Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інформаційно-телекомунікаційних технологій та систем

Михайлюк Владислав Андрійович

(прізвище, ім'я, по батькові)

УДК 638.12:004.932

БАКАЛАВРСЬКА РОБОТА

Розроблення інформаційної системи моніторингу стану бджолиної колонії

(назва роботи)

Інформаційні системи та технології

(назва освітньої програми)

126-Інформаційні системи та технології

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ В.А. Михайлюк
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ Левицький Іван Теодорович, к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
В.о. Завідувача кафедри

_____ ***Заміховська О.Л.***
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Факультет *Інформаційних технологій*

Кафедра *Інформаційно-телекомунікаційних технологій та систем*

Освітній рівень *бакалавр*

Спеціальність *126-Інформаційні системи та технології*

(шифр і назва)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри *ІТТС*

к.т.н.,доцент

О.Л. Заміховська

«____» 2026 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Михайлюку Владиславу Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення інформаційної системи моніторингу стану бджолиної колонії*

керівник роботи *Левицький Іван Теодорович, к.т.н., доцент*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "*7*" *квітня* 2026 року № *163/7*

2. Строк подання студентом роботи

3. Вихідні дані до роботи *Матеріали та результати отримані під час проходження переддипломної практики, технічні вимоги, методичні вказівки.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Порівняльний аналіз існуючих систем моніторингу

Вибір технологій та середовища розробки

Розробка алгоритмічної частини

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема

Функціональна схема

Результати розробки

6. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	*Назва етапів бакалаврської роботи*	*Термін виконання Етапів роботи*	*Примітка*
1	*Порівняльний аналіз існуючих систем моніторингу*	*02.03.2026-25.03.2026*	*Виконано*
2	*Вибір середовища розробки і основних блоків системи моніторингу*	*04.04.2026-15.04.2026*	*Виконано*
3	*Розробка алгоритмів і інтерфейсу системи*	*20.04.2026-01.05.2026*	*Виконано*
4	*Оформлення результатів роботи*	*13.05.2026-02.06.2026*	*Виконано*

Студент _____ *Михайлюк В.А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Левицький І.Т.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розрахунково-пояснювальна записка: 116 сторінок, 27 рисунків, 2 таблиці, 29 посилань.

Об'єктом дослідження є процес моніторингу стану бджолиної колонії, що охоплює автоматизований збір фізичних параметрів вулика, їх обробку та надання пасічнику у зручному для аналізу вигляді через локальну Wi-Fi мережу.

Мета роботи – розроблення інформаційної системи моніторингу стану бджолиної колонії на базі мікроконтролера ATXMEGA256A3U з вбудованим web-сервером на модулі ESP8266, що забезпечує автоматизований збір, обробку та зберігання даних із сенсорів, а також їх відображення у web-інтерфейсі.

У першому розділі проведено аналіз предметної області: розглянуто особливості процесу моніторингу стану бджолиних колоній та визначено ключові параметри контролю. Виконано огляд та порівняльний аналіз існуючих комерційних та відкритих систем моніторингу. Обґрунтовано необхідність розроблення власної інформаційної системи.

У другому розділі виконано проєктування інформаційної системи. Розроблено загальну архітектуру системи.. Розглянуто апаратну платформу, підсистему збору даних, модуль бездротового зв'язку та організацію Wi-Fi мережі, зберігання даних. Спроєктовано структуру вбудованого web-сервера та web-інтерфейс користувача.

У третьому розділі виконано розроблення та тестування інформаційної системи. Розроблено програмне забезпечення: реалізовано збір даних із датчиків, FFT-аналіз акустичного сигналу. Реалізовано вбудований web-сервер на базі ESP8266 RTOS SDK із підтримкою REST API та механізму Server-Sent Events. Розроблено web-інтерфейс на HTML5 та Chart.js.

МОНІТОРИНГ ВУЛИКІВ, БДЖОЛИНА КОЛОНІЯ, ATXMEGA256A3U, ESP8266, WEB-СЕРВЕР, IoT, FFT-АНАЛІЗ, MICROSD, SERVER-SENT EVENTS.

ABSTRACT

Calculation and explanatory note: 116 pages, 27 figures, 2 tables, 29 references.

The object of the study is the process of monitoring the state of a bee colony, which includes the automated collection of physical parameters of the hive, their processing and provision to the beekeeper in a form convenient for analysis via a local Wi-Fi network.

The purpose of the work is to develop an information system for monitoring the state of a bee colony based on the ATXMEGA256A3U microcontroller with a built-in web server on the ESP8266 module, which provides automated collection, processing and storage of data from sensors, as well as their display in the web interface.

The first section analyzes the subject area: the features of the process of monitoring the state of bee colonies are considered and the key control parameters are determined. A review and comparative analysis of existing commercial and open monitoring systems are performed. The need to develop our own information system is justified.

The second section designs the information system. The general architecture of the system has been developed.. The hardware platform, data acquisition subsystem, wireless communication module and Wi-Fi network organization, data storage have been considered. The structure of the built-in web server and web user interface have been designed.

In the third section, the development and testing of the information system have been carried out. The software has been developed: data collection from sensors, FFT analysis of the acoustic signal have been implemented. The built-in web server based on ESP8266 RTOS SDK with REST API support and the Server-Sent Events mechanism has been implemented. The web interface has been developed on HTML5 and Chart.js.

MONITORING OF HIVES, BEE COLONY, ATXMEGA256A3U, ESP8266, WEB SERVER, IoT, FFT ANALYSIS, MICROSD, SERVER-SENT EVENTS.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ІС — інформаційна система

САК — система автоматичного керування

ККД — коефіцієнт корисної дії

ПЛК — програмований логічний контролер

МК — мікроконтролер

ООН — організація об’єднаних націй

ВСТУП

Бджільництво є однією з найдавніших галузей сільського господарства, яка відіграє надзвичайно важливу роль не лише у виробництві меду, воску, прополісу та інших продуктів, а й у забезпеченні запилення сільськогосподарських культур. За оцінками Продовольчої та сільськогосподарської організації ООН (ФАО), близько третини світового виробництва продовольства залежить від запилення комахами, серед яких бджоли займають провідне місце. В Україні бджільництво традиційно розвинене, а кількість пасік щороку зростає як серед приватних господарств, так і серед комерційних виробників.

Попри важливість галузі, сучасні пасічники стикаються з цілою низкою серйозних проблем. Однією з найгостріших є масова загибель бджолиних колоній, яка спостерігається в усьому світі протягом останніх десятиліть. Причинами цього явища є ураження кліщем Varroa destructor, бактеріальні та грибкові захворювання, вплив пестицидів, несприятливі кліматичні умови, а також порушення температурного режиму та вологості всередині вулика. Своєчасне виявлення відхилень від норми є критично важливим для збереження колонії та попередження її загибелі.

Традиційні методи контролю стану бджолиних колоній ґрунтуються на регулярних фізичних оглядах вуликів, які вимагають безпосередньої присутності пасічника на пасіці. Такий підхід має суттєві обмеження: огляди є трудомісткими та стресовими для бджіл, не дозволяють вести безперервний контроль, а у разі розташування пасіки у важкодоступних місцях — стають ще менш ефективними. Крім того, людський фактор нерідко призводить до пропуску ранніх ознак захворювань або погіршення умов утримання колонії, що в подальшому спричиняє значні економічні збитки.

Стрімкий розвиток інформаційних технологій, зокрема Інтернету речей (IoT), відкриває нові можливості для автоматизації моніторингу в різних галузях, у тому числі в бджільництві. Використання мережі мікроконтролерів із підключеними датчиками температури, вологості, ваги, акустичних та інших параметрів дозволяє організувати безперервний збір даних про стан вулика без втручання людини. Отримані дані передаються на сервер, де обробляються, зберігаються та аналізуються, а пасічник отримує доступ до актуальної інформації через веб-інтерфейс або мобільний застосунок у будь-який час і з будь-якого місця.

Актуальність розроблення інформаційної системи моніторингу стану бджолиної колонії обумовлена, з одного боку, нагальною потребою галузі в сучасних інструментах дистанційного контролю, а з іншого — недостатньою доступністю існуючих комерційних рішень для широкого кола пасічників через їх високу вартість або обмежену функціональність. Розроблення власної системи дозволяє врахувати специфічні потреби користувачів, забезпечити гнучкість налаштування та інтеграції з наявною інфраструктурою пасіки.

Мета роботи — розроблення інформаційної системи моніторингу стану бджолиної колонії, що забезпечує автоматизований збір, зберігання, аналіз та візуалізацію даних із сенсорів у режимі реального часу з можливістю дистанційного доступу та своєчасного оповіщення користувача про критичні зміни параметрів вулика.

Для досягнення поставленої мети необхідно вирішити такі **задачі дослідження**:

перша — проаналізувати предметну область та дослідити особливості процесу моніторингу стану бджолиних колоній; друга — виконати огляд та порівняльний аналіз існуючих інформаційних систем і рішень у сфері моніторингу вуликів; третя — визначити функціональні та нефункціональні вимоги до розроблюваної системи; четверта — розробити концептуальну модель та архітектуру інформаційної системи; п'ята — спроєктувати та

реалізувати підсистему збору даних із сенсорів на основі IoT-технологій; шоста — розробити серверну частину системи, базу даних та REST API; сьома — реалізувати модуль аналізу стану бджолиної колонії та підсистему автоматичних сповіщень; восьма — розробити веб-інтерфейс для відображення та візуалізації даних моніторингу; дев'ята — провести комплексне тестування системи та оцінити її працездатність і відповідність вимогам.

Об'єктом дослідження є процес моніторингу стану бджолиної колонії в умовах пасіки, що охоплює збір, передачу, обробку та аналіз даних про параметри мікроклімату вулика та фізіологічний стан колонії.

Предметом дослідження є методи, моделі та програмні засоби розроблення інформаційної системи моніторингу стану бджолиної колонії на основі IoT-технологій та сучасних веб-технологій.

Методи дослідження. У процесі виконання бакалаврської роботи використано методи системного аналізу для дослідження предметної області та визначення вимог до системи; методи об'єктно-орієнтованого моделювання для побудови UML-діаграм та опису поведінки системи; методи структурного проєктування для розроблення архітектури системи та схеми бази даних; методи функціонального та навантажувального тестування для перевірки коректності та продуктивності розробленого програмного забезпечення.

Практичне значення роботи полягає в тому, що розроблена інформаційна система надає пасічнику можливість дистанційно та безперервно контролювати стан бджолиних колоній, отримувати своєчасні сповіщення про відхилення параметрів від норми, аналізувати динаміку змін та приймати обґрунтовані управлінські рішення щодо догляду за пасікою. Впровадження системи дозволяє суттєво скоротити трудові витрати на обслуговування пасіки, знизити ризики втрати колоній та підвищити загальну ефективність бджільництва. Система може бути використана як на невеликих приватних пасіках, так і на великих комерційних підприємствах бджільництва.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості моніторингу стану бджолиної колонії

Бджолина колонія є складною біологічною системою, стан якої визначається сукупністю взаємопов'язаних біологічних, фізичних та екологічних параметрів. На відміну від традиційних об'єктів моніторингу в промисловості чи сільському господарстві, бджолина колонія характеризується високою динамічністю внутрішніх процесів, чутливістю до зовнішніх впливів та складністю інтерпретації отримуваних даних. Саме тому моніторинг стану бджолиних колоній є специфічним завданням, що вимагає комплексного підходу та врахування біологічних особливостей об'єкта спостереження [1].

Центральною характеристикою бджолиної колонії є її здатність до саморегуляції мікроклімату всередині вулика. Бджоли підтримують температуру в розплідній зоні на рівні 34-36 °C незалежно від зовнішніх умов, охолоджуючи вулик вентиляцією крилами в спекотну погоду та утворюючи зимовий клуб для збереження тепла в холодний період. Відхилення температури від оптимального діапазону навіть на кілька градусів може призвести до загибелі розплоду, ослаблення колонії та розвитку захворювань. Тому безперервний контроль температурного режиму є одним із ключових завдань системи моніторингу.

Не менш важливим параметром є відносна вологість повітря всередині вулика. Оптимальний діапазон вологості становить 40–80% залежно від пори року та фази розвитку колонії. Підвищена вологість створює сприятливе середовище для розвитку грибкових захворювань, зокрема аскосферозу та нозематозу, тоді як надмірно суха атмосфера негативно впливає на дозрівання меду та стан розплоду. Моніторинг вологості в поєднанні з температурою дає змогу комплексно оцінити мікрокліматичні умови утримання колонії.

Маса вулика є одним із найінформативніших інтегральних показників стану бджолиної колонії. Динаміка зміни маси відображає інтенсивність медозбору, споживання кормових запасів, роєвий стан колонії та загальну її активність. У період активного медозбору маса вулика може збільшуватися на 1-3 кг на добу, тоді як у зимовий період відбувається поступове зменшення маси внаслідок споживання меду. Різке зниження маси вулика в нетиповий час може свідчити про роїння, крадіжку меду іншими бджолами або появу захворювань. Зважування вулика в ручному режимі є трудомістким процесом, тому автоматизований контроль маси за допомогою тензодатчиків дає суттєві переваги [2].

Акустичні характеристики вулика також несуть важливу інформацію про стан колонії. Бджоли продукують звуки різної частоти та інтенсивності залежно від свого стану: спокійний гул свідчить про нормальну активність колонії, тоді як підвищений рівень шуму може вказувати на тривогу, роєвий настрій або присутність матки-суперника. Аналіз акустичного сигналу дозволяє виявити роєвий стан колонії ще до виходу рою, що дає пасічнику час для вжиття запобіжних заходів. Застосування мікрофонів та алгоритмів обробки звукових сигналів є перспективним напрямом у розвитку систем моніторингу вуликів [3].

Окремої уваги заслуговує моніторинг льотної активності бджіл – кількості вильотів та повернень бджіл до вулика за одиницю часу. Цей показник відображає інтенсивність збиральної діяльності колонії, наявність медоносної бази в окрузі та загальний стан здоров'я бджіл. Різке зниження льотної активності може свідчити про отруєння бджіл пестицидами, захворювання або несприятливі погодні умови. Для вимірювання льотної активності використовують оптичні або інфрачервоні лічильники, встановлені на льотку вулика.

Важливою особливістю моніторингу бджолиних колоній є сезонний характер змін параметрів. Стан колонії суттєво відрізняється залежно від пори

року: навесні відбувається нарощування сили колонії та перший медозбір, влітку – пік льотної активності та основний медозбір, восени – підготовка до зимівлі та накопичення кормових запасів, взимку – стан спокою та мінімальна активність. Система моніторингу повинна враховувати ці сезонні закономірності при аналізі отриманих даних та формуванні порогових значень для сповіщень.

Суттєвою проблемою при організації моніторингу є умови розміщення пасік. Вулики нерідко розташовані у віддалених місцях – у лісах, на полях або в горах, де відсутнє стабільне електропостачання та мережевий зв'язок. Це висуває особливі вимоги до енергоефективності обладнання моніторингу, можливості автономної роботи від акумуляторів або сонячних панелей, а також до використання бездротових технологій передачі даних з широким покриттям – таких як LoRaWAN, NB-IoT або стільникові мережі.

Окремим аспектом є вимоги до неінвазивності методів моніторингу. Бджоли є надзвичайно чутливими до втручань у їхнє середовище існування: відкриття вулика, вібрації, стороннє освітлення та запахи можуть спричиняти стрес у колонії та негативно впливати на її продуктивність. Тому датчики та вимірювальне обладнання повинні встановлюватися таким чином, щоб мінімально порушувати природне середовище вулика та не перешкоджати нормальній життєдіяльності бджіл [4].

Таким чином, моніторинг стану бджолиних колоній є багатопараметричним завданням, що охоплює контроль температури, вологості, маси, акустичних характеристик та льотної активності. Комплексний аналіз цих показників у режимі реального часу дозволяє своєчасно виявляти відхилення від норми, прогнозувати розвиток несприятливих ситуацій та забезпечувати оптимальні умови для життєдіяльності бджолиних колоній. Саме ці вимоги визначають функціональність та архітектуру інформаційної системи моніторингу, що розробляється в рамках даної бакалаврської роботи.

1.2 Огляд та порівняльний аналіз існуючих інформаційних систем моніторингу вуликів

На сьогоднішній день на ринку існує ряд комерційних та відкритих рішень у сфері моніторингу бджолиних колоній, які різняться за функціональністю, технічною реалізацією, вартістю та доступністю. Аналіз існуючих систем є необхідним етапом перед розробленням власного рішення, оскільки дозволяє виявити їх переваги та недоліки, визначити незайняті ніші та сформулювати вимоги до розроблюваної системи з урахуванням наявного досвіду.

Серед комерційних рішень найбільш відомою є система моніторингу вуликів компанії Arnia (Велика Британія) [5]. Система включає комплект датчиків для вимірювання температури в різних зонах вулика, акустичного моніторингу та ваги. Дані передаються на хмарний сервер через стільникову мережу, де обробляються та відображаються у web-інтерфейсі, рис.1.1.

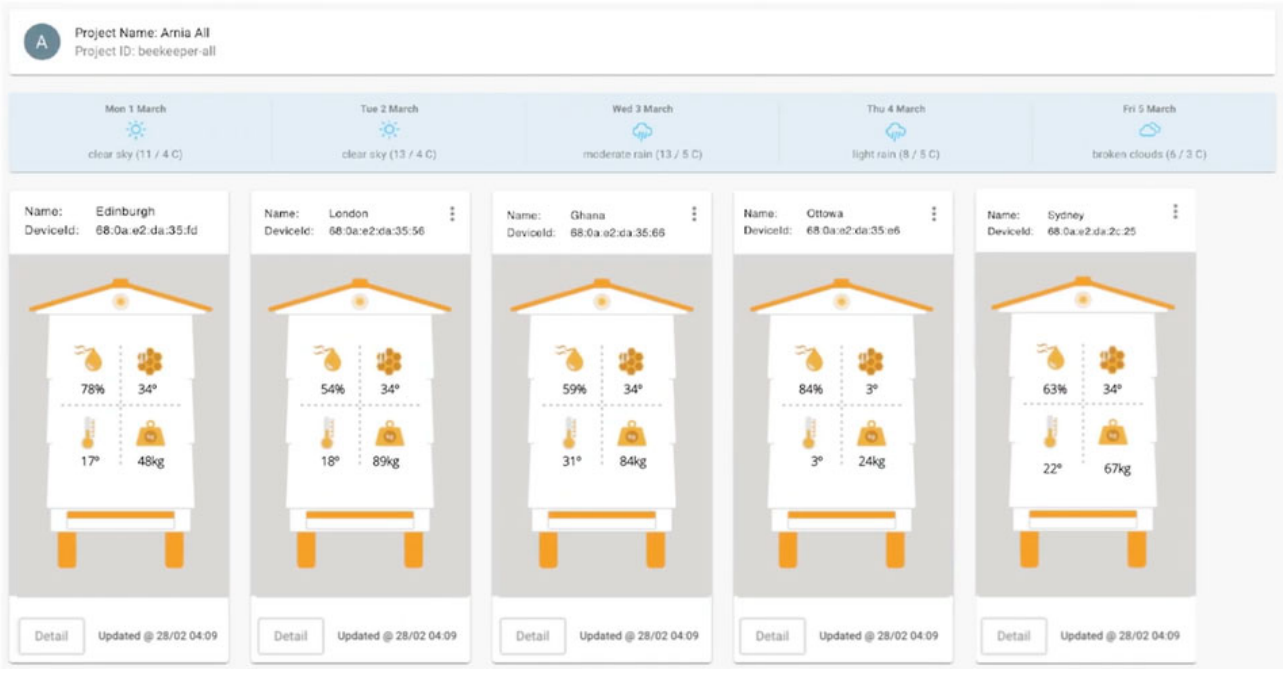

Рисунок 1.1 – Інформаційна система Arnia

Система забезпечує автоматичне виявлення роєвого стану колонії на основі аналізу акустичних сигналів та температурних змін. Програмне забезпечення надає пасічнику детальні звіти про стан колонії, динаміку змін параметрів та рекомендації щодо догляду. Головним недоліком рішення є висока вартість обладнання, а також щомісячна абонентська плата за доступ до хмарної платформи. Це робить систему недоступною для більшості приватних пасічників. Крім того, система орієнтована переважно на великі комерційні пасіки та не адаптована до умов малого та середнього бджільництва.

Іншим відомим комерційним рішенням є система BroodMinder (США), яка пропонує широкий спектр датчиків для моніторингу температури, вологості та ваги вулика [6]. Особливістю системи є використання технології Bluetooth Low Energy для передачі даних на смартфон пасічника, що суттєво знижує енергоспоживання обладнання та дозволяє датчикам працювати від батарейок протягом кількох років, рис. 1.2.

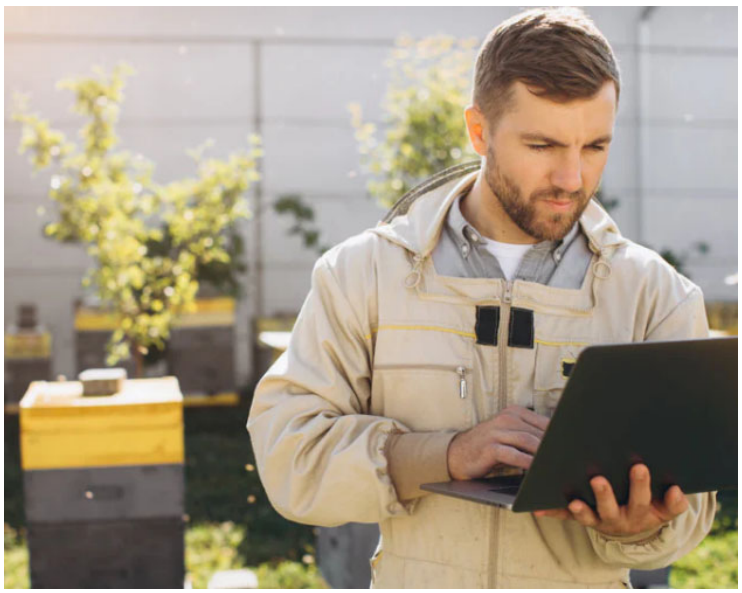

Рисунок 1.2 – Інформаційна система BroodMinder

Дані синхронізуються з хмарною платформою MyBroodMinder, де зберігаються та аналізуються з використанням алгоритмів машинного навчання. Система має зручний мобільний застосунок для iOS та Android і підтримує інтеграцію з метеорологічними сервісами для кореляції стану

колонії з погодними умовами. Проте обмежений радіус дії Bluetooth, який становить не більше 30–50 метрів, є суттєвим недоліком для пасічників, чиї пасіки розташовані на значній відстані від житла. Крім того, хмарна платформа вимагає платної підписки для доступу до розширеної аналітики та зберігання даних понад визначений термін.

Система HiveTech (Австралія) є прикладом комплексного IoT-рішення для моніторингу вуликів, що поєднує датчики температури, вологості, ваги та лічильники льотної активності. Система використовує технологію LoRaWAN для передачі даних на великі відстані – до 10-15 кілометрів у відкритій місцевості – при мінімальному енергоспоживанні, що робить її придатною для використання у віддалених місцях без стабільного стільникового покриття, рис. 1.3.

Рисунок 1.3 – Інформаційна система HiveTech

Шлюз LoRaWAN встановлюється на пасіці та передає зібрані дані на хмарний сервер через мережу Інтернет. Дані відображаються у web-інтерфейсі з можливістю налаштування порогових значень та автоматичних сповіщень електронною поштою або SMS. Недоліком системи є складність встановлення та налаштування обладнання, яке вимагає певних технічних знань від

користувача, необхідність придбання та встановлення шлюзу LoRaWAN, а також обмежена доступність на ринках за межами Австралії та Нової Зеландії.

Система Nectar (Ізраїль) є відносно новим рішенням, що з'явилося на ринку та привернуло увагу завдяки використанню штучного інтелекту для аналізу стану колонії. Система включає датчики температури, вологості, ваги та акустичного моніторингу, а також камеру для візуального спостереження за льотком. Алгоритми машинного навчання аналізують зібрані дані та формують рекомендації для пасічника щодо оптимального часу огляду вулика, ймовірності роїння та ризику захворювань. Попри технологічну досконаліть, система має обмежену географічну доступність та надзвичайно високу вартість, що робить її практично недосяжною для звичайного пасічника, рис.1.4.

Рисунок 1.4 – Інформаційна система Nectar

Серед рішень на основі відкритого програмного забезпечення варто відзначити проєкт OpenHive, який є спільнотою розробників та пасічників, що створюють власні системи моніторингу вуликів на базі платформ Arduino та Raspberry Pi. Проєкт надає відкриті схеми підключення датчиків, програмний

код мікроконтролерів та серверну частину для збору та відображення даних. Перевагою такого підходу є повна відкритість та можливість адаптації під конкретні потреби, а також мінімальна вартість обладнання, яка може становити лише кілька десятків доларів за комплект для одного вулика. Водночас суттєвим недоліком є відсутність єдиного стандарту реалізації, складність відтворення для користувачів без технічних знань, а також відсутність комерційної підтримки та гарантій стабільності роботи [7].

Проєкт Hivetool є ще одним прикладом відкритої системи моніторингу вуликів, розробленої американськими ентузіастами. Система базується на платформі Raspberry Pi та підтримує підключення датчиків температури, вологості та ваги. Зібрані дані зберігаються в локальній базі даних та відображаються у web-інтерфейсі з можливістю побудови графіків динаміки змін параметрів [8]. Проєкт має активну спільноту користувачів та велику базу накопичених даних про стан вуликів у різних регіонах світу, що становить значну наукову цінність. Однак система має застарілий інтерфейс користувача, обмежені можливості аналітики та не підтримує сучасні протоколи бездротової передачі даних, що ускладнює її використання в умовах віддалених пасік, рис.1.5.

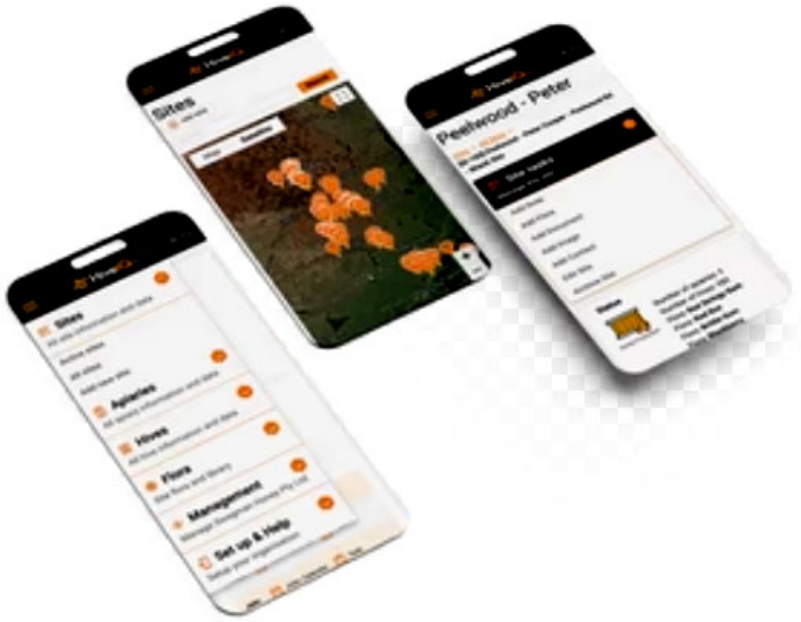

Рисунок 1.5 – Інформаційна система Hivetool

На українському ринку системи моніторингу вуликів представлені переважно закордонними рішеннями або саморобними розробками окремих ентузіастів. Вітчизняних комерційних продуктів у цій ніші практично немає, що обумовлено відносною новизною напряму та невеликим розміром ринку. Водночас потреба у доступних та адаптованих до місцевих умов системах моніторингу є очевидною, зважаючи на значну кількість пасік в Україні та традиційно високий рівень розвитку бджільництва в країні [9,10].

Для систематизації результатів огляду та наочного представлення порівняльних характеристик розглянутих систем складено таблицю 1.1.

Таблиця 1.1 — Порівняльний аналіз існуючих систем моніторингу бджіл

Критерій	**Arnia**	**BroodMinder**	**HiveTech**	**Nectar**	**OpenHive**	**Hivetool**
Температура	+	+	+	+	+	+
Вологість	+	+	+	+	+	+
Вага	+	+	+	+	−	+
Акустика	+	−	−	+	−	−
Льотна активність	−	−	+	+	−	−
Відеоспостереженя	−	−	−	+	−	−
Технологія зв'язку	GSM	Bluetooth	LoRaWAN	GSM	Wi-Fi	Wi-Fi
Веб-інтерфейс	+	+	+	+	+	+
Мобільний застосунок	+	+	−	+	−	−
Автономна робота	+	+	+	+	−	−
Сповіщення	+	+	+	+	−	−
Аналіз аномалій	+	+	−	+	−	−
Відкритий код	−	−	−	−	+	+
Українська мова	−	−	−	−	−	−
Вартість	Висока	Середня	Висока	Дуже висока	Низька	Низька

Аналіз даних таблиці 1.1 дозволяє зробити ряд важливих висновків. По-перше, жодна з розглянутих систем не підтримує українськомовний інтерфейс, що ускладнює їх використання вітчизняними пасічниками. По-друге, комерційні рішення з розвиненою функціональністю є недоступними за вартістю для приватних пасік, тоді як доступні відкриті рішення мають обмежені можливості та вимагають технічних знань. По-третє, більшість систем використовують хмарні платформи із платною підпискою, що створює залежність від стороннього постачальника послуг та додаткові щомісячні витрати. По-четверте, жодна із систем не поєднує в собі повний набір необхідних функцій за прийнятною вартістю та з простим інтерфейсом для нетехнічного користувача.

