

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЕНЕРГОЕФЕКТИВНОЇ

БЕЗДРОТОВОЇ МЕРЕЖІ

SOFTWARE FOR THE ENERGY EFFICIENT

WIRELESS NETWORK

Виконала: студентка 4 курсу, групи КНТ-129

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЯЦЕНКО А.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЯЦЕНКО Анастасії Костянтинівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення енергоефективної бездротової мережі. Software for the Energy Efficient Wireless Network.

керівник проєкту (роботи) к.т.н., доцент, ПАРХОМЕНКО
Анжеліка Володимирівна

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

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз особливостей реалізації бездротових сенсорних мереж. 2. Дослідження методів та засобів мінімізації енергоспоживання бездротових сенсорних мереж. 3. Проєктування програмного забезпечення бездротової сенсорної мережі. 4. Конструювання програмного забезпечення бездротової сенсорної мережі БСМ. 5. Розробка та тестування прототипу БСМ.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання «04» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	1 тиждень	Розділ 1
2	Дослідження методів вирішення проблеми	2 тиждень	Розділ 2
3	Проектування ПЗ	3 тиждень	Розділ 3
4	Конструювання ПЗ	4-5 тижні	Розділ 4
5	Тестування та експериментальне дослідження програмного забезпечення.	6 тиждень	Розділ 5
6	Оформлення пояснювальної записки та документів до неї.	7 тиждень	ПЗ, Додатки
7	Нормоконтроль та рецензування.	7 тиждень	
8	Захист роботи.	8 тиждень	

Студент(ка)

_____ Анастасія ЯЦЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
131 с., 14 табл., 47 рис., 2 дод., 36 джерел.

БЕЗДРОТОВІ СЕНСОРНІ МЕРЕЖІ, ІНТЕРНЕТ РЕЧЕЙ, НЕЧІТКА
ЛОГІКА, МІКРОКОНТРОЛЕР.

Об'єкт дослідження – організація функціонування бездротової сенсорної мережі (БСМ) локальної зони покриття.

Предмет дослідження – методи, алгоритми та програмні засоби організації енергоефективної роботи вузлів БСМ.

Мета роботи – розробка програмного забезпечення БСМ на основі удосконаленого алгоритму збору та передачі даних для мінімізації енергоспоживання системи.

Матеріали, методи та технічні засоби: структурне програмування, мова програмування C, середовище програмування Arduino IDE, модулі ESP-32, плати Devkit з модулем ESP-32, персональний комп'ютер з процесором Intel Core i5-3330 під управлінням операційної системи Microsoft Windows 7.

Результати. Розроблено програмне забезпечення прототипу бездротової сенсорної мережі на основі удосконаленого алгоритму збору та передачі даних з використанням нечіткої логіки.

Висновки. Впровадження розробленого програмного забезпечення дозволить знизити енергоспоживання вузлів бездротової сенсорної мережі, що забезпечить більш тривалий термін її функціонування без сервісного обслуговування, а також заощадити на акумуляторах.

Галузь використання – системи типу «Розумний будинок» та «Розумна будівля».

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 131 pages, 14 tables, 47 figures, 2 appendixes, 36 sources.

WIRELESS SENSOR NETWORKS, INTERNET OF THINGS, Fuzzy LOGIC, MICROCONTROLLER.

Object of study is the organization of the functioning of the wireless sensor network (WSN) of the local coverage area.

Subject of study are methods, algorithms and software tools for organizing energy-efficient operation of WSN nodes.

The purpose of the work is to develop WSN software based on an improved data collection and transmission algorithm to minimize system energy consumption.

Materials, methods, and tools: structured programming, ESP-32 modules, Devkit boards with the ESP-32 module, C programming languages, development environment Arduino IDE, a personal computer with an Intel Core i5-3330 processor under the control of the Microsoft Windows 7 operating system.

Results. The software of a wireless sensor network prototype was developed based on an improved algorithm of data collection and transmission using fuzzy logic.

Conclusions. The implementation of the developed software will reduce the energy consumption of wireless sensor network nodes, which will ensure a longer period of its operation without maintenance, as well as save on batteries.

The field of use are "Smart House" and "Smart Building" type systems.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз особливостей реалізації бездротових сенсорних мереж	10
1.1 Поняття БСМ	10
1.2 Різновиди БСМ	10
1.3 Сфери застосування БСМ	13
1.4 Проблематика реалізації БСМ	20
1.5 Огляд архітектур БСМ	23
1.6 Протоколи зв'язку БСМ	27
1.7 Висновки до розділу 1	30
2 Дослідження методів та засобів мінімізації енергоспоживання бездротових сенсорних мереж	32
2.1 Огляд існуючих рішень	32
2.2 Аналіз виявлених методів мінімізації енергоспоживання	33
2.3 Огляд алгоритмів роботи вузлів-сенсорів	34
2.4 Постановка завдань дипломної роботи	36
3 Проєктування програмного забезпечення бездротової сенсорної мережі	38
3.1 Аналіз вимог	38
3.2 Розробка архітектури мережі	39
3.3 Вибір програмно-апаратної платформи	40
3.4 Вибір мови програмування	42
3.5 Вибір середовища розробки	43
3.6 Висновки до розділу 3	44
4 Конструювання програмного забезпечення бездротової сенсорної мережі	45
4.1 Розробка алгоритму роботи БСМ	45
4.2 Контролер нечіткої логіки	50
4.3 Структура ПЗ	
4.4 Опис особливостей програмної реалізації	56

4.5 Висновки до розділу 4.....	67
5 Розробка та тестування прототипу БСМ	68
5.1 Опис особливостей реалізації прототипу	68
5.2 Тестування прототипу.....	72
5.3 Порівняння енергоефективності модуля при використанні звичайної затримки і режиму deep_sleep	85
5.4 Приблизна оцінка енергоефективності розробленого рішення	91
5.5 Висновки до розділу 5.....	92
Висновки.....	93
Перелік джерел посилань	95
Додаток А Текст програми	99
Додаток Б Слайди презентації.....	123

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BLE	– Bluetooth Low Energy;
CAN	– Controller Area Network;
COM	– Communications Port;
CSS	– Chirp Spread Spectrum;
FTSP	– The Flooding Time Synchronization Protocol;
GPIO	– General-Purpose Input/Output;
I2C	– Inter-Integrated Circuit;
I2S	– Inter-IC Sound;
IoT	– Internet of Things;
LAN	– Local Area Network;
LPWAN	– Low-power Wide-area Network;
MAC	– Media Access Control;
RBS	– Reference Broadcast Synchronization;
RMS	– Root Mean Square;
SPI	– Serial Peripheral Interface;
SRAM	– Static Random Access Memory;
TPSN	– Timing-Sync Protocol for Sensor Networks;
UART	– Universal Asynchronous Receiver-Transmitter;
WLAN	– Wireless Local Area Network;
WPAN	– Wireless Personal Area Network;
БСМ	– бездротова сенсорна мережа;
ПЗ	– програмне забезпечення;
ЦП	– центральний процесор.

ВСТУП

На сьогоднішній день задача економії ресурсів, зокрема електроенергії, є надзвичайно актуальною. Сучасні технології Інтернету речей (Internet of Things, IoT) дозволяють досягати запланованих результатів шляхом впровадження систем Розумне місто, Розумне виробництво, Розумна будівля тощо [1]. Подібні технології передбачають підключення великої кількості датчиків для контролю показників оточуючого середовища та різноманітних процесів. Для злагодженої роботи великої кількості датчиків виникає необхідність об'єднувати їх в мережу. Однак, коли площі охоплюваних територій збільшуються, з'єднання компонентів мереж десятками й сотнями метрів дротів стає незручним. У зв'язку з цим особливої популярності набули бездротові сенсорні мережі (БСМ).

Для організації роботи БСМ необхідно не лише забезпечити живлення її компонентів від автономних джерел електроенергії, але й забезпечити достатню тривалість їх роботи. При цьому, багато датчиків необхідно встановлювати в місцях, де розміри компонента стають критичними та немає можливості підключити потужне, але велике за розміром джерело живлення. В той же час, при неефективному енергоспоживанні ці джерела доведеться постійно замінювати. Отже, розробка енергоефективної БСМ сьогодні є актуальним завданням, оскільки зі збільшенням кількості датчиків задача заощадження електроенергії стає питанням не тільки вартості, але й надійності роботи системи. Дослідження показали, що частиною робочого процесу БСМ, під час якого витрачається найбільше електроенергії, є обмін даними [2]. Цю проблему можна вирішити шляхом удосконалення комунікаційних алгоритмів, які будуть коригувати поведінку вузлів БСМ в залежності від ситуації. Ще однією проблемою БСМ є суттєве споживання електроенергії власне вузлами, що потребує організації енергоефективних режимів їх роботи.

Мета роботи – розробка ПЗ БСМ на основі удосконаленого алгоритму збору та передачі даних для мінімізації енергоспоживання системи.

1 АНАЛІЗ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖ

1.1 Поняття БСМ

БСМ призначені для зручної організації контролю за характеристиками навколишнього середовища та різноманітних процесів. Бездротовий зв'язок дозволяє налаштувати обмін даними, уникаючи прокладання дротів, що робить систему компактнішою та більш зручною для установки та використання.

Завдяки розвитку в області бездротових технологій та енергоефективних пристроїв стало можливим розробляти малопотужні та недорогі багатофункціональні сенсорні вузли [3].

БСМ – це мережа, що складається з автономних вузлів, обладнаних сенсорами, і розгорнута у середовищі з можливостями бездротового зв'язку [4].

1.2 Різновиди БСМ

БСМ зазвичай мають структуру, специфічну для вирішуваної практичної задачі. Схема класифікації БСМ наведена на рис. 1.1.

Класифікувати БСМ можна за такими категоріями [5]:

- за рівнем детермінованості: детерміновані або недетерміновані. У кібернетиці, детерміновані системи – це системи, в яких між входами та виходами є визначені співвідношення та відсутня ймовірнісна складова [6]. У детермінованих БСМ положення кожного вузла визначається заздалегідь та не змінюється в процесі роботи. Для більшості реальних застосувань, подібне жорстке визначення позиції вузла не є можливим через умови середовища чи використання;

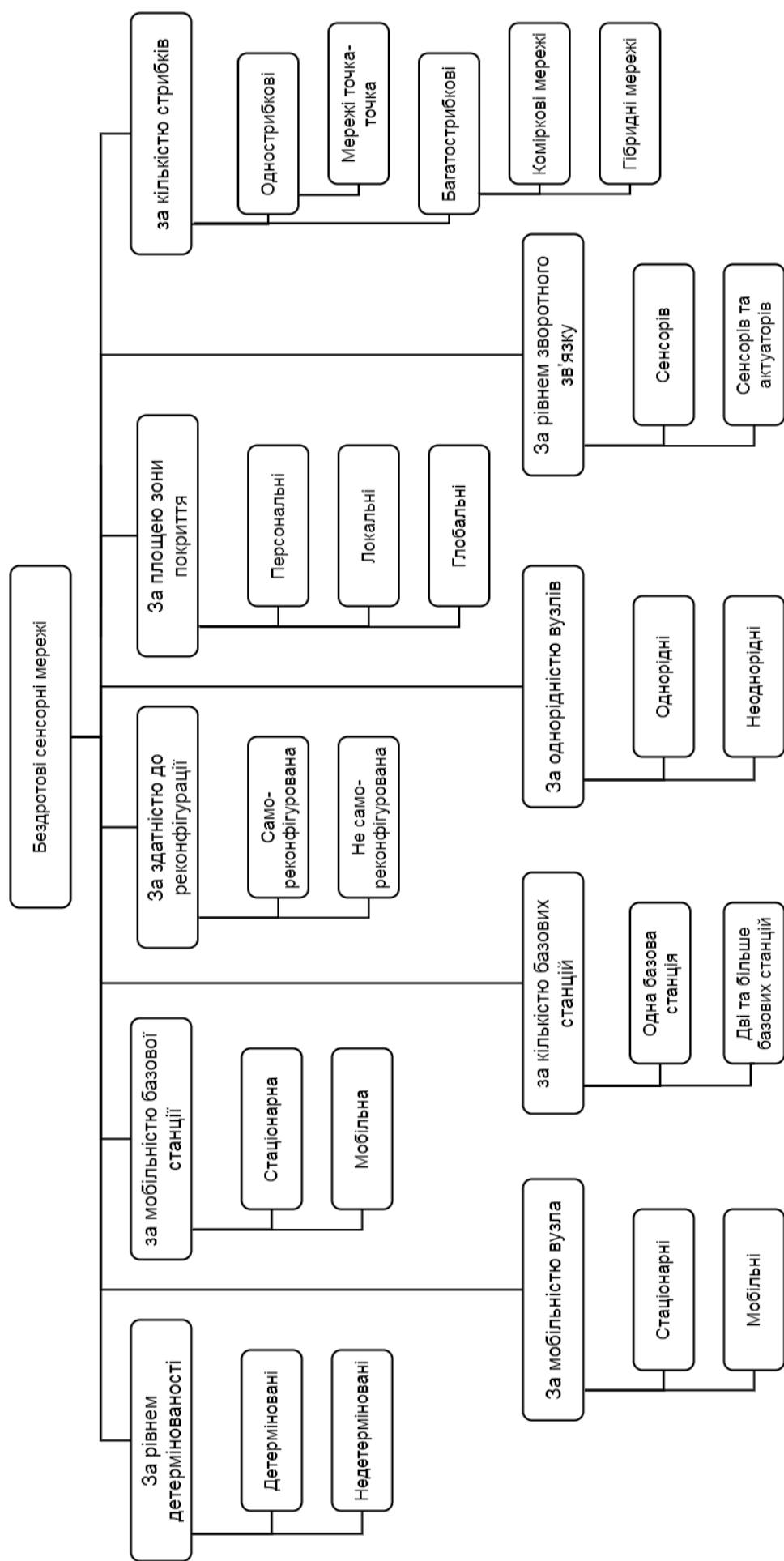


Рисунок 1.1 – Класифікація БСМ

- за мобільністю вузла: стаціонарний або мобільний. У мобільних БСМ вузли можуть бути переміщені у просторі. Наприклад, на базі таких мереж можуть бути реалізовані системи медичного моніторингу стану людини, або системи моніторингу поведінки тварин. У стаціонарних БСМ вузли зафіксовані на одному положенні;

- за мобільністю базової станції: стаціонарна або мобільна. Мобільність базової станції аналогічна до мобільності вузлів. Мобільна базова станція дозволяє збирати дані з кожного вузла в зоні сприйняття шляхом переміщення ближче до вузла;

- за кількістю базових станцій: одна або декілька базових станцій. В залежності від вирішуваної задачі, кількість базових станцій може бути більше однієї на мережу. В такому випадку, вузли надсилають дані на найближчу базову станцію;

- за кількістю стрибків: одно- або багатострибкові. У БСМ з одним стрибком вузли датчиків безпосередньо підключені до базової станції. Автори [7] відносять до однострибкових БСМ мережі точка-точка, а до багатострибкових – коміркові мережі. Автори [8] пропонують ідею мережі i-Bean, що є гібридною зірково-комірковою мережею;

- за здатністю до реконфігурації: самореконфігурована чи ні. Здатність до реконфігурації полягає в можливості вузлів організовуватися у мережу, не полягаючи на зовнішній блок керування. Більшість сучасних БСМ мають здатність до реконфігурації;

- за однорідністю вузлів: однорідні або неоднорідні. До однорідності вузлів БСМ відносять споживання енергії, обчислювальну потужність та можливості зберігання даних. Якщо вимоги до зазначених характеристик різні у деяких вузлів, то мережа є гетерогенною;

- за площею зони покриття: персональні, локальні, глобальні. Для вирішення різних практичних задач потребується різна площа зони покриття БСМ. Основною відмінністю між різними за площею БСМ є використовуваний

протокол зв'язку. Таким чином, БСМ можна поділити на: персональні (WPAN, до 10 метрів між вузлами), локальні (LAN, WLAN, 10-100 метрів між вузлами) та глобальні (LPWAN, декілька кілометрів між вузлами);

– за рівнем зворотного зв'язку: мережі лише сенсорів або сенсорів і актуаторів. Розрізняють мережі, до складу яких входять лише сенсори, та мережі, що складаються одночасно і з сенсорів, і з актуаторів. До перших відносяться системи моніторингу показників вимірюваного середовища. На основі мереж сенсорів та актуаторів можна реалізовувати, наприклад, системи типу «Розумний будинок», де окрім вимірювання показників середовища також буде реалізовано зворотню взаємодію – як, наприклад, увімкнення обігрівачів чи зволожувачів повітря.

1.3 Сфери застосування БСМ

БСМ використовуються у багатьох сферах людської діяльності (рис. 1.2).



Рисунок 1.2 – Сфери застосування БСМ

Системи моніторингу стану споруд. БСМ набули широкого використання у системах моніторингу стану споруд, таких, як будівлі, дамби, мости та інших [9]. Використання БСМ дозволяє здійснювати діагностику загального стану та пошкоджень споруд у режимі реального часу. Окрім того, БСМ можуть бути розгорнуті у важкодоступних або небезпечних для людини місцях, покриваючи усі необхідні зони споруди для повного моніторингу.

Дані, зібрані датчиками, можуть використовуватися для різних цілей, таких як виявлення та локалізація пошкоджень, прогнозування терміну служби конструкції, що залишився, і раннє попередження про потенційні відмови. БСМ також можна використовувати для моніторингу реакції конструкції на умови довкілля, такі як вітер, температура та вологість.

БСМ мають кілька переваг у порівнянні з традиційними дротовими системами моніторингу, такі як зниження витрат на встановлення та обслуговування, масштабованість та гнучкість.

Отже, БСМ є цінним інструментом для моніторингу стану споруд, забезпечуючи діагностику у реальному часі, збір та аналіз даних. Вони можуть допомогти виявити потенційні проблеми та вжити превентивних заходів, що зрештою підвищить безпеку та надійність конструкцій.

Системи типу «Розумний будинок» та «Розумна будівля». БСМ можна використовувати для створення систем розумного будинку [10]. У розумному будинку використовуються різні датчики для контролю параметрів, а дані, зібрані з цих датчиків, аналізуються, щоб надати корисну інформацію та покращити якість життя мешканців.

Одним із найпоширеніших застосувань БСМ у розумних будинках є автоматизація. Розміщуючи датчики в різних місцях приміщень, мережа може виявляти, серед іншого, зміни температури, освітленості, вологості та присутності людей. Потім цю інформацію можна використовувати для автоматичного налаштування систем опалення та охолодження, увімкнення та вимкнення світла.

БСМ також можна використовувати в якості систем безпеки. Датчики можуть бути розміщені на дверях та вікнах для виявлення несанкціонованого

проникнення, а датчики руху можуть виявляти рух усередині приміщень. Дані, зібрані з цих датчиків, можуть бути відправлені на центральний блок управління, який може попередити власника або співробітників служби безпеки у разі виявлення вторгнення.

Також такі мережі можна використовувати для керування споживанням енергії в розумних будинках. Використовуючи датчики для виявлення змін температури, освітленості та зайнятості, мережа може автоматично регулювати системи опалення та охолодження, вимикати світло та прилади, коли вони не використовуються, і навіть перемикатися на відновлювані джерела енергії, такі як сонячні панелі або вітряні турбіни.

БСМ також можна інтегрувати з іншими технологіями розумного будинку, такими як голосові помічники, розумні замки та домашні розважальні системи, для створення повністю автоматизованого житлового середовища.

Окрім квартир та житлових будинків, подібні системи можна розгортати у офісах та будівлях багатоквартирних будинків для моніторингу показників середовища.

В цілому, БСМ пропонують ряд переваг для додатків розумного будинку, включаючи зручність, енергоефективність та підвищену безпеку.

Індустріальна автоматизація. БСМ все частіше використовуються в промисловій автоматизації [7] для підвищення ефективності, зниження витрат та оптимізації процесів. БСМ розподілені по великій території для збору даних про різні параметри, такі як температура, тиск, вологість та вібрація.

Використання БСМ у промисловості дає можливість контролювати стан обладнання у режимі реального часу. Таким чином, можна заздалегідь виявляти потенційні проблеми та поламки, та вчасно приймати міри з налагодження або заміни устаткування.

Окрім того, БСМ дозволяють контролювати якість продукції, що виробляється, а також матеріалів, що використовуються для виробництва. Виміри розмірів, ваги, кольору та хімічного складу продукції дозволить чітко визначати їх якість та знизити кількість браку.

Також важливим аспектом використання БСМ дозволить створити безпечний простір для роботи людей. Контроль над показниками робочого середовища дозволить знизити шкідливість умов роботи, а також знизить ризик нещасних випадків.

Отже, БСМ забезпечують економічний та дієвий спосіб збирання даних, моніторингу процесів та підвищення ефективності та безпеки у промислових середовищах.

Застосування у медичній галузі. БСМ для медичних систем – це нова технологія, що дозволяє віддалено відстежувати фізіологічні параметри та параметри довкілля пацієнтів у режимі реального часу [7].

У медичних БСМ датчики зазвичай прикріплюються до тіла пацієнта або розміщуються у навколишньому середовищі для моніторингу показників життєдіяльності, таких як частота серцевих скорочень, артеріальний тиск, температура тіла та насичення киснем. Потім ці датчики можуть передавати зібрані дані бездротової мережі на центральний комп'ютер або пристрій, які можуть використовуватися медичними працівниками для моніторингу та аналізу стану пацієнта.

Медичні БСМ також можна використовувати для відстеження активності та переміщень пацієнтів, що може допомогти медичним працівникам зрозуміти поведінку пацієнтів та забезпечити персоналізований догляд. Крім того, БСМ можна використовувати у віддалених районах, де доступ до медичної допомоги обмежений, а дані, зібрані датчиками, можуть передаватися медичним працівникам, які знаходяться в іншому місці, для діагностики та лікування.

БСМ мають кілька переваг у медичних системах. Вони можуть забезпечити безперервний моніторинг пацієнтів, знизити потребу в госпіталізації, забезпечити раннє виявлення захворювань та покращити результати лікування пацієнтів. Крім того, вони неінвазивні, що знижує ризик зараження, а їхнє бездротове підключення дозволяє здійснювати віддалений моніторинг та аналіз даних у режимі реального часу, що може заощадити час та скоротити витрати на охорону здоров'я.

Однак при впровадженні БСМ у медичні системи також необхідно враховувати деякі проблеми, такі як необхідність надійного бездротового зв'язку, забезпечення конфіденційності та безпеки даних, а також керування великим обсягом даних, що генеруються датчиками. Тим не менш, БСМ можуть підвищити ефективність системи охорони здоров'я та покращити результати лікування пацієнтів, забезпечуючи моніторинг у реальному часі та персоналізований догляд.

Контроль на дорогах [9]. У БСМ для управління дорожнім рухом датчики зазвичай розміщуються на дорогах, світлофорах та іншій інфраструктурі для збору даних, таких як швидкість, щільність та напрямок руху транспортних засобів. Потім ці датчики можуть передавати зібрані дані на центральний комп'ютер або пристрій, які можуть використовуватися співробітниками служби керування дорожнім рухом для моніторингу та керування транспортним потоком.

Однією з основних переваг використання БСМ в управлінні дорожнім рухом є те, що вони можуть надавати дані щодо умов дорожнього руху в режимі реального часу, що дозволяє посадовим особам приймати обґрунтовані рішення про те, як оптимізувати транспортний потік та зменшити затори. Наприклад, якщо датчики виявляють інтенсивний рух на певній дорозі, співробітники служби дорожнього руху можуть налаштувати сигнали світлофора, щоб змінити пріоритет цієї дороги та зменшити затори.

Крім того, БСМ можна використовувати для виявлення аварій, перекриттів доріг та інших подій, які можуть вплинути на транспортний потік, що дозволяє владі швидко реагувати та перенаправляти трафік за потреби. БСМ також можна використовувати для збирання даних про шаблони трафіку з часом, які можна використовувати для планування трафіку та розвитку інфраструктури.

Однак при впровадженні БСМ у системи керування дорожнім рухом необхідно враховувати деякі проблеми, такі як забезпечення надійного бездротового зв'язку, керування великим обсягом даних, що генеруються датчиками, а також забезпечення конфіденційності та безпеки даних.

В цілому, БСМ мають значний потенціал для покращення систем управління дорожнім рухом і допомагають зменшити затори, підвищити безпеку та підвищити загальну ефективність транспорту.

Контроль трубопроводів [9]. При моніторингу трубопроводу датчики БСМ зазвичай встановлюються в різних точках вздовж трубопроводу для збору даних про температуру, тиск і вібрацію. Потім ці дані можуть бути передані бездротовою мережею в центральну систему управління, яка може використовуватися операторами для контролю та управління станом трубопроводу, виявлення будь-яких аномалій і виконання завдань з технічного обслуговування.

Однією з ключових переваг використання БСМ для моніторингу трубопроводу є можливість виявлення проблем, перш ніж вони стануть серйозними. Збираючи та аналізуючи дані в режимі реального часу, БСМ можуть попереджати операторів трубопроводів про потенційні проблеми, такі як виток або корозія, до того, як вони призведуть до значних пошкоджень або загроз безпеки. Крім того, БСМ можуть надати цінну інформацію про продуктивність трубопроводу, яку можна використовувати для оптимізації операцій та зниження витрат на обслуговування.

При впровадженні БСМ для моніторингу трубопроводів необхідно враховувати такі проблеми, як необхідність надійного бездротового зв'язку, забезпечення точності та надійності даних та керування великим обсягом даних. Крім того, особливу увагу слід приділити суворим та віддаленим умовам, в яких часто працюють трубопроводи, що може вимагати застосування міцних та спеціалізованих датчиків.

Тим не менш, БСМ можуть значно підвищити безпеку та надійність трубопроводів, що робить їх важливим інструментом сучасного керування трубопроводами.

Сільське господарство [9]. Мережа бездротових датчиків для сільського господарства – це тип технології, в якій використовується мережа невеликих недорогих бездротових пристроїв для моніторингу та збору даних про різні

аспекти сільськогосподарських операцій, таких як вологість ґрунту, температура, вологість та зростання врожаю.

БСМ для сільського господарства можуть надавати фермерам дані та інформацію в режимі реального часу, щоб допомогти їм приймати більш обґрунтовані рішення щодо своїх культур. Наприклад, фермери можуть використовувати дані цих датчиків для оптимізації графіків зрошення, виявлення ранніх ознак хвороб або нашествия шкідників, а також для спостереження за здоров'ям та зростанням своїх культур. Це може призвести до більш ефективного використання ресурсів, підвищення врожайності та зниження витрат.

