Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

КОМПЛЕКСНА БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 2. Програмне забезпечення»

08-54. КБКР.039.00.000 ПЗ

Виконав: студент 4 курсу, групи 1КІ-21б
спеціальності 123 — Комп'ютерна інженерія
освітня програма — «Комп'ютерна інженерія»
Довгань В. В

Керівник: к.т. н., доц. кафедри ОТ
Богомолов С. В.

Рецензент: к.фіз.-мат. н., доц. кафедри МБіС
Шиян А. А.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
10.06 2025 року

Вінниця ВНТУ — 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 – Інформаційні технології
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф._____Азаров О.Д.
«21» березня 2025 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Довганю Владиславу Володимировичу

1 Тема роботи: Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 3. Програмне забезпечення, керівник роботи Богомолов Сергій Віталійович, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Термін подання студентом роботи 13.05.2025

3 Вихідні дані до роботи: схема підключення датчиків BMP28, BH1750 та MQ-135, бібліотеки та функції обробки даних з датчиків, бібліотеки та функції для забезпечення Wi-Fi — з'єднання, середовище розробки — Arduino IDE.

4 Зміст розрахунково-пояснювальної записки: вступ, сучасного стану галузі дослідження, обробка даних з датчиків у коді, компонування частин коду у єдине програмне забезпечення, тестування працездатності програмного забезпечення, висновки.

5 Перелік ілюстративного матеріалу: вигляд сторінки налаштування мережі WiFiManager, результат повідомлення адреси плати у Telegram, результат отримання даних у локальній мережі, код Python-застосунку для запуску NGROK, результат отримання даних через мережу Інтернет, відображення даних у веб-

застосунку.

6 Консультанти розділів роботи приведено в т аблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, д	
		видав	пр
1-4	Богомолов С. В., к.т.н., доцент каф. ОТ	21.03.25	13
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план виконання БКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Термін виконання	Підпи
1	Аналіз завдання	21.03.25 — 30.05.25	
2	Огляд сучасного стану галузі дослідження	21.03.25 — 30.05.25	
3	Аналіз апаратно забезпечення	21.03.25 — 30.05.25	
4	Розробка частин коду для опрацювання даних з датчиків	14.04.25—18.04.25	
5	Розробка частини коду для під'єднання до мережі Wi-Fi	18.04.25—25.04.25	
6	Налагодження зв'язку ESP32-WROOM з NGROK	25.04.25—2.05.25	
7	Тестування програмного забезпечення	20.05.25—23.05.25	
8	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25—30.05.25	
9	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	

Студент Владислав ДОВГАНЬ

Керівник роботи к.т.н., доц. каф. ОТ Сергій БОГОМОЛОВ

АНОТАЦІЯ

УДК: 004.738.5:004.77:621.396.96

Довгань В. В. Програмне забезпечення для апаратно-програмного комплеку моніторингу стану оточуючого середовища. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025, 79 с.

На укр.мові. Бібліогр.: 39 назв, рис.: 9, табл: 2.

Комплексна бакалаврська робота описує розробку програмного забезпечення для апаратно-програмного комплексу моніторингу стану навколишнього середовища на основі концепції Інтернету речей (Internet of Things, IoT). Основною метою дослідження стала розробка ефективного та надійного програмного забезпечення, що забезпечує збір, обробку та передачу даних для можливості їх подальшого відображення у веб-застосунку.

У рамках виконання комплексної бакалаврської роботи реалізовано взаємодію між мікроконтролером ESP32 та модулями контролю показників, а саме: модуль контролю температури та атмосферного тиску BMP280, модуля контролю рівня освітлення BH1750 та датчика газів MQ-135. Реалізовано бездротову передачу даних у форматі JSON у вебзастосунок.

Програмне забезпечення реалізоване мовою C++, передача даних у мережу Інтернет здійснюється за допомогою програмного забезпечення NGROK.

Ключові слова: Інтернет речей, ESP32, моніторинг середовища, C++, мікроконтролер, Wi-Fi.

ABSTRACT

Dovgan V. V. Software for a hardware-software complex for monitoring the state of the environment. Bachelor's qualification work in specialty 123 "Computer Engineering", educational program "Computer Engineering". Vinnytsia: VNTU, 2025, 79 p.

In Ukrainian. Bibliogr.: 39 titles, figures: 9, tables: 2.

The comprehensive bachelor's thesis is devoted to the development of software for a hardware-software complex for monitoring the state of the environment based on the concept of the Internet of Things (IoT). The main goal of the research was to develop effective and reliable software that provides collection, processing and transferring data for further display in a web application.

As part of the comprehensive bachelor's thesis, interaction between the ESP32 microcontroller and indicator control modules was implemented, namely: the BMP280 temperature and atmospheric pressure control module, the BH1750 lighting level control module, and the MQ-135 gas sensor. Wireless data transmission in JSON format to a web application was implemented.

The software is implemented in C++, data transfer to the Internet is carried out using NGROK software.

Keywords: Internet of Things, ESP32, environmental monitoring, C++, microcontroller, Wi-Fi.

ЗМІСТ

ВСТУП

Розвиток мікроелектроніки та мікроконтролерних систем стає все більш значущим у сучасному технологічному середовищі, оскільки інновації безпосередньо впливають на підвищення якості життя та вирішення екологічних проблем. Особливої уваги набувають рішення, що базуються на концепції Інтернету речей (IoT), які відкривають нові можливості для дистанційного та безперервного моніторингу навколишнього середовища з високою точністю.

Актуальність теми зумовлена зростаючою потребою в системах, що здатні зчитувати екологічні показники у режимі реального часу, зокрема якість повітря, температуру, вологість, освітленість тощо. Наукові дослідження та прикладні розробки демонструють зацікавлення у впровадженні IoT-пристроїв на базі мікроконтролерів ESP32, які є енергоефективними, багатозадачнимим та економічно вигідними.

Метою дослідження є розробка та реалізація програмного забезпечення для апаратно-програмного комплексу моніторингу, побудованого на основі мікроконтролера ESP32-WROOM з використанням датчиків контролю параметрів навколишнього середовища. Завданням є забезпечення безперервного збору, обробки та передачі екологічних даних у форматі JSON з використанням сучасних засобів мережевої комунікації.

Об’єктом дослідження є програмна реалізація IoT-системи екологічного моніторингу, яка функціонує на базі ESP32-WROOM та датчиків BMP280, BH1750 і MQ-135.

Предметом дослідження є методи обміну даними між мікроконтролером і зовнішнім програмним середовищем, реалізація протоколів передачі даних, форматування інформації та інтеграція з веб-застосунками.

Методи дослідження, що використовуються у дослідженні: розробка алгоритмів збору даних, налаштування комунікаційних протоколів, синхронізація роботи з віддаленим сервером, тестування програмної частини системи на стабільність і точність передачі інформації.

Апробація результатів бакалаврської роботи: зроблено доповідь на Міжнародну науково-практичну Інтернет-конференцію «Молодь в науці: дослідження, проблеми, перспективи».

Публікація за темою роботи: Використання технологій бездротового зв’язку у проєктах концепції Інтернету Речей (Internet of Things) / Павленко П. І., Довгань В. В., Богомолов С. В. // Матеріали Міжнародної науково-практичної Інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи», 2025 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/25218

1 СУЧАСНИЙ СТАН ГАЛУЗІ ДОСЛІДЖЕННЯ

1.1 Обробка цифрових та аналогових сигналів мікроконтролерами

Мікроконтролери становлять основу сучасних вбудованих систем, що широко застосовуються у різних галузях — від промислової автоматизації до медицини і побутової електроніки. Ключова функція мікроконтролера — це обробка сигналів, які надходять від різноманітних датчиків і пристроїв. Ці сигнали бувають аналоговими та цифровими, що визначає специфіку методів їхньої обробки. Аналогові сигнали, як правило, є безперервними величинами, які відображають реальні фізичні параметри — температуру, тиск, освітленість, концентрацію газів тощо. Цифрові ж сигнали мають дискретну природу, що дозволяє їм приймати обмежену кількість станів, зазвичай два — «0» або «1». Мікроконтролери призначені для збору, перетворення, аналізу і керування на основі цих сигналів, забезпечуючи ефективну взаємодію з навколишнім середовищем.

Аналогові сигнали мають безперервну змінність у часі і значеннях. Вони характеризують природні фізичні процеси, які в реальному світі не мають дискретних стрибків. Для взаємодії з мікроконтролером аналоговий сигнал необхідно перетворити в цифрову форму, оскільки сам мікроконтролер працює з дискретними даними. Це перетворення здійснюється за допомогою аналогово-цифрового перетворювача (АЦП), який є невід’ємною частиною більшості сучасних мікроконтролерів.

Цифрові сигнали відрізняються тим, що їхні значення дискретні і чітко визначені. Вони можуть виникати як результат прямого формування сигналу електронними пристроями або після оцифрування аналогових даних. Цифрова форма інформації значно полегшує її зберігання, передачу та обробку, а також підвищує надійність систем.

Вибір типу сигналу для конкретного застосування визначається вимогами точності, швидкодії, складності апаратної реалізації та енергоспоживання системи. Ці фактори є критичними при проектуванні вбудованих систем контролю та моніторингу [1].

Основним етапом роботи з аналоговими сигналами є їхнє перетворення у цифрову форму. Вбудовані АЦП мікроконтролерів дозволяють перетворювати змінну напругу в числовий код, що дає змогу обробляти сигнал за допомогою цифрових алгоритмів. Розрядність АЦП — один з ключових параметрів, який визначає точність вимірювань і кількість доступних рівнів квантування.

Розрядність АЦП безпосередньо впливає на роздільну здатність системи вимірювань. Наприклад, при 10-бітному перетворенні можна отримати 1024 рівні квантування, що може бути недостатнім для вимірювань з високою точністю, де краще використовувати 12 або 16 біт. Швидкість перетворення визначає частоту оновлення сигналів і є важливою для систем реального часу.

Крім того, сучасні мікроконтролери підтримують багатоканальне зчитування аналогових сигналів, що дозволяє підключати до одного контролера декілька датчиків і здійснювати їх послідовний моніторинг [2]. Це є надзвичайно зручним для побудови комплексних систем для контролю довкілля, виробничих процесів чи охорони здоров’я.

Після отримання цифрового представлення сигналу мікроконтролер виконує його обробку, яка може включати фільтрацію, корекцію, аналіз та інші операції. Основна перевага цифрової обробки полягає у можливості реалізувати складні алгоритми, які неможливо або дуже складно виконати на аналоговому рівні.

Однією з найпоширеніших процедур є фільтрація сигналів для усунення шумів і перешкод. Програмні фільтри, такі як ковзне середнє або цифрові фільтри низьких частот, застосовуються для згладжування даних та підвищення якості інформації. У сучасних мікроконтролерах часто використовують адаптивні фільтри, які динамічно підлаштовуються під умови зовнішнього середовища [3].

Серед інших методів цифрової обробки важливу роль відіграють алгоритми виявлення подій, класифікації сигналів та обробки імпульсних послідовностей. Завдяки цифровому формату можна реалізувати дискретне перетворення Фур’є, спектральний аналіз, а також застосовувати методи машинного навчання для прогнозування поведінки систем.

Водночас аналогова обробка сигналів залишається актуальною і необхідною

для попередньої підготовки сигналу перед оцифруванням. Вона включає підсилення, фільтрацію, масштабування та інші операції, що підвищують якість сигналу.

Підсилювачі з низьким рівнем шумів і апаратні фільтри допомагають зменшити вплив зовнішніх перешкод, що особливо важливо для точних вимірювань у складних промислових і природних середовищах. Крім того, апаратна обробка сигналів дозволяє зменшити навантаження на процесор, збільшуючи швидкодію системи в цілому [4].

Сучасні мікроконтролери все частіше інтегрують аналогові функції, що забезпечує більш компактні та енергоефективні рішення. Це дозволяє створювати високотехнологічні прилади з великим набором функцій у компактному корпусі [5].

В сучасних умовах розробки вбудованих систем особливо важливим є поєднання високої точності, швидкодії та енергоефективності. Обробка сигналів у мікроконтролерах має відповідати цим вимогам, водночас забезпечуючи стабільність і надійність роботи.

Одним із головних викликів є баланс між складністю алгоритмів обробки і обмеженими ресурсами мікроконтролера. Нові покоління мікроконтролерів оснащені апаратними прискорювачами цифрової обробки сигналів (DSP), що значно розширює можливості систем [6].

Паралельно розвиваються і методи інтелектуальної обробки даних із застосуванням штучного інтелекту та машинного навчання, які вже починають інтегруватися у мікроконтролерні платформи для підвищення адаптивності та точності систем контролю [7].

1.2 Бездротовий зв’язок у системах IoT

Інтернет речей (Internet of Things, IoT) — це одна з найбільш розвинутих галузей сучасних технологій, що значно покращує способи збору, передачі та аналізу даних у реальному часі. В основі будь-якої IoT-системи лежить ефективний канал комунікації між пристроями, і саме бездротовий зв’язок став ключовим

чинником її поширення та впровадження. Бездротові технології зв’язку дозволяють поєднувати численні пристрої, розташовані у різних фізичних локаціях, забезпечуючи гнучкість, мобільність і масштабованість систем.

Бездротовий зв’язок — це передача інформації за допомогою електромагнітних хвиль без використання фізичних провідників. У контексті IoT він набув особливого значення, оскільки датчики, виконавчі пристрої, контролери часто розташовані в складнодоступних або віддалених місцях, де прокладання кабелів неможливе або економічно недоцільне.

Відповідно до сучасних наукових оглядів, бездротовий зв’язок у IoT характеризується низьким рівнем енергоспоживання, масштабованістю, різноманіттям протоколів, а також підтримкою різних топологій мереж. Проте при цьому існують і виклики, такі як обмежена пропускна здатність, питання безпеки передачі даних, а також взаємодія великої кількості пристроїв в одному середовищі [8].

Важливим питанням також є оптимізація ресурсів мережі та розробка енергоефективних протоколів передачі для IoT-систем. Це особливо актуально для пристроїв з живленням від акумуляторів, що експлуатуються у віддалених або мобільних об’єктах [9].

Усі бездротові технології можна умовно поділити на кілька категорій залежно від радіусу дії, пропускної здатності та енергоспоживання. Важливо розуміти, що вибір конкретної технології залежить від специфіки завдань системи IoT.

Bluetooth Low Energy (BLE) — одна з найбільш популярних технологій для зв’язку IoT-пристроїв на малих відстанях. Вона характеризується низьким рівнем енергоспоживання, підтримкою великої кількості одночасних підключень та простотою реалізації. BLE використовується у портативних пристроях, медичній електроніці, смарт-годинниках, домашній автоматизації тощо.

Wi-Fi, хоча і споживає більше енергії, широко застосовується у випадках, коли потрібна висока пропускна здатність і інтеграція із існуючою інфраструктурою інтернету. Зокрема, нові стандарти Wi-Fi 6 802.11ax пропонують

покращену енергоефективність і підтримку більшої кількості одночасних з’єднань [10].

Використання BLE є доцільним для систем моніторингу стану навколишнього середовища завдяки її можливостям адаптивного енергозбереження та простоті інтеграції з мобільними платформами [11].

Для систем, які охоплюють більші території, часто застосовуються технології LPWAN (Low Power Wide Area Network). Найпоширенішими стандартами є LoRaWAN та NB-IoT. Вони характеризуються низькою пропускною здатністю, але дозволяють передавати дані на десятки кілометрів при мінімальному енергоспоживанні.

LoRaWAN є відкритим протоколом, що широко поширений у Європі та світі. Його перевагою є можливість організації приватних мереж, що особливо важливо для промислових об’єктів, сільського господарства та інфраструктури міст (Smart City) [12].

NB-IoT (Narrowband IoT), базований на мобільних мережах LTE, забезпечує високу надійність та якість зв’язку, а також інтеграцію з існуючими телекомунікаційними мережами. Це робить його популярним вибором для масштабних IoT-проектів із високими вимогами до безпеки передачі даних [13].

Типова IoT-система складається з багатьох елементів: сенсорів, виконавчих пристроїв, шлюзів, серверів і користувацьких інтерфейсів. Бездротові технології використовуються для комунікації між сенсорами та шлюзами, а також між шлюзами і хмарними сервісами.

Відповідно до моделі OSI, бездротовий зв’язок охоплює фізичний рівень (радіохвилі, частотні канали), канальний рівень (управління доступом до середовища), а також транспортний і мережевий рівні, що здійснюють маршрутизацію і доставку пакетів.

Однією з ключових проблем у бездротових IoT-системах є захист переданої інформації. Оскільки дані проходять через відкриті радіоканали, вони вразливі до прослуховування, підміни та атак типу «людина посередині» (Man-in-the-Middle, MitM).

Захисні механізми включають шифрування даних, автентифікацію пристроїв, використання VPN та інших засобів мережевої безпеки. Ефективна система безпеки має враховувати обмежені ресурси IoT-пристроїв, зокрема обмежену обчислювальну потужність і енергозабезпечення [14].

Крім того, впровадження блокчейн-технологій для захисту цілісності даних та забезпечення довіри між пристроями поступово набуває популярності [15].

Бездротовий зв'язок у IoT продовжує активно розвиватися завдяки інноваціям у сфері апаратного забезпечення, алгоритмах енергозбереження та стандартах зв'язку. Технології 5G відкривають нові горизонти для IoT, забезпечуючи більшу пропускну здатність, нижчу затримку і покращену підтримку масових пристроїв.

Серед перспективних напрямів також виділяють інтеграцію штучного інтелекту безпосередньо на пристроях IoT для автономної обробки даних, а також розвиток гібридних мереж, що поєднують різні типи бездротових технологій для оптимальної роботи систем [16].

