

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА
“ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “ ІоТ-система автоматизованого управління та моніторингу систем
безперебійного живлення ”

Виконала: студентка 4 курсу, групи ТР-24

_____ Цекот Мар’яна Володимирівна

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник асистент Антон ПАСІЧНЮК

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент доцент каф. ТАЕ, к.т.н., Артур РАЧИНСЬКИЙ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль асистент Володимир РУДИК

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студентка _____

Київ – 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

(підпис)

“ ” _____ 2026 р.

З А В Д А Н Н Я
на дипломну роботу студентці

Цекот Мар’яні Володимирівні

(прізвище, ім’я, по батькові)

1. Тема роботи «ІоТ-система автоматизованого управління та моніторингу систем безперебійного живлення».

керівник роботи Пасічник Антон Олексійович

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

затверджені наказом по університету від «28» травня 2026р. № 1992-с

2. Термін подання роботи студентом 04.06.2026р.

3. Вихідні дані до роботи: мікроконтролер ESP32, мікросхема узгодження логічних рівнів MAX3232 (інтерфейс RS-232/UART), мова програмування MicroPython, вебтехнології (HTML, CSS, JavaScript, Fetch API), бібліотека Chart.js.

4. Перелік питань, які потрібно розробити: 1) провести аналіз методів та алгоритмів для розв’язання задачі локального моніторингу енергетичного обладнання; 2) обґрунтувати вибір апаратних засобів та програмних інструментів для реалізації комплексу; 3) спроектувати апаратну частину та розробити серверне програмне забезпечення мікроконтролера; 4) створити прикладний програмний веб-інтерфейс з модулями дистанційного керування, прогнозування часу

автономної роботи та автентифікації за графічним ключем; 5) провести обчислювальні експерименти та тестування коректності роботи системи на фізичному стенді 6) представити процес застосування веб-інтерфейсу.

5. Орієнтовний перелік ілюстративного матеріалу: структурна схема програмно-апаратного комплексу, схема алгоритму циклічного опитування інвертора, блок-схема алгоритму розрахунку часу автономної роботи, архітектура клієнт-серверної взаємодії, скріншоти роботи користувачів з веб-застосунком.

6. Дата видачі завдання 15.09.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	12.09.2025 – 14.09.2025	Виконано
2.	Аналіз методів та засобів розв'язання задачі	14.10.2024-29.11.2025	Виконано
3.	Розробка архітектури та загальної структури системи	16.12.2024-17.01.2025	Виконано
4.	Розробка окремих підсистем	20.01.2025-14.03.2026	Виконано
5.	Програмна реалізація системи	12.04.2026 – 17.05.2026	Виконано
6.	Оформлення пояснювальної записки	18.05.2026-26.05.2026	Виконано
7.	Захист програмного забезпечення	13.05.2026	Виконано
8.	Передзахист	03.06.2026	Виконано
9.	Захист	16.06.2026	Виконано

Студентка

Керівник роботи

(підпис)

(підпис)

Мар'яна Цекот

(ім'я, ПРІЗВИЩЕ)

Антон ПАСІЧНЮК

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота бакалавра виконана на 52 сторінках та містить 20 ілюстрацій, 1 додаток та 11 найменувань у списку використаних джерел.

Мета роботи – створення локальної, автономної програмно-апаратної системи для безперервного моніторингу та управління безперебійним джерелом живлення, яка не залежить від хмарних серверів та провайдера.

Методи та засоби: концепція периферійних обчислень (Edge Computing), протокол UART для послідовного зв'язку, алгоритм математичного прорахунку надлишкового коду CRC-16. Апаратна частина складається з мікроконтролера ESP32 та модуля перетворювача рівнів RS232-TTL для безпечної роботи та узгодження промислових рівнів безперебійного джерела живлення. Програмна частина, а саме бекенд реалізований мовою MicroPython, користувацький інтерфейс реалізований за допомогою Vanilla JavaScript, HTML5/CSS3 і бібліотеки Chart.js.

Результат: локальна система, що забезпечує безперервний збір телеметрії (напруга мережі, напруга акумулятора, поточний струм, рівень навантаження), побудову динамічних графіків, розрахунок орієнтовного часу автономної роботи безперебійного джерела живлення та керування струмом заряду.

Ключові слова: ІНТЕРНЕТ РЕЧЕЙ, ESP32, БЕЗПЕРЕБІЙНЕ ДЖЕРЕЛО ЖИВЛЕННЯ, ЛОКАЛЬНИЙ СЕРВЕР.

ABSTRACT

The bachelor's thesis is executed on 52 pages and contains 20 illustrations, 1 appendix, and 11 titles in the list of references.

The purpose of the work is to create a local, autonomous hardware and software system for continuous monitoring and management of an uninterruptible power supply, which is independent of cloud servers and internet providers.

Methods and tools: Edge Computing concept, UART protocol for serial communication, mathematical algorithm for calculating the redundancy code CRC-16. The hardware consists of an ESP32 microcontroller and an RS232-TTL level converter module for safe operation and matching the industrial levels of the uninterruptible power supply. The software part, specifically the backend, is implemented in MicroPython, while the user interface is developed using Vanilla JavaScript, HTML5/CSS3, and the Chart.js library.

Result: a local system that ensures continuous telemetry collection (grid voltage, battery voltage, current, load level), dynamic charting, calculation of the estimated autonomous operation time of the uninterruptible power supply, and charge current control.

Keywords: INTERNET OF THINGS, ESP32, UNINTERRUPTIBLE POWER SUPPLY, LOCAL SERVER.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
1 ІОТ-СИСТЕМА МОНІТОРИНГУ ТА КЕРУВАННЯ	10
1.1 Постановка прикладної задачі	11
1.2 Огляд методів необхідних для розв’язання поставленої задачі	11
1.3 Огляд існуючих підходів	13
1.4 Аналіз функціональних та нефункціональних вимог до системи	14
2 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ	17
2.1 Алгоритм циклічного опитування та апаратного узгодження інтерфейсів	17
2.2 Математичне забезпечення цілісності даних (алгоритм CRC-16)	20
2.3 Алгоритми синтаксичного розбору та безпечного структурування телеметрії	21
2.4 Математична модель та алгоритм прогнозування часу автономної роботи...	23
2.5 Алгоритми оптимізації генерування графіків в інтерфейсі користувача.....	24
3 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ	26
3.1 Огляд програмних засобів, необхідних для розв’язання поставленої задачі .	26
3.2 Огляд апаратних засобів необхідних для розв’язання поставленої задачі	27
3.3 Функціональна модель та загальна архітектура системи	28
3.4 Архітектура серверного програмного забезпечення	31
3.5 Реалізація апаратного зв’язку та синтаксичного розбору протоколу безперебійного джерела живлення.....	33
3.6 Клієнтська частина, візуалізація даних та інтерфейс користувача.....	34
3.7 Модель представлення інформації та процес серіалізації	36
3.8 Апаратна архітектура та топологія фізичного підключення вузлів	37
3.9 Динаміка інформаційної взаємодії та синхронізація процесів.....	38

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ.....	40
4.1 Підготовка фізичного випробувального стенда, інсталяція програмного забезпечення та ініціалізація системи.....	40
4.2 Демонстрація функціоналу та розширені можливості користувацької взаємодії	44
ВИСНОВКИ.....	51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	53
ДОДАТОК А.....	55

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) — прикладний програмний інтерфейс, набір засобів для взаємодії компонентів програмного забезпечення.

ASCII (American Standard Code for Information Interchange) — текстовий формат для кодування символів інформаційного обміну.

CRC (Cyclic Redundancy Check) — циклічний надлишковий код, алгоритм математичної перевірки цілісності інформації.

Edge Computing — концепція периферійних обчислень, що забезпечує обробку даних безпосередньо на локальному вузлі.

GND (Ground) — спільна шина заземлення або шина нульового потенціалу.

HEX (Hexadecimal) — шістнадцятковий масив даних.

HTTP (HyperText Transfer Protocol) — протокол передачі гіпертексту для маршрутизації мережевих запитів та відповідей.

IDE (Integrated Development Environment) — інтегроване середовище розробки.

JSON (JavaScript Object Notation) — легковаговий текстовий формат передачі та збереження даних.

LAN (Local Area Network) — локальна мережа.

RS-232 (Recommended Standard 232) — комунікаційний інтерфейс, промисловий стандарт передачі сигналів.

SRAM (Static Random Access Memory) — оперативна пам'ять мікроконтролера.

TTL (Transistor-Transistor Logic) — транзисторно-транзисторна логіка, цифровий стандарт рівнів напруги.

TX / RX (Transmit / Receive) — контакти передачі та приймання даних.

UART (Universal Asynchronous Receiver-Transmitter) — універсальний асинхронний приймач-передавач, протокол та апаратний інтерфейс для послідовного зв'язку.

ВСТУП

Сучасні системи безперебійного живлення (СБЖ) та інвертори потребують надійних інструментів для постійного відстеження їхнього стану та збору точних даних. В умовах сучасної енергетичної нестабільності та регулярних перебоїв з електропостачанням, наявність керованого резервного живлення стає критичною вимогою для функціонування як робочих, так і побутових просторів. Замість незручного перегляду показників на маленькому вбудованому екрані пристрою, виникає потреба у комплексних інформаційних системах, які забезпечують дистанційне та безперервне спостереження за процесами енергоспоживання. У таких умовах IoT-система виступає ефективною платформою для детальної фіксації поточного навантаження, збору метрик про стан акумуляторів та довгострокового моніторингу стабільності всієї системи в реальному часі.

Більшість існуючих інформаційних рішень покладаються на хмарні сервери, що робить їх непрацездатними за відсутності доступу до Інтернету. Ефективним шляхом подолання цієї вразливості є застосування концепції Edge Computing (периферійних обчислень) [1]. Такий підхід переносить обробку даних безпосередньо на локальний рівень, усуваючи критичну залежність від зовнішніх провайдерів та гарантуючи стабільний доступ до метрик виключно в межах домашньої мережі [2]. Додатковою перевагою такого архітектурного рішення є підвищений рівень інформаційної безпеки, оскільки чутливі дані про звички енергоспоживання не передаються на сторонні сервери, а залишаються повністю під контролем користувача.

Метою роботи є розробка IoT-системи моніторингу та керування джерелами безперебійного живлення. Створений комплекс призначений для забезпечення безперервного збору, обробки та наочної візуалізації телеметрії в режимі реального часу, надаючи користувачеві повний контроль над робочими параметрами обладнання.

Для досягнення поставленої мети було поставлено такі завдання:

- 1) спроектувати загальну архітектуру програмно-апаратного комплексу на базі концепції периферійних обчислень;
- 2) налаштувати надійну апаратну взаємодію та обмін даними з обладнанням через комунікаційний інтерфейс UART ;
- 3) розробити серверну частину на базі мікроконтролера ESP32 з використанням інтерпретатора MicroPython;
- 4) створити інтерактивний клієнтський веб-інтерфейс для дистанційного управління, моніторингу та візуалізації показників;.
- 5) реалізувати аналітичний модуль прогнозування для динамічного розрахунку залишкового часу автономної роботи безперебійного джерела живлення.

Практична цінність розробки полягає у забезпеченні користувача інтерактивною панеллю, яка наочно відображає динамічні тренди енергоспоживання. Постійна візуалізація цих даних дозволяє ефективно розподіляти навантаження, вчасно фіксувати аварійні стрибки потужності та оптимізувати використання побутових приладів для максимального подовження часу автономної роботи системи.

1 ІОТ-СИСТЕМА МОНІТОРИНГУ ТА КЕРУВАННЯ

Залежність сучасних систем безперебійного живлення від хмарних сервісів створює суттєві ризики, адже під час знеструмлень доступ до глобальної мережі часто стає нестабільним або відсутнім. Це призводить до того, що користувач втрачає контроль над енергоспоживанням саме тоді, коли ресурс акумуляторів необхідно використовувати максимально раціонально та обережно. Така вразливість нівелює переваги дистанційного моніторингу саме в моменти, коли він є найбільш критичним. У зв'язку з цим, постає необхідність розробки повністю автономного локального інструменту, здатного забезпечити безперервний контроль за роботою безперебійного джерела живлення в умовах відсутності зовнішнього підключення.

Програмне забезпечення орієнтоване як на широке коло користувачів: від звичайних власників домогосподарств, яким важливий простий інтерфейс, до більш досвідчених користувачів, що потребують детальної аналітики. Визначені вимоги до системи враховують потреби обох категорій, спрямовуючи розробку на досягнення стабільності, швидкодії та високої якості візуалізації даних.

Методологічну базу роботи складає концепція периферійних обчислень (Edge Computing), що дозволяє здійснювати агрегацію та обробку даних безпосередньо на місці. Впроваджені підходи охоплюють надійні протоколи апаратного обміну даними, а також застосування математичних алгоритмів перевірки цілісності інформації, що гарантує точність передачі телеметрії. Фундаментом реалізації системи є обґрунтований вибір комплексу апаратних та програмних інструментів, що забезпечують оптимальну ефективність архітектури. Такий комплексний підхід не лише вирішує проблему автономного спостереження, але й формує гнучку апаратну базу для подальшого масштабування проєкту, зокрема для впровадження технологій кросплатформних месенджерів як додаткового каналу оперативного сповіщення.

1.1 Постановка прикладної задачі

Сучасні системи безперебійного живлення є складними пристроями, що поєднують функції контролера заряду та мережевого зарядного пристрою. Оскільки фізичний контроль параметрів через вбудований екран є незручним через типовий монтаж обладнання у технічних приміщеннях, виникає потреба у дистанційному моніторингу.