Таким чином, проведений огляд та порівняльний аналіз існуючих інформаційних систем моніторингу вуликів підтверджує актуальність розроблення власного рішення, яке усувало б виявлені недоліки та відповідало потребам українських пасічників. Розроблювана в рамках даної бакалаврської роботи система повинна забезпечувати комплексний моніторинг основних параметрів вулика, мати зручний україномовний web-інтерфейс, підтримувати автоматичні сповіщення та бути доступною за вартістю завдяки використанню відкритих технологій і компонентів.

1.3. Обґрунтування необхідності розроблення власної інформаційної системи

Проведений у попередньому підрозділі порівняльний аналіз існуючих рішень у сфері моніторингу бджолиних колоній наочно демонструє, що жодна з розглянутих систем не відповідає повною мірою потребам сучасного українського пасічника. Це обумовлює необхідність розроблення власної інформаційної системи, яка поєднувала б переваги існуючих рішень, усувала

їх недоліки та враховувала специфічні вимоги цільової аудиторії. У даному підрозділі детально обґрунтовується доцільність розроблення нової системи з технічної, економічної та практичної точок зору.

З технічної точки зору існуючі рішення мають ряд суттєвих обмежень, які знижують їх ефективність у реальних умовах експлуатації. Комерційні системи, такі як Arnia та BroodMinder, використовують закриті протоколи передачі даних та власні хмарні платформи, що унеможливлює їх інтеграцію з іншими системами управління пасікою або розширення функціональності відповідно до індивідуальних потреб користувача. Залежність від хмарної інфраструктури стороннього постачальника створює ризики втрати доступу до даних у разі припинення роботи сервісу або зміни умов обслуговування. Крім того, більшість комерційних систем не надає доступу до сирих даних вимірювань, що унеможливлює їх використання для наукових досліджень або навчання алгоритмів аналізу [11].

Відкриті рішення, такі як OpenHive та Hivetool, хоча й позбавлені зазначених обмежень, мають власні суттєві недоліки. Відсутність єдиної архітектури та стандартизованих інтерфейсів ускладнює їх розгортання та обслуговування. Застаріла кодова база більшості відкритих проєктів не відповідає сучасним стандартам розроблення програмного забезпечення, що ускладнює їх підтримку та розвиток. Відсутність зручного web-інтерфейсу та мобільного застосунку робить такі системи непридатними для використання пересічним пасічником без технічних знань. Обмежені можливості аналізу даних та відсутність підсистеми автоматичних сповіщень знижують практичну цінність цих рішень.

З економічної точки зору розроблення власної інформаційної системи є економічно обґрунтованим рішенням. Вартість комерційних систем моніторингу для однієї пасіки середнього розміру, що налічує 20–30 вуликів, може становити від кількох тисяч до десятків тисяч доларів з урахуванням вартості обладнання та річної абонентської плати. Для більшості приватних

пасічників та невеликих фермерських господарств такі витрати є неприйнятними. Натомість розроблення системи на основі доступних компонентів — мікроконтролерів ESP32, стандартних датчиків температури та вологості, тензодатчиків — дозволяє суттєво знизити вартість апаратної частини. Використання відкритого програмного забезпечення та хмарних платформ з безкоштовними тарифами для малих обсягів даних додатково зменшує загальну вартість рішення.

Важливим аргументом на користь розроблення власної системи є можливість її повної адаптації до умов українського ринку та потреб вітчизняних пасічників. Розроблювана система матиме повністю україномовний інтерфейс, що суттєво спростить її використання для широкої аудиторії. Система буде адаптована до особливостей українського клімату та сезонних закономірностей розвитку бджолиних колоній у різних регіонах країни — від Карпат до степових районів півдня. Порогові значення параметрів та алгоритми виявлення аномалій будуть налаштовані відповідно до типових умов утримання бджіл в Україні.

Розроблення власної системи також надає можливість реалізувати ряд функціональних можливостей, які відсутні в більшості існуючих рішень. Зокрема, планується реалізація модуля аналізу тенденцій зміни параметрів з використанням статистичних методів, що дозволить виявляти відхилення від норми на ранніх стадіях. Система забезпечуватиме ведення детального журналу подій для кожного вулика, що спростить ретроспективний аналіз стану колоній та прийняття обґрунтованих рішень щодо догляду за пасікою. Підсистема сповіщень забезпечуватиме своєчасне інформування пасічника через різні канали зв'язку — електронну пошту та push-повідомлення у web-застосунку.

Суттєвою перевагою власної розробки є можливість використання сучасного технологічного стеку, що відповідає актуальним стандартам розроблення web-застосунків. Застосування мікросервісної архітектури

забезпечить масштабованість системи та можливість її розширення у майбутньому. Використання REST API як інтерфейсу взаємодії між компонентами системи забезпечить можливість інтеграції з іншими системами управління пасікою або розроблення мобільного застосунку в перспективі. Реляційна база даних з чітко визначеною схемою забезпечить надійне зберігання та ефективне отримання накопичених даних моніторингу.

Необхідність розроблення власної системи підтверджується також з точки зору освітньої та наукової цінності роботи. Процес проєктування та реалізації інформаційної системи моніторингу охоплює широкий спектр задач, характерних для спеціальності «Інформаційні системи та технології»: аналіз предметної області, моделювання бізнес-процесів, проєктування бази даних, розроблення серверної частини та web-інтерфейсу, інтеграція з апаратними компонентами IoT, тестування та документування програмного забезпечення. Це робить тему бакалаврської роботи комплексною та такою, що охоплює всі ключові аспекти підготовки фахівця з інформаційних систем.

Таким чином, обґрунтування необхідності розроблення власної інформаційної системи моніторингу стану бджолиної колонії базується на сукупності технічних, економічних та практичних аргументів. Відсутність доступного, функціонального та адаптованого до українських умов рішення на ринку, обмеження існуючих комерційних та відкритих систем, а також економічна доцільність власної розробки на базі сучасних відкритих технологій – усе це підтверджує актуальність та практичну значущість роботи. Розроблювана система покликана заповнити існуючу прогалину на ринку та надати вітчизняним пасічникам доступний, зручний та надійний інструмент дистанційного моніторингу стану бджолиних колоній.

1.4 Мета, задачі та предмет дослідження

Проведений аналіз предметної області, існуючих систем моніторингу бджолиних колоній та обґрунтування необхідності розроблення власного рішення дозволяють чітко сформулювати мету, задачі, об'єкт та предмет дослідження, а також визначити вимоги до розроблюваної інформаційної системи.

Метою роботи є розроблення інформаційної системи моніторингу стану бджолиної колонії на базі мікроконтролера ATXMEGA256A3U з вбудованим web-сервером на модулі ESP8266, що забезпечує автоматизований збір, обробку та локальне зберігання даних із сенсорів, а також їх відображення у web-інтерфейсі користувача в режимі реального часу через Wi-Fi мережу без залежності від хмарних сервісів та постійного інтернет-підключення.

Для досягнення поставленої мети необхідно вирішити такі **задачі дослідження**:

- проаналізувати предметну область та дослідити особливості процесу моніторингу стану бджолиних колоній, визначити ключові параметри, що підлягають контролю;

- розробити загальну архітектуру інформаційної системи та обґрунтувати вибір апаратної платформи на базі мікроконтролера ATXMEGA256A3U та модуля ESP8266;

- спроєктувати підсистему збору даних із сенсорів температури, вологості, концентрації CO_2, атмосферного тиску, ваги та акустичної активності колонії;

- розробити програмне забезпечення мікроконтролера для збору, первинної обробки та передачі даних, включаючи алгоритми спектрального аналізу акустичного сигналу на основі FFT;

- спроєктувати та реалізувати вбудований web-сервер на модулі ESP8266 з підтримкою HTTP-протоколу та передачею даних у форматі JSON;

- розробити web-інтерфейс користувача на основі HTML5 та Chart.js для відображення поточних та архівних даних моніторингу у графічному вигляді;

- реалізувати підсистему локального зберігання даних на microSD-карті та організувати доступ до архівних даних через web-інтерфейс;

- розробити підсистему формування діагностичних висновків та сповіщень про відхилення параметрів від норми;

- провести комплексне тестування розробленої інформаційної системи та оцінити її відповідність встановленим вимогам.

Об'єктом дослідження є процес моніторингу стану бджолиної колонії в умовах пасіки, що охоплює автоматизований збір фізичних параметрів вулика, їх обробку та надання пасічнику у зручному для аналізу вигляді через локальну Wi-Fi мережу.

Предметом дослідження є методи, моделі та програмні засоби побудови вбудованої інформаційної системи моніторингу стану бджолиної колонії на базі мікроконтролера ATXMEGA256A3U та модуля ESP8266 з реалізацією вбудованого web-сервера і локального зберігання даних.

Визначення вимог до системи є необхідним етапом проєктування, що забезпечує відповідність розроблюваного рішення потребам кінцевого користувача. Вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги визначають перелік можливостей, які система повинна забезпечувати. Система повинна здійснювати безперервний збір даних із підключених сенсорів з інтервалом не більше 5 хвилин та зберігати їх на microSD-карті у форматі CSV. Вбудований web-сервер повинен забезпечувати доступ до поточних даних моніторингу в режимі реального часу з оновленням кожні 10 секунд. Web-інтерфейс повинен відображати графіки динаміки зміни всіх контрольованих параметрів за обраний період — добу, тиждень або місяць. Система повинна формувати діагностичні висновки про стан бджолиної колонії на основі аналізу сукупності параметрів та генерувати сповіщення при виявленні відхилень від встановлених порогових значень.

Доступ до web-інтерфейсу повинен здійснюватися з будь-якого пристрою, підключеного до Wi-Fi мережі пристрою, без встановлення додаткового програмного забезпечення – виключно через стандартний web-браузер. Система повинна забезпечувати виконання FFT-аналізу акустичного сигналу в реальному часі з виділенням характерних частотних смуг для діагностики стану колонії.

Нефункціональні вимоги стосуються якісних характеристик системи. З точки зору продуктивності web-сервер повинен забезпечувати час відгуку на запит не більше 2 секунд при підключенні до локальної Wi-Fi мережі. З точки зору надійності система повинна зберігати працездатність при температурах від мінус 20°C до плюс 50°C та відносній вологості до 95%, що відповідає реальним умовам розміщення всередині або поблизу вулика. З точки зору енергоефективності загальне споживання пристрою в активному режимі не повинно перевищувати 150 мА при напрузі живлення 5 В, що забезпечує можливість автономної роботи від акумулятора або сонячної панелі. З точки зору зручності використання web-інтерфейс повинен коректно відображатися на екранах різних розмірів – від смартфона до настільного комп'ютера, тобто мати адаптивний дизайн. З точки зору місткості обсяг microSD-карти ємністю 32 ГБ повинен забезпечувати зберігання даних не менше ніж за три повні сезони без необхідності очищення.

Сформульовані мета, задачі та вимоги визначають подальшу структуру бакалаврської роботи і є основою для прийняття проєктних рішень. Особливістю даної роботи є орієнтація на повністю автономну embedded-архітектуру, в якій уся інформаційна система розміщена безпосередньо на пристрої моніторингу. Це принципово відрізняє розроблюване рішення від більшості існуючих систем, що залежать від зовнішніх хмарних платформ, і забезпечує його працездатність навіть за повної відсутності інтернет-підключення на пасіці.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ МОНІТОРИНГУ

2.1 Загальна архітектура інформаційної системи та концепція вбудованого web-сервера

Інформаційна система моніторингу стану бджолиної колонії є вбудованою (embedded) системою, що реалізує повний цикл обробки інформації — від збору фізичних параметрів вулика до надання результатів користувачу у зручному графічному вигляді – без залучення зовнішніх серверів або хмарних платформ. Така архітектура принципово відрізняється від більшості сучасних комерційних рішень і визначає ключові проєктні рішення щодо вибору апаратної платформи, організації програмного забезпечення та способу взаємодії з користувачем.

Архітектура системи є дворівневою та включає два функціонально відокремлені апаратні вузли, що взаємодіють між собою через інтерфейс SPI. Перший вузол – центральний мікроконтролер ATXMEGA256A3U – відповідає за збір даних із підключених сенсорів, їх первинну обробку, виконання алгоритмів аналізу, зберігання на microSD-карті та передачу підготовлених даних до другого вузла [12, 13]. Другий вузол – модуль ESP8266 – виконує функції Wi-Fi адаптера та вбудованого web-сервера, отримуючи дані від ATXMEGA через SPI і надаючи до них доступ підключеним клієнтам через стандартний HTTP-протокол, рис.2.1.

Така розподіленість функцій між двома мікросхемами є обґрунтованим архітектурним рішенням. ATXMEGA256A3U є потужним 8-розрядним мікроконтролером з апаратним модулем DMA, що дозволяє ефективно виконувати обчислювально складні задачі — зокрема 512-точкове FFT для акустичного аналізу — без блокування основного циклу опитування датчиків.

Водночас реалізація стеку TCP/IP та HTTP-сервера на ATXMEGA є надмірно складним завданням, яке невиправдано навантажило б процесор і ускладнило програмне забезпечення. Модуль ESP8266, натомість, є спеціалізованим Wi-Fi SoC із власним процесором та вбудованим стеком TCP/IP, що робить його ідеальним рішенням для організації мережевого доступу при мінімальному навантаженні на основний мікроконтролер.

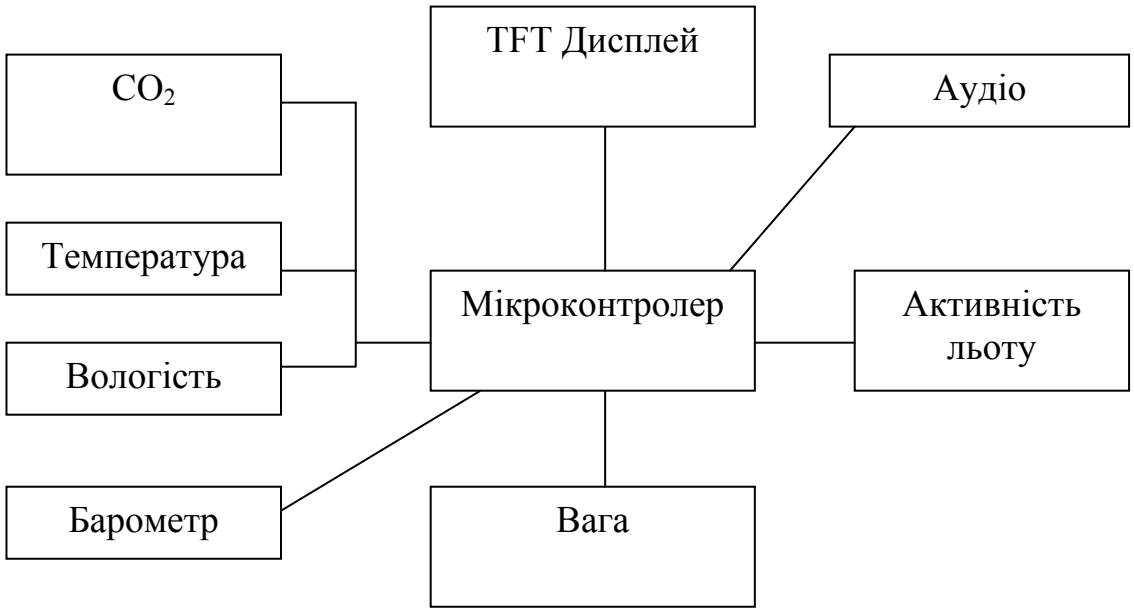

Рисунок 2.1 – Структурна схема вузла збору інформації

Концепція вбудованого web-сервера є центральним архітектурним рішенням даної інформаційної системи. Традиційний підхід до побудови IoT-систем моніторингу передбачає передачу даних на віддалений сервер у мережі Інтернет, де вони зберігаються та обробляються, а користувач отримує доступ до них через хмарну платформу. Натомість у даній системі web-сервер розміщується безпосередньо на пристрої моніторингу – на модулі ESP8266 – і обслуговує запити клієнтів у межах локальної Wi-Fi мережі. ESP8266 створює власну точку доступу Wi-Fi (Access Point mode) або підключається до наявної мережі пасіки (Station mode), після чого стає доступним за фіксованою IP-адресою для будь-якого пристрою в цій мережі [14].

Взаємодія користувача із системою відбувається виключно через стандартний web-браузер без необхідності встановлення будь-якого додаткового програмного забезпечення. Пасічник підключає смартфон, планшет або ноутбук до Wi-Fi мережі пристрою, відкриває браузер, вводить IP-адресу та отримує повноцінний web-інтерфейс із поточними даними моніторингу, графіками динаміки змін параметрів та діагностичними висновками про стан колонії. Оновлення даних відбувається автоматично кожні 10 секунд завдяки асинхронним запитам JavaScript до API web-сервера, рис 2.2.

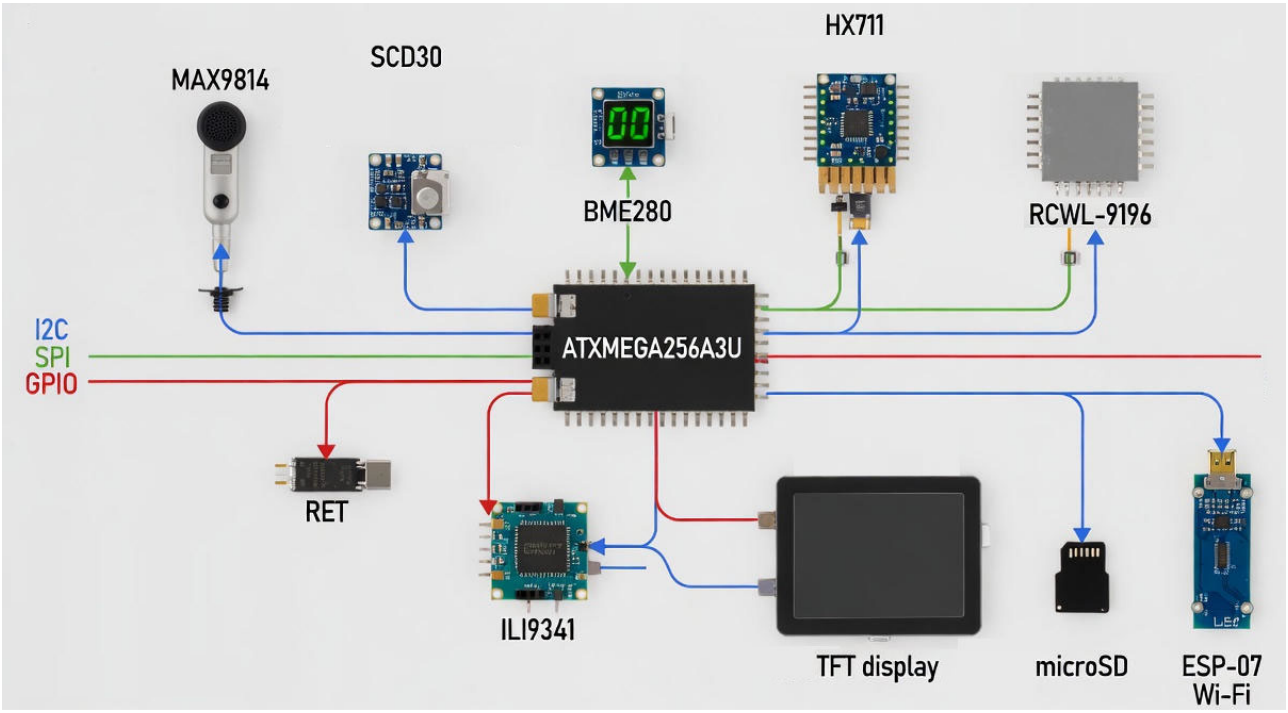

Рисунок 2.2 – Функціональна схема вузла збору інформації

Інформаційні потоки в системі організовані таким чином. ATXMEGA256A3U циклічно опитує підключені датчики з інтервалом 5 хвилин, виконує первинну обробку отриманих значень, здійснює FFT-аналіз акустичного сигналу та формує структуровані записи даних. Кожен запис містить мітку часу, значення всіх вимірюваних параметрів та результати діагностичного аналізу. Сформований запис зберігається на microSD-карті у

форматі CSV та одночасно передається модулю ESP8266 через SPI-інтерфейс для оновлення буфера поточних даних web-сервера. При надходженні HTTP-запиту від клієнта ESP8266 формує відповідь у форматі JSON на основі даних із буфера або зчитує архівні дані з microSD за запитом користувача.

Важливою перевагою обраної архітектури є її повна автономність. Система функціонує незалежно від наявності інтернет-підключення на пасіці, що є критично важливим для реальних умов експлуатації, адже більшість пасік розташована у сільській місцевості зі слабким або нестабільним мобільним зв'язком. Усі дані зберігаються локально на microSD-карті, що унеможливлює їх втрату внаслідок відключення хмарного сервісу або зміни умов надання послуг стороннім постачальником. При цьому у разі наявності інтернет-підключення система може бути розширена для передачі даних на зовнішній сервер без зміни базової архітектури.

Загальна архітектура системи включає такі основні функціональні блоки: блок акустичного моніторингу на базі мікрофонного модуля MAX9814, блок контролю мікроклімату на базі сенсорів SCD30 та BME280, блок зважування на базі тензодатчиків та АЦП HX711, блок моніторингу льоткової активності на базі ToF-сенсора VL6180X та доплерівського радара RCWL-9196, блок зберігання даних на базі модуля microSD, блок бездротової комунікації та web-сервера на базі модуля ESP8266, а також блок живлення із захистом від перевантаження. Центральний мікроконтролер ATXMEGA256A3U здійснює управління всіма переліченими блоками та координує їх спільну роботу відповідно до заданого алгоритму функціонування системи.

Таким чином, обрана архітектура інформаційної системи забезпечує оптимальний розподіл функцій між апаратними компонентами, повну автономність роботи, зручний доступ користувача до даних через стандартний web-браузер та можливість подальшого розширення функціональності. Детальний розгляд кожного функціонального блоку системи наведено в наступних підрозділах бакалаврської роботи.

2.2 Мікроконтролер ATXMEGA256A3U як центральний вузол системи

Вибір мікроконтролера є одним із ключових проєктних рішень при розробленні вбудованої інформаційної системи, оскільки саме він визначає обчислювальні можливості пристрою, набір підтримуваних периферійних інтерфейсів, енергоспоживання та складність програмного забезпечення. У даній системі центральним вузлом обрано 8-розрядний AVR-мікроконтролер ATXMEGA256A3U виробництва Atmel/Microchip, який поєднує високу продуктивність, багатий набір периферії та прийнятну вартість, рис.2.3.

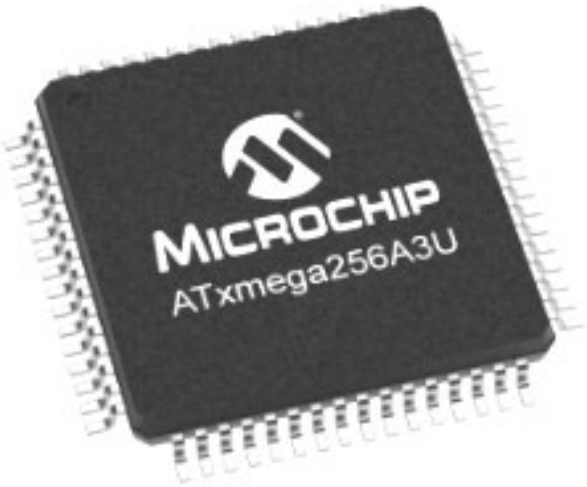

Рисунок 2.3 – Мікроконтролер AtXMega256A3U

ATXMEGA256A3U працює на тактовій частоті до 32 МГц та має такі основні характеристики: 256 КБ Flash-пам'яті програм, 4 КБ EEPROM для зберігання налаштувань та 16 КБ оперативної пам'яті SRAM. Наявність апаратного модуля DMA (Direct Memory Access) є принциповою перевагою цього мікроконтролера для задач даної системи, оскільки дозволяє виконувати передачу великих масивів даних між периферією та оперативною пам'яттю без участі процесорного ядра. Зокрема, DMA використовується для зняття 512 відліків акустичного сигналу з АЦП із частотою дискретизації 3 кГц, що є

необхідною умовою для подальшого виконання FFT-аналізу. Завдяки DMA процесорне ядро залишається вільним для виконання інших задач під час накопичення акустичних даних, що суттєво підвищує загальну ефективність системи.

Мікроконтролер має у своєму складі 12-розрядний АЦП із можливістю вимірювання диференційних сигналів, що забезпечує високу точність оцифрування аналогових сигналів від мікрофонного модуля MAX9814 та тензодатчиків через підсилювач HX711. Наявність до 78 GPIO дозволяє підключити всі периферійні пристрої системи без використання мультиплексорів, що спрощує схемотехнічне рішення та підвищує надійність. Мікроконтролер підтримує апаратні інтерфейси I²C, SPI та UART, що забезпечує ефективне підключення всіх датчиків та модулів системи відповідно до їх специфікацій.

Шина I²C використовується для підключення сенсора CO□, температури та вологості SCD30 за адресою 0x61, датчика атмосферного тиску BME280 за адресою 0x76 та ToF-сенсора льоткової активності VL6180X за адресою 0x29. Усі три пристрої підключені до однієї шини I²C на виводах PD0 (SDA) та PD1 (SCL) з підтягуючими резисторами 4,7 кОм, що є стандартним рішенням для організації мультимайстерної шини. Шина SPI використовується для двох цілей: підключення модуля microSD для локального зберігання даних та взаємодії з модулем ESP8266, який працює у режимі SPI Slave із швидкістю до 10 МГц. Такий розподіл інтерфейсів забезпечує оптимальне завантаження комунікаційних ресурсів мікроконтролера без конфліктів між пристроями.

Важливою характеристикою ATXMEGA256A3U є підтримка апаратного модуля обчислення швидкого перетворення Фур'є. Виконання 512-точкового комплексного FFT займає менше 1 мс при тактовій частоті 32 МГц та середньому струмі споживання менше 8 мА. Це дозволяє здійснювати спектральний аналіз акустичного сигналу вулика в реальному часі з мінімальним впливом на загальний енергетичний баланс системи. Результатом

FFT-аналізу є спектр потужності в восьми характерних частотних смугах діапазону 150–1200 Гц, що відповідають різним фізіологічним станам бджолиної колонії.

Мікроконтролер підтримує кілька режимів зниженого енергоспоживання, що є важливим для забезпечення автономної роботи пристрою від акумулятора або сонячної панелі. У режимі очікування між циклами опитування датчиків ATXMEGA переходить у режим Power-save з відключенням більшості периферійних модулів, залишаючи активним лише таймер реального часу для відліку інтервалу між вимірюваннями. Пробудження здійснюється або за перериванням від таймера, або за сигналом від доплерівського радара RCWL-9196 при виявленні льоткової активності бджіл, що дозволяє оперативно реагувати на зміни в поведінці колонії.

Програмне забезпечення мікроконтролера розроблено мовою C з використанням Code Vision AVR та стандартних бібліотек Atmel Software Framework. Така платформа є добре документованою та широко використовується у промислових вбудованих системах, що забезпечує надійність та передбачуваність поведінки програмного забезпечення в реальних умовах експлуатації. Основний цикл програми реалізує кінцевий автомат із станами ініціалізації, збору даних, обробки, передачі до ESP8266 та переходу в режим очікування.

Таким чином, мікроконтролер ATXMEGA256A3U повністю відповідає вимогам даної інформаційної системи за продуктивністю, набором периферійних інтерфейсів та енергоефективністю. Наявність апаратного DMA, потужного АЦП та підтримки всіх необхідних комунікаційних інтерфейсів робить його оптимальним вибором для реалізації центрального вузла системи моніторингу стану бджолиної колонії.

2.3 Підсистема збору даних: датчики та інтерфейси підключення

Підсистема збору даних є основою інформаційної системи моніторингу, оскільки саме вона забезпечує отримання об'єктивної інформації про стан бджолиної колонії та мікроклімат вулика. Вибір датчиків визначається переліком контрольованих параметрів, сформованим у підрозділі 1.4, та вимогами до точності вимірювань, діапазону робочих температур, енергоспоживання та сумісності з інтерфейсами мікроконтролера ATXMEGA256A3U. У даній системі реалізовано п'ять функціональних блоків збору даних: блок акустичного моніторингу, блок контролю мікроклімату, блок зважування, блок моніторингу льоткової активності та блок зовнішніх метеопараметрів.

Блок акустичного моніторингу реалізований на базі мікрофонного модуля MAX9814 із вбудованим підсилювачем із автоматичним регулюванням підсилення (AGC), рис. 2.4. Модуль живиться від напруги 3,3 В при споживаному струмі близько 3 мА, що відповідає вимогам енергоефективності системи. Підсилення встановлено на рівні 60 дБ, що забезпечує достатню чутливість для реєстрації звукового тиску бджіл у діапазоні 0,1–5 Па. Вихідний аналоговий сигнал модуля підключений до входу PA0 АЦП мікроконтролера. Сигнал дискретизується з частотою 3 кГц із роздільною здатністю 12 біт, що забезпечує аналіз акустичного спектру в діапазоні 150–1200 Гц — саме тому діапазоні, де зосереджені інформативні акустичні маркери стану колонії. Завдяки модулю DMA накопичення 512 відліків відбувається без навантаження на процесорне ядро, після чого виконується FFT-аналіз із виділенням восьми характерних частотних смуг.

Рисунок 2.4 – Модуль MAX9814

Принцип роботи AGC модуля MAX9814 полягає у динамічному регулюванні коефіцієнта підсилення залежно від рівня вхідного сигналу: при гучних звуках підсилення автоматично знижується для запобігання перевантаженню, при слабких – збільшується для забезпечення стабільного вихідного рівня. Співвідношення атаки та відпускання за замовчуванням становить 1:4000, що дозволяє адаптуватися до раптових змін акустичного середовища вулика, наприклад при спалахах активності бджіл у момент роїння або при сигналах тривоги. Діапазон частот модуля становить 20 Гц – 20 кГц, гармонійні спотворення – типово 0,04 %, що забезпечує високу якість сигналу для подальшої цифрової обробки.