1.3 Способи зв'язку пристроїв у локальній мережі з мережею Інтернет

У сучасному світі питання взаємодії локальних мереж із глобальною мережею Інтернет набуває дедалі більшої актуальності, особливо у контексті розвитку концепції Інтернету речей (IoT), децентралізованих обчислень та хмарних технологій. Локальні мережі (LAN) забезпечують внутрішню взаємодію пристроїв у межах певного простору, наприклад у будівлі, офісі чи приватному помешканні. Проте для реалізації повноцінного функціонування цих пристроїв, зокрема для обміну даними з віддаленими серверами, хмарними платформами чи іншими пристроями поза межами локального середовища, необхідно забезпечити ефективне з'єднання з Інтернетом [17].

В основі зв'язку локальних пристроїв з глобальною мережею лежить поняття маршрутизації, трансляції адрес (NAT), а також використання шлюзів і проміжних серверів. Стандартна архітектура передбачає наявність маршрутизатора, який виконує роль посередника між локальною мережею та глобальним середовищем.

Завдяки NAT (Network Address Translation) можливо використовувати одну глобальну IP-адресу для багатьох локальних пристроїв, що надсилають і отримують пакети ззовні [18].

Пряме з’єднання локальних пристроїв з Інтернетом часто неможливе через обмежену кількість публічних IP-адрес, тому саме NAT дозволяє здійснювати ефективний розподіл і безпеку. При цьому маршрутизатор не лише забезпечує трансляцію адрес, а й виконує функції фільтрації, маршрутизації, управління портами (port-forwarding), а також організації демілітаризованих зон (DMZ) [19].

Один із найпоширеніших способів з’єднання пристроїв у локальній мережі з Інтернетом — використання маршрутизатора або шлюзу з функцією NAT. У такій моделі кожен пристрій отримує приватну IP-адресу через DHCP (Dynamic Host Configuration Protocol), після чого маршрутизатор, маючи публічну IP-адресу, виконує трансляцію і переспрямовує трафік на відповідний локальний вузол [18]. Такий підхід дозволяє підвищити безпеку мережі та спростити керування підключеннями.

Інший підхід передбачає використання віртуальних приватних мереж (VPN). VPN забезпечує створення зашифрованого тунелю між локальною мережею та віддаленим сервером або хмарною платформою. Це дозволяє забезпечити безпечний обмін даними та створити віртуальну єдину мережу для пристроїв, фізично розташованих у різних місцях [20]. В умовах зростаючої кількості загроз безпеці, VPN-технології набувають особливого значення для промислових систем, корпоративних мереж і домашніх IoT-систем [21]. Завдяки цьому підходу можливо забезпечити додаткову анонімність, цілісність та конфіденційність трафіку [22].

У практиці також використовуються проксі-сервери та шлюзи, які виступають посередниками між локальною мережею та зовнішнім Інтернетом. Проксі-сервери можуть кешувати вміст, фільтрувати трафік і забезпечувати контроль доступу, що робить їх актуальними в освітніх закладах, підприємствах і мережах з підвищеними вимогами до безпеки [23]. Крім цього, застосування шлюзів дозволяє централізовано регулювати політику доступу, а також реалізовувати додаткові рівні захисту за допомогою антивірусного аналізу чи

міжмережевого екрану [24].

З розвитком мобільного Інтернету та бездротових технологій дедалі частіше використовуються альтернативні канали зв’язку, зокрема стільникові мережі 4G/5G або мобільні точки доступу. У випадках, коли дротове підключення неможливе або недоцільне, такі варіанти дозволяють забезпечити стабільний зв’язок з Інтернетом навіть у віддалених або тимчасових точках доступу [19]. Зокрема, у сфері IoT часто використовуються мобільні модулі з підтримкою GPRS, NB-IoT чи LTE-M.

Особливу увагу в архітектурі зв’язку слід приділяти адресації та маршрутизації. Використання протоколів IPv4 і IPv6 визначає здатність мережі масштабуватися та підтримувати велику кількість пристроїв. У той час як IPv4 обмежений 32-бітною адресацією, що створює дефіцит адрес, перехід на IPv6 дозволяє уникнути цієї проблеми та забезпечити кращу маршрутизацію й безпеку [25]. У корпоративних мережах все частіше впроваджуються механізми подвійного стеку «dual-stack», що дозволяє підтримку обох протоколів одночасно.

Водночас із технічним зростанням мережевої інфраструктури виникають і нові виклики. До них належать питання кібербезпеки, автентифікації пристроїв, захисту від DDoS-атак, а також потреба у масштабованих рішеннях для великих IoT-систем. Використання динамічного DNS, систем балансування навантаження та хмарних сервісів є відповіддю на частину цих викликів, однак вимагає складної інтеграції та постійного моніторингу [23].

Перспективним напрямом розвитку є використання Software Defined Networking (SDN) — підходу, за якого логіка керування мережею відокремлюється від фізичного рівня. Це дозволяє швидше адаптуватися до зміни умов та оптимізувати маршрутизацію трафіку. Крім того, технології Network Function Virtualization (NFV) забезпечують гнучкість і масштабованість, дозволяючи реалізовувати функції маршрутизаторів, проксі, брандмауерів на базі програмного забезпечення [19].

Таким чином, сучасні способи зв’язку локальних пристроїв з Інтернетом є результатом еволюції мережевих технологій, в якій значну роль відіграють

безпекові механізми, гнучкість архітектури, доступність каналів зв’язку та здатність до масштабування. Надалі спостерігатиметься ще більша інтеграція між локальними мережами та хмарними платформами, із застосуванням розумних протоколів маршрутизації, динамічного керування ресурсами та штучного інтелекту для оптимізації мережевих процесів [25].

1.4 Програмування мікроконтролерів

Програмування мікроконтролерів охоплює низку теоретичних і практичних аспектів, пов’язаних із розробкою вбудованого програмного забезпечення для керування апаратними ресурсами, що інтегровані в єдину систему на основіф мікроконтролера. Цей процес не обмежується лише написанням коду – він включає аналіз вимог до пристрою, вибір архітектури, врахування обмежень пам’яті та продуктивності, а також інтеграцію зовнішніх периферійних модулів. Особливістю мікроконтролерів як об’єктів програмування є їхня тісна інтеграція з фізичним світом, що вимагає від розробника чіткого розуміння як електронної частини, так і поведінки об'єктів у середовищі, з якими пристрій взаємодіє.

Сучасні мікроконтролери найчастіше програмуються за допомогою мов низького або середнього рівня, серед яких найпоширенішими є C, C++ та Assembly. Обґрунтованим вибором є C, оскільки ця мова забезпечує достатню близькість до апаратного рівня й водночас дозволяє реалізовувати складні логічні структури і обробку даних без суттєвих втрат у продуктивності. У випадках, коли потрібно забезпечити максимальну швидкодію або виконати специфічну конфігурацію регістрів, застосовується мова асемблера, яка дозволяє керувати кожною інструкцією процесора вручну, але значно ускладнює підтримку й масштабування коду.

Важливою особливістю програмування мікроконтролерів є безпосередня робота з апаратними ресурсами – портами введення-виведення, таймерами, АЦП, ШІМ, інтерфейсами UART, SPI, I2C тощо. Відповідно, велике значення має глибоке розуміння архітектури мікроконтролера, яка визначає доступність та структуру внутрішніх блоків. Наприклад, програмісту доводиться враховувати

конкретну кількість регістрів, об’єм оперативної та постійної пам’яті, розрядність шин і тактову частоту, щоб оптимально використати наявні ресурси. Типовим прикладом є налаштування таймера для генерації імпульсів ШІМ – ця операція вимагає точної конфігурації відповідних регістрів, що відповідають за частоту, роздільну здатність та режим роботи таймера [33].

На рівні програмного забезпечення взаємодія з апаратними ресурсами часто здійснюється через бібліотеки низького рівня, які постачаються виробником мікроконтролера. Такі бібліотеки дозволяють абстрагуватися від прямого запису в регістри і надають більш зручні методи ініціалізації периферії. Наприклад, платформи STM32 широко використовують HALL-бібліотеки, що забезпечують крос-платформеність і пришвидшують процес розробки. Аналогічно, для платформ Arduino розроблено численні бібліотеки з відкритим кодом, що покривають більшість популярних сенсорів і периферійних пристроїв. Хоча використання таких бібліотек може зменшити ефективність коду, воно значно підвищує зручність розробки і дозволяє зосередитися на вищому рівні логіки програми.

Ще одним аспектом, що вирізняє програмування мікроконтролерів, є логіка, орієнтована на події, яка реалізується за допомогою переривань. Переривання дають змогу реагувати на зовнішні або внутрішні події без необхідності постійного опитування стану апаратних модулів, що суттєво підвищує енергоефективність і швидкодію системи. Наприклад, замість безперервної перевірки натиснення кнопки, мікроконтролер може перебувати у сплячому режимі й активуватись лише при отриманні відповідного сигналу. Однак, робота з перериваннями вимагає особливої уваги до синхронізації доступу до спільних ресурсів і врахування можливих конфліктів при зміні значення змінних у різних контекстах виконання. Загальна схема обробки переривань у контролері наведена на рисунку 1.1.

Оскільки мікроконтролери часто використовуються у системах, таких як медичні пристрої або промислові контролери, до програмного забезпечення висуваються підвищені вимоги щодо надійності, безпеки та передбачуваності. У цьому контексті важливим є проведення ретельного тестування програмного

забезпечення на реальному обладнанні або в середовищах моделювання, таких як симулятори або емулятори

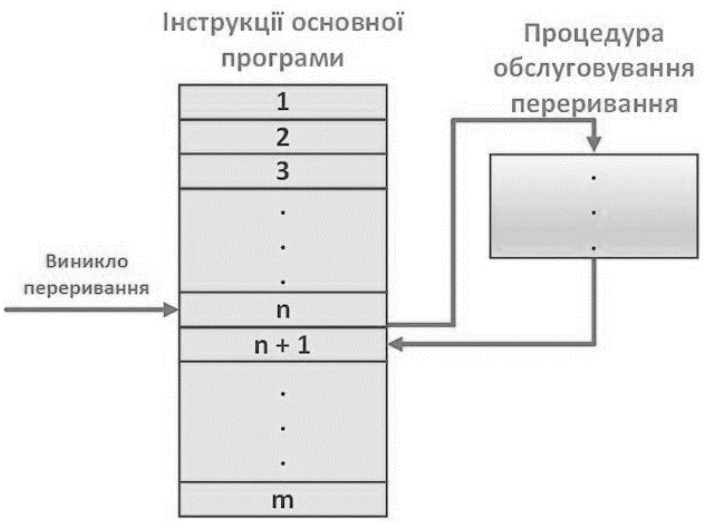

Рисунок 1.1 – Загальна схема обробки переривань у контролері

. Крім того, велике значення має розробка систем з урахуванням концепції безпеки, коли будь-який збій у роботі не повинен призводити до неконтрольованої поведінки пристрою.

Ще один напрямок, що активно розвивається у програмуванні мікроконтролерів, – це інтеграція із засобами зв’язку, зокрема бездротовими протоколами. Сучасні мікроконтролери часто мають вбудовані модулі Wi-Fi, Bluetooth, LoRa або ZigBee, що дозволяє створювати пристрої Інтернету речей без потреби в додаткових комунікаційних модулях. Програмування таких пристроїв охоплює використання стеків протоколів, організацію обміну даними з віддаленими серверами, реалізацію протоколів MQTT, HTTP або CoAP. Це вимагає знань не лише апаратної частини, але й мережевих технологій та принципів побудови клієнт-серверної архітектури.

Однією з характерних рис програмування мікроконтролерів є циклічна природа виконання програм. На відміну від застосунків для комп’ютерів або серверів, де поширене подієво-орієнтоване програмування з багатозадачністю, більшість програм для мікроконтролерів реалізуються як нескінченний цикл, у якому постійно виконується обробка вхідних сигналів, формування вихідних дій,

оновлення стану та моніторинг подій. Такий підхід зумовлений обмеженими ресурсами пристрою, де відсутня операційна система або використовується лише мікроядро реального часу. Тому програмісту необхідно самостійно продумувати послідовність подій, реалізовувати планування задач та враховувати часові затримки.

Розробка програмного забезпечення для мікроконтролерів часто супроводжується використанням спеціалізованих середовищ розробки (IDE), які забезпечують не лише зручний редактор коду, а й компілятор, інструменти налагодження та засоби програмування мікроконтролера. Популярними середовищами є STM32CubeIDE, MPLAB X, KEIL uVision, Arduino IDE (рис. 1.2), PlatformIO та інші. Вони дають змогу автоматизувати процес збірки, здійснювати покрокове відлагодження, переглядати значення регістрів, відстежувати змінні та часові характеристики виконання коду. Це дозволяє ефективно знаходити помилки на етапі розробки й уникати серйозних збоїв у роботі пристрою після впровадження.

Значною складністю під час розробки є керування пам’яттю, зокрема в системах, де обсяг оперативної пам’яті (SRAM) обмежений кількома кілобайтами. Програміст має дбати про ефективне використання масивів, уникати динамічного виділення пам’яті, обмежувати використання стеку та контролювати розміщення змінних у пам’яті. Багато сучасних компіляторів підтримують атрибути або директиви для розміщення даних у певних сегментах пам’яті, що дозволяє більш гнучко керувати ресурсами. У критичних випадках, коли необхідна максимальна оптимізація, застосовується профілювання пам’яті та аналіз розміщення функцій і змінних у прошивці.

Неможливо оминути питання тестування програмного забезпечення, яке на мікроконтролерах має свої особливості. Оскільки часто йдеться про пристрої, які працюють у реальному часі, класичні підходи до тестування, зокрема автоматизовані юніт-тести, застосовуються з обмеженнями.

Пріоритетним є тестування на реальному обладнанні, у тому числі функціональне тестування, перевірка коректності реакцій на зовнішні події,

симуляція збоїв живлення або втрати зв’язку. Для цього використовуються апаратні емулятори, логічні аналізатори, осцилографи, а також спеціалізовані тести на споживання енергії, що особливо актуально для автономних систем на батареях

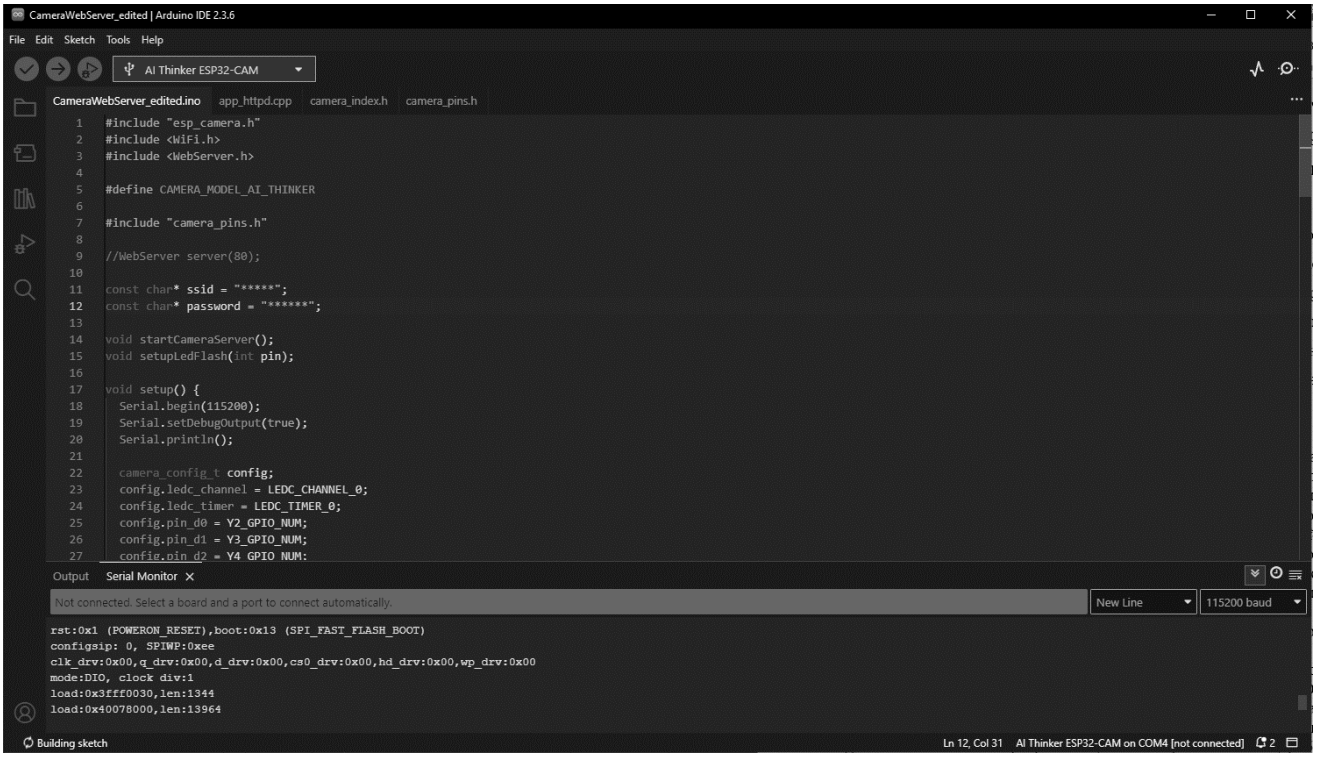

Рисунок 1.2 — Вигляд середовища розробки Arduino IDE

У більш складних проєктах, де необхідно виконувати кілька задач паралельно, використовуються операційні системи реального часу (RTOS). Вони дозволяють організувати багатозадачне виконання, розподіляти пріоритети, керувати ресурсами та синхронізувати доступ до спільних структур даних. До найпоширеніших RTOS належать FreeRTOS, Zephyr, RTX. Їхнє використання додає гнучкості та масштабуємість, але одночасно потребує вищої кваліфікації від розробника і дотримання принципів безпечного паралельного програмування. Зокрема, розробник повинен враховувати стан блокування задач, уникати дедлоків та грамотно використовувати семафори й черги.