Існуючі штатні рішення базуються на використанні Wi-Fi модулів, які передають телеметрію на зовнішні хмарні сервери. Такий підхід має критичні недоліки: повну втрату працездатності при втраті зв'язку з провайдером, великі затримки оновлення даних, які унеможливають оперативне реагування на перевантаження, та ризики витоку внутрішніх даних про енергоспоживання.

Таким чином, прикладною задачею даної роботи є розробка автономної локальної IoT-системи, здатної безперервно збирати, обробляти та візуалізувати телеметрію з інвертора виключно в межах домашньої бездротової мережі (LAN).

Для чіткого визначення функціональних кордонів системи було побудовано діаграму прецедентів використання (UML), де основним актором виступає Адміністратор (Власник).

1.2 Огляд методів необхідних для розв'язання поставленої задачі

Для успішного вирішення поставленої прикладної задачі застосовується комплексний підхід, що охоплює методи обробки, захисту та передачі інформації на різних рівнях системної архітектури. Основним архітектурним підходом виступає метод периферійних обчислень (Edge Computing). На противагу традиційним хмарним технологіям, цей метод передбачає виконання

обчислювальних операцій та агрегацію масивів даних безпосередньо на локальному вузлі (мікроконтролері). Розташування обчислювального ядра у фізичній близькості до інвертора гарантує повну незалежність від зовнішніх інтернет-провайдерів, мінімізує затримки передачі телеметрії та підвищує загальну відмовостійкість системи під час знеструмлень.

Для безпечного отримання сирих метрик від промислового силового обладнання застосовується метод послідовної передачі даних на базі протоколу UART [3]. Оскільки інвертор функціонує з використанням сигналів стандарту RS-232, цей метод вимагає обов'язкового апаратного узгодження логічних рівнів до цифрового стандарту TTL. Обмін пакетами даних реалізується за принципом «Ведучий-Ведений» (Master-Slave), де мікроконтролер циклічно ініціює запити статусних параметрів. Інвертор у цій конфігурації виступає виключно пасивним пристроєм, що лише відповідає на цільові команди, не ініціюючи самостійного передавання.

Зважаючи на умови високого рівня електромагнітних завад, що генеруються потужними силовими транзисторами інвертора, критично важливим є використання методу математичного захисту цілісності даних. Спотворення навіть одного байта інформації під час передачі може призвести до хибних розрахунків або аварійної зупинки моніторингу.

Метод базується на алгоритмі обчислення циклічного надлишкового коду CRC-16 з використанням твірного полінома $0x1021$. Це дозволяє приймальній стороні ефективно виявляти, фільтрувати та відкидати пошкоджені під час транзакції пакети шляхом суворої звірки розрахованих і отриманих контрольних сум.

Крім того, на рівні клієнтського інтерфейсу використовується метод асинхронного полінгу (Asynchronous Polling) для забезпечення безперервного моніторингу. Цей підхід дозволяє веб-застосунку регулярно, у фоновому режимі, опитувати локальний сервер і динамічно оновлювати виключно нові значення

метрик на екрані користувача, уникаючи повного перезавантаження сторінки та суттєво економлячи обчислювальні ресурси клієнтського пристрою.

1.3 Огляд існуючих підходів

Стрімке зростання кількості встановлених систем безперебійного живлення (СБЖ) та автономних сонячних інверторів у приватних домогосподарствах стимулювало розвиток ринку програмного забезпечення для їх моніторингу. На сьогодні існуючі рішення алгоритмічно та архітектурно поділяються на дві великі категорії: фірмові додатки, що постачаються виробниками обладнання, та універсальні відкриті платформи домашньої автоматизації. Кожен із цих класів систем має специфічні особливості обробки інформації, які за певних умов стають критичними недоліками.

До першої категорії належать такі популярні комерційні продукти, як SolarPower, WatchPower, SmartESS та ShinePhone. Ці системи розробляються безпосередньо виробниками інверторів (наприклад, корпорацією Voltronic Power) і є стандартом «з коробки» [4]. Архітектура таких систем базується на використанні зовнішніх апаратних реєстраторів даних). З алгоритмічної точки зору, цей модуль не здійснює жодної локальної обробки інформації. Його єдиною функцією є циклічне опитування інвертора через порт RS-232 та ретрансляція «сирих» шістнадцяткових (HEX) масивів на віддалений хмарний сервер компанії-розробника через глобальну мережу Інтернет.

Головною проблемою фірмових додатків є їхня абсолютна залежність від хмарних обчислень (Cloud Computing). Уся математична обробка, розбір пакетів, масштабування графіків та збереження історії відбуваються на закордонних серверах. В умовах вітчизняних реалій, коли застосовуються графіки стабілізаційних або аварійних відключень електроенергії, обладнання інтернет-провайдерів часто знеструмлюється. За відсутності зовнішнього з'єднання

Datalogger втрачає зв'язок із сервером, і користувацький додаток перестає функціонувати, залишаючи власника «наосліп» саме в той момент, коли контроль за залишком заряду акумулятора є критично необхідним.

Існує альтернативний підхід, а саме використання локальних відкритих платформ розумного будинку, однією з найвідоміших є Home Assistant. Ця платформа дозволяє здійснювати контроль над пристроями в межах локальної мережі. Проте інтеграція безперебійного джерела живлення в екосистему Home Assistant вимагає розгортання складного програмного стеку та постійно працюючого сервера, що є надмірним для завдання моніторингу одного пристрою.

На основі проведеного аналізу було прийнято рішення щодо проєктування власного алгоритмічного підходу, який базується на концепції периферійних обчислень (Edge Computing). Відмова від проміжних серверів та перенесення логіки HTTP-сервера, алгоритмів парсингу та математичного розрахунку безпосередньо на мікроконтролер ESP32 дозволяє об'єднати переваги двох інших систем: забезпечити надзвичайно низьке енергоспоживання, абсолютну незалежність від глобальної мережі та високу частоту оновлення екрана на рівні 2 секунд [5].

1.4 Аналіз функціональних та нефункціональних вимог до системи

На основі проведеного огляду комерційних рішень, які переважно функціонують завдяки інтеграції з віддаленими серверами збереження даних, було виявлено їхню критичну вразливість до перебоїв у роботі глобальної інформаційної мережі. Зважаючи на потребу забезпечення безперервного локального контролю енергетичного обладнання в умовах тотального знеструмлення телекомунікаційної інфраструктури, сформовано чіткий перелік вимог до розроблюваного програмно-апаратного комплексу. Ці вимоги поділяються на дві основні категорії:

функціональні, що визначають безпосередні можливості та поведінку системи, та нефункціональні, які описують критерії якості, надійності та жорсткі обмеження під час експлуатації.

До групи функціональних вимог насамперед належить здатність серверної частини комплексу здійснювати безперервне циклічне опитування силового інвертора через апаратний послідовний інтерфейс. Система повинна автоматично формувати інформаційні запити, зчитувати отримані масиви байтів, здійснювати їхній лексичний аналіз та вилучати ключові телеметричні показники: напругу зовнішньої мережі, напругу акумуляторів, відсотковий рівень заряду, активний струм та сумарну потужність поточного навантаження. Наступною вимогою є наявність інтерактивної графічної оболонки, яка має миттєво відображати отримані дані у вигляді динамічних просторових графіків та числових індикаторів.

Окрім пасивного збору телеметрії, розроблюваний комплекс повинен забезпечувати можливість дистанційного керування силовим обладнанням. Зокрема, користувач повинен мати змогу безпечно змінювати граничні ліміти зарядного струму від загальної електромережі через панель керування.

Окремою важливою функціональною вимогою, що базується на експлуатаційних потребах користувачів, є наявність вбудованого аналітичного модуля. Цей інструмент має дозволяти віртуально моделювати підключення різних побутових приладів та автоматично розраховувати очікуваний час автономної роботи на основі введених параметрів загальної ємності акумуляторних батарей. Також система повинна містити надійний механізм автентифікації на базі векторного графічного ключа для блокування несанкціонованого доступу до зміни параметрів роботи інвертора.

Група нефункціональних вимог формується на основі жорстких апаратних обмежень мікроконтролерів та стандартів промислової відмовостійкості. Фундаментальною умовою є повна мережева автономність комплексу (концепція периферійних обчислень), що передбачає його стабільне функціонування

виключно в межах локальної бездротової мережі без жодних звернень до зовнішніх серверів.

Зважаючи на обмежений обсяг оперативної пам'яті обчислювального ядра мікроконтролера, програмне забезпечення має містити алгоритми автоматичного вивільнення ресурсів для запобігання переповненню під час тривалої роботи.

Для клієнтської частини критичною нефункціональною вимогою є забезпечення високої швидкодії оновлення об'єктної моделі документа, що вимагає обов'язкового впровадження математичних методів просторового стиснення масивів історичних даних перед їхньою візуалізацією на графіках. З точки зору надійності, апаратна та програмна частини комплексу повинні бути стійкими до електромагнітних завад та мимовільних обривів ліній зв'язку, володіючи здатністю самостійно відновлювати трансляцію телеметрії після налагодження фізичного контакту.

Додатковою вимогою до безпеки експлуатації є наявність програмних алгоритмів захисту силового обладнання. Система повинна безперервно зіставляти поточне споживання з номінальними можливостями інвертора та миттєво реагувати на критичне перевантаження шляхом блокування інтерфейсу взаємодії та активації візуально-звукової сигналізації.

2 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

Цей розділ присвячено детальному опису методів та алгоритмів, що лягли в основу розробленого програмно-апаратного комплексу. На початку здійснюється критичний огляд існуючих комерційних та відкритих аналогів систем моніторингу для обґрунтування обраного архітектурного підходу. Далі наводиться математичний апарат розрахунку контрольних сум, алгоритми опитування апаратних інтерфейсів, логіка синтаксичного розбору пакетів телеметрії, а також математична модель аналітичного модуля прогнозування часу автономної роботи системи.

2.1 Алгоритм циклічного опитування та апаратного узгодження інтерфейсів

Організація надійного каналу зв'язку між локальним сервером на базі мікроконтролера ESP32 та промисловим сонячним інвертором вимагає вирішення комплексу як апаратних, так і алгоритмічних завдань. Взаємодія між пристроями будується за класичною архітектурою «Ведучий-Ведений» (Master-Slave), де мікроконтролер виступає ініціатором транзакцій, а інвертор очікує на керівні команди. Першочерговою проблемою на фізичному рівні є розбіжність стандартів передачі сигналів. Комунікаційний порт безперебійного джерела живлення функціонує за промисловим стандартом RS-232, використовуючи біполярні рівні напруги (від -15 В до +15 В), що є критично небезпечним для входів мікроконтролера, які розраховані на логіку транзисторно-транзисторної взаємодії (TTL) з максимальним рівнем 3.3 В [6]. Для апаратного узгодження цих потенціалів у систему інтегровано спеціалізований модуль перетворювача рівнів на базі

мікросхеми MAX3232, що забезпечує безпечну двонаправлену трансляцію сигналів без спотворень.

Структурну схему апаратного узгодження логічних рівнів зображено на рисунку 2.1.

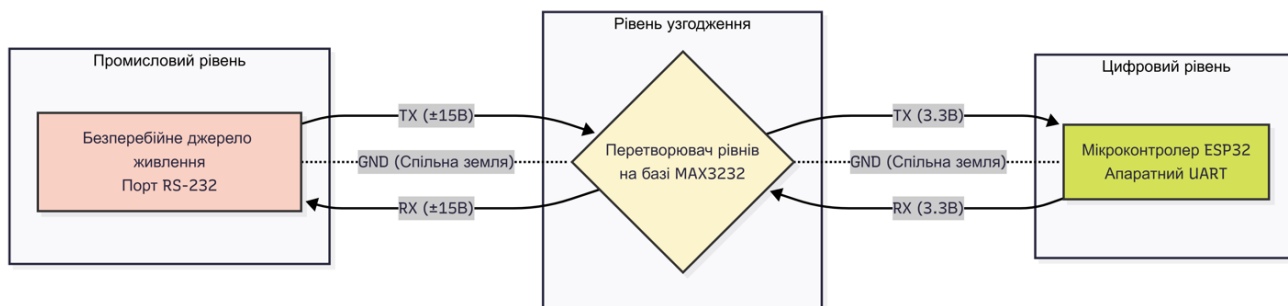


Рисунок 2.1 — Структурна схема апаратного узгодження логічних рівнів.

На програмному рівні обмін даними здійснюється через апаратний інтерфейс UART (2400 бод, 8 біт даних, 1 стоп-біт, без перевірки парності). З огляду на сильні електромагнітні завади від силової частини інвертора, алгоритм розпочинається з обов'язкового примусового очищення приймального буфера мікроконтролера (Flush). Це видаляє випадкові «сміттєві» байти та гарантує початок транзакції з чистого стану.

Після очищення каналу мікроконтролер відправляє команду запиту (наприклад, QPIGS) із контрольною сумою та переходить у стан асинхронного читання з жорстким таймаутом 1500 мс. Упродовж цього вікна система побайтово зчитує вхідний потік до виявлення символу повернення каретки (0x0D), що маркує кінець повідомлення. Якщо таймаут вичерпується раніше, операція безпечно переривається для запобігання зависанню сервера.

Фінальним етапом є підготовка «сирого» байтового масиву до розбору шляхом усічення службової інформації. Алгоритм примусово видаляє з кінця пакета три байти: два символи CRC-коду та символ каретки. Ця операція є критично важливою, оскільки спроба декодування недрукованих байтів

контрольної суми у формат UTF-8 спричиняє фатальну помилку програми. Після очищення масив безпечно конвертується в текстовий рядок для подальшого лексичного аналізу.

