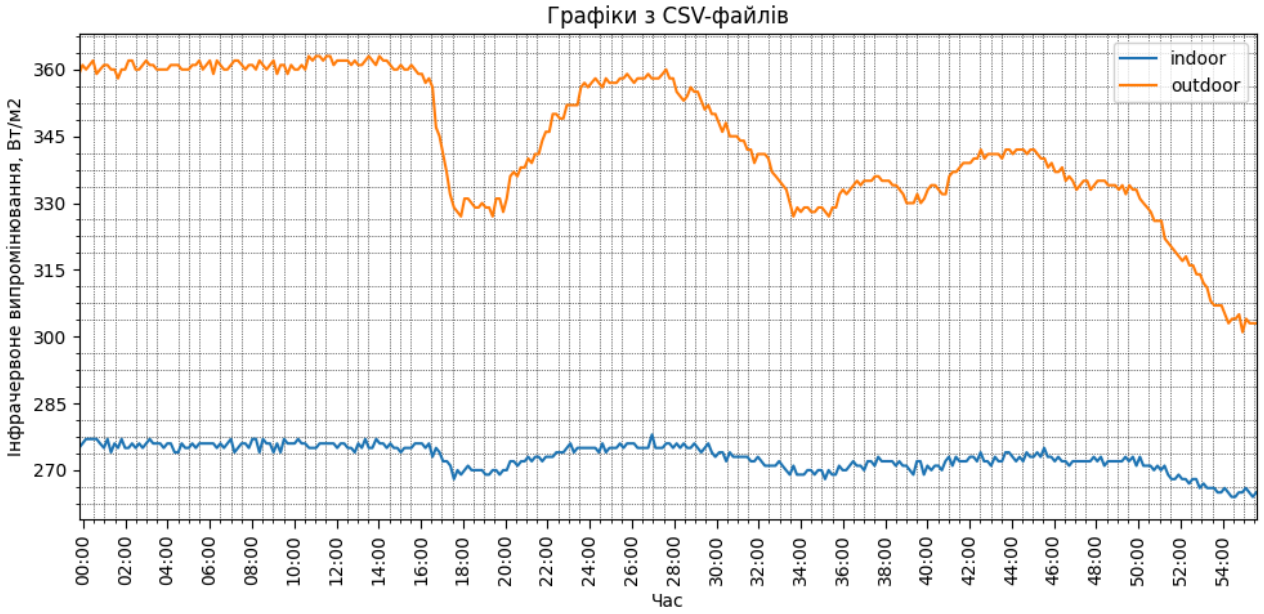


Рис. 3.30. Показники інфрачервоного випромінювання в середині дня

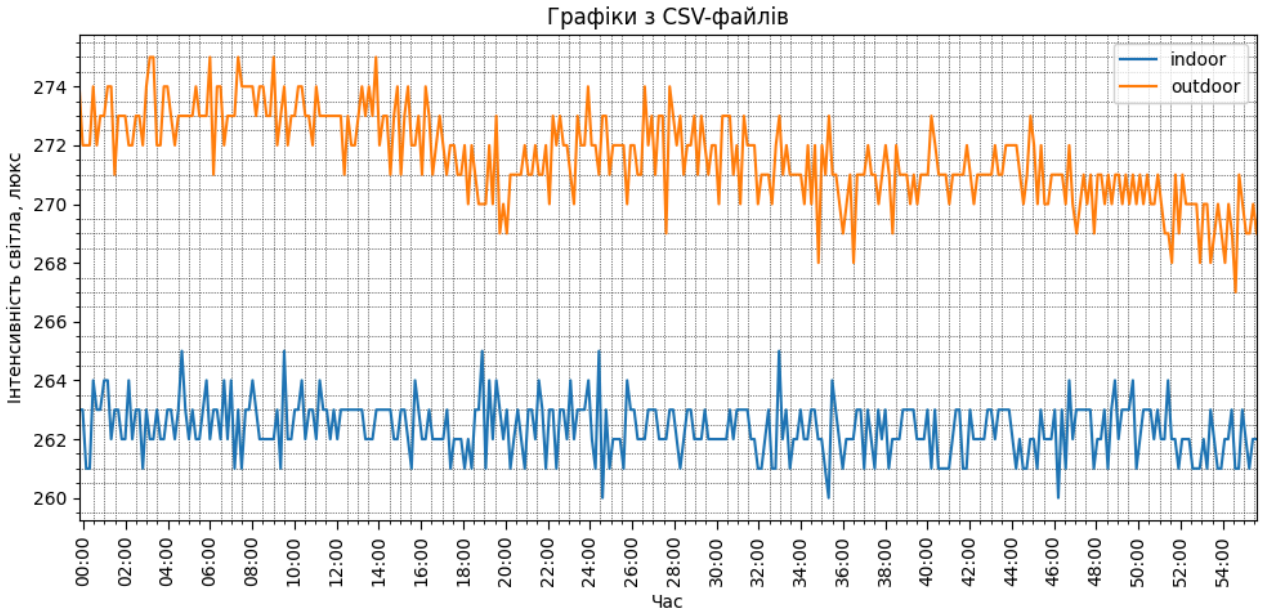


Рис. 3.31. Показники інтенсивності світла в середині дня

На рис. 3.32, 3.33 відображено показники ультрафіолетового випромінювання та кількість вільної пам’яті у кучі. Значення ультрафіолетового випромінювання є нормальним для такого часу. Графік вільної пам’яті демонструє, що всі ресурси в пристрої як отримують пам’ять, так і звільняють її, без витоку пам’яті.