Окрему категорію становить віддалене оновлення прошивки (FOTA – Firmware Over-The-Air), що стає особливо актуальним в умовах розгортання великої кількості пристроїв у реальному середовищі. Програмування такої функціональності вимагає створення захищеного механізму перевірки цілісності

нової версії прошивки, резервної області пам’яті, яка зберігає попередню стабільну версію, та відповідного бутлоадера, що здійснює оновлення. Застосування FOTA потребує також реалізації протоколів шифрування та автентифікації для запобігання несанкціонованому втручанню. Розробка таких функцій ставить програмування мікроконтролерів на рівень системного програмування, де важливо враховувати навіть такі фактори, як стійкість до збоїв живлення під час оновлення.

Таким чином, програмування мікроконтролерів не є простою реалізацією алгоритмів керування, а інтегрує цілий спектр знань з апаратної інженерії, комп’ютерної архітектури, електроніки, системного та прикладного програмуванн

2 ПРОГРАМНА ІНТЕГРАЦІЯ З ЕЛЕМЕНТНОЮ БАЗОЮ ІОТ-СИСТЕМИ

Цей розділ присвячений розробці частин коду для забезпечення функціонування апаратного комплексу, розробка якого описана у першій частині комплексної бакалаврської кваліфікаційної роботи з темою «Апаратно-програмний комплекс моніторингу стану оточучого середорвища на базі IoT-засобів. Частина перша. Апаратне забезпечення». Відображення даних, що є результатом роботи готового програмного забезпечення, описаного у розділі 3 цієї роботи, здійснюється у веб-застосунку, розробка якого описана у третій частині комплексної бакалаврської кваліфікаційної роботи з темою « Апаратно-програмний комплекс моніторингу стану оточучого середорвища на базі IoT-засобів. Частина 3. Вебзастосунок для трансляції відеопотоку».

2.1 Мікропроцесорна платформа ESP32-WROOM

У сучасних системах моніторингу навколишнього середовища значне місце займають вбудовані обчислювальні платформи, здатні автономно зчитувати показники з датчиків, обробляти отримані дані та передавати їх на віддалені сервери або клієнтські пристрої. Однією з найпопулярніших платформ для таких цілей є мікроконтролерна плата ESP32-WROOM, яка поєднує високий рівень продуктивності, вбудовані засоби бездротового зв’язку (Wi-Fi, Bluetooth) та доступність для широкого кола розробників.

ESP32-WROOM — це модуль на базі мікроконтролера ESP32 від компанії Espressif Systems, що має двоядерний процесор Tensilica LX6 із тактовою частотою до 240 МГц, до 520 КБ SRAM, флеш-пам'ять до 4 МБ, а також численні периферійні інтерфейси, зокрема SPI, I2C, UART, ADC та PWM. Завдяки цим характеристикам плата дозволяє ефективно організовувати зчитування даних з аналогових і цифрових датчиків, попередню обробку сигналів, а також формування та передачу інформації у форматі JSON у мережу Інтернет [26].

Передача даних у форматі JSON (JavaScript Object Notation) стала стандартом

для обміну структурованою інформацією в сучасних веб-орієнтованих додатках та пристроях. Цей формат є легким для обробки як на стороні клієнта, так і на стороні сервера, та добре підтримується більшістю мов програмування й платформ, зокрема JavaScript, Python, Node.js, Angular тощо. ESP32-WROOM може формувати JSON-об’єкти безпосередньо у прошивці за допомогою бібліотек, таких як ArduinoJson, що суттєво спрощує процес розробки систем збору даних.

У типових системах на базі ESP32-WROOM збір даних виконується циклічно: періодично активуються датчики (наприклад, температури, вологості, освітленості, газів), дані зчитуються з їх аналогових або цифрових виходів, проходять попередню обробку (наприклад, усереднення, фільтрація або масштабування), після чого на їх основі формується JSON-структура з полями, що відображають поточні показники. Ця структура передається на сервер за допомогою мережевого з’єднання, яке встановлюється через вбудований Wi-Fi-модуль.

Загальна архітектура таких систем передбачає взаємодію ESP32-WROOM із віддаленим сервером, що реалізує API для приймання та обробки вхідних запитів. Часто використовується протокол HTTP з методом POST, який дозволяє передати сформований JSON-документ до заданої адреси. ESP32 може виступати як HTTP-клієнт, реалізований за допомогою бібліотек ESPAsyncWebServer, WiFiClient або стандартного WiFi.h. У разі необхідності безперервної передачі даних або реакції на події доцільно використовувати протокол WebSocket або MQTT. Останній, зокрема, є надзвичайно ефективним у системах з обмеженими ресурсами, оскільки передбачає легку систему публікації/підписки з мінімальним навантаженням на канал зв’язку.

Переваги ESP32-WROOM у порівнянні з аналогами, такими як Arduino Uno, Raspberry Pi Pico W чи STM32, полягають насамперед у гнучкості, високому рівні інтеграції та наявності бездротових інтерфейсів. Наприклад, Arduino Uno не має вбудованого Wi-Fi і потребує додаткових модулів (ESP8266, Ethernet Shield) для передачі даних по мережі. Raspberry Pi Pico W, хоча й підтримує Wi-Fi, поступається ESP32 за кількістю доступних периферійних інтерфейсів і

потужністю. STM32 — потужна платформа, однак її налаштування потребує складнішого програмного забезпечення, що ускладнює швидкий запуск проєкту [27].

Таким чином, ESP32-WROOM забезпечує оптимальний баланс між продуктивністю, енергоефективністю, простотою програмування й доступом до мережі Інтернету. Завдяки наявності великої кількості відкритих бібліотек, прикладів та документації, ця платформа стала популярним вибором для реалізації апаратно-програмних рішень у сфері моніторингу, зокрема для екологічних задач, smarthome-проєктів, систем контролю клімату та інших IoT-рішень.

Ефективна передача даних з мікроконтролера на віддалений сервер є одним з ключових аспектів функціонування апаратно-програмних комплексів моніторингу. ESP32-WROOM, маючи вбудований модуль Wi-Fi, дозволяє використовувати цілу низку мережевих протоколів, кожен з яких має свої переваги залежно від вимог до швидкості, стабільності з'єднання, енергоспоживання і складності реалізації.

Найчастіше для передачі екологічних або кліматичних даних, сформованих у форматі JSON, використовується протокол HTTP (HyperText Transfer Protocol). Цей протокол є основою роботи веб-серверів і клієнтів, а також добре підтримується всіма мовами програмування та фреймворками. На ESP32 HTTP реалізується через бібліотеки HTTPClient.h, WiFi.h та інші. У цьому прикладі продемонстровано базове підключення до Wi-Fi, проте в реальній реалізації було використано WiFiManager для автоматичного налаштування. Приклад використання бібліотеки WiFi.h наведено у лістингу 2.1.

Лістинг 2.1 — Використання бібліотеки WiFi.h для підключення до Wi-Fi

```
#include <WiFi.h>
const char* ssid = "kvitkova7b";
const char* password = "0978708818";
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());
}
void loop() {
}
```

Дані передаються методом POST у тілі запиту у вигляді JSON-об'єкта.

Цей підхід є простим у реалізації, але має певні обмеження щодо енергоефективності та швидкості — кожен запит потребує встановлення TCP-з’єднання, що може бути ресурсомістким.

Для зменшення мережевого навантаження та підвищення ефективності при великій кількості переданих пакетів застосовується протокол MQTT (Message Queuing Telemetry Transport). MQTT є легким брокерським протоколом, що базується на архітектурі «видавець–підписник» і працює поверх TCP/IP. Його головною перевагою є здатність підтримувати стабільне з’єднання навіть при низькій якості мережі, а також мінімальне споживання енергії. У системах на ESP32 зазвичай використовується бібліотека PubSubClient, яка забезпечує підключення до брокера, наприклад, Mosquitto, HiveMQ, або Adafruit IO, публікацію повідомлень та підписку на теми. Дані у форматі JSON можуть передаватися як рядки в MQTT-повідомленнях.

Ще одним варіантом є використання WebSocket-з’єднань, які дозволяють встановити постійний двонаправлений канал між клієнтом і сервером. Це дає змогу миттєво передавати дані без необхідності повторного встановлення з’єднання. Для ESP32 існує бібліотека WebSocketsClient.h, що реалізує цю функціональність. Однак WebSocket-взаємодія потребує складнішої серверної частини, здатної обробляти постійні з’єднання, і більше підходить для інтерактивних застосунків.

У ряді випадків застосовуються і UDP-з’єднання, проте через відсутність гарантії доставки пакетів вони не так широко використовуються у системах екологічного моніторингу, де точність даних має вирішальне значення. Однак для періодичної передачі великої кількості даних без потреби в зворотному зв’язку цей

протокол може бути доцільним.

Усі зазначені мережеві протоколи використовують стеки TCP/IP, реалізовані в SDK для ESP32 — ESP-IDF або Arduino Core for ESP32. Вибір відповідного протоколу залежить від конкретних вимог до системи: необхідності у двосторонньому зв’язку, обсягу даних, стабільності підключення, безпеки та швидкості реакції [28].

Безпека передачі даних також є важливим аспектом. У багатьох проєктах використовуються HTTPS або захищені версії MQTT (MQTTS), які забезпечують шифрування трафіку через TLS. Хоча це потребує додаткових обчислювальних ресурсів, ESP32 завдяки своїй архітектурі здатний обробляти TLS-з’єднання. Зокрема, бібліотека WiFiClientSecure дозволяє працювати з HTTPS без особливих ускладнень, а зберігання сертифікатів можливе у флеш-пам’яті модуля.

Розробка програмного забезпечення для ESP32-WROOM є критичним етапом створення апаратно-програмного комплексу, що передбачає зчитування даних з датчиків, їх попередню обробку, формування структури JSON та подальшу передачу в мережу. Для цього існують два основних середовища розробки: ESP-IDF (офіційний фреймворк від Espressif) і Arduino IDE (спрощене середовище для швидкої розробки на основі знайомої платформи Arduino).

Arduino IDE є найбільш поширеним серед незалежних розробників і освітніх закладів завдяки простому інтерфейсу та великій кількості бібліотек. Для ESP32 в Arduino IDE використовується спеціальний core-пакет (ESP32 Arduino Core), який забезпечує реалізацію основних функцій роботи з апаратними ресурсами, Wi-Fi, Bluetooth, таймерами, ADC, PWM тощо.

Серед основних бібліотек, що застосовуються для збору й передачі даних, можна виділити такі:

— WiFi.h — забезпечує підключення до бездротової мережі;

— HTTPClient.h — використовується для реалізації HTTP-запитів (GET, POST);

— ArduinoJson.h — одна з найефективніших бібліотек для створення, парсингу та серіалізації JSON-структур, зручна у використанні навіть на

мікроконтролерах з обмеженою пам’яттю;

— PubSubClient.h — реалізує MQTT-клієнт з підтримкою тем і публікацій;

— Wire.h — для роботи з датчиками через інтерфейс I2C (наприклад, BH1750 або BMP280);

— Adafruit Sensor та інші датчик-орієнтовані бібліотеки, що підтримують специфічні пристрої.

У базових реалізаціях підключення до Wi-Fi з ESP32 використовується бібліотека WiFi.h, що потребує жорсткого задання SSID та пароля в коді. Проте у випадках, коли важлива мобільність пристрою або доступ до серійного порту обмежений, доцільним є використання бібліотеки WiFiManager, яка забезпечує динамічне налаштування Wi-Fi з допомогою вбудованого веб-інтерфейсу. При першому запуску або втраті мережі ESP32 створює власну точку доступу (Access Point), до якої користувач може під’єднатися і вказати нові параметри Wi-Fi. Цей підхід усуває потребу у перепрошивці пристрою або підключенні до порту UART для зміни налаштувань мережі [29]. Приклад використання описаної бібліотеки наведено у лістингу 2.2.

Лістинг 2.2 — Використання бібліотеки WifiManager для підключення до мережі

```
#include <WiFiManager.h>
void setup() {
  Serial.begin(115200);
  WiFiManager wm;
  bool res = wm.autoConnect("ESP32-Setup");
  if (!res) {
    Serial.println("Не вдалося підключитися до WiFi");
  } else {
    Serial.println("Підключено!");
    Serial.println(WiFi.localIP());
  }
}
void loop() {
}
```

Важливим аспектом є ефективна організація коду: зчитування даних з

датчиків слід реалізовувати в окремих функціях або класах, з мінімізацією затримок у головному циклі loop(). Формування JSON-структур за допомогою StaticJsonDocument або DynamicJsonDocument дозволяє точно налаштовувати обсяг пам’яті під дані, що передаються, забезпечуючи стабільність роботи пристрою.

Наприклад, після зчитування температури, вологості, освітленості й рівня забруднення повітря з відповідних датчиків, на ESP32 формуються поля JSON-об'єкта (лістинг 2.3).

Лістинг 2.3 — Вигляд JSON-об’єкта

```
{  "temperature": 24.6, "humidity": 58, "pressure": 1013.2, "light": 315, "air_quality": 65  }
```

Такий об’єкт надсилається HTTP POST-запитом або публікується у відповідну тему MQTT.

У більш складних застосунках доцільно використання FreeRTOS — реального часу операційної системи, яка вбудована у SDK ESP32 та дозволяє виконувати паралельні задачі: окремо зчитування сенсорів, обробку даних, контроль зв’язку тощо. Це дозволяє підвищити надійність системи, зменшити затримки, уникати блокування головного потоку.

Порівнюючи підходи Arduino та ESP-IDF, слід зазначити, що Arduino надає швидкий старт із великою кількістю бібліотек, але має обмежену гнучкість у складних системах. Натомість ESP-IDF забезпечує повний контроль над ресурсами, кращу оптимізацію й можливість інтеграції промислових функцій, зокрема OTA-оновлень, захищеного зберігання та шифрування.

Завдяки доступності численних бібліотек, ESP32-WROOM дозволяє легко інтегрувати сенсори, що підтримують різні інтерфейси: аналогові (наприклад, MQ-135), цифрові I2C (BMP280, BH1750), UART (GPS, CO2-сенсори), що дає змогу масштабувати систему для моніторингу найрізноманітніших показників довкілля.

Не менш важливим компонентом програмного забезпечення є обробка помилок та забезпечення відновлення зв’язку. У нестабільних мережах ESP32 може втратити з’єднання з Wi-Fi або брокером MQTT, тому у програмному коді

передбачаються перевірки стану підключення й повторне підключення в разі помилки. Також варто враховувати можливість буферизації даних при тимчасовій відсутності мережі, наприклад, зберігаючи останні значення у масив до моменту відновлення передачі.

Завдяки високій енергоефективності та можливості використання режимів сну ESP32-WROOM підходить і для автономних систем, які живляться від батарей або сонячних панелей. У таких випадках програмне забезпечення має чітко регламентувати цикли пробудження, зчитування, передачі даних і повернення у сон — це реалізується через функції esp_sleep_enable_timer_wakeup() та esp_deep_sleep_start().

Підсумовуючи, можна стверджувати, що програмування платформи ESP32-WROOM з використанням сучасних бібліотек і мережевих протоколів дозволяє створювати масштабовані, гнучкі та надійні системи збору екологічних даних. Високий рівень інтеграції компонентів, підтримка JSON, наявність потужних інструментів обробки та відправки інформації робить цю платформу однією з найбільш придатних для розробки інтелектуальних IoT-рішень [30].

2.2 Інтерфейси з’єднання

У сучасних мікроконтролерних системах інтерфейси з’єднання є ключовою складовою комунікації між центральним обчислювальним блоком і сенсорними або виконавчими пристроями. Вони забезпечують не лише фізичне з’єднання, а й логіку передачі даних, узгодження сигналів і керування обміном. Для ефективного функціонування мікроконтролера, такого як ESP32, важливо враховувати особливості кожного типу інтерфейсу — від кількості необхідних виходів до швидкості передачі інформації та підтримки кількох підключених пристроїв.

Одним із найбільш універсальних інтерфейсів є I2C. Він дозволяє з’єднувати декілька пристроїв за допомогою лише двох провідників — лінії даних SDA і тактової лінії SCL. Таке з’єднання особливо зручне для компактних рішень, де кількість доступних виводів обмежена. За допомогою унікальної адресації кожен пристрій ідентифікується на шині, що дозволяє одночасно використовувати кілька

датчиків. Наприклад, сенсор освітленості BH1750 працює саме через I²C, і завдяки простому протоколу обміну легко інтегрується в систему на базі ESP32.

Іншим поширеним цифровим інтерфейсом є SPI, який забезпечує вищу швидкість передачі, але потребує більшої кількості з'єднань. Його структура передбачає окремі лінії для даних, тактування і вибору пристрою. У системах, де важлива швидкодія або потрібна передача великих обсягів інформації, SPI часто є більш доцільним за I2C. У контексті ESP32 це дозволяє реалізувати, наприклад, швидкісне зчитування з датчиків або керування дисплеями. Проте, через складнішу схему підключення SPI менш зручний для використання декілької пристроїв одночасно.

Окрім цифрових, мікроконтролери часто використовують аналогові інтерфейси. Такі сенсори, як терморезистори типу NTC, подають на мікроконтролер аналоговий сигнал, який потрібно перетворити на цифровий. Для цього ESP32 має вбудований АЦП. Однак точність зчитування залежить від стабільності напруги живлення, зовнішніх шумів і якості електричного монтажу. Аналогові з'єднання прості в реалізації, але поступаються цифровим у точності й надійності. У проєктах екологічного моніторингу вони використовуються, коли немає цифрової альтернативи або коли не потрібна висока точність.