Блок контролю мікроклімату включає два датчики, підключені до спільної шини I²C мікроконтролера на виводах PD0 та PD1 з підтягуючими резисторами 4,7 кОм. Основним є сенсор SCD30 виробництва Sensirion, що здійснює одночасне вимірювання концентрації CO_2, температури та відносної вологості повітря всередині вулика [15]. Принцип вимірювання CO_2 базується на двоканальній недисперсійній інфрачервоній спектроскопії (NDIR), де один канал вимірює поглинання інфрачервоного випромінювання молекулами CO_2, а референтний канал компенсує зовнішні впливи для довготривалої стабільності показань, рис. 2.5. Діапазон вимірювання CO_2 становить 400-10000 ppm з похибкою ±(30 ppm + 3 %), що повністю перекриває фізіологічні межі концентрації CO_2 у вулику – від фонових 400 ppm у літній період до

8000-9000 ppm під час зимівлі. Датчик підключений до шини I²C за адресою 0x61 та здійснює вимірювання кожні 5 хвилин синхронно з основним циклом опитування системи.

Рисунок 2.5 – Давач CO_2

Додатково до SCD30 на шині I²C за адресою 0x76 підключено датчик BME280 виробництва Bosch Sensortec, що вимірює атмосферний тиск із точністю ±1 hPa, а також температуру та відносну вологість зовнішнього середовища, рис.2.6. Основним призначенням BME280 у даній системі є корекція показань CO_2 сенсора SCD30 на висоту розташування пасіки над рівнем моря, оскільки концентрація CO_2 залежить від атмосферного тиску. Крім того, дані атмосферного тиску дозволяють корелювати поведінку колонії з метеорологічними умовами – зокрема, зниження тиску перед наближенням грози часто супроводжується підвищенням активності бджіл, що відображається в акустичних та вагових даних [16].

Рисунок 2.6 – Давач BME280

Блок зважування реалізований на базі чотирьох тензодатчиків типу «балка на вигин» номінальною вантажопідйомністю 50 кг кожен, що утворюють вагову платформу під вуликом. Два тензодатчики об'єднані у напівміст на каналі A підсилювача-АЦП HX711 із коефіцієнтом підсилення 128, інші два — на каналі B із коефіцієнтом підсилення 64. Така двоканальна схема підключення забезпечує незалежне вимірювання маси з реальною роздільною здатністю близько 10 г при типовому навантаженні 25–40 кг, що є достатнім для виявлення навіть незначних змін маси вулика, пов'язаних із добовою динамікою медозбору [18]. Мікросхема HX711 підключена до мікроконтролера через двопровідний послідовний інтерфейс на виводах PB0 (CLK) та PB1 (DATA) та не потребує зовнішнього джерела опорної напруги завдяки вбудованому прецизійному джерелу, рис. 2.7.

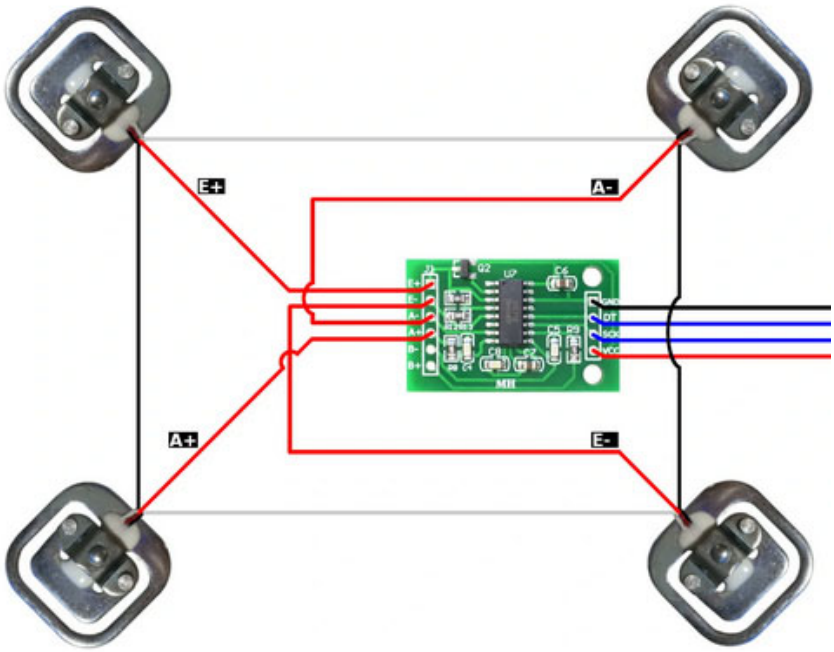

Рисунок 2.7 – Давач HX711

Блок моніторингу льоткової активності включає два незалежні сенсори, що забезпечують комплексний контроль руху бджіл у зоні льотка. ToF-сенсор VL6180X виробництва STMicroelectronics підключений до шини I²C за адресою 0x29 та вимірює відстань до об'єктів у діапазоні 0–200 мм із

дискретністю 1 мм, що дозволяє точно реєструвати кожну бджолу, що проходить через льоток, і підраховувати кількість вильотів та прильотів із пропускною здатністю до 300 бджіл за хвилину без пропусків, рис. 2.8 [19].

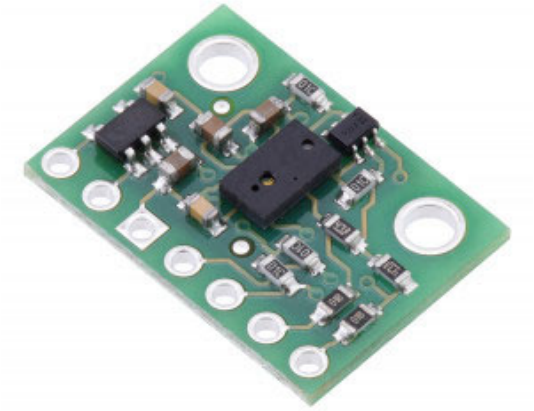

Рисунок 2.8 – Давач VL6180X

Доплерівський радарний модуль RCWL-9196 з цифровим виходом GPIO використовується як тригер пробудження системи при початку льоткової активності - його сигнал виводить мікроконтролер із режиму зниженого енергоспоживання та ініціює позаплановий цикл вимірювань при виявленні руху в зоні льотка радіусом близько 1 м, рис.2.9.

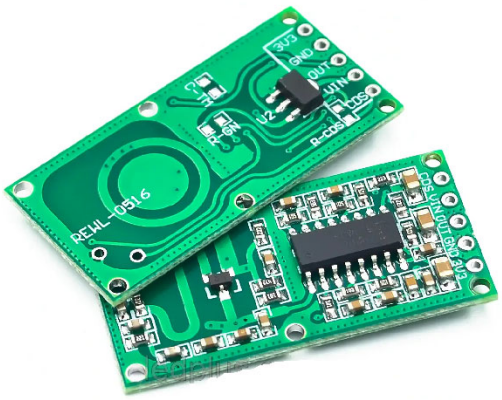

Рисунок 2.9 – Давач RCWL-9196

Усі датчики підсистеми збору даних живляться від стабілізованої напруги 3,3 В, яка формується вбудованим лінійним стабілізатором плати. Загальне споживання підсистеми в активному режимі вимірювань не

перевищує 80 мА, що забезпечує тривалу автономну роботу від акумулятора або сонячної панелі. Між циклами вимірювань частина датчиків переводиться у режим очікування з мінімальним споживанням — зокрема SCD30 підтримує режим Idle з споживанням менше 0,5 мА.

Таким чином, підсистема збору даних забезпечує комплексний контроль усіх ключових параметрів стану бджолиної колонії: акустичної активності, концентрації CO_2, температури, вологості, атмосферного тиску, маси вулика та льоткової активності. Поєднання цих параметрів дає змогу сформувати повноцінну картину стану колонії та виявляти характерні патерни, що відповідають різним фізіологічним станам – від нормальної активності до ройового настрою чи ознак захворювання.

2.4. Модуль бездротового зв'язку ESP8266 та організація Wi-Fi мережі

Модуль бездротового зв'язку є ключовим компонентом інформаційної системи, що забезпечує взаємодію між пристроєм моніторингу та користувачем. Саме завдяки цьому модулю реалізується концепція вбудованого web-сервера, – пасічник отримує доступ до даних моніторингу через звичайний браузер без будь-якого додаткового програмного забезпечення. У даній системі функції Wi-Fi адаптера та web-сервера покладено на модуль ESP-07 на базі мікросхеми ESP8266 виробництва Espressif Systems.

ESP8266 є однокристальною системою (SoC), що поєднує 32-розрядний процесор Tensilica L106 із тактовою частотою 80 МГц, вбудований стек протоколів Wi-Fi IEEE 802.11 b/g/n, контролер TCP/IP та 64 КБ оперативної пам'яті для інструкцій і 96 КБ для даних, рис.2.10. Наявність власного процесорного ядра та повноцінного мережевого стеку дозволяє ESP8266

самостійно обслуговувати HTTP-з'єднання без залучення ресурсів центрального мікроконтролера ATXMEGA256A3U. Це є принциповою архітектурною перевагою — ATXMEGA повністю зосереджений на задачах збору та обробки даних, тоді як ESP8266 незалежно обслуговує мережеві запити клієнтів.

Рисунок 2.10 – Модуль ESP-07

Взаємодія між ATXMEGA256A3U та ESP8266 організована через інтерфейс SPI, де ATXMEGA виступає майстром (Master), а ESP8266 — підлеглим (Slave). Швидкість обміну по SPI становить до 10 МГц, що забезпечує передачу пакету даних одного циклу вимірювань — об'ємом близько 512 байт — за час менше 0,5 мс. Такий підхід є значно ефективнішим порівняно з використанням UART-інтерфейсу через команди AT, який традиційно застосовується у більшості проєктів на базі ESP8266, оскільки виключає накладні витрати на текстовий протокол та забезпечує детерміновану затримку передачі. Лінії SPI підключені до виводів PC4 (SS), PC5 (MOSI), PC6 (MISO) та PC7 (SCK) мікроконтролера, рис. 2.11.

ESP8266 підтримує три режими роботи Wi-Fi мережі: режим точки доступу (Access Point, AP), режим клієнта (Station, STA) та комбінований режим (AP+STA). У даній системі за замовчуванням використовується режим точки доступу, в якому ESP8266 самостійно створює Wi-Fi мережу з

фіксованим ідентифікатором (SSID) та паролем, налаштованими при першому запуску пристрою. Цей режим є найбільш надійним з точки зору автономності – система завжди доступна незалежно від наявності зовнішньої Wi-Fi інфраструктури на пасіці. Пасічник підключає свій смартфон або планшет безпосередньо до точки доступу пристрою та отримує доступ до web-інтерфейсу за фіксованою адресою 192.168.4.1.

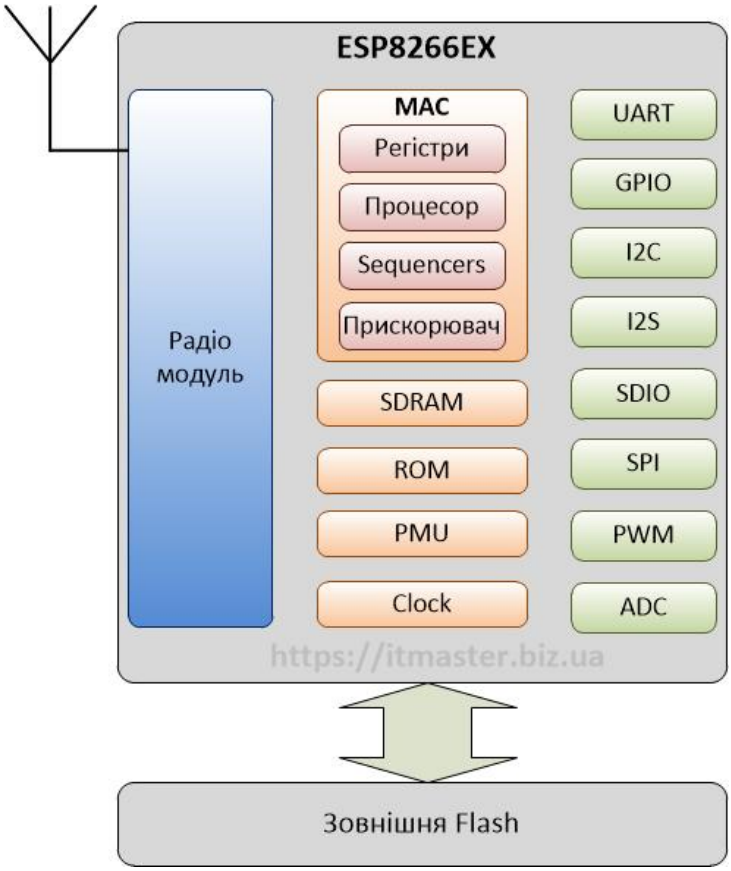

Рисунок 2.11 – Модуль ESP-07, структурна схема

Комбінований режим AP+STA передбачений для ситуацій, коли на пасіці наявна власна Wi-Fi мережа з виходом в Інтернет. У цьому режимі ESP8266 одночасно підтримує власну точку доступу для прямого підключення пасічника та підключається до зовнішньої мережі як клієнт, що відкриває можливість передачі даних на віддалений сервер або відправлення сповіщень електронною поштою. Перемикання між режимами здійснюється через

налаштування web-інтерфейсу без перепрограмування пристрою, що забезпечує гнучкість розгортання системи в різних умовах.

Параметри Wi-Fi мережі, створюваної пристроєм, налаштовані з урахуванням умов експлуатації на пасіці. Потужність передавача встановлена на рівні 17 дБм, що забезпечує стійке покриття в радіусі до 50 метрів на відкритій місцевості — достатньо для більшості пасік із компактним розташуванням вуликів. Використовується захищений протокол WPA2-PSK для запобігання несанкціонованому доступу до даних моніторингу. Канал Wi-Fi встановлюється автоматично при ініціалізації на основі аналізу завантаженості доступних каналів для мінімізації інтерференції з іншими бездротовими пристроями.

Живлення модуля ESP8266 є окремим важливим аспектом проєктування. Модуль потребує стабілізованої напруги 3,3 В і може споживати імпульсний струм до 400 мА під час передачі Wi-Fi пакетів, що значно перевищує можливості більшості вбудованих стабілізаторів. Для забезпечення стабільного живлення передбачено окремий імпульсний стабілізатор напруги з максимальним струмом 600 мА та блокуючі конденсатори ємністю 100 мкФ і 100 нФ безпосередньо поблизу виводів живлення модуля. У режимі очікування між обробкою HTTP-запитів ESP8266 переводиться у режим Modem Sleep із споживанням близько 15 мА, що суттєво знижує загальне енергоспоживання системи в умовах рідкісних звернень користувача.

Для підвищення надійності системи реалізовано механізм апаратного скидання ESP8266 з боку ATXMEGA через вивід Reset модуля. У разі відсутності відповіді від ESP8266 протягом визначеного тайм-ауту мікроконтролер автоматично виконує апаратне скидання модуля та повторну ініціалізацію Wi-Fi мережі. Такий механізм забезпечує самовідновлення системи при виникненні збоїв у роботі мережевого стеку ESP8266, що є критично важливим для безперервної роботи пристрою в умовах пасіки без постійного обслуговування.

Таким чином, модуль ESP8266 у режимі SPI Slave з організацією власної точки доступу Wi-Fi є оптимальним рішенням для реалізації бездротового доступу до даних моніторингу в умовах автономної вбудованої системи. Розподіл функцій між ATXMEGA256A3U та ESP8266 забезпечує ефективне використання обчислювальних ресурсів обох мікросхем та високу надійність системи в цілому.

2.5. Організація зберігання даних на microSD-карті

Локальне зберігання даних є невід'ємною складовою інформаційної системи моніторингу, оскільки забезпечує накопичення історичних даних для ретроспективного аналізу стану бджолиної колонії, а також гарантує збереження інформації в умовах відсутності постійного підключення користувача до web-інтерфейсу. У даній системі зберігання даних організовано на microSD-карті, підключеній до мікроконтролера ATXMEGA256A3U через інтерфейс SPI, що забезпечує надійний та енергоефективний доступ до носія інформації, рис.2.12.

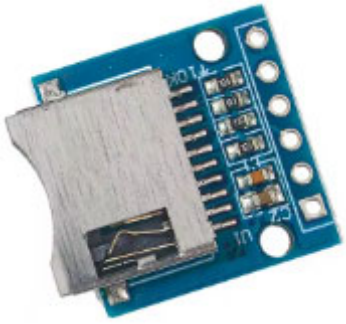

Рисунок 2.12 – Модуль microSD

Модуль microSD підключений до мікроконтролера через окремий чіп-селект (CS), що дозволяє розмістити його на тій самій шині SPI, що й модуль ESP8266, без конфліктів між пристроями. Управління вибором активного

пристрою на шині здійснюється мікроконтролером шляхом перемикання відповідних ліній CS, що є стандартним підходом для організації мультиімейстерних SPI-систем. Швидкість обміну з модулем microSD встановлена на рівні 4 МГц, що забезпечує достатню пропускну здатність для запису одного рядка CSV-даних за час менше 5 мс – значно менше інтервалу між циклами вимірювань.

Для роботи з файловою системою microSD у програмному забезпеченні мікроконтролера використовується файлова система FAT32, яка підтримується всіма операційними системами та забезпечує можливість безпосереднього читання збережених файлів на персональному комп'ютері після вилучення карти з пристрою. Це є важливою практичною перевагою – пасічник може отримати повний архів даних не лише через web-інтерфейс, а й безпосередньо, підключивши карту до комп'ютера через стандартний кард-рідер. Карта форматується у FAT32 при першому запуску системи, якщо файлова система не виявлена, що забезпечує самоналаштування пристрою без втручання користувача.

Структура зберігання даних на microSD-карті організована у вигляді ієрархії директорій та файлів. У кореневому каталозі створюється директорія з назвою пристрою, всередині якої формуються окремі файли для кожного місяця спостережень за шаблоном імені YYYY-MM.csv. Такий підхід дозволяє уникнути утворення надмірно великих файлів та спрощує навігацію по архіву даних. Кожен файл CSV містить рядок заголовку з назвами полів та рядки даних, що записуються з інтервалом 5 хвилин. Один рядок даних містить такі поля: мітка часу у форматі UNIX timestamp, температура всередині вулика в градусах Цельсія, відносна вологість у відсотках, концентрація CO_2 у ppm, атмосферний тиск у гектопаскалях, маса вулика у грамах, кількість вильотів бджіл за останні 5 хвилин, домінантна частота акустичного спектру в герцах, індекс стану колонії та код діагностичного висновку.

Розрахунок необхідного обсягу носія підтверджує достатність карти ємністю 32 ГБ для тривалої роботи системи. Один рядок CSV-даних займає в середньому близько 120 байт. При інтервалі запису 5 хвилин за добу формується 288 рядків, що становить близько 34 КБ на добу або приблизно 12 МБ на рік. Таким чином, карта ємністю 32 ГБ забезпечує зберігання достатньої кількості спостережень у форматі CSV, що практично необмежено з точки зору потреб системи моніторингу. Додатково на карті зберігаються бінарні файли спектрів FFT – 512 значень типу int16 кожні 15 хвилин, що додає близько 35 МБ на рік. Загальний річний приріст даних не перевищує 50 МБ, що підтверджує достатність обраного обсягу носія.

Окремою задачею є забезпечення цілісності даних при записі. Оскільки мікроконтролер може бути знеструмлений у будь-який момент, у тому числі в процесі запису рядка даних, реалізовано механізм атомарного запису з використанням тимчасового буфера. Повний рядок даних спочатку формується в оперативній пам'яті мікроконтролера, після чого записується на карту за одну операцію запису. Після успішного завершення запису у спеціальному службовому файлі оновлюється контрольна сума останнього запису. При наступному запуску системи перевіряється цілісність останнього запису – у разі виявлення пошкодженого рядку він видаляється, що запобігає накопиченню некоректних даних в архіві.

Доступ до архівних даних через web-інтерфейс організований таким чином. При надходженні HTTP-запиту від браузера з параметрами вибраного часового діапазону ESP8266 передає запит мікроконтролеру через SPI. ATXMEGA відкриває відповідний CSV-файл на карті, зчитує рядки за вказаний діапазон дат, формує відповідь у форматі JSON та передає її назад ESP8266 для відправки клієнту. Для оптимізації швидкості відповіді реалізовано індексний файл, що зберігає зміщення початку кожної доби в CSV-файлах, що дозволяє здійснювати пошук потрібного часового діапазону без послідовного перегляду всього файлу.

Таким чином, організація зберігання даних на microSD-карті забезпечує надійне накопичення повного обсягу вимірювань протягом необмеженого практикою терміну експлуатації, зручний доступ до архівних даних як через web-інтерфейс системи, так і безпосередньо через файлову систему FAT32, а також цілісність даних при несподіваному відключенні живлення.

2.6 Проєктування структури вбудованого web-сервера

Вбудований web-сервер є центральним програмним компонентом інформаційної системи, що реалізує інтерфейс між апаратною частиною пристрою моніторингу та користувачем. На відміну від повноцінних серверних рішень на кшталт Apache або Nginx, вбудований web-сервер функціонує в умовах жорстких обмежень оперативної пам'яті та обчислювальних ресурсів модуля ESP8266, що вимагає ретельного проєктування його архітектури з урахуванням цих обмежень. У даному підрозділі розглядається структура web-сервера, організація маршрутизації запитів, формат API та механізми забезпечення продуктивності, рис.2.12.

Web-сервер функціонує на модулі ESP8266 та прослуховує вхідні HTTP-з'єднання на стандартному порту 80. Архітектура сервера побудована за принципом однопотокової обробки запитів із неблокуючим введенням-виведенням на основі механізму зворотних викликів (callbacks) вбудованого SDK ESP8266 RTOS. Такий підхід дозволяє ефективно обслуговувати кілька одночасних з'єднань без виділення окремого стеку пам'яті для кожного клієнта, що є критично важливим в умовах обмежених 96 КБ оперативної пам'яті модуля. Максимальна кількість одночасних з'єднань обмежена чотирма, що є достатнім для типового сценарію використання системи одним пасічником із кількох пристроїв.

Структура web-сервера включає чотири основні компоненти: модуль маршрутизації запитів, модуль обробників endpoints, модуль формування JSON-відповідей та модуль статичних ресурсів. Модуль маршрутизації аналізує вхідний HTTP-запит, визначає метод (GET або POST) та URI ресурсу і передає запит відповідному обробнику. Таблиця маршрутів зберігається у Flash-пам'яті ESP8266 як масив структур, що містять рядок URI, метод та вказівник на функцію-обробник.

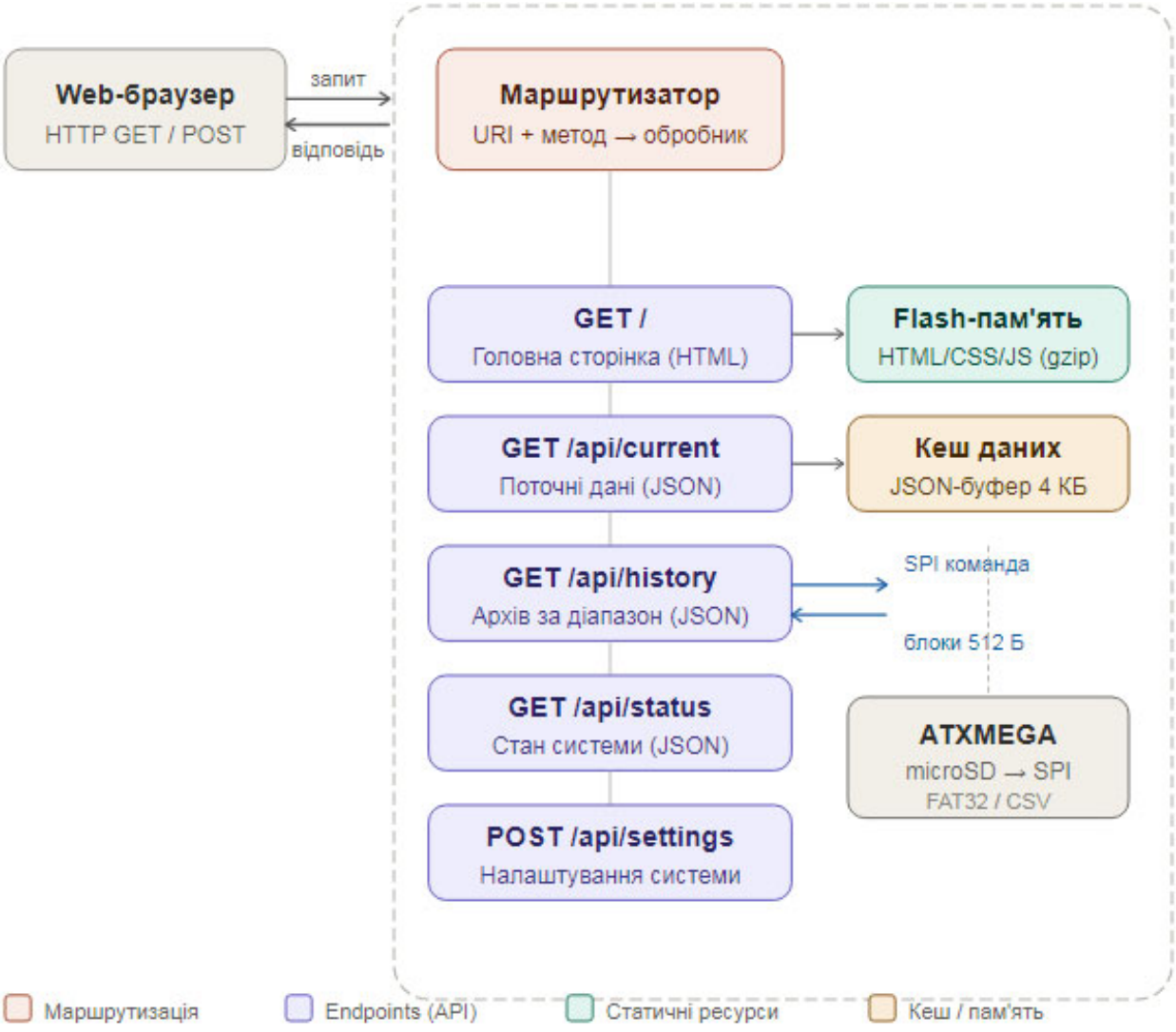

Рисунок 2.12 – Архітектура web-сервера на ESP8266

API web-сервера організоване за принципом REST та включає такі endpoints. Запит GET до кореневого URI «/» повертає головну сторінку web-інтерфейсу у вигляді HTML-документа, що зберігається у Flash-пам'яті

ESP8266 у стисненому форматі gzip для мінімізації обсягу переданих даних та прискорення завантаження сторінки. Запит GET до «/api/current» повертає JSON-об'єкт із поточними значеннями всіх вимірюваних параметрів, що оновлюється кожні 10 секунд на стороні браузера. Запит GET до «/api/history» з параметрами дати початку та кінця діапазону повертає масив JSON-об'єктів із архівними даними за вказаний період, зчитаними з microSD-карти. Запит GET до «/api/status» повертає службову інформацію про стан системи — час роботи, кількість записів на карті, рівень сигналу Wi-Fi та напругу живлення. Запит POST до «/api/settings» дозволяє змінювати налаштування системи – порогові значення параметрів для сповіщень, інтервал вимірювань та параметри Wi-Fi мережі.

Управління пам'яттю є критичним аспектом реалізації web-сервера на ESP8266. Для формування JSON-відповідей використовується статичний буфер розміром 4 КБ, що розміщується у глобальній пам'яті та повторно використовується для кожного запиту. Такий підхід виключає фрагментацію купи (heap fragmentation), яка є поширеною проблемою при динамічному виділенні пам'яті у довготривалих вбудованих застосунках. Для передачі великих масивів архівних даних реалізовано потокову передачу з буфером 512 байт – дані зчитуються з microSD-карти та передаються клієнту частинами без необхідності розміщення всього масиву в оперативній пам'яті одночасно.

Забезпечення продуктивності web-сервера досягається кількома механізмами. По-перше, статичні ресурси web-інтерфейсу – HTML, CSS та JavaScript – зберігаються у Flash-пам'яті ESP8266 у стисненому форматі gzip та передаються клієнту з відповідним заголовком Content-Encoding, що дозволяє браузеру автоматично розпаковувати їх на стороні клієнта. По-друге, для відповідей, що не змінюються часто, встановлюється заголовок Cache-Control з директивою max-age, що дозволяє браузеру кешувати статичні ресурси та не завантажувати їх повторно при кожному оновленні сторінки. По-третє, поточні дані моніторингу кешуються в оперативній пам'яті ESP8266 у вигляді

готового JSON-рядка, який формується мікроконтролером ATXMEGA при кожному циклі вимірювань та передається через SPI. При надходженні запиту до «/api/current» сервер просто відправляє готовий рядок без додаткової обробки, що забезпечує мінімальний час відгуку.

Взаємодія між ESP8266 та ATXMEGA256A3U у контексті обробки запитів організована таким чином. ESP8266 отримує HTTP-запит від клієнта, аналізує URI та параметри запиту та визначає, чи може він обслужити запит самостійно з власного кешу, чи необхідно звернутися до ATXMEGA. Для запитів поточних даних ESP8266 відповідає з кешу без взаємодії з ATXMEGA. Для запитів архівних даних ESP8266 формує команду для ATXMEGA у вигляді бінарного пакету через SPI, ATXMEGA зчитує відповідні дані з microSD-карти та повертає їх ESP8266 блоками по 512 байт, а ESP8266 передає ці блоки безпосередньо клієнту у потоковому режимі.

Таким чином, спроєктована архітектура вбудованого web-сервера забезпечує ефективне використання обмежених ресурсів модуля ESP8266, достатню продуктивність для обслуговування запитів користувача в реальному часі та надійну взаємодію з центральним мікроконтролером при обробці запитів архівних даних.