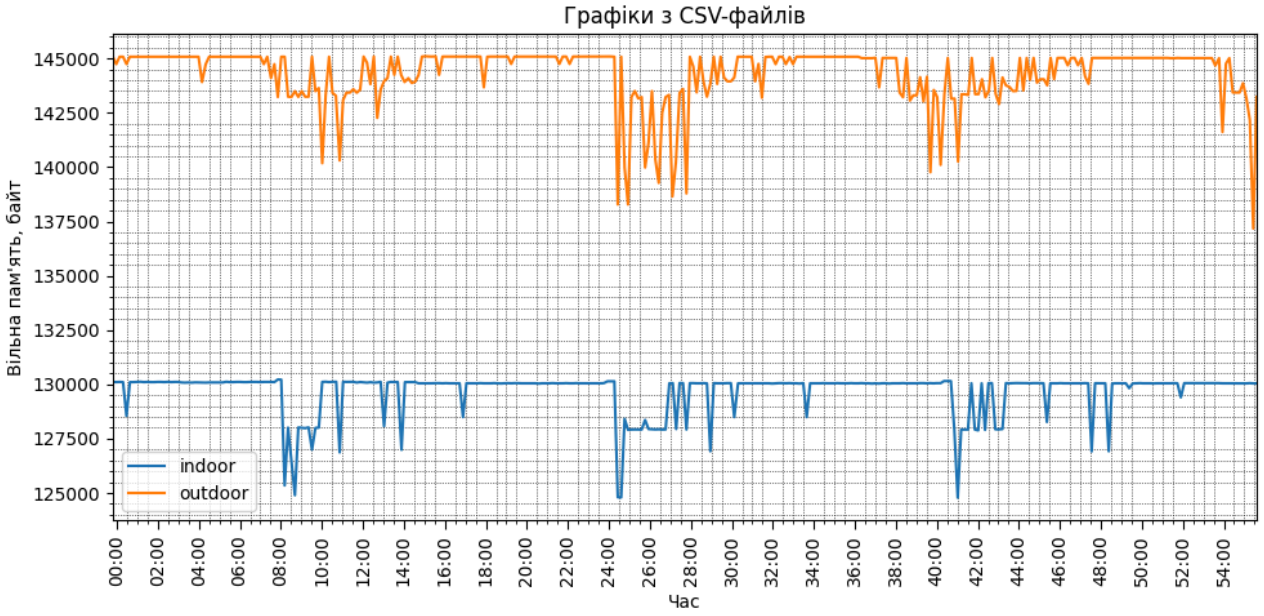


Рис. 3.32. Показники вільної пам’яті мікроконтролера в середині дня

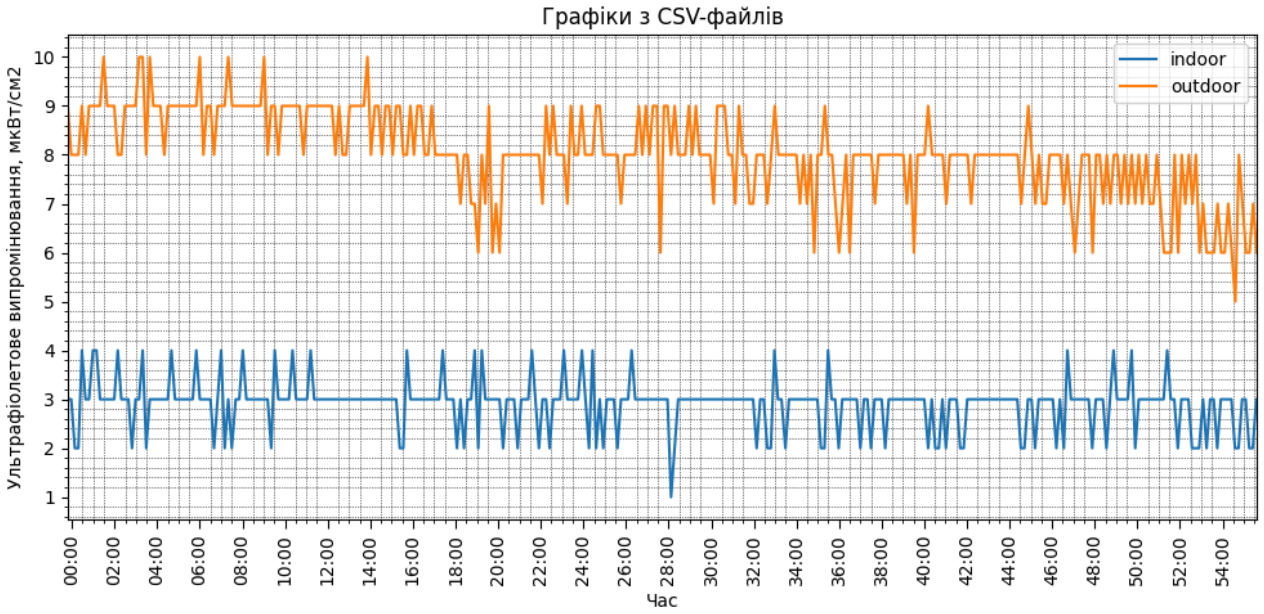


Рис. 3.33. Показники ультрафіолетового випромінювання в середині дня

Розглянемо тепер графіки даних отриманих в нічний час(рис. 3.34-3.41).

На рис. 3.34, 3.35 відображено показники пов’язані з якістю повітря. На графіках помітно суттєву різницю між зовнішніми та внутрішніми показниками.

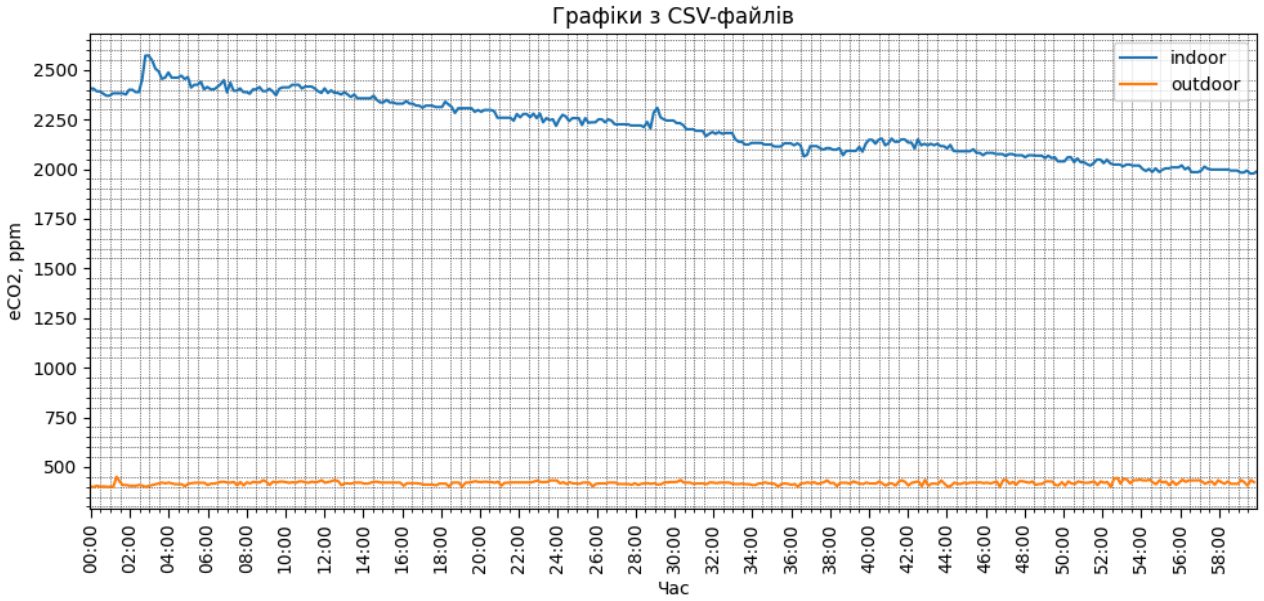


Рис. 3.34. Показники *CO2* в нічний час

Рис. 3.35. Показники *TVOC* в нічний час

На рис. 3.36, 3.37 відображено показники вологості та температури. Різниця вологості між зовнішнім та внутрішнім середовищем досить низька і складає менше 10 %, проте ззовні вона коливається, це пов’язано із опадами. Різниця в температурі лишається статичною.