Деталізовану послідовність виконання програмних операцій - починаючи від примусового очищення буфера і закінчуючи безпечним декодуванням пакета показано на рисунку 2.2.

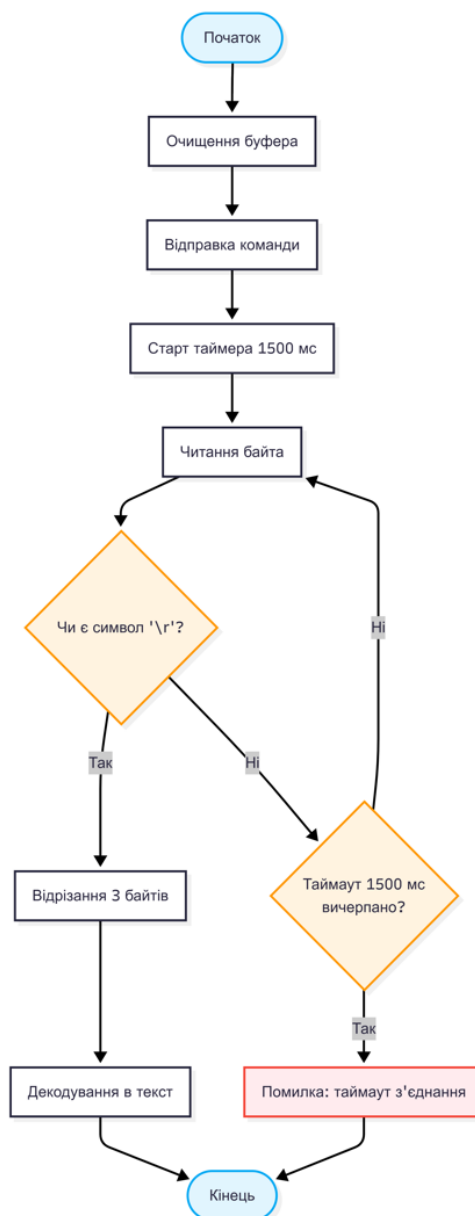


Рисунок 2.2 — Блок-схема алгоритму циклічного опитування безперебійного джерела живлення

Таким чином, запропонований алгоритм гарантує безперебійне зчитування телеметрії в умовах завад і формує чистий рядок даних, готовий до подальшого лексичного розбору та вилучення конкретних метрик інвертора.

2.2 Математичне забезпечення цілісності даних (алгоритм CRC-16)

Двонаправлена взаємодія з промисловим інвертором (зокрема надсилання запитів на отримання телеметрії, як-от QPIGS) вимагає суворого дотримання протоколів передачі даних. Через високий рівень електромагнітних завад від силових транзисторів існує ризик спотворення окремих бітів. Для гарантованого виявлення пошкоджених пакетів і відкидання невалідних команд у системі реалізовано модуль генерації циклічного надлишкового коду CRC-16. Його математична основа полягає у діленні двійкового полінома за модулем 2 з використанням стандартизованого твірного полінома $0x1021$.

Програмна реалізація алгоритму ініціалізується встановленням нульового значення для 16-бітної змінної контрольної суми. Алгоритм послідовно обробляє кожен ASCII-символ текстового запиту, виконуючи операцію виключного АБО (XOR) між поточним регістром та числовим значенням символу. Далі запускається внутрішній цикл із восьми ітерацій. На кожній перевіряється стан старшого біта за допомогою маски $0x8000$: якщо він дорівнює одиниці, регістр зсувається вліво на один розряд і до нього застосовується XOR із поліномом $0x1021$. Для запобігання переповненню пам'яті результат кожної ітерації примусово обмежується розміром у 16 біт за допомогою маски $0xFFFF$.

Після розрахунку контрольна сума ділиться на два байти. Програма перевіряє їх на збіг із забороненими керуючими символами (наприклад, дужками чи знаком переносу рядка). Якщо такий символ виявлено, до нього додається одиниця, щоб інвертор помилково не сприйняв його як кінець чи початок повідомлення.

Фінальний командний пакет, що відправляється на інвертор, складається з тексту команди, двох перевірених байтів CRC та спеціального символу завершення.

2.3 Алгоритми синтаксичного розбору та безпечного структурування телеметрії

Отримана від інвертора сира інформація є цілним ASCII-рядком великої довжини, який потребує глибокої структурної обробки. Алгоритм синтаксичного розбору (парсингу) активується лише за умови наявності стартового символу відкритої дужки на початку повідомлення. Якщо перевірка успішна, алгоритм виконує відсікання службових символів і здійснює токенизацію рядка за символом пробілу.

Оскільки отриманий масив токенів містить виключно текстові дані, для подальших математичних розрахунків реалізовано алгоритм безпечного приведення типів. Кожен цільовий індекс масиву передається у спеціальні функції обгортки (`safe_float` та `safe_int`), які використовують блоки перехоплення виключень (`try-except`). Якщо під час конвертації рядка у число з плаваючою комою виникає помилка (`ValueError`), алгоритм не зупиняє роботу сервера, а безпечно повертає нульове значення для конкретного параметра.

Окремим алгоритмічним завданням на етапі парсингу є визначення фактичного напрямку струму. Протокол інвертора повертає значення струму заряду акумуляторів від мережі та струму розряду акумуляторів у різних позиціях масиву. Алгоритм порівнює ці значення: якщо струм заряду більший за нуль, він записується як активний параметр системи; у протилежному випадку активним струмом вважається струм розряду. Після успішного завершення всіх етапів перетворення, дані упаковуються у структуру словника (`Dictionary`) і серіалізуються у легковаговий формат JSON для подальшої відправки на клієнтський веб-інтерфейс [7]. В результаті формується стандартизований пакет

даних, очищений від апаратного шуму та повністю готовий до візуалізації, саме це оптимізує навантаження на процесор мікроконтролера. Послідовність описаних операцій зображено на рисунку 2.3.

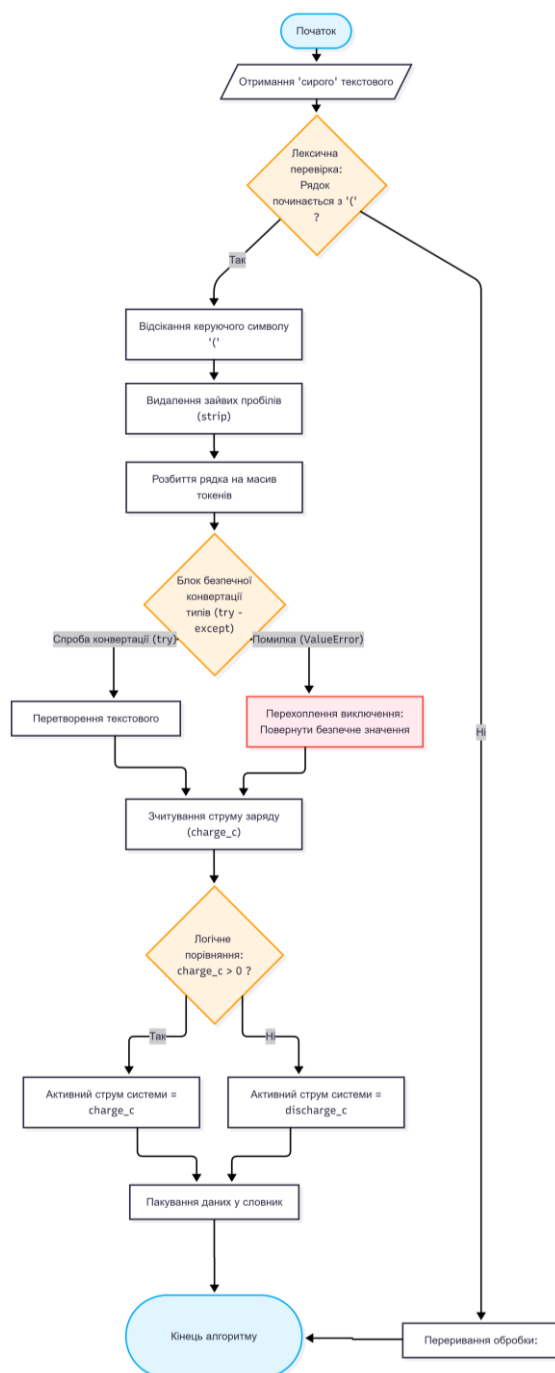


Рисунок 2.3 — Блок-схема алгоритму синтаксичного розбору та безпечного структурування телеметрії

Таким чином, запропонований алгоритм гарантує безперебійне зчитування та безпечну обробку телеметрії навіть в умовах виникнення апаратних помилок чи часткового спотворення пакетів.

2.4 Математична модель та алгоритм прогнозування часу автономної роботи

Для забезпечення користувача інформацією про очікуваний час роботи системи від акумуляторних батарей, на стороні клієнтського веб-інтерфейсу реалізовано аналітичний модуль калькулятора навантажень.

Цей алгоритм динамічно розраховує енергетичний баланс на основі введених користувачем базових констант: загальної ємності акумуляторного блоку у кВт*год (C) та максимальної допустимої потужності інвертора у Ватах (P_{max}). Початковим кроком є переведення ємності батарей у Ват-години для приведення всіх величин до єдиної розмірності:

$$E_{total} = C \cdot 1000 \quad (1)$$

Алгоритм здійснює ітеративний обхід динамічного списку підключених приладів, через зведення їхніх показників. Кожен прилад має задану споживану потужність (P_i) та два режими роботи, постійний або з фіксованою тривалістю. Під час ітерації система паралельно акумулює загальне максимальне навантаження для перевірки на перевантаження та розділяє споживання за режимами роботи. Для приладів із фіксованим часом роботи, що вказаний у годинах h_i та хвилинах m_i , вираховується їхня фіксована частка спожитої енергії (E_{fixed}):

$$E_{fixed} = \sum (P_i \cdot (h_i + \frac{m_i}{60})) \quad (2)$$

Для приладів, що працюють у постійному режимі, алгоритм просто сумує їхню потужність у змінну P_{const} . Після завершення обходу масиву приладів, система в першу чергу виконує перевірку безпеки інвертора. Розраховується відсоткове

відношення сумарної потужності всіх приладів до максимального ліміту інвертора. Якщо цей показник перевищує 100%, алгоритм генерує критичне попередження про перевантаження; при досягненні 70% формується попередження про ризик.

Ключовим етапом є прогнозування залишкового часу автономної роботи. Алгоритм віднімає від загальної ємності акумулятора енергію, яку гарантовано використовують прилади з фіксованим часом роботи, отримуючи залишковий обсяг енергії (E_{rem}):

$$E_{rem} = E_{total} - E_{fixed} \quad (3)$$

Якщо значення E_{rem} виявляється від'ємним, система фіксує дефіцит ємності акумулятора. Якщо залишок ємності позитивний і наявні прилади з постійним споживанням ($P_{const} > 0$), розрахунковий час автономної роботи (T у годинах) обчислюється за формулою:

$$T = \frac{E_{rem}}{P_{const}}$$

На фінальній стадії отримане десяткове значення часу конвертується у зручний для сприйняття формат годин та хвилин за допомогою вбудованих математичних функцій заокруглення.

2.5 Алгоритми оптимізації генерування графіків в інтерфейсі користувача

Для запобігання витоку оперативної пам'яті браузера та падінню продуктивності під час безперервного моніторингу реалізовано алгоритм математичного стиснення (децимації) історичних даних. Алгоритм накопичує отриману від інвертора телеметрію в єдиному глобальному масиві. Як тільки довжина цього масиву досягає жорстко заданої межі у 60 точок (константа MAX_POINTS), автоматично активується процедура просторового усереднення.

Програмний цикл ітерує по переповненому масиву із кроком у два елементи. Під час кожної ітерації алгоритм перевіряє наявність суміжної (наступної) точки в часі. Якщо пара існує, обчислюється середнє арифметичне значення їхніх параметрів: для показників напруги та струму результат заокруглюється до двох знаків після коми, а для потужності у Ватах та рівня заряду у відсотках застосовується математичне округлення до найближчого цілого числа.

Згенеровані усереднені вузли записуються до тимчасового масиву. У випадку непарної кількості елементів, остання точка без пари копіюється у новий масив без математичних змін. По завершенню обробки всіх елементів, старий масив повністю замінюється новим стиснутим масивом, розмір якого зменшується рівно вдвічі. Лише після виконання цієї редукції оптимізований масив передається на графічне полотно для візуалізації, що забезпечує константне споживання пам'яті.

3 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ

Проектування локальної IoT-системи моніторингу джерела безперебійного живлення вимагає комплексного поєднання логіки апаратної взаємодії, серверної частини на базі мікроконтролера та клієнтського веб-інтерфейсу.

Опис системи охоплює побудову функціональних UML-моделей, електричної топології підключення та аналіз обміну даними через інтерфейс UART. Важливою складовою є впровадження алгоритмів перевірки цілісності інформації та оптимізації швидкодії клієнтської оболонки. Особлива увага приділяється синхронізації інформаційних потоків, ефективному управлінню пам'яттю обчислювального ядра та забезпеченню повної мережевої автономності комплексу на основі концепції периферійних обчислень (Edge Computing).

3.1 Огляд програмних засобів, необхідних для розв'язання поставленої задачі

Остаточний вибір програмних засобів продиктований жорсткими обмеженнями ресурсів мікроконтролера та критичною необхідністю забезпечення максимальної швидкодії користувацького інтерфейсу. Програмну частину системи поділено на серверне та клієнтське середовище.