UART — ще один поширений інтерфейс, особливо корисний для зв'язку на більші відстані або з модулями, що вже мають вбудовану логіку. Це асинхронний послідовний інтерфейс, який потребує лише двох ліній — для передавання та прийому даних. ESP32 підтримує кілька UART-каналів, що дозволяє підключати кілька пристроїв одночасно, наприклад, модулі GPS або газові сенсори з цифровим виходом. UART відзначається простотою протоколу, але потребує точного налаштування швидкості обміну. У випадках, коли важлива сумісність з промисловими пристроями або модемами, UART є оптимальним вибором.

Програмна підтримка інтерфейсів також відіграє важливу роль. У середовищі Arduino для ESP32 реалізовано стандартні бібліотеки для роботи з усіма основними інтерфейсами. Wire.h використовується для I2C, SPI.h — для SPI, а Serial — для UART. Це спрощує процес налаштування зв'язку, оскільки більшість бібліотек для

сенсорів вже мають готові методи ініціалізації та зчитування даних. Розробнику достатньо вказати відповідні виводи й параметри передачі, щоб почати отримувати інформацію від пристроїв.

Важливим аспектом є й електрична сумісність. ESP32 працює з рівнем логіки 3.3 В, тоді як частина сенсорів розрахована на 5 В. У таких випадках необхідно використовувати понижуючі логічні перетворювачі, які захищають мікроконтролер від надмірної напруги. Ігнорування цієї вимоги може призвести до пошкодження обладнання або нестабільної роботи системи. Застосування таких перетворювачів часто передбачене в модульних версіях сенсорів, однак у складніших системах інженер повинен окремо подбати про це.

Під час створення IoT-системи важливо також враховувати питання конфліктів між пристроями. Наприклад, деякі сенсори I2C мають фіксовану адресу і не можуть працювати на одній шині з ідентичним пристроєм без мультиплексора або використання іншого інтерфейсу. Це потребує планування ще на етапі вибору компонентів. У свою чергу, SPI дозволяє уникнути таких конфліктів через індивідуальні лінії вибору пристрою, але обмежує кількість доступних портів у мікроконтролері.

Загалом вибір інтерфейсу залежить від низки факторів — кількості підключених пристроїв, необхідної швидкості обміну, точності вимірювань, вимог до стабільності роботи та програмної реалізації. У типовій системі моніторингу на базі ESP32 оптимальним часто є поєднання кількох інтерфейсів: наприклад, I²C — для сенсорів температури та освітленості, UART — для газоаналізатора, а аналоговий вхід — для простого термодатчика. Такий підхід дозволяє максимально ефективно використати ресурси мікроконтролера та досягти балансу між точністю, енергоспоживанням і стабільністю системи.

2.3 Програмне опрацювання модулів

2.3.1 Обробка даних модуля для вимірювання температури та атмосферного тиску BMP280

Сенсор BMP280, розроблений компанією Bosch Sensortec, є одним із

найпопулярніших барометричних датчиків для застосування в IoT-проєктах. Його ключові функції полягають у вимірюванні атмосферного тиску та температури з високою точністю та низьким енергоспоживанням. У поєднанні з мікроконтролером ESP32-WROOM, цей модуль може бути використаний для побудови інтелектуальних систем моніторингу навколишнього середовища, зокрема в складі розподілених сенсорних мереж.

Завдяки підтримці SPI та I2C інтерфейсів, BMP280 надає розробникам гнучкість у виборі типу підключення до ESP32-WROOM. У практичному застосуванні частіше використовується саме I2C через його простоту, особливо при підключенні кількох сенсорів до однієї шини. Це дозволяє реалізовувати компактні багатофункціональні рішення.

BMP280 функціонує як цифровий сенсор, який забезпечує вимірювання абсолютного атмосферного тиску, що може бути перетворено на висоту над рівнем моря. Крім того, він має вбудований температурний сенсор, що дозволяє використовувати його і як допоміжний термометр. Ці можливості відкривають широкі перспективи для застосування модуля у метеостанціях, безпілотних літальних апаратах, автоматизованих теплицях тощо.

Для інтеграції BMP280 з ESP32-WROOM необхідно використати відповідні бібліотеки, які реалізують інтерфейс доступу до регістрів пристрою, автоматизуючи процес зчитування даних та їхнього калібрування. Однією з найпопулярніших бібліотек є Adafruit BMP280, яка має високу сумісність з ESP32 і активну спільноту користувачів, використання якої для ініціалізації датчика наведено у лістингу 2.4. Крім того, існують альтернативні бібліотеки, зокрема SparkFun BMP280, які також забезпечують базовий функціонал [31].

Лістинг 2.4 — Приклад ініціалізації BMP280 у коді ESP32.

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
Adafruit_BMP280 bmp;
void setup() {
  Serial.begin(115200);
  delay(1000);
  if (!bmp.begin(0x76)){
```

```
  Serial.println("Помилка: не знайдено BMP280!");
  while (1);
 }
 bmp.setSampling(
  Adafruit_BMP280::MODE_NORMAL,
  Adafruit_BMP280::SAMPLING_X2,
  Adafruit_BMP280::SAMPLING-X16,
  Adafruit_BMP280::FILTER_X16,
  Adafruit_BMP280::STANDBY_MS_500
 );
 Serial.println("BMP280 ініціалізовано");}
```

Після успішного підключення сенсора до ESP32-WROOM, наступним етапом є обробка зчитаних даних. BMP280 надає дані у вигляді цифрових значень, які представляють температуру в сотих частках градуса Цельсія та тиск у Паскалях. Ці значення потребують обробки для приведення до зрозумілої інтерпретації (лістинг 2.5).

Наприклад, тиск, що зчитується у Паскалях, зазвичай перетворюється у гектопаскалі (гПа) шляхом поділу на 100. Це значення може бути використане для розрахунку висоти над рівнем моря, застосовуючи барометричну формулу. Щодо температури, її можна використовувати як окремий параметр для оцінки мікроклімату в досліджуваній зоні.

Лістинг 2.5 — Зчитування та обробка даних з ініціалізованого датчика BMP280

```
void loop() {
  float temperature = bmp.readTemperature();
  float pressure bmp.readPressure() / 100.0;

  Serial.print("Температура: ");
  Serial.print(temperature);
  Serial.print(" C | Тиск: ");
  Serial.print(pressure);
  Serial.println(" hPa");
  delay(2000);
}
```

Особливу увагу слід приділяти налаштуванням режиму роботи BMP280. Датчик підтримує кілька режимів: sleep, forced та normal. У режимі normal датчик

здійснює безперервні вимірювання, що є зручним для систем моніторингу в реальному часі. Проте у випадках, коли важливо зекономити енергію, можна переходити в режим forced, в якому сенсор активується лише на час вимірювання.

BMP280 дозволяє змінювати частоту вимірювань та параметри внутрішнього цифрового фільтру, що є критично важливим для усунення шумів при змінних умовах середовища. ESP32-WROOM, завдяки своїм високопродуктивним ядрам, здатен обробляти навіть великі масиви даних з високою частотою оновлення.

При виборі параметрів конфігурації важливо балансувати між точністю, часом відгуку та енергоспоживанням. Для стаціонарних систем з живленням від мережі можна встановити максимальну частоту вимірювання, тоді як у мобільних або автономних рішеннях доцільно зменшити частоту зчитування до 1 разу на кілька секунд. Приклад задання параметрів датчика при ініціалізації наведено у лістингу 2.6.

Лістинг 2.6 — Зміна параметрів датчика BMP280 при ініціалізації

```
bmp.setSampling(
  Adafruit_BMP280::MODE_NORMAL,
  Adafruit_BMP280::SAMPLING_X2,
  Adafruit_BMP280::SAMPLING-X16,
  Adafruit_BMP280::FILTER_X16,
  Adafruit_BMP280::STANDBY_MS_500
);
Serial.println("BMP280 ініціалізовано із новими параметрами");
```

Завдяки підтримці FreeRTOS, ESP32-WROOM дозволяє обробляти дані з BMP280 у виділеному задачному потоці. Це дає змогу уникнути блокування основного циклу програми при виконанні обчислень чи очікуванні відповідей від сенсора. Оброблені дані можуть бути передані далі — наприклад, збережені у внутрішню пам’ять, надіслані через MQTT або HTTP, або виведені на екран користувача. Приклад обробки BMP280 у виділеному задачному потоці наведено у лістингу 2.7.

Лістинг 2.7 — Обробка BMP280 у виділеному задачному потоці

```
xTaskCreatePinnedToCore(
   bmp280Task,
```

```
    "BMP280 Reader",
    4096,
    NULL,
    1,
    NULL,
    1
  );
void loop() {
  delay(2000);
}
void bmp280Task(void *parameter){
  while (true) {
    float temperature = bmp.readTemperature();
    float pressure bmp.readPressure() / 100.0;
    Serial.print("Температура: ");
    Serial.print(temperature);
    Serial.print(" C | Тиск: ");
    Serial.print(pressure);
    Serial.println(" hPa");
    vTaskDelay(pdMS_TO_TICKS(5000));
  }}
```

Одним із важливих аспектів є правильне керування таймерами та затримками. Так як BMP280 має певну затримку при обробці запиту на вимірювання (в межах кількох мілісекунд), необхідно дотримуватись рекомендованих інтервалів між зверненнями. Це гарантує отримання точних та стабільних результатів.

Зібрані дані про температуру і атмосферний тиск не мають практичного сенсу, якщо їх неможливо обробити чи передати далі для збереження або візуалізації. Саме тому однією з важливих частин обробки даних з BMP280 є формування цих даних у зручному для транспортування форматі — таким, зокрема, є формат JSON (JavaScript Object Notation). Його використання є стандартом у сучасних IoT-системах, оскільки JSON легко парситься як на клієнтських інтерфейсах (наприклад, у вебзастосунках), так і на серверній частині (Node.js, Python, Java тощо).

Для формування JSON-рядка на ESP32-WROOM часто використовують стандартні методи Arduino (через String) або більш структуровані бібліотеки, такі як ArduinoJson, яка дозволяє створювати вкладені структури, масиви та передбачає

перевірку помилок при парсингу (див. лістинг 2.3).

Такий формат зручно передавати за допомогою HTTP або MQTT протоколів на сервер або інший пристрій для подальшої обробки.

ESP32-WROOM підтримує широкий спектр бездротових інтерфейсів для передачі даних — зокрема, Wi-Fi, Bluetooth та BLE. Найчастіше в IoT-проєктах використовується Wi-Fi, оскільки він дозволяє швидко й надійно передавати дані на сервер у локальній мережі або через Інтернет. Типовим варіантом є реалізація HTTP-клієнта, який відправляє POST-запити з JSON-даними на сервер.

Альтернативним підходом є використання протоколу MQTT (Message Queuing Telemetry Transport). Це легкий протокол публікації/підписки, який ідеально підходить для передачі даних з великої кількості сенсорів. У цьому випадку ESP32-WROOM підключається до MQTT-брокера (наприклад, Mosquitto, HiveMQ), де публікує повідомлення з даними з BMP280 у вигляді JSON.

Для зчитування та обробки даних з BMP280 у середовищі Arduino та ESP-IDF існує кілька бібліотек. Найпоширенішими з них є такі:

— Adafruit BMP280 Library – офіційна бібліотека від Adafruit, підтримує як SPI, так і I2C. Має зрозумілий інтерфейс, приклади та сумісність з Arduino;

— SparkFun BMP280 Library – альтернативна бібліотека з додатковими можливостями для конфігурації режимів роботи;

— Bosch BMP280 Driver – офіційний драйвер від Bosch, рекомендований для використання у промислових рішеннях, але вимагає більше коду для інтеграції.

На ESP32 можна також працювати з BMP280 у середовищі ESP-IDF, однак реалізація є складнішою. Arduino-бібліотеки надають спрощений доступ до сенсора, тому найчастіше використовуються в проектах для дослідницьких або навчальних цілей.

Незважаючи на простоту підключення, при роботі з BMP280 можуть виникати типові помилки, які слід враховувати при розробці системи:

— невірна адреса I2C — модуль BMP280 може мати декілька адрес залежно від конфігурації піна SDO. Якщо використовується невірна адреса, ESP32 не зможе встановити зв’язок;

— переплутані SDA/SCL пін — часто трапляється при ручному підключенні на макетній платі;

— недостатнє живлення — BMP280 потребує стабільного живлення 3.3 В. При просадках живлення можуть виникати збої у зчитуванні;

— конфлікт бібліотек — наприклад, використання одночасно Adafruit BMP280 та BME280 може призводити до конфліктів через однакові назви методів.

Для діагностики можна використовувати функцію Wire.scan() (або сторонні I2C-сканери), яка дозволяє виявити всі підключені пристрої на шині. Також слід перевіряти значення, що повертає функція begin() з бібліотеки BMP280 — у разі невдачі вона повертає false.

Хоча BMP280 є надійним та точним сенсором для вимірювання температури та тиску, у деяких випадках доцільно розглянути інші варіанти:

— BME280 – сенсор того ж виробника, але на відміну від BMP280, має ще й гігрометр (вимірювання вологості). Це зручно для систем, де потрібно комплексно моніторити мікроклімат;

— DHT22 (AM2302**)** – дешевший сенсор температури та вологості. Має меншу точність, повільніший відгук, але є бюджетною альтернативою;

— BMP180 – попередник BMP280, має нижчу точність і більшу похибку, не рекомендується для нових проєктів;

— LPS22HB – альтернативний сенсор тиску від STMicroelectronics, сумісний із I2C, має схожі характеристики.

Основною перевагою BMP280 перед DHT22 є точність вимірювання тиску, менша похибка температури, а також підтримка гнучких режимів роботи. Порівняно з BME280, єдиним недоліком BMP280 є відсутність сенсора вологості. Порівняльні характеристики наведено у таблиці 2.1 Інтеграція BMP280 у систему на основі ESP32-WROOM дозволяє реалізувати точні, надійні та гнучкі рішення для вимірювання атмосферних параметрів. Формат JSON забезпечує зручність у передачі даних на сервер чи до інтерфейсів користувача, а підтримка різних протоколів, такий як Wi-Fi, MQTT, HTTP, дозволяє вибрати оптимальну стратегію комунікації. При цьому вибір бібліотек та усвідомлення можливих помилок

суттєво спрощують процес розробки. Порівняння з іншими сенсорами підтверджує, що BMP280 є оптимальним вибором для задач, де вологість не є критично важливим параметром.

Таблиця 2.1 – Порівняння BMP280 з іншими відомими аналогами

Сенсор	Температура	Тиск	Вологість	Інтерфейс	Точність тиску
BMP280	Наявна	Наявний	Відсутня	I2C/SPI	±1 hPa
BME280	Наявна	Наявний	Наявна	I2C/SPI	±1 hPa
DHT22	Наявна	Відсутній	Наявна	1-Wire	Тиск відсутній
BMP180	Наявна	Наявний	Наявна	I2C	±2 hPa

У більшості практичних застосунків дані з сенсорів, зокрема з BMP280, потрібно зчитувати не один раз, а через визначені інтервали часу. Зчитування з надто великою частотою призводить до перевантаження системи й непотрібного навантаження на мережу. Надто рідке — до втрати важливої інформації.

Такий підхід забезпечує енергоефективність, зменшує обсяг переданих даних і дозволяє організовувати обробку в окремих потоках, не блокуючи основну логіку мікроконтролера.

Також ESP32 дозволяє обробляти асинхронні події, наприклад, через асинхронний вебсервер ESPAsyncWebServer, коли клієнт може ініціювати зчитування даних через HTTP-запит.

Замість миттєвої відправки кожного зчитування до сервера, часто ефективніше тимчасово зберігати дані у буфері, а вже потім передавати «пакетом». Такий підхід використовується в умовах нестабільного з’єднання або обмеженої пропускної здатності мережі.

На ESP32 для цього можна використовувати:

— масиви String або std::vector (якщо використовується C++);

— структури та черги у FreeRTOS;

— збереження даних у SPIFFS або LittleFS, якщо обсяг великий;

— тимчасове кешування в RTC пам’яті між перезавантаженнями.

Формування «батча» з останніх 10 вимірювань дозволяє зменшити кількість

HTTP/MQTT-з’єднань, знижуючи споживання енергії та підвищуючи надійність.

У випадках, коли мережа недоступна, або потрібно вести локальний запис температури й тиску, доречно зберігати дані локально.

Запис у EEPROM або SPIFFS виконується з урахуванням обмеження кількості циклів запису, тому зберігання має бути оптимізованим.

У реальному IoT-проєкті, що базується на ESP32-WROOM, BMP280 використовується як джерело даних про мікроклімат. Сенсор підключається до шини I2C, налаштовується в режимі «Forced mode» (одноразове вимірювання при кожному запиті) або «Normal mode» (періодичне вимірювання за розкладом).

Алгоритм роботи пристрою може бути такий: спочатку відбувається підключення до Wi-Fi, далі виконується перевірка наявності сервера або MQTT-брокера, потім зчитуються дані з BMP280, формується JSON-структура, відбувається передача на сервер, або, у разі відсутості мережі – збереження у файл. Алгоритм викнується циклічно.

У системах моніторингу оточення BMP280 часто комбінується з іншими сенсорами: MQ-135 (якість повітря), BH1750 (освітлення), BME280 (вологість). Дані з усіх сенсорів об’єднуються в один JSON.

2.3.2 Обробка даних модуля контролю рівня освітлення BH1750

Датчик освітленості BH1750 є одним із найпоширеніших датчиків для вимірювання рівня освітлення в системах автоматизації та моніторингу навколишнього середовища. Цей цифровий датчик, що працює по протоколу I2C, дозволяє отримувати точні та стабільні значення освітленості в люксах (lx). Використання BH1750 у поєднанні з потужним та енергоефективним контролером ESP32-WROOM дає змогу створити надійну IoT-систему для моніторингу освітлення у різних середовищах.

Інтерфейс I2C, необхідний для BH1750, підтримується багатьма виводами ESP32, що дає можливість гнучко обрати пін-контакти для підключення.

Типове підключення передбачає з'єднання контактів SDA та SCL датчика з відповідними контактами ESP32, а також подачу живлення 3.3 В та загальний GND.

Важливо дотримуватись рекомендованих рівнів напруги, оскільки BH1750 працює при 3.3 В логіці.

Протокол I2C (Inter-Integrated Circuit) — це синхронний послідовний протокол зв’язку, що використовується для обміну даними між мікроконтролерами та периферійними пристроями. У випадку BH1750 ESP32 виступає як майстер, ініціюючи передачі, а BH1750 — як підлеглий пристрій із фіксованою адресою.

Завдяки широкому поширенню ESP32, для нього розроблено кілька бібліотек, що спрощують роботу з BH1750. Однією з найбільш популярних є BH1750.h, яка реалізує всі необхідні функції для ініціалізації датчика, запуску вимірювань та зчитування результатів у люксах (лістинг 2.8) [32].

Лістинг 2.8 — Приграма використання датчика BH1750

```
#include <Wire.h>
#include <BH1750>
BH1750 lightMeter;
void setup() {
  Serial.begin(115200);
  Wire.begin();
  if(ligthMeter.begin()){
    Serial.println("BH1750 ініціалізовано");
  }
  else {
    Serial.println("Помилка ініціалізації");
  }
}
void loop() {
  float lux = lightMeter.readLightLevel();
  Serial.print("Освітленість : ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

Завдяки цифровому виходу BH1750 не потребує складної обробки сигналу, що прискорює цикл отримання даних.