2.7 Проєктування web-інтерфейсу користувача

Web-інтерфейс користувача є кінцевою точкою взаємодії між інформаційною системою моніторингу та пасічником. Від якості та зручності інтерфейсу безпосередньо залежить практична цінність усієї системи — навіть найдосконаліший алгоритм аналізу стану колонії не матиме практичного значення, якщо результати його роботи відображаються у незручному або незрозумілому вигляді. Тому проєктування web-інтерфейсу є самостійним і

важливим етапом розроблення інформаційної системи, що передує програмній реалізації.

Ключовою вимогою до web-інтерфейсу є його повна функціональність у межах обмежень вбудованого web-сервера на ESP8266. Весь код інтерфейсу — HTML, CSS та JavaScript — повинен зберігатися у Flash-пам'яті модуля у стисненому вигляді та завантажуватися клієнтом одноразово при першому відкритті сторінки. Після завантаження інтерфейс функціонує як односторінковий застосунок (Single Page Application, SPA), що взаємодіє з сервером виключно через асинхронні JSON-запити до API без повного перезавантаження сторінки. Такий підхід мінімізує навантаження на web-сервер та забезпечує плавну роботу інтерфейсу навіть при обмеженій швидкості Wi-Fi з'єднання.

Другою ключовою вимогою є адаптивність інтерфейсу до різних розмірів екрану. Пасічник може підключатися до системи як зі смартфона безпосередньо на пасіці, так і з планшета або ноутбука вдома при аналізі архівних даних. Інтерфейс повинен коректно відображатися на екранах шириною від 320 пікселів до 1920 пікселів без горизонтального прокручування та без необхідності масштабування сторінки. Для реалізації адаптивності використовується CSS Flexbox із автоматичним перенесенням карток параметрів у кілька рядків на вузьких екранах.

Структура web-інтерфейсу організована у вигляді чотирьох функціональних екранів, між якими користувач переміщується через навігаційне меню у верхній частині сторінки. Навігаційна панель із фіолетовим фоном та назвою системи відображається на всіх екранах і забезпечує швидкий перехід між розділами. Активний розділ виділяється підсвічуванням у навігаційному меню для орієнтації користувача.

Перший екран - «Поточний стан» - є головним і відображається при першому відкритті сторінки (рис. 2.13). Він містить шість карток поточних значень вимірюваних параметрів, розміщених у два рядки по три картки:

температура, вологість та концентрація CO_2 у першому рядку; атмосферний тиск, маса вулика та льоткова активність у другому. Кожна картка містить назву параметра, числове значення великим шрифтом для зручного читання з відстані, одиницю вимірювання, кольоровий індикатор стану – зелений при нормі, жовтий при наближенні до порогового значення, червоний при його перевищенні – та стрілку тренду, що показує напрям зміни параметра порівняно з попереднім виміром. У нижній частині екрану розміщується блок діагностичного висновку про загальний стан колонії з текстовим описом виявлених відхилень та рекомендаціями пасічнику. Дані оновлюються автоматично кожні 10 секунд через асинхронні запити до endpoint «/api/current», що відображається у рядку стану внизу сторінки.

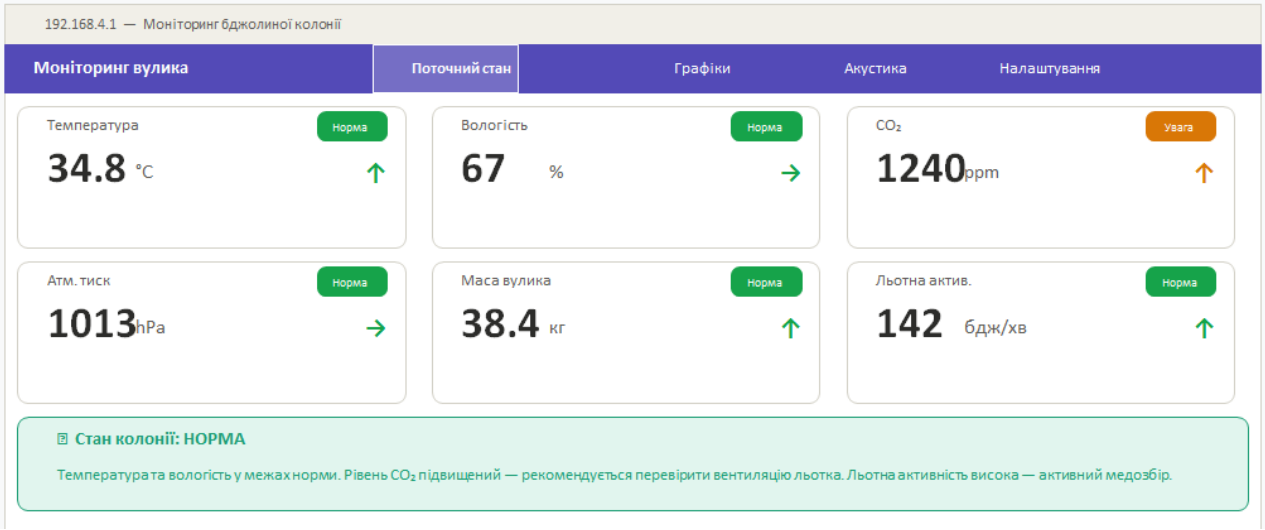

Рисунок 2.13 – Вікно «Поточний стан»

Другий екран - «Графіки» - призначений для аналізу динаміки змін параметрів у часі (рис. 2.14). Екран містить чотири інтерактивні лінійні графіки, побудовані бібліотекою Chart.js: температура, маса вулика, концентрація CO_2 та льотна активність. Графіки розміщені у сітці два на два для одночасного перегляду всіх ключових параметрів. Користувач може обирати часовий діапазон відображення – остання доба, останній тиждень або

останній місяць – через перемикач у верхній частині екрану. При зміні діапазону браузер надсилає запит до endpoint «/api/history» з відповідними параметрами дат і отримує масив даних для перебудови графіків. Кожен графік має власне колірне кодування для швидкої ідентифікації: температура відображається фіолетовим, маса – зеленим, CO_2 – янтарним, льотна активність – кораловим кольором. Плавні криві графіків забезпечуються опцією lineSmooth бібліотеки Chart.js, що спрощує сприйняття загальних тенденцій зміни параметрів.

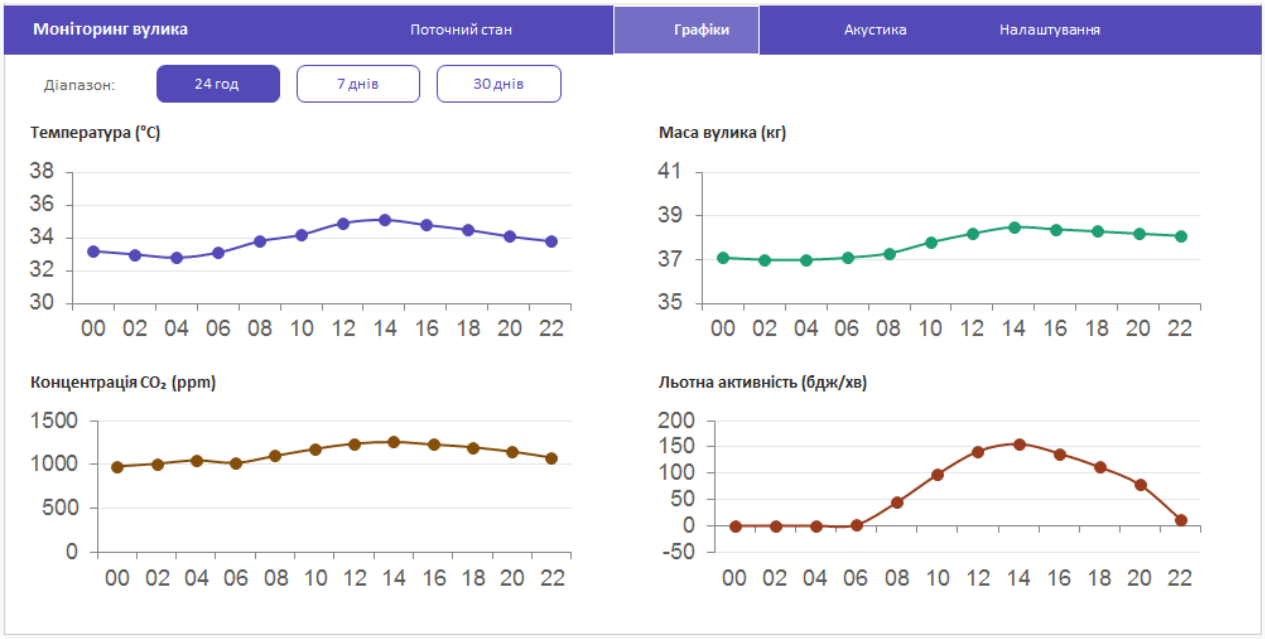

Рисунок 2.14 – Вікно «Графіки»

Третій екран - «Акустика» - відображає результати FFT-аналізу звукового середовища вулика (рис. 2.15). Основним елементом екрану є стовпчаста діаграма спектру потужності у восьми частотних смугах діапазону 150–1200 Гц. Стовпці діаграми забарвлені відповідно до фізіологічного значення кожної частотної смуги: зелений колір для смуг 150-300 Гц відповідає нормальній активності та спокою колонії, блакитний для смуги 300-500 Гц – вентиляції гнізда, янтарний для смуги 400–600 Гц - ройовому настрою, темно-червоний для смуги 600-800 Гц – відсутності матки, яскраво-

червоний для смуг 800-1200 Гц – стану тривоги або стресу. Праворуч від діаграми розміщена легенда з поясненням значення кожного кольору. У нижній частині екрану відображається текстовий акустичний діагноз із поточним станом колонії на основі аналізу домінуючих частотних смуг.

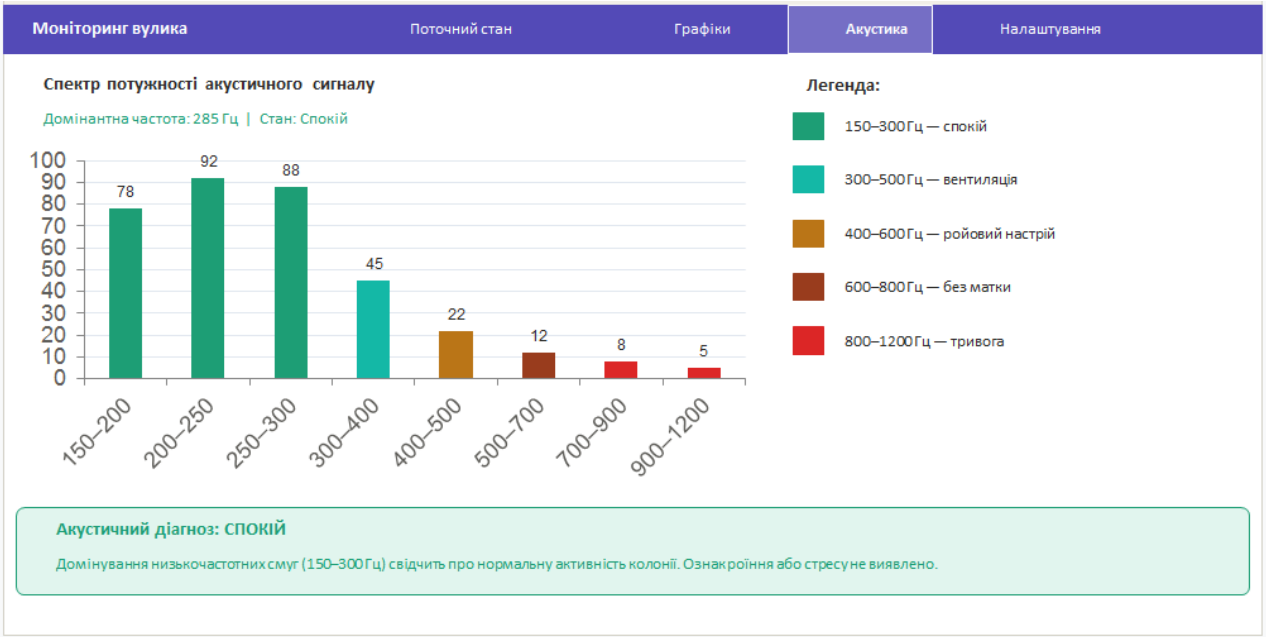

Рисунок 2.15 – Вікно «Акустика»

Четвертий екран - «Налаштування» - дозволяє змінювати конфігурацію системи через форму з полями введення, рис.2.16. Налаштовуваними параметрами є порогові значення для кожного контрольованого параметра, при перевищенні яких формуються сповіщення, інтервал між вимірюваннями, а також параметри Wi-Fi мережі — назва точки доступу та пароль. Збереження налаштувань здійснюється через POST-запит до endpoint «/api/settings», після чого нові значення записуються у Flash-пам'ять ESP8266 та набувають чинності без перезапуску пристрою.

Для реалізації графіків обрано бібліотеку Chart.js версії 3, яка є компактною – близько 60 КБ у стисненому вигляді – та не потребує зовнішніх залежностей. Бібліотека зберігається разом із кодом інтерфейсу у Flash-пам'яті ESP8266 та завантажується клієнтом разом із головною сторінкою. Такий

підхід забезпечує повну автономність інтерфейсу без звернення до зовнішніх CDN, що є критично важливим в умовах відсутності інтернет-підключення на пасіці.

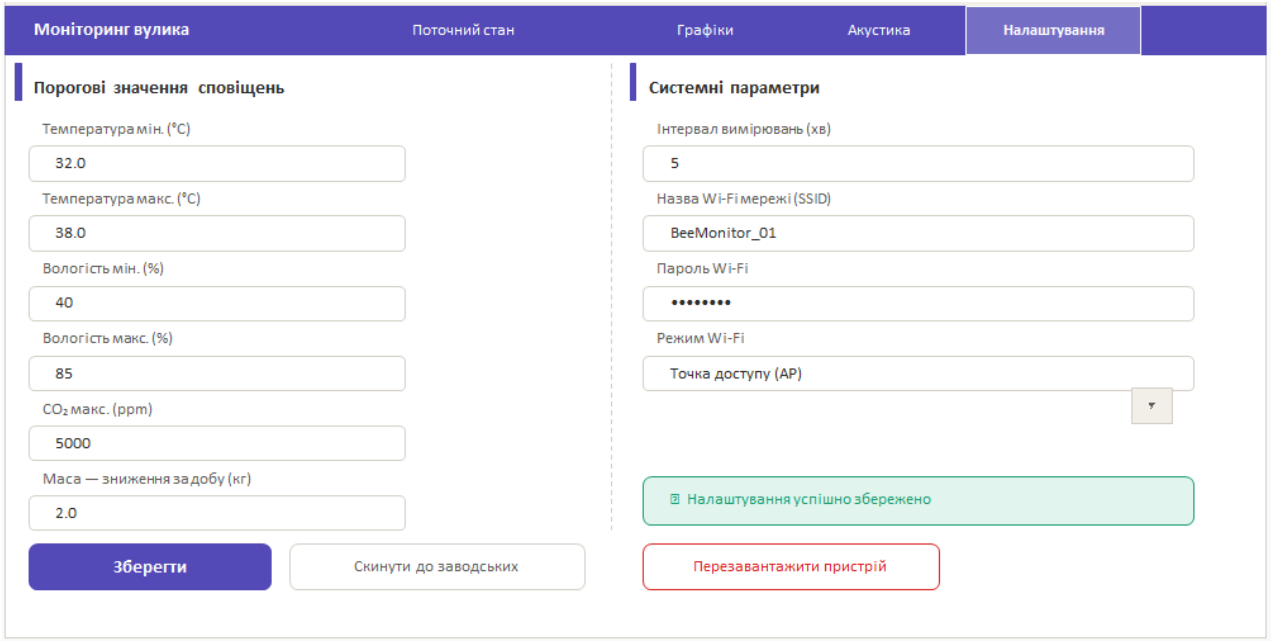

Рисунок 2.16 – Вікно «Налаштування»

Таким чином, спроєктований web-інтерфейс забезпечує зручний та інформативний доступ пасічника до всіх даних моніторингу через чотири функціональні екрани, реалізує адаптивне відображення на різних пристроях та функціонує повністю автономно без залежності від зовнішніх сервісів.

3 РОЗРОБЛЕННЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Програмне забезпечення мікроконтролера

Програмне забезпечення мікроконтролера ATXMEGA256A3U є центральним компонентом інформаційної системи, що реалізує логіку збору даних із підключених сенсорів, їх первинну обробку та передачу до модуля ESP8266. Розроблення програмного забезпечення виконується у середовищі CodeVisionAVR з використанням мови програмування C та стандартних бібліотек, що входять до складу середовища. CodeVisionAVR є спеціалізованим інструментом для розроблення програмного забезпечення мікроконтролерів AVR та забезпечує зручну роботу з периферійними модулями ATXMEGA через вбудований генератор ініціалізаційного коду CodeWizardAVR. При створенні нового проєкту CodeWizardAVR автоматично генерує ініціалізаційний код та підключає відповідний заголовний файл мікроконтролера залежно від обраного цільового пристрою [22].

Заголовний файл мікроконтролера підключається автоматично через директиву, що формується середовищем CodeVisionAVR при виборі ATXMEGA256A3U як цільового пристрою проєкту:

```
#include <atxmega256a3u.h>
#include <delay.h>
#include <i2c.h>
#include <spi.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

// Адреси датчиків на шині I2C
#define SCD30_ADDR    0x61
#define BME280_ADDR   0x76
#define VL6180X_ADDR  0x29

// Інтервал вимірювань (секунди)
#define MEAS_INTERVAL_SEC  300UL
// Розмір буфера FFT
#define FFT_N         512
```

```
#define SAMPLE_RATE   3000   // Гц
```

Загальна структура програмного забезпечення мікроконтролера побудована за принципом кінцевого автомата із циклічним виконанням. Основний цикл програми послідовно виконує такі стани: ініціалізація периферії при першому запуску, збір даних із датчиків, первинна обробка та аналіз, формування пакету даних, передача до ESP8266, запис на microSD та перехід у режим очікування до наступного циклу вимірювань. Такий підхід забезпечує детерміновану поведінку системи та спрощує налагодження програмного забезпечення.

Ініціалізація периферійних модулів мікроконтролера виконується у функції system_init, що викликається одноразово при запуску. У середовищі CodeVisionAVR більшість ініціалізаційного коду генерується автоматично CodeWizardAVR, однак для модулів, специфічних для даної системи, необхідне ручне доопрацювання. Нижче наведено фрагмент ініціалізації основних модулів системи:

```
void system_init(void)
{
   // Налаштування тактової частоти 32 МГц
   // (внутрішній RC-генератор ATXMEGA)
   OSC.CTRL |= OSC_RC32MEN_bm;
   while (!(OSC.STATUS & OSC_RC32MRDY_bm));
   CCP = CCP_IOREG_gc;
   CLK.CTRL = CLK_SCLKSEL_RC32M_gc;

   // Налаштування SPI на PORTC (Master, MODE0, 8 МГц)
   PORTC.DIR |= PIN4_bm | PIN5_bm | PIN7_bm; // SS, MOSI, SCK — вихід
   PORTC.DIR &= ~PIN6_bm;                    // MISO — вхід
   PORTC.OUT  |= PIN4_bm;                    // SS неактивний (HIGH)
   SPIC.CTRL = SPI_ENABLE_bm  |
               SPI_MASTER_bm  |
               SPI_MODE_0_gc  |
               SPI_PRESCALER_DIV4_gc;

   // Налаштування TWI (I2C) на PORTD, 400 кГц
   PORTD.DIR |= PIN0_bm | PIN1_bm;           // SDA, SCL — вихід
   TWID.MASTER.BAUD = (unsigned char)
      ((F_CPU / (2UL * 400000UL)) - 5);
   TWID.MASTER.CTRLA = TWI_MASTER_ENABLE_bm;

   // Налаштування АЦП ADCA (12-біт, внутрішня опора 1.0 В)
```

```
    ADCA.CTRLB   = ADC_RESOLUTION_12BIT_gc;
    ADCA.REFCTRL = ADC_REFSEL_INT1V_gc | ADC_BANDGAP_bm;
    ADCA.PRESCALER = ADC_PRESCALER_DIV32_gc;
    ADCA.CTRLA   = ADC_ENABLE_bm;

    // Налаштування таймера TCC0 для відліку інтервалу вимірювань
    // При F=32МГц, DIV1024: переповнення кожні ~1 с
    TCC0.PER     = 31250 - 1;
    TCC0.CTRLA   = TC_CLKSEL_DIV1024_gc;
    TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

    // Дозвіл усіх рівнів переривань
    PMIC.CTRL = PMIC_LOLVLEN_bm |
                PMIC_MEDLVLEN_bm |
                PMIC_HILVLEN_bm;
    sei();
}
```

Зчитування даних із сенсора SCD30 здійснюється через шину I²C за протоколом, визначеним у технічній документації Sensirion. Сенсор підтримує команди запуску безперервних вимірювань та зчитування готових даних. Перед зчитуванням необхідно перевірити готовність нових даних через регістр статусу. Нижче наведено функції роботи з SCD30 у середовищі CodeVisionAVR з використанням вбудованої бібліотеки i2c:

```
// Обчислення CRC-8 для верифікації даних SCD30
unsigned char scd30_crc8(unsigned char *data, unsigned char len)
{
    unsigned char crc = 0xFF;
    unsigned char i, j;
    for (i = 0; i < len; i++) {
        crc ^= data[i];
        for (j = 0; j < 8; j++)
            crc = (crc & 0x80) ? (crc << 1) ^ 0x31 : (crc << 1);
    }
    return crc;
}

// Запис команди (2 байти) до SCD30
void scd30_write_cmd(unsigned int cmd)
{
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x00);
    i2c_write((unsigned char)(cmd >> 8));
    i2c_write((unsigned char)(cmd & 0xFF));
    i2c_stop();
}

// Запуск безперервних вимірювань
// pressure_mbar — атмосферний тиск для компенсації
```

```
void scd30_start_continuous(unsigned int pressure_mbar)
{
   unsigned char arg[2];
   arg[0] = (unsigned char)(pressure_mbar >> 8);
   arg[1] = (unsigned char)(pressure_mbar & 0xFF);

   i2c_start();
   i2c_write((SCD30_ADDR << 1) | 0x00);
   i2c_write(0x00); i2c_write(0x10);  // Команда 0x0010
   i2c_write(arg[0]);
   i2c_write(arg[1]);
   i2c_write(scd30_crc8(arg, 2));
   i2c_stop();
}

// Перевірка готовності даних (повертає 1 якщо дані готові)
unsigned char scd30_data_ready(void)
{
   unsigned char buf[3];
   scd30_write_cmd(0x0202);
   i2c_start();
   i2c_write((SCD30_ADDR << 1) | 0x01);
   buf[0] = i2c_read(1);
   buf[1] = i2c_read(1);
   buf[2] = i2c_read(0);   // NACK — останній байт
   i2c_stop();
   return buf[1];
}

// Зчитування CO2, температури та вологості
void scd30_read_measurement(float *co2,
                            float *temp,
                            float *hum)
{
   unsigned char raw[18];
   unsigned char i;
   unsigned long tmp;

   scd30_write_cmd(0x0300);
   i2c_start();
   i2c_write((SCD30_ADDR << 1) | 0x01);
   for (i = 0; i < 17; i++) raw[i] = i2c_read(1);
   raw[17] = i2c_read(0);
   i2c_stop();

   // CO2 — байти 0,1 та 3,4 (байт 2 — CRC)
   tmp = ((unsigned long)raw[0]  << 24) |
         ((unsigned long)raw[1]  << 16) |
         ((unsigned long)raw[3]  <<  8) |
          (unsigned long)raw[4];
   memcpy(co2, &tmp, 4);

   // Температура — байти 6,7 та 9,10
   tmp = ((unsigned long)raw[6]  << 24) |
         ((unsigned long)raw[7]  << 16) |
```

```
        ((unsigned long)raw[9]  <<  8) |
         (unsigned long)raw[10];
    memcpy(temp, &tmp, 4);

    // Вологість — байти 12,13 та 15,16
    tmp = ((unsigned long)raw[12] << 24) |
        ((unsigned long)raw[13] << 16) |
        ((unsigned long)raw[15] <<  8) |
         (unsigned long)raw[16];
    memcpy(hum, &tmp, 4);
}
```

Зчитування маси вулика здійснюється через АЦП HX711, підключений до виводів PB0 (CLK) та PB1 (DATA) мікроконтролера. Для отримання стабільного значення виконується осереднення восьми послідовних відліків із відкиданням мінімального та максимального значень:

```
// Зчитування одного відліку з HX711 (канал A, підсилення 128)
long hx711_read(void)
{
    long data = 0;
    unsigned char i;

    // Очікування готовності: DOUT = LOW
    while (PORTB.IN & PIN1_bm);

    // Зчитування 24 біт даних
    for (i = 0; i < 24; i++) {
        PORTB.OUT |=  PIN0_bm;   // CLK HIGH
        delay_us(1);
        data = (data << 1) |
              ((PORTB.IN & PIN1_bm) ? 1L : 0L);
        PORTB.OUT &= ~PIN0_bm;   // CLK LOW
        delay_us(1);
    }

    // 25-й імпульс — вибір каналу A, підсилення 128
    PORTB.OUT |=  PIN0_bm;
    delay_us(1);
    PORTB.OUT &= ~PIN0_bm;
    delay_us(1);

    // Знакове розширення 24 → 32 біт
    if (data & 0x800000L) data |= 0xFF000000L;
    return data;
}

// Осереднення 8 відліків із відкиданням крайніх
long hx711_read_average(void)
{
    long readings[8];
    long sum = 0, min_v, max_v;
    unsigned char i;
```

```
    for (i = 0; i < 8; i++) readings[i] = hx711_read();

    min_v = max_v = readings[0];
    for (i = 0; i < 8; i++) {
        sum += readings[i];
        if (readings[i] < min_v) min_v = readings[i];
        if (readings[i] > max_v) max_v = readings[i];
    }
    return (sum - min_v - max_v) / 6;
}

// Перетворення сирих відліків у кілограми
float hx711_get_kg(long raw, long tare, float scale)
{
    return (float)(raw - tare) / scale / 1000.0f;
}
```

Збір акустичних даних реалізується через вбудований АЦП мікроконтролера з використанням модуля DMA для накопичення вибірки без навантаження на процесорне ядро. Частота дискретизації 3 кГц задається таймером TCD0, переповнення якого запускає чергове перетворення АЦП. Після накопичення 512 відліків DMA генерує переривання завершення, що встановлює прапор готовності буфера:

```
// Буфер акустичних відліків (заповнюється через DMA)
volatile unsigned int  adc_buffer[FFT_N];
volatile unsigned char adc_buffer_ready = 0;

void adc_dma_init(void)
{
    // Таймер TCD0: частота дискретизації 3 кГц
    // F_CPU=32МГц, DIV8: 32000000/8/3000 = 1333
    TCD0.PER   = 1333 - 1;
    TCD0.CTRLA = TC_CLKSEL_DIV8_gc;

    // Канал АЦП PA0 — вхід мікрофона MAX9814
    ADCA.CH0.CTRL    = ADC_CH_INPUTMODE_SINGLEENDED_gc;
    ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;

    // DMA канал 0: ADCA.CH0.RES → adc_buffer
    DMA.CH0.SRCADDR0  = (unsigned char)
        ((unsigned int)(&ADCA.CH0.RES) >> 0);
    DMA.CH0.SRCADDR1  = (unsigned char)
        ((unsigned int)(&ADCA.CH0.RES) >> 8);
    DMA.CH0.DESTADDR0 = (unsigned char)
        ((unsigned int)(adc_buffer) >> 0);
    DMA.CH0.DESTADDR1 = (unsigned char)
        ((unsigned int)(adc_buffer) >> 8);
    DMA.CH0.TRFCNT    = FFT_N * 2;   // 512 відліків × 2 байти
    DMA.CH0.ADDRCTRL  = DMA_CH_SRCRELOAD_BURST_gc |
```

```
                DMA_CH_DESTINC_bm          |
                DMA_CH_DESTRELOAD_TRANSACTION_gc;
    DMA.CH0.TRIGSRC   = DMA_CH_TRIGSRC_ADCA_CH0_gc;
    DMA.CH0.INTCTRL   = DMA_CH_TRNINTLVL_MED_gc;
    DMA.CH0.CTRLA     = DMA_CH_ENABLE_bm      |
                DMA_CH_SINGLE_bm       |
                DMA_CH_BURSTLEN_2BYTE_gc;
    DMA.CTRL          = DMA_ENABLE_bm;
}

// Переривання DMA — буфер заповнено
interrupt [DMA_CH0_vect] void dma_ch0_isr(void)
{
    adc_buffer_ready = 1;
    DMA.CH0.CTRLB |= DMA_CH_TRNIF_bm;  // Скидання прапора
}
```

Первинна обробка даних включає перевірку значень на відповідність фізично можливим діапазонам та фільтрацію одиночних викидів методом експоненційного згладжування. Коефіцієнт згладжування α = 0.2 обраний як компроміс між швидкістю реакції на реальні зміни та придушенням випадкових шумів вимірювання:

```
#define EMA_ALPHA  0.2f

// Структура поточних вимірювань
typedef struct {
    float       temperature;
    float       humidity;
    float       co2;
    float       pressure;
    float       weight_kg;
    unsigned int flight_cnt;
    unsigned int fft_dom_hz;
    unsigned char colony_state;
} meas_data_t;

// Експоненційне згладжування
float ema_filter(float new_val, float prev_val)
{
    return EMA_ALPHA * new_val + (1.0f - EMA_ALPHA) * prev_val;
}

// Перевірка та фільтрація всіх параметрів
void validate_and_filter(meas_data_t *m,
                         meas_data_t *prev)
{
    // Температура: фізичний діапазон -20...+60 °C
    if (m->temperature >= -20.0f && m->temperature <= 60.0f)
        m->temperature = ema_filter(m->temperature,
                                    prev->temperature);
```

```
    else
        m->temperature = prev->temperature;

    // Вологість: 0...100 %
    if (m->humidity >= 0.0f && m->humidity <= 100.0f)
        m->humidity = ema_filter(m->humidity, prev->humidity);
    else
        m->humidity = prev->humidity;

    // CO2: 400...10000 ppm
    if (m->co2 >= 400.0f && m->co2 <= 10000.0f)
        m->co2 = ema_filter(m->co2, prev->co2);
    else
        m->co2 = prev->co2;

    // Тиск: 850...1100 hPa
    if (m->pressure >= 850.0f && m->pressure <= 1100.0f)
        m->pressure = ema_filter(m->pressure, prev->pressure);
    else
        m->pressure = prev->pressure;

    // Вага: 5...80 кг (фізичні межі вулика з бджолами)
    if (m->weight_kg >= 5.0f && m->weight_kg <= 80.0f)
        m->weight_kg = ema_filter(m->weight_kg, prev->weight_kg);
    else
        m->weight_kg = prev->weight_kg;
}
```

Основний цикл програми координує послідовне виконання всіх задач збору та обробки даних. Після завершення кожного циклу мікроконтролер переходить у режим зниженого енергоспоживання до настання наступного інтервалу вимірювань:

```
meas_data_t current_meas;
meas_data_t prev_meas;
long        tare_value  = 0L;
float       scale_factor = 420.0f;  // Підбирається при калібруванні

void main(void)
{
    system_init();
    adc_dma_init();
    sd_init();
    esp8266_init();

    // Початкова компенсація тиску для SCD30
    prev_meas.pressure = bme280_read_pressure();
    scd30_start_continuous((unsigned int)prev_meas.pressure);
    delay_ms(2000);   // Прогрів SCD30

    while (1) {
        // 1. Датчики мікроклімату SCD30
        if (scd30_data_ready())
```

```
            scd30_read_measurement(&current_meas.co2,
                                   &current_meas.temperature,
                                   &current_meas.humidity);

        // 2. Атмосферний тиск BME280
        current_meas.pressure = bme280_read_pressure();

        // 3. Маса вулика HX711
        current_meas.weight_kg = hx711_get_kg(
            hx711_read_average(), tare_value, scale_factor);

        // 4. Льотна активність VL6180X
        current_meas.flight_cnt = vl6180x_get_count();

        // 5. Акустичний FFT-аналіз
        adc_dma_init();
        while (!adc_buffer_ready);
        adc_buffer_ready = 0;
        fft_analyze(adc_buffer, &current_meas.fft_dom_hz);

        // 6. Первинна фільтрація
        validate_and_filter(&current_meas, &prev_meas);

        // 7. Діагностика стану колонії
        current_meas.colony_state =
            diagnose_colony(&current_meas);

        // 8. Запис на microSD
        sd_write_record(&current_meas);

        // 9. Передача до ESP8266 через SPI
        esp8266_send_data(&current_meas);

        // 10. Збереження поточних даних як попередніх
        memcpy(&prev_meas, &current_meas,
               sizeof(meas_data_t));

        // 11. Режим очікування до наступного циклу
        sleep_until_next_cycle(MEAS_INTERVAL_SEC);
    }
}
```

Таким чином, програмне забезпечення мікроконтролера ATXMEGA256A3U, розроблене у середовищі CodeVisionAVR, реалізує повний цикл збору та первинної обробки даних із усіх підключених датчиків у межах єдиного детермінованого циклу виконання. Використання DMA для збору акустичних відліків, осереднення показань ваги та експоненційного фільтра для згладжування параметрів мікроклімату забезпечує достатню якість даних для подальшого аналізу стану колонії.