Для реалізації логіки серверного середовища (бекенду) на мікроконтролері ESP32 обрано оптимізовану мову програмування MicroPython. На відміну від класичного низькорівневого підходу з використанням C/C++, MicroPython надає розробнику гнучкі високорівневі структури даних, такі як словники та списки [8]. Це суттєво спрощує синтаксичний розбір (парсинг) довгих рядків телеметрії від інвертора. Разом з тим, вбудовані бібліотеки для роботи з мережевими сокетми та

апаратними перериваннями дають змогу швидко розгорнути стабільний і легковаговий HTTP-сервер у багатозадачному режимі.

Для розробки інтерактивної панелі клієнтського середовища (фронтенду) обрано базовий стек вебтехнологій — HTML5, CSS3 та Vanilla JavaScript, без залучення важких сучасних фреймворків типу React чи Angular. Відмова від масивних бібліотек є свідомим архітектурним рішенням, оскільки всі файли інтерфейсу фізично зберігаються в обмеженій флеш-пам'яті мікроконтролера [9]. Використання «чистого» JavaScript гарантує швидку реалізацію логіки Fetch API для асинхронного отримання JSON-пакетів та забезпечує миттєву обробку подій користувача, зокрема введення графічного ключа блокування [10].

Для розв'язання задачі динамічного відображення часових трендів енергоспоживання застосовується легковагова JavaScript-бібліотека Chart.js. Її інтеграція дозволяє генерувати плавні лінійні графіки безпосередньо на елементі HTML5 Canvas, ефективно використовуючи апаратне прискорення браузера [11]. Бібліотека містить вбудовані методи для керування життєвим циклом графіків, включаючи механізми усереднення точок та автоматичного очищення масивів історії, що надійно запобігає витокам оперативної пам'яті під час довготривалих сесій моніторингу.

3.2 Огляд апаратних засобів необхідних для розв'язання поставленої задачі

Базовим обчислювальним ядром розроблюваної Edge-системи обрано мікроконтролер ESP32 від компанії Espressif Systems. На відміну від класичних 8-бітних рішень (наприклад, плат сімейства Arduino Uno), архітектура ESP32 містить потужний 32-бітний процесор Xtensa, який забезпечує достатню обчислювальну швидкодію для одночасного виконання завдань апаратного опитування через порт UART та підтримки роботи асинхронного локального вебсервера. Наявність

інтегрованого модуля Wi-Fi стандарту 802.11 b/g/n дозволяє мікроконтролеру безперешкодно інтегруватися у домашню бездротову мережу (в режимі Station) без використання жодних додаткових зовнішніх мережевих шилдів чи модулів. Крім того, обсяг оперативної (SRAM) та енергонезалежної флеш-пам'яті ESP32 є ідеальним для розгортання інтерпретатора MicroPython та фізичного зберігання статичних файлів клієнтського вебзастосунку. Вагомим фактором при виборі стала також висока енергоефективність мікросхеми, власне споживання якої у робочому режимі не перевищує 0.5 Вт, що ідеально відповідає вимогам до автономних енергонезалежних систем.

Другим, не менш важливим апаратним компонентом, є спеціалізований модуль узгодження логічних рівнів. Комунікаційний інтерфейс промислового сонячного інвертора генерує сигнали за стандартом RS-232, напруга яких варіюється у біполярному діапазоні від -15 В до +15 В. Пряме підключення таких інформаційних ліній до контактів мікроконтролера ESP32, які розраховані на логіку транзисторно-транзисторної взаємодії (TTL) з максимальною напругою 3.3 В, гарантовано призведе до незворотного електричного пробоя та виходу з ладу процесора. Для усунення цього ризику обґрунтовано використання перетворювача рівнів на базі мікросхеми MAX3232. Цей апаратний модуль виконує безпечну інверсію та зниження амплітуди сигналів, забезпечуючи надійний гальванічний та логічний зв'язок між промисловим обладнанням та чутливою цифровою електронікою.

3.3 Функціональна модель та загальна архітектура системи

Для чіткого визначення функціональних меж розробленої системи було побудовано діаграму прецедентів (UML Use Case), де головним актором виступає «Адміністратор (Власник)». Ця діаграма відображає базові сценарії взаємодії користувача з програмним комплексом (рис. 3.1).

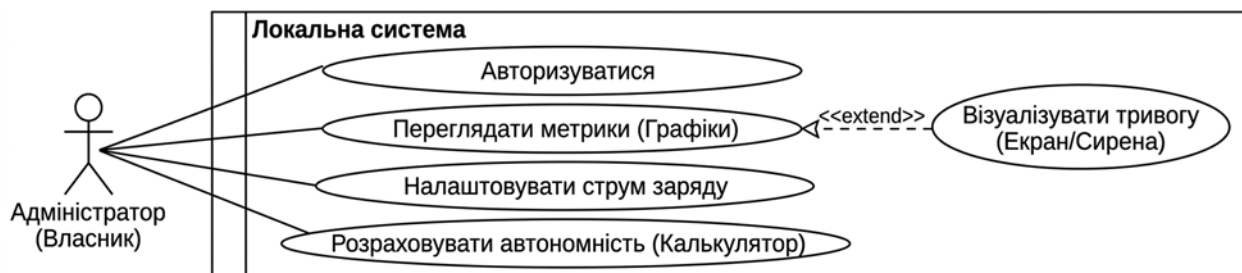


Рисунок 3.1 — Діаграма прецедентів системи моніторингу

Взаємодія із системою розпочинається з обов’язкової авторизації, що гарантує захист від несанкціонованого доступу. Після підтвердження прав адміністратор отримує можливість переглядати телеметричні метрики на динамічних графіках, дистанційно налаштовувати струм заряду інвертора та використовувати калькулятор для розрахунку часу автономної роботи. Окремо виділено прецедент візуалізації тривоги (спрацьовування візуального попередження та звукової сирени). Він є логічним розширенням (<<extend>>) базової функції перегляду метрик і активується автоматично у разі фіксації критичного перевантаження силової мережі.

Практична реалізація описаних прецедентів та комплексних систем Інтернету речей вимагає ретельного підбору технологічного стека, здатного безперебійно працювати в умовах жорстких апаратних обмежень мікроконтролерів і водночас забезпечувати високу швидкодію користувацького інтерфейсу. Загальну архітектуру програмно-апаратного комплексу, що ілюструє ієрархічну декомпозицію та послідовність проходження інформаційних потоків, наведено на рисунку 3.2.

Наведена структурна схема демонструє поділ системи на чотири взаємопов’язані рівні, кожен з яких виконує специфічну функцію для забезпечення цілісності даних на шляху від інвертора до кінцевого користувача.

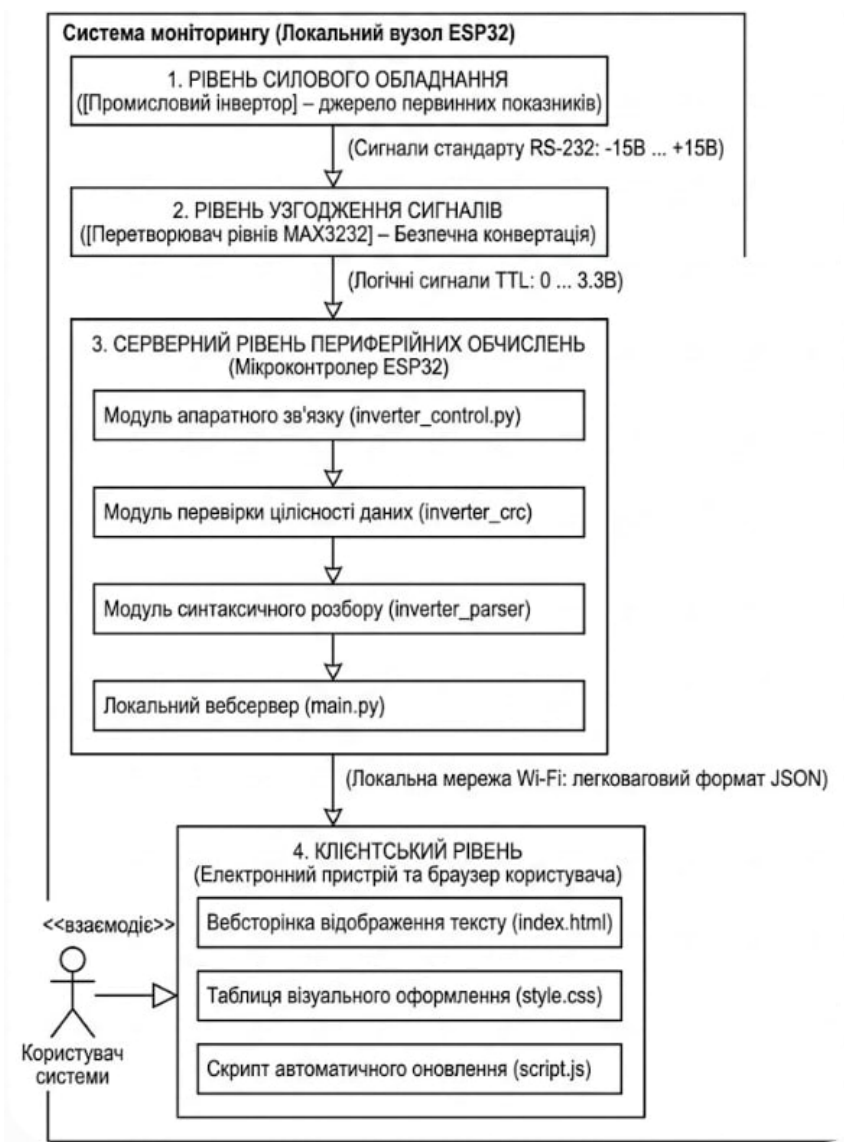


Рисунок 3.2 — Загальна структурна схема системи

Перший та другий рівні відповідають за генерацію первинних сигналів силовим обладнанням та їхню безпечну конвертацію зі стандарту RS-232 у логічні TTL-рівні за допомогою перетворювача MAX3232.

Третій (серверний) рівень реалізовано на базі мікроконтролера ESP32 з використанням концепції периферійних обчислень (Edge Computing). Він інкапсулює бізнес-логіку системи та включає модулі апаратного зв'язку, перевірки цілісності інформації (CRC), синтаксичного розбору пакетів та локальний вебсервер. Четвертий (клієнтський) рівень забезпечує візуалізацію телеметрії у

браузері користувача за допомогою базових вебтехнологій. Взаємодія між серверним та клієнтським рівнями відбувається через локальну мережу Wi-Fi з використанням легковагового формату передачі даних JSON.

3.4 Архітектура серверного програмного забезпечення

Традиційним підходом до програмування мікроконтролерів сімейства ESP32 є використання компільованих мов у середовищі Arduino IDE. Однак для реалізації даного проєкту було обрано платформу MicroPython. Цей вибір обґрунтований необхідністю швидкого прототипування, наявністю потужного вбудованого механізму збирання сміття для автоматичного управління пам'яттю та вбудованою підтримкою високорівневих структур даних, що критично важливо для серіалізації телеметрії.

Серверну частину реалізовано у вигляді синхронного мікросервера на базі стандартного модуля мережевих сокетів. Відмова від використання важких сторонніх вебфреймворків дозволила зекономити ресурси флешпам'яті та забезпечити максимальний контроль над обробкою HTTP-запитів на низькому рівні. Для розуміння організації програмного комплексу на фізичному рівні важливо розглянути склад його компонентів (рисунок 3.3).

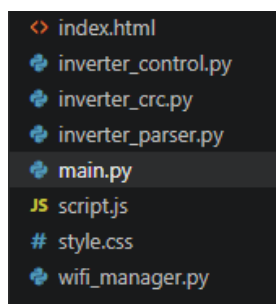


Рисунок 3.3 — Структура каталогів та файлів розробленого програмно-апаратного комплексу

Коренева директорія проєкту містить серверні скрипти на мові MicroPython та статичні файли клієнтського веб-інтерфейсу. Серверна логіка розподілена між вузькоспеціалізованими модулями: `wifi_manager.py` відповідає за ініціалізацію підключення мікроконтролера до локальної безпроводової мережі, `inverter_control.py` забезпечує низькорівневу апаратну взаємодію з інвертором через інтерфейс UART, `inverter_crc.py` генерує математичні контрольні суми для команд керування, а `inverter_parser.py` виконує лексичний аналіз та структурування отриманої телеметрії. Клієнтська частина представлена документами `index.html`, `style.css` та `script.js`, які разом формують інтерактивну графічну оболонку користувача та реалізують логіку математичного прогнозування. Запуск усього серверного контуру, ініціалізація компонентів та маршрутизація мережевих HTTP-запитів здійснюється через головний керівний файл `main.py`.

Архітектура сервера містить набір умовних маршрутів для віддачі статичних файлів інтерфейсу, запиту свіжих метрик та встановлення нових параметрів інвертора. Для забезпечення мережевої надійності сервер використовує апаратний механізм таймаутів.

Оскільки бездротові клієнти можуть несподівано втратити зв'язку по Wi-Fi під час передачі даних, сервер примусово закриває з'єднання, якщо запит не обробляється протягом заданого вікна у дві секунди, що гарантує захист від блокування головного циклу мікроконтролера.

Усі зазначені програмні артефакти розгорнуті у кореневій директорії мікроконтролера. Діаграма розгортання, наведена на рисунку 3.4, деталізує розміщення цих компонентів та їхню взаємодію у межах вузла ESP32.

Головним компонентом системи є контролер `main.py`, який ініціалізує вебсервер на стандартному 80-му порту та здійснює маршрутизацію вхідних HTTP-запитів. Отримавши запит від браузера, сервер звертається до модуля апаратного зв'язку, який ініціює обмін даними через UART-порт.