БСМ для сільського господарства пропонують кілька переваг у порівнянні з традиційними методами моніторингу сільськогосподарських культур, такими як ручний збір даних та системи провідних датчиків. Ці переваги включають:

- рентабельність: БСМ набагато рентабельніші, ніж традиційні методи моніторингу, оскільки їх легко розгорнути і вони вимагають мінімального обслуговування;
- моніторинг у режимі реального часу: БСМ забезпечують моніторинг різних факторів довкілля у режимі реального часу, що дозволяє фермерам приймати більш обґрунтовані рішення щодо зрошення, внесення добрив та боротьби зі шкідниками;
- підвищена точність: БСМ забезпечують точніші дані, ніж традиційні методи моніторингу, оскільки вони можуть збирати дані з кількох точок на полі і передавати їх у центральне місце для аналізу;
- підвищення ефективності: БСМ можуть допомогти фермерам оптимізувати свою діяльність, визначаючи області, що вимагають більшої чи меншої кількості зрошення чи добрив, що призводить до підвищення врожайності та зниження витрат;
- стійкий розвиток. Зменшуючи використання води та хімікатів, БСМ можуть допомогти фермерам знизити вплив на навколишнє середовище та просувати стійкі методи ведення сільського господарства.

В цілому, БСМ для сільського господарства пропонують багатообіцяючі варіанти вирішення проблем, з якими стикаються сучасні фермери, надаючи їм дані та ідеї, необхідні для прийняття обґрунтованих рішень та підвищення врожайності за одночасного зниження впливу на довкілля.

1.4 Проблематика реалізації БСМ

БСМ – це перспективний та багатообіцяючий напрямок. Але при їх реалізації розробники стикаються з рядом проблем, таких як: забезпечення енергоефективної роботи датчиків, безпека та конфіденційність, синхронізація часу. На рис. 1.3 наведено схему проблематики реалізації БСМ.

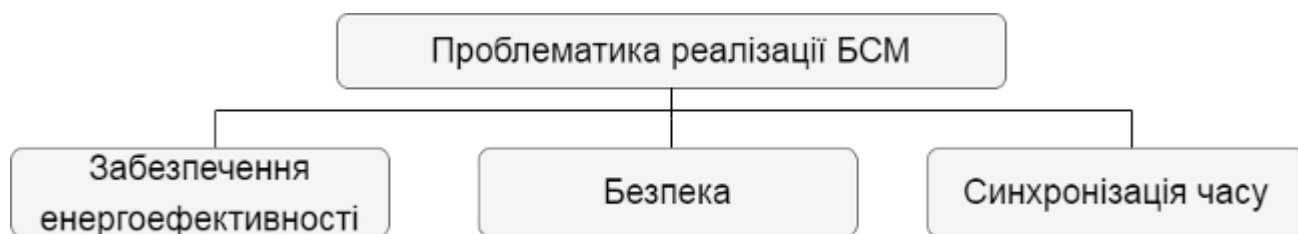


Рисунок 1.3 – Проблематика реалізації БСМ

Забезпечення енергоефективності. БСМ – важлива технологія для збору та передачі даних з великої кількості датчиків. Однак вони зазвичай працюють від автономних джерел енергії, і енергоефективність є критичним фактором для їхнього успішного розгортання. Серед стратегій зниження енерговитрат можна виділити наступні:

- обладнання з низьким енергоспоживанням: вибір апаратних компонентів з низьким енергоспоживанням для вузлів датчиків. Такі компоненти, як мікроконтролери, радіоприймачі та датчики, які споживають менше енергії, можуть подовжити термін служби автономних джерел енергії;
- сплячі режими: сенсорні вузли можуть бути запрограмовані на перехід в режим глибокого сну, коли немає даних для передачі, і періодично прокидатися для перевірки нових даних;

- протоколи маршрутизації: використання протоколів маршрутизації, що використовують багатовузловий зв'язок та чергуються між різними вузлами, може знизити енергоспоживання вузлів;
- робочий цикл: використання методу, при якому сенсорні вузли чергуються між активним та сплячим станами, щоб мінімізувати споживання енергії;
- локалізація: знаючи розташування вузлів, можна оптимізувати маршрутизацію та заощадити енергію за рахунок зменшення кількості переходів, необхідних для зв'язку.

Вирішення цих проблем потребує міждисциплінарного підходу, що включає досвід роботи в мережах, комунікаціях, обробці сигналів та інших суміжних галузях. Розробляючи інноваційні рішення цих проблем, можна проектувати та впроваджувати ефективні та дієві БСМ для різних програм.

Безпека. БСМ використовуються в широкому спектрі програм, таких як моніторинг навколишнього середовища, охорона здоров'я, військова сфера, розумні будинки. Проте через їх розподілений характер, обмежені ресурси та вразливість бездротового зв'язку, вони дуже чутливі для різних атак.

Загрози безпеки у БСМ можна поділити на три категорії: фізичні атаки, атаки перехоплення даних [11] та активні атаки [11]. Фізичні атаки пов'язані з фізичним пошкодженням вузлів або мережевої інфраструктури, наприклад, зломом або знищенням датчиків, порушенням енергопостачання або фізичною крадіжкою вузлів. Атаки перехоплення даних включають прослуховування мережного трафіку, моніторинг мережевої активності, а також перехоплення та аналіз даних, що передаються по мережі. Активні атаки включають зміну, вставку або видалення даних у мережі, створення підроблених вузлів і порушення зв'язку в мережі.

Для захисту від цих загроз можуть бути реалізовані різні заходи безпеки на різних рівнях, таких як фізичний, мережевий та прикладний рівень. Фізично для захисту вузлів від фізичних атак можуть бути реалізовані такі заходи, як захищений від несанкціонованого доступу корпус, безпечне електропостачання

та моніторинг навколишнього середовища. На мережевому рівні можуть бути розгорнуті механізми шифрування, аутентифікації та контролю доступу для запобігання несанкціонованому доступу та захисту цілісності та конфіденційності даних. На прикладному рівні можуть використовуватися безпечні протоколи та алгоритми для забезпечення автентичності, цілісності та конфіденційності даних.

На додаток до цих заходів для виявлення атак та реагування на них можна використовувати системи виявлення вторгнень та реагування. Ці системи можуть бути реалізовані на різних рівнях, таких як рівень вузла, рівень кластера або рівень мережі, і можуть використовувати різні методи, такі як виявлення аномалій, аналіз поведінки та виявлення на основі сигнатур.

Насамкінець, безпека БСМ є критичною проблемою, яка потребує комплексної стратегії захисту від різних типів загроз. Комбінація превентивних заходів, систем виявлення та реагування на вторгнення, а також протоколів та алгоритмів безпеки може використовуватися для забезпечення безпеки та надійності БСМ у різних додатках.

Синхронізація часу. Синхронізація часу є важливою вимогою у БСМ, оскільки вона допомагає координувати зв'язок між сенсорними вузлами та гарантує, що дані, зібрані вузлами, мають точну часову мітку [9]. Однак досягнення синхронізації часу у БСМ є складним завданням через декілька факторів:

- апаратні обмеження. Вузли датчиків зазвичай мають обмежену обчислювальну потужність, пам'ять і час автономної роботи. Ці обмеження ускладнюють досягнення точної та ефективної синхронізації часу;

- затримки зв'язку. Затримки зв'язку між вузлами можуть призвести до відхилень у показаннях годинника, що ускладнює досягнення точної синхронізації часу. Це особливо актуально для великомасштабних мереж, де відстань між вузлами може бути значною;

- топологія мережі. Топологія мережі може також впливати на синхронізацію часу. У багатоскачкових мережах, де дані передаються через

кілька вузлів, дрейф годинника кожного вузла може накопичуватися, що призводить до значних помилок синхронізації;

- чинники довкілля. Фактори довкілля, такі як температура, вологість та електромагнітні перешкоди, можуть вплинути на роботу годинника та призвести до помилок синхронізації.

Для вирішення цих проблем у літературі було запропоновано декілька механізмів синхронізації часу, таких як еталонна ширококомовна синхронізація (RBS) [12], протокол синхронізації часу для сенсорних мереж (TPSN) [9] та протокол синхронізації часу повені (FTSP) [9]. Ці механізми використовують різні методи, такі як еталонні ширококомовні передачі, оцінку дрейфу годинника та алгоритми розподіленого консенсусу для досягнення синхронізації часу в БСМ.

Таким чином, досягнення синхронізації часу в БСМ є складним завданням через різні фактори, такі як апаратні обмеження, затримки зв'язку, топологія мережі та фактори навколишнього середовища. Однак при використанні відповідних протоколів синхронізації часу ці проблеми можна подолати, забезпечивши точну та ефективну координацію між вузлами мережі.

1.5 Огляд архітектур БСМ

Аналіз типів БСМ (п. 1.2) дозволив виділити три типи архітектури БСМ: мережі точка-точка, коміркові та гібридні мережі.

Мережі точка-точка. БСМ точка-точка (також їх називають деревоподібними або зіркоподібними) є одним з типів БСМ, де кожен вузол має безпосереднє бездротове з'єднання тільки з одним іншим вузлом.

Прикладом БСМ точка-точка може бути пара бездротових пристроїв, які використовують Bluetooth або Wi-Fi для передачі безпосередньо між собою. У цьому випадку пристрій-джерело надсилає дані безпосередньо пристрою-призначенню, без необхідності проходити через проміжні вузли.

БСМ точка-точка можуть бути корисні в тих випадках, коли потрібно встановити бездротове пряме з'єднання між двома пристроями без необхідності

використовувати проміжні вузли. Вони також можуть бути корисними, коли потрібна висока швидкість передачі даних, оскільки кожен пристрій може передавати дані безпосередньо один одному.

Однак, БСМ точка-точка можуть бути обмежені дальністю передачі та може бути складно забезпечити покриття у великих зонах. Вони також можуть бути менш стійкими до відмов, ніж більш складні мережі, такі як коміркові.

На рис. 1.4 та 1.5 наведено спрощену схему мережі точка-точка та схему деревоподібної мережі точка-точка.

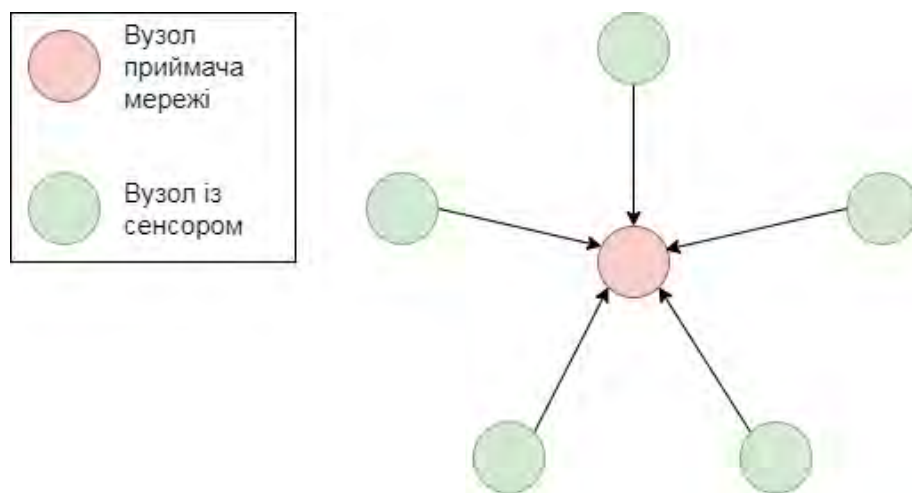


Рисунок 1.4 – Спрощена схема мережі точка-точка

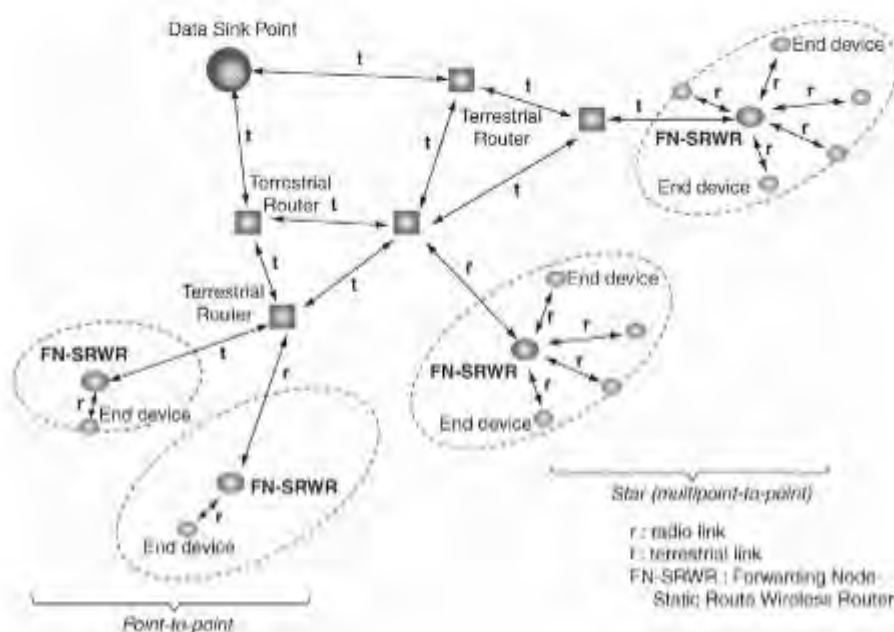


Рисунок 1.5 – Схема деревоподібної мережі точка-точка [7]

Коміркові мережі. Коміркові мережі в БСМ являють собою мережі, в яких вузли можуть працювати як ретранслятори передачі даних між іншими вузлами. Це дозволяє забезпечити більшу надійність та стійкість мережі, оскільки дані можуть бути доставлені через кілька маршрутів та не залежать від одного вузла. Такі мережі також можуть бути більш гнучкими та адаптивними, оскільки нові вузли можуть бути додані або видалені з мережі без припинення її роботи.

Завдяки своїй гнучкості та надійності, коміркові мережі стають все більш популярними в таких областях, як розумний будинок, розумне місто та промисловість.

Проте, коміркові мережі мають деякі недоліки. Наприклад, через підвищену складність, ці мережі можуть бути менш ефективними у використанні енергії, ніж традиційні. Крім того, такі мережі можуть мати більшу затримку передачі даних через необхідність маршрутизації через декілька вузлів.

Також ця архітектура мережі передбачає, що більшість пристроїв завжди будуть доступні для обміну даними. В цьому випадку буде неможливо організувати мінімальне енергоспоживання, оскільки офлайн-стан і глибокий сон більшості датчиків можуть порушити цілісність всієї системи збору даних.

На рис. 1.6 наведено архітектуру бездротової коміркової мережі.



Рисунок 1.6 – Архітектура коміркової БСМ [13]

Гібридні мережі. Автори [8] запропонували ідею гібридної мережі i-Bean, яка складається з трьох типів пристроїв:

- кінцеві точки (ендпоінти, або i-Beans) – пристрої, до яких підключені датчики;
- роутери (або ретранслятори) – відповідають за пересилання та маршрутизацію даних між кінцевими точками і базовими станціями;
- головна станція, або центральний шлюз.

Кінцеві точки з'єднані з сусідніми роутерами за допомогою топології «точка-точка», тоді як роутери з'єднані між собою за допомогою коміркової топології. Кінцеві точки ніколи не ретранслюють пакети для інших вузлів, а роутери можуть спілкуватися з будь-яким іншим вузлом поблизу, і вони виконують ретрансляцію та маршрутизацію пакетів для інших роутерів і кінцевих точок.

Такий тип мережі дозволяє налаштувати ефективні алгоритми обміну даними для логічних пар кінцева точка-маршрутизатор і маршрутизатор-шлюз. Ця структура також надає можливість підключення датчика до різних маршрутизаторів, що забезпечить надійність системи у випадках, коли один маршрутизатор буде недоступний.

Схему гібридної мережі i-Bean наведено на рис. 1.7.



Рисунок 1.7 – Гібридна мережа i-Bean (E - кінцева точка, R - роутер, G - центральний шлюз) [8]

Результати порівняння характеристик цих трьох моделей наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика моделей архітектури мереж [7], [8], [13]

Архітектура Властивість	Мережа точка-точка	Коміркові мережі	Гібридні мережі
Можливість передавати дані через різні вузли	Відсутня	Присутня	Присутня
Можливість переводити точки у глибокий сон	Відсутня	Відсутня	Присутня

1.6 Протоколи зв'язку БСМ

Аналіз типів БСМ (п. 1.2) дозволив виділити три типи мереж за зоною покриття: персональні, локальні та глобальні БСМ. На рис. 1.8 наведено протоколи для цих типів мереж.



Рисунок 1.8 – Протоколи зв'язку для БСМ

Персональні БСМ. У персональних БСМ в якості протоколу зв'язку можна використовувати Bluetooth та UWB.

Технологія Bluetooth схожа на технологію Wi-Fi, але сила сигналу набагато слабша, тож передача даних можлива на малій відстані між пристроями [14]. Крім того, через те, що при використанні технології Bluetooth сигнал проходить з одного пристрою на інший і не відправляє дані на окремий сервер, це

дозволяє споживати менше енергії і, таким чином, покращує роботу акумуляторів і термін їх використання [15].

UWB, або ультраширокий діапазон – це радіотехнологія, яка може використовувати дуже низький рівень енергії для зв'язку на короткій відстані з надширокою смугою пропускання у значній частині радіоспектру [16].

Діапазон частот UWB становить від 3,1 до 10,6 ГГц. На даний момент в Україні не дозволено використання даних частот для цивільних потреб.

Таблиця 1.2. – Порівняльна характеристика протоколів зв'язку персональних БСМ [14]-[16]

Протокол Характеристика	Bluetooth	UWB
Енергоспоживання	Середнє	Низьке
Швидкість передачі даних	Середня (до 3 Мб/с)	Дуже висока (до 500Мб/с)
Потреба у спеціалізованому модулі	Ні	Так (чіп U1)
Вартість	Низька	Висока

Локальні БСМ. Для локальних БСМ можливе використання таких протоколів, як Wi-Fi, ESP-NOW та ZigBee.

Wi-Fi – це технологія бездротової локальної мережі з пристроями на основі стандартів IEEE 802.11, що дозволяє передавати дані на відстані до 100 м на швидкості від 0.1 Мб/с до 600 Мб/с (802.11n) [17].

ZigBee – це бездротова технологія та протокол зв'язку високого рівня, специфікація, яка базується на стандарті IEEE 802.15.4. Це енергоефективний протокол, що підтримує велику кількість вузлів у мережі та використовується для створення персональних мереж з малими цифровими радіостанціями малої потужності [18]. Однак у протоколу ZigBee також є ряд недоліків, серед яких низька швидкість передачі даних (40-250 Кб/с) та висока вартість впровадження.

Низька швидкість передачі суттєво впливає на енергоефективність при обміні великими пакетами даних.

ESP-NOW – це протокол, розроблений Espressif, який дозволяє кільком пристроям спілкуватися один з одним на відстані до 100 м. Протокол подібний до малопотужного бездротового зв'язку 2,4 ГГц. ESP-NOW підтримує зашифрований і незашифрований одноадресний зв'язок і обмін невеликими повідомленнями (до 250 байт) між платами ESP32 [19].

Обмеження однорангових пристроїв у ESP-NOW:

- максимум 10 зашифрованих вузлів у станційному режимі, не більше 6 у режимі SoftAP або SoftAP + Station;
- менше 20 незашифрованих вузлів.

Споживання струму приблизно однакове для протоколів ESP-NOW, Wi-Fi, але розробники ESP-NOW стверджують, що ця технологія є зв'язком без з'єднання, що означає, що для цього протоколу не потрібен час узгодження з'єднання.

Таблиця 1.3 – Порівняльна характеристика протоколів зв'язку локальних БСМ [17]-[19]

Протокол	ESP-NOW	Wi-Fi	ZigBee
Характеристика			
Енергоспоживання	Низьке	Середнє	Низька
Швидкість передачі даних	Середня (до 1 Мб/с)	Висока (до 600 Мб/с)	Низька (до 250 кб/с)
Потреба у спеціалізованому модулі	Так (модулі на базі ESP-8266 та ESP32)	Ні	Так (модем ZigBee)
Вартість	Низька	Низька	Висока

Глобальні БСМ. Для глобальних БСМ може бути використано такі протоколи зв'язку, як LoRa.

LoRa – це техніка бездротової модуляції, яка походить від технології Chirp Spread Spectrum (CSS). Вона кодує інформацію на радіохвилях, використовуючи

імпульси чіпкання (chirp). Модульована передача LoRa стійка до перешкод і може прийматися на великих відстанях [20]. Для контролю перевантаження LoRa використовує фактори поширення – загалом 6 факторів від SF7 до SF12. Фактори поширення впливають на швидкість передачі даних, час перебування в ефірі, час автономної роботи та чутливість приймача. Енерговитрати при передачі повідомлень дуже сильно залежать від використовуваного фактору та розміру повідомлення [21]. Таким чином, найбільш енергоефективним буде режим роботи з передачею малих пакетів даних, оскільки це буде потребувати менше часу. Основною особливістю протоколу LoRa є вкрай низькі швидкості передачі даних – 0,3-50 Кб/с, але й найбільша відстань зв'язку (LONg RANge).

1.7 Висновки до розділу 1

У даному розділі було розглянуто поняття БСМ, їх різновиди та сфери застосування. Було розглянуто проблематику реалізації БСМ та методи вирішення цих проблем. Окрім того, було розглянуто архітектури мереж та протоколи зв'язку.

Одним з перспективних напрямків застосування БСМ є розробка систем типу «Розумний будинок» та «Розумна будівля», які дозволяють покращити якість життя людей за рахунок систем моніторингу показників навколишнього середовища.

Зазвичай у подібних системах відсутня необхідність змінювати місцезнаходження датчиків, так як виміри проводяться в одних і тих самих місцях. Окрім того, відстань між найближчими вузлами мережі з малою вірогідністю буде перевищувати 100 метрів, так як це обмежується розмірами самих приміщень. Отже, для реалізації подібних систем підійдуть стаціонарні БСМ локальної зони покриття.

На основі проведеного дослідження, гібридна архітектура може бути рекомендована для використання, оскільки вона дозволяє мінімізувати енергоспоживання мережі за рахунок використання сплячого режиму кінцевих

точок, а також передбачає можливість підключення кінцевої точки до різних роутерів. Проведений огляд протоколів локальної групи показав, що оптимальним вибором буде ESP-NOW, так як це енергоефективний протокол зв'язку, модулі з підтримкою якого мають доступну вартість.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ МІНІМІЗАЦІЇ ЕНЕРГОСПОЖИВАННЯ БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖ

2.1 Огляд існуючих рішень

Як показали проведені дослідження, запропоновано декілька підходів для зменшення енергоспоживання БСМ.

Так, у [2] розглянуто можливості використання теорії ігор для налаштування взаємодії між вузлами-сенсорами. Автори розглядають топологію мережі, де проміжні вузли виконують також функцію ретрансляції пакетів від інших вузлів. Однак, через обмеженість енергетичного ресурсу вузлів, спостерігається так звана егоїстична поведінка, коли вузли резервують свою енергію для досягнення довшого терміну служби, і відправляють дані одразу до приймача даних. Через це вузли, розташовані далеко від приймача, витрачають свою енергію раніше вузлів, близьких до приймача.

Використання теорії ігор дозволяє налаштувати енергоефективні протоколи передачі даних та контроль топології мережі. Окрім того, це дозволяє використовувати технологію збору енергії – заряджати вузли або мережу шляхом збору відновлюваної енергії з джерел навколишнього середовища, таких як сонце, вітер, вібрація тощо.

Автори [22] запропонували розподілений підхід до агрегації даних на основі нечітких чисел і середньозважених операторів, щоб зменшити передачу даних у БСМ, коли цікавить оцінка агрегованого значення, такого як максимальна або мінімальна температура, виміряна в мережі. Основу запропонованого підходу складає те, що кожен вузол підтримує оцінку сукупної вартості. Грунтуючись на цій оцінці, вузол вирішує, чи потрібно передавати по мережі нове значення, виміряне датчиком на борту вузла або отримане через повідомлення.

Архітектура мережі i-Beans [8], описана у п.1.5, передбачає розділення ролей вузлів БСМ на кінцеві точки (вузли з сенсорами), роутери та центральний шлюз (приймач даних). Ця архітектура є гібридом архітектур точка-точка та

коміркової мережі, що дозволяє ефективно використовувати переваги обох архітектур.

Дана архітектура передбачає можливість підключення кінцевої точки до декількох роутерів, що підвищує відмовостійкість системи. Таким чином, якщо один з роутерів вийде з ладу, точка зможе передавати дані на центральний шлюз через інший.

Використовуючи цю гібридну топологію, енергоспоживання кінцевих точок мінімізується, а мережа є відмовостійкою, оскільки між кінцевими точками та шлюзом може існувати кілька шляхів.

У [23] в якості інструментів зменшення енергоспоживання мережі запропоноване інтервальна робота вузлів-сенсорів, мінімізація обсягу даних для передачі та скорочення відстані передачі.

Інтервальна робота зменшить надлишкове вимірювання шляхом чергування активного та неактивного стану вузлів. Під час переходу до активного стану виникають додаткові витрати енергії, але вони досить малі відносно загального споживання енергії. Для зменшення споживання енергії у процесі передачі даних авторами було запропоновано зв'язок із кількома стрибками та ієрархії на основі кластеризації.

2.2 Аналіз виявлених методів мінімізації енергоспоживання

У п. 2.1 було розглянуто декілька підходів до зниження енергоспоживання мережі, серед яких: використання теорії ігор, використання розподіленого підходу до агрегації даних, розподіл ролей вузлів у мережі на кінцеві точки, роутери та центральний шлюз, а також інтервальна робота вузлів та мінімізація обсягу даних для передачі та скорочення відстані передачі.

Агрегація даних є ефективним інструментом, коли цікавить саме агреговане значення з мережі. Проте, якщо цікавлять саме окремі значення з сенсорів, то цей підхід не буде гарантувати ефективну роботу БСМ.

Запропонована авторами [8] архітектура дозволить налаштувати як оптимальну маршрутизацію, так і енергоефективні алгоритми роботи вузлів за рахунок розділу ролей. Так, вузли-сенсори не будуть вирішувати задачу маршрутизації пакетів від інших вузлів, що дозволить знизити їх енергоспоживання та продовжити термін їх роботи.

Запропоновані [23] методи зниження енергоспоживання БСМ можуть бути застосовані разом з іншими моделями енергоефективних мереж для оптимізації внутрішніх процесів. Однак, інтервальна робота сенсорів можлива в тому випадку, коли вузли-сенсори не виконують функцію маршрутизації даних, що унеможливило б використання чистих коміркових мереж. Окрім того, при використанні чергування активного та неактивного стану вузлів, ускладнюється реалізація методів, що передбачують динамічні знання точки про мережу, як при використанні теорії ігор.