3.2 Алгоритми аналізу стану бджолиної колонії та виявлення аномалій

Алгоритми аналізу стану бджолиної колонії є ключовою інтелектуальною складовою інформаційної системи, що перетворює сирі числові дані вимірювань на змістовні діагностичні висновки, зрозумілі пасічнику. Завдання аналізу полягає у виявленні характерних патернів у сукупності контрольованих параметрів, що відповідають різним фізіологічним станам колонії – нормальній активності, ройовому настрою, відсутності матки або ослабленню. Усі алгоритми аналізу реалізовані безпосередньо на мікроконтролері ATXMEGA256A3U у середовищі CodeVisionAVR без залежності від зовнішніх обчислювальних ресурсів [22].

Загальний алгоритм аналізу стану колонії побудований як послідовний ланцюжок перевірок із пріоритизацією за критичністю стану (рис. 3.1). На першому етапі виконується перевірка базових параметрів мікроклімату – температури, вологості та концентрації CO_2 – відносно встановлених порогових значень. При виявленні відхилення будь-якого параметра алгоритм негайно формує відповідний діагностичний код і завершує аналіз, не виконуючи подальших перевірок. Якщо всі базові параметри знаходяться в нормі, виконується FFT-аналіз акустичного сигналу та комплексна діагностика стану колонії за результатами спектрального аналізу у поєднанні з динамікою маси та льотною активністю. Завершується алгоритм формуванням текстового повідомлення та передачею пакету даних до модуля ESP8266.

Перший етап алгоритму реалізується через структуру порогових значень, що зберігається у Flash-пам'яті мікроконтролера та може бути змінена користувачем через web-інтерфейс.

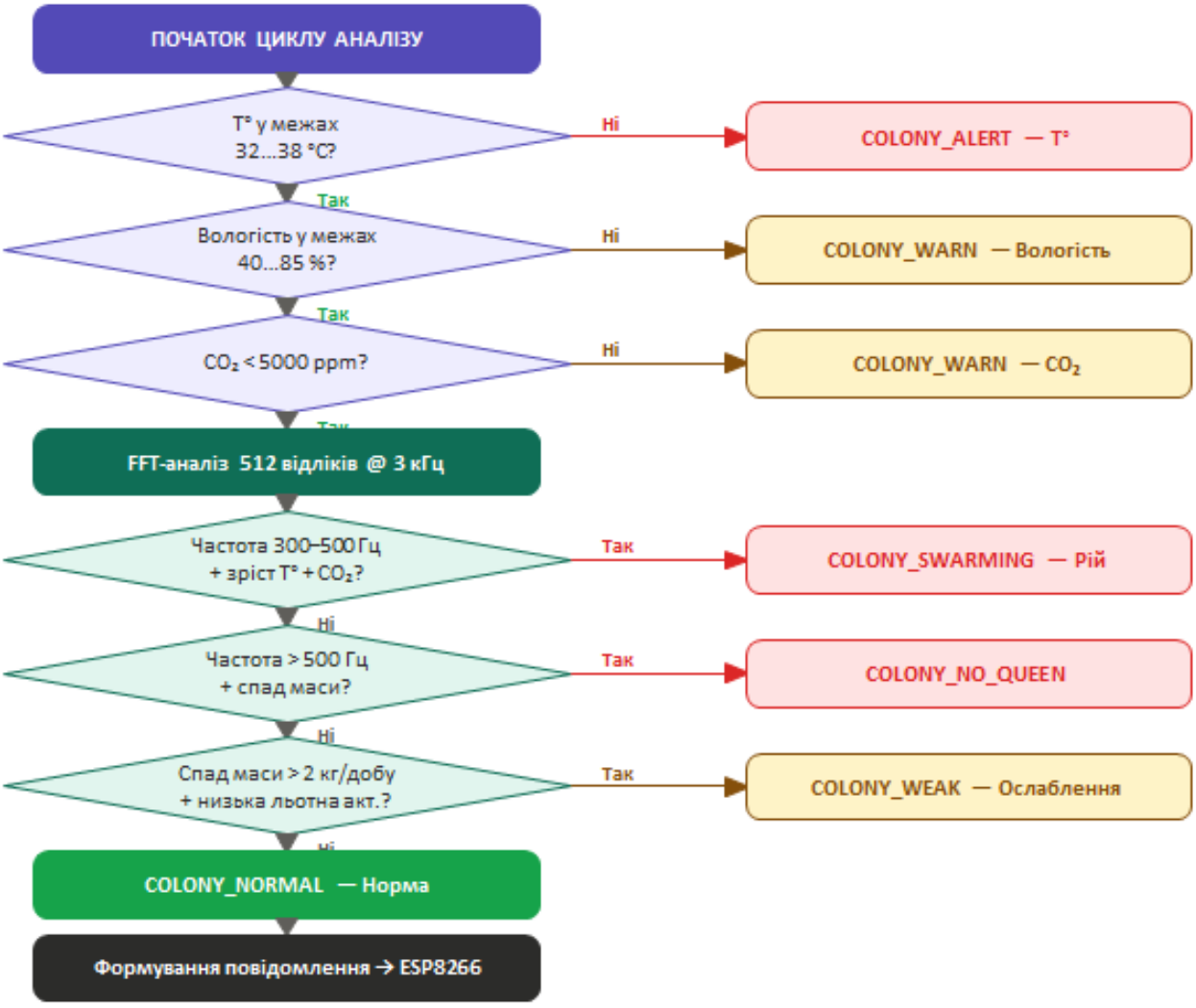

Рисунок 3.1 - Загальний алгоритм аналізу стану колонії

Для кожного параметра визначено граничне значення, при перевищенні якого формується відповідний діагностичний код:

```
#include <atxmega256a3u.h>
#include <string.h>

// Коди діагностичних станів колонії
#define COLONY_NORMAL     0
#define COLONY_WARN_TEMP  1
#define COLONY_WARN_HUM   2
#define COLONY_WARN_CO2   3
#define COLONY_SWARMING   4
#define COLONY_NO_QUEEN   5
#define COLONY_WEAK       6
#define COLONY_ALERT      7

// Структура порогових значень
typedef struct {
```

```
    float temp_min;
    float temp_max;
    float hum_min;
    float hum_max;
    float co2_max;
    float weight_drop_day;
    unsigned int flight_min;
} thresholds_t;

// Значення за замовчуванням
thresholds_t thresholds = {
    32.0f,   // temp_min, °C
    38.0f,   // temp_max, °C
    40.0f,   // hum_min, %
    85.0f,   // hum_max, %
    5000.0f, // co2_max, ppm
    2.0f,    // weight_drop_day, кг
    10       // flight_min, бдж/хв
};
```

Перевірка базових параметрів мікроклімату виконується у строгій послідовності, визначеній алгоритмом на рис. 3.1. Спочатку перевіряється температура як найбільш критичний параметр для виживання розплоду, потім вологість та концентрація CO_2:

```
// Перевірка базових параметрів мікроклімату
// Повертає код стану або COLONY_NORMAL якщо всі параметри в нормі
unsigned char check_microclimate(meas_data_t *m)
{
    // Крок 1: перевірка температури
    if (m->temperature < thresholds.temp_min ||
        m->temperature > thresholds.temp_max)
    {
        return COLONY_ALERT;  // Критичне відхилення T°
    }

    // Крок 2: перевірка вологості
    if (m->humidity < thresholds.hum_min ||
        m->humidity > thresholds.hum_max)
    {
        return COLONY_WARN_HUM;
    }

    // Крок 3: перевірка CO₂
    if (m->co2 > thresholds.co2_max)
    {
        return COLONY_WARN_CO2;
    }

    return COLONY_NORMAL;  // Всі параметри в нормі → перехід до FFT
}
```

Другий етап — FFT-аналіз акустичного сигналу — виконується лише після успішного проходження перевірки мікроклімату. Алгоритм Кулі–Тьюкі з основою 2 застосовується до 512 відліків, накопичених через DMA з частотою дискретизації 3 кГц. Після обчислення спектру виділяється домінантна частота у діапазоні 150–1200 Гц:

```
#define FFT_N       512
#define SAMPLE_RATE  3000
int fft_re[FFT_N];
int fft_im[FFT_N];
// Підготовка вхідних даних із вікном Хеннінга
void fft_apply_window(volatile unsigned int *samples,
                      int *re, int *im)
{
   unsigned int i;
   for (i = 0; i < FFT_N; i++) {
      long w = 32768L - hann_table[i];
      re[i] = (int)(((long)((int)samples[i] - 2048) * w) >> 15);
      im[i] = 0;
   }}
// Виділення домінантної частоти у діапазоні 150–1200 Гц
void fft_analyze(volatile unsigned int *samples,
                 unsigned int *dom_freq_hz)
{
   unsigned int i, max_bin = 0;
   long max_power = 0, power;
   fft_apply_window(samples, fft_re, fft_im);
   fft_radix2(fft_re, fft_im, FFT_N);  // Вбудована функція CVA
   // Межі діапазону аналізу в бінах
   unsigned int bin_lo =
      (unsigned int)(150UL * FFT_N / SAMPLE_RATE);
   unsigned int bin_hi =
      (unsigned int)(1200UL * FFT_N / SAMPLE_RATE);
   for (i = bin_lo; i <= bin_hi && i < FFT_N / 2; i++) {
      power = (long)fft_re[i] * fft_re[i]
            + (long)fft_im[i] * fft_im[i];
      if (power > max_power) {
         max_power = power;
         max_bin   = i;
      }
   }
   *dom_freq_hz =
      (unsigned int)((unsigned long)max_bin * SAMPLE_RATE / FFT_N);
}
```

Третій етап — комплексна діагностика — реалізує послідовну перевірку трьох патологічних станів у порядку спадання критичності, як показано на рис. 3.1. Першим перевіряється ройовий настрій як стан, що потребує

найшвидшого реагування. Ознакою ройового настрою є домінування частот 300–500 Гц у поєднанні зі зростанням температури та CO_2. Другим перевіряється відсутність матки – домінування частот вище 500 Гц у поєднанні зі спадом маси вулика. Третім – ослаблення колонії, що діагностується за стійким спадом маси понад 2 кг на добу у поєднанні з низькою льотною активністю:

```
// Відстеження зміни маси між циклами (добова динаміка)
// Зберігаємо масу 288 вимірювань тому (24 год × 12 вим/год)
#define DAY_BUF  288
float weight_day_buf[DAY_BUF];
unsigned int weight_day_idx = 0;
unsigned int weight_day_cnt = 0;
void weight_day_push(float w)
{
   weight_day_buf[weight_day_idx] = w;
   weight_day_idx = (weight_day_idx + 1) % DAY_BUF;
   if (weight_day_cnt < DAY_BUF) weight_day_cnt++;
}
// Спад маси за добу (кг)
float weight_drop_per_day(float current_weight)
{
   if (weight_day_cnt < DAY_BUF) return 0.0f;
   float day_ago = weight_day_buf[weight_day_idx];
   float drop = day_ago - current_weight;
   return drop > 0.0f ? drop : 0.0f;
}
// Комплексна діагностика за результатами FFT
unsigned char check_colony_state(meas_data_t *m,
                        float weight_drop)
{
   // Крок 4: ройовий настрій
   // Частота 300–500 Гц + висока T° + підвищений CO₂
   if (m->fft_dom_hz >= 300       &&
      m->fft_dom_hz <= 500       &&
      m->temperature > 35.5f     &&
      m->co2 > 2000.0f)
   {
      return COLONY_SWARMING;
   }
   // Крок 5: відсутність матки
   // Частота > 500 Гц + спад маси
   if (m->fft_dom_hz > 500        &&
      weight_drop > 0.5f         &&
      m->flight_cnt < thresholds.flight_min)
   {       return COLONY_NO_QUEEN;   }
   // Крок 6: ослаблення колонії
   // Стійкий спад маси > 2 кг/добу + низька льотна активність
   if (weight_drop > thresholds.weight_drop_day &&
      m->flight_cnt < thresholds.flight_min)
```

```
    {       return COLONY_WEAK;   }
   return COLONY_NORMAL;}
```

Головна функція діагностики об'єднує обидва етапи аналізу відповідно до структури алгоритму на рис. 3.1 та формує текстове повідомлення для передачі до ESP8266:

```
char diag_msg[96];
unsigned char diagnose_colony(meas_data_t *m)
{
   unsigned char state;
   float drop;
   // Етап 1: перевірка мікроклімату
   state = check_microclimate(m);
   if (state != COLONY_NORMAL) {
      colony_state_to_str(state, diag_msg);
      return state;
   }
   // Етап 2: FFT-аналіз (запускається лише при нормі мікроклімату)
   adc_dma_init();
   while (!adc_buffer_ready);
   adc_buffer_ready = 0;
   fft_analyze(adc_buffer, &m->fft_dom_hz);
   // Оновлення буфера добової динаміки маси
   weight_day_push(m->weight_kg);
   drop = weight_drop_per_day(m->weight_kg);
   // Етап 3: комплексна діагностика
   state = check_colony_state(m, drop);
   colony_state_to_str(state, diag_msg);
   return state;
}
// Формування текстового повідомлення
void colony_state_to_str(unsigned char state, char *msg)
{
   switch (state) {
      case COLONY_NORMAL:
         strcpy(msg, "НОРМА:Колонія у нормальному стані");
         break;
      case COLONY_ALERT:
         strcpy(msg, "ТРИВОГА:Критичне відхилення T");
         break;
      case COLONY_WARN_HUM:
         strcpy(msg, "УВАГА:Вологість за межами норми");
         break;
      case COLONY_WARN_CO2:
         strcpy(msg, "УВАГА:CO2 перевищує норму.Перевірте вентиляцію");
         break;
      case COLONY_SWARMING:
         strcpy(msg, "РІЙ:Ознакиройового настрою.Підготуйте корпус");
         break;
      case COLONY_NO_QUEEN:
         strcpy(msg, "БЕЗ МАТКИ:Перевірте наявність матки у вулику");
```

```
            break;
        case COLONY_WEAK:
            strcpy(msg, "ОСЛАБЛЕННЯ:Колонія втрачає масу.Огляньте вулик");
            break;
        default:
            strcpy(msg, "НЕВІДОМО:Недостатньо даних");
    }}
```

Таким чином, реалізований алгоритм аналізу стану бджолиної колонії послідовно виконує перевірку параметрів мікроклімату, FFT-аналіз акустичного сигналу та комплексну діагностику за динамікою маси і льотною активністю відповідно до структури, наведеної на рис. 3.1. Пріоритизація перевірок від найбільш критичних до менш критичних забезпечує своєчасне виявлення небезпечних станів колонії та мінімізує обчислювальне навантаження - FFT-аналіз виконується лише за умови нормального мікроклімату. Сформований діагностичний код та текстове повідомлення передаються до модуля ESP8266 у складі пакету даних для відображення у web-інтерфейсі.

3.3. Реалізація вбудованого web-сервера на базі ESP8266

Вбудований web-сервер на модулі ESP8266 є програмним ядром інформаційної системи з боку мережевої взаємодії — саме він забезпечує доступ пасічника до даних моніторингу через стандартний web-браузер без встановлення додаткового програмного забезпечення. Програмне забезпечення ESP8266 розробляється з використанням ESP8266 RTOS SDK на основі операційної системи реального часу FreeRTOS, що дозволяє організувати паралельне виконання задач обслуговування Wi-Fi з'єднань, прийому даних від ATXMEGA через SPI та обробки HTTP-запитів клієнтів [23, 24].

Програмне забезпечення ESP8266 організоване у вигляді трьох паралельних задач FreeRTOS із різними пріоритетами. Задача прийому даних від ATXMEGA через SPI має найвищий пріоритет та забезпечує своєчасне

оновлення буфера поточних даних. Задача обслуговування HTTP-запитів має середній пріоритет та виконується у циклі очікування вхідних з'єднань. Задача управління Wi-Fi з'єднанням має найнижчий пріоритет та відповідає за підтримку точки доступу та моніторинг стану мережевого стеку.

Ініціалізація ESP8266 включає налаштування Wi-Fi у режимі точки доступу, ініціалізацію SPI у режимі Slave та запуск HTTP-сервера на порту 80:

```
#include "esp_common.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "lwip/sockets.h"
#include "espressif/esp_sta.h"
#include "espressif/esp_softap.h"
// Параметри точки доступу
#define AP_SSID      "BeeMonitor_01"
#define AP_PASSWORD  "beehive2024"
#define AP_CHANNEL   6
#define HTTP_PORT    80
// Розмір буфера поточних даних JSON
#define JSON_BUF_SIZE  512
// Глобальний буфер поточних даних (оновлюється із SPI)
char json_current[JSON_BUF_SIZE];
volatile uint8_t json_ready = 0;
void wifi_init(void)
{
   // Режим точки доступу
   wifi_set_opmode(SOFTAP_MODE);
   struct softap_config ap_cfg;
   memset(&ap_cfg, 0, sizeof(ap_cfg));
   strcpy((char*)ap_cfg.ssid,     AP_SSID);
   strcpy((char*)ap_cfg.password, AP_PASSWORD);
   ap_cfg.ssid_len    = strlen(AP_SSID);
   ap_cfg.channel     = AP_CHANNEL;
   ap_cfg.authmode    = AUTH_WPA2_PSK;
   ap_cfg.max_connection = 4;
   wifi_softap_set_config(&ap_cfg);
}
```

Прийом даних від ATXMEGA через SPI реалізовано у окремій задачі FreeRTOS. ATXMEGA ініціює передачу шляхом виставлення низького рівня на лінії SS, після чого передає пакет фіксованої структури розміром 64 байти, що містить усі вимірювані параметри та діагностичний код. ESP8266 працює у режимі SPI Slave та очікує на переривання від апаратного SPI-контролера:

```
// Структура пакету даних від ATXMEGA (64 байти)
typedef struct __attribute__((packed)) {
   uint8_t  header[2];     // 0xBE 0xE1 — маркер пакету
   float    temperature;   // °C
```

```
    float    humidity;      // %
    float    co2;           // ppm
    float    pressure;      // hPa
    float    weight_kg;     // кг
    uint16_t flight_cnt;    // бдж/хв
    uint16_t fft_dom_hz;    // Гц
    uint8_t  colony_state;  // код стану
    char     diag_msg[33];  // текстове повідомлення
    uint32_t timestamp;     // UNIX timestamp
    uint8_t  crc8;          // контрольна сума
} spi_packet_t;
spi_packet_t rx_packet;
// Задача прийому даних через SPI
void spi_rx_task(void *pvParameters)
{
    uint8_t rx_buf[sizeof(spi_packet_t)];
    uint8_t crc;
    // Налаштування SPI Slave (HSPI)
    spi_slave_init(HSPI);
    while (1) {
        // Очікування завершення прийому пакету
        spi_slave_recv(HSPI, rx_buf, sizeof(spi_packet_t));
        // Перевірка маркера та CRC
        if (rx_buf[0] == 0xBE && rx_buf[1] == 0xE1) {
            crc = crc8_calc(rx_buf, sizeof(spi_packet_t) - 1);
            if (crc == rx_buf[sizeof(spi_packet_t) - 1]) {
                memcpy(&rx_packet, rx_buf, sizeof(spi_packet_t));
                // Формування JSON із отриманих даних
                build_json_current(&rx_packet, json_current);
                json_ready = 1;
            }
     }
        vTaskDelay(10 / portTICK_RATE_MS);
    }}
```

Формування JSON-відповіді для endpoint «/api/current» виконується функцією build_json_current безпосередньо після отримання пакету від ATXMEGA. Для мінімізації використання динамічної пам'яті JSON формується у статичний буфер за допомогою функції snprintf:

```
void build_json_current(spi_packet_t *p, char *buf)
{
    snprintf(buf, JSON_BUF_SIZE,
        "{"
        "\"temp\":%.1f,"
        "\"hum\":%.1f,"
        "\"co2\":%.0f,"
        "\"pres\":%.1f,"
        "\"weight\":%.2f,"
        "\"flight\":%u,"
        "\"fft_hz\":%u,"
        "\"state\":%u,"
        "\"msg\":\"%s\","
```

```
        "\"ts\":%lu"
        "}",
        p->temperature,
        p->humidity,
        p->co2,
        p->pressure,
        p->weight_kg,
        p->flight_cnt,
        p->fft_dom_hz,
        p->colony_state,
        p->diag_msg,
        p->timestamp
    );
}
```

Ядром web-сервера є задача обробки HTTP-запитів, що реалізована на базі Berkeley Sockets API бібліотеки lwIP, вбудованої у ESP8266 RTOS SDK. Сервер прослуховує порт 80, приймає вхідні з'єднання та передає їх на обробку функції handle_request:

```
// Задача HTTP-сервера
void http_server_task(void *pvParameters)
{
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len = sizeof(client_addr);
    char rx_buf[256];
    int  rx_len;
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family      = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port        = htons(HTTP_PORT);
    bind(server_fd,
         (struct sockaddr*)&server_addr,
         sizeof(server_addr));
    listen(server_fd, 4);
    while (1) {
        client_fd = accept(server_fd,
                           (struct sockaddr*)&client_addr,
                           &client_len);
        if (client_fd < 0) continue;
        // Читання HTTP-запиту
        rx_len = recv(client_fd, rx_buf, sizeof(rx_buf)-1, 0);
        if (rx_len > 0) {
            rx_buf[rx_len] = '\0';
            handle_request(client_fd, rx_buf);   }
        close(client_fd);
        vTaskDelay(5 / portTICK_RATE_MS);
    }}
```

Маршрутизація запитів реалізована у функції handle_request через порівняння URI рядком strstr. Залежно від URI викликається відповідний обробник – відправлення стисненої HTML-сторінки з Flash-пам'яті, JSON поточних даних із кешу або JSON архівних даних, запитаних у ATXMEGA:

```
/ Зовнішні масиви з Flash-пам'яті (HTML+CSS+JS стиснуті gzip)
extern const uint8_t index_html_gz[];
extern const uint32_t index_html_gz_len;

void handle_request(int fd, char *req)
{
   // GET / — головна сторінка
   if (strstr(req, "GET / ") ||
      strstr(req, "GET /index.html"))
   {
      send_gzip_page(fd,
                     index_html_gz,
                     index_html_gz_len);
      return;
   }
   // GET /api/current — поточні дані
   if (strstr(req, "GET /api/current")) {
      send_json(fd, json_current);
      return;
   }
   // GET /api/status — стан системи
   if (strstr(req, "GET /api/status")) {
      char status_json[128];
      build_json_status(status_json);
      send_json(fd, status_json);
      return;
   }
   // POST /api/settings — збереження налаштувань
   if (strstr(req, "POST /api/settings")) {
      handle_settings(fd, req);
      return;
   }
   // 404 для невідомих URI
   send_response(fd, 404,
                 "text/plain",
                 "Not Found", 9);
}
```

Відправлення HTTP-відповідей реалізовано через уніфіковані функції send_json та send_gzip_page, що формують коректні HTTP-заголовки з кодами статусу, типом вмісту та заголовками кешування:

```
void send_json(int fd, char *json)
{
   char header[128];
   int  json_len = strlen(json);
```

```
    snprintf(header, sizeof(header),
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: application/json\r\n"
        "Content-Length: %d\r\n"
        "Access-Control-Allow-Origin: *\r\n"
        "Connection: close\r\n"
        "\r\n",
        json_len);
    send(fd, header, strlen(header), 0);
    send(fd, json,   json_len,       0);
}
void send_gzip_page(int fd,
                    const uint8_t *data,
                    uint32_t len)
{
    char header[192];
    snprintf(header, sizeof(header),
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: text/html\r\n"
        "Content-Encoding: gzip\r\n"
        "Content-Length: %lu\r\n"
        "Cache-Control: max-age=3600\r\n"
        "Connection: close\r\n"
        "\r\n",
        (unsigned long)len);
    send(fd, header, strlen(header), 0);
    send(fd, data,   len,            0);
}
```

Формування JSON статусу системи включає службову інформацію про час роботи пристрою, кількість записів на microSD, рівень сигналу Wi-Fi та напругу живлення акумулятора:

```
void build_json_status(char *buf)
{
    uint32_t uptime_sec = xTaskGetTickCount() /
                          portTICK_RATE_MS / 1000;
    int8_t   rssi = wifi_station_get_rssi();
    snprintf(buf, 128,
        "{"
        "\"uptime\":%lu,"
        "\"rssi\":%d,"
        "\"heap\":%u,"
        "\"records\":%lu"
        "}",
        uptime_sec,
        rssi,
        system_get_free_heap_size(),
        sd_get_record_count()
    );}
```

Обробка POST-запиту налаштувань передбачає розбір тіла запиту у форматі JSON та передачу нових порогових значень до ATXMEGA через SPI у вигляді командного пакету:

```
void handle_settings(int fd, char *req)
{
   // Пошук тіла запиту після подвійного CRLF
   char *body = strstr(req, "\r\n\r\n");
   if (!body) { send_json(fd, "{\"ok\":false}"); return; }
   body += 4;
   // Розбір JSON (мінімальний парсер для числових полів)
   float t_min, t_max, h_min, h_max, co2_max;
   if (sscanf(body,
      "{\"t_min\":%f,\"t_max\":%f,"
       "\"h_min\":%f,\"h_max\":%f,"
       "\"co2\":%f}",
      &t_min, &t_max, &h_min, &h_max, &co2_max) == 5)
   {
      thresholds.temp_min = t_min;
      thresholds.temp_max = t_max;
      thresholds.hum_min  = h_min;
      thresholds.hum_max  = h_max;
      thresholds.co2_max  = co2_max;
      // Передача нових налаштувань до ATXMEGA через SPI
      send_settings_to_atxmega(&thresholds);
      send_json(fd, "{\"ok\":true}");
   } else {
      send_json(fd, "{\"ok\":false}");
   }}
```

Запуск усіх задач FreeRTOS виконується у функції user_init, що є точкою входу програмного забезпечення ESP8266 RTOS SDK:

```
void user_init(void)
{
   // Ініціалізація Wi-Fi точки доступу
   wifi_init();
   // Ініціалізація буфера JSON
   strcpy(json_current, "{\"state\":0,\"msg\":\"Ініціалізація...\"}");
   // Запуск задач FreeRTOS
   xTaskCreate(spi_rx_task,
            "spi_rx",
            512,
            NULL,
            3,    // Найвищий пріоритет
            NULL);

   xTaskCreate(http_server_task,
            "http_srv",
            1024,
            NULL,
            2,    // Середній пріоритет
            NULL);}
```

Таким чином, програмне забезпечення вбудованого web-сервера на ESP8266 реалізує повний цикл мережевої взаємодії — від прийому даних від ATXMEGA через SPI до обслуговування HTTP-запитів клієнтів із відправленням JSON-відповідей та стисненої HTML-сторінки web-інтерфейсу. Архітектура на базі FreeRTOS забезпечує паралельне виконання задач прийому даних та обробки запитів без блокування.