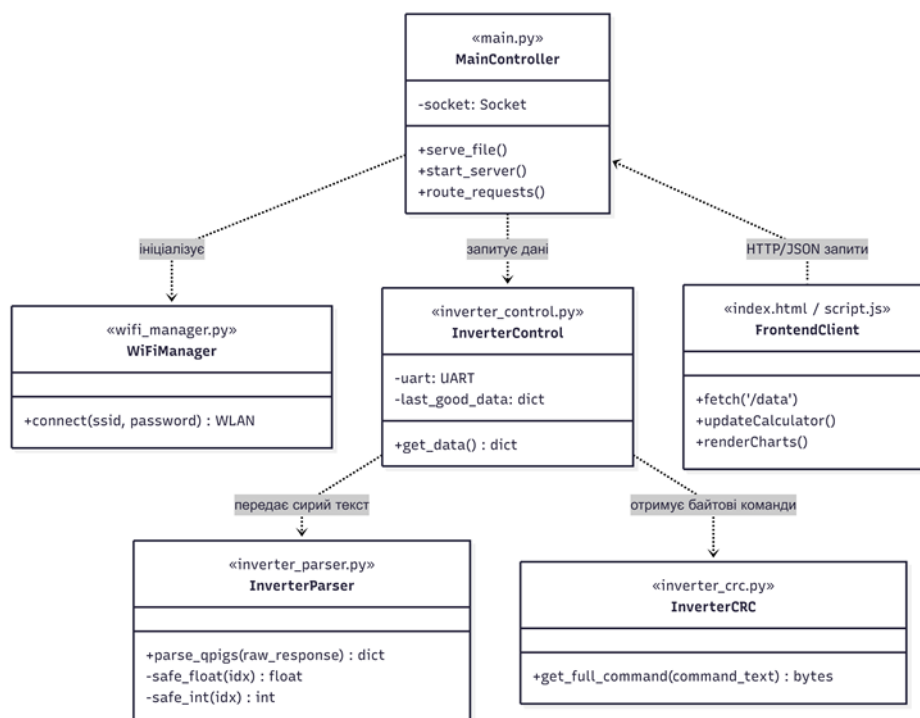


Рисунок 3.4 — Сценарій розгортання (Deployment Diagram) серверної частини

Важливим аспектом серверної реалізації є механізм жорстких таймаутів при роботі з мережевими сокетом. Це архітектурне рішення унеможливорює зависання мікроконтролера у випадках, коли клієнтський пристрій розриває сесію під час передачі даних, оскільки сервер примусово закриває з'єднання через дві секунди відсутності відповіді.

3.5 Реалізація апаратного зв'язку та синтаксичного розбору протоколу безперебійного джерела живлення

Наступним логічним рівнем серверної частини є модулі взаємодії з силовим обладнанням. Запит телеметрії ініціюється викликом відповідної функції з модуля `inverter_control.py`. Цей компонент відповідає за формування правильної послідовності байтів, яка вимагається закритим протоколом виробника інвертора.

Кожна команда, що відправляється до інвертора, наприклад, команда запиту загальних показників, повинна супроводжуватися розрахованою контрольною сумою. Цю задачу делеговано ізольованому математичному модулю `inverter_crc.py`, який за допомогою алгоритму циклічного надлишкового коду обчислює два контрольні байти та додає їх разом із символом повернення каретки до кінця рядка команди.

Після відправки сформованого запиту через апаратний порт мікроконтролера, система переходить у режим очікування відповіді. Отриманий від інвертора сирий текстовий рядок містить послідовність чисел, розділених пробілами, що вимагає складної лексичної обробки. Завдання токенизації та валідації покладено на модуль `inverter_parser.py`.

Розроблений алгоритм розбору передбачає послідовне розділення рядка на окремі фрагменти даних та їхнє безпечне приведення до числових типів. У випадку виникнення електромагнітних завад на лінії зв'язку, які можуть призвести до спотворення окремих символів, парсер перехоплює винятки перетворення типів і підставляє попередні валідовані значення, гарантуючи таким чином цілісність масиву даних перед його відправкою клієнту.

3.6 Клієнтська частина, візуалізація даних та інтерфейс користувача

Frontend-частина системи базується на стандартних технологіях HTML5, CSS3 та JavaScript, що забезпечує повну сумісність із мобільними браузерами. Взаємодія з сервером реалізована за допомогою Fetch API, який працює за моделлю асинхронного опитування. Клієнтський скрипт `script.js` виконує циклічні запити до endpoint-у `/data`, отримує структурований JSON-об'єкт та проводить оновлення значень у відповідних DOM-елементах.

Отримана інформація миттєво розподіляється між текстовими індикаторами на панелі приладів та модулем графічної візуалізації. Для побудови динамічних графіків напруги, струму та потужності інтегровано сторонню бібліотеку Chart.js.

Для вирішення проблеми надмірного споживання оперативної пам'яті браузером користувача під час тривалих сесій моніторингу, було розроблено алгоритм просторової децимації точок. Коли кількість накопичених вимірювань у масиві графіка перевищує визначений ліміт, алгоритм автоматично обчислює середнє арифметичне для кожної пари сусідніх елементів, зменшуючи розмір масиву вдвічі без втрати візуальної інформативності тренду.

На рисунку 3.5 наведено вигляд розробленого інтерфейсу користувача під час активного моніторингу інвертора.

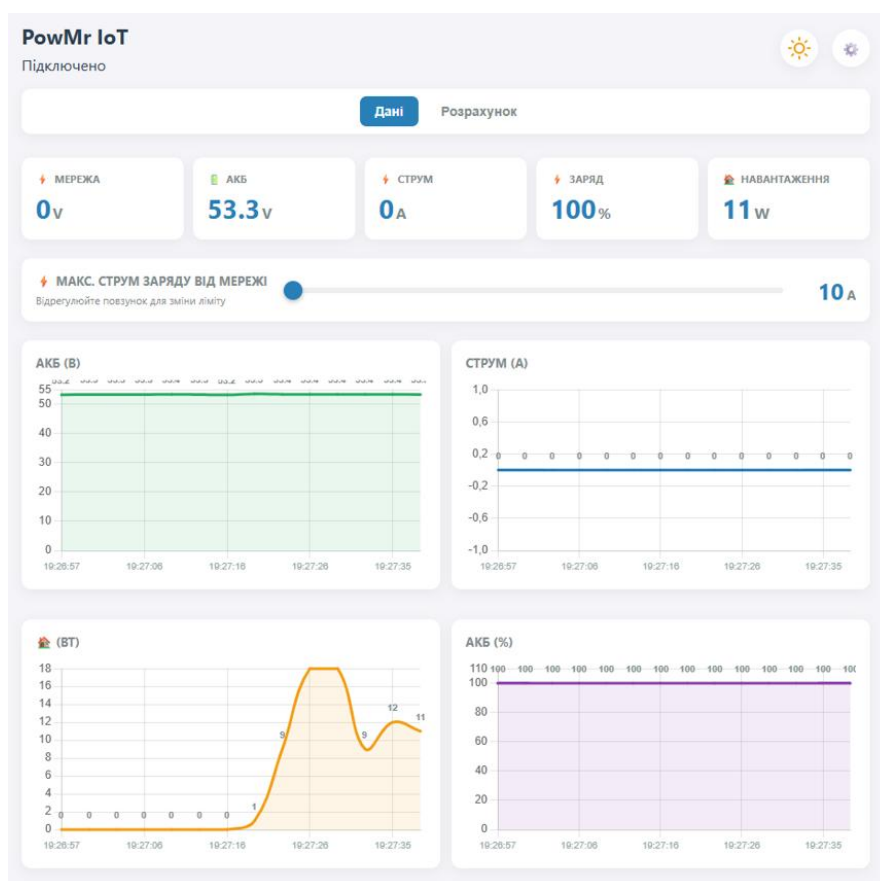


Рисунок 3.5 — Інтерфейс моніторингу безперебійного джерела живлення в реальному часі

Як показано на скриншоті, інтерфейс розділений на інформаційні блоки. У верхній частині відображаються поточні метрики, тоді як центральна зона відведена під графічну візуалізацію. Застосування алгоритму стиснення точок дозволяє уникати переповнення пам'яті браузера при тривалій роботі системи.

3.7 Модель представлення інформації та процес серіалізації

Ефективність комунікації між апаратним та клієнтським рівнями забезпечується оптимізованою моделлю представлення даних. Внутрішня модель у пам'яті мікроконтролера ESP32 реалізована у вигляді асоціативного масиву, де кожен параметр стану інвертора зберігається за унікальним текстовим ключем.

Типізація даних є суворою: параметри напруги змінного та постійного струму зберігаються як числа з плаваючою комою, тоді як показники потужності навантаження та відсоток заряду акумуляторної батареї представлені цілими числами. При отриманні мережевого запиту маршрутизатор викликає стандартну бібліотеку для серіалізації цього словника у текстовий формат об'єктної нотації.

Використання саме цього формату серіалізації забезпечує найвищу швидкість десеріалізації на стороні клієнта, оскільки цей формат є нативним для середовища виконання браузера.

Для зберігання конфіденційних налаштувань, таких як графічний ключ доступу до зміни параметрів інвертора, а також констант ємності акумуляторів для калькулятора, використовується локальне сховище браузера. Це забезпечує енергонезалежність налаштувань інтерфейсу і дозволяє системі миттєво відновлювати попередній робочий стан при оновленні сторінки без генерації зайвого мережевого трафіку.

3.8 Апаратна архітектура та топологія фізичного підключення вузлів

Програмна стабільність системи неможлива без надійного апаратного підґрунтя. Головною проблемою інтеграції сучасних мікроконтролерів із промисловим обладнанням є несумісність рівнів логічних сигналів.

Комунікаційний порт безперебійного джерела живлення працює з біполярними напругами, тоді як цифрові контакти ESP32 розраховані на низьковольтну логіку. Пряме фізичне з'єднання цих ліній неминуче призвело б до руйнування транзисторів мікроконтролера. Вирішення цієї задачі полягає у застосуванні проміжного модуля узгодження рівнів на базі мікросхеми MAX3232, яка забезпечує двонаправлену трансляцію сигналів. Принципову структурну схему електричного підключення розробленої системи наведено на рисунку 3.6.

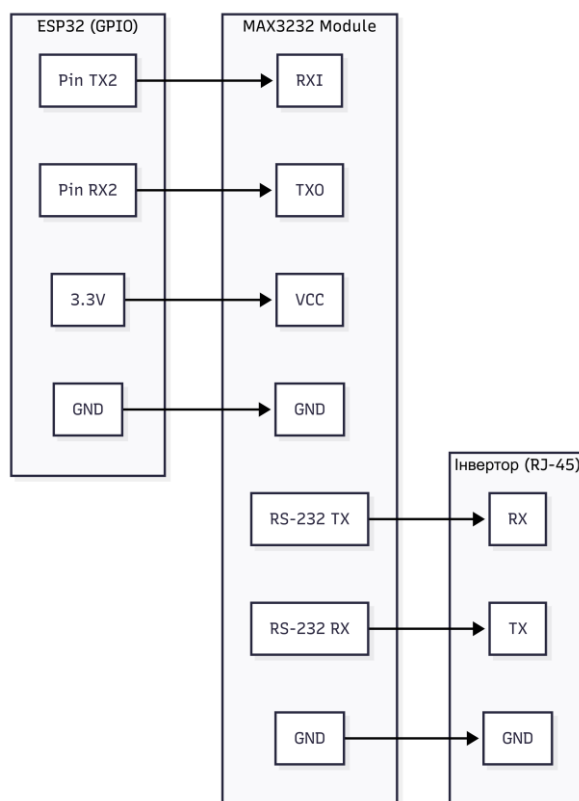


Рисунок 3.6 — Електрична схема підключення компонентів апаратної частини

Комутація інформаційних ліній виконана за перехресною (нуль-модемною) схемою: контакти передачі (TX) та приймання (RX) мікроконтролера з'єднані із зустрічними виводами перетворювача MAX3232. Аналогічне підключення реалізовано між інтерфейсом RS-232 перетворювача та комунікаційним портом інвертора. Для коректного узгодження потенціалів та мінімізації завад усі апаратні вузли об'єднані спільною шиною заземлення (GND). Живлення мікросхеми MAX3232 здійснюється напругою 3.3 В безпосередньо від вбудованого стабілізатора ESP32, що виключає потребу в додаткових джерелах живлення та оптимізує апаратну базу пристрою.

3.9 Динаміка інформаційної взаємодії та синхронізація процесів

Процеси обміну інформацією між апаратними та програмними вузлами розробленої системи підпорядковуються строгій хронологічній послідовності. Для специфікації динаміки програмної взаємодії та опису життєвого циклу інформаційної транзакції розроблено UML-діаграму послідовності. Порядок виклику функцій, передачі керування та повернення результатів обчислень протягом одного повного циклу опитування телеметрії відображено на рисунку 3.7.

Сеанс зв'язку розпочинається виключно на рівні клієнтського застосунку: програма перегляду формує та відправляє асинхронний мережевий запит до головного модуля маршрутизації на мікроконтролері. Обробник мережевих запитів передає виконання підпрограмі керування апаратним зв'язком. Цей модуль розпочинає формування байтової команди опитування з розрахованою контрольною сумою та виконує її передачу через послідовну шину до порту зв'язку силового обладнання. На час обробки запиту інвертором потік виконання мікроконтролера переходить у стан очікування вхідних даних у приймальній пам'яті із застосуванням фіксованого апаратного часового обмеження.