Гібридність розглянутої мережі i-Beans дозволить використовувати інтервальну роботу для кінцевих точок, що значно продовжить термін їх служби. Окрім того, у такій мережі все ще можливо використовувати складніші методи виявлення оптимального маршруту, так як вузли-роутери з'єднані між собою за допомогою коміркової топології.

Отже, на основі проведеного аналізу можна зробити висновок про ефективність використання в якості основних методів мінімізації енергоспоживання підхід мережі i-Beans та інтервальної роботи вузлів.

2.3 Огляд алгоритмів роботи вузлів-сенсорів

Було розглянуто алгоритми, що можуть забезпечити енергоефективну роботу вузлів до відправки даних: дерева рішень, нечітка логіка та генетичні алгоритми.

Дерева рішень. Дерева рішень групують або класифікують випадки, розташовуючи їх у дереві, від кореня до листових вузлів. Це забезпечує характеристику або класифікацію подій. Випадок класифікується, починаючи з

кореневого вузла дерева, перевіряючи ознаки, вказані цим вузлом, а потім просуваючись вниз по гілці дерева, порівнюючи випадок зі значенням атрибута. Процес повторюється для піддерева, яке корениться в новому вузлі [24].

Таким чином, дерева рішень прості та прозорі для розуміння, але є ризик створення надскладного дерева, яке втратить здатність узагальнювати дані.

Генетичні алгоритми. Генетичні алгоритми – це стохастичні методи пошуку, що працюють з популяцією індивідуумів, де кожен індивідуум кодує в собі варіант рішення. На кожній ітерації алгоритму формується нова популяція: для кожного індивідуума обчислюється цільова функція, проходить відбір найкращих за значенням функцій і до них застосовуються операції схрещення та/або мутації [25].

Оскільки генетичні алгоритми – це методи випадкової вибірки, то ефективність їх роботи повністю залежить від якості операцій схрещення та мутації. В іншому випадку, розроблений алгоритм буде не краще за чисто випадковий підхід, або настільки обмеженим, що буде витратити багато часу на надлишкові рішення.

Нечітка логіка. У [26] запропоновано алгоритми збору даних на основі нечіткої логіки. Завдяки евристичним алгоритмам нечітка логіка уникає складних математичних моделей, але зберігає здатність працювати з конфліктними ситуаціями та нелінійністю даних. Нечітка логіка дозволяє розробити систему прийняття рішень на основі лінгвістичних правил [27].

Процес нечіткої обробки складається з наступних кроків [28]:

- визначення проблеми;
- фаззифікація ;
- застосування нечітких правил;
- дефаззифікація ;
- інтерпретація результатів.

Результати порівняння характеристик описаних підходів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика підходів для збору та передачі даних [24]-[28]

Підхід Характеристика	Дерева рішень	Генетичні алгоритми	Нечітка логіка
Складність математичної моделі	Середня	Низька	Низька
Легкість інтерпретації результатів	Середньо	Складно	Легко
Ризик надскладних ланцюжків рішення	Середній	Високий	Низький

Отже, для реалізації може бути рекомендовано підхід на основі нечіткої логіки, оскільки він дозволяє знайти рішення для конкретного випадку на основі правил, встановлених для подібних випадків.

2.4 Постановка завдань дипломної роботи

Проведені дослідження довели актуальність розробки БСМ, для яких важливим є завдання мінімізації енергоспоживання. Вчені пропонують декілька різних підходів, більшість з яких базується на оптимізації маршрутів та процесу передачі даних. Однак, дані методи потребують подальшого удосконалення та врахування особливостей окремих систем. Тому, тема роботи є актуальною.

Було зроблено огляд існуючих рішень для мінімізації енергоспоживання БСМ: використання теорії ігор, агрегація даних, архітектура i-Beans з розподілом ролей вузлів у мережі на кінцеві точки, роутери та центральний шлюз, а також інтервальна робота вузлів та мінімізація обсягу даних для передачі та скорочення відстані передачі. Результати дослідження наведено в табл. 2.2.

Було зроблено висновок про ефективність використання архітектури мережі i-Beans у поєднанні з інтервальною роботою вузлів, так як це дозволить енергоефективно налаштувати роботу як мережі в цілому, так і вузлів окремо.

Таблиця 2.2 - Задачі мінімізації енергоспоживання та методи їх вирішення

Задача	Метод вирішення
Оптимізація передачі даних	Використання теорії ігор [2]
	Використання агрегації даних [22]
	Впровадження гібридної архітектури [8]
	Зменшення відстаней для зв'язку [23]
Оптимізація роботи вузлів	Використання інтервальної роботи вузлів [23]

Більшість існуючих рішень базуються на оптимізації передачі даних. Було розглянуто алгоритми, що можуть забезпечити енергоефективну роботу вузлів до відправки даних: дерева рішень, генетичні алгоритми та нечітка логіка.

Було зроблено висновок про ефективність використання нечіткої логіки в якості основи для алгоритму роботи кінцевих точок, так як вона дозволяє знайти рішення для конкретного випадку на основі правил, встановлених для подібних випадків, уникаючи при цьому складних розрахунків.

Отже, актуальною задачею є розробка БСМ, в якій буде поєднано гібридну архітектуру з алгоритмом роботи вузлів-сенсорів на базі нечіткої логіки.

Мета роботи – розробка програмного забезпечення БСМ на основі удосконаленого алгоритму збору та передачі даних для мінімізації енергоспоживання системи.

Отже, для досягнення поставленої мети необхідно розв'язати завдання:

- дослідити існуючі архітектури та протоколи БСМ;
- проаналізувати методи та засоби мінімізації енергоспоживання БСМ;
- виконати аналіз вимог та розробити архітектуру системи;
- обрати інструментарій розробки;
- виконати розробку програмного забезпечення;
- розробити прототип та провести тестування БСМ.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БЕЗДРОТОВОЇ СЕНСОРНОЇ МЕРЕЖІ

3.1 Аналіз вимог

У розділах 1 та 2 було проведено дослідження, яке дозволило сформулювати постановку завдання дипломної роботи. Було прийнято рішення розробки стаціонарної БСМ локальної зони покриття для систем типу «Розумний будинок» та «Розумна будівля».

Нефункціональні вимоги. Виходячи з постановки завдання дипломної роботи, можна виділити наступні нефункціональні вимоги до проєктованої мережі:

- БСМ повинна мати гібридну архітектуру точка-точка/коміркова;
- БСМ повинна використовувати протокол зв'язку ESP-NOW для обміну даними між вузлами мережі;
- вузли БСМ мають бути стаціонарними.

Функціональні вимоги. Виходячи з постановки завдання дипломної роботи, можна виділити наступні функціональні вимоги до проєктованої мережі:

- вузли-сенсори повинні вимірювати показники навколишнього середовища та власний стан (заряд автономного джерела живлення, заповненість пам'яті);
- вузли-сенсори повинні коригувати свою послідовність дій в залежності від результатів роботи контролера нечіткої логіки;
- вузли-сенсори повинні відправляти значення вимірів, якщо того вимагає отримане контролером нечіткої логіки рішення;
- вузли-сенсори повинні записувати у пам'ять значення вимірів, якщо відправлення неможливе з технічних причин або якщо того вимагає отримане контролером нечіткої логіки рішення;
- вузли-сенсори повинні відправляти виміри, збережені в пам'яті вузла, якщо того вимагає отримане контролером нечіткої логіки рішення;

- вузли-сенсори повинні працювати в режимі інтервальної роботи;
- вузли-роутери повинні відправляти вузлам-роутерам зі списку сусідів повідомлення-keepalive з певним визначеним інтервалом;
- вузли-роутери повинні оновлювати статус роутера-сусіда як «недоступний» у списку сусідів при відсутності повідомлень від цього роутера тривалий час;
- вузли-роутери повинні оновлювати статус роутера-сусіда як «доступний» у списку сусідів у разі отримання від цього роутера повідомлення;
- вузли-роутери повинні отримувати дані про виміри від вузлів-сенсорів;
- вузли-роутери повинні відправляти власний список роутерів-сусідів та поточний час вузлу-сенсору при отриманні від нього даних;
- вузли-роутери повинні відправляти отримані від вузла-сенсора дані вузлу-роутеру, що знаходиться ближче до шлюзу, або центральному шлюзу;
- центральний шлюз повинен отримувати дані від вузлів-роутерів;
- центральний шлюз повинен виводити отримані від вузлів-роутерів дані на екран.

3.2 Розробка архітектури мережі

Проектована мережа буде складатися з трьох типів пристроїв: вузли-сенсори (кінцеві точки), вузли-роутери, центральний шлюз.

Кінцева точка буде збирати дані з датчика та відправляти на найближчий доступний роутер. Передбачається, що у полі видимості кінцевої точки буде декілька роутерів. Це допоможе уникнути втрати даних або затримки у їх отриманні, коли найближчий до кінцевої точки роутер буде неактивним.

Оскільки більша частина логіки роботи кінцевої точки буде проходити, коли вона не підключена до мережі, для коректної оцінки стану роутерів вона буде зберігати локальний список роутерів та відповідний їм стан, що буде оновлюватись при кожному підключенні.

Роутери розташовуватимуться таким чином, що будуть також знаходитись у зоні видимості один одного. Окрім того, що це дасть можливість отримувати інформацію про стан мережі для передачі кінцевій точці, це також буде ключовим моментом для передачі даних на центральний шлюз. Роутер, що отримав дані від кінцевої точки, буде передавати ці дані роутеру, що знаходиться ближче до шлюзу, і так далі.

Центральний шлюз буде отримувати інформацію від роутерів та структурувати її. Окрім самих даних вимірів кінцевих точок, буде також зберігатися інформація про останній час відправки та стан автономного джерела живлення кінцевих точок. Це дасть оператору можливість аналізувати стан кінцевих точок та своєчасно приймати рішення про заміну або підзарядку автономного джерела живлення.

На рисунку 3.1 наведено запропоновану узагальнену архітектуру мережі.

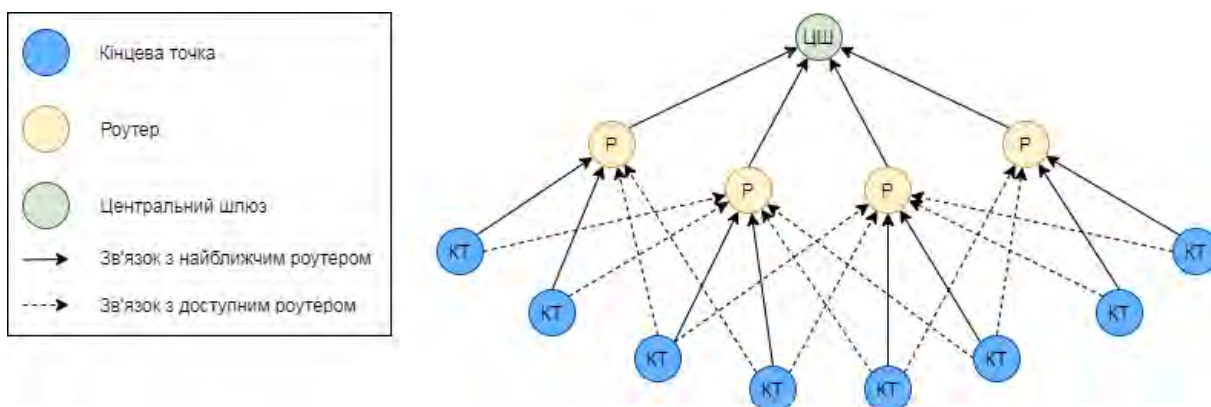


Рисунок 3.1 – Узагальнена архітектура мережі

3.3 Вибір програмно-апаратної платформи

Вибір програмно-апаратної платформи напряму залежить від обраного протоколу зв'язку. ESP-NOW підтримується тільки сімейством процесорів ESP. ESP32 – це серія недорогих та енергоефективних мікроконтролерів із вбудованим Wi-Fi і дворежимним Bluetooth. Виробник пропонує обширний вибір варіантів плат на базі цього сімейства мікроконтролерів [29].

ESP32 є наступником ESP8266. Він має додаткове ядро ЦП, швидший Wi-Fi, більше GPIO і підтримує Bluetooth 4.2 і BLE. Крім того, у ESP32 декілька GPIO є сенсорними, і їх можна використовувати для виведення ESP32 із глибокого сну. Обидві плати мають вбудовані Wi-Fi-адаптери, що підтримують стандарти 802.11 b/g/n [30]. У таблиці 3.1 наведено порівняльну характеристику цих плат.

До складу пристроїв - кінцевих точок будуть входити: плата з модулем ESP-32, автономне джерело живлення (батарея), датчик. Пристрої будуть встановлені безпосередньо у місцях для зняття вимірів з показників навколишнього середовища або процесів.

В якості роутерів будуть виступати плати з модулем ESP-32, підключені до стаціонарної мережі живлення або до акумулятора більшої ємності. Ці пристрої будуть отримувати дані від кінцевих точок та відправляти їх на центральний шлюз.

Таблиця 3.1 - Порівняльна характеристика модулів ESP8266 та ESP32 [29]-[30]

Модуль Характеристика	ESP8266	ESP32
1	2	3
Процесор	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
Частота процесору	80 МГц	160 МГц
Bluetooth	Відсутній	Bluetooth 4.2 та BLE
Кількість GPIO	17	34
Годинник реального часу (RTC)	Нема	Є

Продовження таблиці 3.1

1	2	3
Флеш-пам'ять	Нема	Є
Статична пам'ять SRAM	Нема	Є
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
Аналогово-цифровий перетворювач	10-бітний	12-бітний
CAN-шина	Нема	Є
Шляхи виходу з глибокого сну	Через сигнал скидування (подається через окрему ніжку)	За таймером, за сенсорним GPIO

Центральний шлюз може бути організовано на базі плати з модулем ESP-32, готових апаратно-програмних платформ, таких, як мінікомп'ютер Raspberry Pi або подібні, або персонального комп'ютера.

3.4 Вибір мови програмування

Для програмування модулів ESP-32 можливе використання таких мов програмування, як C, C++, Rust, Go (TinyGo) та Python (MicroPython) [31].

Аналіз літератури [32] дозволив виділити основні переваги та недоліки перелічених мов, актуальні для розробки вузлів БСМ: споживання енергії, час виконання та використання пам'яті. Результати аналізу представлені у табл. 3.2.

Таблиця 3.2 - Порівняльна характеристика мов програмування [31]-[32]

Мова Властивість	C	C++	Rust	Go (TinyGo)	Python (MicroPython)
Споживання енергії	Низьке	Низьке	Низьке	Високе	Дуже високе
Час виконання	Малий	Малий	Малий	Великий	Великий
Використання пам'яті	Мале	Мале	Середнє	Мале	Середнє

Отже, за результатами аналізу, мови програмування C/C++ можуть бути рекомендовані для програмування вузлів БСМ, так як це енергоефективні мови з малим часом виконання та використанням пам'яті пристрою.

3.5 Вибір середовища розробки

Розробка ПЗ мовою C/C++ для модулів ESP-32 може проводитись з використанням таких середовищ розробки, як ESP-IDF, PlatformIO IDE та Arduino IDE.

ESP-IDF [33] є офіційною інтегрованою платформою розробки від Espressif для розробки ПЗ пристроїв ESP. ESP-IDF надає найбільший контроль над пристроями, однак це також одне з найскладніших середовищ для налаштування та використання, його потрібно інстальювати як додаток до існуючої IDE.

PlatformIO IDE – це набір інструментів для розробки вбудованих систем мовою C/C++ [34]. PlatformIO працює на базі редактора коду VSCode.

Редактор коду VSCode має багато переваг стосовно зручності написання коду: швидка навігація за кодом, підсвітка синтаксису, автодоповнення слів. Однак, для роботи з платами PlatformIO необхідно встановлювати додатковий плагін. Окрім того, при великій кількості плагінів можуть виникати конфлікти.

Arduino IDE – інтегроване середовище розробки, призначене для створення і завантаження програм на Arduino-сумісні плати, а також на плати інших виробників [35].

Arduino IDE має інструменти для керування платами та вбудований монітор COM-порту, що дозволяє зручно завантажувати ПЗ на плати та проводити тестування.

На основі проведеного огляду було складено порівняльну характеристику, наведену у табл. 3.3.

На основі проведеного аналізу було обрано для розробки ПЗ середовище розробки Arduino IDE.

Таблиця 3.3 - Порівняльна характеристика середовищ програмування [33]-[35]

Середовище Характеристика	Arduino IDE	PlatformIO IDE	ESP-IDF
Легкість налаштування	Легко	Середньо	Складно
Зручність процесу кодування	Середньо	Легко	Легко
Зручність керування платами	Легко	Середньо	Середньо

3.6 Висновки до розділу 3

На основі постановки завдання дипломної роботи було виділено функціональні та нефункціональні вимоги до розроблюваної БСМ.

У відповідності до виявлених вимог було розроблено гібридну архітектуру мережі, що складається з трьох типів пристроїв: вузли-сенсори (кінцеві точки), вузли-роутери, центральний шлюз.

Було проведено вибір інструментарію розробки, до якого входить: вибір програмно-апаратної платформи, вибір мови програмування, вибір середовища програмування.

В якості програмно-апаратної платформи було обрано мікроконтролер ESP-32 за рахунок збалансованого набору характеристик.

Було проаналізовано мови програмування, які можуть бути використані для розробки ПЗ для модулів ESP-32. На основі проведеного порівняння було зроблено висновок про доцільність використання мов C/C++ для подальшої розробки.

Було також проаналізовано характеристики таких середовищ розробки, як Arduino IDE, PlatformIO IDE та ESP-IDF. Було обрано Arduino IDE через легкість налаштування середовища та зручність керування платами.

4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БЕЗДРОТОВОЇ СЕНСОРНОЇ МЕРЕЖІ

4.1 Розробка алгоритму роботи БСМ

Для мінімізації енергоспоживання кінцевих точок було розроблено алгоритм на базі нечіткої логіки, який коригує поведінку пристрою у залежності від його стану (заряд батареї та стан флеш-пам'яті), оцінки зміни даних та часу останнього підключення.

Після початкової ініціалізації, БСМ працює за наступним алгоритмом, схема якого представлена у виді UML-діаграми для трьох потоків на рис. 4.1.

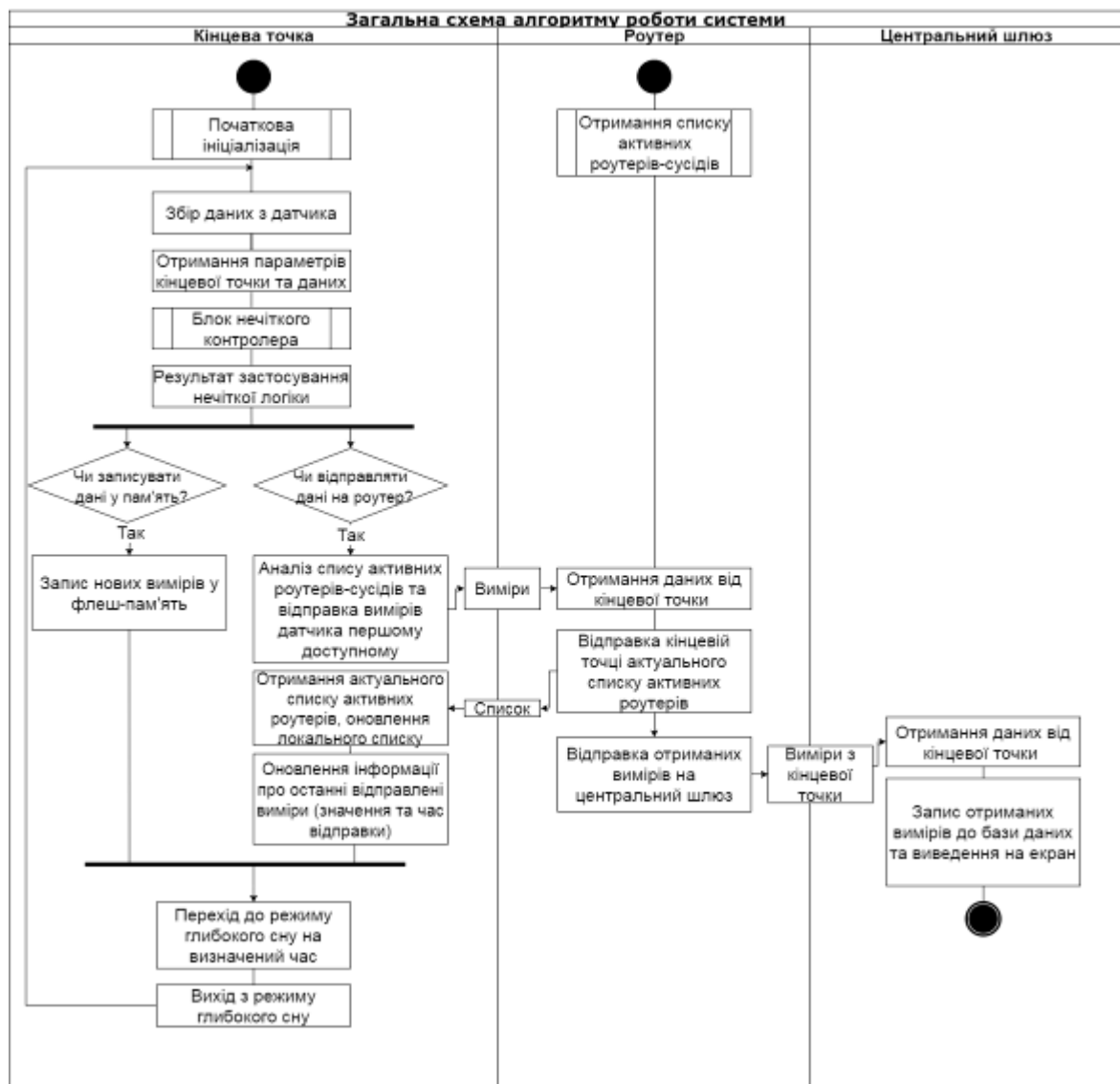


Рисунок 4.1 – Загальна схема алгоритму роботи системи

Спочатку, кінцева точка виходить з глибокого сну та на наступному кроці збирає дані з датчика. Після отримання даних, визначаються параметри для перетворення у лінгвістичні змінні:

- вимірювання заряду батареї;
- вимірювання заповненості флеш-пам'яті;
- оцінка зміни даних з датчика у порівнянні з останнім відправленим значенням;
- оцінка часу, що пройшов з моменту останньої відправки даних.

Варіанти значень лінгвістичних змінних:

- дуже негативний (very negative);
- негативний (negative);
- нуль (zero);
- позитивний (positive);
- дуже позитивний (very positive).

На наступному кроці до сформованих лінгвістичних змінних застосовуються нечіткі правила та отримується нечіткий вихід. Нечіткі правила – це чотирьохмірна матриця (за кількістю параметрів), в комірках якої вказані умовні індекси результату застосування правил.

Далі, в залежності від результату застосування нечітких правил, коригується поведінка кінцевої точки та перезаписуються глобальні змінні (останнє відправлене значення з датчика, час останнього підключення та ін.).

Після цього кінцева точка входить в режим глибокого сну.

Алгоритм початкової ініціалізації кінцевої точки (рис. 4.2) містить наступні блоки.

Блок 1. Початок.

Блок 2. Ініціалізація списку доступних роутерів. Список роутерів кінцевої точки – це роутери у зоні видимості точки з відповідним статусом активності. Роутери у списку розташовані за пріоритетом від найближчого до найдалшого. На початку всі роутери відзначені як активні.

Блок 3. Ініціалізація змінних для значень заряду джерела живлення – 100%, стану заповненості пам'яті – незаповнена, час останньої відправки даних – 0 за UNIX-часом.

Блок 4. Кінець початкової ініціалізації.

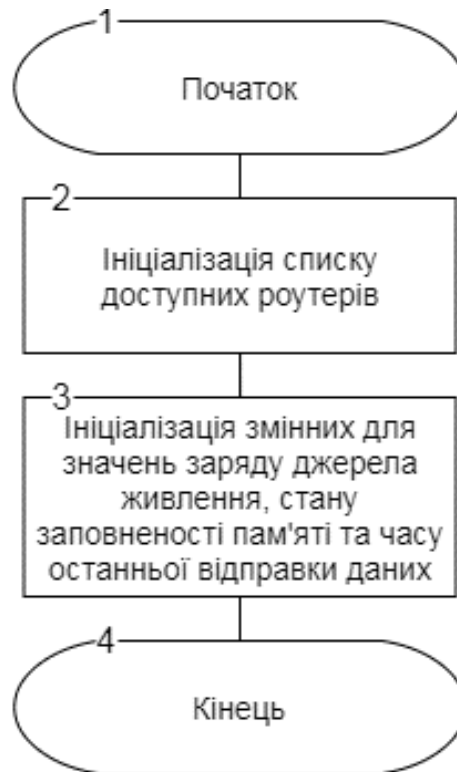


Рисунок 4.2 – Алгоритм початкової ініціалізації кінцевої точки

Алгоритм роботи контролера нечіткої логіки (рис. 4.3) наступний.

Блок 1. Початок роботи контролера.

Блок 2. Отримання значень параметрів від кінцевої точки – заряд батареї, заповненість флеш-пам'яті, величина зміни даних з датчика та час, який пройшов з моменту останньої відправки даних.

Блок 3. Отримання лінгвістичних значень для відповідних лінгвістичних змінних шляхом застосування функції приналежності до параметрів кінцевої точки.

Блок 4. Застосування до отриманих лінгвістичних змінних нечітких правил.

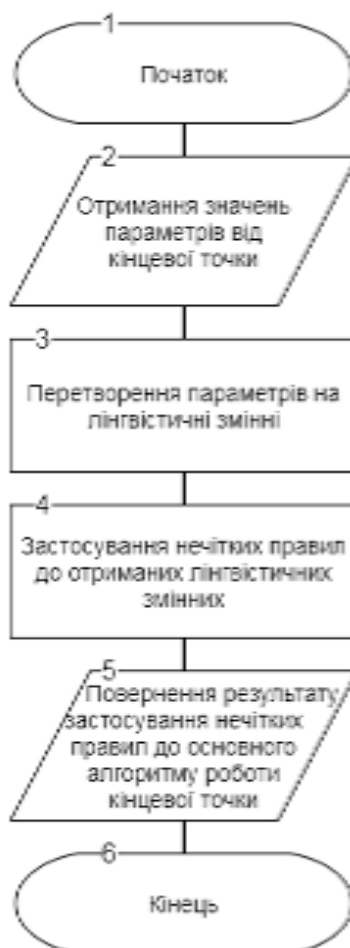


Рисунок 4.3 – Алгоритм роботи контролера нечіткої логіки

Блок 5. Результат застосування нечітких правил повертається в якості результату роботи контролера.