Цей код ініціалізує I2C, запускає BH1750 у нормальному режимі, і кожну секунду зчитує рівень освітленості, виводячи його у послідовний монітор.

При розробці програмного забезпечення для роботи з BH1750 слід

враховувати кілька важливих аспектів, перерахованих нижче.

Час вимірювання. BH1750 потребує певного часу для формування точного результату (близько 120 мс у стандартному режимі). Період опитування датчика має враховувати це, щоб уникнути отримання некоректних значень.

Використання режимів низького енергоспоживання є критичним для автономних пристроїв, що працюють на батарейках. Змінюючи режим роботи BH1750, можна знизити споживання енергії без суттєвої втрати точності.

Хоч BH1750 й постачається каліброваним, у деяких випадках (наприклад, при особливих оптичних умовах) може знадобитися додаткове калібрування на рівні програмного забезпечення.

Інтеграція з іншими сенсорами**:** Вбудовування BH1750 у систему, що містить інші датчики (температури, вологості, газів), вимагає узгодження частоти опитування та оптимізації логіки обробки для ефективного використання ресурсів ESP32.

2.3.3 Обробка даних газового датчика MQ-135

У процесі побудови апаратно-програмного комплексу для моніторингу навколишнього середовища важливим етапом є забезпечення точного зчитування концентрації газів, зокрема вуглекислого газу. Для реалізації цього завдання використовується газовий сенсор MQ-135, який змінює свій електричний опір залежно від складу повітря. Оскільки модуль MQ-135 формує аналоговий сигнал, що залежить від концентрації різних газів, виникає необхідність у відповідному програмному перетворенні цього сигналу в цифрові значення, які можна інтерпретувати як концентрацію CO_2 в одиницях ppm - частин на мільйон або ж parts per million.

У зв’язку з цим було реалізовано дві допоміжні функції, getPPM та getSmoothedPPM, які дозволяють зчитувати аналоговий сигнал з датчика, перетворювати його у відповідну концентрацію газу, а також згладжувати результати для підвищення точності вимірювання. Опис цих функцій містить як програмну реалізацію, так і математичні формули, що використовуються для

обчислення концентрації вуглекислого газу.

Описані функції getPPM та getSmoothedPPM є ключовими складовими програмного забезпечення для обчислення концентрації вуглекислого газу в повітрі за допомогою датчика MQ-135, підключеного до ESP32-WROOM через подільник напруги. Їхня робота базується на використанні аналогового сигналу, що надходить з датчика, подальшому математичному опрацюванні та застосуванні емпіричної залежності, яка наближено зв'язує зміну опору сенсора з концентрацією CO_2 у повітрі.

У функції getPPM основним параметром є цифрове значення, отримане з АЦП мікроконтролера. Це значення необхідно перетворити у відповідну напругу, яка подавалась на вхід ESP32 після проходження через дільник напруги. Цей етап реалізується за допомогою формули:

$$V_{out} = \frac{ADC}{4095} * 3.3 * 2$$

Коефіцієнт 2 враховує використання симетричного подільника напруги з двох резисторів по 10 кОм. Цей розрахунок дозволяє відновити повне значення напруги на аналоговому виході модуля MQ-135 до того, як воно було поділено.

Далі розраховується опір сенсора Rs, що змінюється у відповідь на концентрацію газу. Формула для Rs базується на відношенні між живленням сенсора та виміряною напругою:

$$R_S = \frac{V_C - V_{out}}{V_{out}} * R_L$$

де Vc = 5.0В — це напруга живлення модуля,

R_L — опір навантаження, що зазвичай дорівнює 10 кОм.

Цей розрахунок використовує закон Ома для обчислення невідомого опору в послідовному з'єднанні сенсора з резистором R_L.

Знаючи значення Rs і маючи попередньо відкаліброване значення R0 (опору сенсора в чистому повітрі), можна знайти їх відношення:

$$\frac{R_S}{R_0}$$

Це відношення потім підставляється у степеневу функцію, яка апроксимує залежність між концентрацією газу та опором сенсора на основі характеристик, взятих із технічної документації MQ-135. Формула набуває вигляду:

$$ppm = 116.6020682 * (\frac{R_S}{R_0})^{-2.769034857}$$

Цей вираз є результатом логарифмічного спрощення, яке MQ-135 використовує для опису вмісту CO2 у ppm.

Щоб забезпечити більшу стабільність вимірювання, реалізована функція getSmoothedPPM, яка повторює зчитування напруги з аналогового входу кілька разів і обчислює середнє значення ppm. Зчитування повторюються з невеликою затримкою, що дозволяє уникнути миттєвих шумів. Основна ідея цієї функції полягає в тому, щоб багаторазово викликати getPPM і накопичувати результати у змінну sumPPM, після чого повертати середнє значення за допомогою формули:

$$ppm_{\text{сер.}} = \frac{1}{N}\sum_{i=1}^{N} ppm_i,$$

де N— кількість вимірювань.

Цей метод дозволяє отримати згладжене значення концентрації, придатне для візуалізації або передачі в інші частини системи.

Таким чином, ці дві функції в комплексі реалізують надійний і математично обґрунтований підхід до отримання концентрації CO_2 у повітрі, використовуючи

тільки можливості аналогового входу мікроконтролера та знання про електричні характеристики датчика.

2.4 Надсилання даних у вебзастосунок у форматі JSON

Однією з ключових задач при розробці IoT-систем є забезпечення надійної та безпечної передачі даних із пристроїв, що знаходяться у локальній мережі, до віддалених серверів або веб-додатків. У випадку використання мікроконтролера ESP32, який зазвичай підключається до інтернету через домашній або офісний роутер із NAT (Network Address Translation), безпосередній доступ до нього ззовні ускладнюється відсутністю публічної IP-адреси та необхідністю налаштовувати переадресацію портів на маршрутизаторі.

Для подолання цих обмежень застосовують технології тунелювання — створення віртуальних каналів між локальною мережею та глобальним інтернетом. У рамках цього проєкту обрана технологія NGROK, яка надає простий і гнучкий інструмент для створення безпечних тунелів з публічною URL-адресою до пристрою, що працює у локальній мережі.

NGROK — це сервіс, який встановлює вихідне з'єднання від локального пристрою (у цьому випадку — ПК, що має доступ до ESP32) до свого хмарного сервера. Цей сервер, у свою чергу, надає публічну адресу, через яку можна отримати доступ до локального сервера або пристрою. За рахунок цього створюється зворотній тунель, що обходить NAT і firewall, і забезпечує зручний спосіб доступу до пристрою без складних налаштувань маршрутизатора.

Для роботи NGROK необхідно виконати декілька простих кроків. Спочатку необхідно зареєструватися на офіційному сайті сервісу і завантажити клієнтську програму. Наступним кроком стане запуск NGROK на локальному комп'ютері із вказаним порто ESP32, який приймає HTTP-запити. Далі відбувається отримання публічного URL, наприклад «https://a1b2c3d4.ngrok.io», який буде слугувати точкою доступу до ESP32 з мережі Інтернет.

Таким чином, замість прямого підключення до локальної IP-адреси ESP32 через домашню мережу, веб-додаток або будь-який інший клієнт може надсилати

HTTP-запити на публічний URL NGROK, а сервіс переадресує їх до ESP32.

Для інтеграції ESP32 з тунелем NGROK було виконано кілька ключових кроків.

Перший крок полягає у підключенні ESP32 до Wi-Fi мережі з внутрішньою IP-адресою, призначеною маршрутизатором. У прошивці налаштовано веб-сервер, який працює на певному порту.

У ході другого кроку NGROK запускається на комп'ютері, який має доступ до ESP32 у тій же локальній мережі.

Команда запуску NGROK для HTTP-тунелю з використанням зарезервованого у командному рядку комп'ютера домену для адреси ESP32 наведена у лістингу 2.9.

Лістинг 2.9 — Команда для запуску NGROK

```
http --url=esp32environment.ngrok.app 192.168.68.111
```

Це створює тунель, що здійснює пересилання трафіку з публічної адреси NGROK на локальну адресу плати.

Третій крок полягає у триманні публічної адреси. Після запуску NGROK у консолі відображається унікальна публічна URL-адреса виду «https://esp32environment.ngrok.app», яка доступна з мережі Інтернет.

Наступним, четвертим кроком, є конфігкрація самої плати ESP32. У попередніх розділах описано програмування ESP32 таким чином, що дані з датчиків формуються у форматі JSON і надсилаються через HTTP-запити на сервер із публічною адресою NGROK. Веб-додаток, у свою чергу, отримує ці дані, виконуючи запити на цю адресу.

Для перевірки успішності, після запуску NGROK потрібно у пошуковій адресі браузера ввести адресу «https://esp32environment.ngrok.app» та додати до неї «/api/data» для імітування запиту з додатку. Повний вид адреси для перевірки «https://esp32environment.ngrok.app/api/data». У результаті повинна відобразитись сторінка, що містить данні, передані з плати, що свідчить про успішне налагодження зв'язку (рис. 2.1). Для правильності перевірки, пристрій, з якого

виконується запит, повинен бути під’єднаний до іншої мережі, окремої від тієї, в якій знаходиться ESP32.

Основною перевагою використання NGROK є простота налаштування — не потрібно змінювати конфігурацію маршрутизатора або мати публічну IP-адресу. Крім того, створений тунель є безпечним, оскільки NGROK шифрує трафік HTTPS, що підвищує захист даних. Також до переваг NGROK можна віднести мобільність — тунель можна запускати на будь-якому пристрої, де доступна клієнтська програма NGROK, а також швидкий початок роботи — сервіс надає готові URL, що спрощує доступ до пристрою.

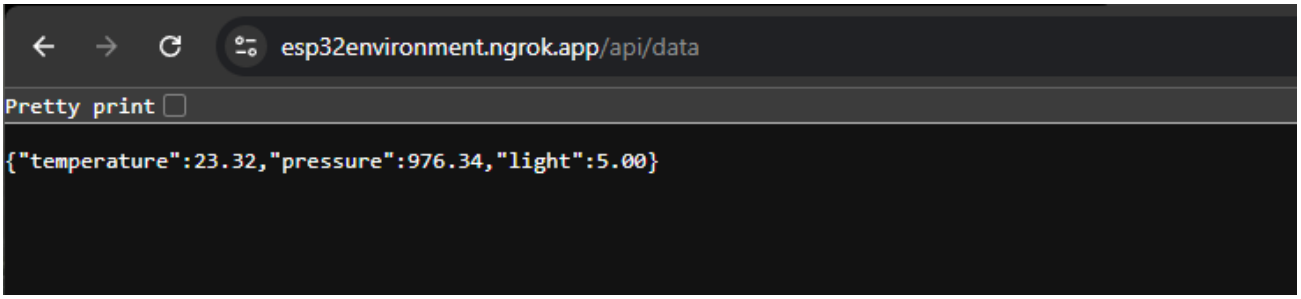

Рисунок 2.1 — Результат виконання запиту

Використання NGROK має також і свої недоліки, серед яких:

— обмежений час сесії у безкоштовній версії — тунелі часто мають часові обмеження і змінюють URL при перезапуску;

— залежність від зовнішнього сервісу — при відсутності інтернету або проблемах на сервері NGROK зв’язок буде недоступним;

— безкоштовна версія не підтримує фіксовані URL та більші об’єми трафіку.

Хоча NGROK є зручним і популярним рішенням, існують й інші методи для забезпечення доступу до ESP32 з інтернету.

Одним з таких методів є метод переадресації портів (Port Forwarding). Цей метод передбачає налаштування маршрутизатора для перенаправлення запитів з певного зовнішнього порту на внутрішню IP-адресу ESP32 і відповідний порт. Перевагами такого способу, у порівнянні з NGROK, є відсутність додаткових сервісів та повний контроль над підключеннями. Головними ж недоліками є

потреба доступу до конфігурації роутера, складність у разі динамічної IP-адреси та ризики безпеки через відкриті порти.

Ще одним методом є використання динамічного DNS, або ж DDNS. Це дозволяє отримати постійну доменну адресу для динамічної IP-адреси інтернет-з'єднання. У комбінації з переадресацією портів це дає можливість стабільно звертатися до ESP32. Основною перевагою є автоматична синхронізація IP-адреси, а недоліками – потреба в додатковому налаштуванні та можливі затримки оновлення.

Можливим варіантом також є використання власного VPN-сервера. Створення віртуальної приватної мережі VPN дозволяє безпечно підключатися до локальної мережі, у якій знаходиться ESP32.

Із переваг такого методу: високий рівень безпеки, можливість доступу до кількох пристроїв.

Недоліки: складність налаштування, потреба у сервері VPN, більші вимоги до ресурсів.

Замість прямого HTTP-запиту ESP32 може надсилати дані на публічний MQTT-брокер (наприклад, Mosquitto, HiveMQ), а веб-додаток — підписуватись на відповідні теми. Перевагами використання даного методу можна назвати легкість масштабування, низьку затримку та підтримку QoS.

Серед недоліків ж потреба у додатковій інфраструктурі (брокері) та складність інтеграції з HTTP-запитами.

У таблиці 2.2 наведено порівняльну характеристику методів забезпечення доступу до локальної адреси ESP32 з мережі Інтернет.

Існують альтернативні сервіси, схожі на NGROK, такі як LocalTunnel, Serveo, Pagekite, які також надають можливість швидко отримати публічний доступ до локальних сервісів.

Їхніми перевагами перед NGROK є безкоштовність та простота використання, але, попри це, вони менш стабільні. Використання NGROK у проєкті дозволило швидко і без зайвих складних налаштувань організувати віддалений доступ до ESP32, що є важливим для ефективної роботи IoT-системи. Хоча NGROK

має деякі обмеження, його переваги роблять його оптимальним варіантом для розробки та тестування.

Для промислового впровадження та масштабування системи варто розглянути інші методи, такі як VPN або MQTT, які забезпечують більш стабільне і безпечне середовище. Вибір конкретного способу залежить від технічних вимог, бюджету та складності проєкту.

Таблиця 2.2 - Порівняльна характеристика методів забезпечення доступу до локальної адреси

Метод	Простота налаштування	Безпека	Надійність	Вартість	Особливості
NGROK	Дуже проста	Висока (HTTPS)	Залежить від сервісу	Безкоштовно/платно	Тимчасові URL у безкоштовній версії
Port Forwarding	Середня	Низька	Висока	Безкоштовно	Відкриття портів, можливі ризики
DDNS + Port Forwarding	Середня	Низька	Середня	Безкоштовно/платно	Потрібен додатковий сервіс DDNS
VPN	Складна	Дуже висока	Висока	Власна інфраструктура	Потрібен сервер VPN
MQTT	Середня	Висока	Висока	Залежить від брокера	Потрібен MQTT-брокер
Інші тунелі (LocalTunnel, Serveo)	Дуже проста	Середня	Нижча	Безкоштовно	Менша підтримка, нестабільність

.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Огляд використаних бібліотек

У процесі розробки програмного забезпечення для мікроконтролерів, зокрема для платформи ESP32, важливу роль відіграє використання готових бібліотек. Бібліотеки значною мірою спрощують доступ до апаратних ресурсів та реалізацію функціональних можливостей, оскільки надають розробнику вже реалізовані програмні інтерфейси для взаємодії з периферійними пристроями, мережевими модулями, сенсорами тощо. Завдяки цьому значно зменшується обсяг ручного кодування, прискорюється розробка, а також знижується ймовірність помилок, пов’язаних із низькорівневим програмуванням.

Особливо актуальним є використання бібліотек у проєктах на базі IoT, де ESP32 виступає як центральний вузол системи збору, обробки та передавання даних. У таких системах часто одночасно використовуються декілька типів датчиків, мережеві інтерфейси, а також протоколи передачі даних, що потребує застосування цілого набору бібліотек, які дозволяють реалізувати зазначену функціональність на високому рівні абстракції.