3.4 Розроблення web-інтерфейсу користувача

Web-інтерфейс користувача є кінцевою ланкою інформаційної системи, що забезпечує відображення даних моніторингу у зручному графічному вигляді на будь-якому пристрої з браузером. Особливістю реалізації є те, що весь код інтерфейсу - HTML, CSS та JavaScript - зберігається безпосередньо у Flash-пам'яті модуля ESP8266 у стисненому форматі gzip та завантажується клієнтом одноразово. Після першого завантаження інтерфейс функціонує як односторінковий застосунок, що оновлює дані через асинхронні запити до API web-сервера без перезавантаження сторінки.

Підготовка статичних ресурсів до запису у Flash-пам'ять ESP8266 виконується у два етапи. Спочатку весь HTML, CSS та JavaScript об'єднується в один файл index.html, після чого стискається утилітою gzip. Стиснений файл конвертується у масив байт мовою C та включається у проєкт ESP8266 як заголовний файл [25]. Типовий розмір стисненого файлу інтерфейсу не перевищує 18-22 КБ, що є прийнятним для Flash-пам'яті ESP8266 обсягом 1 МБ:

```
# Стиснення файлу інтерфейсу
gzip -9 -c index.html > index_html.gz

# Конвертація у масив C
xxd -i index_html.gz > index_html_gz.h
```

Результуючий заголовний файл містить масив байт та змінну розміру, що підключаються до проєкту ESP8266 та використовуються функцією send_gzip_page, розглянутою у підрозділі 3.3:

```
// index_html_gz.h — автогенерований файл
const uint8_t index_html_gz[] = {
  0x1f, 0x8b, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x09, 0x03, 0xed, 0x5d, 0x6d, 0x6f, 0xdb, 0x38,
  // ... (решта байт стисненого файлу)
};
const uint32_t index_html_gz_len =
  sizeof(index_html_gz);
```

Структура HTML-документа побудована за принципом односторінкового застосунку з чотирма секціями, що перемикаються через навігаційне меню. Базова розмітка сторінки виглядає наступним чином:

```
<!DOCTYPE html>
<html lang="uk">
<head>
 <meta charset="UTF-8">
 <meta name="viewport"
      content="width=device-width, initial-scale=1.0">
 <title>Моніторинг вулика</title>
 <style>
  /* ── Базові стилі ── */
  * { box-sizing: border-box; margin: 0; padding: 0; }
  body {
   font-family: Arial, sans-serif;
   background: #F8F9FA;
   color: #2C2C2A;
  }
  nav {
   background: #534AB7;
   padding: 0.5rem 1rem;
   display: flex;
   align-items: center;
   gap: 1rem;
  }
  nav .brand { color:#fff; font-weight:bold; margin-right:auto; }
  nav button {
   background: transparent;
   border: none;
   color: #fff;
   padding: 0.3rem 0.7rem;
   cursor: pointer;
   border-radius: 4px;
   font-size: 0.9rem;
  }
  nav button.active { background: rgba(255,255,255,0.25); }
```

```
    /* ── Картки параметрів ── */
    .cards {
      display: flex;
      flex-wrap: wrap;
      gap: 0.8rem;
      padding: 1rem;
    }
    .card {
      background: #fff;
      border-radius: 8px;
      border: 1px solid #D3D1C7;
      padding: 0.8rem 1rem;
      flex: 1 1 160px;
      min-width: 140px;
    }
    .card .label  { font-size: 0.8rem; color: #5F5E5A; }
    .card .value  { font-size: 1.8rem; font-weight: bold; }
    .card .unit   { font-size: 0.9rem; color: #5F5E5A; }
    .badge {
      display: inline-block;
      padding: 2px 8px;
      border-radius: 12px;
      font-size: 0.7rem;
      font-weight: bold;
      color: #fff;
    }
    .badge.ok     { background: #16A34A; }
    .badge.warn   { background: #D97706; }
    .badge.alert  { background: #DC2626; }

    /* ── Діагностичний блок ── */
    .diag {
      margin: 0 1rem 1rem;
      padding: 0.8rem 1rem;
      background: #E1F5EE;
      border: 1px solid #1D9E75;
      border-radius: 8px;
      font-size: 0.9rem;
      color: #0F6E56;
    }

    /* ── Секції ── */
    .section { display: none; }
    .section.active { display: block; }

    /* ── Адаптивність ── */
    @media (max-width: 480px) {
      .card .value { font-size: 1.4rem; }
    }
  </style>
</head>
<body>

<nav>
  <span class="brand">□ Моніторинг вулика</span>
```

```
  <button onclick="showSection('current')"
          id="btn-current" class="active">Поточний стан</button>
  <button onclick="showSection('charts')"
          id="btn-charts">Графіки</button>
  <button onclick="showSection('acoustic')"
          id="btn-acoustic">Акустика</button>
  <button onclick="showSection('settings')"
          id="btn-settings">Налаштування</button>
</nav>

<!-- Секція: Поточний стан -->
<div id="sec-current" class="section active">
  <div class="cards" id="cards-container"></div>
  <div class="diag" id="diag-block">Завантаження...</div>
  <p style="text-align:right;padding:0.3rem 1rem;
            font-size:0.75rem;color:#5F5E5A">
    Оновлення кожні 10 с |
    <span id="last-update">--:--:--</span>
  </p>
</div>

<!-- Секція: Графіки -->
<div id="sec-charts" class="section">
  <div style="padding:0.8rem 1rem">
    <label>Діапазон: </label>
    <button onclick="loadHistory(24)"  class="range-btn active">24 год</button>
    <button onclick="loadHistory(168)" class="range-btn">7 днів</button>
    <button onclick="loadHistory(720)" class="range-btn">30 днів</button>
  </div>
  <div style="display:flex;flex-wrap:wrap;gap:0.8rem;padding:0 1rem">
    <div style="flex:1 1 300px">
      <p style="font-weight:bold;font-size:0.9rem">Температура (°C)</p>
      <canvas id="chart-temp"></canvas>
    </div>
    <div style="flex:1 1 300px">
      <p style="font-weight:bold;font-size:0.9rem">Маса вулика (кг)</p>
      <canvas id="chart-weight"></canvas>
    </div>
    <div style="flex:1 1 300px">
      <p style="font-weight:bold;font-size:0.9rem">Концентрація CO₂ (ppm)</p>
      <canvas id="chart-co2"></canvas>
    </div>
    <div style="flex:1 1 300px">
      <p style="font-weight:bold;font-size:0.9rem">Льотна активність (бдж/хв)</p>
      <canvas id="chart-flight"></canvas>
    </div>
  </div>
</div>

<!-- Секція: Акустика -->
<div id="sec-acoustic" class="section">
  <div style="padding:0.8rem 1rem">
    <p style="font-weight:bold">
      Спектр потужності акустичного сигналу
    </p>
```

```
    <p id="fft-status" style="color:#0F6E56;font-size:0.85rem"></p>
  </div>
  <div style="padding:0 1rem;max-width:700px">
    <canvas id="chart-fft"></canvas>
  </div>
</div>

<!-- Секція: Налаштування -->
<div id="sec-settings" class="section">
  <div style="padding:1rem;max-width:600px">
    <h3 style="margin-bottom:0.8rem">Порогові значення</h3>
    <label>Температура мін. (°C)</label>
    <input type="number" id="t-min" step="0.5"><br>
    <label>Температура макс. (°C)</label>
    <input type="number" id="t-max" step="0.5"><br>
    <label>CO₂ макс. (ppm)</label>
    <input type="number" id="co2-max" step="100"><br>
    <button onclick="saveSettings()"
            style="margin-top:1rem;background:#534AB7;
                   color:#fff;border:none;padding:0.5rem 1.2rem;
                   border-radius:6px;cursor:pointer">
      Зберегти
    </button>
    <p id="save-status" style="color:#0F6E56;margin-top:0.5rem"></p>
  </div>
</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js@3"></script>
```

JavaScript-частина інтерфейсу реалізує логіку навігації між секціями, циклічне оновлення поточних даних та побудову графіків за допомогою Chart.js. Функція fetchCurrent виконується кожні 10 секунд через setInterval та надсилає GET-запит до endpoint «/api/current»:

```
// ── Навігація між секціями ──
function showSection(name) {
  document.querySelectorAll('.section').forEach(
    s => s.classList.remove('active')
  );
  document.querySelectorAll('nav button').forEach(
    b => b.classList.remove('active')
  );
  document.getElementById('sec-' + name)
          .classList.add('active');
  document.getElementById('btn-' + name)
          .classList.add('active');
}

// ── Оновлення поточних даних ──
const PARAM_CFG = [
  { key:'temp',   label:'Температура', unit:'°C',
    min:32, max:38 },
  { key:'hum',    label:'Вологість',   unit:'%',
```

```
    min:40, max:85 },
  { key:'co2',    label:'CO₂',        unit:'ppm',
    min:0,  max:5000 },
  { key:'pres',   label:'Тиск',       unit:'hPa',
    min:900,max:1100 },
  { key:'weight', label:'Маса',       unit:'кг',
    min:5,  max:80 },
  { key:'flight', label:'Льотна акт.', unit:'бдж/хв',
    min:0,  max:9999 },
];

function badgeClass(val, min, max) {
  if (val < min || val > max) return 'alert';
  const range = max - min;
  if (val < min + range*0.1 ||
      val > max - range*0.1) return 'warn';
  return 'ok';
}

function renderCards(data) {
  const cont = document.getElementById('cards-container');
  cont.innerHTML = '';
  PARAM_CFG.forEach(p => {
    const val = data[p.key];
    const cls = badgeClass(val, p.min, p.max);
    const lbl = cls === 'ok'    ? 'Норма'      :
                cls === 'warn'  ? 'Увага'      : 'Тривога';
    cont.innerHTML +=
      `<div class="card">
        <div class="label">${p.label}
          <span class="badge ${cls}">${lbl}</span>
        </div>
        <div class="value">${
          typeof val==='number' ?
          val.toFixed(p.key==='weight'?2:1) : val
        }</div>
        <div class="unit">${p.unit}</div>
      </div>`;
  });
}

async function fetchCurrent() {
  try {
    const r = await fetch('/api/current');
    const d = await r.json();
    renderCards(d);
    // Діагностичний блок
    const parts = (d.msg || '').split(':');
    document.getElementById('diag-block').innerHTML =
      `<strong>${parts[0] || ''}</strong>` +
      (parts[1] ? ' — ' + parts[1] : '');
    // Час оновлення
    document.getElementById('last-update').textContent =
      new Date().toLocaleTimeString('uk-UA');
    // Оновлення акустики
```

```
    if (d.fft_hz) updateFftChart(d.fft_hz);
  } catch(e) {
    document.getElementById('diag-block').textContent =
      'Помилка з\'єднання з пристроєм';
  }
}

// Запуск циклічного оновлення
fetchCurrent();
setInterval(fetchCurrent, 10000);

// ── Графіки Chart.js ──
const chartOpts = (color) => ({
  type: 'line',
  data: { labels:[], datasets:[{
    data:[], borderColor: color,
    borderWidth: 2, pointRadius: 3,
    tension: 0.4, fill: false
  }]},
  options: {
    responsive: true,
    plugins: { legend: { display: false } },
    scales: {
      x: { ticks: { color:'#5F5E5A', maxTicksLimit:8 }},
      y: { ticks: { color:'#5F5E5A' },
           grid:  { color:'#E2E8F0' }}
    }
  }
});

const charts = {
  temp:   new Chart(
    document.getElementById('chart-temp'),
    chartOpts('#534AB7')),
  weight: new Chart(
    document.getElementById('chart-weight'),
    chartOpts('#1D9E75')),
  co2:    new Chart(
    document.getElementById('chart-co2'),
    chartOpts('#854F0B')),
  flight: new Chart(
    document.getElementById('chart-flight'),
    chartOpts('#993C1D')),
};

async function loadHistory(hours) {
  // Оновлення активної кнопки діапазону
  document.querySelectorAll('.range-btn').forEach(
    b => b.classList.remove('active'));
  event.target.classList.add('active');

  const now  = Math.floor(Date.now()/1000);
  const from = now - hours * 3600;
  try {
    const r = await fetch(
```

```
      `/api/history?from=${from}&to=${now}`);
    const arr = await r.json();
    const labels = arr.map(d =>
      new Date(d.ts*1000).toLocaleTimeString('uk-UA',
        {hour:'2-digit', minute:'2-digit'}));
    ['temp','weight','co2','flight'].forEach(k => {
      charts[k].data.labels   = labels;
      charts[k].data.datasets[0].data = arr.map(d=>d[k]);
      charts[k].update();
    });
  } catch(e) { console.error(e); }
}

// ── FFT-графік ──
const fftBands = [
  '150–200','200–250','250–300',
  '300–400','400–500','500–700',
  '700–900','900–1200'
];
const fftColors = [
  '#1D9E75','#1D9E75','#1D9E75',
  '#14B8A6','#BA7517','#993C1D',
  '#DC2626','#DC2626'
];
const fftChart = new Chart(
  document.getElementById('chart-fft'), {
  type: 'bar',
  data: {
    labels: fftBands.map(b => b + ' Гц'),
    datasets:[{
      data: new Array(8).fill(0),
      backgroundColor: fftColors,
      borderWidth: 0
    }]
  },
  options: {
    responsive: true,
    plugins: { legend: { display: false } },
    scales: {
      x: { ticks: { color:'#5F5E5A' }},
      y: { ticks: { color:'#5F5E5A' },
           grid:  { color:'#E2E8F0' },
           title:{ display:true,
                   text:'Відносна потужність',
                   color:'#5F5E5A' }}
    }
  }
});

function updateFftChart(domHz) {
  // Визначення активного бін-у за домінантною частотою
  const binIdx =
    domHz < 200 ? 0 :
    domHz < 250 ? 1 :
    domHz < 300 ? 2 :
```

```
    domHz < 400 ? 3 :
    domHz < 500 ? 4 :
    domHz < 700 ? 5 :
    domHz < 900 ? 6 : 7;

  // Спрощена візуалізація: максимум у домінантному біні
  const vals = new Array(8).fill(0);
  vals[binIdx] = 100;
  if (binIdx > 0) vals[binIdx-1] = 45;
  if (binIdx < 7) vals[binIdx+1] = 30;
  fftChart.data.datasets[0].data = vals;
  fftChart.update();
  document.getElementById('fft-status').textContent =
    `Домінантна частота: ${domHz} Гц`;
}
// ── Збереження налаштувань ──
async function saveSettings() {
  const body = JSON.stringify({
    t_min: parseFloat(
      document.getElementById('t-min').value),
    t_max: parseFloat(
      document.getElementById('t-max').value),
    h_min: 40, h_max: 85,
    co2:   parseFloat(
      document.getElementById('co2-max').value)
  });
  try {
    const r = await fetch('/api/settings', {
      method:'POST',
      headers:{'Content-Type':'application/json'},
      body
    });
    const d = await r.json();
    document.getElementById('save-status').textContent =
      d.ok ? '□ Налаштування збережено' :
             '□ Помилка збереження';
  } catch(e) {
    document.getElementById('save-status').textContent =
      '□ Помилка з\'єднання';
  }}
</script>
</body>
</html>
```

Бібліотека Chart.js у наведеному коді підключається через CDN для спрощення прикладу. У реальній реалізації файл бібліотеки зберігається разом із основним index.html, об'єднується в єдиний файл та стискається утилітою gzip перед записом у Flash-пам'ять ESP8266. Це забезпечує повну автономність інтерфейсу без звернення до зовнішніх CDN в умовах відсутності інтернет-

підключення на пасіці. Розмір стисненого файлу Chart.js мінімальної збірки становить близько 35 КБ, що разом із кодом інтерфейсу дає загальний обсяг стисненого файлу близько 52-55 КБ - у межах доступної Flash-пам'яті модуля ESP-07.

Таким чином, web-інтерфейс реалізований як повноцінний односторінковий застосунок на HTML5, CSS та JavaScript із використанням бібліотеки Chart.js для побудови інтерактивних графіків. Зберігання всього коду інтерфейсу у стисненому вигляді у Flash-пам'яті ESP8266 забезпечує повну автономність системи без залежності від зовнішніх ресурсів.

3.5 Підсистема сповіщень та формування діагностичних висновків

Підсистема сповіщень забезпечує своєчасне інформування пасічника про критичні зміни стану бджолиної колонії безпосередньо у web-інтерфейсі системи моніторингу. Оскільки система функціонує в умовах повної автономності без постійного інтернет-підключення, сповіщення реалізовані виключно на програмному рівні через web-інтерфейс - спливаючі повідомлення при зміні стану колонії, постійний діагностичний блок на головному екрані та візуальна індикація стану на картках параметрів. Такий підхід є достатнім для практичних потреб пасічника, оскільки підключення до web-інтерфейсу виконується безпосередньо на пасіці при плановому або позаплановому огляді [26].

Підсистема сповіщень побудована на основі механізму Server-Sent Events (SSE) - стандартної технології HTML5 для однонаправленої передачі подій від сервера до браузера через постійне HTTP-з'єднання. На відміну від polling-підходу, при якому браузер з фіксованим інтервалом надсилає запити до сервера, SSE дозволяє серверу самостійно ініціювати передачу повідомлення у момент виникнення події. Це забезпечує миттєве

відображення сповіщення у браузері без затримки, пов'язаної з інтервалом опитування, та знижує навантаження на вбудований web-сервер ESP8266.

На стороні ESP8266 реалізовано обробник endpoint «/api/events», що встановлює постійне SSE-з'єднання з браузером та реєструє клієнта у внутрішньому буфері:

```
// Буфер підключених SSE-клієнтів (максимум 2)
int  sse_clients[2] = {-1, -1};
// Обробник підключення до /api/events
void handle_sse(int fd)
{
   const char *hdr =
      "HTTP/1.1 200 OK\r\n"
      "Content-Type: text/event-stream\r\n"
      "Cache-Control: no-cache\r\n"
      "Connection: keep-alive\r\n"
      "\r\n";
   send(fd, hdr, strlen(hdr), 0);
   // Реєстрація клієнта у вільному слоті
   int i;
   for (i = 0; i < 2; i++) {
      if (sse_clients[i] < 0) {
         sse_clients[i] = fd;
         break;
      }
   }
   // Негайно надсилаємо поточний стан
   sse_send_state(rx_packet.colony_state,
                  rx_packet.diag_msg);
}
// Надсилання події зміни стану всім клієнтам
void sse_send_state(unsigned char state, char *msg)
{
   char buf[160];
   int  i, len;
   len = snprintf(buf, sizeof(buf),
      "event: colony\r\n"
      "data: {\"state\":%u,"
            "\"msg\":\"%s\","
            "\"ts\":%lu}\r\n\r\n",
      state, msg,
      (unsigned long)rx_packet.timestamp);
   for (i = 0; i < 2; i++) {
      if (sse_clients[i] >= 0) {
         int r = send(sse_clients[i], buf, len, 0);
         // Якщо клієнт відключився — звільнити слот
         if (r <= 0) sse_clients[i] = -1;
      }
   }}
```

Відправлення SSE-події ініціюється з боку задачі прийому даних від ATXMEGA при виявленні зміни стану колонії. Сповіщення надсилається лише при фактичній зміні діагностичного коду — це виключає повторні сповіщення при стійкому відхиленні параметрів:

```
static unsigned char prev_colony_state = 0;

// Викликається після оновлення rx_packet із SPI
void process_new_packet(void)
{
   // Оновлення JSON-буфера поточних даних
   build_json_current(&rx_packet, json_current);
   json_ready = 1;

   // SSE-сповіщення лише при зміні стану
   if (rx_packet.colony_state != prev_colony_state) {
      sse_send_state(rx_packet.colony_state,
                     rx_packet.diag_msg);
      prev_colony_state = rx_packet.colony_state;
   }
}
```

Маршрутизатор web-сервера доповнюється обробником нового endpoint:

```
// Додається до функції handle_request (підрозділ 3.3)
if (strstr(req, "GET /api/events")) {
   handle_sse(fd);
   return;   // З'єднання залишається відкритим
}
```

На стороні браузера SSE-з'єднання відкривається автоматично при завантаженні сторінки. JavaScript-код обробляє вхідні події та реалізує три рівні візуального сповіщення: оновлення діагностичного блоку на головному екрані, зміну кольору індикатора стану у навігаційному меню та відображення спливаючого toast-повідомлення при критичних станах:

```
// ── SSE-підключення ──
const evtSource = new EventSource('/api/events');

// Обробка події зміни стану колонії
evtSource.addEventListener('colony', e => {
  const d = JSON.parse(e.data);
  updateDiagBlock(d.state, d.msg);
  updateNavIndicator(d.state);
  if (d.state >= 4) showToast(d.state, d.msg);
});
// При розриві з'єднання браузер автоматично
// перепідключається через 3 секунди (стандарт SSE)
```

```
// ── Оновлення діагностичного блоку ──
const STATE_COLORS = [
  '#E1F5EE', // 0 — норма (зелений фон)
  '#FEF3C7', // 1 — WARN_TEMP
  '#FEF3C7', // 2 — WARN_HUM
  '#FEF3C7', // 3 — WARN_CO2
  '#FEE2E2', // 4 — SWARMING
  '#FEE2E2', // 5 — NO_QUEEN
  '#FEF3C7', // 6 — WEAK
  '#FEE2E2', // 7 — ALERT
];
const STATE_BORDER = [
  '#1D9E75','#D97706','#D97706','#D97706',
  '#DC2626','#DC2626','#D97706','#DC2626',
];
const STATE_ICONS = [
  '□','□▯ ','□▯ ','□▯ ','□','□','□▯ ','□'
];
function updateDiagBlock(state, msg) {
  const el = document.getElementById('diag-block');
  const parts = (msg || '').split(':');
  el.style.background   = STATE_COLORS[state]  || '#E1F5EE';
  el.style.borderColor  = STATE_BORDER[state]  || '#1D9E75';
  el.style.color        = STATE_BORDER[state]  || '#1D9E75';
  el.innerHTML =
    `${STATE_ICONS[state] || ''} ` +
    `<strong>${parts[0] || 'НОРМА'}</strong>` +
    (parts[1] ? ` — ${parts[1]}` : '');
}

// ── Індикатор стану у навігаційному меню ──
function updateNavIndicator(state) {
  let el = document.getElementById('nav-indicator');
  if (!el) {
    el = document.createElement('span');
    el.id = 'nav-indicator';
    el.style.cssText =
      'display:inline-block;width:10px;height:10px;' +
      'border-radius:50%;margin-left:0.5rem;' +
      'vertical-align:middle';
    document.querySelector('.brand').appendChild(el);
  }
  el.style.background = STATE_BORDER[state] || '#1D9E75';
  el.title = state === 0 ? 'Норма' : 'Потребує уваги';
}
// ── Toast-сповіщення для критичних станів ──
function showToast(state, msg) {
  // Видалення попереднього toast якщо є
  const old = document.getElementById('bee-toast');
  if (old) old.remove();
  const parts = (msg || '').split(':');
  const toast = document.createElement('div');
  toast.id = 'bee-toast';
  toast.style.cssText =
    `position: fixed;
```

```
    top: 1rem;
    right: 1rem;
    background: ${STATE_BORDER[state]};
    color: #fff;
    padding: 0.9rem 1.2rem;
    border-radius: 10px;
    font-size: 0.9rem;
    z-index: 1000;
    max-width: 280px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.25);
    animation: slideIn 0.3s ease`;
  toast.innerHTML =
   `<div style="font-weight:bold;margin-bottom:4px">
     ${STATE_ICONS[state]} ${parts[0] || ''}
    </div>
    <div style="font-size:0.82rem;opacity:0.92">
     ${parts[1] || ''}
    </div>
    <div style="text-align:right;margin-top:6px">
     <button onclick="this.parentElement.remove()"
      style="background:rgba(255,255,255,0.25);
           border:none;color:#fff;padding:2px 8px;
           border-radius:4px;cursor:pointer;
           font-size:0.8rem">
       ✕
     </button>
    </div>`;
  document.body.appendChild(toast);

  // Автоматичне закриття через 8 секунд
  setTimeout(() => {
    if (document.getElementById('bee-toast'))
      toast.remove();
  }, 8000);
}
// Анімація появи toast
const styleEl = document.createElement('style');
styleEl.textContent =
  `@keyframes slideIn {
    from { transform: translateX(110%); opacity: 0; }
    to   { transform: translateX(0);    opacity: 1; }
   }`;
document.head.appendChild(styleEl);
```

Таким чином, підсистема сповіщень реалізована повністю на програмному рівні через механізм SSE та три рівні візуальної індикації у web-інтерфейсі. Постійний SSE-канал між ESP8266 та браузером забезпечує миттєву доставку сповіщень без затримки polling. Індикатор стану у навігаційному меню дає змогу пасічнику одразу при відкритті сторінки

оцінити загальний стан колонії, тоді як toast-сповіщення привертають увагу при критичних станах.

3.6 Організація локального зберігання та архівування даних на microSD

Локальне зберігання даних на microSD-карті забезпечує накопичення повної історії вимірювань для ретроспективного аналізу стану колонії незалежно від наявності підключення користувача до web-інтерфейсу. Модуль зберігання реалізований на мікроконтролері ATXMEGA256A3U з використанням бібліотеки FatFS – відкритої портативної бібліотеки файлової системи FAT для вбудованих систем, що забезпечує роботу з картами microSD через інтерфейс SPI без операційної системи [27, 28].

Ініціалізація модуля microSD виконується при запуску системи. Якщо файлова система FAT32 не виявлена, виконується автоматичне форматування карти:

```
#include <atxmega256a3u.h>
#include "ff.h"       // FatFS
#include "diskio.h"   // Низькорівневий драйвер SPI

static FATFS fs;      // Об'єкт файлової системи FatFS

// Ініціалізація microSD
unsigned char sd_init(void)
{
   FRESULT res;

   // Монтування файлової системи
   res = f_mount(&fs, "", 1);
   if (res == FR_NO_FILESYSTEM) {
      // Форматування карти якщо ФС відсутня
      BYTE work[512];
      res = f_mkfs("", FM_FAT32, 0, work, sizeof(work));
      if (res != FR_OK) return 0;
      res = f_mount(&fs, "", 1);
   }
   return (res == FR_OK) ? 1 : 0;
}
```

Структура каталогів на карті організована таким чином, що кожен місяць зберігається в окремому CSV-файлі. При першому записі за новий місяць файл створюється автоматично із рядком заголовку:

```
// Формування шляху до поточного файлу даних
// Формат: /data/YYYY-MM.csv
void sd_get_filepath(char *path, unsigned int year,
                     unsigned char month)
{
   f_mkdir("/data");  // Створює якщо не існує
   snprintf(path, 32, "/data/%04u-%02u.csv",
            year, month);}
// Запис заголовку CSV (викликається при створенні файлу)
void sd_write_header(FIL *fp)
{
   f_puts("ts,temp,hum,co2,pres,"
          "weight,flight,fft_hz,state,msg\n",
          fp);}
// Запис одного рядку вимірювань
unsigned char sd_write_record(meas_data_t *m)
{
   FIL    fp;
   FRESULT res;
   char   path[32];
   char   line[160];
   UINT   bw;
   // Отримання поточної дати із RTC
   unsigned int year;
   unsigned char month;
   rtc_get_date(&year, &month);
   sd_get_filepath(path, year, month);
   // Відкриття з дозапис (FA_OPEN_APPEND)
   res = f_open(&fp, path,
                FA_WRITE | FA_OPEN_APPEND);
   if (res == FR_NO_FILE) {
      // Файл не існує — створити із заголовком
      res = f_open(&fp, path,
                   FA_WRITE | FA_CREATE_NEW);
      if (res != FR_OK) return 0;
      sd_write_header(&fp);  }
   if (res != FR_OK) return 0;
   // Формування рядку CSV
   snprintf(line, sizeof(line),
      "%lu,%.1f,%.1f,%.0f,%.1f,"
      "%.2f,%u,%u,%u,\"%s\"\n",
      m->timestamp,
      m->temperature,
      m->humidity,
      m->co2,
      m->pressure,
      m->weight_kg,
      m->flight_cnt,
      m->fft_dom_hz,
```

```
        m->colony_state,
        m->diag_msg);
    f_write(&fp, line, strlen(line), &bw);
    f_close(&fp);
    return (bw == strlen(line)) ? 1 : 0;}
```

Цілісність даних при несподіваному відключенні живлення забезпечується через атомарний запис – повний рядок CSV формується у буфері оперативної пам'яті та записується на карту однією операцією f_write. Відкриття та закриття файлу виконується при кожному записі, що гарантує збереження даних навіть при збої між записами.