Блок 6. Кінець роботи контролера

Окрім алгоритму роботи кінцевої точки було розроблено алгоритм роботи роутера, задачею якого є з коректна робота кінцевих точок та обміну даними. Так, роутери, постійно знаходячись у режимі підключення до мережі, транслюють сусіднім роутерам прості повідомлення для формування списку «живих сусідів» - список роутерів, доступних для підключення. Цей алгоритм полягає в наступному (рис. 4.4).

Блок 1. Початок.

Блок 2. Відправлення повідомлення-keepalive усім роутерам-сусідам. Повідомлення-keepalive в даній системі – це просте текстове повідомлення з зазначенням MAC-адреси роутера-відправника та часу відправки.

Блок 3. Отримання повідомлення-keepalive від роутера-сусіда. Оскільки роутери завжди підключені до мережі, вони мають змогу отримувати повідомлення від інших роутерів та кінцевих точок.

Блок 4. Оновлення списку сусідів: MAC-адресі відправника у відповідність встановлюється стан "активний" (1) та записується час останнього повідомлення від цього роутера.

Блок 5. Перевірка списку на предмет неактивних роутерів. Якщо роутер не виходив на зв'язок більше за n часу, позначити його стан як "неактивний" (0)

Блок 6. Кінець.

Сформований список «живих сусідів» роутер надсилає кінцевій точці при її підключенні для оновлення даних точки щодо стану мережі.



Рисунок 4.4 – Алгоритм отримання списку активних роутерів-сусідів

Розроблений алгоритм роботи БСМ дозволяє зменшити енерговитрати кінцевих точок за рахунок того, що система буде обмежувати кількість

підключень кінцевої точки до мережі у випадках, коли в цьому немає необхідності (наприклад, виміри датчика не змінилися або змінилися несуттєво).

4.2 Контролер нечіткої логіки

Лінгвістичні змінні. Для роботи контролера нечіткої логіки одним з важливих етапів є перетворення параметрів кінцевої точки та даних у лінгвістичні змінні.

Для цього необхідно визначити лінгвістичні значення (нечіткі множини) цих змінних та границі для їх обчислення. Для зручності обчислень значення змінних для різних параметрів мають загальний вигляд, описаний в п.4.1.

Для кожного з параметрів ці множини мають власний зміст.

Для отримання лінгвістичного значення обчислюється функція приналежності за визначеними границями (в даному випадку використано трикутну функцію).

На рисунку 4.5 наведено нечіткі множини для лінгвістичної змінної "Заряд автономного джерела живлення".

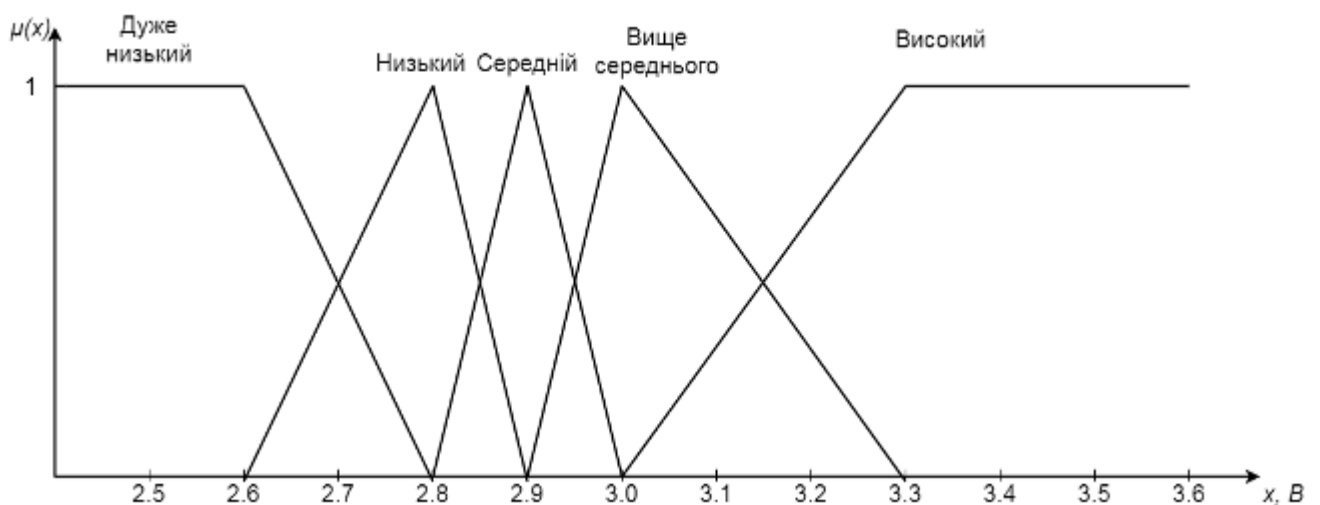


Рисунок 4.5 – Нечіткі множини для лінгвістичної змінної "Заряд автономного джерела живлення"

На осі x наведено напругу автономного джерела живлення від 2.5 В до 3.6 В. На цьому діапазоні напруги кінцева точка буде працювати. Напруга нижче за 2.5 В буде недостатньою для роботи пристрою, а вище 3.6 В – зовеликою.

На рисунку 4.6 наведено нечіткі множини для лінгвістичної змінної «Стан заповненості пам'яті».

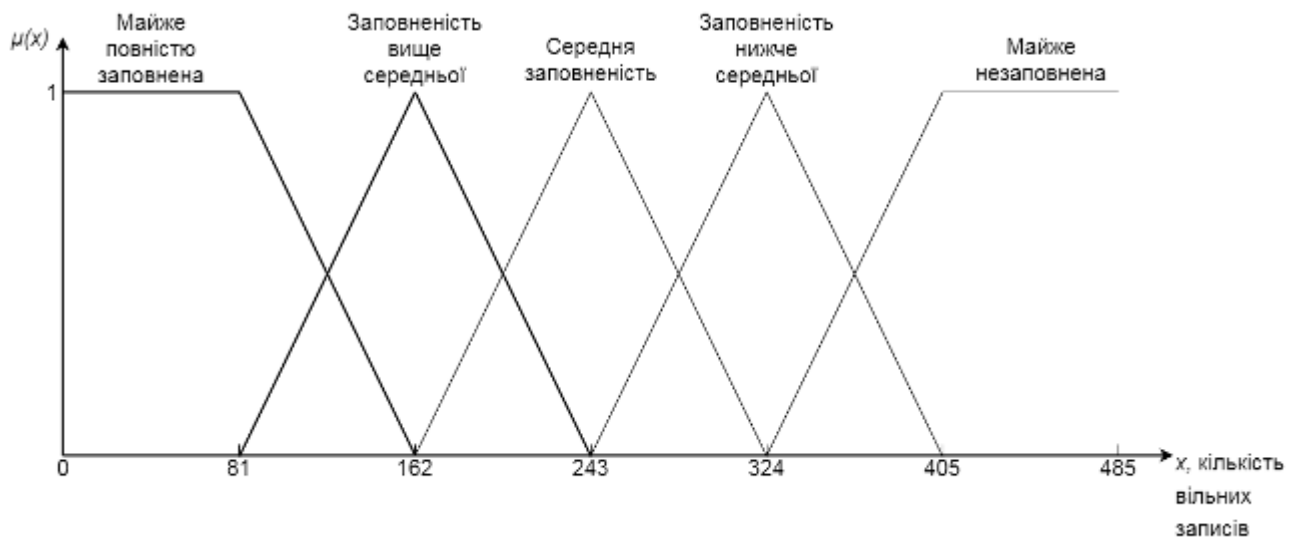


Рисунок 4.6 – Нечіткі множини для лінгвістичної змінної «Стан заповненості пам'яті»

На осі x наведено кількість вільних записів від 0 до 485. Ця кількість обумовлена структурою пам'яті модуля кінцевої точки. На рисунку 4.7 наведено нечіткі множини для лінгвістичної змінної «Час останньої відправки даних».

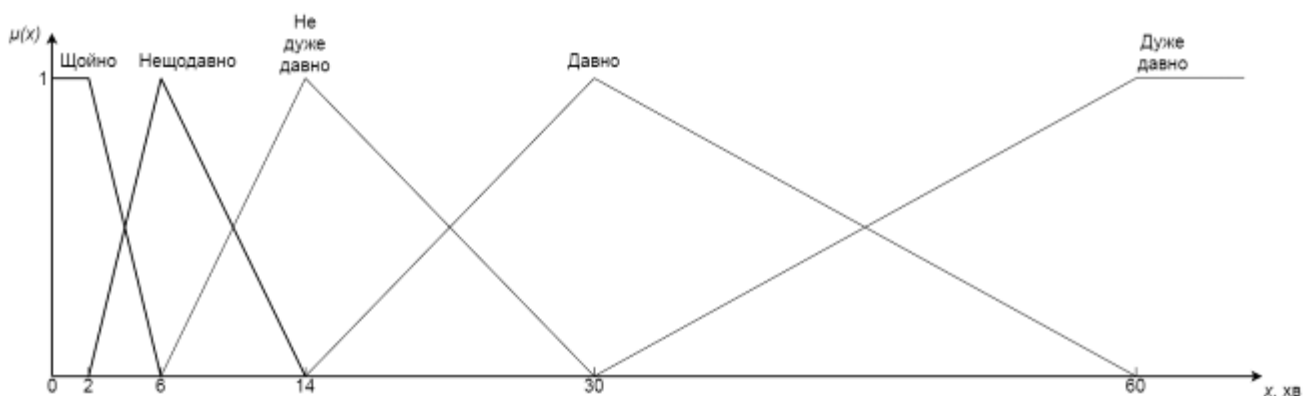


Рисунок 4.7 – Нечіткі множини для лінгвістичної змінної «Час останньої відправки даних»

На осі x наведено кількість часу у хвиликах, що пройшов з моменту останньої відправки даних від 0 до 60. При нормальному режимі роботи кінцевої точки відправка даних передбачається раз на хвилину (за умови достатнього заряду автономного джерела живлення та достатній величині зміни вимірів датчика).

На рисунку 4.8 наведено нечіткі множини для лінгвістичної змінної «Величина зміни даних».

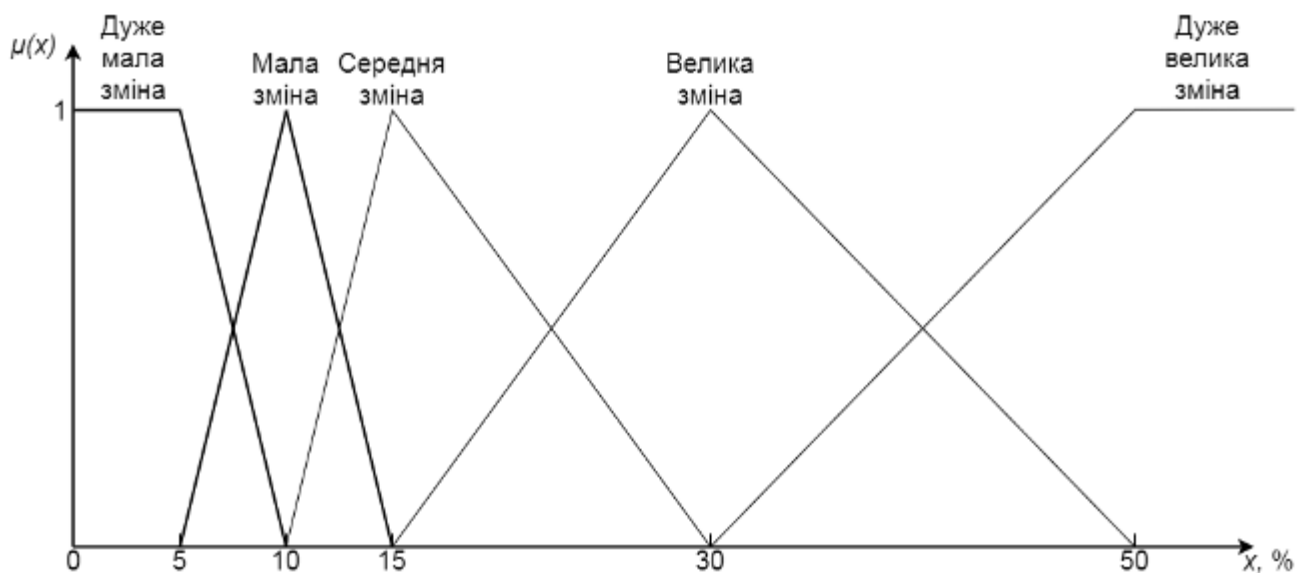


Рисунок 4.8 – Нечіткі множини для лінгвістичної змінної «Величина зміни даних»

На осі x наведено величину зміни даних по відношенню до останнього відправленого значення. Границі для обчислення нечітких множин можуть бути змінені в залежності від необхідної точності вимірів.

Дані правила дозволяють перетворити показники джерела живлення, пам'яті та характеристики останніх відправлених даних з датчиків у вигляд, з яким може працювати контролер нечіткої логіки – тобто у лінгвістичні змінні.

Нечіткі правила. Нечіткі правила, що визначають набір послідовність дій кінцевої точки, представлено у вигляді чотирьохвимірному масиву, у комірках якого записано номер рішення - гілки алгоритму, оптимальна для використання в

поточних обставинах (за наявного заряду автономного джерела живлення, стану пам'яті, величини зміни вимірів та тривалості часу мовчання).

Індекси у масиві – це числове представлення лінгвістичного значення параметра. Так, індекси [4][0][1][2] будуть аналогічні такому набору змінних: високий заряд автономного джерела живлення, майже повністю заповнена пам'ять, мала зміна даних датчика, остання відправка даних була не дуже давно.

Передбачені варіанти рішень наступні:

- рішення 1 – кінцева точка не відправляє дані та не записує нічого до пам'яті. Дане рішення передбачене для випадків, коли з останньої відправки даних пройшло мало часу, а виміри з датчика майже не змінилися у порівнянні з останніми відправленими, заряд джерела живлення – вище середнього та високий;

- рішення 2 – кінцева точка відправляє останні виміри, а також відправляє дані з пам'яті пристрою, заряд джерела живлення – вище середнього та високий;

- рішення 3 – новий вимір записується до пам'яті пристрою. Дане рішення передбачено для невеликих змін у вимірах датчика при тривалості часу мовчання від «Щойно» до «Не дуже давно», а також для дуже низьких зарядів автономного джерела живлення;

- рішення 4 – кінцева точка відправляє останні виміри. Дане рішення передбачено для середньої заповненості пам'яті при середньому заряді автономного джерела живлення;

- рішення 5 – кінцева точка відправляє останні виміри з датчика та збільшує тривалість глибокого сну для більшого заощадження. Це рішення дозволить при низьких зарядах автономного джерела живлення довше підтримувати зв'язок з центральним шлюзом;

- рішення 6 - кінцева точка відправляє останні виміри з датчика та ще збільшує тривалість глибокого сну для більшого заощадження.

4.3 Структура ПЗ

Програмна реалізація БСМ складається з чотирьох блоків: блок роботи кінцевої точки, блок контролера нечіткої логіки, блок роботи роутера та блок роботи центрального шлюзу.

Блок роботи кінцевої точки складається з модулів, що забезпечують функціонування кінцевої точки:

- модуль отримання значень параметрів;
- модуль роботи з пам'яттю;
- модуль відправлення та отримання повідомлень;
- модуль виконання рішень контролера нечіткої логіки.

Блок роботи контролера нечіткої логіки включає в себе модулі, що забезпечують застосування нечіткої логіки:

- модуль нечітких правил;
- модуль отримання рішення;
- модуль роботи з лінгвістичними змінними.

Блок роботи роутера забезпечує обмін даними між роутерами та між роутером та кінцевими точками і включає в себе:

- модуль відправлення та отримання повідомлень;
- модуль роботи з часом.

Блок роботи центрального шлюза включає в себе:

- модуль отримання даних;
- модуль роботи з часом.

На рис. 4.9 наведено структурну схему ПЗ БСМ.

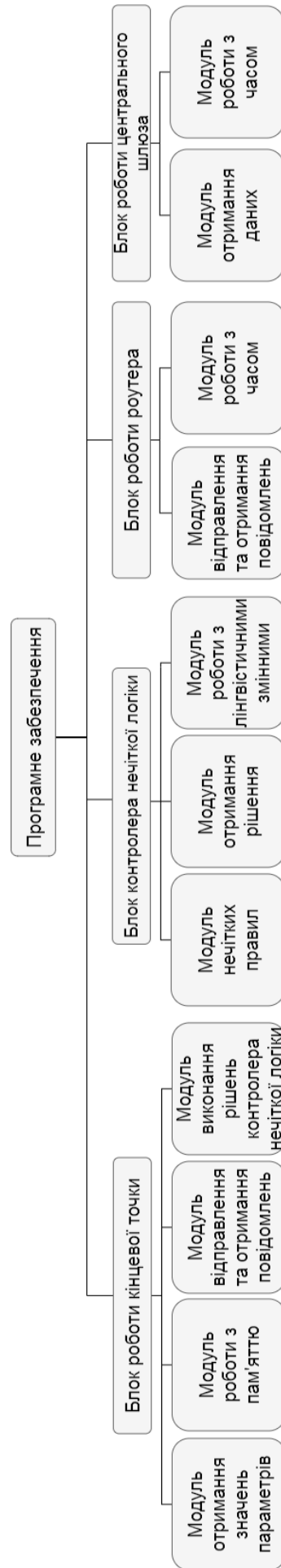


Рисунок 4.9 – Структурна схема ПЗ БСМ

4.4 Опис особливостей програмної реалізації

У табл. 4.1 наведено змінні, константи, структури та функції, що належать до модулів роботи кінцевої точки.

Таблиця 4.1 - Змінні, константи, структури та функції блоку роботи кінцевої точки (Додаток А.1)

Ім'я та тип	Опис
1	2
Глобальні змінні та константи	
static uint8_t selfMAC[]	Власна MAC-адреса точки
const uint8_t peerLength	Довжина MAC-адреси
static uint8_t PEERS[][peerLength]	Список MAC-адрес роутерів
const int PEERS_COUNT	Кількість роутерів у списку
const uint8_t maxPeerCount	Максимально можлива кількість роутерів
static uint8_t channel	Канал для підключення Wi-Fi
const int sensorDataPin	Канал GPIO для зчитування даних з сенсору
const int sensorPowerPin	Канал GPIO для подання живлення на сенсор
const int batterySensorPin	Канал GPIO для зчитування напруги джерела живлення
int sent_status	Статус відправлення даних на роутер (0 – невизначено, 1 – успішно, 2 – невдало)
int recv_status	Статус отримання даних від роутера (0 – невизначено, 1 – успішно, 2 – невдало)

Продовження таблиці 4.1

1	2
const int DURATION_SLEEP_NORMAL	Нормальна тривалість сну
const int DURATION_SLEEP_LONG	Довга тривалість сну
const int DURATION_SLEEP_VERY_LONG	Дуже довга тривалість сну
float battery	Значення напруги джерела живлення
msg_data msg	Змінна для даних для відправки на роутер
sensor_data new_data	Змінна для нових вимірів сенсора
int num_of_values_in_flash	Кількість записів у пам'яті
RTC_DATA_ATTR uint8_t ALIVE_PEERS[]	Список статусів роутерів зі списку (0 – недоступний, 1 – доступний)
RTC_DATA_ATTR unsigned long counter	Лічильник часу (unix-час)
RTC_DATA_ATTR int duration_sleep	Поточне значення тривалості сну
RTC_DATA_ATTR sensor_data last_sent_data	Значення останніх вимірів сенсорю
Структури	
struct sensor_data	Виміри сенсора (включають в себе значення вимірів та час зняття вимірів)
struct msg_data	Дані для відправки на роутер (включають в себе значення вимірів сенсора, час зняття вимірів, значення напруги джерела живлення та кількість записів у пам'яті)

Продовження таблиці 4.1

1	2
struct router_data	Дані, що отримані від роутера (включають в себе отримане значення часу, кількість сусідів роутера та локальний список сусідів роутера)
Функції	
int count_alive_peers()	Функція підрахунку доступних роутерів
void make_all_peers_alive()	Функція для встановлення усім роутерам зв списку статусу «доступний»
int decision_1()	Функція, що виконує послідовність дій за рішенням 1 контролеру нечіткої логіки
int decision_2()	Функція, що виконує послідовність дій за рішенням 2 контролеру нечіткої логіки
int decision_3()	Функція, що виконує послідовність дій за рішенням 3 контролеру нечіткої логіки
int decision_4()	Функція, що виконує послідовність дій за рішенням 4 контролеру нечіткої логіки
int decision_5()	Функція, що виконує послідовність дій за рішенням 5 контролеру нечіткої логіки

Продовження таблиці 4.1

1	2
int decision_6()	Функція, що виконує послідовність дій за рішенням 6 контролеру нечіткої логіки
void sensorPowerOn()	Функція подання живлення на сенсор
void sensorPowerOff()	Функція вимкнення живлення сенсору
void get_sensor_data(uint8_t delay_time=0)	Функція отримання вимірів з сенсору
float get_battery_value()	Функція отримання значення напруги джерела живлення
int get_memory_value()	Функція отримання значення кількості записів у пам'яті
float get_delta_value()	Функція отримання значення величини зміни вимірів у порівнянні з останнім відправленим значенням
float get_silent_time_value()	Функція отримання значення часу, протягом якого кінцева точка не надсилала дані на роутер
void get_msg_text()	Функція для запису даних у структуру повідомлення
void write_data()	Функція запису вимірів до пам'яті
void send_message(bool send_archive)	Функція відправки повідомлення. Параметр send_archive – «прапорець», що відображає необхідність надсилання даних з пам'яті
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)	Функція, що викликається при відправці повідомлень (оновлює статус відправки даних на роутер)

Продовження таблиці 4.1

1	2
<code>void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)</code>	Функція, що викликається при отриманні повідомлень (оновлює локальний лічильник часу та список роутерів)
<code>bool addPeer(const uint8_t mac[6])</code>	Функція для додавання MAC-адреси
<code>void init_espnw()</code>	Функція ініціалізації роботи ESP-NOW
<code>void setup()</code>	Функція ініціалізації роботи точки. Так як у неактивному стані точка входить в режим глибокого сну, ця функція виконується після кожного пробудження точки. Містить виклики функцій для отримання значень параметрів точки, отримання рішення від контролеру нечіткої логіки та виклик функцій в залежності від отриманого рішення
<code>void loop()</code>	Функція, обов'язкова для компілятора Arduino IDE

У табл. 4.2 наведено змінні та функції, що належать до модулів контролера нечіткої логіки.

Таблиця 4.2 - Змінні та функції блоку контролера нечіткої логіки (Додаток А.2)

Ім'я та тип	Опис
1	2
Глобальні змінні	
<code>static int fuzzy_rules[5][5][5][5]</code>	Масив нечітких правил

Продовження таблиці 4.2

1	2
Функції	
void init_fuzzy_rules()	Функція ініціалізації масиву нечітких правил
float triangular(float a, float b, float c, float x, int bounds)	Трикутна функція для обчислення значень лінгвістичних змінних
int membership_battery(float x)	Функція обчислення значення лінгвістичної змінної заряду джерела живлення
int membership_memory(float x)	Функція обчислення значення лінгвістичної змінної стану заповненості пам'яті
int membership_delta(float x)	Функція обчислення значення лінгвістичної змінної величини зміни даних
int membership_silent_time(float x)	Функція обчислення значення лінгвістичної змінної часу останньої відправки даних
int fuzzy_controller(float b, float m, float d, float st)	Функція роботи контролера нечіткої логіки

У табл. 4.3 наведено змінні, константи, структури та функції, що належать до модулів роботи роутера.

Таблиця 4.3 - Змінні, константи, структури та функції модулів роботи роутера (Додаток А.3)

Ім'я та тип	Опис
1	2
Глобальні змінні та константи	
static uint8_t selfMAC[]	Власна MAC-адреса точки

Продовження таблиці 4.3

1	2
const uint8_t peerLength	Довжина MAC-адреси
static uint8_t PEERS[][peerLength]	Список MAC-адрес роутерів
static uint8_t ENDPOINTS[][peerLength]	Список MAC-адрес кінцевих точок
static uint8_t GATEWAY[peerLength]	MAC-адреса центрального шлюзу
const uint8_t maxPeerCount	Максимальна кількість роутерів
uint8_t ALIVE_PEERS[][8]	Список MAC-адрес роутерів-сусідів зі статусом (0 – недоступний, 1 – доступний)
static uint8_t channel	Канал для підключення Wi-Fi
router_data outgoingData	Змінна для даних для відправки на кінцеву точку
endpoint_data incomingReadings	Змінна для даних, що надійшли від кінцевої точки
router_to_gateway_data new_data	Змінна для даних для відправки на центральний шлюз
const char* ssid const char* password const char* ntpServer1 const char* ntpServer2 const long gmtOffset_sec const int daylightOffset_sec const char* time_zone	Константи, необхідні для підключення до Wi-Fi для отримання актуального часу
time_t now tm myTimeInfo	Змінні для даних про час
bool need_to_send_to_cg	«Прапорець», що відображає необхідність відправлення даних на центральний шлюз

Продовження таблиці 4.3

1	2
RTC_DATA_ATTR unsigned int counter	Лічильник часу (unix-час)
Структури	
struct endpoint_data	Дані, що отримані від кінцевої точки (включають в себе отримане значення часу, кількість сусідів роутера та локальний список сусідів роутера)
struct router_data	Дані для відправки на кінцеву точку (включають в себе значення часу, кількість сусідів роутера та локальний список сусідів роутера)
struct router_to_gateway_data	Дані для відправки на центральний шлюз (включають в себе MAC-адресу кінцевої точки та дані, отримані від кінцевої точки)
Функції	
void printLocalTime()	Функція виведення поточного часу на екран
unsigned long getTime()	Функція для отримання значення поточного часу
int local_router_number(const uint8_t *mac)	Функція для отримання порядкового номеру роутера у списку роутерів-сусідів за MAC-адресою
int local_endpoint_number(const uint8_t *mac)	Функція для отримання порядкового номеру кінцевої точки у списку кінцевих точок за MAC-адресою

Продовження таблиці 4.3

1	2
<code>void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)</code>	Функція, що викликається при відправці повідомлень
<code>void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)</code>	Функція, що викликається при отриманні повідомлень (оновлює статус роутера-відправника, якщо повідомлення від роутера; зберігає дані кінцевої точки для відправки на роутер та надсилає необхідну інформацію кінцевій точці, якщо повідомлення від кінцевої точки)
<code>bool addPeer(const uint8_t mac[6])</code>	Функція для додавання MAC-адреси
<code>void synchroniseWith_NTP_Time()</code>	Функція синхронізації часу із сервером
<code>void init_espnow()</code>	Функція ініціалізації роботи ESP-NOW
<code>void setup()</code>	Функція ініціалізації роботи роутера (синхронізація часу, ініціалізація протоколу зв'язку)
<code>void loop()</code>	Функція циклу роботи роутера (відправлення повідомлень-keepalive роутерам-сусідам, відправлення збережених даних від кінцевої точки на центральний шлюз)

У табл. 4.4 наведено змінні, константи, структури та функції, що належать до модулів роботи центрального шлюзу.