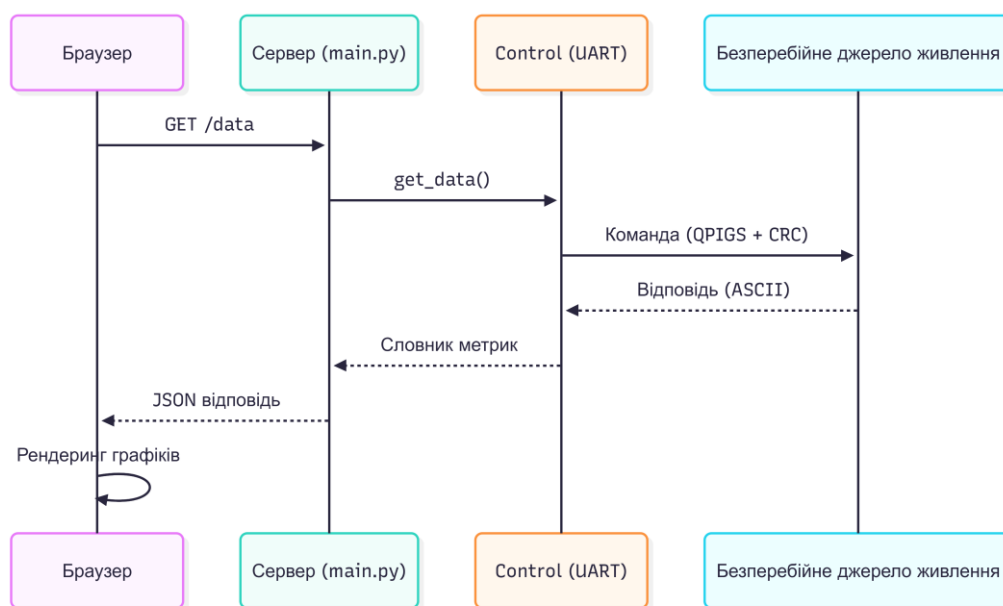


Рисунок 3.7 — UML-діаграма послідовності обміну даними між компонентами системи

Отриманий від силового обладнання пакет у вигляді неструктурованого текстового рядка перенаправляється до модуля синтаксичного аналізу. Аналізатор виконує процедури лексичної перевірки, поділу інформації та безпечної конвертації строкових значень у відповідні числові типи, після чого генерує типізований асоціативний масив метрик. Валідований словник повертається до контролера вищого рівня, серіалізується у текстовий формат JSON та відправляється по бездротовому каналу Wi-Fi як тіло HTTP-відповіді.

Після приймання пакета клієнтський скрипт десеріалізує об'єкт та викликає процедури точкового оновлення об'єктної моделі документа (DOM) і перерахунку координат для візуалізації графіків. Описаний алгоритм виконується циклічно із заданим періодом. Така транзакційна модель забезпечує строге розділення рівнів абстракції: клієнтський інтерфейс повністю ізольований від керування фізичними портами, а обчислювальне ядро мікроконтролера захищене від виникнення колізій під час паралельної обробки мережевих та апаратних переривань.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

На завершальному етапі створення обчислювальної системи критично важливим кроком є перевірка її працездатності в реальних умовах експлуатації. У даному розділі наведено розгорнутий опис процесу розгортання та послідовного запуску програмного забезпечення на цільовій апаратній платформі. Також визначено мінімальні системні вимоги до обладнання кінцевого користувача та детально продемонстровано всі функціональні можливості створеної графічної оболонки.

На основі проведених натурних експериментів підтверджено повну відповідність розробленого комплексу початковим вимогам технічного завдання. Додатково здійснено аналіз експлуатаційних переваг створеного локального рішення у порівнянні з наявними комерційними системами, що функціонують виключно завдяки підключенню до віддалених серверів обробки даних.

4.1 Підготовка фізичного випробувального стенда, інсталяція програмного забезпечення та ініціалізація системи

Для забезпечення коректного функціонування розробленого комплексу необхідно дотримуватися визначеної послідовності підготовки апаратного та програмного забезпечення. Першим кроком є збірка фізичного випробувального стенда. До його складу увійшли базовий обчислювальний мікроконтролер, плата узгодження логічних рівнів, промислове джерело безперебійного живлення та ПК.

Апаратні вимоги до обчислювального вузла передбачають наявність щонайменше чотирьох мегабайтів вбудованої енергонезалежної пам'яті для

розміщення середовища виконання. Окрім цього, критично важливим є використання стабілізованого джерела живлення мікроконтролера.

Усі компоненти стенда обов'язково об'єднані спільною шиною нульового потенціалу. Створення єдиного контуру є критичною вимогою для запобігання пошкодженню логічних входів мікроконтролера. Загальний вигляд зібраного апаратного стенда наведено на рисунку 4.1.

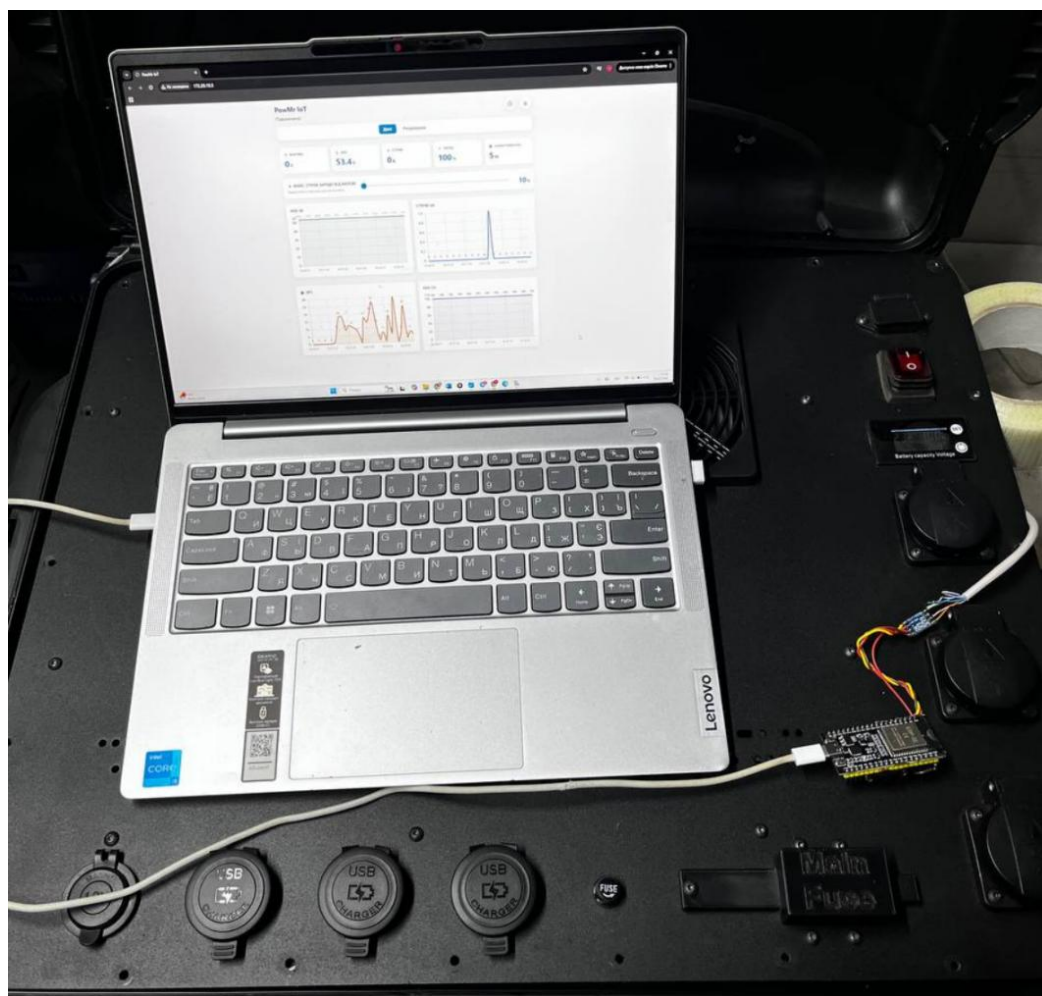


Рисунок 4.1 — Фотографія фізичного випробувального стенда

Процес розгортання розпочинається із запису мікропрограми базової системи введення-виведення у пам'ять контролера. Після успішного створення файлової системи, у кореневий каталог пристрою копіюються розроблені файли

математичної логіки, алгоритми керування мережею та документи візуального оформлення.

З боку кінцевого користувача для взаємодії з системою підходить будь-який електронний пристрій, оснащений сучасною програмою для перегляду вебсторінок. Відсутність вимог до встановлення спеціалізованих клієнтських програм робить систему максимально універсальною та незалежною від операційних систем користувачів.

Процес запуску розробленого комплексу є багатоетапним та включає послідовну ініціалізацію апаратних підсистем мікроконтролера. Відразу після подачі електричного живлення апаратне ядро пристрою виконує процедуру початкового скидання.

Далі завантажується середовище виконання, яке автоматично здійснює пошук та запуск головного керівного файлу системи. На цьому етапі обчислювальне ядро розпочинає конфігурацію безпроводового модуля зв'язку, перемикаючи його у режим клієнтської станції.

Програмний сценарій керування мережею виконує пошук збереженої точки доступу та ініціює процес встановлення зв'язку з маршрутизатором. Увесь цей процес прозоро відображається у консолі середовища розробки. Зокрема, на діагностичному екрані можна спостерігати повідомлення про спроби підключення, а у разі виникнення затримок - інформацію про зайнятість пристрою внутрішніми процесами.

Після успішного отримання локальної мережевої адреси система переходить до ініціалізації комунікаційних портів. Створюється програмний мережевий вузол, який починає прослуховувати вхідні запити. Одночасно з цим налаштовується апаратний послідовний порт зв'язку з силовим інвертором. Для підтвердження повної працездатності фізичного з'єднання система виконує перше тестове опитування обладнання.

```

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:2
load:0x3fff0030,len:4200
load:0x40078000,len:15236
load:0x40080400,len:3332
entry 0x400805a8
Підключення до мережі 'iPhone'...
..
Device is busy or does not respond. Your options:

- wait until it completes current work;
- use Ctrl+C to interrupt current work;
- reset the device and try again;
- check connection properties;
- make sure the device has suitable MicroPython / CircuitPython / firmware;
- make sure the device is not in bootloader mode.

...
Успішно підключено!
IP адреса сторінки: http://172.20.10.5
Сервер запущено...

[UART RAW BYTES]: b'(000.0 00.0 229.8 50.0 0183 0008 002 399 53.00 000 100 0024 00.0 000.0 00.00 00001 00
010000 00 00 00000 010D \r'

```

Рисунок 4.2 — Завантаження системи, отримання мережевої адреси та приймання першого масиву даних

На рисунку 4.2 зафіксовано масив неструктурованих байтів, отриманих від безперебійного джерела живлення під час першого циклу опитування, де простежуються первинні показники напруги. Наявність цього масиву даних у консолі є стовідсотковим індикатором успішного налагодження апаратного зв'язку.

Успішне зчитування першого інформаційного масиву, продемонстроване на діагностичному екрані, є фактичним підтвердженням правильного узгодження логічних рівнів та коректного налаштування протоколу обміну даними між пристроями.

З цього моменту обчислювальне ядро переходить у штатний режим безперервного циклічного опитування силового обладнання. Водночас програмний мережевий вузол розпочинає прослуховування призначеного порту в очікуванні вхідних з'єднань.

Цей стан сигналізує про повне та успішне завершення етапу апаратно-програмної ініціалізації, що означає стовідсоткову готовність комплексу до

активної експлуатації та візуальної взаємодії з користувачем через розроблену панель керування.

4.2 Демонстрація функціоналу та розширені можливості користувацької взаємодії

Перший сеанс зв'язку користувача із системою розпочинається з етапу конфігурації та налаштування безпеки. Це фундаментальний елемент захисту параметрів силового обладнання від несанкціонованого втручання. Процес первинного налаштування розпочинається із введення базових параметрів обладнання у відповідному діалоговому вікні, зовнішній вигляд якого наведено на рисунку 4.3.

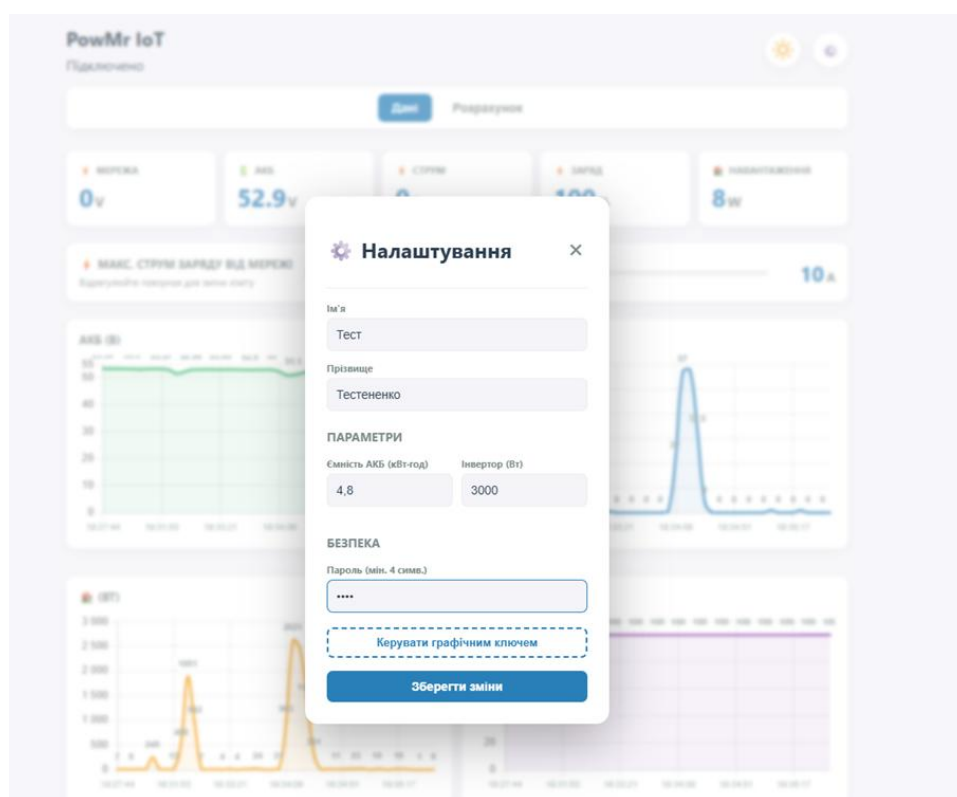


Рисунок 4.3 — Вікно налаштування базових параметрів системи

Розроблена підсистема безпеки передбачає можливість використання графічного ключа замість традиційних символічних паролів. Математичний алгоритм надійно зберігає вектор руху у локальній пам'яті програми перегляду вебсторінок, не передаючи його у відкритому вигляді через мережу.