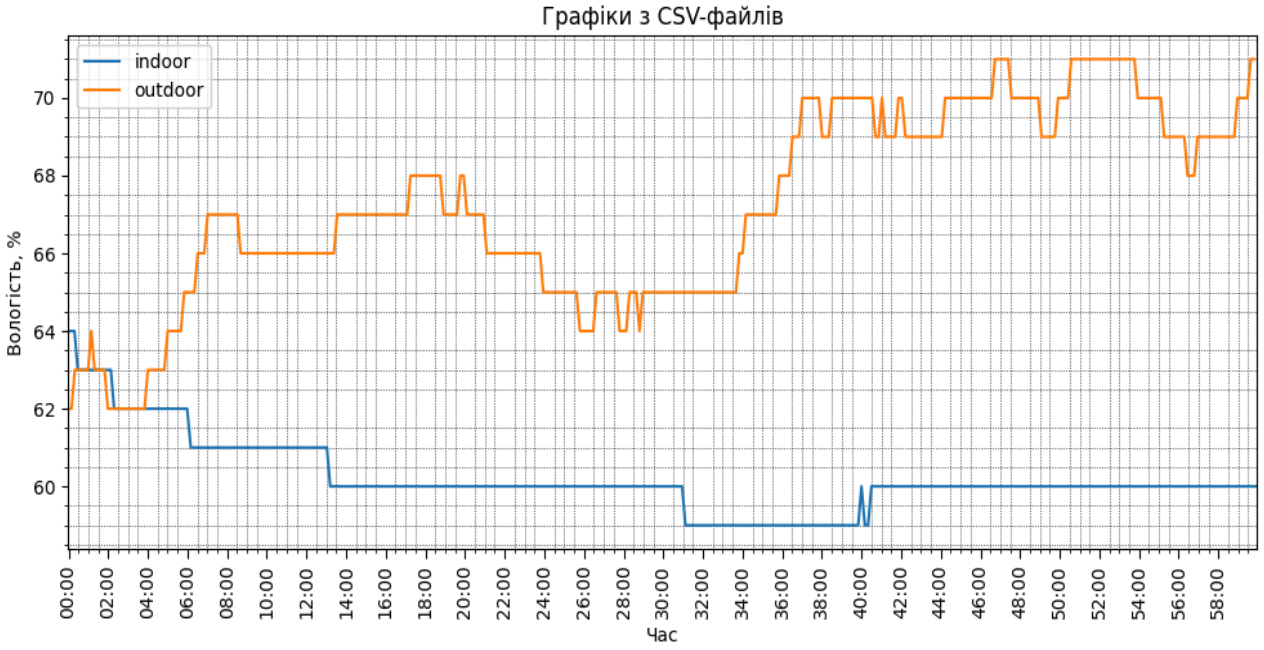


Рис. 3.36. Показники вологості в нічний час

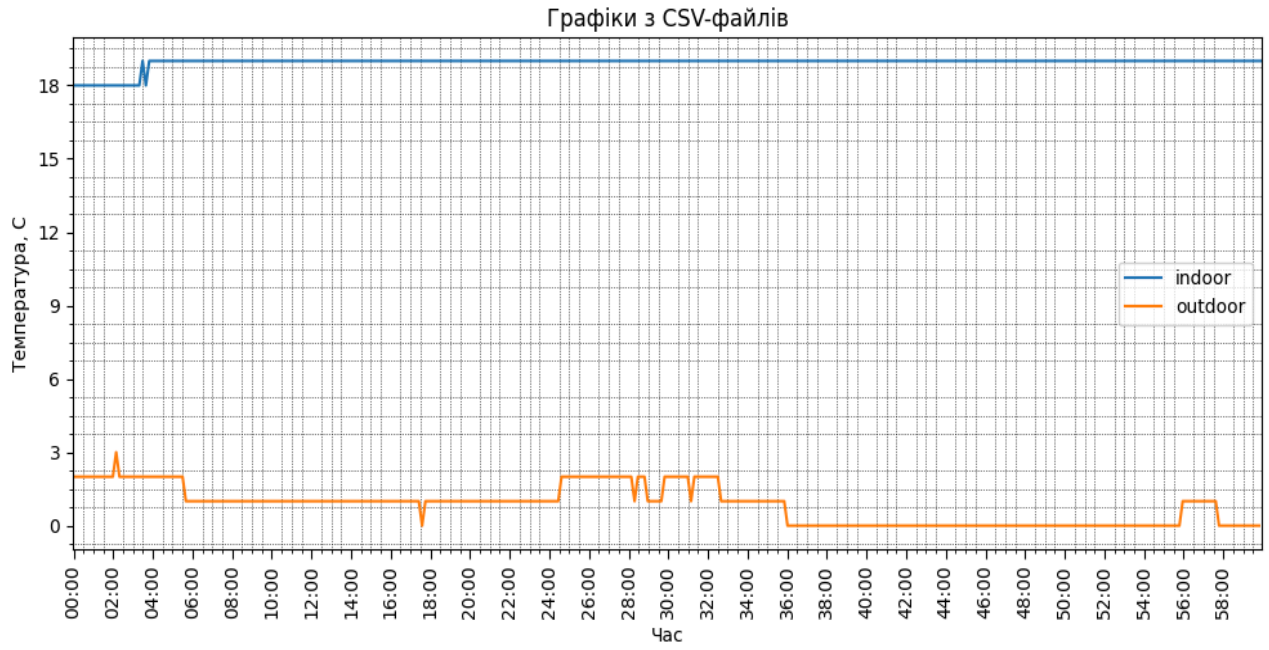


Рис. 3.37. Показники температури в нічний час

На рис. 3.38, 3.39 відображено показники інтенсивності світла та інфрачервоне випромінювання. Оскільки був нічний час то різниці між зовнішніми показниками та внутрішніми майже немає.

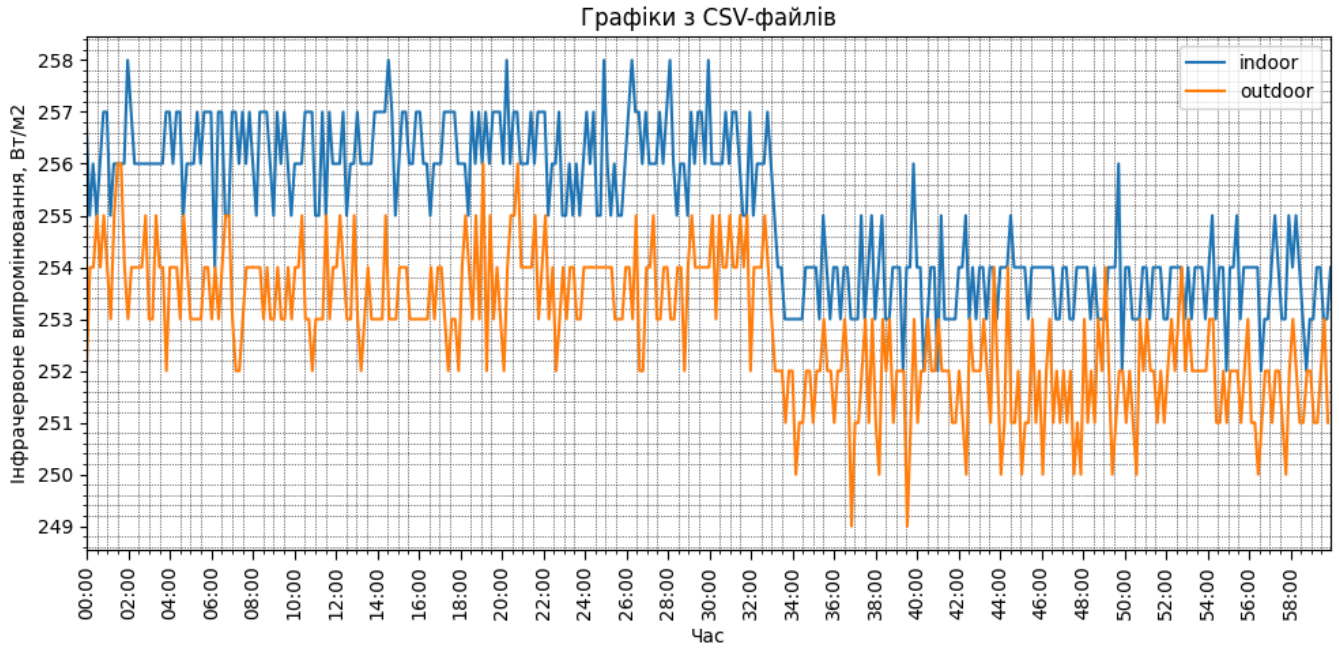


Рис. 3.38. Показники інфрачервоного випромінювання в нічний час

Рис. 3.39. Показники інтенсивності світла в нічний час

На рис. 3.40, 3.41 відображено показники ультрафіолетового випромінювання та кількість вільної пам’яті у кучі. Значення ультрафіолетового випромінювання ззовні також є приблизно рівним внутрішнім. Графік вільної пам’яті коливається так само як і в денний час, але помітно що ресурси все рівно звільняються і не створюють витік пам’яті.