Таблиця 4.4 - Змінні, константи, структури та функції модулів роботи центрального шлюзу (Додаток А.4)

Ім'я та тип	Опис
1	2
Глобальні змінні та константи	
static uint8_t selfMAC[]	Власна MAC-адреса точки
const uint8_t peerLength	Довжина MAC-адреси
static uint8_t ROUTERS[][peerLength]	Список MAC-адрес роутерів
static uint8_t ENDPOINTS[][peerLength]	Список MAC-адрес кінцевих точок
const uint8_t maxPeerCount	Максимально можлива кількість роутерів
static uint8_t channel	Канал для підключення Wi-Fi
endpoint_data incomingReadings	Змінна для даних, що надійшли від кінцевої точки
const char* ssid const char* password const char* ntpServer1 const char* ntpServer2 const long gmtOffset_sec const int daylightOffset_sec const char* time_zone	Константи, необхідні для підключення до Wi-Fi для отримання актуального часу
time_t now tm myTimeInfo	Змінні для даних про час
Структури	
struct router_data	Дані, отримані від роутера (включають в себе MAC-адресу кінцевої точки та її параметри)

Продовження таблиці 4.4

1	2
Функції	
void printLocalTime()	Функція виведення поточного часу на екран
unsigned long getTime()	Функція для отримання значення поточного часу
int local_router_number(const uint8_t *mac)	Функція для отримання порядкового номеру роутера у списку роутерів-сусідів за MAC-адресою
int local_endpoint_number(const uint8_t *mac)	Функція для отримання порядкового номеру кінцевої точки у списку кінцевих точок за MAC-адресою
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)	Функція, що викликається при відправці повідомлень
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)	Функція, що викликається при отриманні повідомлень (виводить отриману інформацію на екран)
bool addPeer(const uint8_t mac[6])	Функція для додавання MAC-адреси
void synchroniseWith_NTP_Time()	Функція синхронізації часу із сервером
void init_espnow()	Функція ініціалізації роботи ESP-NOW
void setup()	Функція ініціалізації роботи центрального шлюзу (синхронізація часу, ініціалізація протоколу зв'язку)
void loop()	Функція циклу роботи роутера

4.5 Висновки до розділу 4

У цьому розділі було описано алгоритми роботи прототипу майбутньої БСМ, наведено схеми роботи алгоритмів. Було розглянуто алгоритм роботи вузлів мережі: алгоритм роботи кінцевих точок та роутерів.

Окремо було розглянуто роботу контролера нечіткої логіки: було наведено опис лінгвістичних змінних, розглянуто можливі рішення контролеру.

Було також розглянуто методику перетворення значень параметрів кінцевої точки та даних у вигляд лінгвістичних змінних, наведено можливі лінгвістичні значення для кожної змінної.

Окрім того, було наведено опис особливостей програмної реалізації прототипу БСМ: було наведено схеми модулів для блоків роботи кінцевої точки, роутера та центрального шлюза. Також, описи змінних, констант, структур та функцій для наведених модулів були представлені у вигляді таблиць.

5 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОТОТИПУ БСМ

5.1 Опис особливостей реалізації прототипу

На основі результатів проведеного в розділах 3 та 4 проектування та конструювання БСМ, було розроблено прототип мережі. Прототип складається з трьох кінцевих точок, двох роутерів та центрального шлюза. Кінцеві точки реалізовано на основі модуля ESP32-WROOM. Роутери реалізовано на основі плати Devkit з модулем ESP32-WROOM. В якості центрального шлюза виступає персональний комп'ютер. До кінцевих точок під'єднано датчики: освітленості, температури та Холла. На рисунку 5.1 наведено структурну схему прототипу.

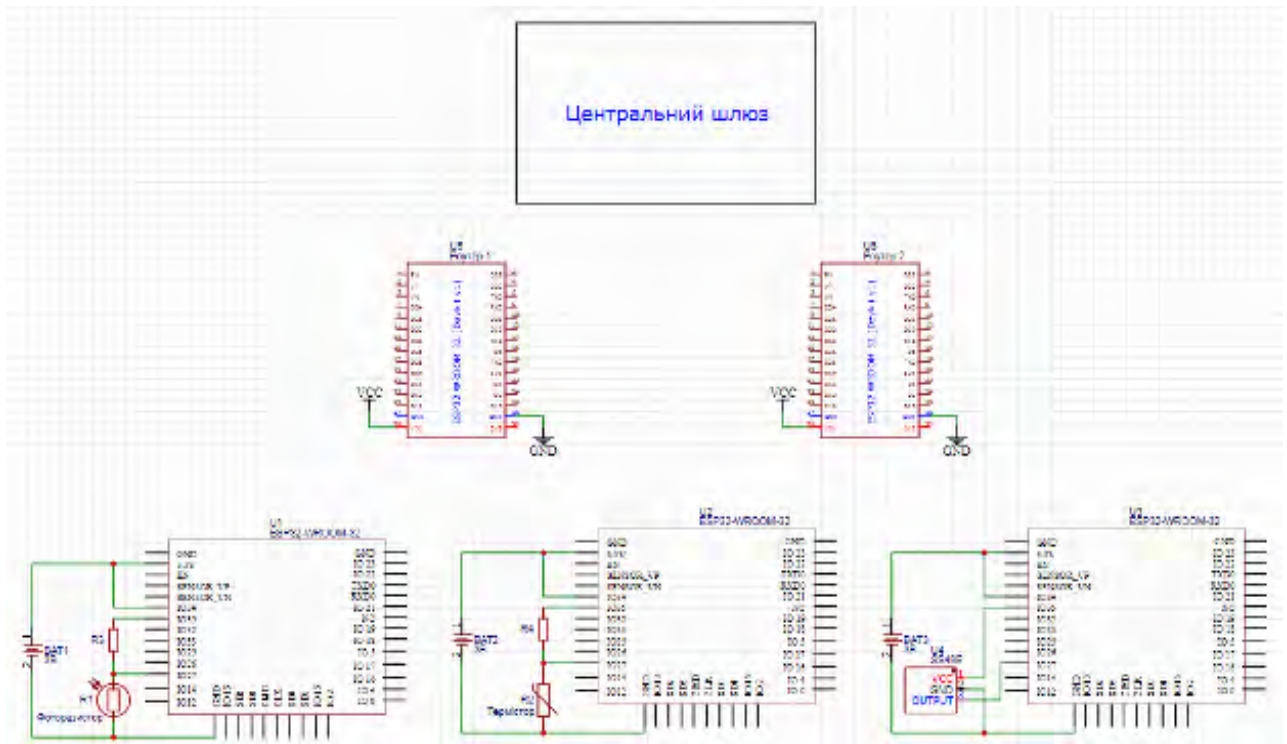


Рисунок 5.1 - Структурна схема прототипу мережі

Для зручності прошивки та налагодження було зібрано модулі-програматори для кожної кінцевої точки. Схему програматора наведено на рисунку 5.2.

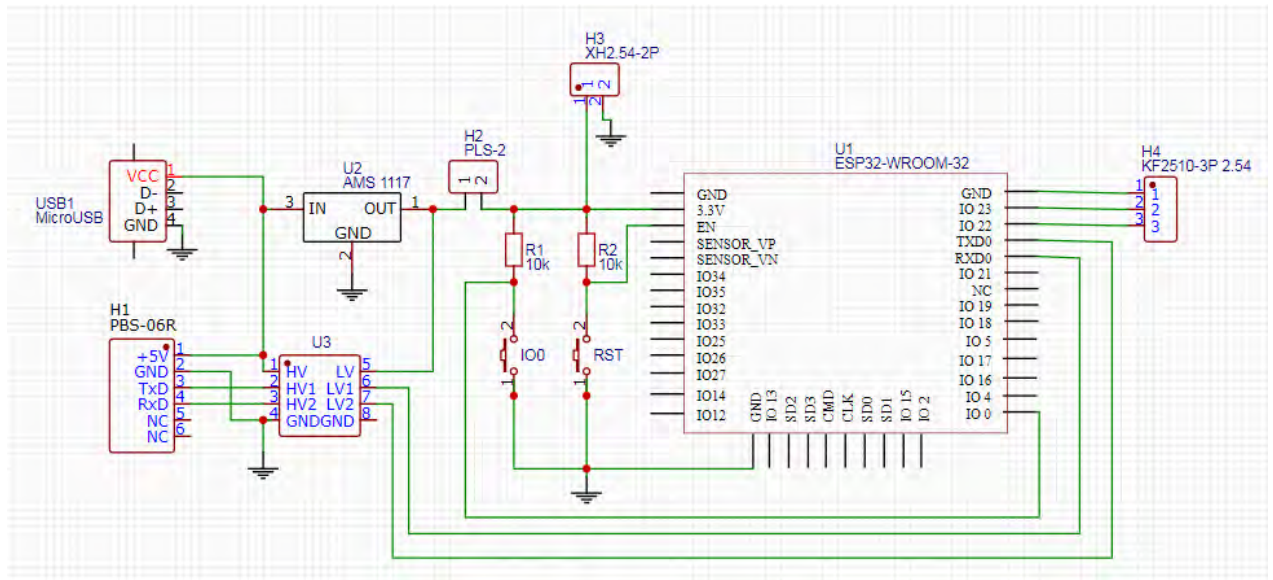


Рисунок 5.2 - Схема програматора

Елементи програматора наступні:

- U1 – модуль ESP-32 WROOM;
- U2 – стабілізатор на 3.3В;
- U3 – перетворювач логічних рівнів;
- Н1 – роз’єм підключення USB-UART адаптеру;
- Н2 – перемичка підключення стабілізатора у режимі налагодження (без автономного джерела живлення);
- Н3 – роз’єм для підключення автономного джерела живлення;
- Н4 – роз’єм для підключення датчика;
- USB1 – роз’єм для підключення живлення у режимі налагодження.

На рисунках 5.3 та 5.4 наведено зовнішній вигляд зібраних кінцевих точок з датчиком освітленості та датчиком температури відповідно.

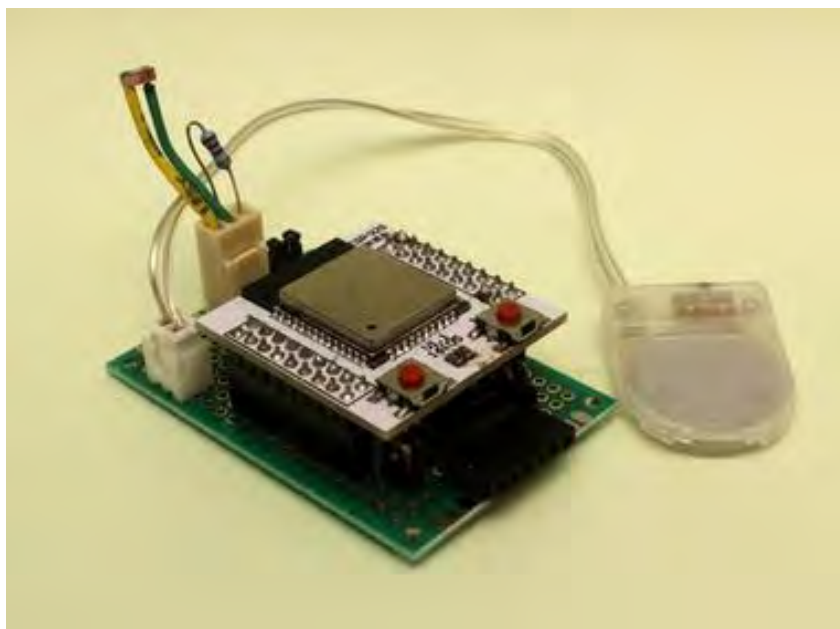


Рисунок 5.3 - Кінцева точка (датчик освітленості)

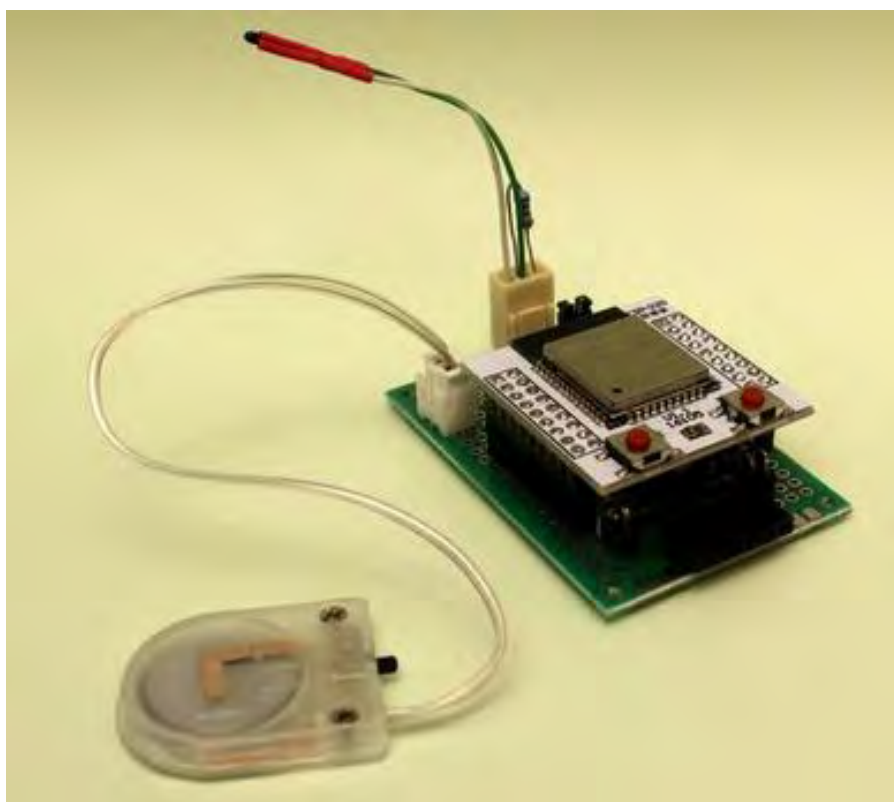


Рисунок 5.4 - Кінцева точка (датчик температури)

На рисунку 5.5 наведено зовнішній вигляд плати Devkit з модулем ESP32-WROOM, на основі якого реалізовано роутери.



Рисунок 5.5 - Плата Devkit з модулем ESP32-WROOM (роутер)

На рисунку 5.6 наведено зовнішній вигляд готового прототипу мережі (три кінцеві точки, два роутери).

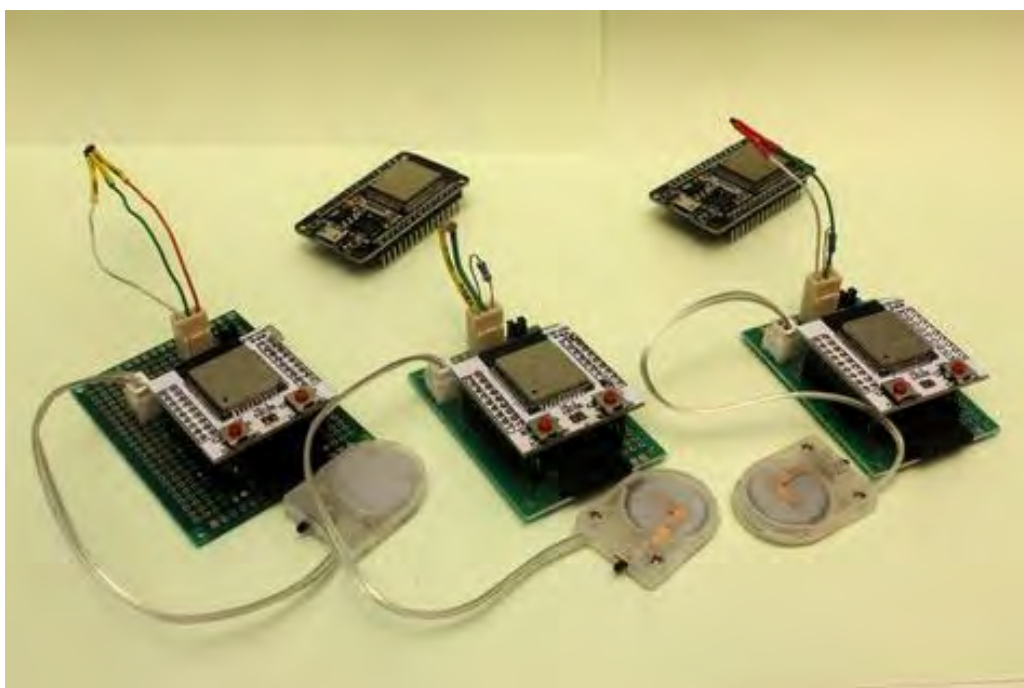


Рисунок 5.6 - Готовий прототип мережі

5.2 Тестування прототипу

Для тестування роботи розробленого прототипу БСМ було розроблено наступні сценарії тестування кінцевої точки:

- сценарій 1. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – високий, заповненість пам'яті – майже незаповнена, величина зміни даних – дуже мала зміна, час останньої відправки даних – щойно. Дані значення параметрів відповідають нечіткому рішенню 1;

- сценарій 2. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – високий, заповненість пам'яті – майже незаповнена, величина зміни даних – дуже велика зміна, час останньої відправки даних – щойно; у пам'яті пристрою відсутні записи, у мережі є доступні для підключення роутери. Дані значення параметрів відповідають нечіткому рішенню 2;

- сценарій 3. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – високий, заповненість пам'яті – майже незаповнена, величина зміни даних – дуже велика зміна, час останньої відправки даних – щойно; у пам'яті пристрою наявні записи, у мережі є доступні для підключення роутери. Дані значення параметрів відповідають нечіткому рішенню 2;

- сценарій 4. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – високий, заповненість пам'яті – майже незаповнена, величина зміни даних – дуже велика зміна, час останньої відправки даних – щойно; у мережі відсутні доступні для підключення роутери. Дані значення параметрів відповідають нечіткому рішенню 2;

- сценарій 5. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – середній, заповненість пам'яті – майже незаповнена, величина зміни даних – дуже мала зміна, час

останньої відправки даних – не дуже давно. Дані значення параметрів відповідають нечіткому рішенням 3;

- сценарій 6. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – середній, заповненість пам'яті – середня заповненість, величина зміни даних – дуже мала зміна, час останньої відправки даних – не дуже давно. Дані значення параметрів відповідають нечіткому рішенням 4;

- сценарій 7. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – низький, заповненість пам'яті – середня заповненість, величина зміни даних – дуже мала зміна, час останньої відправки даних – дуже давно. Дані значення параметрів відповідають нечіткому рішенням 5;

- сценарій 8. результат роботи кінцевої точки при умовних значеннях параметрів: заряд автономного джерела живлення – низький, заповненість пам'яті – майже повністю заповнена, величина зміни даних – середня зміна, час останньої відправки даних – дуже давно. Дані значення параметрів відповідають нечіткому рішенням 6;

- сценарій 9. Оновлення локального списку роутерів кінцевої точки при отриманні даних від роутера;

- сценарій 10. Підключення кінцевої точки до різних роутерів в залежності від їх статусу доступності до підключення.

Окрім того, було розроблено сценарії тестування роутерів та центрального шлюза:

- сценарій 1. Отримання роутером вимірів від кінцевої точки;
- сценарій 2. Отримання роутером архіву кінцевої точки – записів, які були збережені у пам'яті кінцевої точки;
- сценарій 3. Відправлення роутером локального списку роутерів-сусідів та поточного часу кінцевій точці при отриманні даних від неї;

- сценарій 4. Відправлення роутером даних, отриманих від кінцевої точки, на центральний шлюз;
- сценарій 5. Отримання центральним шлюзом даних від роутерів та виведення їх на екран.

Тестування кінцевої точки. На рис. 5.7 – 5.14 наведено скріншоти роботи кінцевої точки. У монітор СОМ-порту виводиться інформація від контролера нечіткої логіки (значення лінгвістичних змінних), поточний стан точки, номер обраного рішення та короткий опис дій.

Рис. 5.7 відображає роботу кінцевої точки при роботі за сценарієм 1.

```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 1: battery - 4, memory - 4, delta - 0, st - 4
Current state: sensor - 0.64, timestamp - 1, battery - 3.20, memory state - 507
Decision 1
ets Jul 29 2019 12:21:46
```

Рисунок 5.7 - Сценарій 1

Рис. 5.8 відображає роботу кінцевої точки при роботі за сценарієм 2. Можна побачити, що точка виводить наступну інформацію: відправлене повідомлення, MAC-адресу роутера-отримувача, статус відправленого повідомлення, отримане від роутера значення unix-часу та список роутерів у полі зору роутера-отримувача з відповідними їм статусами (0 – роутер недоступний, 1 – роутер доступний), час очікування відповіді від роутера.

Наявна на екрані фраза «Archive was sent» свідчить про те, що було виконано алгоритм відправлення збережених у архіві даних. Рис. 5.8 відображає ситуацію, коли у пам'яті точки не було збережено вимірів датчика.

```

ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.61, timestamp - 1, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.61, timestamp - 1, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success
Archive was sent

Time obtained: 1675871132
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 1
ets Jul 29 2019 12:21:46

```

Рисунок 5.8 - Сценарій 2

При виконанні сценарію 3, тобто за наявності записів у пам'яті, точка виведе їх у СОМ-порт при відправці, як показано на рис. 5.9.

```

Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.59, timestamp - 1675871269, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.59, timestamp - 1675871269, battery - 3.30%, memory state - 427
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675871277
Peer: {40:22:D8:EB:29:4C}, status: 00

Last Packet Send Status:      Delivery Success
Archive msg: sensor - 0.59, timestamp - 1675871245
Archive msg: sensor - 0.59, timestamp - 1675871246

Last Packet Send Status:      Delivery Success
Archive msg: sensor - 0.59, timestamp - 1675871247

Last Packet Send Status:      Delivery Success
Archive msg: sensor - 0.59, timestamp - 1675871248

Last Packet Send Status:      Delivery Success
Archive msg: sensor - 0.59, timestamp - 1675871249

```

Рисунок 5.9 - Сценарій 3

На рис. 5.10 наведено роботу точки за сценарієм 4, тобто усі роутери зі списку виявилися недоступними. У цьому випадку кінцева точка занесе поточні виміри датчика у пам'ять.

```
Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.59, timestamp - 1675871246, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.59, timestamp - 1675871246, battery - 3.30%, memory state - 499
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Fail
No router available; Saved data instead
ets Jul 29 2019 12:21:46
```

Рисунок 5.10 - Сценарій 4

На рис. 5.11 наведено роботу кінцевої точки за сценарієм 5. Відбувається запис у пам'ять.

```
rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 3: battery - 2, memory - 4, delta - 0, st - 2
Current state: sensor - 0.61, timestamp - 2, battery - 2.95, memory state - 419
Decision 3
```

Рисунок 5.11 - Сценарій 5

На рис. 5.6 наведено скріншоти роботи кінцевої точки за сценарієм 6. Зі скріншоту видно, що алгоритм відправлення збережених даних не був відпрацьований, як того і потребує рішення 4 контролера нечіткої логіки. Для тестування в якості нормальної, довгої та дуже довгої тривалостей глибокого сну було встановлено 1, 2 та 3 секунди відповідно. Можна побачити, що різниця між

часом, що надіслав роутер, та часом наступних вимірів на рис. 5.12 складає 1 секунду, що відповідає нормальній тривалості глибокого сну.

```
rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 4: battery - 2, memory - 2, delta - 0, st - 2
Current state: sensor - 0.65, timestamp - 1675872713, battery - 2.95, memory state - 244
Decision 4
Msg: sensor - 0.65, timestamp - 1675872713, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675872714
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 4: battery - 2, memory - 2, delta - 0, st - 2
Current state: sensor - 0.65, timestamp - 1675872715, battery - 2.95, memory state - 244
Decision 4
Msg: sensor - 0.65, timestamp - 1675872715, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675872715
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46
```

Рисунок 5.12 - Сценарій 6

На рис. 5.13 наведено скріншоти роботи кінцевої точки сценарієм 7. Можна побачити, що різниця між часом, що надіслав роутер, та часом наступних вимірів на рис. 5.13 складає 2 секунди, що відповідає довгій тривалості глибокого сну.

```
Fuzzy | Decision 5: battery - 1, memory - 2, delta - 0, st - 0
Current state: sensor - 0.65, timestamp - 1675872944, battery - 2.79, memory state - 244
Decision 5
Msg: sensor - 0.65, timestamp - 1675872944, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675872945
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 5: battery - 1, memory - 2, delta - 0, st - 0
Current state: sensor - 0.65, timestamp - 1675872947, battery - 2.79, memory state - 244
Decision 5
Msg: sensor - 0.65, timestamp - 1675872947, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675872947
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46
```

Рисунок 5.13 - Сценарій 7

На рис. 5.14 наведено скріншоти роботи кінцевої точки за сценарієм 8. Можна побачити, що різниця між часом, що надіслав роутер, та часом наступних вимірів на рис. 5.14 складає 3 секунди, що відповідає дуже довгій тривалості глибокого сну.

```

Fuzzy | Decision 6: battery - 1, memory - 0, delta - 2, st - 0
Current state: sensor - 0.63, timestamp - 1675873069, battery - 2.79, memory state - 10
Decision 6
Msg: sensor - 0.63, timestamp - 1675873069, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675873069
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Fuzzy | Decision 6: battery - 1, memory - 0, delta - 2, st - 0
Current state: sensor - 0.63, timestamp - 1675873072, battery - 2.79, memory state - 10
Decision 6
Msg: sensor - 0.63, timestamp - 1675873072, battery - 3.30%, memory state - 508
Sent message to: {40:22:D8:7A:35:04};

Last Packet Send Status:      Delivery Success

Time obtained: 1675873073
Peer: {40:22:D8:EB:29:4C}, status: 00
Response time: 2
ets Jul 29 2019 12:21:46

```

Рисунок 5.14 - Сценарій 8

На рис. 5.15 наведено тестування за сценарієм 9 – оновлення локального списку роутерів. Для даної точки встановлено список з двох роутерів-сусідів – роутер з MAC-адресою 40:22:D8:EB:29:4C (далі R1) та 40:22:D8:7A:35:04 (далі R2). Роутер R1 є для точки більш пріоритетним, але на момент відправлення даних позначений статусом 0 – тобто недоступний для підключення. Таким чином, точка відправляє виміри на роутер R2. На рис. 5.15 можна побачити, що при наступній відправці даних роутер R2 надіслав інформацію стосовно статусу 1 для роутеру R1 – тобто роутер R1 було підключено до мережі.

```

Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.58, timestamp - 1675871902, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.58, timestamp - 1675871902, battery - 3.30%, memory state - 508
Sent message to: [40:22:D8:7A:35:04];

Last Packet Send Status:      Delivery Success
Archive was sent

Time obtained: 1675871902
Peer: [40:22:D8:EB:29:4C], status: 00
Response time: 0
ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SDI_FAST_FLASH_BOOT)
configsip: 0, SPIWD:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3ffff000, len:1344
load:0x40078000, len:13516
load:0x40080400, len:3604
entry 0x400805f0

Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.58, timestamp - 1675871903, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.58, timestamp - 1675871903, battery - 3.30%, memory state - 508
Sent message to: [40:22:D8:7A:35:04];

Last Packet Send Status:      Delivery Success
Archive was sent

Time obtained: 1675871903
Peer: [40:22:D8:EB:29:4C], status: 01
Response time: 1
ets Jul 29 2019 12:21:46

```

Рисунок 5.15 - Оновлення списку статусів роутерів

На рис. 5.16 наведено скріншот відправки даних на роутер R1, що свідчить про успішне оновлення локального списку кінцевої точки та виконання сценарію тестування 10.

```

Fuzzy | Decision 2: battery - 4, memory - 4, delta - 4, st - 4
Current state: sensor - 0.61, timestamp - 1675871904, battery - 3.20, memory state - 507
Decision 2
Msg: sensor - 0.61, timestamp - 1675871904, battery - 3.30%, memory state - 508
Sent message to: [40:22:D8:EB:29:4C];

Last Packet Send Status:      Delivery Success
Archive was sent

Time obtained: 1675871905
Peer: [40:22:D8:7A:35:04], status: 01
Response time: 1
ets Jul 29 2019 12:21:46

```

Рисунок 5.16 - Відправка даних на роутер R1

У табл. 5.1 наведено чекліст тестування кінцевої точки. У описі сценаріїв 1-8 наведено номер рішення контролера нечіткої логіки, якому відповідають значення параметрів кінцевої точки та, за наявності, додаткові умови.