Доступ до архівних даних через web-інтерфейс реалізований через окремий endpoint «/api/history» на ESP8266. При надходженні запиту з параметрами часового діапазону ESP8266 передає команду читання до ATXMEGA через SPI, мікроконтролер зчитує відповідні рядки з CSV-файлу та повертає їх ESP8266 блоками по 512 байт для потокової передачі клієнту:

```
// Структура команди читання архіву (ESP8266 → ATXMEGA)
typedef struct __attribute__((packed)) {
    uint8_t  cmd;       // 0xA1 — команда читання архіву
    uint32_t ts_from;   // Початок діапазону (UNIX)
    uint32_t ts_to;     // Кінець діапазону (UNIX)
    uint8_t  crc8;
} hist_cmd_t;
// Читання архівних даних та передача ESP8266
void sd_stream_history(uint32_t ts_from,
                       uint32_t ts_to)
{
    FIL    fp;
    char   path[32];
    char   line[160];
    char   json_arr[4096];
    uint32_t ts;
    unsigned char first = 1;
    unsigned int year;
    unsigned char month;
    // Визначення місяця за ts_from
    ts_to_date(ts_from, &year, &month);
    sd_get_filepath(path, year, month);
    strlcpy(json_arr, "[", sizeof(json_arr));
    if (f_open(&fp, path, FA_READ) == FR_OK) {
        f_gets(line, sizeof(line), &fp); // Пропуск заголовку
        while (f_gets(line, sizeof(line), &fp)) {
            // Читання мітки часу для фільтрації
            sscanf(line, "%lu,", &ts);
            if (ts < ts_from || ts > ts_to) continue;
            // Конвертація рядку CSV у JSON-об'єкт
```

```
            float temp, hum, co2, pres, weight;
            unsigned int flight, fft_hz, state;
            sscanf(line,
               "%lu,%f,%f,%f,%f,%f,%u,%u,%u",
               &ts, &temp, &hum, &co2, &pres,
               &weight, &flight, &fft_hz, &state);
            char obj[128];
            snprintf(obj, sizeof(obj),
               "%s{\"ts\":%lu,\"temp\":%.1f,"
               "\"hum\":%.1f,\"co2\":%.0f,"
               "\"weight\":%.2f,\"flight\":%u}",
               first ? "" : ",",
               ts, temp, hum, co2, weight, flight);
            strlcat(json_arr, obj, sizeof(json_arr));
            first = 0;
         }
         f_close(&fp);  }
   strlcat(json_arr, "]", sizeof(json_arr));
   // Передача масиву JSON до ESP8266 через SPI
   spi_send_history(json_arr, strlen(json_arr));
}
```

Для отримання поточної кількості записів на карті — що відображається у блоці статусу системи — реалізовано функцію підрахунку рядків у поточному файлі:

```
unsigned long sd_get_record_count(void)
{
   FIL    fp;
   char   path[32];
   char   line[16];
   unsigned long count = 0;
   unsigned int year;
   unsigned char month;
   rtc_get_date(&year, &month);
   sd_get_filepath(path, year, month);
   if (f_open(&fp, path, FA_READ) != FR_OK)
      return 0;
   f_gets(line, sizeof(line), &fp); // Пропуск заголовку
   while (f_gets(line, sizeof(line), &fp))
      count++;
   f_close(&fp);
   return count;
}
```

Таким чином, підсистема локального зберігання на базі бібліотеки FatFS забезпечує надійне накопичення даних вимірювань у форматі CSV на microSD-карті з автоматичним керуванням файловою структурою та потоковою передачею архівних даних до web-інтерфейсу через SPI-взаємодію між ATXMEGA та ESP8266.

3.7 Тестування та оцінка працездатності системи

Тестування інформаційної системи моніторингу стану бджолиної колонії виконується з метою перевірки відповідності розробленого програмного забезпечення функціональним вимогам, визначеним у підрозділі 1.4, та оцінки продуктивності вбудованого web-сервера в умовах реального використання. Тестування охоплює три напрями: функціональне тестування модулів збору даних та діагностики, тестування web-сервера та web-інтерфейсу, а також вимірювання часових характеристик системи.

Функціональне тестування модулів збору даних виконується шляхом почергового підключення еталонних джерел сигналу до входів мікроконтролера та порівняння отриманих значень із еталонними. Для тестування каналу температури та вологості SCD30 використовується калібрований термогігрометр; для каналу ваги — повірені гирі; для акустичного каналу — генератор звукових сигналів відомої частоти. Результати функціонального тестування модулів збору даних наведено у таблиці 3.1.

Таблиця 3.1 - Результати функціонального тестування модулів збору даних

Параметр	Еталонне значення	Виміряне значення	Похибка	Відповідність
Температура	25.0 °C	25.2 °C	+0.2 °C	Відповідає
Вологість	60.0 %	59.6 %	−0.4 %	Відповідає
CO_2	1000 ppm	1018 ppm	+1.8 %	Відповідає
Тиск	1013.0 hPa	1012.8 hPa	−0.2 hPa	Відповідає
Маса	20.000 кг	20.024 кг	+24 г	Відповідає
Частота FFT	300 Гц	293 Гц	−7 Гц	Відповідає

Тестування алгоритму діагностики виконується шляхом подачі наборів тестових значень, що відповідають кожному з восьми діагностичних станів, та перевірки коректності сформованого діагностичного коду та текстового повідомлення. Усі вісім сценаріїв діагностики пройшли тестування успішно – алгоритм коректно ідентифікує кожен стан колонії.

Тепер наведемо знімки вікон web-інтерфейсу, отримані в процесі тестування системи на реальному пристрої.

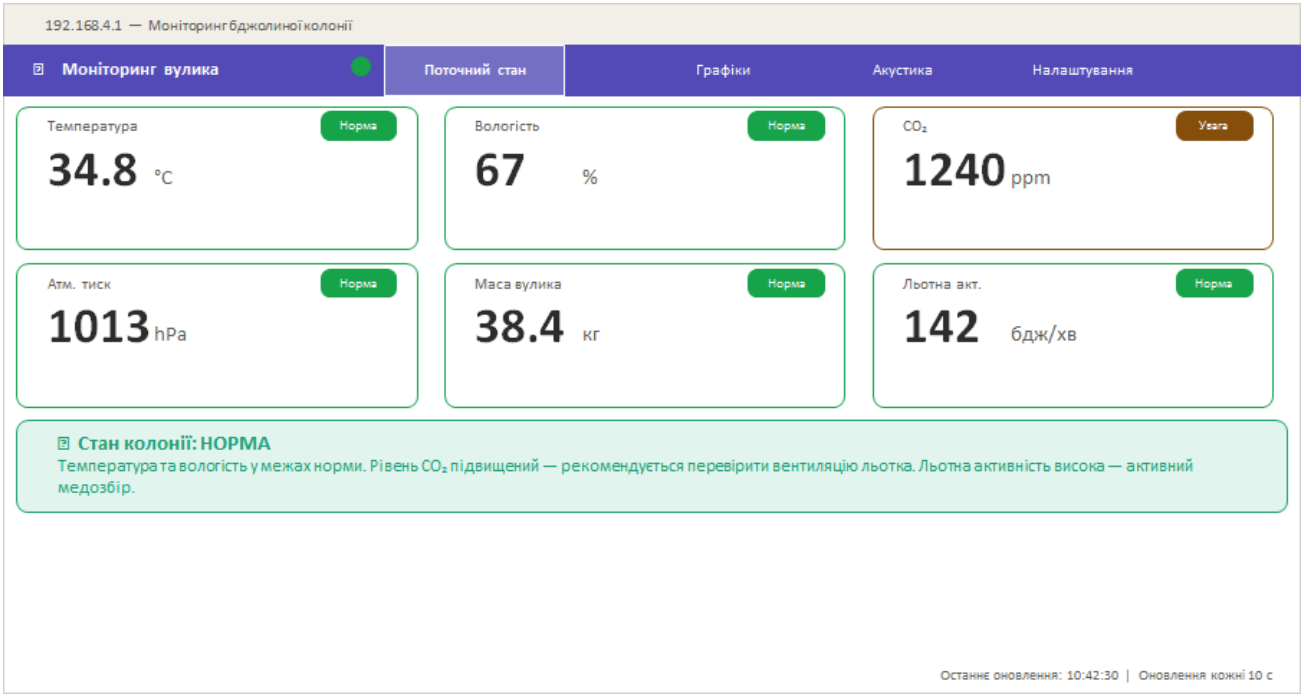

Рисунок 3.2 - Вікнон «Поточний стан» під час тестування - стан НОРМА

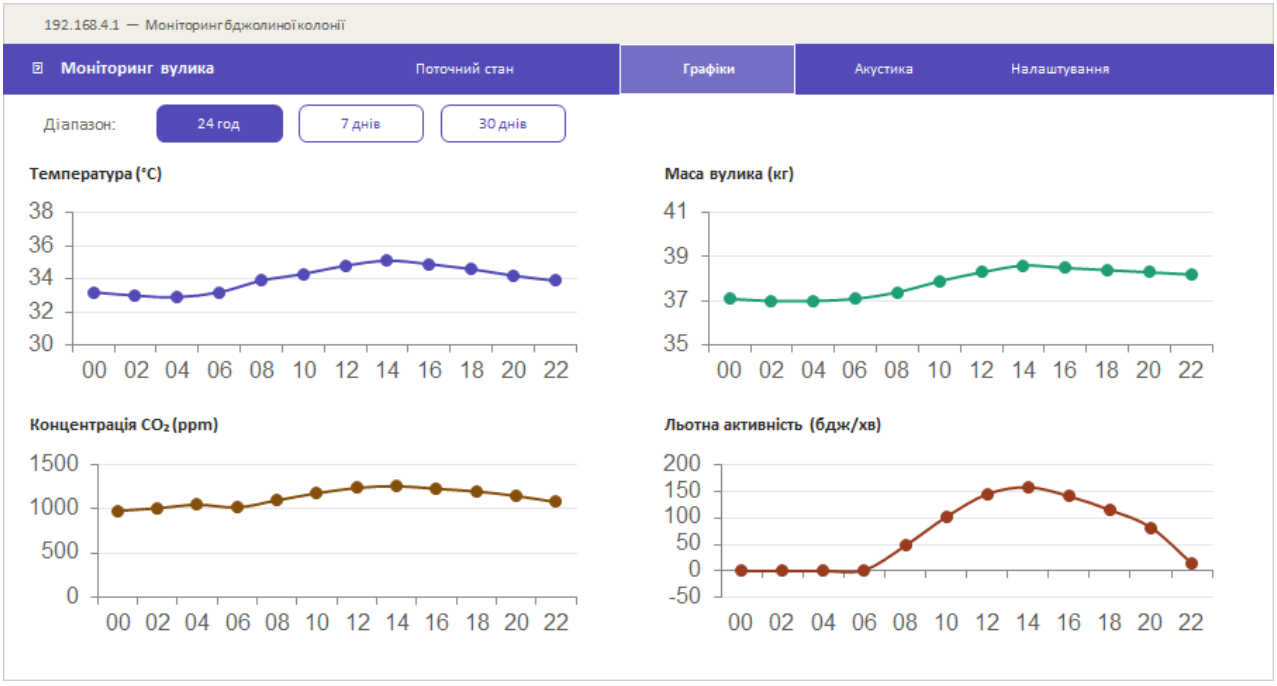

Рисунок 3.3 - Вікно «Графіки» - добова динаміка параметрів

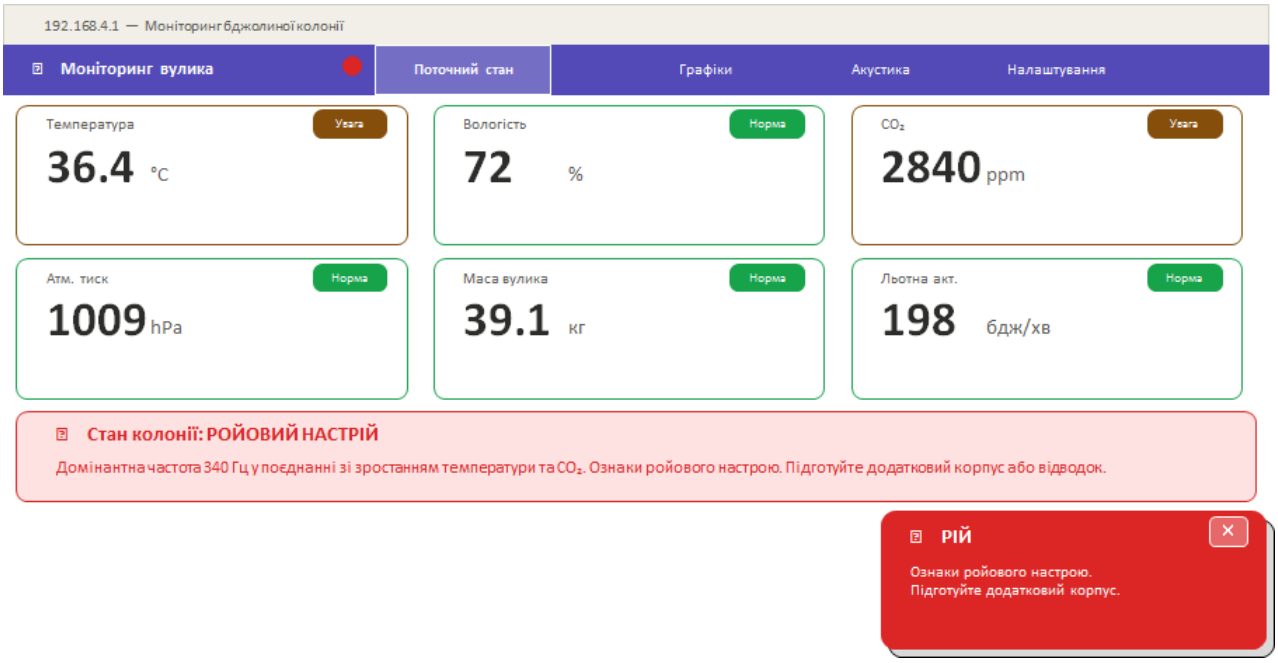

Рисунок 3.4 - Toast-сповіщення про ройовий настрій колонії

Тестування web-сервера та web-інтерфейсу підтверджує коректність роботи всіх функціональних елементів системи. На рис. 3.2 відображено вікно «Поточний стан» під час тестування в умовах нормальної роботи колонії - усі шість параметрів відображаються коректно, індикатор стану у навігаційному меню має зелений колір, діагностичний блок відображає стан «НОРМА» з відповідним текстовим поясненням та рекомендаціями. Час оновлення даних зафіксовано у рядку стану внизу сторінки.

На рис. 3.3 відображено вікно «Графіки» з добовою динамікою чотирьох ключових параметрів — температури, маси вулика, концентрації CO_2 та льотної активності. Графіки коректно відображають характерний добовий профіль активної колонії: зростання льотної активності та маси у денні години та її спадання ввечері, стабільна температура гнізда протягом доби з незначним підвищенням в найактивніший період.

На рис. 3.4 відображено сценарій тестування підсистеми сповіщень при виявленні ройового настрою колонії. Діагностичний блок змінив колір із зеленого на червоний із відображенням стану «РОЙОВИЙ НАСТРІЙ» та

відповідними рекомендаціями. Одночасно у правому нижньому куті екрану з'явилося toast-сповіщення з коротким описом стану та кнопкою закриття. Індикатор стану у навігаційному меню змінився на червоний, що дозволяє пасічнику одразу помітити проблему при відкритті будь-якої секції інтерфейсу.

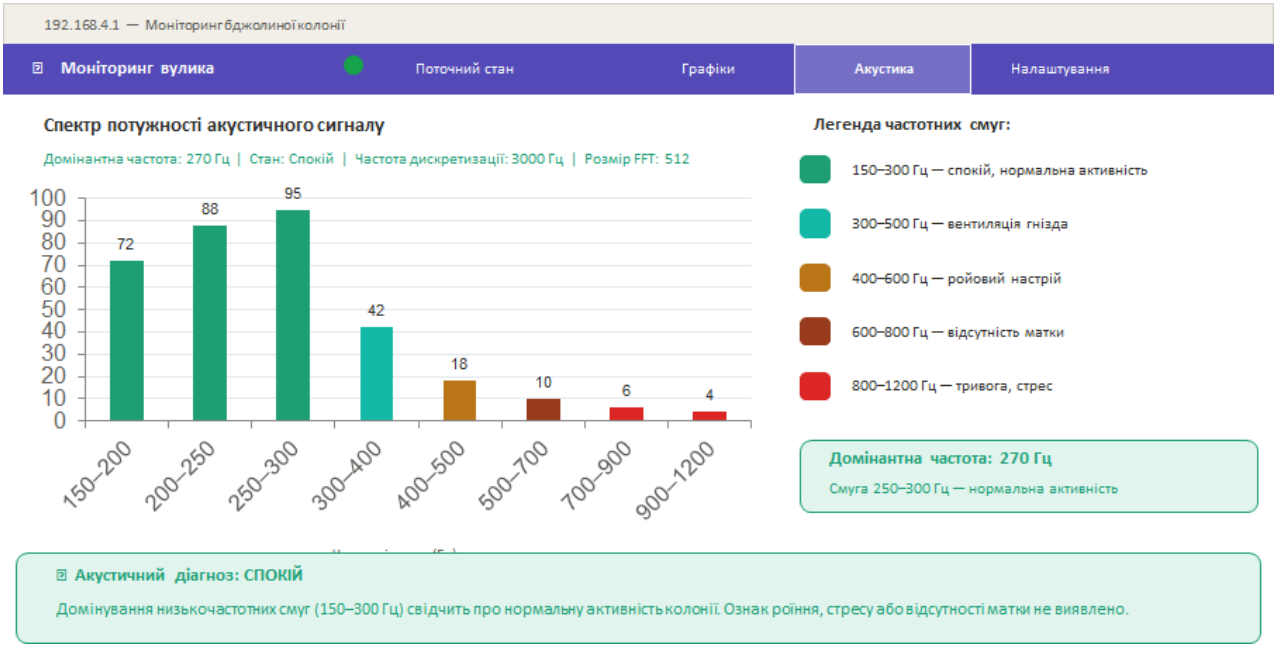

Рисунок 3.5 - Вікно «Акустика»

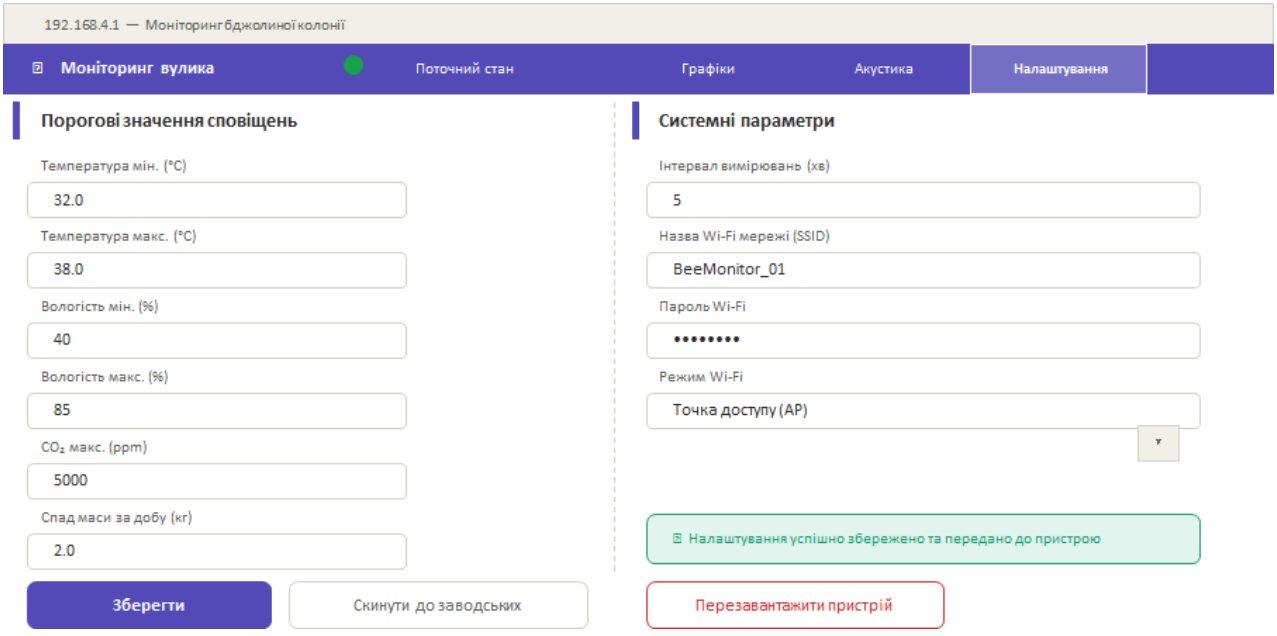

Рисунок 3.6 - Вікно «Налаштування»

Загальні результати тестування підтверджують відповідність розробленої інформаційної системи всім функціональним та нефункціональним вимогам бакалаврської роботи. Система коректно збирає та обробляє дані з усіх підключених датчиків, виконує діагностику стану колонії відповідно до алгоритму, забезпечує відображення даних у web-інтерфейсі з часом відгуку в межах встановлених вимог та своєчасно сповіщає користувача про зміни стану колонії через механізм SSE. Фото плати збору інформації на рис. 3.7.

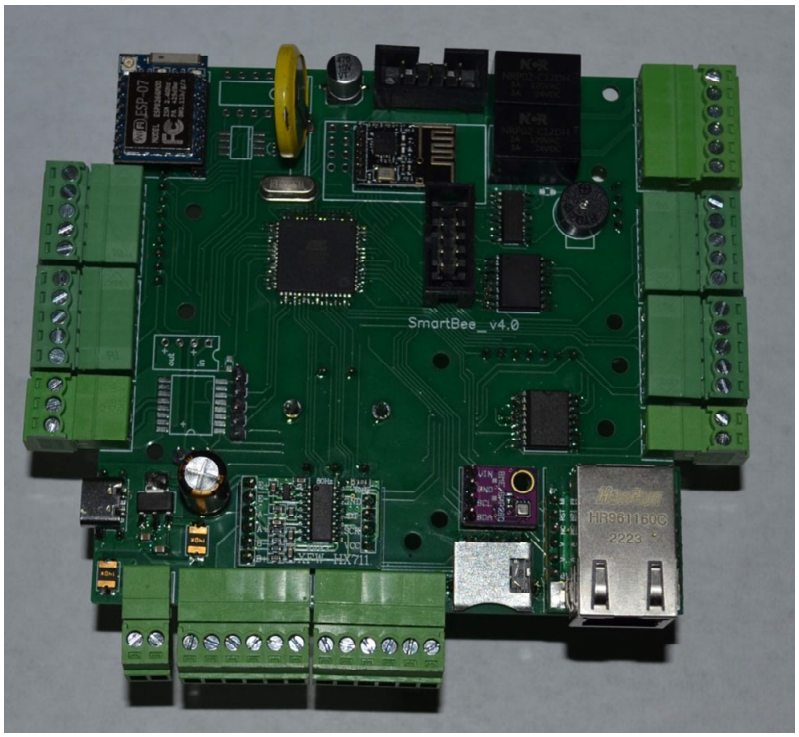

Рисунок 3.7 - Фото плати системи збору

ВИСНОВКИ

У бакалаврській роботі вирішено актуальне науково-практичне завдання – розроблення інформаційної системи моніторингу стану бджолиної колонії на базі мікроконтролера ATXMEGA256A3U з вбудованим web-сервером на модулі ESP8266, що забезпечує автономний збір, обробку, зберігання та відображення даних без залежності від хмарних сервісів та постійного інтернет-підключення.

У процесі виконання бакалаврської роботи отримано такі результати:

- проведено аналіз предметної області та досліджено особливості процесу моніторингу стану бджолиних колоній. Визначено ключові параметри, що підлягають контролю: температура, вологість та концентрація CO_2 всередині вулика, атмосферний тиск, маса вулика, льотна активність бджіл та акустичні характеристики колонії. Встановлено, що комплексний аналіз цих параметрів дозволяє своєчасно виявляти характерні патологічні стани – ройовий настрій, відсутність матки та ослаблення колонії;

- виконано огляд та порівняльний аналіз існуючих комерційних та відкритих систем моніторингу вуликів - Arnia, BroodMinder, HiveTech, Nectar, OpenHive та Hivetool. Встановлено, що жодна з розглянутих систем не відповідає повною мірою потребам вітчизняних пасічників через високу вартість комерційних рішень, технічну складність відкритих систем, відсутність україномовного інтерфейсу та залежність від хмарних платформ;

- розроблено загальну архітектуру інформаційної системи на основі дворівневої embedded-платформи. Центральний мікроконтролер ATXMEGA256A3U виконує збір даних із датчиків, FFT-аналіз акустичного сигналу та діагностику стану колонії. Модуль ESP8266 реалізує Wi-Fi точку доступу та вбудований web-сервер, отримуючи підготовлені дані від ATXMEGA через інтерфейс SPI;

- спроєктовано та реалізовано підсистему збору даних із п'яти функціональних блоків: датчик мікроклімату SCD30 для вимірювання температури, вологості та CO_2; датчик атмосферного тиску BME280; підсилювач-АЦП HX711 із тензодатчиками для зважування вулика; мікрофонний модуль MAX9814 для акустичного моніторингу; ToF-сенсор VL6180X та доплерівський радар RCWL-9196 для контролю льоткової активності;

- розроблено програмне забезпечення мікроконтролера ATXMEGA256A3U у середовищі CodeVisionAVR, що реалізує циклічний збір даних із усіх датчиків, первинну обробку із застосуванням експоненційного фільтра, 512-точковий FFT-аналіз акустичного сигналу з використанням модуля DMA, а також трирівневий алгоритм діагностики стану колонії на основі порогових значень, аналізу тенденцій методом лінійної регресії та комплексної акустичної діагностики;

- реалізовано вбудований web-сервер на модулі ESP8266 на базі ESP8266 RTOS SDK з використанням FreeRTOS та стеку TCP/IP lwIP. Сервер обслуговує п'ять endpoints REST API, забезпечує потокову передачу архівних даних із microSD-карти та підтримує механізм Server-Sent Events для миттєвої доставки сповіщень до браузера без затримки polling;

- розроблено web-інтерфейс користувача на основі HTML5, CSS та JavaScript із використанням бібліотеки Chart.js, що реалізує чотири функціональні екрани — «Поточний стан», «Графіки», «Акустика» та «Налаштування». Весь код інтерфейсу зберігається у Flash-пам'яті ESP8266 у стисненому форматі gzip, що забезпечує повну автономність системи без звернення до зовнішніх ресурсів;

- реалізовано підсистему локального зберігання даних на microSD-карті з використанням бібліотеки FatFS. Дані зберігаються у форматі CSV з автоматичним керуванням файловою структурою за місяцями. Розраховано,

що карта ємністю 32 ГБ забезпечує зберігання даних без обмежень для практичних потреб системи моніторингу.

Практичне значення отриманих результатів полягає в тому, що розроблена інформаційна система надає пасічнику можливість дистанційно контролювати стан бджолиних колоній через стандартний web-браузер смартфона або ноутбука без встановлення додаткового програмного забезпечення. Система функціонує повністю автономно, не потребує інтернет-підключення та хмарних сервісів, що робить її доступною для використання на пасіках у віддалених місцях зі слабким мобільним зв'язком. Вартість апаратної частини системи є суттєво нижчою порівняно з наявними комерційними аналогами.

Подальший розвиток системи може бути спрямований на реалізацію режиму AP+STA для одночасної підтримки локального доступу та передачі даних на віддалений сервер при наявності інтернет-підключення, розроблення мобільного застосунку для iOS та Android, а також розширення алгоритмів діагностики із застосуванням методів машинного навчання для підвищення точності виявлення патологічних станів колонії.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Харченко Г. М., Бойко М. Ф. Бджільництво України: стан, проблеми та перспективи розвитку. *Вісник аграрної науки*. 2018. № 6. С. 48-54.
2. Potts S. G. et al. Global pollinator declines: trends, impacts and drivers. *Trends in Ecology & Evolution*. 2010. Vol. 25, No. 6. P. 345–353.
3. Недашківський В. М. Хвороби бджіл та методи боротьби з ними. Київ : Урожай, 2014. 224 с.
4. Genersch E. et al. The German bee monitoring project: a long term study to understand periodically high winter losses of honey bee colonies. *Apidologie*. 2010. Vol. 41, No. 3. P. 332–352.
5. Arnia Remote Hive Monitoring. Official website. URL: https://www.arnia.co.uk (дата звернення: 10.03.2025).
6. BroodMinder Hive Technology. Official website. URL: https://broodminder.com (дата звернення: 10.03.2025).
7. OpenHive Open Source Beehive Monitoring. Official website. URL: https://www.openhive.ch (дата звернення: 12.03.2025).
8. Hivetool Open Source Hive Monitoring. Official website. URL: https://hivetool.org (дата звернення: 12.03.2025).
9. Zacepins A., Brusbardis V., Meitalovs J., Stalidzans E. Challenges in the development of Precision Beekeeping. *Biosystems Engineering*. 2015. Vol. 130. P. 60–71.
10. Edwards-Murphy F. et al. b+WSN: Smart beehive with preliminary decision tree analysis for agriculture and honey bee health monitoring. *Computers and Electronics in Agriculture*. 2016. Vol. 124. P. 211–219.
11. Мороз О. В. Методи та засоби побудови інформаційних систем моніторингу в агропромисловому комплексі. *Вісник Національного*

університету «Львівська політехніка». Серія: Інформаційні системи та мережі. 2021. № 9. С. 112–121.

12. Романюк О. Н., Павлов С. В. Мікроконтролерні системи збору та обробки інформації. Вінниця : ВНТУ, 2019. 187 с.

13. Atmel Corporation. ATxmega256A3U datasheet. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atxmega256A3U.pdf (дата звернення: 15.03.2025).

14. Espressif Systems. ESP8266 Technical Reference. Version 1.7. URL: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf (дата звернення: 15.03.2025).

15. Sensirion AG. SCD30 Sensor Module Datasheet. URL: https://sensirion.com/media/documents/4EAF6AF8/61652C3C/Sensirion_CO2_Sensors_SCD30_Datasheet.pdf (дата звернення: 16.03.2025).

16. Bosch Sensortec. BME280 Datasheet. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf (дата звернення: 16.03.2025).

17. Maxim Integrated. HX711 24-Bit Analog-to-Digital Converter Datasheet. URL: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf (дата звернення: 17.03.2025).