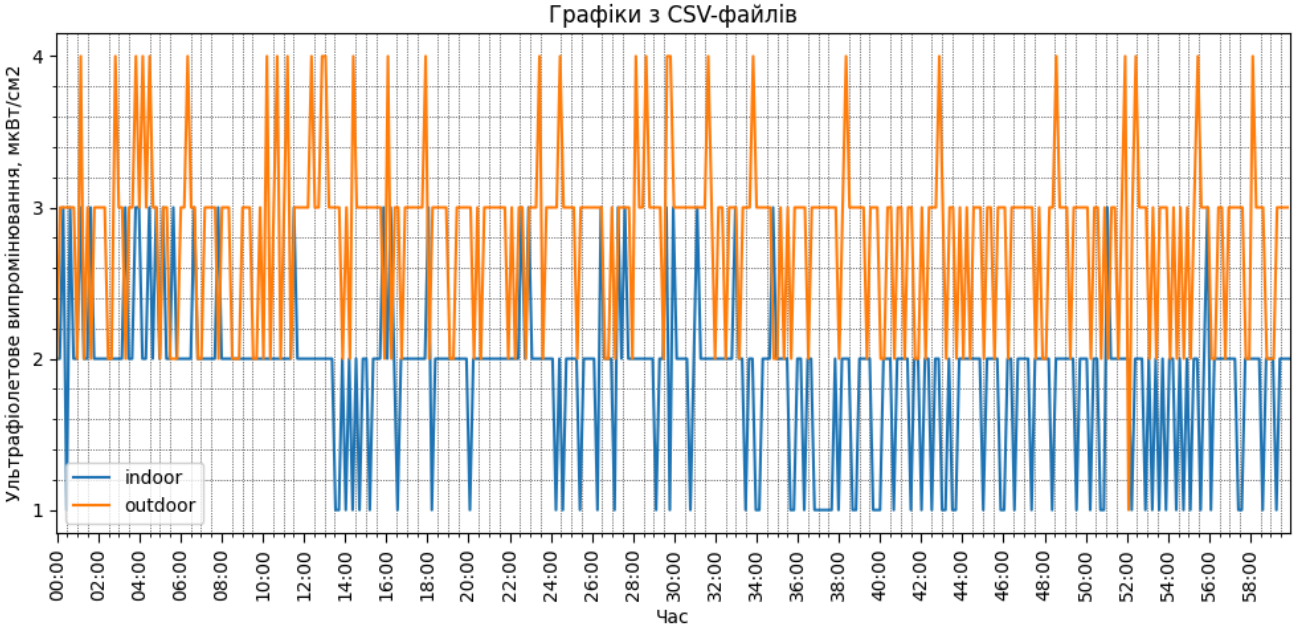


Рис. 3.40. Показники ультрафіолетового випромінювання в нічний час

Графіки були побудовані на основі *CSV*-файлів, які містять дані, отримані з реальних сенсорних модулів. Здійснюючи візуальний аналіз цих графіків, можна

зробити висновок, що програмний код для сенсорів реалізований коректно, оскільки відображає значення як у денний, так і у нічний періоди. Отримані дані з сенсорних модулів становлять цінний матеріал для подальшого аналізу та обробки, спрямованого на виявлення можливих аномалій, вивчення закономірностей чи встановлення залежностей між показниками та іншими аспектами. Цей підхід може допомогти в отриманні інформації та вдосконаленні функціональності системи.

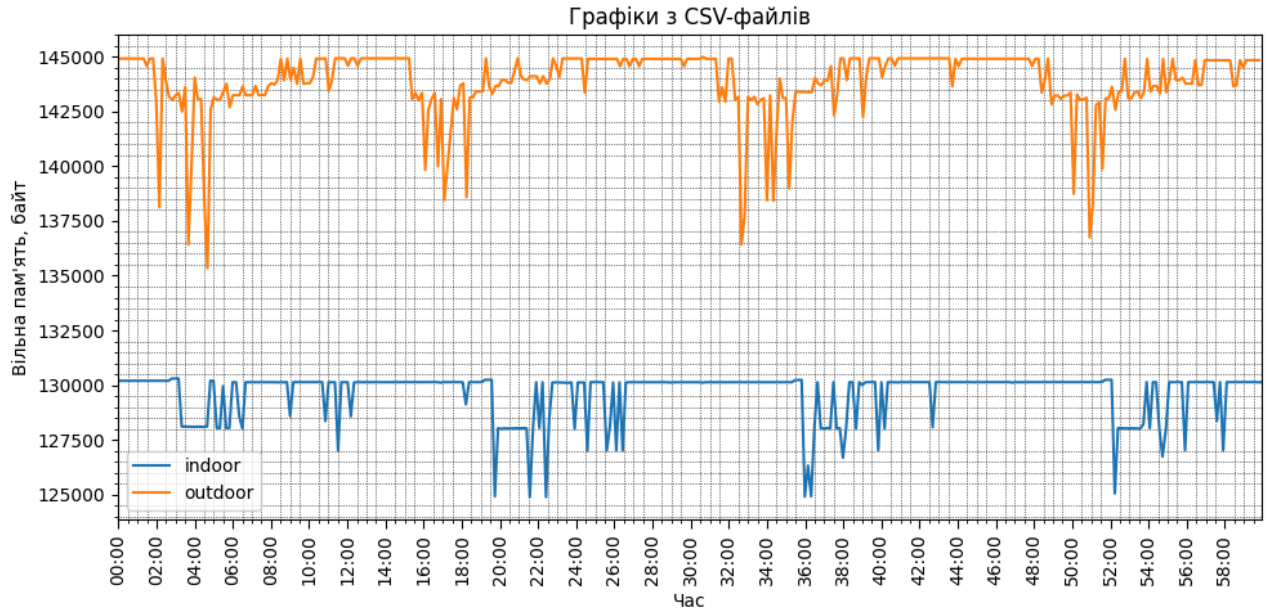


Рис. 3.41. Показники вільної пам’яті мікроконтролера в нічний час

Під час отримання даних із сенсорних модулів слід зазначити, що жоден пристрій не виявив ознак перезавантаження, і не було зафіксовано непереробленних помилок. Аналіз графіків вільної пам’яті вказує на виникнення "сплесків" виділення пам’яті, проте важливо відзначити, що ці ресурси з часом повертаються пристрою без будь-яких витоків пам’яті.

Можливі причини "сплесків" виділення пам’яті можуть бути пов'язані з призупиненням виконання певної задачі, яка взяла певний обсяг пам’яті із кучі, але ще не встигла її повернути, оскільки в той час виконувалась інша задача. Це не представляє собою критичної ситуації, оскільки в подальшому пам’ять звільняється.

Важливо відзначити відсутність суттєвих відмінностей між графіками вільної пам’яті в денний та нічний час, що свідчить про стабільну роботу пристрою.

Також було проведено дослідження радіусу дії пристроїв. На платі мікроконтролера є вбудована антена. Ця антена дуже зручна при розробці та налагоджуванні, оскільки є дуже маленькою, але вона не є потужної і для використання в реальних умовах бажано використовувати зовнішню. Проте навіть вбудована антена дозволяє пристроям підтримувати зв’язок між собою на відстані понад 80 метрів, за умови прямої видимості.

3.6. Висновки до розділу

У цьому розділі було розглянуто структуру мереж, що використовуються в системі для збирання та зберігання даних із розподілених сенсорних модулів. Система складається з серверу, кореневого пристрою та багатьох рядових пристроїв. Фактично розподілена сенсорна система складається з двох частин.

У першій частині знаходиться сервер та кореневий пристрій. Між цими пристроями виконується обмін даних за допомогою стеку протоколів *TCP/IP*. Зокрема між цими пристроями встановлюється з’єднання на основі протоколу *TCP*.

У другій частині знаходяться кореневий пристрій та рядові пристрої. Між цими пристроями для обміну даних використовується протокол *ESP-WIFI-MESH*.

Кореневий пристрій є мостом між цими двома частинами.

Вся комунікація між сервером та пристроями виконується із використанням безпровідної мережі *Wi-Fi*.

Програмний засіб для мікроконтролера умовно поділено на ініціалізацію, обробку подій мережі, обробка пакетів даних від сервера та інших пристроїв. Для забезпечення постійного з’єднання із сервером, кореневий пристрій має задачу для контролю з’єднання, що завжди виконується паралельно з іншими.

Для забезпечення обробки пакетів, що надходять з інших пристроїв чи сервера використовуються паралельна задача для обробки даних, що надійшли із

MESH-мережі. Також кореневий пристрій має ще одну додаткову задачу для обробки даних, що надійшли з мережі *TCP/IP*.

Для керування сенсорними модулями використовується спеціальний механізм. Цей механізм уніфікує всі сенсорні модулі і при роботі з ними не бачить різниці між кожним з ним. За допомогою цього менеджера можна працювати з кожним сенсорним модулем окремо або з усіма разом. Цей механізм також має свою власну задачу, що виконується паралельно. Головне завдання цієї задачі – періодичне отримання даних із усіх сенсорних модулів.

Для сенсорних модулів було визначено абстрактний клас, від якого кожен такий модуль успадковується. Це полегшує керування сенсорним модулем.