Таблиця 5.1 - Чекліст тестування кінцевої точки

№	Умови					Результат
	Опис сценарію	Параметри кінцевої точки				
		Заряд автономного джерела живлення	Стан заповненості пам'яті	Величина зміни даних	Час останньої відправки даних	
1	2	3	4	5	6	7
1	Рішення 1	Високий	Майже незаповнена	Дуже мала	Щойно	Пройдено
2	Рішення 2; у пам'яті пристрою відсутні записи, у мережі є доступні для підключення роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
3	Рішення 2; у пам'яті пристрою є записи, у мережі є доступні для підключення роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
4	Рішення 2; у мережі відсутні доступні для підключення роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено

Продовження таблиці 5.1

1	2	3	4	5	6	7
5	Рішення 3	Середній	Майже незаповнена	Дуже мала	Не дуже давно	Пройдено
6	Рішення 4	Середній	Середня заповненість	Дуже мала	Не дуже давно	Пройдено
7	Рішення 5	Низький	Середня заповненість	Дуже мала	Дуже давно	Пройдено
8	Рішення 6	Низький	Майже повністю заповнена	Середня зміна	Дуже давно	Пройдено
9	Оновлення локального списку роутерів кінцевої точки при отриманні даних від роутера	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
10	Підключення кінцевої точки до різних роутерів в залежності від їх статусу доступності до підключення	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено

Тестування роботи роутерів та центрального шлюза. Під час роботи роутери, як і кінцеві точки, виводять інформацію у СОМ-порт. Роутери виводять інформацію стосовно статусу відправленого пакету та отримані від кінцевих точок дані. На рис. 5.17 наведено скріншот роботи роутера при виконанні сценарію 1 – під час отримання вимірів від кінцевої точки.

```
Last Packet Send Status:      Delivery Success
OnDataRecv: Remote readings: sensor - 0.69, measured at - 1676043853, battery - 3.30, memory state - 508
Friday, February 10 2023 17:44:14

Last Packet Send Status:      Delivery Success
Router 0x40:0x22:0x08:0x7A:0x35:0x04 is 0x01
```

Рисунок 5.17 - Отримання вимірів від кінцевої точки (сценарій 1)

При отриманні даних від кінцевої точки, роутер у відповідь надсилає поточний час у форматі unix-часу та свій список роутерів-сусідів з відповідними статусами доступності, що відповідає успішному проходженню сценарію тестування 3.

Окрім того, при оновленні списку роутерів-сусідів роутер виводить у СОМ-порт сусідів зі списку з указівкою їх статусу.

На рис. 5.18 наведено скріншот роботи роутера під час отримання архіву від кінцевої точки, що відповідає роботі за сценарієм 2.

```
Last Packet Send Status:      Delivery Success
OnDataRecv: Remote readings: sensor - 0.68, measured at - 1, battery - 3.30, memory state - 496
Friday, February 10 2023 17:51:13

Last Packet Send Status:      Delivery Success
OnDataRecv: Remote readings: sensor - 0.69, measured at - 1, battery - -1.00, memory state - -1
OnDataRecv: Remote readings: sensor - 0.69, measured at - 2, battery - -1.00, memory state - -1
OnDataRecv: Remote readings: sensor - 0.70, measured at - 3, battery - -1.00, memory state - -1
Router 0x40:0x22:0x08:0x7A:0x35:0x04 is 0x01

Last Packet Send Status:      Delivery Success
Router 0x40:0x22:0x08:0x7A:0x35:0x04 is 0x01
```

Рисунок 5.18 - Отримання вимірів від кінцевої точки (сценарій 2)

Так як кінцева точка при записі не зберігає дані стосовно заряду автономного джерела живлення та стану заповненості пам’яті, значення цих змінних у повідомленні дорівнює -1. Це дозволяє зберегти уніфікований вигляд повідомлень, а також дає можливість швидко розпізнати «архівні» повідомлення.

На рис. 5.19-5.20 наведено дані, отримані центральним шлюзом та виведені у СОМ-порт. Це відображає роботу за сценаріями 4-5.

```
-----
Router 1 | Endpoint: E2 - Luminosity sensor
Sensor - 0.72, measured at - 1676928294, battery - 3.30, memory state - 500
-----
-----
Router 2 | Endpoint: E1 - Thermo sensor
Sensor - 0.43, measured at - 1, battery - 3.30, memory state - 464
-----
```

Рисунок 5.19 - Отримання даних від роутерів (1)

```
-----
Router 2 | Endpoint: E2 - Luminosity sensor
Sensor - 0.74, measured at - 1676929338, battery - 3.30, memory state - 500
-----
-----
Router 1 | Endpoint: E3 - Hall sensor
Sensor - 30.00, measured at - 1676929340, battery - 3.30, memory state - 508
-----
-----
Router 2 | Endpoint: E2 - Luminosity sensor
Sensor - 0.73, measured at - 1676929341, battery - 3.30, memory state - 500
-----
```

Рисунок 5.20 - Отримання даних від роутерів (2)

У табл. 5.2 наведено чекліст тестування роботи роутерів та центрального шлюза.

Таблиця 5.2 - Чекліст тестування роботи роутерів та центрального шлюза

№	Опис сценарію	Результат
1	2	3
1	Отримання роутером вимірів від кінцевої точки	Пройдено

Продовження таблиці 5.2

1	2	3
2	Отримання роутером архіву кінцевої точки	Пройдено
3	Відправлення роутером локального списку роутерів-сусідів та поточного часу кінцевій точці при отриманні даних від неї	Пройдено
4	Відправлення роутером даних, отриманих від кінцевої точки, на центральний шлюз	Пройдено
5	Отримання центральним шлюзом даних від роутерів та виведення їх на екран	Пройдено

5.3 Порівняння енергоефективності модуля при використанні звичайної затримки і режиму `deep_sleep`

Використання режиму глибокого сну. Була написана проста програма для циклічного опитування датчика та надсилення повідомлення за допомогою методів ESP-NOW.

Наприкінці основного циклу `loop()` команда `delay(100)` і режим `deep_sleep` були використані для порівняння енергоефективності (таймер було встановлено на 100 000 мікросекунд).

Споживана потужність модуля вимірювалася за допомогою осцилографа як падіння напруги на резисторі, підключеному послідовно з модулем до джерела живлення. Для зменшення імпульсного шуму в якості джерела живлення використовувався акумулятор на 12 вольт.

Загальну схему вимірювання наведено на рис. 5.21.



Рисунок 5.21 - Загальна схема вимірювання

Номінал вимірювального резистора - 0.33 ом, це відповідає 3 мА струму, що протікає, на 1 мВ падіння напруги.

Загальний вигляд графіку наведено на рис. 5.22. Після виходу з режиму сну процесор довго завантажується (0..65 мс), потім (65..85 мс) виконується програма (включаючи зчитування стану датчика), різкий сплеск (85мс..105мс) – вмикання WiFi модему та прийом/передача даних.

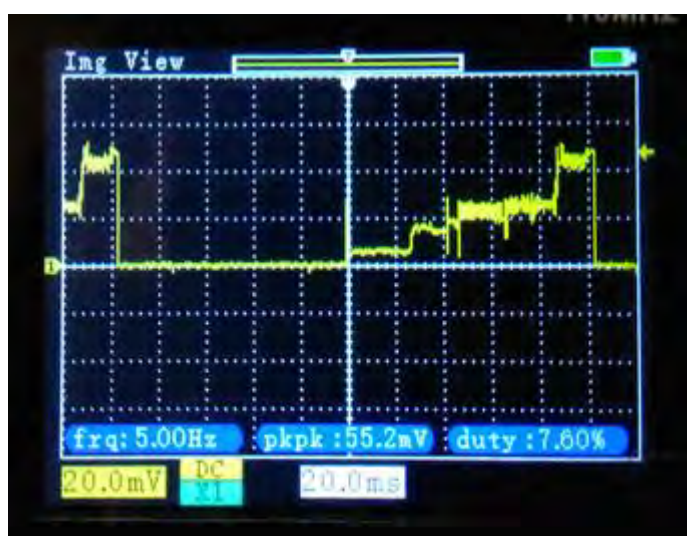


Рисунок 5.22 - Загальний вигляд графіка

Середньоквадратичне споживання (RMS) за період 105 мс – 22 мВ (66 мА). Зразок, наведений на рис. 5.23, включає всі частини робочого циклу: завантаження, опитування датчика та активна робота Wi-Fi. На жаль, вибірка

осцилографа включала дві невеликі ділянки до (5 мс) і після (10 мс), але їх вплив на якісне порівняння режимів не настільки істотний.



Рисунок 5.23 - Характеристики графіка

Графік площі глибокого сну показано на рис. 5.24. Напруга надзвичайно низька для візуальної ідентифікації. Осцилограф не зміг достовірно визначити падіння напруги на резисторі (макс. +800 мкВ, мін. -1,6 мВ, коливання навколо нуля).



Рисунок 5.24 - Графік зони глибокого сну

Мультиметр також не зміг відрізнити струм споживання модуля без процесора (власний струм споживання регулятора 3 мА = падіння 1 мВ) від

процесора в режимі глибокого сну, принаймні струм не більше, ніж роздільна здатність мультиметра, $0.01 \text{ мВ} = 30 \text{ мкА}$. Відповідно до документації [36] споживання в режимі глибокого сну становить близько 10 мкА , що узгоджується з експериментом.

При опитуванні датчиків з періодом T секунд середнє споживання модуля буде дорівнювати:

$$I_{\text{ср}} = \frac{66 \cdot 0.12 + 0.01 \cdot T}{T + 0.12} [\text{мА}] \quad (5.1)$$

При $T=0\text{с}$ (активний цикл триває $100\text{-}120 \text{ мс}$) середнє споживання становитиме 66 мА .

При $T \gg 800 \text{ с}$ середній струм споживання зменшується до струму глибокого сну.

Використання затримки. Енергоспоживання без режиму глибокого сну помітно вище (рис. 5.25-5.26). Середньоквадратичне споживання за повний цикл із затримкою 100 мс становить приблизно $41,5 \text{ мВ}$ ($124,5 \text{ мА}$).

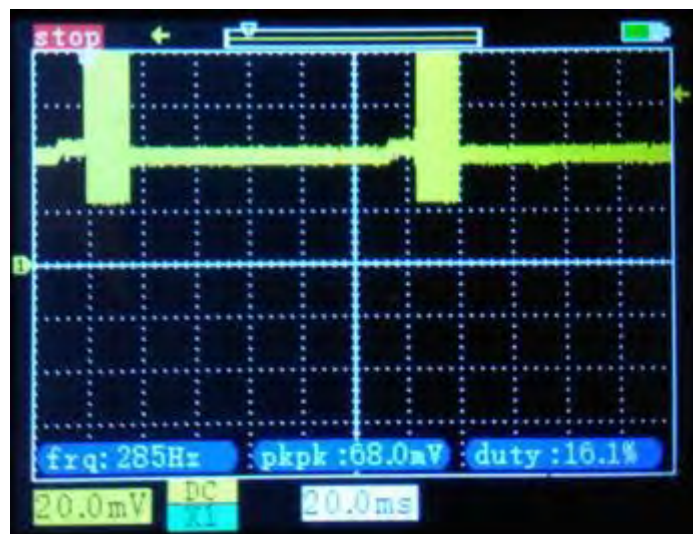


Рисунок 5.25 - Активний режим із затримкою 100 мс

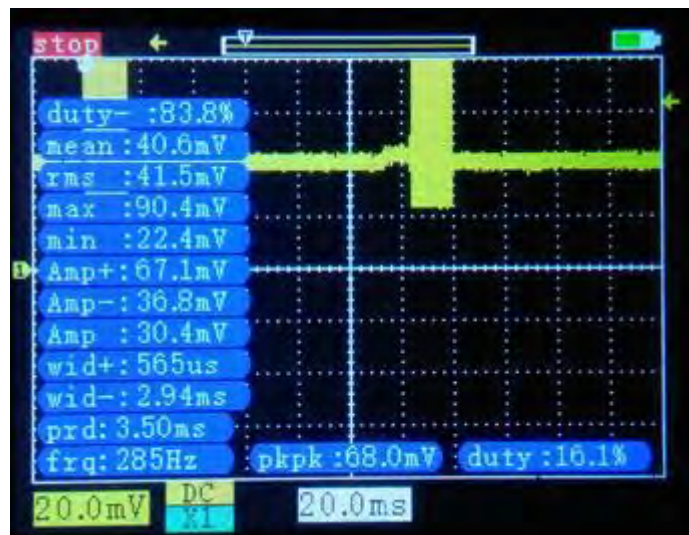


Рисунок 5.26 - Середньоквадратичне споживання

На рис. 5.27 показана зона активної роботи Wi-Fi (нуль зміщений у нижню частину дисплея). Середнє споживання – 60 мВ (180 мА), час роботи Wi-Fi – близько 15 мс

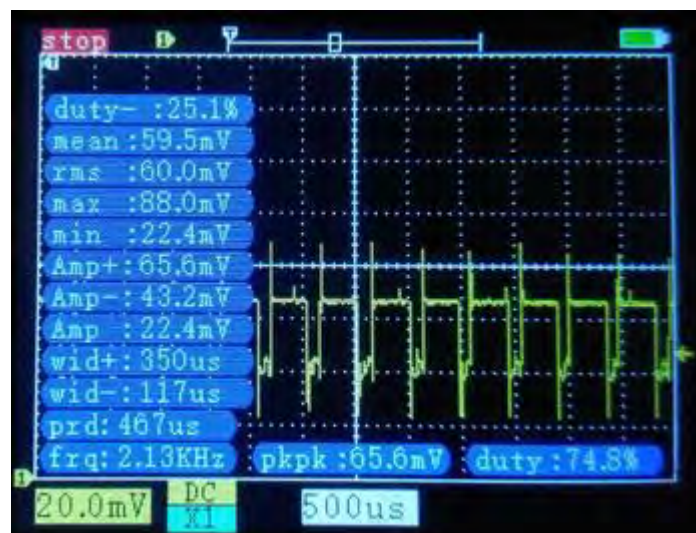


Рисунок 5.27 - Зона активної роботи Wi-Fi

Енергоспоживання під час опитування датчика (10 мс) і очікування (100 мс) практично однакове. Середнє споживання – 37 мВ (111 мА) (рис. 4.32).



Рисунок 5.28 - Споживання енергії під час очікування

При опитуванні датчиків з періодом часу T секунд середнє споживання модуля буде дорівнювати:

$$I_{cp} = \frac{180 \cdot 0.015 + 111 \cdot (T + 0.01)}{T + 0.01 + 0.015} \text{ [mA]} \quad (5.2)$$

При $T=0$ с (активний цикл становить 15 мс, що набагато швидше, ніж у режимі глибокого сну) струм споживання становитиме 152 мА.

При $T \gg 60$ с середній струм споживання зменшується до струму споживання в режимі очікування (111 мА) і не зменшується.

Порівняння енергоспоживання. Середнє споживання струму було розраховано для $T=60$ с і $T=15$ хв з використанням формул (1) і (2).

Для $T=60$ с середнє споживання струму для режиму затримки становить 111,02 мА, для режиму глибокого сну – 0,1253 мА.

Для $T=15$ хв = 900 с середнє споживання струму для режиму затримки становить 111 001 мА, для режиму глибокого сну – 0,0177 мкА.

5.4 Приблизна оцінка енергоефективності розробленого рішення

Був проведений експеримент для оцінки енергозбереження розробленого алгоритму. Для цього було налаштовано кілька точок таким чином, щоб одна використовувала вдосконалений алгоритм, а друга працювала за традиційним підходом. Тривалість сплячого режиму для обох точок становить 3 секунди.

Точка, що працювала за традиційним підходом, працювала 85 годин (3 доби 13 годин), після чого заряду автономного джерела живлення стало недостатньо для відправки даних.

Точка, що використовувала розроблений алгоритм, працювала 260 годин (10 діб 20 годин).

За 85 годин роботи експериментальної установки точка, що працювала за традиційним підходом, зробила 94859 відправки даних, тоді як точка, що використовувала поліпшений алгоритм – 2400 відправки.

Як свідчать отримані дані, вдосконалений алгоритм дозволяє підвищити ефективність використання батареї та уникнути отримання та зберігання зайвої інформації.

На рис. 5.29 наведено графік розряду точок експериментальної установки.

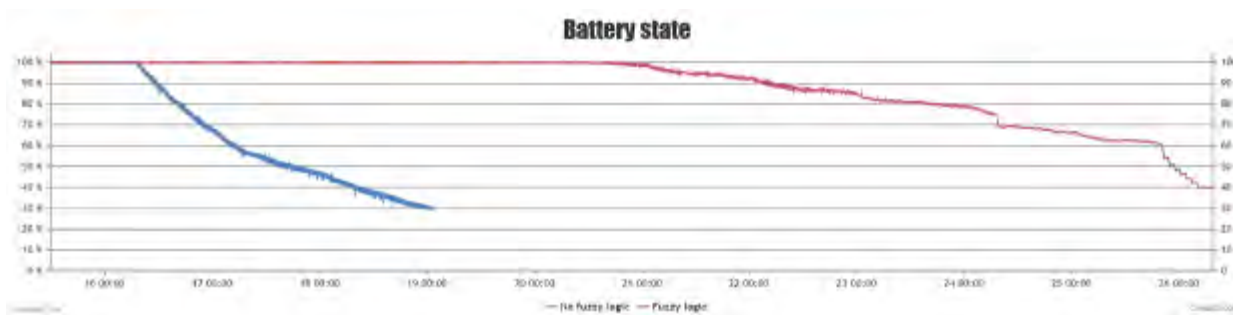


Рисунок 5.29 - Графік розряду точок експериментальної установки

5.5 Висновки до розділу 5

У цьому розділі було описано сценарії тестування прототипу розробленої БСМ. Результати тестування представлені у вигляді чеклістів тестування для кінцевої точки та для роутера і центрального шлюза.

Було наведено результати практичного експерименту для порівняння енергоефективності модуля при використанні звичайної затримки і режиму `deep_sleep`. За результатами експерименту було доведено ефективність використання режиму глибокого сну – при використанні затримки середнє енергоспоживання 111 001 мА, а при використанні режиму глибокого сну – 0,0177 мкА.

Окрім того, було доведено ефективність використання розробленого алгоритму роботи кінцевої точки. Для цього було налаштовано кілька точок таким чином, щоб одна використовувала вдосконалений алгоритм, а друга працювала за традиційним підходом. Заряду точки, що працювала за традиційним підходом, вистачило на 85 годин, тоді як заряду точки, що використовувала розроблений алгоритм – на 260 годин.

ВИСНОВКИ

У даній роботі було розроблено ПЗ БСМ на основі удосконаленого алгоритму збору та передачі даних для мінімізації енергоспоживання системи. За основу архітектури було взято гібридну архітектуру i-Beans, в якості протоколу зв'язку між вузлами було використано ESP-NOW. Було розроблено прототип БСМ та проведено тестування, що підтвердило працездатність та ефективність розробки.

Розроблене програмне рішення оптимізує процес роботи кінцевої точки до відправки даних за рахунок використання нечіткої логіки. Відповідно до розроблених алгоритмів, послідовність дій кінцевої точки коригується в залежності від її стану та зміни показників навколишнього середовища. Зокрема, мінімізується кількість підключень до мережі у тих випадках, коли відсутня необхідність (наприклад, коли виміри сенсору змінилися незначно) або якщо заряд автономного джерела живлення низький. Подібний підхід дозволить зменшити енергоспоживання кінцевих точок та продовжити термін їх роботи, а також значно знизить кількість надлишкової інформації, яку необхідно передавати та зберігати. Це зменшить навантаження на мережу та зменшить вимоги до обладнання баз даних, що покращить роботу мережі в цілому та може знизити собівартість обладнання, так як не буде необхідності використовувати дороге устаткування.

Наукова новизна роботи полягає в тому, що алгоритм збору та передачі даних БСМ удосконалено за рахунок використання нечіткої логіки, що на відміну від існуючих підходів дозволяє не тільки визначити оптимальний шлях передачі даних, але й мінімізувати енергоспоживання.

Практична цінність роботи полягає в тому, що розроблене програмно-апаратне рішення може бути використане для побудови стаціонарних БСМ локальної зони покриття, зокрема, для систем «Розумний будинок» та «Розумна будівля», що дозволить контролювати використання ресурсів та заощаджувати кошти.

У подальшій роботі планується перевірити роботу БСМ, яка складається з великої кількості кінцевих точок, а також розглянути можливості використання альтернативних джерел живлення.

Результати роботи доповідалися та обговорювалися на наукових конференціях в Україні та за кордоном та опубліковані в наступних публікаціях:

- Methods and tools for minimizing the power consumption of wireless sensor network / [A. Yatsenko, C. Wolff, A. Parkhomenko et. al.] 2023 IEEE European technology & engineering management summit (ETEMS), Kaunas, Liuania, 20-22 April 2023: proceedings. – Los Alamitos: IEEE, 2023. – P. 1–6;

- Яценко, А.К. Мінімізація енергоспоживання бездротової сенсорної мережі на основі програмних і апаратних рішень / А.К. Яценко, А.В. Пархоменко, А.В. Туленков // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : XVI Міжнародна науково-практична конференція, Запоріжжя, 12–14 грудня 2022 р.: тези доповідей. – Запоріжжя: НУЗП, 2022. – С. 201–202;

- Яценко, А.К. Дослідження методів та засобів розробки енергоефективних бездротових сенсорних мереж / А.К. Яценко, А.В. Пархоменко // Тиждень науки-2023: Щорічна науково-практична конференція викладачів, науковців, молодих учених, аспірантів та студентів, Запоріжжя, 24-28 квітня 2023 р.: тези доповідей. – Запоріжжя: НУЗП, 2023. – С.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kumar, P. Smart neighborhood to enhance social sustainability and inclusive planning in smart cities / P. Kumar // National Conference on Sustainable built environment, Roorkee, 10-12 April 2015. – Roorkee: Indian Institute of Technology, 2015. – P. 1-11.
2. Lin, D. The game theory: applications in the wireless networks / D. Lin, W. Quan, P. Yang // Game Theory - Applications in Logistics and Economy / ed. by D. Tuljak-Suban. – London, 2018. - P. 1-16
3. Hossam, M. A. F. Concepts, Applications, Experimentation and Analysis of Wireless Sensor Networks/ M. A. F., Hossam. – Cham: Springer, 2021. - 716 p.
4. A Survey on Sensor Networks / [I. F. Akyildiz, W. Su; Y. Sankarasubramaniam et al.] // IEEE Communications Magazine. – 2002. – №40. - P.102-114.
5. Basics of Wireless Sensor Networks (WSN) | Classification, Topologies, Applications [Electronic resource]. – Access mode: <https://www.electronicshub.org/wireless-sensor-networks-wsn/>.
6. Енциклопедія кібернетики: [у 2 т.] / редкол.: В. М. Глушков (відп. ред) [та ін.]; АН Української РСР. - Київ : Голов. ред. Укр, рад. енцикл., Т.1, 1973. - 582 С.
7. Sohraby, K. Wireless sensor networks: technology, protocols, and applications / K. Sohraby, D. Minoli, T. Znati. – Hoboken: John Wiley & Sons, Inc., 2007. – 307 p.
8. Rhee, S. Techniques for Minimizing Power Consumption in Low Data-Rate Wireless Sensor Networks / S. Rhee, D. Seetharam, S. Liu // Wireless Communications and Networking Conference, Atlanta, 21-25 March 2004. – Piscataway: IEEE, 2004. – P. 1727–1737.
9. Dargie, W. Fundamentals of wireless sensor networks : theory and practice / W. Dargie, C. Poellabauer. – Chichester: John Wiley & Sons, Inc., 2010. - 311 p.