У даному розділі розглядається доцільність використання бібліотек у скетчі для ESP32, визначаються їх основні функції, принципи підключення та взаємодії з основною програмною логікою. Особливу увагу приділено бібліотекам для роботи з датчиками навколишнього середовища, обміну даними через мережеві протоколи та формування структури даних у форматі JSON. Аналізована роль бібліотек не лише з погляду технічної реалізації, але й як інструменту оптимізації розробницького процесу в умовах обмежених апаратних ресурсів мікроконтролера.

Бібліотека Wire.h є стандартною бібліотекою Arduino для роботи з протоколом I2C (Inter-Integrated Circuit). Вона забезпечує обмін даними між мікроконтролером, таким як ESP32, і зовнішніми цифровими пристроями — наприклад, датчиками, енергонезалежною пам’яттю, дисплеями тощо. Wire.h дозволяє мікроконтролеру працювати як головний (master) або підлеглий (slave) пристрій, реалізуючи функції ініціалізації шини, надсилання та прийому даних. Для використання бібліотеки необхідно підключити її в початку програми, після

чого викликаються методи на кшталт begin(), beginTransmission(), write() і endTransmission(). Бібліотека широко підтримується та є сумісною з більшістю популярних пристроїв, що використовують I2C, завдяки чому вона є важливою складовою програмного забезпечення для IoT-систем на базі ESP32 [34].

Бібліотека Adafruit_BMP280.h є засобом для взаємодії з цифровим сенсором атмосферного тиску і температури BMP280, розробленим компанією Bosch. Вона створена компанією Adafruit і надає зручний інтерфейс для ініціалізації сенсора, зчитування вимірювань та налаштування режимів його роботи. Завдяки підтримці як I2C, так і SPI інтерфейсів, бібліотека легко адаптується до різних апаратних платформ, зокрема до мікроконтролерів на базі Arduino та ESP32.

Однією з особливостей бібліотеки є те, що вона побудована на базі абстрактного класу Adafruit_Sensor, що забезпечує сумісність з іншими сенсорами в межах екосистеми Adafruit. Це спрощує інтеграцію кількох сенсорів у єдиній системі збору даних, особливо у проектах екологічного моніторингу або Інтернету речей. Вона дозволяє точно зчитувати температуру та атмосферний тиск, а також конфігурувати параметри роботи сенсора залежно від вимог конкретного застосування.

Бібліотека активно підтримується спільнотою розробників і має відкритий вихідний код. Це сприяє її широкому використанню у проектах з відкритим апаратним забезпеченням. Простота у використанні, стабільність роботи та сумісність із популярними платформами роблять Adafruit_BMP280.h одним із найпоширеніших інструментів для роботи з BMP280 у сфері любительської та професійної електроніки [35].

Бібліотека BH1750.h створена для спрощення роботи з датчиком освітленості BH1750, який вимірює рівень освітлення у люксах. Вона забезпечує зручний інтерфейс для ініціалізації датчика, налаштування режимів вимірювання і зчитування значень освітленості. Завдяки цій бібліотеці можна легко інтегрувати BH1750 у проекти на базі мікроконтролерів, таких як ESP32 або Arduino, без необхідності розбиратися у низькорівневих деталях протоколу I2C. Вона автоматично обробляє передачу команд і отримання даних, що значно спрощує код

і прискорює розробку. Крім того, бібліотека підтримує різні режими роботи датчика, що дозволяє адаптувати виміри до умов освітлення та оптимізувати енергоспоживання. Використання BH1750.h широко поширене у системах моніторингу освітленості, автоматизації освітлення та інших IoT-проектах, де важливим є точне та швидке визначення рівня світла [37].

Бібліотека WiFi.h є основним інструментом для роботи з бездротовими мережами на платформах, таких як ESP32 та ESP8266. Вона забезпечує простий і зручний інтерфейс для підключення пристрою до Wi-Fi мережі, управління з’єднанням, а також отримання інформації про стан мережі. Використання цієї бібліотеки значно спрощує процес інтеграції пристроїв у мережеве середовище, дозволяючи розробникам швидко створювати проекти з підтримкою бездротового зв’язку. Крім того, WiFi.h містить функції, які допомагають управляти як клієнтськими, так і точковими мережами, що робить її універсальним рішенням для багатьох IoT-застосунків. Завдяки своїй стабільності та широкій підтримці, ця бібліотека стала невід’ємною частиною програмування мікроконтролерів із Wi-Fi модулем [38].

Бібліотека WebServer.h є важливою складовою для створення веб-сервера на платформах, що базуються на мікроконтролерах, зокрема ESP32. Вона забезпечує простий і зручний спосіб організації обробки HTTP-запитів, дозволяючи приймати запити від клієнтів та відповідати на них без складної конфігурації. Використання цієї бібліотеки дає можливість легко реалізувати веб-інтерфейс для управління пристроєм або отримання даних з датчиків у режимі реального часу. Вона працює у тісній інтеграції з мережею Wi-Fi, що дозволяє створювати локальні чи навіть віддалені сервіси, де мікроконтролер виступає в ролі сервера. Завдяки WebServer.h значно спрощується процес розробки IoT-проєктів, де потрібна взаємодія з користувачем через браузер або інші клієнтські додатки [39].

Бібліотека WiFiClientSecure.h використовується для створення захищених мережевих з'єднань на базі ESP32 та інших мікроконтролерів, що підтримують Wi-Fi. Вона дозволяє встановлювати захищені TLS або SSL з’єднання, що забезпечує шифрування переданих даних і підвищує безпеку комунікації з серверами. Це

особливо важливо при передачі чутливої інформації через інтернет, адже бібліотека допомагає уникнути перехоплення чи підробки даних. WiFiClientSecure активно застосовується для роботи з веб-сервісами, API та іншими ресурсами, які вимагають безпечного каналу зв’язку, інтегруючи в себе функції сертифікатів та криптографічних протоколів. Таким чином, вона є ключовим інструментом для реалізації надійного захисту інформації у проектах на базі ESP32 [40].

Бібліотека WiFiManager.h широко використовується в проектах на базі ESP8266 та ESP32 для спрощення процесу підключення до Wi-Fi мережі. Вона дозволяє пристрою автоматично створювати точку доступу, якщо він не може підключитися до раніше збереженої мережі, що значно полегшує налаштування без необхідності перепрограмування. Через веб-інтерфейс користувач може вибрати потрібну мережу та ввести пароль, після чого пристрій автоматично підключиться до неї. Такий підхід робить використання ESP-модулів більш зручним, особливо для кінцевих користувачів, і допомагає уникнути складнощів при зміні умов мережі або при першому запуску пристрою. Бібліотека інтегрується з основними мережевими функціями ESP, надаючи простий і надійний спосіб управління підключенням без необхідності глибоких знань про роботу Wi-Fi [41].

Бібліотека UniversalTelegramBot.h створена для спрощення роботи з Telegram Bot API на мікроконтролерах, таких як ESP8266 або ESP32. Вона дозволяє легко відправляти та отримувати повідомлення через Telegram, реалізуючи взаємодію користувача з пристроєм за допомогою популярного месенджера. Ця бібліотека автоматично обробляє комунікацію з сервером Telegram, приховуючи складнощі мережевих запитів і форматування даних. Завдяки їй розробник може швидко інтегрувати функції віддаленого керування або моніторингу, що значно розширює можливості IoT-проектів. UniversalTelegramBot.h підтримує різні типи повідомлень і надає простий інтерфейс для налаштування та використання, що робить її зручною для початківців і досвідчених програмістів [42].

3.2 Компонування обробки даних з усіх датчиків у єдиній програмі мікроконтролера

Для працездатності системи та виконання поставлених перед нею завдань, необхідно створити єдиний код прошивки центрального модуля, яким є ESP32-WROOM. Цей код має поєднувати у собі забезпечення можливості під'єднання плати ESP32-WROOM до локальної мережі без використання послідовного порту та постійного перепрошивання самої плати, збору та обробки даних з датчиків, підготовки їх до передачі у вебзастосунок у форматі JSON, а також повідомлення про локальну адресу плати у під'єднаній мережі за допомогою Telegram-бота для забезпечення переадресації цієї адреси за допомогою NGROK.

Повний варіант коду наведено у додатку додатку Д цієї роботи.

Першим і найважливішим етапом є забезпечення підключення плати до локальної мережі. Найпростішим способом є використання методів бібліотеки Wifi.h. Але цей метод має один суттєвий недолік, а саме потребу у жорсткому заданні даних про мережу, до якої здійснюється приєднання, що, в свою чергу, змушує здійснювати перепрошивання і внесення змін у код для здійснення підключення до іншої мережі.

Для усунення наведених вище незручностей у ході розробки програмного забезпечення для плати ESP32-WROOM, було прийнято рішення про використання бібліотеки WifiManager. Цей метод дозволяє виконувати підключення до будь-якої мережі без потреби у перепрошиванні. Працює це наступним чином. У коді викликається команда .autoConnect("ESP32_Env_Cont-Setup"), яка здійснює прийднання до мережі, інформація про яку зберігається у пам'яті плати. У випадку невдачі, ця функція змушує плату запустити власну точку доступу з назвою ESP32_Env_Cont-Setup, яка дозволяє внести дані про мережу та виконати автоматичне під'єднання (лістинг 3.1).

Лістинг 3.1 – Використання бібліотеки WiFiManager

```
#include <WiFiManager.h>
void setup() {
  Serial.begin(115200);
```

```
  WiFiManager wm;
  bool res = wm.autoConnect("ESP32-Setup");
  if (!res) {
    Serial.println("Не вдалося підключитися до WiFi");
  } else {
    Serial.println("Підключено!");
    Serial.println(WiFi.localIP());
  }
}
void loop() {
}
```

Наступним завданням коду прошивки, після налагодження з'єднання з мережею, є забезпечення повідомлення про адресу плати ESP32-WROOM з приєднаними до неї датчиками у локальній мережі. Звичайним рішенням такого завдання є виведення адреси у монітор послідовного порту за допомогою команди Serial.print(WiFi.local.IP). Але, задля зручності використання системи у випадку потреби частого перепідключення до нових мереж, необхідно забезпечити відображення адреси без застосування дротового з'єднання з комп'ютером чи іншим пристроєм для читання послідовного порту плати. Для виконання цього завдання було обрано спосіб, за якого інформація про адресу плати надсилається у Telegram – бота одразу після того, як плата успішно приєднується до мережі.

Для реалізації надсилання платою інформації про свою адресу у локальній мережі, у коді використанно наступні поля з типом даних char*, наведені у лістингу 3.2.

Лістинг 3.2 – Поля для функціонування зв'язку з Telegram

```
const char* BOTtoken =
"8024695514:AAF9TMQFZtpPOtiW2LgS9KhkRrgmzmVJm04";
const char* CHAT_ID = "799798585";
```

У поле з назвою BOTtoken записується токен Telegram-бота, у який буде надсилатись інформація про адресу

У полі CHAT_ID записується індивідуальний ідентифікатор користувача Telegram, якому буде здійснюватись надсилання адреси за допомогою бота.

Далі значення цих полів використовуються у функції sendTelegramMessage(),

яка забезпечує надсилання повідомлення, та у якості аргумента приймає саме повідомлення у форматі даних String. Зміст функції наведено у лістингу 3.3.

Лістинг 3.3 – Зміст функції sendTelegramMessage()

```
void sendTelegramMessage(String message) {
  client.setInsecure();
  if (client.connect("api.telegram.org", 443)) {
    String url = "/bot" + String(BOTtoken) + "/sendMessage?chat_id=" + CHAT_ID
+ "&text=" + message;
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: api.telegram.org\r\n" +
                 "Connection: close\r\n\r\n");
    while (client.connected() || client.available()) {
      String line = client.readStringUntil('\n');
      Serial.println(line);
    }
  } else {
    Serial.println("Помилка підключення до Telegram");
  }
}
```

У функції setup(), піля успішного приєднання плати до мережі, відбувається надсилання адреси, отриманої в результаті виконання WiFi.localIP() (лістинг 3.4).

Лістинг 3.4 – Надсилання адреси плати у Telegram

```
String ipMessage = "ESP32_Env IP: " + WiFi.localIP().toString();
sendTelegramMessage(ipMessage);
```

Наступним етапом роботи програмного забезпечення, або ж прошивки плати, стає обробка даних з датчиків та формування JSON – рядка для подальшої передачі його у вигляді відповіді на HTTP – запит, отриманий з веб-додатка на адресу плати з використанням тунелю NGROK.

Першим кроком реалізації цього етапу є ініціалізація шини I2C за допомогою функції Wire.begin(), яка, у випадку відсутності переданих аргументів, ініціалізує шину з стандартними пінами SDA та SCL, для плати ESP32-WROOM це GPIO21 та GPIO22 відповідно. Крім стандартних параметрів, функція може приямати на вхід три аргументи, якими є користувацькі піни SDA та SCL, а також швидкість здійснення зв’язку.

Після виконання ініціалізації шини немає ніяких обмежень для початку роботи з датчиками, які використовують цю шину. У апаратно-програмному комплексі, для якого створюється описане у цій роботі програмне забезпечення, використовуються два таких датчики – BMP280 для вимірювання температури та атмосферного тиску та BH1750 для вимірювання рівня освітлення. Обробка даних у коді з цих датчиків вимагає їхньої ініціалізації, для здійснення якої у функції setup() прописано код, наведений у лістингу 3.5.

Лістинг 3.5 – Ініціалізація датчиків

```
bmp.begin(0x76);
lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE);
```

Цей код описує ініціалізацію датчиків BMP280 із заданням його I2C-адреси за допомогою виклику функції bmp.begin(0x76), де 0x76 – опис I2C-адреси, та BH1750 із встановленням режиму його роботи за допомогою виклику функції lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE), де переданий у дужках аргумент описує режим роботи датчика – безперервне вимірювання з високою роздільною здатністю.

Обробка даних, отриманих з датчика MQ-135 полягає у обробці аналогового сигналу, отриманого з датчика. Обробка цього сигналу здійснюється за допомогою функції getPPM(int raw_adc) та getSmoothedPPM(), розробка який описана пункті 2.4 розділу 2 цієї роботи.

Далі необхідно забезпечити передачу JSON – рядка з даними, отриманими в результаті обробки даних з датчиків, у якості відповіді на HTTP – запит, що надходитиме з веб-додатку на адресу плати. Для було використано бібліотеку WebServer.h, яка дозволяє створити вбудований веб-сервер для прийому HTTP-запитів з браузера та їх обробку. Обробка здійснюється шляхом використання у коді setup() функції server.on(), зміст якої наведено у лістингу 3.6

Лістинг 3.6 – Зміст функції server.on()

```
server.on("/api/data", HTTP_GET, []() {
   String json = "{";
```

```
  json += "\"temperature\":" + String(temperature) + ",";
  json += "\"pressure\":" + String(pressure) + ",";
  json += "\"light\":" + String(light) + ",";
  json += "\"ppm\":" + String(ppm);
  json += "}";
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.send(200, "application/json", json);
 });
```

Аргументи, що отримує викликана функція, описують шлях отримання запитів для обробника GET-запитів, тобто «/api/data».

Далі відбувається зчитування даних з датчиків та їх обробка шляхом застосування функцій та присвоєння отриманих значеннь глобальним змінним типу float.

Наступним кроком є формування JSON рядка з даними, отриманими від датчиків, та його запис у змінну типу String.

Далі формується відповідь на HTTP-запит за допомогою двох функцій.

Функція server.sendHeader("Access-Control-Allow-Origin", "*") надсилає заголовок «Access-Control-Allow-Origin: *» для надання дозволу для CORS-доступу з інших доменів.

Функція server.send(200, "application/json", json) здійснює надсилання основного повідомлення, до складу якого входить статус «200 OK», що є стандартним кодом відповіді HTTP, який означає успішне опрацювання запиту сервером, тип контенту, що передається, та створений раніше JSON-рядок з даними, який записано у змінну json з типом String.

Зрештою, відбувається запуск веб-сервера для обробки запитів за допомогою функції server.begin();

У тілі циклічної функції loop() відбувається виклик функції server.handleClient() для обробки запитів, що надходять, за допомогою описаної раніше функції server.on().

З огляду на те, що дана бакалаврська кваліфікаційна робота є комплексною, створене програмне забезпечення для подальшого тестування його працездатності було завантажено на завчасно зібрану апаратну систему, проектування якої

описано у частині першій комплексної бакалаврської кваліфікаційної роботи, яка має тему «Апаратно-програмний комплекс моніторингу стану навколишнього середовища на базі IoT-засобів. Частина перша. Апаратне забезпечення».

4 ТЕСТУВАННЯ ПРАЦЕЗДАТНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування коректного зчитування даних з датчиків

Для виконання покрокової перевірки працездатності апаратно-програмного комплексу, розробка програмного забезпечення якого описана у цій роботі, необхідно виконати декілька етапів перевірки, перший з яких – перевірка коректності читання та обробки даних з датчиків, які входять до складу АПК. Для цього необхідно під’єднати АПК, а саме роз’єм USB плати ESP32-WROOM до комп’ютера за допомогою дротового з’єднання для отримання можливості читання пповідомлень, що надходять до комп’ютера через послідовний порт плати. У коді прошивки, необхідно скористатись функцією Serial.print(), якій у якості аргументів потрібно передати змінні, що використовуються для зберігання поточних результатів обробки сигналів з датчиків. Також, можна використати наявну у програмному забезпеченні користувацьку функцію sensorsSerialWrite(), створену для забезпечення зручного виведення даних з наявних датчиків у послідовний порт, що і було зроблено у ході перевірки.

Використання даного методу дає можливість зрозуміти, чи відбувається зчитування даних з усіх датчиків, що входять до апаратного забезпечення комплексу та чи коректними є ці данні. Це дає змогу виявити можливі проблеми, наприклад, відсутність даних з одного з датчиків, а також неточності вихідних даних.

Для перевірки коректності показників, отриманих в результаті обробки даних з датчика BMP280, було використанно кімнатний термометр, для порівняння значень температури.