Керування пристроями виконується за допомогою серверної програми. Для отримання показників із пристроїв потрібно лише визначити для кожного пристрою період зчитування даних. Приєднання до системи, отримання інформації про пристрій, отримання даних виконується автоматично.

Дані, що накопичує сервер автоматично зберігаються на накопичувачі у *CSV*-файли. Також якщо потрібно, їх можна зберегти вручну.

Взаємодія між сервером та пристроєм є чіткою і послідовною. Детальніше про обмін пакетами прописано у Розділі 3.4.

Програмний засіб було протестовано на реальних пристроях, використовуючи кілька сенсорних модулів. Отримані дані були використані для створення графіків, які були піддані візуальному аналізу. Під час тестування не було виявлено проблем або перезавантажень пристроїв. Сенсорні дані регулярно та надійно отримувались від пристроїв за допомогою серверів.

Таким чином у цьому розділі було описано ключові особливості роботи програмного пристрою для збирання та зберігання даних із розподіленої сенсорної системи та описано результати випробувань програмного модулю на тестовій системі.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення програмного засобу для збирання та зберігання даних із розподілених сенсорних модулів. Для досягнення поставленої мети було досліджено існуючі засоби. На основі отриманої інформації було визначено основні особливості розподілених сенсорних систем та визначено, використання безпровідних мереж буде дуже зручною технологією, оскільки пристрої системи будуть мати можливість зміни фізичного положення в просторі, ремонт системи буде простішим і вартість системи є дешевшою, оскільки є відсутні фізичні провідники, якими могла бути з'єднана система.

У Розділі 1 було визначено загальні положення про розподілені системи, розглянуто різні типи розподілених систем. Також визначено існуючі структури розподілених систем. Всі ці знання можна також застосувати до розподілених сенсорних систем.

Також у Розділі 1 було розглянуто існуючі розподілені сенсорні системи та як вони працюють. Зокрема було розглянуті системи відеоспостереження, IoT(Інтернет речей), системи моніторингу здоров'я, контролю за дорожнім рухом, моніторингу навколишнього здоров'я, віртуальної реальності(*VR*).

Було оцінено способи обміну даних між компонентами системи. Зокрема було визначено, що для розподіленої сенсорної системи провідний зв'язок є менш зручним, оскільки вони є більш дорогими та складнішими в експлуатації, більше того можливість переміщення сенсорних модулів в системі є проблемним процесом.

Застосування безпровідного зв'язку дозволяє пристроям змінювати положення в просторі без втрати зв'язку. Відсутність кабелів в системі полегшує ремонтні та діагностичні роботи. Використання безпровідного зв'язку дозволяє простіше додавати пристрої до системи.

Було розглянуто безпровідні технології *Bluetooth*, *ZigBee* та *Wi-Fi*. Розглянуто переваги та недоліки кожної із технологій. Технології *Bluetooth* та *ZigBee* мають досить низьку пропускну здатність та менший радіус дії, порівняно із технологією *Wi-Fi*. Для комунікації між пристроями в мережі було обрано технологію *Wi-Fi.*

У Розділі 2 було розглянуто існуючі мови програмування для вбудованих систем та для десктопних рішень. Обрано мови *C/C++* для створення програмного коду, це дозволяє створити баланс між високою швидкодією коду та швидкістю розробки.

Кожен пристрій системи містить складається із мікроконтролера та сенсорних модулів. Кандидатами на роль мікроконтролера були *STM32F407, ESP32, Arduino Uno Rev3*. Було обрано мікроконтролер *ESP32* через більшу продуктивність та наявність вбудованого радіомодуля, що полегшує роботу з ним.

Створення програмного коду для мікроконтролерів *ESP32* виконується із використанням фреймворку *ESP-IDF*. Цей фреймворк написаний мовою *C*, але він чудово адаптований для використання в мові *C++*. Поєднання *ESP-IDF* та *C++* пришвидшить швидкість розробки програмного коду та надасть нові можливості для створення більш гнучкої програмної архітектури прошивки.

Для створення серверної частини системи було обрано фреймворк *Qt*. Цей фреймворк написаний мовою *C++*. Фреймворк містить багато вже готових класів для створення графічного інтерфейсу та роботи із мережею, що значно полегшує створення програмного коду.

Qt Creator – це *IDE*, що використовується для створення проектів із використанням фреймворку *Qt*. Цю *IDE* було обрано для роботи із *Qt*, оскільки він є інтегрований в *Qt Creator*.

Для створення програмного коду для мікроконтролерів використовується *Visual Studio Code*. Це досить швидкий та зручний засіб для розробки програмного забезпечення. *IDE* має просту систему для підключення додаткових плагінів. Існує спеціальний плагін, що дозволяє легко додати інструменти для роботи з фреймворком *ESP-IDF*.

У Розділі 2 було розглянуто, проаналізовано та обрано інструменти для створення програмного коду для мікроконтролера та серверної частини системи.

У Розділі 3 було розглянуто структуру мереж, що використовуються в системі для збирання та зберігання даних із розподілених сенсорних модулів. Система

складається з серверу, кореневого пристрою та багатьох рядових пристроїв. Фактично розподілена сенсорна система складається з двох частин.

У першій частині знаходиться сервер та кореневий пристрій. Між цими пристроями виконується обмін даних за допомогою стеку протоколів *TCP/IP*. Зокрема між цими пристроями встановлюється з'єднання на основі протоколу *TCP*.

У другій частині знаходяться кореневий пристрій та рядові пристрої. Між цими пристроями для обміну даних використовується протокол *ESP-WIFI-MESH*.

Кореневий пристрій є мостом між цими двома частинами.

Вся комунікація між сервером та пристроями виконується із використанням безпровідної мережі *Wi-Fi*.

Програмний засіб для мікроконтролера умовно поділено на ініціалізацію, обробку подій мережі, обробка пакетів даних від сервера та інших пристроїв. Для забезпечення постійного з'єднання із сервером, кореневий пристрій має задачу для контролю з'єднання, що завжди виконується паралельно з іншими.

Для забезпечення обробки пакетів, що надходять з інших пристроїв чи сервера використовуються паралельна задача для обробки даних, що надійшли із *MESH*-мережі. Також кореневий пристрій має ще одну додаткову задачу для обробки даних, що надійшли з мережі *TCP/IP*.

Для керування сенсорними модулями використовується спеціальний механізм. Цей механізм уніфікує всі сенсорні модулі і при роботі з ними не бачить різниці між кожним з ним. За допомогою цього менеджера можна працювати з кожним сенсорним модулем окремо або з усіма разом. Цей механізм також має свою власну задачу, що виконується паралельно. Головне завдання цієї задачі – періодичне отримання даних із усіх сенсорних модулів.

Для сенсорних модулів було визначено абстрактний клас, від якого кожен такий модуль успадковується. Це полегшує керування сенсорним модулем.

Керування пристроями виконується за допомогою серверної програми. Для отримання показників із пристроїв потрібно лише визначити для кожного пристрою період зчитування даних. Приєднання до системи, отримання інформації про пристрій, отримання даних виконується автоматично.

Дані, що накопичує сервер автоматично зберігаються на накопичувачі у *CSV*-файли. Також якщо потрібно, їх можна зберегти вручну.

Взаємодія між сервером та пристроєм є чіткою і послідовною. Детальніше про обмін пакетами прописано у розділі 3.4.

Програмний засіб було протестовано на реальних пристроях, використовуючи кілька сенсорних модулів. Отримані дані були використані для створення графіків, які були піддані візуальному аналізу. Під час тестування не було виявлено проблем або перезавантажень пристроїв. Сенсорні дані регулярно та надійно отримувались від пристроїв за допомогою серверів.

Було також проведено роботу із визначення радіусу дії пристроїв, при яких вони здатні утворювати систему і відповідно забезпечувати доступ до мережі. В ході дослідження налагоджувальних пристроїв було визначено, що на відстані 80 метрів між двома пристроями забезпечується стабільний зв’язок за умови прямої видимості та з використанням вбудованої антени. Подальше збільшення відстані між пристроями було неможливе через брак простору в середовищі де тестувалась система.