10. Cetinkaya, O. Use of Wireless Sensor Networks in Smart Homes / O. Cetinkaya, O. B. Akan // Emerging Communication Technologies Based on Wireless Sensor Networks / ed. by M. H. Rehmani, A. K. Pathan. – Boca Raton, 2016. – P. 233-259.
11. Zheng, J. Wireless sensor networks. A Networking Perspective / J. Zheng, A. Jamalipour. – Hoboken: John Wiley & Sons, Inc., 2009. – 489 p.
12. Elson, J. Fine-grained network time synchronization using reference broadcasts / J. Elson, L. Girod, D. Estrin // Operating Systems Design and Implementation, Boston, 9-11 December 2002 : proceedings. – Berkeley: USENIX, 2002. – P. 147–163.
13. Manikandan, A. Optimized Group Channel Assignment Using Computational Geometry over Wireless Mesh Networks / A. Manikandan, Y. Palanichamy // Mobile Information Systems. – 2015. – P. 1-18.
14. Ferrara, J. How do Bluetooth devices work? [Electronic resource] / J. Ferrara. – Access mode: <https://scienceline.org/2016/04/how-do-bluetoothdevices-work/>.
15. Franklin, C. How Bluetooth Works [Electronic resource] / C. Franklin. – Access mode: <https://electronics.howstuffworks.com/bluetooth.htm>.
16. Zhou, Y. Ultra low-power UWB-RFID system for precise location-aware applications / Y. Zhou, C. L. Law and J. Xia. // Wireless Communications and Networking Conference Workshops (WCNCW), Paris, 01-01 April 2012: proceedings. – Piscataway: IEEE, 2009. - P. 154–158.
17. Thomas, J. The history of Wi-Fi [Electronic resource] / J. Thomas. – Access mode: <https://purple.ai/blogs/history-wifi/>.
18. Prasanna. Zigbee Technology Advantages and Disadvantages | ZigBee Technology Architecture and Its Applications [Electronic resource] / Prasanna – Access mode: <https://www.aplustopper.com/zigbee-technology-advantages-and-disadvantages/>.
19. ESP-NOW protocol [Electronic resource]. – Access mode: <https://www.espressif.com/en/products/software/esp-now/overview>.

20. Spreading factors [Electronic resource]. – Access mode: <https://www.thethingsnetwork.org/docs/lorawan/spreading-factors>.
21. Spiess, A. #172 Hidden: ESP32 and ESP8266 point-to-point (ESP-Now): Fast and efficient. Comparison with LoRa [Electronic resource] / A. Spiess. – Access mode: <https://www.youtube.com/watch?v=6NsBN42B80Q>.
22. Croce, S. Reducing Power Consumption in Wireless Sensor Networks Using a Novel Approach to Data Aggregation / S. Croce, F. Marcelloni, M. Vecchio // The Computer Journal. – 2008. – №2. – P. 227–239.
23. Energy consumption in wireless sensor network / [J. T. Ogbiti, H. C. Ukwuoma, S. Danjuma et al.] // Computer Engineering and Intelligent Systems. – 2016. – №8. – P. 63–67.
24. Rokach, L. Decision Trees / L. Rokach, O. Maimon // The Data Mining and Knowledge Discovery Handbook / ed. by O. Maimon, L. Rokach – NY, 2005. – P. 165-192.
25. Applying genetic algorithms to decision making in autonomic computing systems / [A. J Ramirez., D. B. Knoester, B. H. C. Cheng et al.] // 6th International Conference on Autonomic Computing, Barcelona, 15-19 June 2009 : proceedings. – NY: Association for Computing Machinery, 2009. - P. 97–106.
26. Fuzzy Logic-based Routing Algorithm for Lifetime Enhancement in Heterogeneous Wireless Sensor Networks / [R. Al-Kiyumi, C. H. Foh, S. Vural et al.] // IEEE Transactions on Green Communications and Networking. – 2018. – Vol. 2, № 2. – P. 517–532.
27. Tyagi, N. A Fuzzy-Logic Approach In Decision-Making [Electronic resource] / N. Tyagi. – Access mode: <https://www.analyticssteps.com/blogs/fuzzy-logic-approach-decision-making>.
28. Valášková, K. The Role of Fuzzy Logic in Decision Making Process / K. Valášková, T. Klieštík, M. Mišánková // Management Innovation and Business Innovation : 2nd International Conference, Bangkok, 08-09 December 2014: proceedings. – Singapore: Singapore management & sports science Inst PTE LTD, 2014. – P. 143-148.

29. ESP32 Series of Modules [Electronic resource]. – Access mode: <https://www.espressif.com/en/products/modules/esp32>.

30. Santos S., ESP32 vs ESP8266 – Pros and Cons [Electronic resource] / S. Santos. – Access mode: <https://makeradvisor.com/esp32-vs-esp8266>.

31. Plauska, I. Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller / I. Plauska, A. Liutkevicius, A. Janaviciute // Electronics. – 2023. Vol. 12, № 1. – P. 143.

32. Energy efficiency across programming languages: how do energy, time, and memory relate? / [R. Pereira, M. Couto, F. Ribeiro et al.] // Software language engineering: The 10th ACM SIGPLAN International conference, Vancouver, 23-24 October 2017: proceedings. – NY: Association for computing machinery, 2017. – P. 256-267.

33. ESP-IDF [Electronic resource]. – Access mode: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/>.

34. PlatformIO [Electronic resource]. – Access mode: <https://platformio.org/platformio-ide>.

35. Arduino IDE [Electronic resource]. – Access mode: <https://www.arduino.cc/en/software>.

36. ESP32 Series Datasheet [Electronic resource]. – Access mode: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

ДОДАТОК А
Текст програми

A.1 Текст програми кінцевих точок

```

/**
linguistic variables - very negative (0),
                        negative(1),
                        zero(2),
                        positive(3),
                        very positive(4)

parameters:
battery state (b)
memory state (m)
when was the last msg sent (silent time)(st)
how much the data changed since the last sending - delta (d)
R1 40:22:d8:eb:29:4c
R2 40:22:d8:7a:35:04
E1 c8:f0:9e:30:91:54
E2 c8:f0:9e:30:9d:4c
E3 c8:f0:9e:30:97:7c
*/

bool ok;

#include "fuzzy.h"
//#include "decisions.h"

#include <esp_now.h>
#include <WiFi.h>
#include "time.h"
#include <Preferences.h>

Preferences preferences;

// The recipient MAC address. It must be modified for each device.
#define WIFIESPNOW_ALEN 6

static uint8_t selfMAC[] = {0,0,0,0,0,0}; // own MAC

const uint8_t peerLength = 6; // MAC-address length
// router MACs
static uint8_t PEERS[][peerLength]{
    {0x40, 0x22, 0xD8, 0xEB, 0x29, 0x4C}, // R1
    {0x40, 0x22, 0xD8, 0x7A, 0x35, 0x04} // R2
};
//statuses of peers must be in the same order as in PEERS!
RTC_DATA_ATTR uint8_t ALIVE_PEERS[] {1,1};
const int PEERS_COUNT = 2;
RTC_DATA_ATTR unsigned long counter = 0; //real time

static uint8_t channel = 0;

const int sensorDataPin = 32; // sensor i/o data pin
const int sensorPowerPin = 33; // pin for powering sensor

```

```

const int batterySensorPin = 34; // battery voltage read pin
int sent_status = 0; // 0 - undef, 1 - ok, 2 - fail
int recv_status = 0; // 0 - undef, 1 - ok, 2 - fail

const int DURATION_SLEEP_NORMAL = 1;
const int DURATION_SLEEP_LONG = 2;
const int DURATION_SLEEP_VERY_LONG = 3;
RTC_DATA_ATTR int duration_sleep = DURATION_SLEEP_NORMAL;

float battery=0;

int count_alive_peers(){
    int count_peers = 0;
    for (int i=0; i<PEERS_COUNT; i++){
        if (ALIVE_PEERS[i] == 1){
            count_peers++;
        }
    }
    return count_peers;
}

void make_all_peers_alive(){
    for (int i=0; i<PEERS_COUNT; i++){
        ALIVE_PEERS[i] = 1;
    }
}

int decision_1() {
    // Don't save or send data
    duration_sleep = DURATION_SLEEP_NORMAL;
    return 0;
}

int decision_2() {
    // Send (if the are data in the memory, send data)
    get_msg_text();
    duration_sleep = DURATION_SLEEP_NORMAL;
    send_message(true);
    return 0;
}

int decision_3() {
    // write the new measurment to the memory
    write_data();
    duration_sleep = DURATION_SLEEP_NORMAL;
    return 0;
}

int decision_4() {
    //send new measurment and notification about memory (50% are
    occupied)
    get_msg_text();
    duration_sleep = DURATION_SLEEP_NORMAL;

```

[illegible]

```

        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
    }; // 6 bytes for MAC, 7th for alive status
};

msg_data msg;
sensor_data new_data;
RTC_DATA_ATTR sensor_data last_sent_data; //need to save
int num_of_values_in_flash = 0; //need to save

void get_sensor_data(uint8_t delay_time=0) {
    sensorPowerOn();
    if(delay_time>0) {
        delay(delay_time);
    }
    new_data.x = float(hallRead());
    new_data.measured_at = counter;
    sensorPowerOff();
}

//COMMUNICATION AND DATA GATHERING
void get_msg_text() {
    int m = get_memory_value();
    msg.sensor_data = new_data.x;
    msg.measured_at = new_data.measured_at;
    msg.battery = battery; // global var
    msg.memory_state = m;
}

float get_battery_value() {
// float analogValue = 0;
    for (int i=0; i<100; i++) {
        battery += float(analogRead(batterySensorPin));
        delay(1);
    }
    battery /= 100;
    Serial.printf("Battery: %7.2f ",battery);
    battery = (battery-3845)/250*100;
    Serial.printf("= %7.2f\n",battery);
    return battery;
}

int get_memory_value() {
    preferences.begin("archive", true);
    size_t whatsLeft = preferences.freeEntries(); //0-507(?)
    preferences.end();
    return whatsLeft;
}

float get_delta_value() {
    return abs(new_data.x - last_sent_data.x)/abs(last_sent_data.x);
}

```

```

float get_silent_time_value() {
    return new_data.measured_at - last_sent_data.measured_at;
}

void write_data(){
    preferences.begin("archive", false);

    char cstr[11];
    itoa(new_data.measured_at, cstr, 10);
    preferences.putFloat(cstr, new_data.x);
    float a = preferences.getFloat(cstr);

    if (preferences.isKey("keys")) {
        size_t schLen = preferences.getBytes("keys", NULL, NULL);
        char buffer[schLen];
        preferences.getBytes("keys", buffer, schLen);
        unsigned long *key_array = (unsigned long *) buffer;
        int key_len = sizeof(buffer) / sizeof(unsigned long);
        unsigned long new_key_array[key_len+1];
        for (int i=0; i<key_len; i++){
            new_key_array[i] = key_array[i];
        }
        new_key_array[key_len] = new_data.measured_at;
        preferences.putBytes("keys", &new_key_array,
sizeof(new_key_array));
    }
    else {
        unsigned long key_array[] = {new_data.measured_at};
        preferences.putBytes("keys", &key_array, sizeof(key_array));
    }

    preferences.end();
}

void send_message(bool send_archive){
    bool sent = false;
    int i = 0;
    while (!sent && i<PEERS_COUNT)
    {
        sent_status = 0;
        if (ALIVE_PEERS[i] == 1){
            esp_err_t result1 = esp_now_send(PEERS[i], (uint8_t *) &msg,
sizeof(msg)); //FROM EXAMPLE
            Serial.printf("Msg: sensor - %5.2f, timestamp - %lu, battery -
%5.2f, memory state - %d\n", msg.sensor_data, msg.measured_at,
msg.battery, msg.memory_state);
            if (result1 == ESP_OK) {
                Serial.printf("Sent message to:
{%02X:%02X:%02X:%02X:%02X:%02X};\n", PEERS[i][0], PEERS[i][1],
PEERS[i][2], PEERS[i][3], PEERS[i][4], PEERS[i][5]);
                int time_count = 0;
                while (sent_status == 0 && time_count++ <= 20){delay(1);}
                if (sent_status == 1){

```

```

last_sent_data.x = new_data.x;
last_sent_data.measured_at = new_data.measured_at;
sent=true;

if (send_archive){
  preferences.begin("archive", false);
  if (preferences.isKey("keys")) {
    size_t schLen = preferences.getBytes("keys", NULL,
NULL);

    char buffer[schLen];
    preferences.getBytes("keys", buffer, schLen);
    unsigned long *key_array = (unsigned long *) buffer;
    int key_len = sizeof(buffer) / sizeof(unsigned long);
    Serial.printf("key_len=%d : ",key_len);
    for (int i=0; i<key_len; i++){
      Serial.printf("[%d]=%d ",i,key_array[i]);
    }

    for (int i=0; i<key_len; i++){
      char cstr[11];
      itoa(key_array[i], cstr, 10);
      float          archive_meurment          =
preferences.getFloat(cstr);
      msg_data archive_data;
      archive_data.sensor_data = archive_meurment;
      archive_data.measured_at = key_array[i];
      archive_data.battery = -1;
      archive_data.memory_state = -1;
      esp_err_t result2 = esp_now_send(PEERS[1], (uint8_t
*) &archive_data, sizeof(archive_data));
      Serial.printf("Archive   msg:   sensor   -   %5.2f,
timestamp      -      %lu\n",          archive_data.sensor_data,
archive_data.measured_at);
    }
    Serial.println("Archive was sent");
  }
  preferences.clear();
  preferences.end();
}

time_count = 0;
while (recv_status == 0 && time_count++ <= 5000){delay(1);}
Serial.printf("Response time: %d\n", time_count);
}
else if (sent == 2){
  ALIVE_PEERS[i]=0;
}
}
else {
  Serial.println("Error sending the data");
}
}
i += 1;

```

```

    }
    if (!sent){
        write_data();
        Serial.println("No router available; Saved data instead");
    }
}

//CONTROLLERS
// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t
status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success"
: "Delivery Fail");
    status == ESP_NOW_SEND_SUCCESS ? sent_status=1 : sent_status=2;
}

int local_peer_number(uint8_t *peer) {
    int match_count;
    for (int i=0; i<PEERS_COUNT; i++){
        match_count = 0;
        for (int j=0; j<6; j++){
            if (peer[j] == PEERS[i][j]){
                match_count++;
            }
        }
        if (match_count==6){
            return i;
        }
    }
    return -1;
}

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int
len) {
    router_data incomingReadings;
    memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
    counter = incomingReadings.obtained_time;
    Serial.println("");
    Serial.printf("Time                obtained:                %d\n",
incomingReadings.obtained_time);
    for (int i=0; i<incomingReadings.peer_count; i++){

        uint8_t                peer[6]{incomingReadings.peers[i][0],
incomingReadings.peers[i][1],                incomingReadings.peers[i][2],
incomingReadings.peers[i][3],                incomingReadings.peers[i][4],
incomingReadings.peers[i][5]};
        int i_match = local_peer_number(peer);
        if (i_match >= 0){
            ALIVE_PEERS[i] = incomingReadings.peers[i][6];
        }
    }
}

```

```

        Serial.printf("Peer:   {%02X:%02X:%02X:%02X:%02X:%02X},   status:
%02X\n", incomingReadings.peers[i][0], incomingReadings.peers[i][1],
incomingReadings.peers[i][2],           incomingReadings.peers[i][3],
incomingReadings.peers[i][4],           incomingReadings.peers[i][5],
incomingReadings.peers[i][6]);
    }
    recv_status = 1;
}

bool addPeer(const uint8_t mac[6]) {
    esp_now_peer_info_t pi{};
    static_assert(6 == sizeof(pi.peer_addr), "");
    std::copy_n(mac, 6, pi.peer_addr);
    pi.channel = channel;
    pi.encrypt = false;

    if (esp_now_is_peer_exist(mac)) {
        return esp_now_mod_peer(&pi) == ESP_OK;
    }
    return esp_now_add_peer(&pi) == ESP_OK;
}

void init_espnow(){
    // Set device as a Wi-Fi Station
    Serial.println("INIT");
    WiFi.mode(WIFI_STA);
    WiFi.macAddress(selfMAC);    // Storing self mac-address

    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        ESP.restart();
    }

    esp_now_register_send_cb(OnDataSent);

    for (int i=0; i<PEERS_COUNT; i++){
        Serial.printf("Add Peer:   {%02X:%02X:%02X:%02X:%02X:%02X}\n",
PEERS[i][0], PEERS[i][1], PEERS[i][2], PEERS[i][3], PEERS[i][4],
PEERS[i][5], PEERS[i][6]);

        addPeer(PEERS[i]);
    }

    esp_now_register_recv_cb(OnDataRecv);
}

void setup(){
    init_fuzzy_rules();

    Serial.begin(115200);
    Serial.println();

    // setup ADC resolution to 12 bits

```

```

analogReadResolution(12);

recv_status = 0;
sent_status = 0;
counter += duration_sleep;
get_sensor_data();
get_battery_value();
int m = get_memory_value();
float d = get_delta_value();
float st = get_silent_time_value();

if ((st >= 6*60000)&&(count_alive_peers == 0)){
    make_all_peers_alive();
}
int decision = fuzzy_controller(battery, m, d, st);
Serial.printf("Current state: sensor - %5.2f, timestamp - %lu,
battery - %5.2f, memory state - %d\n", new_data.x,
new_data.measured_at, battery, m);
switch (decision) {
    case 1:
        Serial.println("Decision 1");
        decision_1();
        break;
    case 2:
        Serial.println("Decision 2");
        init_espnw();
        decision_2();
        break;
    case 3:
        Serial.println("Decision 3");
        decision_3();
        break;
    case 4:
        Serial.println("Decision 4");
        init_espnw();
        decision_4();
        break;
    case 5:
        Serial.println("Decision 5");
        init_espnw();
        decision_5();
        break;
    case 6:
        Serial.println("Decision 6");
        init_espnw();
        decision_6();
        break;
}
esp_sleep_enable_timer_wakeup(duration_sleep*3000000);
esp_deep_sleep_start();
}

void loop(){}

```

A.2 Текст програми контролера нечіткої логіки

A.2.1 Файл-заголовок

```
#ifndef _FUZZY_RULES_H_
#define _FUZZY_RULES_H_

#include "Arduino.h"

static int fuzzy_rules[5][5][5][5];

void init_fuzzy_rules();

float triangular(float a, float b, float c, float x, int bounds);
int membership_battery(float x);
int membership_memory(float x);
int membership_delta(float x);
int membership_silent_time(float x);
int fuzzy_controller(float b, float m, float d, float st);

#endif
```

A.2.2 Файл з вихідним кодом

```
/*
 * This is fuzzy rule set
 * In the cells placed the number of decision - branch of
 * the algorithm that is optimal to use with current circumstances,
 * aka battery state, memory state, delta of measurment and silent
time.
 * The indices are the id of linguistic value. For example,
 * [4][0][1][2] - vp battery, vn memory, n delta and z silent time.
 * Since the main parameter for us is the state of the battery, we
 * will start from it when describing the rules
 */

/*
 * DECISIONS
 * Decision 1 - the endpoint does not send data or write anything to
memory
 * Decision 2 - the endpoint sends the latest measurements and also
sends data from the device's memory
 * Decision 3 - the new measurement is written to the device memory
 * Decision 4 - the endpoint sends the latest measurements
 * Decision 5 - the endpoint sends the latest measurements from the
sensor and increases the duration of deep sleep to
DURATION_SLEEP_LONG
 * Decision 6 - the endpoint sends the latest measurements from the
sensor and increases the duration of deep sleep to
DURATION_SLEEP_VERY_LONG
 */
#include "fuzzy.h"
```

```

// b m d st
void init_fuzzy_rules() {

//battery full (very positive)
fuzzy_rules[4][0][0][0] = fuzzy_rules[4][1][0][0] =
fuzzy_rules[4][2][0][0] = fuzzy_rules[4][3][0][0] =
fuzzy_rules[4][4][0][0] = 2;
...
fuzzy_rules[0][0][4][4] = fuzzy_rules[0][1][4][4] =
fuzzy_rules[0][2][4][4] = fuzzy_rules[0][3][4][4] =
fuzzy_rules[0][4][4][4] = 3;

}

float triangular(float a, float b, float c, float x, int bounds){
    float ans = 0;
    if (((bounds==1)&&(x<=b)) || ((bounds==2)&&(x>=b))) {
        ans = 1.0;
    }
    else if ((x>=a)&&(x<=b)) {
        ans = 1.0 - (b-x)/(b-a);
    }
    else if ((x>=b)&&(x<=c)) {
        ans = 1.0 - (x-b)/(c-b);
    }
    return ans;
}

int membership_battery(float x){
    float ling[] = {triangular(0.0, 30.0, 50.0, x, 1), //vn
                    triangular(30.0, 50.0, 70.0, x, 0), //n
                    triangular(50.0, 70.0, 80.0, x, 0), //z
                    triangular(70.0, 80.0, 90.0, x, 0), //p
                    triangular(80.0, 90.0, 100.0, x, 2)}; //vp

    int ans = 0;
    for (int i=0; i<5; i++){
        if (ling[i]>=ling[ans]){
            ans = i;
        }
    }
    if (ans>0 && ling[ans]==ling[ans-1]){
        ans = ans-1;
    }
    return ans;
}

int membership_memory(float x){
    float ling[] = {triangular(0, 84.0, 168.0, x, 1), //vn
                    triangular(84.0, 168.0, 254.0, x, 0), //n
                    triangular(168.0, 254.0, 338.0, x, 0), //z
                    triangular(254.0, 338.0, 422.0, x, 0), //p

```

```

        triangular(338.0, 422.0, 507.0, x, 2)); //vp
int ans = 0;
for (int i=0; i<5; i++){
    if (ling[i]>=ling[ans]){
        ans = i;
    }
}
return ans;
}

int membership_delta(float x){
    float ling[] = {triangular(0, 0.05, 0.1, x, 1), //vn
                    triangular(0.05, 0.1, 0.15, x, 0), //n
                    triangular(0.1, 0.15, 0.3, x, 0), //z
                    triangular(0.15, 0.3, 0.5, x, 0), //p
                    triangular(0.3, 0.5, 1.0, x, 2)}; //vp

    int ans = 0;
    for (int i=0; i<5; i++){
        if (ling[i]>=ling[ans]){
            ans = i;
        }
    }
    if (ans>0 && ling[ans]==ling[ans-1]){
        ans = ans-1;
    }
    return ans;
}

int membership_silent_time(float x){
    int min_millis = 60000;
    float ling[] = {triangular(0, 2*min_millis, 6*min_millis, x, 1),
//vp
                    triangular(2*min_millis, 6*min_millis,
14*min_millis, x, 0), //p
                    triangular(6*min_millis, 14*min_millis,
30*min_millis, x, 0), //z
                    triangular(14*min_millis, 30*min_millis,
60*min_millis, x, 0), //n
                    triangular(30*min_millis, 60*min_millis,
180*min_millis, x, 2)}; //vn
    int ans = 0;
    for (int i=0; i<5; i++){
        if (ling[i]>=ling[ans]){
            ans = i;
        }
    }
    if (ans>0 && ling[ans]==ling[ans-1]){
        ans = ans-1;
    }
    return (4-ans);
}

int fuzzy_controller(float b, float m, float d, float st){

```

```

    int m_b = membership_battery(b);
    int m_m = membership_memory(m);
    int m_d = membership_delta(d);
    int m_st = membership_silent_time(st);
    int decision = fuzzy_rules[m_b][m_m][m_d][m_st]; //Infer the fuzzy
output using the linguistic rules;
    printf("Fuzzy | Decision %d: battery - %d, memory - %d, delta - %d,
st - %d\n", decision, m_b, m_m, m_d, m_st);
    return decision;
}

```

A.3 Текст програми роутерів

```

/**
linguistic variables - very negative (0),
                        negative(1),
                        zero(2),
                        positive(3),
                        very positive(4)

parameters:
battery state (b)
memory state (m)
when was the last msg sent (silent time)(st)
how much the data changed since the last sending - delta (d)
R1 40:22:d8:eb:29:4c
R2 40:22:d8:7a:35:04
E1 c8:f0:9e:30:91:54
E2 c8:f0:9e:30:9d:4c
E3 c8:f0:9e:30:97:7c
*/

#include <esp_now.h>
#include <WiFi.h>
#include <time.h>
#include <RTCLib.h>

bool need_to_send_to_cg = false;

// The recipient MAC address. It must be modified for each device.
#define WIFIESPNOW_ALEN 6

static uint8_t selfMAC[] = {0,0,0,0,0,0};

const uint8_t peerLength = 6; // MAC-address length
const uint8_t maxPeerCount = 10; // max count of peers
static uint8_t ENDPOINTS[][peerLength]{
    {0xC8, 0xF0, 0x9E, 0x30, 0x91, 0x54}, // E1 thermo
    {0xC8, 0xF0, 0x9E, 0x30, 0x9D, 0x4C}, // E2 luminosity
    {0xC8, 0xF0, 0x9E, 0x30, 0x97, 0x7C} // E3 hall
};

static uint8_t PEERS[][peerLength]{

```

```

    {0x40, 0x22, 0xD8, 0x7A, 0x35, 0x04}, // R2
};

//58:bf:25:37:cf:dc
static uint8_t GATEWAY[peerLength]{0x58, 0xBF, 0x25, 0x37, 0xCF,
0xDC};

uint8_t ALIVE_PEERS[][8]{{0x40, 0x22, 0xD8, 0x7A, 0x35, 0x04,0,0}};
RTC_DATA_ATTR unsigned int counter = 0; //real time

static uint8_t channel = 0;

struct endpoint_data {
    float sensor_data;           // 4 bytes
    unsigned long measured_at;   // 4 bytes
    float battery;               // 4 bytes
    int memory_state;           // 2 bytes
};
endpoint_data incomingReadings;

struct router_data {
    long unsigned send_time;
    int peer_count;
    uint8_t peers[maxPeerCount][7] {
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
        {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
    }; // 6 bytes for MAC, 7th for alive status
};
router_data outcomingData;

struct router_to_gateway_data {
    uint8_t endpoint[peerLength]{0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
    float sensor_data;           // 4 bytes
    unsigned long measured_at;   // 4 bytes
    float battery;               // 4 bytes
    int memory_state;           // 2 bytes
};
router_to_gateway_data new_data;

const char* ssid      = "***";
const char* password  = "***";

const char* ntpServer1 = "pool.ntp.org";
const char* ntpServer2 = "time.nist.gov";
const long  gmtOffset_sec = 7200;

```

```

const int    daylightOffset_sec = 3600;
const char*  time_zone = "";

time_t now;                                     // this is the epoch
tm myTimeInfo;

void printLocalTime(){
    struct tm timeinfo;
    if(!getLocalTime(&timeinfo)){
        Serial.println("No time available (yet)");
        return;
    }
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}

// Function that gets current epoch time
unsigned long getTime() {
    time_t now;
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Failed to obtain time");
        return(0);
    }
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
    time(&now);
    return now;
}

//CONTROLLERS
// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr,  esp_now_send_status_t
status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success"
: "Delivery Fail");
}

int local_router_number(const uint8_t *mac) {
    int match_count;
    for (int i=0; i<sizeof(PEERS)/peerLength; i++){
        match_count = 0;
        for (int j=0; j<6; j++){
            if (mac[j] == PEERS[i][j]){
                match_count++;
            }
        }
        if (match_count==6){
            return i;
        }
    }
    return -1;
}