У результаті перевірки було виявлено різницю у 0.6 градуса Цельсію, що є результатом нормальної обробки даних з датчика, зважаючи на можливі неточності у роботі самого датчика BMP280.

Для перевірки даних про атмосферний тиск, було взято значення поточного атмосферного тиску у конкретному регіоні з пошукової системи Google. У результаті отримано значення у 1006 гПа, що є досить близьким до значення,

отриманого в результаті обробки даних з датчика BH1750, а саме 983гПа.

Для перевірки результатів обробки даних з датчика BH1750 було використано дані стосовно рівнів освітлення у одиницях вимірювання lx (люкс) для різних умов. Так, для житлової кімнати з стандартним освітленням показник рівня освітлення становить від 100 lx до 300 lx, що є досить близьким до результатів, отриманих за допомогою BH1750, а саме 259 lx.

У результаті перевірки, було підтверджено коректне зчитування та правильну обробку даних, що отримуються від датчиків BMP280, BH1750 та MQ-135 у результаті здійснення ними вимірювань.

4.2 Перевірка можливості підключення АПК до мережі

Після здійснення успішного завантаження програмного забезпечення, створення якого описано у попередньому пункті, у плату ESP32-WROOM з приєднаними до неї периферійними модулями, а саме датчиками BMP280, BH1750 та MQ-135, було здійснено перевірку роботоздатності системи за параметром можливості здійснення підключення до мережі без використання послідовного порту, дротового з’єднання та перепрошивання плати. Для цього, після завантаження програмного забезпечення у плату, її було від’єднано від комп’ютера та заживлено за допомогою батареї 18650, яка входить до складу апаратного забезпечення апаратно-програмного комплексу.

Після увімкнення плати, у списку з’явилась локальна мережа під назвою ESP32_Env_Cont – Setup. Приєднання до цієї мережі провокує відкриття веб-сторінки, прописаної в бібліотеці WifiManager, яка використовується для здійснення підключення за описаним способом. Вигляд сторінки подано у додатку В цієї роботи. Згідно функцій, закладених у програмне забезпечення апаратно-програмного комплексу, після успішного приєднання до мережі, плата здійснює надсилання повідомлення про свою адресу у Telegram. Результат відправлення сповіщення наведено у додатку В цієї роботи.

У результаті перевірки було з’ясовано, що програмне забезпечення стабільно виконує поставлене перед ним завдання, а саме забезпечення зручного під’єднання

апаратно-програмного комплексу до мережі без використання дротового з’єднання та потреби у перепрошиванні плати ESP32-WROOM.

4.3 Тестування доступу до даних АПК із зовнішніх мереж

Наступним етапом перевірки працездатності програмного забезпечення апаратно-програмного комплексу, розробка якого описана у цій роботі, є перевірка можливості отримання даних спочатку у локальній мережі, і у випадку успіху – з-поза меж локальної мережі, у якій знаходиться апаратно-програмний комплекс.

Для перевірки доступу до даних з локальної мережі, у якій знаходиться АПК, необхідно у полі пошуку браузера персонального комп’ютера чи іншого комп’ютерного пристрою, який підключено до тієї ж мережі, що й АПК, ввести адресу АПК, отриману у Telegram у ході виконання перевірки, описаної у попередньому розділі. Адресу потрібно ввести з використанням протоколу «http://» та додати шлях “/api/data” для надсилання коректного HTTP-запиту на адресу плати, на який буде сформовано відповідь у вигляді JSON-рядка.

У результаті здійснення запиту «http://192.168.68.111/api/data» було отримано відповідь від АПК, що свідчить про успішне здійснення доступу до даних у межах локальної мережі.

Результат отримання даних наведено у додатку В цієї роботи.

Наступним етапом перевірки працездатності програмного забезпечення апаратно-програмного комплексу, розробка якого описана у цій роботі, є перевірка можливості отримання даних з-поза меж локальної мережі, у якій знаходиться АПК. Для цього необхідно використати персональний комп’ютер чи інший комп’ютерний пристрій, який під’єднано до іншої мережі, поза межами мережі, у якій знаходиться АПК. Для виконання перевірки було використано мобільну точку доступу, створену за допомогою смартфона з використанням мобільного інтернет-підключення. Для початку необхідно запустити NGROK для здійснення переадресації адреси плати з метою створення можливості доступу з мережі Інтернет до даних АПК.

З огляду на незмінність адреси, яку NGROK присвоює платі ESP32-WROOM,

через використання зарезервованого домену, для зручності використання було створено Python-застосунок, який виконує введення команди для запуску NGROK у командний рядок комп’ютера. Цей застосунок використовує текстовий документ для зчитування адреси плати для формування коректної команди запуску. Код застосунку наведено у додатку Г цієї роботи.

Після здійснення запуску, у полі пошуку браузера персонального комп’ютера, який використовує мобільну точку доступу, необхідно ввести адресу, отриману у результаті запуску NGROK, тобто “https://esp32environment.ngrok.app” та додати “/api/data” для надсилання коректного HTTP-запиту на адресу плати, на який буде сформовано відповідь у вигляді JSON-рядка.

У результаті здійснення запиту “https://esp32environment.ngrok.app/api/data” було отримано відповідь від АПК, що свідчить про успішне здійснення доступу до даних з-поза меж локальної мережі.

Результат отримання даних наведено у додатку В цієї роботи.

Доступ до даних, а точніше сказати – їхнє відображення буде здійснюватись у вебзастосунку, створення якого описане у третій частині комплексної бакалаврської кваліфікаційної роботи, що має назву «Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 3. Вебзастосунок для трансляції відеопотоку». Результат відображення даних у описаному застосунку наведено у додатку В цієї роботи.

ВИСНОВКИ

У процесі виконання роботи було розроблено та реалізовано програмне забезпечення для апаратно-програмного комплексу моніторингу стану навколишнього середовища на базі мікроконтролера ESP32-WROOM та сенсорів BMP280, BH1750 і MQ-135. Реалізована система забезпечує безперервний збір, обробку та передачу екологічних даних у форматі JSON з використанням мережевого з'єднання через Wi-Fi. Такий підхід дозволив створити гнучке, масштабоване рішення, яке можна адаптувати для різних умов експлуатації та інтегрувати в більші екосистеми Інтернету речей.

Під час реалізації програмної частини особливу увагу було приділено вибору надійних бібліотек, побудові стабільних алгоритмів обміну даними та організації взаємодії між мікроконтролером і віддаленим сервером. Було протестовано різні формати передачі інформації та обґрунтовано вибір JSON як найбільш зручного і поширеного для інтеграції з веб-застосунками та хмарними платформами.

У ході дослідження було доведено ефективність використання IoT-засобів у задачах екологічного моніторингу. Результати тестування системи продемонстрували стабільну роботу комплексу, високу точність зчитування сенсорних даних, а також надійність передачі інформації до віддаленого серверу.

Система може слугувати базою для подальшого розвитку – зокрема, для реалізації мобільних застосунків, веб-інтерфейсів, систем сповіщення або аналітичних модулів.

Таким чином, поставлені у роботі завдання були виконані. Було створено працездатне програмне забезпечення для IoT-комплексу моніторингу, що поєднує апаратні засоби з мережевими технологіями та забезпечує точну й оперативну обробку екологічних даних. Отримані результати підтверджують доцільність та перспективність подальшого використання та вдосконалення подібних рішень у сфері захисту довкілля, розумного міста та систем екологічної безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сидоренко Ю., Петров А. Особливості вибору типу сигналу для вбудованих систем // Системи управління та інформації. — 2017. — Вип. 18, № 2. — С. 14–22.

2. Кравченко С. Багатоканальні системи збору даних на базі мікроконтролерів // Технічний вісник НТУУ "КПІ". — 2021. — Вип. 22, № 5. — С. 33–39.

3. Іванов П. Основи цифрової обробки сигналів у вбудованих системах // Науковий вісник КНУ. — 2020. — Вип. 15, № 1. — С. 72–80.

4. Клименко В. Аналогова обробка сигналів в системах автоматизації // Автоматика та обчислювальна техніка. — 2018. — Вип. 36, № 2. — С. 28–34.

5. Шевченко І. Інтеграція аналогових функцій у сучасні мікроконтролери // Електроніка та системи управління. — 2020. — Вип. 40, № 1. — С. 56–61.

6. Козловський М. Виклики сучасних вбудованих систем: обробка сигналів та енергозбереження // Інформаційні технології і комп’ютерна інженерія. — 2021. — Вип. 12, № 4. — С. 101–109.

7. Бондаренко О. Інтелектуальні системи обробки сигналів на базі мікроконтролерів // Вісник Харківського національного університету радіоелектроніки. — 2023. — Вип. 59, № 3. — С. 45–53.

8. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications // *IEEE Communications Surveys & Tutorials*. — 2015. — Vol. 17, No. 4. — P. 2347–2376. — DOI: 10.1109/COMST.2015.2444095.

9. Козак І. Енергоефективність бездротових мереж IoT // Системи автоматизації та управління. — 2020. — Вип. 7, № 3. — С. 60–67.

10. Niyato D., Kim D. I., Li Y. Wi-Fi 6 and IoT: A review of emerging technologies // *IEEE Network*. — 2020. — Vol. 34, No. 4. — P. 140–147. — DOI: 10.1109/MNET.011.1900707.

11. Петренко А. Використання BLE у системах моніторингу навколишнього середовища // Науковий вісник НТУУ "КПІ". — 2021. — Вип. 18,

№ 4. — С. 112–118.

12. Zhang Q., Yang L., Guo M. Application of LoRaWAN in Smart Cities // *IEEE Access*. — 2019. — Vol. 7. — P. 156350–156362. — DOI: 10.1109/ACCESS.2019.2943847.

13. Liu Y., Zhang L., Li S. Narrowband IoT: A Survey on Techniques and Applications // *IEEE Internet of Things Journal*. — 2020. — Vol. 7, No. 10. — P. 10379–10394. — DOI: 10.1109/JIOT.2020.2993803.

14. Коваленко С. Безпека бездротових IoT-систем: виклики та рішення // Кібербезпека і захист інформації. — 2021. — Вип. 5, № 1. — С. 39–46.

15. Yatsenko D. Блокчейн у бездротових IoT-системах: безпека та довіра // Журнал сучасних технологій. — 2023. — Вип. 14, № 1. — С. 77–84.

16. Singh R., Sharma A., Singh R. Emerging Trends in IoT Communication: 5G and Beyond // *Journal of Network and Computer Applications*. — 2022. — Vol. 192. — Article 103166. — DOI: 10.1016/j.jnca.2021.103166.

17. Бондаренко М. Архітектура локальних комп’ютерних мереж // Інформаційні технології в освіті та науці. — 2021. — Вип. 13, № 2. — С. 34–42.

18. Козак І. Трансляція мережевих адрес (NAT) у сучасних локальних мережах // Системи автоматизації та управління. — 2019. — Вип. 6, № 4. — С. 55–62.

19. Xu J., Li Y. Network Address Translation and Its Applications in Modern Networks // *IEEE Communications Magazine*. — 2020. — Vol. 58, No. 3. — P. 30–36.

20. Петренко А. Використання VPN для захисту корпоративних мереж // Український журнал інформаційної безпеки. — 2022. — Вип. 8, № 3. — С. 44–52.

21. Козак І. Трансляція мережевих адрес (NAT) у сучасних локальних мережах // Системи автоматизації та управління. — 2019. — Вип. 6, № 4. — С. 55–62.

22. Shevchenko S. The importance of VPN technology in protecting information in corporate networks // *Cybersecurity Journal*. — 2021. — Vol. 10, No. 2. — P. 67–75.

23. Zhou X. Network Security in Local and Wide Area Networks // Journal of Network and Computer Applications. — 2019. — Vol. 124. — P. 12–23.

24. Гнатюк О. Безпека комп’ютерних мереж: проблеми та перспективи // Журнал кібербезпеки України. — 2020. — Вип. 6, № 1. — С. 18–27.

25. Рибак В. Особливості маршрутизації у локальних мережах // Вісник Національного університету «Київський політехнічний інститут». — 2021. — Вип. 20, № 1. — С. 101–110.

26. Нікітчук, А. В. Програмування вбудованих систем Інтернету речей : навч. посіб. / А. В. Нікітчук ; Нац. техн. ун-т України "КПІ ім. Ігоря Сікорського". – Київ : НТУУ "КПІ ім. І. Сікорського", 2024. – 180 с

27. ESP32 vs Arduino vs Raspberry Pi Pico: Which is Better? [Електронний ресурс] // *OpenELAB* : блог. – 2023. – Режим доступу: https://openelab.io/blogs/learn/esp32-vs-arduino-vs-raspberry-pi-pico-which-is-better, вільний. – Назва з екрана.

28. ESP-MQTT – ESP-IDF Programming Guide [Електронний ресурс] // Espressif Systems. – Режим доступу: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/mqtt.html, вільний. – Назва з екрана.

29. ESP32: Create a Wi-Fi Manager (AsyncWebServer library) [Електронний ресурс] // *Random Nerd Tutorials*. – Режим доступу: https://randomnerdtutorials.com/esp32-wi-fi-manager-asyncwebserver/, вільний. – Назва з екрана.

30. Programming with the ESP32 / Ubiquity Press. — London: Ubiquity Press, 2020. — 320 с.

31. BMP280 with ESP32 Tutorial (I2C and SPI) [Електронний ресурс] // Random Nerd Tutorials. – Режим доступу: https://randomnerdtutorials.com/esp32-bmp280-barometric-pressure-sensor-arduino/, вільний. – Назва з екрана.

32. ROHM Semiconductor. BH1750FVI цифровий датчик освітленості: технічний опис / ROHM Semiconductor. – Версія 1.2. – Токіо, Японія: ROHM Semiconductor, 2016. – 24 с. – Режим доступу: https://www.mouser.com/datasheet/2/348/bh1750fvi-e-1830678.pdf, вільний. – Назва з екрана.

33. Джексон В. Embedded Systems: Architecture, Programming and Design /

В. Джексон. — New York : Wiley, 2019. — 450 с.

34. Arduino. Wire Library [Електронний ресурс]. – Режим доступу: https://www.arduino.cc/en/Reference/Wire (дата звернення: 30 трав. 2025 р.).

35. Adafruit. Adafruit BMP280 Library [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/adafruit/Adafruit_BMP280_Library – Назва з екрана. – Дата звернення: 30.05.2025.

36. Adafruit_SSD1306 [Електронний ресурс] – Режим доступу: https://github.com/adafruit/Adafruit_SSD1306 (дата звернення: 30.05.2025).

37. BH1750.h library [Електронний ресурс] – Режим доступу: https://github.com/claws/BH1750 (дата звернення: 30.05.2025).

38. Espressif Systems. WiFi library — ESP32 Arduino [Електронний ресурс]. – Режим доступу: https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html, вільний. – Назва з екрана. – Дата звернення: 30.05.2025.

39. ESP32 WebServer library [Електронний ресурс]. – Режим доступу: https://github.com/espressif/arduino-esp32/tree/master/libraries/WebServer, вільний. – Назва з екрану. – Дата доступу: 30.05.2025.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О.Д.

«21» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі iot засобів. Частина 2. Програмне забезпечення»

08-54. КБКР.039.00.000 ТЗ

Науковий керівник: к.т.н., доц. каф. ОТ

_____Богомолов С. В.

Студент групи 1КІ-21б

_____Довгань В. В.

м. Вінниця — 2025

1 Підстава для використання бакалаврської кваліфікаційної роботи (КБКР)

1.1 У сучасних умовах, коли цифрові технології мають визначальне значення для розвитку систем моніторингу, все більшої важливості набуває застосування IoT-рішень для збору, передавання та обробки даних у режимі реального часу. Одним із напрямів є розробка ефективного програмного забезпечення для апаратно-програмного комплексу, що здійснює моніторинг стану навколишнього середовища.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета КБКР і призначення розробки

2.1 Мета роботи – розробка та реалізація програмного забезпечення для апаратно-програмного комплексу моніторингу, побудованого на основі мікроконтролера ESP32-WROOM з використанням датчиків контролю параметрів навколишнього середовища.

2.2 Призначення розробки — забезпечення безперервного збору, обробки та передачі екологічних даних у форматі JSON з використанням сучасних засобів мережевої комунікації.

3 Вихідні дані для виконання КБКР

3.1 Огляд сучасного стану галузі дослідження.

3.2 Розробка частин коду для обробки даних з датчиків.

3.3 Розробка коду для забезпечення зв’язку комплексу з мережею та надсилання даних у форматі JSON.

3.4 Забезпечення доступу до даних апаратно-програмного комплексу з мережі Інтернет.

3.5 Тестування працездатності програмного забезпечення.

4 Вимоги до виконання КБКР

Головна вимога — розробка та реалізація програмного забезпечення для

апаратно-програмного комплексу моніторингу, побудованого на основі мікроконтролера ESP32-WROOM з використанням датчиків контролю параметрів навколишнього середовища.

5 Етапи КБКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 – Етапи роботи та очікувані результати

№ етапу	Назва етапу	Термін виконання	Очікувані результати
1	Огляд сучасного стану галузі дослідження	21.03.25 – 30.03.25	Аналітичний огляд
2	Розробка частин коду для обробки даних датчиків	31.03.25 – 13.04.25	Розділ 2
3	Розробка частини коду для підключення до Wi-Fi	14.04.25 – 27.04.25	Розділ 3
4	Тестування і аналіз результатів	14.04.25 – 27.04.25	Розділ 4
5	Оформлення пояснювальної записки, графічного матеріалу та презентації	28.04.25 – 11.05.25	Пояснювальна записка, графічний матеріал та презентація

6 Матеріали, що подаються до захисту КБКР

До захисту подаються: пояснювальна записка КБКР, графічні та ілюстративні матеріали, протокол попереднього захисту на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до КБКР українською та іноземною мовами, довідка про відповідність оформлення вимогам.