Таким чином у цьому розділі було описано ключові особливості роботи програмного пристрою для збирання та зберігання даних із розподіленої сенсорної системи та описано результати випробувань програмного модулю на тестовій системі.

В результаті виконання кваліфікаційної роботи було отримано програмний засіб для керування розподіленою сенсорною системою. Головним завданням такої системи є збирання та накопичення даних із розподілених сенсорних модулів.

Для додавання нового пристрою в систему достатньо лише створити прошивку із використанням певних сенсорних модулів та завантажити її в мікроконтролер, з’єднати сенсорні модулі з мікроконтролером і встановити у зону досяжності до мережі новий пристрій.

Створена система здатна розширюватись до 1000 пристроїв в одній такій системі. Проте це обмеження також можна подолати, визначивши протокол комунікації між різними такими системами.

Відстань між пристроями, за якої пристрої здатні комунікувати між собою є 80 метрів і більше, при використанні вбудованої антени на мікроконтролері.

Використовуючи таку систему можна створити покриття більше ніж в 6,5 $км^2$. Використання підсилювачів, допоміжних антен чи напрямлених антен дозволить збільшити радіус дії кожного пристрою і збільшити площу, що здатна покрити система.

Результатом роботи системи є файлі із даними. Приклад таких даних зображено на рис. 3.21.

Створений програмний засіб рекомендується до використання у реальних умовах. Також варто зазначити, що система підходить для використання із рухомими об’єктами. Оскільки здатна динамічно перебудовуватись під час роботи.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ГОСТ 7.11–2004 «Система стандартів по інформації, бібліотечній та видавничій справі. Бібліографічний запис. Скорочення слів та словосполучень на іноземних та європейських мовах»
2. ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання»;
3. ДСТУ 3582: 2013 «Бібліографічний опис скорочення слів і словосполучень в українській мові»;
4. ДСТУ 8302-2015 «Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання».
5. Розподілені системи. [Електронний ресурс] – (Дата звернення: 04.09.2023). Режим доступу: *http://surl.li/gxndp*
6. Багатомашинні системи. [Електронний ресурс] – (Дата звернення: 06.09.2023). Режим доступу:*http://surl.li/nvhms*
7. *Distributed Systems Explained.* [Електронний ресурс] – (Дата звернення: 07.09.2023). Режим доступу:*http://surl.li/nvhnn*
8. Розподілені системи. [Електронний ресурс] – (Дата звернення: 07.09.2023). Режим доступу:*http://surl.li/nvhno*
9. Що таке розподілена система. [Електронний ресурс] – (Дата звернення: 07.09.2023). Режим доступу: *https://uk.theastrologypage.com/distributed-system*
10. *Features of Distributed Operating System*. [Електронний ресурс] – (Дата звернення: 08.09.2023). Режим доступу: *https://www.geeksforgeeks.org/features-of-distributed-operating-system/*
11. *Characteristics of Distributed System*. [Електронний ресурс] – (Дата звернення: 09.09.2023). Режим доступу: *https://www.codingninjas.com/studio/library/characteristics-of-distributed-system*
12. Структури розподілених систем керування. [Електронний ресурс] – (Дата звернення: 16.09.2023). Режим доступу: *https://studfile.net/preview/4600808/*

13. Що входить до комплекту камер відеоспостереження. [Електронний ресурс] – (Дата звернення: 17.09.2023). Режим доступу: *http://surl.li/nvhow*

14. Налаштування *Apple Watch* і створення пари з *iPhone*. [Електронний ресурс] – (Дата звернення: 17.09.2023). Режим доступу: *https://support.apple.com/uk-ua/guide/watch/apdde4d6f98e/watchos*

15. Virtual Reality. [Електронний ресурс] – (Дата звернення: 17.09.2023). Режим доступу: *http://surl.li/qoif*

16. Модуль *Bluetooth JDY*-31 *BT*4.0. [Електронний ресурс] – (Дата звернення: 16.09.2023). Режим доступу: *https://arduino.ua/prod4311-bluetooth-modyl-jdy-30-spp-bk3231*

17. Bluetooth модуль *HC*-06. [Електронний ресурс] – (Дата звернення: 17.09.2023). Режим доступу: *https://arduino.ua/prod241-bluetooth-modyl-hc-06*

18. *ZigBee* модулі. [Електронний ресурс] – (Дата звернення: 16.09.2023). Режим доступу: *http://surl.li/nvhon*

19. *Wi-Fi*. [Електронний ресурс] – (Дата звернення: 18.09.2023). Режим доступу: *http://surl.li/blfxn*

20. *ESP-WIFI-MESH*. [Електронний ресурс] – (Дата звернення: 18.09.2023). Режим доступу: *http://surl.li/nvhpe*

21. Performance Evaluation of C/C++, MicroPython and Rust. [Електронний ресурс] – (Дата звернення: 07.10.2023). Режим доступу: *https://www.mdpi.com/2079-9292/12/1/143*

22. *Assembly language*. [Електронний ресурс] – (Дата звернення: 10.10.2023). Режим доступу: *http://surl.li/nvhqa*

23. *Python Vs. C++ Differences In 2023*. [Електронний ресурс] – (Дата звернення: 10.10.2023). Режим доступу: *https://lanexcorp.com/python-vs-c-differences-in-2023/*

24. *Python vs C++ | Top 16 Differences Between C++ and Python*. [Електронний ресурс] – (Дата звернення: 10.10.2023). Режим доступу: *https://www.softwaretestinghelp.com/python-vs-cpp/*

25. *ESP32*. [Електронний ресурс] – (Дата звернення: 12.10.2023). Режим доступу: *http://surl.li/nvhqh*
26. *ESP32 Wi-Fi and Bluetooth SoC*. [Електронний ресурс] – (Дата звернення: 12.10.2023). Режим доступу: *https://www.espressif.com/en/products/socs/esp32*
27. *The Internet of Things with ESP32*. [Електронний ресурс] – (Дата звернення: 12.10.2023). Режим доступу: *http://esp32.net/*
28. *STM32F407/417 - STMicroelectronics*. [Електронний ресурс] – (Дата звернення: 13.10.2023). Режим доступу: *https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html*
29. *STM32F417xx*. [Електронний ресурс] – (Дата звернення: 13.10.2023). Режим доступу: *https://www.st.com/resource/en/datasheet/stm32f415rg.pdf*
30. *Arduino Uno Rev3 Arduino Official Store*. [Електронний ресурс] – (Дата звернення: 13.10.2023). Режим доступу: *https://store.arduino.cc/products/arduino-uno-rev3*
31. *Qt (software)*. [Електронний ресурс] – (Дата звернення: 14.10.2023). Режим доступу: *http://surl.li/nvhqs*
32. *Espressif IoT Development Framework — PlatformIO v6.1 documentation*. [Електронний ресурс] – (Дата звернення: 16.10.2023). *https://docs.platformio.org/en/stable/frameworks/espidf.html*
33. *ESP IoT Development Framework | Espressif Systems*. [Електронний ресурс] – (Дата звернення: 16.10.2023). Режим доступу: *https://www.espressif.com/en/products/sdks/esp-idf*
34. *Embedded Software Development Tools & Cross Platform IDE | Qt Creator*. [Електронний ресурс] – (Дата звернення: 14.10.2023). Режим доступу: *https://www.qt.io/product/development-tools.*
35. *Visual Studio Code - Code Editing*. [Електронний ресурс] – (Дата звернення: 16.10.2023). Режим доступу: *https://code.visualstudio.com/*
36. *CCS811 datasheet*. [Електронний ресурс] – (Дата звернення: 08.11.2023). Режим доступу: *https://arduino.ua/files/CCS811_Datasheet-DS000459.pdf*

37. *HDC1080 datasheet*. [Електронний ресурс] – (Дата звернення: 08.11.2023). Режим доступу: *https://pdf1.alldatasheet.com/datasheet-pdf/view/813671/TI1/HDC1080.html*