Завершальним етапом ініціалізації безпеки є екран автентифікації. Подальший вхід до панелі керування стає можливим виключно після введення правильного графічного ключа, що гарантує захист від стороннього втручання (рисунок 4.4).

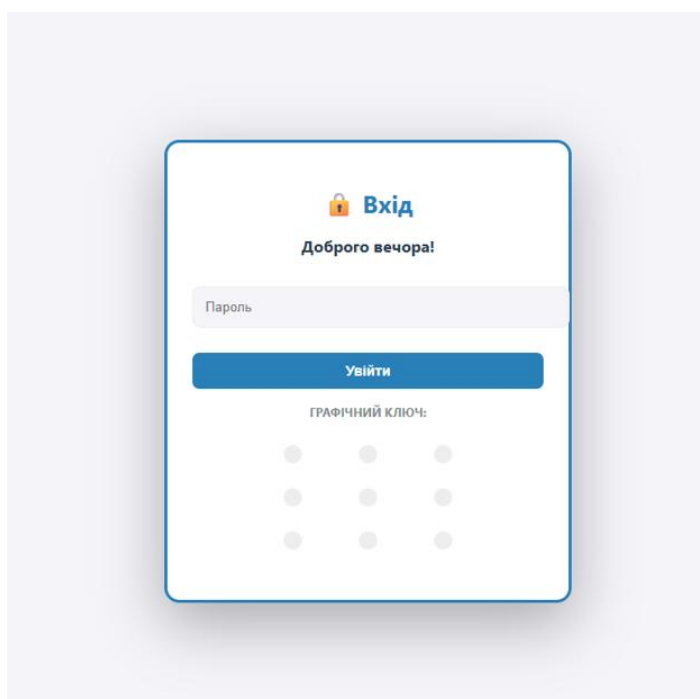


Рисунок 4.4 — Екран автентифікації користувача перед доступом до системи

Після успішної перевірки прав доступу користувач потрапляє на головну інформаційну панель. Для зниження навантаження на зір реалізовано підтримку миттєвого перемикавання між світлою та темною темами оформлення. У стані очікування або за відсутності фізичного з'єднання з обладнанням інтерфейс коректно відображає нульові значення (рисунок 4.5) інформуючи користувача про відсутність потоку даних.

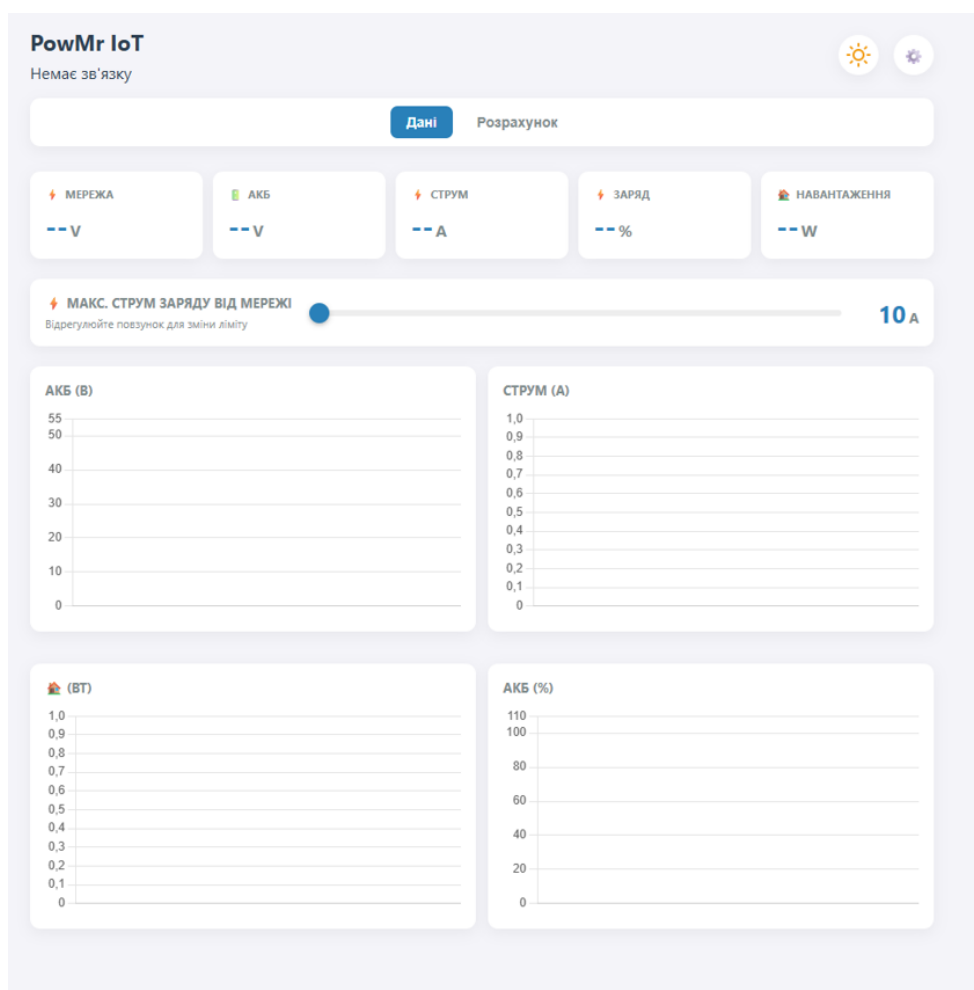


Рисунок 4.5 — Базовий стан головної панелі за відсутності з'єднання

Після налагодження зв'язку динамічні графіки починають безперервно оновлюватися, відображаючи коливання електричних метрик. У верхньому інформаційному блоці зосереджені ключові індикатори: наявність напруги зовнішньої мережі, фактична напруга на клеммах акумуляторного блоку, активний струм, відсотковий рівень заряду та сумарна потужність підключеного навантаження.

Одразу під ними розташовано інтерактивний повзунок керування, що дозволяє дистанційно змінювати ліміт зарядного струму. Зовнішній вигляд панелі моніторингу у штатному робочому режимі при базовому навантаженні продемонстровано на рисунку 4.6.

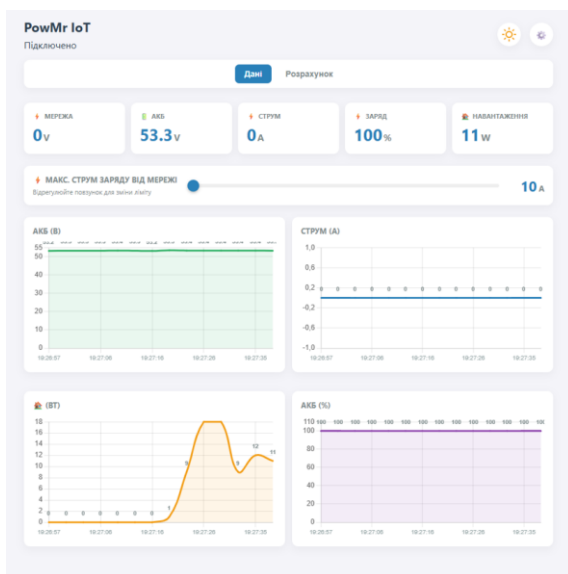


Рисунок 4.6 — Відображення телеметрії у реальному часі

При підключенні додаткових споживачів графіки та індикатори миттєво адаптуються до нових значень. Роботу інтерфейсу під час підвищеного навантаження зафіксовано на рисунку 4.7.

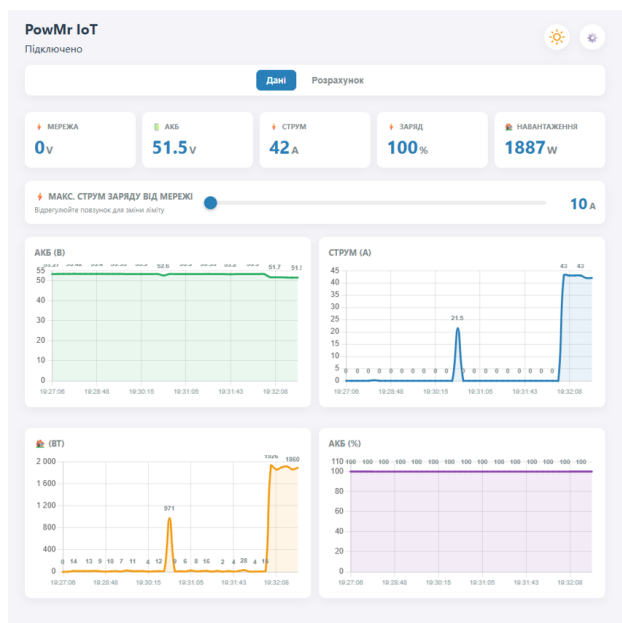


Рисунок 4.7 — Відображення телеметрії у реальному часі після підключення приладів

Окрему увагу в розробленій системі приділено програмному захисту обладнання. Під час тестування було створено ситуацію різкого перевищення номінальної потужності інвертора. Алгоритм системи миттєво розпізнає критичне значення. У такому випадку користувацька оболонка повністю блокує взаємодію з інтерфейсом, виводячи на екран яскраве попереджувальне вікно (рисунок 4.8), що вимагає негайного втручання для уникнення апаратної аварії.

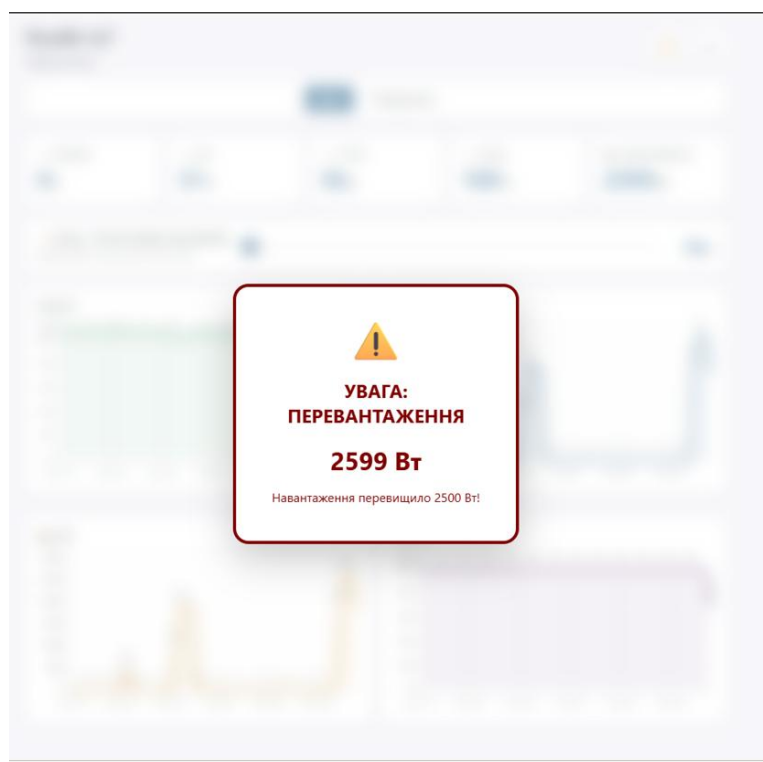


Рисунок 4.8 — Спрацювання програмного алгоритму захисту

Окремим вагомим компонентом програмного забезпечення є модуль математичного прогнозування часу автономної роботи. Цей розділ дозволяє безпечно планувати енергоспоживання під час тривалих відключень зовнішньої мережі. Робота алгоритму базується на попередньо введених користувачем константах ємності батарей та потужності інвертора. Користувач формує перелік побутових приладів, вказуючи їхню споживану потужність. Приєднання приладів,

сумарна потужність яких не перевищує безпечний ліміт, забезпечує стабільний прогнозований час роботи, що показано на рисунку 4.9.