```

```

int local_endpoint_number(const uint8_t *mac) {
    int match_count;
    for (int i=0; i<sizeof(ENDPOINTS)/peerLength; i++){
        match_count = 0;
        for (int j=0; j<6; j++){
            if (mac[j] == ENDPOINTS[i][j]){
                match_count++;
            }
        }
        if (match_count==6){
            return i;
        }
    }
    return -1;
}

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    int router_id = local_router_number(mac);
    int endpoint_id = local_endpoint_number(mac);

    if (router_id >= 0){
        ALIVE_PEERS[router_id][7] = 1;
        ALIVE_PEERS[router_id][6] = millis()/1000;
    }

    if (endpoint_id >= 0){
        memcpy(&incomingReadings, incomingData,
sizeof(incomingReadings));
        Serial.printf("OnDataRecv: Remote readings: sensor - %5.2f,
measured at - %d, battery - %5.2f%, memory state - %d\r\n",
incomingReadings.sensor_data, incomingReadings.measured_at,
incomingReadings.battery, incomingReadings.memory_state);

        if (incomingReadings.battery > 0) {
            outcomingData.send_time = getTime();
            int peer_count = sizeof(PEERS)/peerLength;
            outcomingData.peer_count = peer_count;
            for (int i=0; i<peer_count; i++){
                for (int j=0; j<6; j++){
                    outcomingData.peers[i][j] = PEERS[i][j];
                }
                outcomingData.peers[i][6] = ALIVE_PEERS[i][7];
            }
            esp_err_t result2 = esp_now_send(mac, (uint8_t *)
&outcomingData, sizeof(outcomingData));
        }

        for (int i=0; i<peerLength; i++){
            new_data.endpoint[i] = mac[i];
        }
        delay(10);
    }
}

```

```

    new_data.sensor_data = incomingReadings.sensor_data;
    new_data.measured_at = incomingReadings.measured_at;
    new_data.battery = incomingReadings.battery;
    new_data.memory_state = incomingReadings.memory_state;

    need_to_send_to_cg = true;
}
}

bool addPeer(const uint8_t mac[6]) {
    esp_now_peer_info_t pi{};
    std::copy_n(mac, 6, pi.peer_addr);
    pi.channel = channel;
    pi.encrypt = false;

    if (esp_now_is_peer_exist(mac)) {
        return esp_now_mod_peer(&pi) == ESP_OK;
    }
    return esp_now_add_peer(&pi) == ESP_OK;
}

void synchroniseWith_NTP_Time() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("configTime uses ntpServer ");
    Serial.println(ntpServer1);
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1);
    Serial.print("synchronising time");

    while (myTimeInfo.tm_year + 1900 < 2000 ) {
        time(&now); // read the current time
        localtime_r(&now, &myTimeInfo);
        delay(100);
        Serial.print(".");
    }
    Serial.print("\n time synchronsized \n");
    WiFi.disconnect(true);
    WiFi.mode(WIFI_OFF);
}

void init_espnow(){
    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    WiFi.macAddress(selfMAC); // Storing self mac-address

    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        ESP.restart();
    }

    esp_now_register_send_cb(OnDataSent);

    for (int i=0; i<sizeof(ENDPOINTS)/peerLength; i++) {

```

```

    uint8_t * mac = ENDPOINTS[i];
    if (std::equal(mac, mac + 6, selfMAC)) {
        Serial.printf ("1 %02X:%02X:%02X:%02X:%02X:%02X is my own mac-
address, skipping\n", mac[0], mac[1], mac[2], mac[3], mac[4],
mac[5]);
        continue;
    }
    addPeer(mac);
}
for (int i=0; i<sizeof(PEERS)/peerLength; i++) {
    uint8_t * mac = PEERS[i];
    if (std::equal(mac, mac + 6, selfMAC)) {
        Serial.printf ("1 %02X:%02X:%02X:%02X:%02X:%02X is my own mac-
address, skipping\n", mac[0], mac[1], mac[2], mac[3], mac[4],
mac[5]);
        continue;
    }
    addPeer(mac);
}
addPeer(GATEWAY);
esp_now_register_recv_cb(OnDataRecv);
}

void setup(){
    Serial.begin(115200);
    Serial.println();
    synchroniseWith_NTP_Time();
    init_espnow();
}

void loop(){
    char msg[60];
    int len = snprintf(msg, sizeof(msg), "hello ESP-NOW from %s at
%lu",
                        WiFi.softAPmacAddress().c_str(), millis());
    for (int i=0; i<sizeof(PEERS)/peerLength; i++){
        esp_err_t result1 = esp_now_send(PEERS[i], (uint8_t *) &msg,
sizeof(msg));
    }
    for (int i = 0; i < sizeof(PEERS)/peerLength; i++) {
        if ((millis()/1000) - ALIVE_PEERS[i][6] > 10) {
            ALIVE_PEERS[i][7] = 0;
        }
        Serial.printf("Router  0x%02X:0x%02X:0x%02X:0x%02X:0x%02X:0x%02X
is
                        0x%02X\n",
                        ALIVE_PEERS[i][0],ALIVE_PEERS[i][1],
ALIVE_PEERS[i][2],
                        ALIVE_PEERS[i][3],
                        ALIVE_PEERS[i][4],
ALIVE_PEERS[i][5], ALIVE_PEERS[i][7]);
    }

    if (need_to_send_to_cg){
        esp_err_t result3 = esp_now_send(GATEWAY, (uint8_t *) &new_data,
sizeof(new_data));
    }
}

```

```

    Serial.printf("Try to send: %x\n", result3);
    need_to_send_to_cg = false;
}

delay(1000);
}

```

A.4 Текст програми центрального шлюзу

```

/**
linguistic variables - very negative (0),
                        negative(1),
                        zero(2),
                        positive(3),
                        very positive(4)

parameters:
battery state (b)
memory state (m)
when was the last msg sent (silent time)(st)
how much the data changed since the last sending - delta (d)
R1 40:22:d8:eb:29:4c
R2 40:22:d8:7a:35:04
E1 c8:f0:9e:30:91:54
E2 c8:f0:9e:30:9d:4c
E3 c8:f0:9e:30:97:7c
*/

#include <esp_now.h>
#include <WiFi.h>
#include <time.h>
#include <RTCLib.h>

// The recipient MAC address. It must be modified for each device.
#define WIFIESPNOW_ALEN 6

uint8_t selfMAC[] = {0,0,0,0,0,0}; // self mac-address

// 58:bf:25:37:cf:dc

const uint8_t peerLength = 6;
const uint8_t maxPeerCount = 10; // max count of peers
static uint8_t ENDPOINTS[][peerLength]{
    {0xC8, 0xF0, 0x9E, 0x30, 0x91, 0x54}, // E1 thermo
    {0xC8, 0xF0, 0x9E, 0x30, 0x9D, 0x4C}, // E2 luminosity
    {0xC8, 0xF0, 0x9E, 0x30, 0x97, 0x7C} // E3 hall
};

static uint8_t ROUTERS[][peerLength]{
    {0x40, 0x22, 0xD8, 0xEB, 0x29, 0x4C}, // R1
    {0x40, 0x22, 0xD8, 0x7A, 0x35, 0x04}, // R2
};

```

```

static uint8_t channel = 0;

struct router_data {
    uint8_t endpoint_sender[peerLength];
    float sensor_data;           // 4 bytes
    unsigned long measured_at;   // 4 bytes
    float battery;               // 4 bytes
    int memory_state;            // 2 bytes
};
router_data incomingReadings;

const char* ssid      = "***";
const char* password  = "***";

const char* ntpServer1 = "pool.ntp.org";
const char* ntpServer2 = "time.nist.gov";
const long  gmtoffset_sec = 7200;
const int   daylightOffset_sec = 3600;
const char* time_zone = "";

time_t now;                               // this is the epoch
tm myTimeInfo;

void printLocalTime(){
    struct tm timeinfo;
    if(!getLocalTime(&timeinfo)){
        Serial.println("No time available (yet)");
        return;
    }
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}

// Function that gets current epoch time
unsigned long getTime() {
    time_t now;
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Failed to obtain time");
        return(0);
    }
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
    time(&now);
    return now;
}

//CONTROLLERS
// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t
status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success"
: "Delivery Fail");
}

```

```

int local_router_number(const uint8_t *mac) {
    int match_count;
    for (int i=0; i<sizeof(ROUTERS)/peerLength; i++){
        match_count = 0;
        for (int j=0; j<6; j++){
            if (mac[j] == ROUTERS[i][j]){
                match_count++;
            }
        }
        if (match_count==6){
            return i;
        }
    }
    return -1;
}

int local_endpoint_number(const uint8_t *mac) {
    int match_count;
    for (int i=0; i<sizeof(ENDPOINTS)/peerLength; i++){
        match_count = 0;
        for (int j=0; j<6; j++){
            if (mac[j] == ENDPOINTS[i][j]){
                match_count++;
            }
        }
        if (match_count==6){
            return i;
        }
    }
    return -1;
}

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
    int router_id = local_router_number(mac);
    int endpoint_id = local_endpoint_number(incomingReadings.endpoint_sender);

    char *endpoint_names[] = { "E1 - Thermo sensor", "E2 - Luminosity sensor", "E3 - Hall sensor" };

    Serial.printf("----\nRouter %d | Endpoint: %s\n Sensor - %5.2f, measured at - %d, battery - %5.2f%, memory state - %d\n----\r\n", router_id+1, endpoint_names[endpoint_id], incomingReadings.sensor_data, incomingReadings.measured_at, incomingReadings.battery, incomingReadings.memory_state);
}

bool addPeer(const uint8_t mac[6]) {
    esp_now_peer_info_t pi{};

```

```

std::copy_n(mac, 6, pi.peer_addr);
pi.channel = channel;
pi.encrypt = false;

if (esp_now_is_peer_exist(mac)) {
    return esp_now_mod_peer(&pi) == ESP_OK;
}
return esp_now_add_peer(&pi) == ESP_OK;
}

void synchroniseWith_NTP_Time() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("configTime uses ntpServer ");
    Serial.println(ntpServer1);
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1);
    Serial.print("synchronising time");

    while (myTimeInfo.tm_year + 1900 < 2000 ) {
        time(&now); // read the current time
        localtime_r(&now, &myTimeInfo);
        delay(100);
        Serial.print(".");
    }
    Serial.print("\n time synchronsized \n");
    WiFi.disconnect(true);
    WiFi.mode(WIFI_OFF);
}

void init_espnw(){
    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    WiFi.macAddress(selfMAC); // Storing self mac-address

    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        ESP.restart();
    }

    esp_now_register_send_cb(OnDataSent);
    for (int i=0; i<sizeof(ENDPOINTS)/peerLength; i++) {
        uint8_t * mac = ENDPOINTS[i];
        if (std::equal(mac, mac + 6, selfMAC)) {
            Serial.printf ("1 %02X:%02X:%02X:%02X:%02X:%02X is my own mac-
address, skipping\n", mac[0], mac[1], mac[2], mac[3], mac[4],
mac[5]);
            continue;
        }
        addPeer(mac);
    }
    for (int i=0; i<sizeof(ROUTERS)/peerLength; i++) {
        uint8_t * mac = ROUTERS[i];
        if (std::equal(mac, mac + 6, selfMAC)) {
            Serial.printf ("1 %02X:%02X:%02X:%02X:%02X:%02X is my own mac-
address, skipping\n", mac[0], mac[1], mac[2], mac[3], mac[4],
mac[5]);

```

```
        continue;
    }
    addPeer (mac);
}

esp_now_register_recv_cb (OnDataRecv);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    synchroniseWith_NTP_Time();
    init_espnow();
}

void loop() {
    delay(1000);
}
```

ДОДАТОК Б
Слайди презентації

НУ «Запорізька політехніка»
Кафедра програмних засобів
Дипломна кваліфікаційна робота бакалавра



Програмне забезпечення енергоефективної бездротової мережі

Виконала:
ст. групи КНТ –129

Анастасія ЯЦЕНКО

Керівник:
К.Т.Н., доцент

Анжеліка ПАРХОМЕНКО

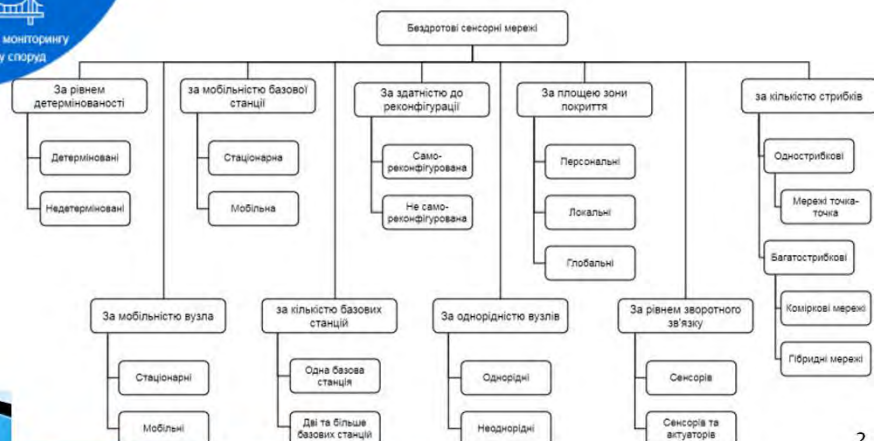
Рисунок Б.1 – Слайд 1

Актуальність роботи



Сфери застосування БСМ

Класифікація БСМ



2

Рисунок Б.2 – Слайд 2

Мета та завдання роботи

Мета роботи: розробка програмного забезпечення БСМ на основі удосконаленого алгоритму збору та передачі даних для мінімізації енергоспоживання системи.

Завдання роботи:

- ▶ дослідити існуючі архітектури та протоколи БСМ;
- ▶ проаналізувати методи та засоби мінімізації енергоспоживання БСМ;
- ▶ виконати аналіз вимог та розробити архітектуру системи;
- ▶ обрати інструментарій розробки;
- ▶ виконати розробку програмного забезпечення;
- ▶ розробити прототип та провести тестування БСМ.



3

Рисунок Б.3 – Слайд 3

Огляд архітектур та протоколів БСМ

Порівняльна характеристика архітектур БСМ

Архітектура	Мережа точка точка	Коміркові мережі	Гібридні мережі
Властивість			
Можливість передавати дані через різні вузли	Відсутня	Присутня	Присутня
Можливість переводити точку у глибокий сон	Відсутня	Відсутня	Присутня

Порівняння протоколів локальної групи

Протокол	ESP-NOW	Wi-Fi	ZigBee
Характеристика			
Енергоспоживання	Низьке	Середнє	Низьке
Швидкість передачі даних	Середня (до 1 Мб/с)	Висока (до 600 Мб/с)	Низька (до 250 кб/с)
Потреба у спеціалізованому модулі	Так (модулі на базі ESP-8266 та ESP32)	Ні	Так (модем ZigBee)
Вартість	Низька	Низька	Висока



4

Рисунок Б.4 – Слайд 4

Методи та засоби мінімізації енергоспоживання БСМ

Огляд існуючих методів вирішення задачі мінімізації

Задача	Метод вирішення
Оптимізація передачі даних	Використання теорії ігор [2]
	Використання агрегації даних [22]
	Впровадження гібридної архітектури [8]
	Зменшення відстаней для зв'язку [23]
Оптимізація роботи вузлів	Використання інтервальної роботи вузлів [23]

Огляд підходів для оптимізації енергоспоживання вузлів-сенсорів

Підхід	Деревні	Генетичні алгоритми	Нечітка логіка
Характеристика			
Складність математичної моделі	Середня	Низька	Низька
Легкість інтерпретації результатів	Середньо	Складно	Легко
Ризик надскладних ланцюжків рішення	Середній	Високий	Низький

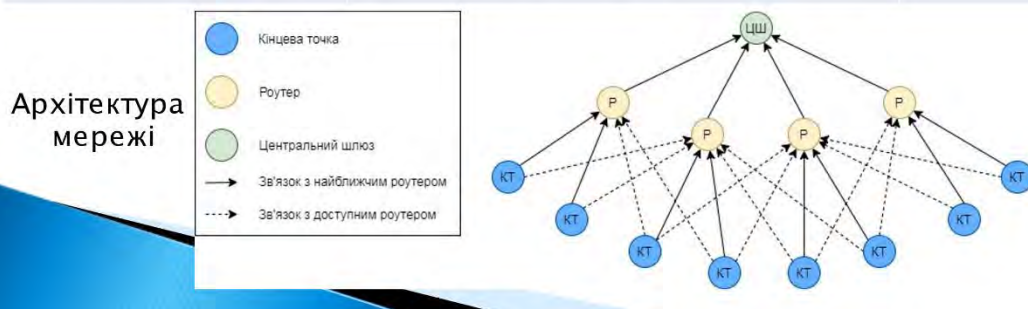
5

Рисунок Б.5 – Слайд 5

Проектування ПЗ

Функціональні вимоги

Вузол-сенсор	Вузол-роутер	Центральний шлюз
Вимір показників середовища та оцінка власного стану	Відправлення роутерам-сусідам повідомлень-keepalive	Отримання даних від роутерів
Коригування послідовності дій в залежності від результатів роботи контролера нечіткої логіки	Оновлення статусу роутера-сусіда як «недоступний» у списку сусідів при відсутності повідомлень від цього роутера тривалий час	Виведення отриманих даних на екран
Відправлення значень вимірів	Оновлення статусу роутера-сусіда як «доступний» у списку сусідів у разі отримання від цього роутера повідомлення	
Запис у пам'ять значень вимірів	Отримання повідомлень від кінцевих точок	
Відправляти виміри, збережених в пам'яті вузла	Відправлення точці локального списку роутерів-сусідів та поточного часу	
Інтервальний режим роботи	Відправлення даних кінцевої точки на центральний шлюз	



6

Рисунок Б.6 – Слайд 6

Вибір інструментарію розробки

Модуль		ESP8266		ESP32		
Характеристика						
Процесор		Xtensa Single-core 32-bit L106		Xtensa Dual-Core 32-bit LX6 with 600 DMIPS		
Частота процесору		80 МГц		160 МГц		
Bluetooth	Мова Властивість	C	C++	Rust	Go (TinyGo)	Python (MicroPython)
Кількість GPIO						
Годинник						
Флеш-пам'ять						
Статична пам'ять						
SPI/I2C/I2S						
Аналогові перетворювачі	Споживання енергії	Низьке	Низьке	Низьке	Високе	Дуже високе
	Час виконання	Малий	Малий	Малий	Великий	Великий
	Використання пам'яті	Мале	Мале	Середнє	Мале	Середнє
Середовище		Arduino IDE		PlatformIO IDE	ESP-IDF	
Характеристика						
Легкість налаштування		Легко		Середньо	Складно	
Зручність процесу керування		Середньо		Легко	Легко	
Зручність керування латами		Легко		Середньо	Середньо	

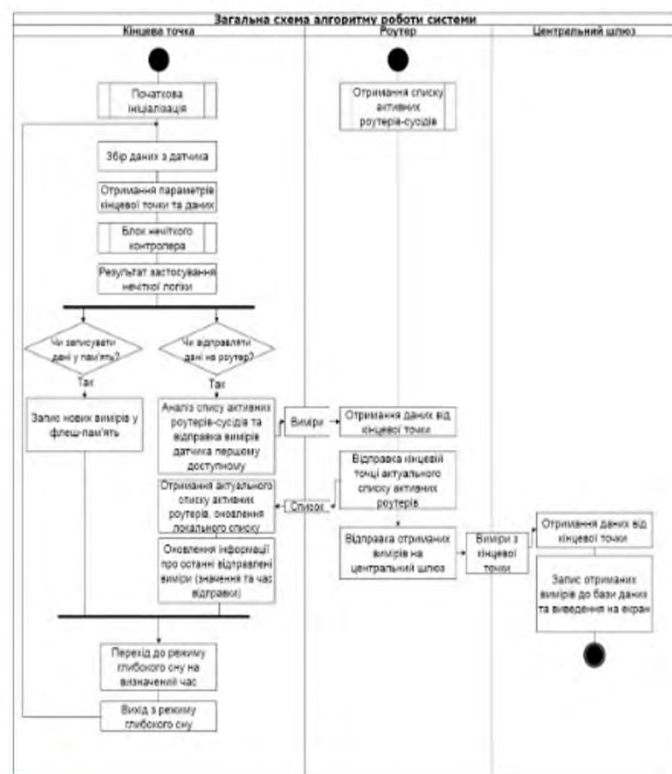
7

7

Рисунок Б.7 – Слайд 7

Розробка алгоритму роботи БСМ

Загальна схема алгоритму роботи системи



8

Рисунок Б.8 – Слайд 8

Розробка алгоритму роботи БСМ



Рисунок Б.9 – Слайд 9

Розробка алгоритму роботи БСМ



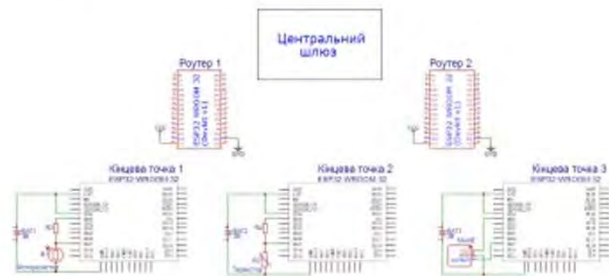
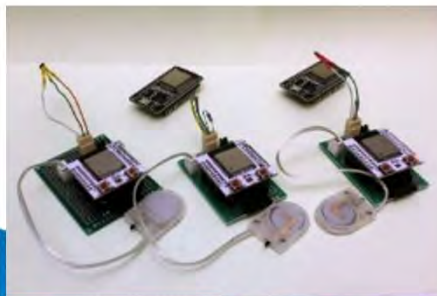
Рисунок Б.10 – Слайд 10

Розробка прототипу БСМ

Структура програмного забезпечення



Структура апаратного забезпечення



11

Рисунок Б.11 – Слайд 11

Результати тестування

Чекліст тестування кінцевої точки

№	Опис сценарію	Умови				Результат
		Заряд автономного джерела живлення	Параметри кінцевої точки	Стан заповненості пам'яті	Величина зміни даних	Час останньої відправки даних
1	Рішення 1	Високий	Майже незаповнена	Дуже мала	Щойно	Пройдено
2	Рішення 2; у пам'яті відсутні записи, Високий у мережі є доступні роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
3	Рішення 2; у пам'яті є записи, у Високий мережі є доступні роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
4	Рішення 2; у мережі відсутні доступні роутери	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
5	Рішення 3	Середній	Майже незаповнена	Дуже мала	Не дуже давно	Пройдено
6	Рішення 4	Середній	Середня	Дуже мала	Не дуже давно	Пройдено
7	Рішення 5	Низький	Середня	Дуже мала	Дуже давно	Пройдено
8	Рішення 6	Низький	Майже повністю заповнена	Середня зміна	Дуже давно	Пройдено
9	Оновлення локального списку роутерів при отриманні даних від роутера	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено
10	Підключення до різних роутерів в залежності від їх доступності	Високий	Майже незаповнена	Дуже велика	Щойно	Пройдено

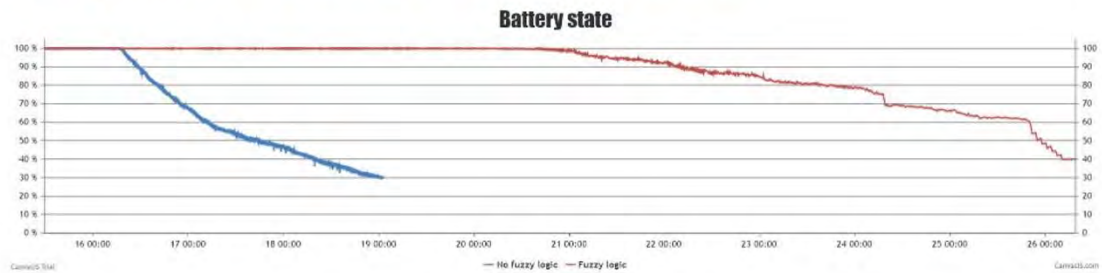
Чекліст тестування роутерів та центрального шлюзу

№	Опис сценарію	Результат
1	Отримання роутером вимірів від кінцевої точки	Пройдено
2	Отримання роутером архіву кінцевої точки	Пройдено
3	Відправлення роутером локального списку роутерів-сусідів та поточного часу кінцевій точці при отриманні даних від неї	Пройдено
4	Відправлення роутером даних, отриманих від кінцевої точки, на центральний шлюз	Пройдено
5	Отримання центральним шлюзом даних від роутерів та виведення їх на екран	Пройдено

12

Рисунок Б.12 – Слайд 12

Результати тестування



Результати	Підхід	Традиційний підхід	Підхід з нечіткою логікою
Загальний час роботи		85 годин	260 годин
Заряд джерела живлення після 85 годин роботи		30% (недостатньо для відправки даних)	100%
Кількість транзакцій за перші 85 годин роботи		94 859	2 400

13

Рисунок Б.13 – Слайд 13

Висновки

- ▶ **Наукова новизна** роботи полягає в тому, що алгоритм збору та передачі даних БСМ удосконалено за рахунок використання нечіткої логіки, що на відміну від існуючих підходів дозволяє не тільки визначити оптимальний шлях передачі даних, але й мінімізувати енергоспоживання.
- ▶ **Практична цінність** роботи полягає в тому, що розроблене програмно-апаратне рішення може бути використане для побудови стаціонарних БСМ локальної зони покриття, зокрема, для систем «Розумний будинок» та «Розумна будівля», що дозволить контролювати використання ресурсів та заощаджувати кошти.
- ▶ **У подальшій роботі** планується перевірити роботу БСМ, яка складається з великої кількості кінцевих точок, а також розглянути можливості використання альтернативних джерел живлення.

14

Рисунок Б.14 – Слайд 14

Перелік публікацій

1) Methods and tools for minimizing the power consumption of wireless sensor network / [A. Yatsenko, C. Wolff, A. Parkhomenko et. al.] 2023 IEEE European technology & engineering management summit (ETEMS), Kaunas, Lithuania, 20–22 April 2023: proceedings. – Los Alamitos: IEEE, 2023. – P. 1–6.

2) Яценко, А.К. Мінімізація енергоспоживання бездротової сенсорної мережі на основі програмних і апаратних рішень / А.К. Яценко, А.В. Пархоменко, А.В. Туленков // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : XVI Міжнародна науково-практична конференція, Запоріжжя, 12–14 грудня 2022 р.: тези доповідей. – Запоріжжя: НУЗП, 2022. – С. 201–202.

3) Яценко, А.К. Дослідження методів та засобів розробки енергоефективних бездротових сенсорних мереж / А.К. Яценко, А.В. Пархоменко // Тиждень науки-2023: Щорічна науково-практична конференція викладачів, науковців, молодих учених, аспірантів та студентів, Запоріжжя, 24 –28 квітня 2023 р.: тези доповідей. – Запоріжжя: НУЗП, 2023. – С.



Рисунок Б.15 – Слайд 15