38. *SI1145 datasheet*. [Електронний ресурс] – (Дата звернення: 08.11.2023). Режим доступу: *https://cdn-shop.adafruit.com/datasheets/Si1145-46-47.pdf*

Програмний код механізму для роботи з сенсорними модулями

Файл *SensorManager.h:*

```
#ifndef SENSOR_MANAGER_H
#define SENSOR_MANAGER_H

#include "SensorModules/BaseSensorModule.h"

#include <map>
#include <string>
#include <list>
#include <climits>

#define DEFAULT_READING_PERIOD 1

class SensorManager
{
public:
    static SensorManager &getInstance(void);

    static void addSensor(BaseSensorModule *sensor);
    static BaseSensorModule *getSensor(const std::string &name);

    static void powerOnAll();
    static void powerOffAll();

    static void resetAll();
    static void readDataAll();

    static void getSensors(std::list<std::string> &list);
    static void getSensorHeader(const std::string &sensorName,
std::map<std::string, uint8_t> &header);
    static void getSensorData(const std::string &sensorName, const std::string
&dataName, std::vector<time_t> &dataTime, std::vector<uint8_t> &data, const
time_t from = 0, const time_t to = INT_LEAST32_MAX);
```

```
    static void getReadingPeriod(time_t &period);
    static void setReadingPeriod(const time_t &period);

private:
    static constexpr const char *moduleTag = "SensorManager";

    SensorManager();
    SensorManager(const SensorManager &) = delete;
    SensorManager &operator=(const SensorManager &) = delete;

    static void readingTask(void *arg);

    std::map<std::string, BaseSensorModule *> m_sensors;
    time_t m_readingPeriod = DEFAULT_READING_PERIOD;
};

#endif // SENSOR_MANAGER_H
```

Файл *SensorManager.cpp:*

```
#include "SensorManager.h"

#include <esp_err.h>
#include <esp_log.h>

#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

#define MAX_DATA_PACKET_SIZE 1000
#define READING_TASK_NAME "reading_task"

SensorManager::SensorManager()
{
    xTaskCreate(SensorManager::readingTask, "reading_task", 8192, NULL, 5,
NULL);
    ESP_LOGI(moduleTag, "initialized");
}
```

```
void SensorManager::readingTask(void * /*arg*/)
{
   vTaskDelay(pdMS_TO_TICKS(10 * 1000));

   while (true)
   {

vTaskDelay(pdMS_TO_TICKS(SensorManager::getInstance().m_readingPeriod *
1000));
      SensorManager::readDataAll();
   }
}

SensorManager &SensorManager::getInstance(void)
{
   static SensorManager object;
   return object;
}

void SensorManager::addSensor(BaseSensorModule *sensor)
{
   if (sensor == nullptr)
   {
      ESP_LOGE(moduleTag, "didn't added the sensor", );
      return;
   }

   ESP_LOGI(moduleTag, "add sensor %s", sensor->getName().c_str());

   SensorManager &manager = SensorManager::getInstance();
   manager.m_sensors[sensor->getName()] = sensor;
}

BaseSensorModule *SensorManager::getSensor(const std::string &name)
{
   SensorManager &manager = SensorManager::getInstance();
   if (manager.m_sensors.find(name) == manager.m_sensors.end())
      return nullptr;
```

```
    return manager.m_sensors[name];
}

void SensorManager::powerOnAll()
{
    SensorManager &manager = SensorManager::getInstance();

    for (auto &it : manager.m_sensors)
        it.second->powerOn();
}

void SensorManager::powerOffAll()
{
    SensorManager &manager = SensorManager::getInstance();

    for (auto &it : manager.m_sensors)
        it.second->powerOff();
}

void SensorManager::resetAll()
{
    SensorManager &manager = SensorManager::getInstance();


 for (auto &it : manager.m_sensors)
        it.second->reset();
}

void SensorManager::readDataAll()
{
    SensorManager &manager = SensorManager::getInstance();

    for (auto &it : manager.m_sensors)
        it.second->readData();
}
```

```
void SensorManager::getSensors(std::list<std::string> &list)
{
   list.clear();

   SensorManager &manager = SensorManager::getInstance();

   for (auto &it : manager.m_sensors)
      list.push_back(it.first);
}

void SensorManager::getSensorHeader(const std::string &sensorName,
std::map<std::string, uint8_t> &header)
{
   header.clear();

   SensorManager &manager = SensorManager::getInstance();
   if (manager.m_sensors.find(sensorName) == manager.m_sensors.end())
      return;

   manager.m_sensors[sensorName]->getHeader(header);
}

void SensorManager::getReadingPeriod(time_t &period)
{
   period = SensorManager::getInstance().m_readingPeriod;
}

void SensorManager::getSensorData(const std::string &sensorName, const
std::string &dataName,
                        std::vector<time_t> &dataTime, std::vector<uint8_t> &data,
                        const time_t from, const time_t to)
{
   dataTime.clear();
   data.clear();

   SensorManager &manager = SensorManager::getInstance();
   if (manager.m_sensors.find(sensorName) == manager.m_sensors.end())
      return;
```

```
    uint8_t singleDataSize;
    manager.m_sensors[sensorName]->getSingleDataSize(dataName,
singleDataSize);
    if (singleDataSize == 0)
        return;

    unsigned int dataCount = MAX_DATA_PACKET_SIZE / singleDataSize;
    manager.m_sensors[sensorName]->getData(dataName, dataTime, data,
dataCount, from, to);
}

void SensorManager::setReadingPeriod(const time_t &period)
{
    SensorManager::getInstance().m_readingPeriod = period;
}
```

Файл *BaseSensorModule.h:*

```
#ifndef BASE_SENSOR_MODULE_H
#define BASE_SENSOR_MODULE_H

#include <string>
#include <vector>
#include <map>

#include <climits>

class BaseSensorModule
{
public:
    BaseSensorModule(const std::string &name);

    const std::string &getName() const;

    virtual void configure();
    virtual void reset();

    virtual void powerOn();
```

```
    virtual void powerOff();

    virtual void readData() = 0;

    virtual void getHeader(std::map<std::string, uint8_t> &header) = 0;
    virtual void getData(const std::string &dataName,
                         std::vector<time_t> &dataTime, std::vector<uint8_t> &data,
                         unsigned int count,
                         const time_t from = 0, const time_t to = INT_LEAST32_MAX) = 0;
    virtual void getSingleDataSize(const std::string &dataName, uint8_t &size) = 0;

    virtual void clearData(const std::string &dataName, const time_t from = 0, const
time_t to = -1);

protected:
    std::string m_name;
};

#endif // BASE_SENSOR_MODULE_H
```

Файл *BaseSensorModule.cpp:*

```
#include "BaseSensorModule.h"
#include <esp_log.h>

BaseSensorModule::BaseSensorModule(const std::string &name) : m_name(name)
{}

const std::string &BaseSensorModule::getName() const
{
    return m_name;
}

void BaseSensorModule::configure()
{
    ESP_LOGW(this->getName().c_str(), "configure() not implemented.");
}
```

```
void BaseSensorModule::reset()
{
   ESP_LOGW(this->getName().c_str(), "reset() not implemented.");
}
void BaseSensorModule::powerOn()
{
   ESP_LOGW(this->getName().c_str(), "powerOn() not implemented.");
}
void BaseSensorModule::powerOff()
{
   ESP_LOGW(this->getName().c_str(), "powerOff() not implemented.");
}

void BaseSensorModule::clearData(const std::string & /*dataName*/, const time_t
/*from*/, const time_t /*to*/)
{
   ESP_LOGW(this->getName().c_str(), "clearData() not implemented.");
}
```