7 Порядок контролю виконання та захисту КБКР

Контроль виконання етапів здійснюється науковим керівником згідно з встановленими термінами. Захист БКР проходить на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання КБКР

8.1 При оформленні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:
Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 2. Програмне забезпечення

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)
Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98% Схожість 2%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Довгань В. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Богомолов С.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС МОНІТОРИНГУ СТАНУ ОТОЧУЮЧОГО СЕРЕДОВИЩА НА БАЗІ ІОТ ЗАСОБІВ. ЧАСТИНА 2. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

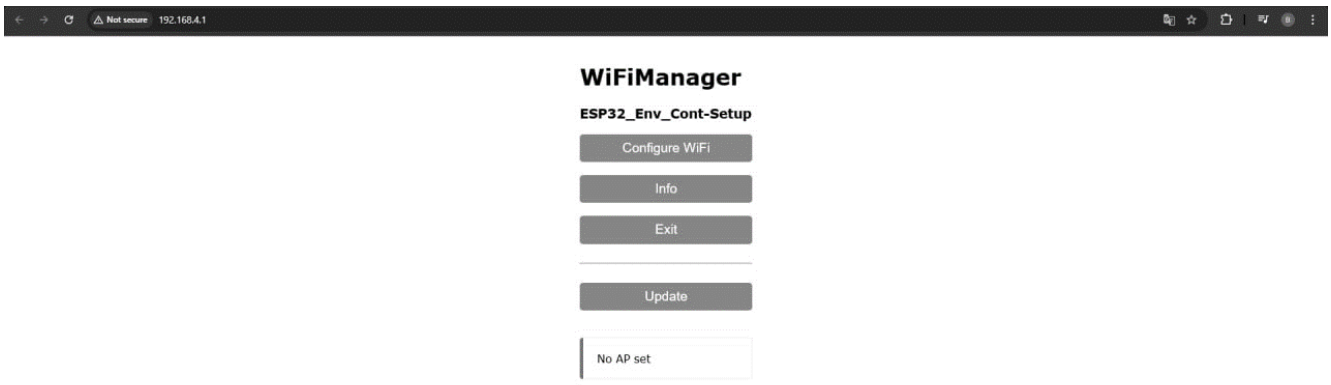

Рисунок В.1 — Вигляд сторінки WiFiManager

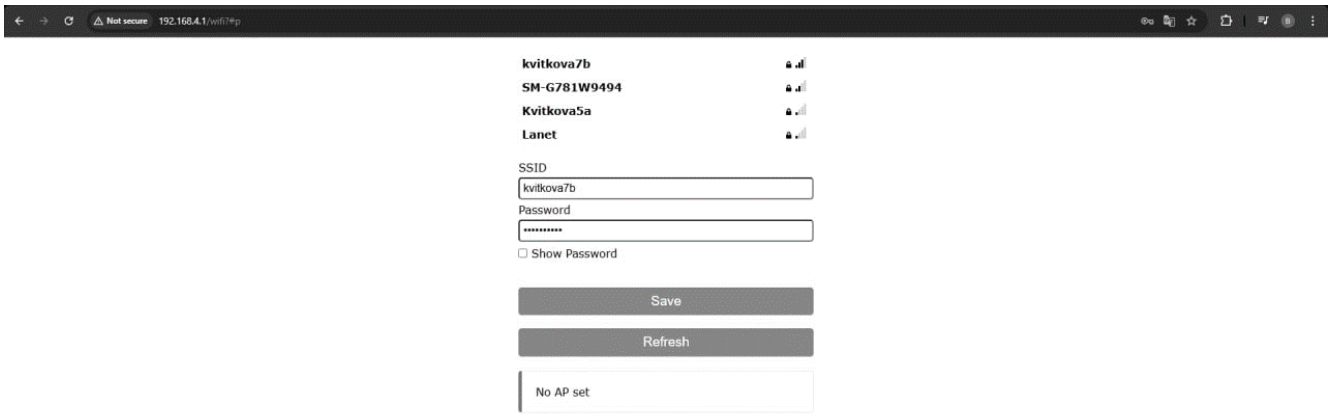

Рисунок В.2 – Вигляд сторінки WiFiManager

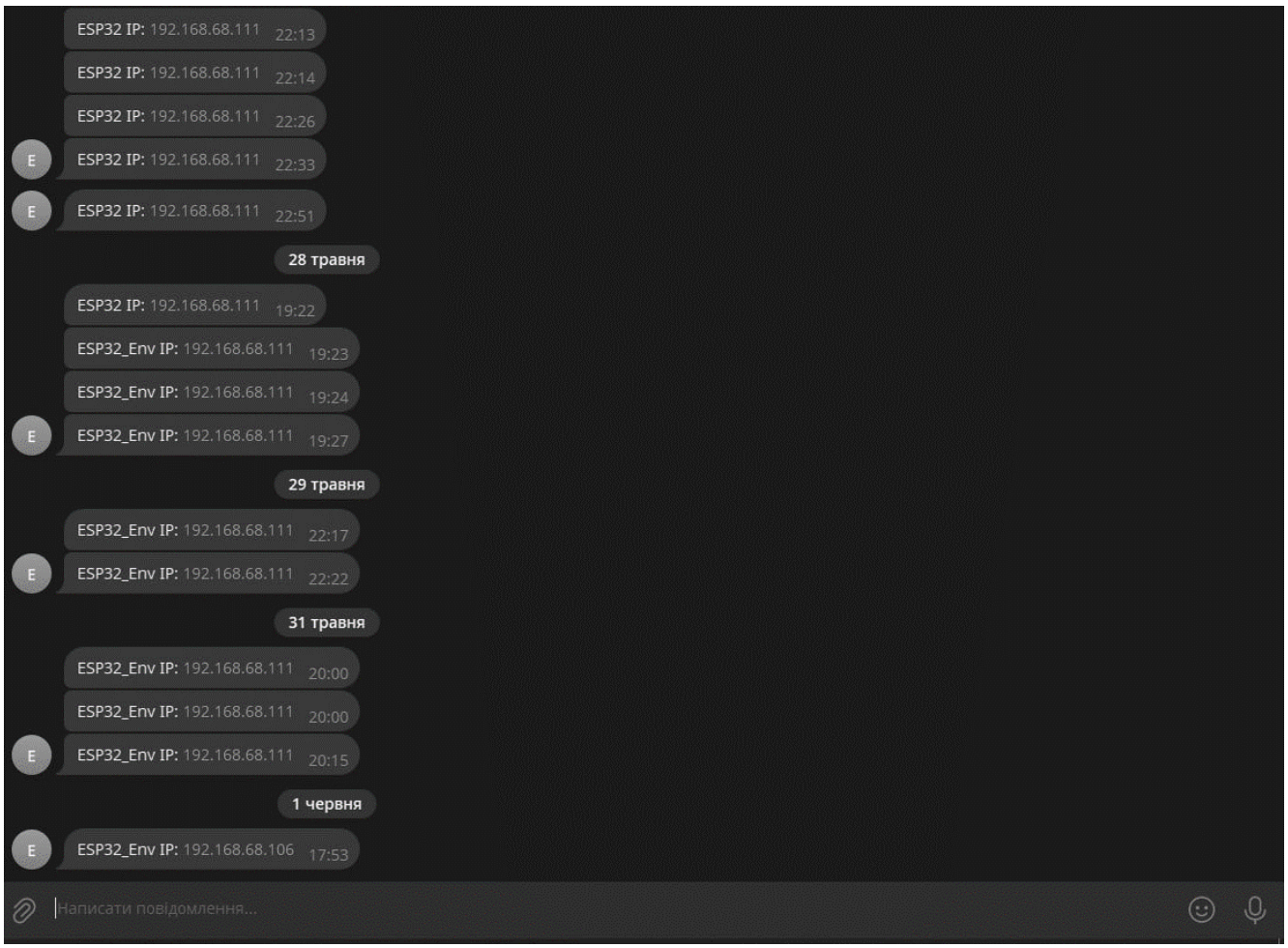

Рисунок В.3 - Результат повідомлення адреси плати у Telegram

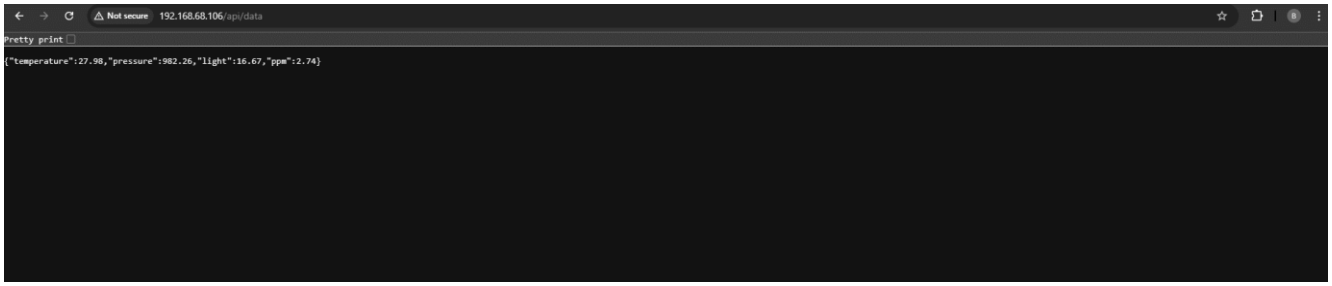

Рисунок В.4 - Результат отримання даних у локальній мережі

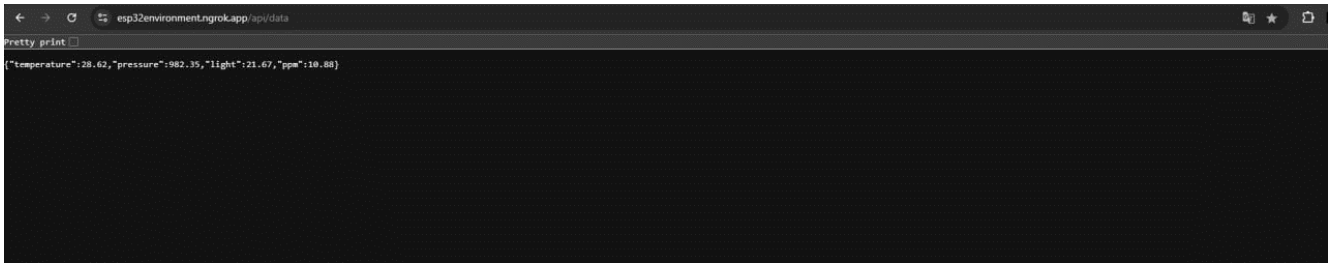

Рисунок В.5 – Результат отримання даних через мережу Інтернет

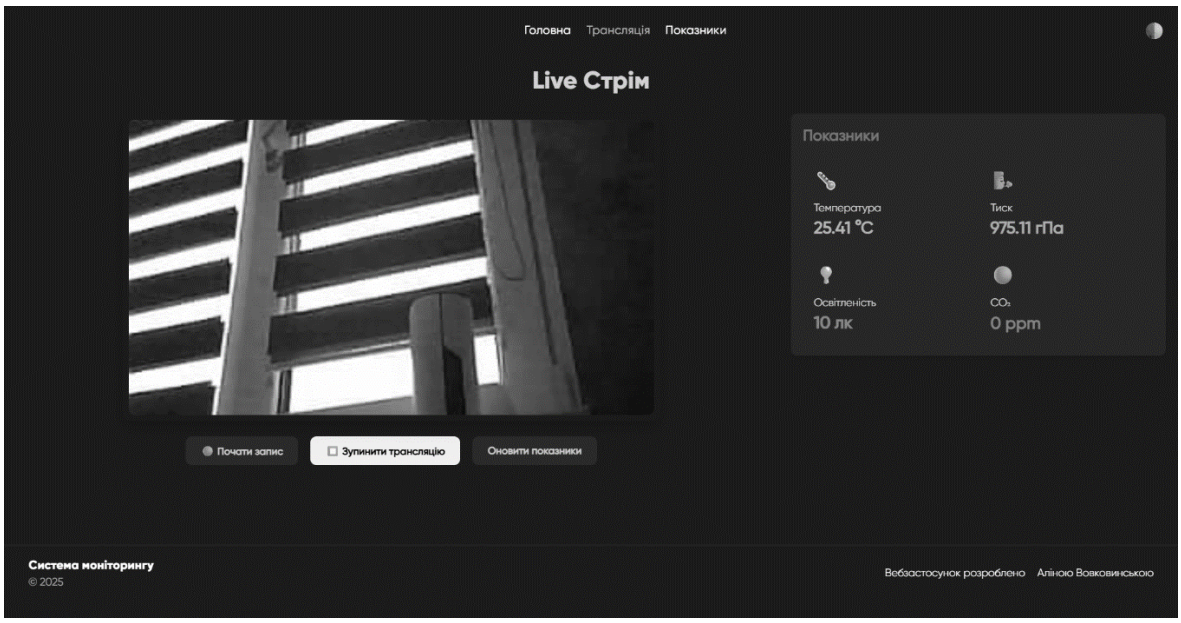

Рисунок В.6 - Відображення даних у веб-застосунку

ДОДАТОК Г

Лістинг коду Python – застосунку для запуску NGROK

```
import subprocess
import time
import requests
output_file = "envControll.txt"
local_ip_file = "local_ip.txt"
try:
    with open(local_ip_file, 'r') as f:
        local_ip = f.read().strip()
    print(f"Локальний IP прочитано з файлу: {local_ip}")
except FileNotFoundError:
    print(f"Файл {local_ip_file} не знайдено. Завершення роботи.")
    exit(1)
command = f'ngrok http --url=esp32environment.ngrok.app {local_ip}'
process = subprocess.Popen(command, shell=True)
print("Чекаємо, поки ngrok запуститься...")
time.sleep(5)
try:
    response = requests.get("http://127.0.0.1:4040/api/tunnels")
    tunnels = response.json()['tunnels']
    if tunnels:
        public_url = tunnels[0]['public_url']
        print(f"Публічна адреса ngrok: {public_url}")
              with open(output_file, 'w') as f:
            f.write(public_url)
        print(f"Адресу записано у файл {output_file}")
    else:
        print("Тунелі не знайдено.")
except Exception as e:
    print("Помилка при отриманні інформації з ngrok API:", e)
```

ДОДАТОК Д

Лістинг коду програмного забезпечення для апаратно програмного комплексу

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <BH1750.h>
#include <WiFi.h>
#include <WebServer.h>
#include <WiFiClientSecure.h>
#include <WiFiManager.h>
#include <UniversalTelegramBot.h>
const char* BOTtoken =
"8024695514:AAF9TMQFZtpPOtiW2LgS9KhkRrgmzmVJm04";
const char* CHAT_ID = "799798585";
const int RL_VALUE = 10;
const int analogPin = 34;
const float R0 = 10.2;
const int samples = 10;
Adafruit_BMP280 bmp;
BH1750 lightMeter;
WebServer server(80);
WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);
float temperature;
float pressure;
float light;
float ppm;
void sendTelegramMessage(String message) {
  client.setInsecure();
  if (client.connect("api.telegram.org", 443)) {
    String url = "/bot" + String(BOTtoken) + "/sendMessage?chat_id=" + CHAT_ID +
"&text=" + message;
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: api.telegram.org\r\n" +
                 "Connection: close\r\n\r\n");
    while (client.connected() || client.available()) {
      String line = client.readStringUntil('\n');
      Serial.println(line);
    }
  } else {
    Serial.println("Помилка підключення до Telegram");
  }
}
```

```
void setup(){
  Serial.begin(115200);
  WiFiManager wm;
  //wm.resetSettings();
  bool res = wm.autoConnect("ESP32_Env_Cont-Setup");
  if (!res) {
    Serial.println("Не вдалося підключитися до WiFi");
  }
  else {
    Serial.println("Підключено!");
    Serial.println(WiFi.localIP());
  }
  String ipMessage = "ESP32_Env IP: " + WiFi.localIP().toString();
  sendTelegramMessage(ipMessage);
  Wire.begin();
  bmp.begin(0x76);
  lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE);
  server.on("/api/data", HTTP_GET, []() {
    String json = "{";
    json += "\"temperature\":" + String(temperature) + ",";
    json += "\"pressure\":" + String(pressure) + ",";
    json += "\"light\":" + String(light) + ",";
    json += "\"ppm\":" + String(ppm);
    json += "}";
    server.sendHeader("Access-Control-Allow-Origin", "*");
    server.send(200, "application/json", json);
  });
  server.begin();
}
void loop(){
  temperature = bmp.readTemperature();
  pressure = bmp.readPressure() / 100.0F;
  light = lightMeter.readLightLevel();
  ppm = getSmoothedPPM();
  server.handleClient();
  delay(2000);
}
float getPPM(int raw_adc) {
  float Vout = (raw_adc / 4095.0) * 3.3 * 2.0;
  float Rs = ((5.0 - Vout) / Vout) * RL_VALUE;
  float ratio = Rs / R0;
  float ppm = 116.6020682 * pow(ratio, -2.769034857);
  return ppm;
}
```

```
float getSmoothedPPM() {
  float sumPPM = 0.0;
  for (int i = 0; i < samples; i++) {
    int adc_val = analogRead(analogPin);
    sumPPM += getPPM(adc_val);
    delay(50);
  }
  return sumPPM / samples;
}
void TempPresLightSerialWrite(){
  Serial.print("Температура: ");
  Serial.print(temperature);
  Serial.print(" °C\n");
  Serial.print("Тиск: ");
  Serial.print(pressure);
  Serial.print(" hPa\n");
  Serial.print("Світло: ");
  Serial.print(light);
  Serial.print(" lx\n");
  Serial.print("\n");
}
```