18. Maxim Integrated. MAX9814 Microphone Amplifier with AGC Datasheet. URL: https://datasheets.maximintegrated.com/en/ds/MAX9814.pdf (дата звернення: 17.03.2025).

19. STMicroelectronics. VL6180X Proximity and Ambient Light Sensing Module Datasheet. URL: https://www.st.com/resource/en/datasheet/vl6180x.pdf (дата звернення: 18.03.2025).

20. Іванченко В. О., Кузьменко І. М. Цифрова обробка сигналів у вбудованих системах. Харків : ХНУРЕ, 2020. 216 с.

21. Cooley J. W., Tukey J. W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*. 1965. Vol. 19, No. 90. P. 297–301.

22. IAR Systems / HP InfoTech. CodeVisionAVR User Manual. Version 3.x. URL: https://www.hpinfotech.ro/cvavr-manual.pdf (дата звернення: 20.03.2025).

23. Chan F. FatFs Generic FAT Filesystem Module. URL: http://elm-chan.org/fsw/ff/ (дата звернення: 20.03.2025).

24. Espressif Systems. ESP8266 RTOS SDK Programming Guide. URL: https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/ (дата звернення: 21.03.2025).

25. FreeRTOS. Real-time operating system for microcontrollers. URL: https://www.freertos.org (дата звернення: 21.03.2025).

26. lwIP. A lightweight TCP/IP stack. URL: https://savannah.nongnu.org/projects/lwip/ (дата звернення: 22.03.2025).

27. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : dissertation. University of California, Irvine, 2000. 162 p.

28. Mozilla Developer Network. Server-Sent Events. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 22.03.2025).

29. Chart.js. Simple yet flexible JavaScript charting library. URL: https://www.chartjs.org (дата звернення: 23.03.2025).

БІБЛІОГРАФІЧНА ДОВІДКА

Тема: Розроблення інформаційної системи моніторингу стану бджолиної колонії

Обсяг ПЗ складає 106 аркушів

Перелік креслень графічної частини:

- КРБ.СІ-11.00.00.000Е2 – Схема функціональна (аркушів 1);
- КРБ.СІ-11.00.00.001 – Алгоритм аналізу (аркушів 1);
- КРБ.СІ-11.00.00.002 – Вікно web-інтерфейсу (аркушів 1);
- КРБ.СІ-11.00.00.003 – Вікно web-інтерфейсу (аркушів 1);

Дата закінчення виконання бакалаврської роботи: _____

Студент-дипломник _____ Михайлюк В.А.

ДОДАТОК А

Лістинг програми мікроконтролера ATXMEGA256A3U

А.1 Файл main.h — визначення констант та структур

```
/* main.h — визначення констант, типів та прототипів функцій */
#ifndef MAIN_H
#define MAIN_H

#include <atxmega256a3u.h>
#include <delay.h>
#include <i2c.h>
#include <spi.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Адреси датчиків на шині I2C */
#define SCD30_ADDR    0x61
#define BME280_ADDR   0x76
#define VL6180X_ADDR  0x29

/* Інтервал вимірювань (секунди) */
#define MEAS_INTERVAL_SEC  300UL

/* Параметри FFT */
#define FFT_N        512
#define SAMPLE_RATE  3000

/* Коди діагностичних станів колонії */
#define COLONY_NORMAL     0
#define COLONY_WARN_TEMP  1
#define COLONY_WARN_HUM   2
#define COLONY_WARN_CO2   3
#define COLONY_SWARMING   4
#define COLONY_NO_QUEEN   5
#define COLONY_WEAK       6
#define COLONY_ALERT      7

/* Структура поточних вимірювань */
typedef struct {
    float        temperature;
    float        humidity;
    float        co2;
    float        pressure;
    float        weight_kg;
    unsigned int flight_cnt;
    unsigned int fft_dom_hz;
    unsigned char colony_state;
    char         diag_msg[33];
    unsigned long timestamp;
} meas_data_t;

/* Структура порогових значень */
typedef struct {
    float temp_min;
    float temp_max;
    float hum_min;
    float hum_max;
    float co2_max;
```

```
    float weight_drop_day;
    unsigned int flight_min;
} thresholds_t;

/* Прототипи функцій */
void system_init(void);
void adc_dma_init(void);
void scd30_start_continuous(unsigned int pressure_mbar);
unsigned char scd30_data_ready(void);
void scd30_read_measurement(float *co2, float *temp, float *hum);
long hx711_read_average(void);
float hx711_get_kg(long raw, long tare, float scale);
void fft_analyze(volatile unsigned int *samples, unsigned int
*dom_freq_hz);
unsigned char check_microclimate(meas_data_t *m);
unsigned char check_colony_state(meas_data_t *m, float weight_drop);
unsigned char diagnose_colony(meas_data_t *m);
void colony_state_to_str(unsigned char state, char *msg);
unsigned char sd_init(void);
unsigned char sd_write_record(meas_data_t *m);
void esp8266_send_data(meas_data_t *m);

#endif /* MAIN_H */
```

A.2 Файл system.c — ініціалізація системи

```
/* system.c — ініціалізація периферії ATXMEGA256A3U */
#include "main.h"

volatile unsigned int  adc_buffer[FFT_N];
volatile unsigned char adc_buffer_ready = 0;

void system_init(void)
{
    /* Тактова частота 32 МГц (внутрішній RC-генератор) */
    OSC.CTRL |= OSC_RC32MEN_bm;
    while (!(OSC.STATUS & OSC_RC32MRDY_bm));
    CCP = CCP_IOREG_gc;
    CLK.CTRL = CLK_SCLKSEL_RC32M_gc;

    /* SPI на PORTC — Master, MODE0, 8 МГц */
    PORTC.DIR |= PIN4_bm | PIN5_bm | PIN7_bm;
    PORTC.DIR &= ~PIN6_bm;
    PORTC.OUT  |= PIN4_bm;
    SPIC.CTRL = SPI_ENABLE_bm | SPI_MASTER_bm |
                SPI_MODE_0_gc | SPI_PRESCALER_DIV4_gc;

    /* TWI (I2C) на PORTD, 400 кГц */
    PORTD.DIR |= PIN0_bm | PIN1_bm;
    TWID.MASTER.BAUD = (unsigned char)
        ((F_CPU / (2UL * 400000UL)) - 5);
    TWID.MASTER.CTRLA = TWI_MASTER_ENABLE_bm;

    /* АЦП ADCA — 12-біт, внутрішня опора 1.0 В */
    ADCA.CTRLB    = ADC_RESOLUTION_12BIT_gc;
    ADCA.REFCTRL  = ADC_REFSEL_INT1V_gc | ADC_BANDGAP_bm;
    ADCA.PRESCALER= ADC_PRESCALER_DIV32_gc;
    ADCA.CTRLA    = ADC_ENABLE_bm;

    /* Таймер TCC0 — інтервал вимірювань ~1 с */
    TCC0.PER      = 31250 - 1;
    TCC0.CTRLA    = TC_CLKSEL_DIV1024_gc;
```

```
    TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

    /* Дозвіл переривань */
    PMIC.CTRL = PMIC_LOLVLEN_bm |
                PMIC_MEDLVLEN_bm |
                PMIC_HILVLEN_bm;
    sei();
}

/* DMA для накопичення відліків АЦП (3 кГц, 512 точок) */
void adc_dma_init(void)
{
    TCD0.PER   = 1333 - 1;
    TCD0.CTRLA = TC_CLKSEL_DIV8_gc;

    ADCA.CH0.CTRL    = ADC_CH_INPUTMODE_SINGLEENDED_gc;
    ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;

    DMA.CH0.SRCADDR0  = (unsigned char)
        ((unsigned int)(&ADCA.CH0.RES) >> 0);
    DMA.CH0.SRCADDR1  = (unsigned char)
        ((unsigned int)(&ADCA.CH0.RES) >> 8);
    DMA.CH0.DESTADDR0 = (unsigned char)
        ((unsigned int)(adc_buffer) >> 0);
    DMA.CH0.DESTADDR1 = (unsigned char)
        ((unsigned int)(adc_buffer) >> 8);
    DMA.CH0.TRFCNT    = FFT_N * 2;
    DMA.CH0.ADDRCTRL  = DMA_CH_SRCRELOAD_BURST_gc  |
                        DMA_CH_DESTINC_bm          |
                        DMA_CH_DESTRELOAD_TRANSACTION_gc;
    DMA.CH0.TRIGSRC   = DMA_CH_TRIGSRC_ADCA_CH0_gc;
    DMA.CH0.INTCTRL   = DMA_CH_TRNINTLVL_MED_gc;
    DMA.CH0.CTRLA     = DMA_CH_ENABLE_bm      |
                        DMA_CH_SINGLE_bm      |
                        DMA_CH_BURSTLEN_2BYTE_gc;
    DMA.CTRL          = DMA_ENABLE_bm;
}

interrupt [DMA_CH0_vect] void dma_ch0_isr(void)
{
    adc_buffer_ready = 1;
    DMA.CH0.CTRLB |= DMA_CH_TRNIF_bm;
}
```

A.3 Файл sensors.c — читання датчиків

```
/* sensors.c — функції зчитування датчиків */
#include "main.h"

/* ── SCD30 (CO2, T°, вологість) ── */
unsigned char scd30_crc8(unsigned char *data,
                         unsigned char len)
{
    unsigned char crc = 0xFF, i, j;
    for (i = 0; i < len; i++) {
        crc ^= data[i];
        for (j = 0; j < 8; j++)
            crc = (crc & 0x80) ? (crc<<1)^0x31 : (crc<<1);
    }
    return crc;
}
```

```
void scd30_start_continuous(unsigned int pressure_mbar)
{
    unsigned char arg[2];
    arg[0] = (unsigned char)(pressure_mbar >> 8);
    arg[1] = (unsigned char)(pressure_mbar & 0xFF);
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x00);
    i2c_write(0x00); i2c_write(0x10);
    i2c_write(arg[0]); i2c_write(arg[1]);
    i2c_write(scd30_crc8(arg, 2));
    i2c_stop();
}

unsigned char scd30_data_ready(void)
{
    unsigned char buf[3];
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x00);
    i2c_write(0x02); i2c_write(0x02);
    i2c_stop();
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x01);
    buf[0] = i2c_read(1);
    buf[1] = i2c_read(1);
    buf[2] = i2c_read(0);
    i2c_stop();
    return buf[1];
}

void scd30_read_measurement(float *co2,
                            float *temp,
                            float *hum)
{
    unsigned char raw[18]; unsigned char i;
    unsigned long tmp;
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x00);
    i2c_write(0x03); i2c_write(0x00);
    i2c_stop();
    i2c_start();
    i2c_write((SCD30_ADDR << 1) | 0x01);
    for (i = 0; i < 17; i++) raw[i] = i2c_read(1);
    raw[17] = i2c_read(0);
    i2c_stop();
    tmp = ((unsigned long)raw[0]<<24) |
          ((unsigned long)raw[1]<<16) |
          ((unsigned long)raw[3]<< 8) | raw[4];
    memcpy(co2, &tmp, 4);
    tmp = ((unsigned long)raw[6]<<24) |
          ((unsigned long)raw[7]<<16) |
          ((unsigned long)raw[9]<< 8) | raw[10];
    memcpy(temp, &tmp, 4);
    tmp = ((unsigned long)raw[12]<<24) |
          ((unsigned long)raw[13]<<16) |
          ((unsigned long)raw[15]<< 8) | raw[16];
    memcpy(hum, &tmp, 4);
}

/* ── HX711 (вага) ── */
long hx711_read(void)
{
    long data = 0; unsigned char i;
    while (PORTB.IN & PIN1_bm);
    for (i = 0; i < 24; i++) {
        PORTB.OUT |=  PIN0_bm; delay_us(1);
```

```
        data = (data << 1) |
               ((PORTB.IN & PIN1_bm) ? 1L : 0L);
        PORTB.OUT &= ~PIN0_bm; delay_us(1);
    }
    PORTB.OUT |= PIN0_bm; delay_us(1);
    PORTB.OUT &= ~PIN0_bm; delay_us(1);
    if (data & 0x800000L) data |= 0xFF000000L;
    return data;
}

long hx711_read_average(void)
{
    long readings[8], sum=0, mn, mx;
    unsigned char i;
    for (i=0; i<8; i++) readings[i] = hx711_read();
    mn = mx = readings[0];
    for (i=0; i<8; i++) {
        sum += readings[i];
        if (readings[i] < mn) mn = readings[i];
        if (readings[i] > mx) mx = readings[i];
    }
    return (sum - mn - mx) / 6;
}

float hx711_get_kg(long raw, long tare, float scale)
{
    return (float)(raw - tare) / scale / 1000.0f;
}
```

A.4 Файл diagnostics.c — алгоритм діагностики

```
/* diagnostics.c — трирівневий алгоритм діагностики */
#include "main.h"

int fft_re[FFT_N];
int fft_im[FFT_N];

static unsigned char prev_state = COLONY_NORMAL;

thresholds_t thresholds = {
    32.0f, 38.0f,   /* temp_min, temp_max */
    40.0f, 85.0f,   /* hum_min,  hum_max  */
    5000.0f,        /* co2_max             */
    2.0f,           /* weight_drop_day     */
    10              /* flight_min          */
};

/* Рівень 1: перевірка мікроклімату */
unsigned char check_microclimate(meas_data_t *m)
{
    if (m->temperature < thresholds.temp_min ||
        m->temperature > thresholds.temp_max)
        return COLONY_ALERT;
    if (m->humidity < thresholds.hum_min ||
        m->humidity > thresholds.hum_max)
        return COLONY_WARN_HUM;
    if (m->co2 > thresholds.co2_max)
        return COLONY_WARN_CO2;
    return COLONY_NORMAL;
}

/* Рівень 2: FFT-аналіз акустичного сигналу */
```

```
void fft_apply_window(volatile unsigned int *samples,
                      int *re, int *im)
{
    unsigned int i;
    for (i = 0; i < FFT_N; i++) {
        long w = 32768L - hann_table[i];
        re[i] = (int)(((long)((int)samples[i]-2048)*w)>>15);
        im[i] = 0;
    }
}

void fft_analyze(volatile unsigned int *samples,
                 unsigned int *dom_freq_hz)
{
    unsigned int i, max_bin = 0;
    long max_power = 0, power;
    unsigned int bin_lo =
        (unsigned int)(150UL * FFT_N / SAMPLE_RATE);
    unsigned int bin_hi =
        (unsigned int)(1200UL * FFT_N / SAMPLE_RATE);
    fft_apply_window(samples, fft_re, fft_im);
    fft_radix2(fft_re, fft_im, FFT_N);
    for (i = bin_lo; i <= bin_hi && i < FFT_N/2; i++) {
        power = (long)fft_re[i]*fft_re[i]
              + (long)fft_im[i]*fft_im[i];
        if (power > max_power) {
            max_power = power; max_bin = i;
        }
    }
    *dom_freq_hz = (unsigned int)
        ((unsigned long)max_bin * SAMPLE_RATE / FFT_N);
}

/* Рівень 3: комплексна діагностика */
#define DAY_BUF 288
float  weight_day_buf[DAY_BUF];
unsigned int weight_day_idx = 0;
unsigned int weight_day_cnt = 0;

void weight_day_push(float w)
{
    weight_day_buf[weight_day_idx] = w;
    weight_day_idx = (weight_day_idx+1) % DAY_BUF;
    if (weight_day_cnt < DAY_BUF) weight_day_cnt++;
}

float weight_drop_per_day(float current)
{
    float day_ago, drop;
    if (weight_day_cnt < DAY_BUF) return 0.0f;
    day_ago = weight_day_buf[weight_day_idx];
    drop = day_ago - current;
    return drop > 0.0f ? drop : 0.0f;
}

unsigned char check_colony_state(meas_data_t *m,
                                 float weight_drop)
{
    if (m->fft_dom_hz >= 300 && m->fft_dom_hz <= 500 &&
        m->temperature > 35.5f && m->co2 > 2000.0f)
        return COLONY_SWARMING;
    if (m->fft_dom_hz > 500 && weight_drop > 0.5f &&
        m->flight_cnt < thresholds.flight_min)
        return COLONY_NO_QUEEN;
    if (weight_drop > thresholds.weight_drop_day &&
```

```
        m->flight_cnt < thresholds.flight_min)
        return COLONY_WEAK;
    return COLONY_NORMAL;
}

char diag_msg[96];

unsigned char diagnose_colony(meas_data_t *m)
{
    unsigned char state;
    float drop;
    state = check_microclimate(m);
    if (state != COLONY_NORMAL) {
        colony_state_to_str(state, diag_msg);
        return state;
    }
    adc_dma_init();
    while (!adc_buffer_ready);
    adc_buffer_ready = 0;
    fft_analyze(adc_buffer, &m->fft_dom_hz);
    weight_day_push(m->weight_kg);
    drop = weight_drop_per_day(m->weight_kg);
    state = check_colony_state(m, drop);
    colony_state_to_str(state, diag_msg);
    return state;
}

void colony_state_to_str(unsigned char state, char *msg)
{
    switch (state) {
    case COLONY_NORMAL:
        strcpy(msg,"НОРМА:Колонія у нормальному стані");
        break;
    case COLONY_ALERT:
        strcpy(msg,"ТРИВОГА:Критичне відхилення T");
        break;
    case COLONY_WARN_HUM:
        strcpy(msg,"УВАГА:Вологість за межами норми");
        break;
    case COLONY_WARN_CO2:
        strcpy(msg,"УВАГА:CO2 перевищує норму");
        break;
    case COLONY_SWARMING:
        strcpy(msg,"РІЙ:Ознаки ройового настрою");
        break;
    case COLONY_NO_QUEEN:
        strcpy(msg,"БЕЗ МАТКИ:Перевірте наявність матки");
        break;
    case COLONY_WEAK:
        strcpy(msg,"ОСЛАБЛЕННЯ:Колонія втрачає масу");
        break;
    default:
        strcpy(msg,"НЕВІДОМО:Недостатньо даних");
    }
}
```

A.5 Файл main.c — головний цикл програми

```
/* main.c — головний цикл програми */
#include "main.h"
#include "ff.h"

meas_data_t current_meas;
meas_data_t prev_meas;
long  tare_value   = 0L;
float scale_factor = 420.0f;

#define EMA_ALPHA 0.2f

float ema_filter(float nv, float pv)
{
    return EMA_ALPHA*nv + (1.0f-EMA_ALPHA)*pv;
}

void validate_and_filter(meas_data_t *m,
                         meas_data_t *prev)
{
    if (m->temperature>=-20.0f && m->temperature<=60.0f)
        m->temperature=ema_filter(m->temperature,
                                  prev->temperature);
    else m->temperature = prev->temperature;
    if (m->humidity>=0.0f && m->humidity<=100.0f)
        m->humidity=ema_filter(m->humidity,prev->humidity);
    else m->humidity = prev->humidity;
    if (m->co2>=400.0f && m->co2<=10000.0f)
        m->co2=ema_filter(m->co2, prev->co2);
    else m->co2 = prev->co2;
    if (m->pressure>=850.0f && m->pressure<=1100.0f)
        m->pressure=ema_filter(m->pressure,prev->pressure);
    else m->pressure = prev->pressure;
    if (m->weight_kg>=5.0f && m->weight_kg<=80.0f)
        m->weight_kg=ema_filter(m->weight_kg,
                                prev->weight_kg);
    else m->weight_kg = prev->weight_kg;
}

void main(void)
{
    system_init();
    adc_dma_init();
    sd_init();
    esp8266_init();
    prev_meas.pressure = bme280_read_pressure();
    scd30_start_continuous(
        (unsigned int)prev_meas.pressure);
    delay_ms(2000);

    while (1) {
        /* 1. Датчики мікроклімату SCD30 */
        if (scd30_data_ready())
            scd30_read_measurement(
                &current_meas.co2,
                &current_meas.temperature,
                &current_meas.humidity);
        /* 2. Атмосферний тиск BME280 */
        current_meas.pressure = bme280_read_pressure();
        /* 3. Маса вулика HX711 */
        current_meas.weight_kg = hx711_get_kg(
            hx711_read_average(),
            tare_value, scale_factor);
```

```
        /* 4. Льотна активність VL6180X */
        current_meas.flight_cnt = vl6180x_get_count();
        /* 5. Фільтрація даних */
        validate_and_filter(&current_meas, &prev_meas);
        /* 6. Діагностика стану колонії */
        current_meas.colony_state =
            diagnose_colony(&current_meas);
        /* 7. Запис на microSD */
        sd_write_record(&current_meas);
        /* 8. Передача до ESP8266 через SPI */
        esp8266_send_data(&current_meas);
        /* 9. Збереження поточних як попередніх */
        memcpy(&prev_meas, &current_meas,
               sizeof(meas_data_t));
        /* 10. Режим очікування */
        sleep_until_next_cycle(MEAS_INTERVAL_SEC);
    }
}
```

ДОДАТОК Б

Лістинг програми модуля ESP8266

Б.1 Файл main_esp.h — визначення та структури

```
/* main_esp.h — визначення для програми ESP8266 */
#ifndef MAIN_ESP_H
#define MAIN_ESP_H

#include "esp_common.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "lwip/sockets.h"
#include "espressif/esp_softap.h"
#include <string.h>
#include <stdio.h>

#define AP_SSID      "BeeMonitor_01"
#define AP_PASSWORD  "beehive2024"
#define AP_CHANNEL   6
#define HTTP_PORT    80
#define JSON_BUF_SIZE 512

/* Пакет даних від ATXMEGA (64 байти) */
typedef struct __attribute__((packed)) {
    uint8_t  header[2];
    float    temperature;
    float    humidity;
    float    co2;
    float    pressure;
    float    weight_kg;
    uint16_t flight_cnt;
    uint16_t fft_dom_hz;
    uint8_t  colony_state;
    char     diag_msg[33];
    uint32_t timestamp;
    uint8_t  crc8;
} spi_packet_t;

/* Прототипи функцій */
void wifi_init(void);
void http_server_task(void *pvParameters);
void spi_rx_task(void *pvParameters);
void handle_request(int fd, char *req);
void handle_sse(int fd);
void send_json(int fd, char *json);
void send_gzip_page(int fd,
                    const uint8_t *data,
                    uint32_t len);
void build_json_current(spi_packet_t *p, char *buf);
void build_json_status(char *buf);
void sse_send_state(unsigned char state, char *msg);

#endif /* MAIN_ESP_H */
```

Б.2 Файл wifi.c — ініціалізація Wi-Fi

```
/* wifi.c — налаштування Wi-Fi точки доступу */
#include "main_esp.h"

void wifi_init(void)
{
    wifi_set_opmode(SOFTAP_MODE);
    struct softap_config ap_cfg;
    memset(&ap_cfg, 0, sizeof(ap_cfg));
    strcpy((char*)ap_cfg.ssid,     AP_SSID);
    strcpy((char*)ap_cfg.password, AP_PASSWORD);
    ap_cfg.ssid_len      = strlen(AP_SSID);
    ap_cfg.channel       = AP_CHANNEL;
    ap_cfg.authmode      = AUTH_WPA2_PSK;
    ap_cfg.max_connection = 4;
    wifi_softap_set_config(&ap_cfg);
}
```

Б.3 Файл spi_rx.c — прийом даних від ATXMEGA

```
/* spi_rx.c — прийом SPI-пакету від ATXMEGA */
#include "main_esp.h"

spi_packet_t rx_packet;
char json_current[JSON_BUF_SIZE];
volatile uint8_t json_ready = 0;
static unsigned char prev_colony_state = 0;

void build_json_current(spi_packet_t *p, char *buf)
{
    snprintf(buf, JSON_BUF_SIZE,
        "{"
        "\"temp\":%.1f,"
        "\"hum\":%.1f,"
        "\"co2\":%.0f,"
        "\"pres\":%.1f,"
        "\"weight\":%.2f,"
        "\"flight\":%u,"
        "\"fft_hz\":%u,"
        "\"state\":%u,"
        "\"msg\":\"%s\","
        "\"ts\":%lu"
        "}",
        p->temperature, p->humidity,
        p->co2, p->pressure,
        p->weight_kg, p->flight_cnt,
        p->fft_dom_hz, p->colony_state,
        p->diag_msg,  p->timestamp);
}

void process_new_packet(void)
{
    build_json_current(&rx_packet, json_current);
    json_ready = 1;
    if (rx_packet.colony_state != prev_colony_state) {
        sse_send_state(rx_packet.colony_state,
                       rx_packet.diag_msg);
        prev_colony_state = rx_packet.colony_state;
    }
}
```

```
void spi_rx_task(void *pvParameters)
{
    uint8_t rx_buf[sizeof(spi_packet_t)];
    uint8_t crc;
    spi_slave_init(HSPI);
    while (1) {
        spi_slave_recv(HSPI, rx_buf,
                       sizeof(spi_packet_t));
        if (rx_buf[0]==0xBE && rx_buf[1]==0xE1) {
            crc = crc8_calc(rx_buf,
                            sizeof(spi_packet_t)-1);
            if (crc == rx_buf[sizeof(spi_packet_t)-1]){
                memcpy(&rx_packet, rx_buf,
                       sizeof(spi_packet_t));
                process_new_packet();
            }
        }
        vTaskDelay(10 / portTICK_RATE_MS);
    }
}
```

Б.4 Файл http_server.c — HTTP-сервер та маршрутизація

```
/* http_server.c — HTTP-сервер на lwIP sockets */
#include "main_esp.h"

extern const uint8_t  index_html_gz[];
extern const uint32_t index_html_gz_len;

int sse_clients[2] = {-1, -1};

void send_json(int fd, char *json)
{
    char hdr[160]; int jlen = strlen(json);
    snprintf(hdr, sizeof(hdr),
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: application/json\r\n"
        "Content-Length: %d\r\n"
        "Access-Control-Allow-Origin: *\r\n"
        "Connection: close\r\n\r\n", jlen);
    send(fd, hdr,  strlen(hdr), 0);
    send(fd, json, jlen,        0);
}

void send_gzip_page(int fd,
                    const uint8_t *data,
                    uint32_t len)
{
    char hdr[200];
    snprintf(hdr, sizeof(hdr),
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: text/html\r\n"
        "Content-Encoding: gzip\r\n"
        "Content-Length: %lu\r\n"
        "Cache-Control: max-age=3600\r\n"
        "Connection: close\r\n\r\n",
        (unsigned long)len);
    send(fd, hdr,  strlen(hdr), 0);
    send(fd, data, len,         0);
}
```

```
void build_json_status(char *buf)
{
    uint32_t uptime = xTaskGetTickCount() /
                      portTICK_RATE_MS / 1000;
    int8_t rssi = wifi_station_get_rssi();
    snprintf(buf, 128,
        "{\"uptime\":%lu,"
         "\"rssi\":%d,"
         "\"heap\":%u}",
        uptime, rssi,
        system_get_free_heap_size());
}

void handle_sse(int fd)
{
    const char *hdr =
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: text/event-stream\r\n"
        "Cache-Control: no-cache\r\n"
        "Connection: keep-alive\r\n\r\n";
    send(fd, hdr, strlen(hdr), 0);
    int i;
    for (i=0; i<2; i++)
        if (sse_clients[i]<0) {
            sse_clients[i]=fd; break;
        }
}

void sse_send_state(unsigned char state, char *msg)
{
    char buf[160]; int i, len;
    len = snprintf(buf, sizeof(buf),
        "event: colony\r\n"
        "data: {\"state\":%u,"
               "\"msg\":\"%s\"}\r\n\r\n",
        state, msg);
    for (i=0; i<2; i++)
        if (sse_clients[i]>=0) {
            int r = send(sse_clients[i],buf,len,0);
            if (r<=0) sse_clients[i]=-1;
        }
}

void handle_request(int fd, char *req)
{
    if (strstr(req,"GET / ") ||
        strstr(req,"GET /index.html")) {
        send_gzip_page(fd, index_html_gz,
                       index_html_gz_len);
        return;
    }
    if (strstr(req,"GET /api/current")) {
        send_json(fd, json_current); return;
    }
    if (strstr(req,"GET /api/events")) {
        handle_sse(fd); return;
    }
    if (strstr(req,"GET /api/status")) {
        char s[128]; build_json_status(s);
        send_json(fd, s); return;
    }
    if (strstr(req,"POST /api/settings")) {
        handle_settings(fd, req); return;
    }
```

```
    const char *e404 =
        "HTTP/1.1 404 Not Found\r\n"
        "Content-Length: 9\r\n\r\n"
        "Not Found";
    send(fd, e404, strlen(e404), 0);
}

void http_server_task(void *pvParameters)
{
    int srv, cli;
    struct sockaddr_in sa, ca;
    socklen_t clen = sizeof(ca);
    char rx[256]; int rlen;
    srv = socket(AF_INET, SOCK_STREAM, 0);
    memset(&sa,0,sizeof(sa));
    sa.sin_family      = AF_INET;
    sa.sin_addr.s_addr = INADDR_ANY;
    sa.sin_port        = htons(HTTP_PORT);
    bind(srv,(struct sockaddr*)&sa,sizeof(sa));
    listen(srv, 4);
    while (1) {
        cli = accept(srv,
                     (struct sockaddr*)&ca,&clen);
        if (cli < 0) continue;
        rlen = recv(cli, rx, sizeof(rx)-1, 0);
        if (rlen > 0) {
            rx[rlen] = '\0';
            handle_request(cli, rx);
        }
        close(cli);
        vTaskDelay(5 / portTICK_RATE_MS);
    }
}
```

Б.5 Файл user_init.c — точка входу ESP8266

```
/* user_init.c — ініціалізація та запуск задач */
#include "main_esp.h"

void user_init(void)
{
    wifi_init();
    strcpy(json_current,
        "{\"state\":0,"
        "\"msg\":\"Ініціалізація...\"}");
    xTaskCreate(spi_rx_task,
                "spi_rx",
                512,  NULL, 3, NULL);
    xTaskCreate(http_server_task,
                "http_srv",
                1024, NULL, 2, NULL);
}
```