Файл *HDC1080.h:*

```
#ifndef HDC1080_H
#define HDC1080_H

#include "../BaseSensorModule.h"

#include <map>

#define HUMIDITY_STR "humidity"
#define TEMPERATURE_STR "temperature"

#define TEMPERATURE_MIN (-40)
#define TEMPERATURE_MAX (125)
#define HUMIDITY_MIN (0)
#define HUMIDITY_MAX (100)

#define HDC1080_ADDR 0x40
```

```
enum HDC1080_register
{
   HDC1080_REG_TEMPERATURE = 0x00,
   HDC1080_REG_HUMIDITY = 0x01,
   HDC1080_REG_CONFIG = 0x02
};

class HDC1080 : public BaseSensorModule
{
public:
   HDC1080();

   void readData() override;

   void getHeader(std::map<std::string, uint8_t> &header) override;
   void getData(const std::string &dataName,
            std::vector<time_t> &dataTime, std::vector<uint8_t> &data,
            unsigned int count,
            const time_t from = 0, const time_t to = INT_LEAST32_MAX) override;
   void getSingleDataSize(const std::string &dataName, uint8_t &size) override;

   void clearData(const std::string &dataName, const time_t from = 0, const time_t
to = INT_LEAST32_MAX);

private:
   void read_ui16(const uint8_t reg, uint16_t &out);
   void write_ui16(const uint8_t reg, const uint16_t value);
   void triggerMeasure(const uint8_t reg);

   void readTemperature(double &temperature);
   void readHumidity(double &humidity);

   void limitValueToRange(double &value, double range1, double range2);

   void getDataHumidity(std::vector<time_t> &dataTime, std::vector<uint8_t>
&data,
                  unsigned int count, const time_t from = 0, const time_t to =
INT_LEAST32_MAX);
```

```
    void getDataTemperature(std::vector<time_t> &dataTime, std::vector<uint8_t>
&data,
                            unsigned int count, const time_t from = 0, const time_t to =
INT_LEAST32_MAX);

    std::map<time_t, uint16_t> m_humidity;
    std::map<time_t, uint16_t> m_temperature;
};

#endif // HDC1080_H
```

Файл *HDC1080.cpp*:

```
#include "HDC1080.h"
#include "TimeManager.h"
#include "I2C_peripheral.h"
#include <cmath>

#include <esp_log.h>

#define HDC1080_I2C_DELAY_FOR_SINGLE_WRITING 1000

static constexpr const double HUMIDITY_MULTIPLIER = 0.0015258789;
static constexpr const double TEMPERATURE_MULTIPLIER = 0.00251770019;
static constexpr const int TEMPERATURE_OFFSET = 40;

HDC1080::HDC1080() : BaseSensorModule("HDC1080")
{
    I2C_peripheral::init();

    write_ui16(HDC1080_register::HDC1080_REG_CONFIG, 0x10);
}

void HDC1080::readData()
{
    double temperature;
    readTemperature(temperature);
```

```
    double humidity;
    readHumidity(humidity);

    time_t currentTime = TimeManager::getTime();
    m_temperature[currentTime] = temperature;
    m_humidity[currentTime] = humidity;

    ESP_LOGI(getName().c_str(), "Time: %llu, humidity: %f, temperature: %f",
            currentTime, humidity, temperature);
}

void HDC1080::readTemperature(double &temperature)
{
    // Start measuring and wait until it will be done
    triggerMeasure(HDC1080_register::HDC1080_REG_TEMPERATURE);
    vTaskDelay(pdMS_TO_TICKS(10));

    uint16_t raw;
    read_ui16(HDC1080_register::HDC1080_REG_TEMPERATURE, raw);

    temperature = (((double)raw) / pow(2, 16)) * (TEMPERATURE_MAX -
TEMPERATURE_MIN) + TEMPERATURE_MIN;
    limitValueToRange(temperature, TEMPERATURE_MIN,
TEMPERATURE_MAX);
}

void HDC1080::readHumidity(double &humidity)
{
    // Start measuring and wait until it will be done
    triggerMeasure(HDC1080_register::HDC1080_REG_HUMIDITY);
    vTaskDelay(pdMS_TO_TICKS(10));

    uint16_t raw;
    read_ui16(HDC1080_register::HDC1080_REG_HUMIDITY, raw);

    humidity = (double)raw * HUMIDITY_MULTIPLIER;
    limitValueToRange(humidity, HUMIDITY_MIN, HUMIDITY_MAX);
}
```

```
void HDC1080::read_ui16(const uint8_t reg, uint16_t &out)
{
   std::vector<uint8_t> vecIn = {reg};
   std::vector<uint8_t> vecOut(sizeof(uint16_t));

   I2C_peripheral::write(HDC1080_ADDR, vecIn,
HDC1080_I2C_DELAY_FOR_SINGLE_WRITING);
   vTaskDelay(pdMS_TO_TICKS(15));
   I2C_peripheral::read(HDC1080_ADDR, vecOut,
HDC1080_I2C_DELAY_FOR_SINGLE_WRITING);

   out = ((uint16_t)vecOut[0] << 8) | ((uint16_t)vecOut[1]);
}

void HDC1080::triggerMeasure(const uint8_t reg)
{
   std::vector<uint8_t> vecIn = {reg};
   I2C_peripheral::write(HDC1080_ADDR, vecIn,
HDC1080_I2C_DELAY_FOR_SINGLE_WRITING);
}

void HDC1080::write_ui16(const uint8_t reg, const uint16_t value)
{
   uint8_t MSB_value = value >> 8;
   uint8_t LSB_value = value & 0xFF;

   std::vector<uint8_t> vecIn = {reg, MSB_value, LSB_value};

   I2C_peripheral::write(HDC1080_ADDR, vecIn,
HDC1080_I2C_DELAY_FOR_SINGLE_WRITING);
   vTaskDelay(pdMS_TO_TICKS(15));
}

void HDC1080::limitValueToRange(double &value, double range1, double range2)
{
   if (range1 > range2)
      std::swap(range1, range2);

   if (value < range1)
```

```
        value = range1;
    else if (value > range2)
        value = range2;
}

void HDC1080::getHeader(std::map<std::string, uint8_t> &header)
{
    header[HUMIDITY_STR] = sizeof(uint16_t);
    header[TEMPERATURE_STR] = sizeof(uint16_t);
}

void HDC1080::getData(const std::string &dataName,
                      std::vector<time_t> &dataTime, std::vector<uint8_t> &data,
                      unsigned int count, const time_t from, const time_t to)
{
    if (dataName == HUMIDITY_STR)
        getDataHumidity(dataTime, data, count, from, to);
    else if (dataName == TEMPERATURE_STR)
        getDataTemperature(dataTime, data, count, from, to);
}
void HDC1080::getSingleDataSize(const std::string &dataName, uint8_t &size)
{
    std::map<std::string, uint8_t> header;
    getHeader(header);

    if (header.find(dataName) == header.end())
    {
        size = 0;
        return;
    }

    size = sizeof(time_t) + header[dataName];
}

void HDC1080::getDataHumidity(std::vector<time_t> &dataTime,
std::vector<uint8_t> &data,
                              unsigned int count, const time_t from, const time_t to)
{
    for (auto &it : m_humidity)
```

```
    {
        if (count == 0)
            break;

        if (it.first < from || it.first > to)
            continue;

        dataTime.push_back(it.first);
        data.push_back(it.second & 0xFF);
        data.push_back(it.second >> 8);

        count--;
    }
}

void HDC1080::getDataTemperature(std::vector<time_t> &dataTime,
std::vector<uint8_t> &data,
                        unsigned int count, const time_t from, const time_t to)
{
    for (auto &it : m_temperature)
    {
        if (count == 0)
            break;

        if (it.first < from || it.first > to)
            continue;


        dataTime.push_back(it.first);
        data.push_back(it.second & 0xFF);
        data.push_back(it.second >> 8);

        count--;
    }
}

void HDC1080::clearData(const std::string &dataName, const time_t from, const
time_t to)
{
```

```
    std::map<time_t, uint16_t> *ptrData = nullptr;

    if (dataName == HUMIDITY_STR)
        ptrData = &m_humidity;
    else if (dataName == TEMPERATURE_STR)
        ptrData = &m_temperature;

    if (ptrData == nullptr)
        return;

    for (auto it = ptrData->begin(); it != ptrData->end();)
    {
        if (it->first >= from && it->first <= to)
            it = ptrData->erase(it);
        else
            ++it;
    }
}
```