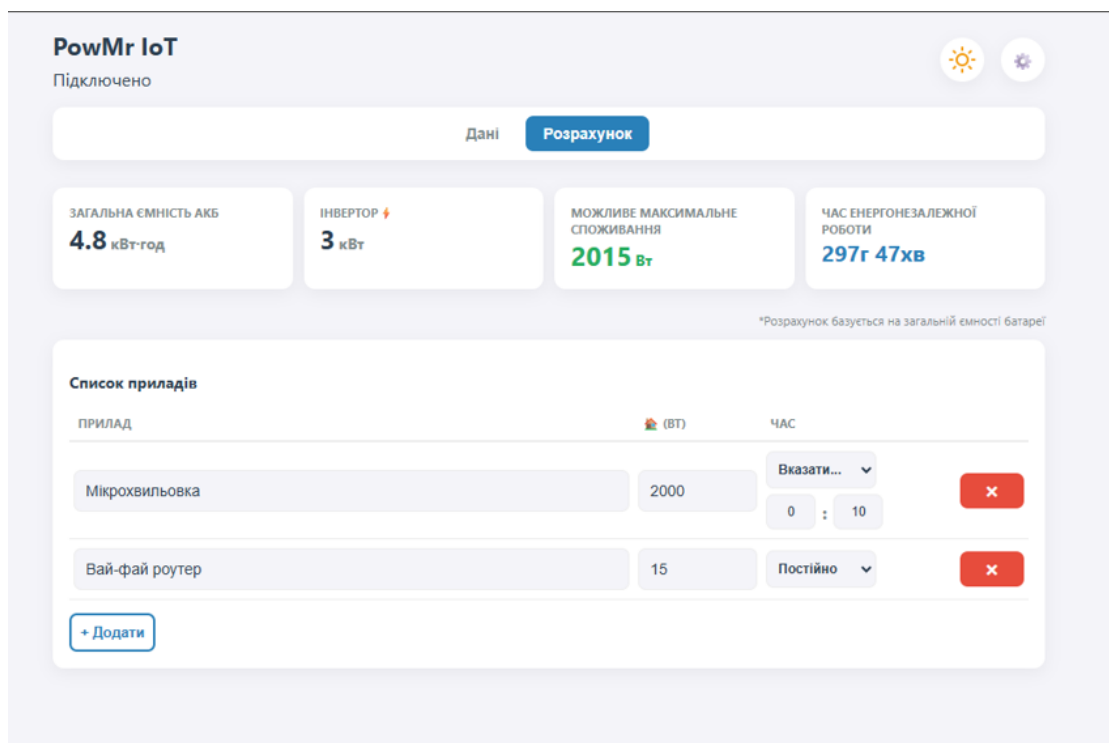


Рисунок 4.9 — Моделювання сценаріїв автономної роботи

У випадку спроби моделювання підключення приладів, сумарна потужність яких фізично перевищує номінальні можливості інвертора, алгоритм миттєво фіксує перевищення допустимих експлуатаційних лімітів.

У такій ситуації система сигналізує про критичну нестачу ресурсів, автоматично забарвлюючи розрахункові індикатори у попереджувальний червоний колір.

Цей механізм візуального зворотного зв'язку виконує важливу превентивну функцію: він наочно демонструє користувачеві неможливість підтримки заданого навантаження та допомагає уникнути потенційних аварійних відключень обладнання в реальних умовах. (рисунок 4.9).

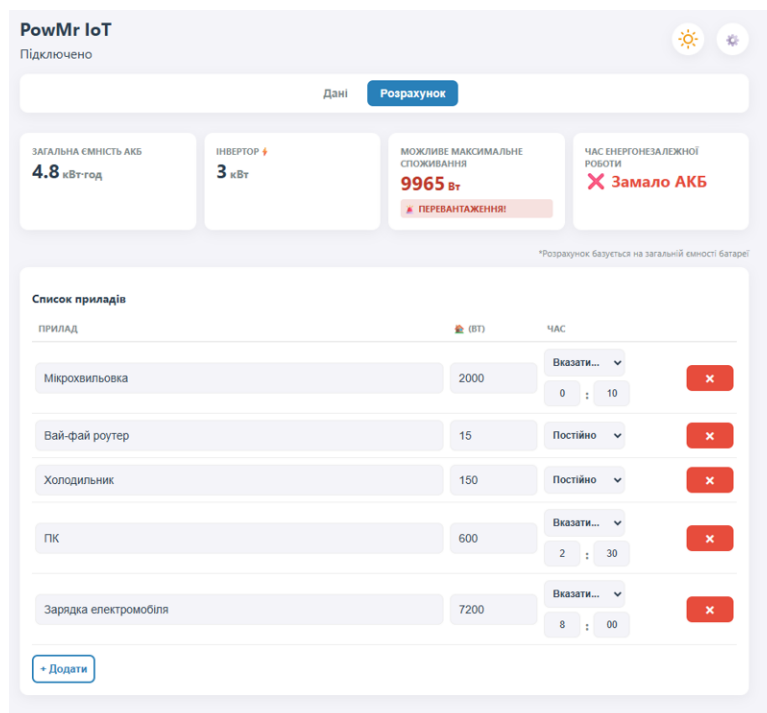


Рисунок 4.9 — Індикація критичного перенавантаження

Описана логіка взаємодії підтверджує високу інформативність розробленої графічної оболонки. Наявність вбудованої системи віртуального моделювання навантажень дозволяє кінцевому користувачу без ризику для обладнання тестувати різні сценарії енергоспоживання, запобігаючи реальним аварійним відключенням під час експлуатації комплексу.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра вирішено актуальне науково-практичне завдання, що полягає у створенні надійної, незалежної від глобальної мережі системи для збору телеметрії та керування силовим енергетичним обладнанням. На основі аналізу підходів та алгоритмів розв’язання задачі локального моніторингу обґрунтовано доцільність використання архітектури периферійних обчислень. Для забезпечення надійної взаємодії застосовано методи асинхронного мережевого опитування, перевірку цілісності пакетів за допомогою циклічного надлишкового коду, просторову децимацію точок для оптимізації відмальовування графіків та векторний графічний ключ для безпечної автентифікації.

Аналіз сучасних програмно-апаратних засобів для екосистеми Інтернету речей дозволив обґрунтувати вибір мікроконтролера ESP32 у поєднанні з мікросхемою узгодження логічних рівнів MAX3232. Таке рішення гарантує безпечну комунікацію з промисловим обладнанням через послідовний інтерфейс. Серверну частину реалізовано на базі інтерпретатора MicroPython для ефективного управління пам’яттю, а клієнтську оболонку побудовано з використанням базових вебтехнологій та інтерфейсу Fetch API, що дозволило відмовитися від громіздких фреймворків.

Практичним результатом роботи стала розроблена програмно-апаратна система локального моніторингу та управління джерелом безперебійного живлення. Створений комплекс успішно реалізує функціонал дистанційного керування лімітами зарядного струму та містить багаторівневий алгоритм програмного захисту. Крім того, розроблено унікальний аналітичний модуль, який дозволяє віртуально моделювати підключення побутових приладів і прогнозувати час автономної роботи з урахуванням загальної ємності акумуляторних батарей.

Коректність роботи створеного програмно-апаратного забезпечення повністю доведено під час натурного тестування на фізичному випробувальному стенді. Обчислювальні експерименти підтвердили високу стабільність

обчислювального ядра мікроконтролера без витоків оперативної пам'яті та мінімальні мережеві затримки на рівні часток секунди. Натурні випробування також засвідчили надійність механізму автентифікації та безпомилкове спрацювання алгоритмів автоматичного блокування інтерфейсу при фіксації перевантаження.

Подальші перспективи вдосконалення розробленої системи полягають в інтеграції підтримки протоколу MQTT для об'єднання комплексу з глобальними екосистемами «розумного будинку». Додатково доцільною є реалізація підсистеми довготривалого збереження історичної телеметрії на зовнішні фізичні носії та розширення модулів синтаксичного розбору для забезпечення сумісності з протоколами інверторів інших виробників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 2016. Vol. 3, No. 5. P. 637–646. URL: https://cse.buffalo.edu/faculty/tkosar/cse710_spring20/shi-iot16.pdf (date of access: 03.06.2026).
2. Edge computing for IoT / Bosch Global Software Technologies GmbH. 2023. URL: https://bosch-iot-suite.com/wp-content/uploads/20231025_white-paper_edge-computing_EN-1.pdf (date of access: 03.06.2026).
3. UART: A Hardware Communication Protocol. *Analog Devices*. URL: <https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html> (date of access: 03.06.2026).
4. Voltronic Power Inverter Communication Protocol. *GitHub*. URL: <https://github.com/ned-kelly/docker-voltronic-homeassistant> (date of access: 03.06.2026).
5. ESP32 Series Datasheet. Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (date of access: 03.06.2026).
6. Бабич Г. В. Мікропроцесорна техніка: навч. посіб. Дніпро: НТУ «Дніпровська політехніка», 2015. 144 с. URL: <https://elprivod.nmu.org.ua/files/microcontrollers/%D0%9D%D0%A2%D0%91452919.pdf> (дата звернення: 03.06.2026).
7. Introducing JSON. *JSON*. URL: <https://www.json.org/json-en.html> (date of access: 03.06.2026).
8. MicroPython documentation. *MicroPython*. URL: <https://docs.micropython.org/> (date of access: 03.06.2026).
9. HTML: HyperText Markup Language. *MDN Web Docs*. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (date of access: 03.06.2026).

10. Fetch API. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (date of access: 03.06.2026).

11. Chart.js documentation. *Chart.js*. URL: <https://www.chartjs.org/docs/latest/> (date of access: 03.06.2026).

12. JavaScript. *MDN Web Docs*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (date of access: 03.06.2026).

ДОДАТОК А

Реалізація програмно-апаратного комплексу локального моніторингу та управління джерелом безперебійного живлення

Програмні засоби:

- мови програмування – MicroPython, JavaScript, HTML, CSS;
- організація мережевої взаємодії – модуль socket (MicroPython);
- апаратна комунікація – клас UART з модуля machine;
- обробка даних – модуль json;
- візуалізація телеметрії – бібліотека Chart.js.

Файл `wifi_manager.py`

Файл `wifi_manager.py` відповідає за ініціалізацію безпроводового інтерфейсу мікроконтролера у режимі клієнтської станції (STA_IF). У ньому реалізовано логіку:

- активації мережевого модуля;
- встановлення з'єднання із заданою точкою доступу Wi-Fi;
- очікування підтвердження авторизації в мережі;
- отримання та виведення локальної IP-адреси пристрою.

Цей модуль забезпечує базову мережеву доступність комплексу для подальшого розгортання вебсервера.

```
import network
import time

def connect(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)

    print(f'Підключення до мережі '{ssid}'...')
```

```

while not wlan.isconnected():
    time.sleep(0.5)
    print(".", end="")

ip_address = wlan.ifconfig()[0]
print("\nУспішно підключено!")
print("IP адреса сторінки: http://" + ip_address)

return wlan

```

Файл main.py

Файл main.py є основною точкою входу до серверної частини системи, реалізованої на базі стандартних сокетів MicroPython. У ньому налаштовано прослуховування порту 80 та ініціалізовано маршрути для:

- отримання актуальної телеметрії інвертора у форматі JSON (/data);
- дистанційного встановлення лімітів струму заряду (/set_charge);
- роздачі статичних файлів візуального оформлення (/style.css);
- завантаження клієнтської логіки (/script.js);
- відображення головної сторінки інтерфейсу (/index.html або /).

Кожен маршрут обробляє вхідні HTTP-запити та взаємодіє з апаратними модулями для керування інвертором.

```

import socket
import json
import time
import wifi_manager
import inverter_control
import inverter_crc

wlan = wifi_manager.connect("Redmi Note 13", "754836192013795631415926535")

addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(addr)
s.listen(5)
print('Сервер запущено...')

```

```

def serve_file(cl, filename, content_type):
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            content = f.read()
            cl.send(b'HTTP/1.1 200 OK\r\n')
            cl.send(f'Content-Type: {content_type}\r\n'.encode('utf-8'))
            cl.send(b'Connection: close\r\n\r\n')
            cl.sendall(content.encode('utf-8'))
    except Exception:
        cl.send(b'HTTP/1.1 404 Not Found\r\n\r\n')

while True:
    try:
        cl, addr_info = s.accept()
        cl.settimeout(2.0)

        try:
            request_bytes = cl.recv(1024)
        except Exception:
            request_bytes = b''

        if not request_bytes:
            cl.close()
            continue

        request = request_bytes.decode('utf-8', 'ignore')

        if 'GET /data' in request:
            data = inverter_control.get_data()
            cl.send(b'HTTP/1.1 200 OK\r\nContent-Type: application/json\r\nConnection: close\r\n\r\n')
            cl.sendall(json.dumps(data).encode('utf-8'))

        elif 'GET /set_charge' in request:
            try:
                val_str = request.split('val=')[1].split(' ')[0]
            except:
                val_str = "010"

            command_str = "MCHGC" + val_str
            cmd = inverter_crc.get_full_command(command_str)

            while inverter_control.uart.any():
                inverter_control.uart.read()

            inverter_control.uart.write(cmd)
            time.sleep(0.5)

            response = inverter_control.uart.read()
            cl.send(b'HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nConnection: close\r\n\r\n')

```

```

cl.sendall(b'OK')

elif 'GET /style.css' in request:
    serve_file(cl, 'style.css', 'text/css')
elif 'GET /script.js' in request:
    serve_file(cl, 'script.js', 'application/javascript')
elif 'GET / ' in request or 'GET /index.html' in request:
    serve_file(cl, 'index.html', 'text/html')

cl.close()
except Exception as e:
    print("Помилка циклу сервера:", e)
try: cl.close()
except: pass

```

Файл inverter_control.py

Файл inverter_control.py відповідає за низькорівневу апаратну взаємодію з інвертором через послідовний порт (UART). У ньому реалізовано:

- ініціалізацію комунікаційного порту із заданою швидкістю (2400 бод);
- очищення приймальних буферів перед запитом;
- відправку сформованих команд до силового обладнання;
- зчитування відповіді з урахуванням апаратних таймаутів;
- первинну обробку (очищення) сирих байтів для подальшого парсингу.

```

from machine import UART
import time
import inverter_crc
import inverter_parser

uart = UART(2, baudrate=2400, tx=17, rx=16)

last_good_data = {
    "grid_voltage": 0.0, "load_power": 0, "batt_voltage": 0.0,
    "charge_current": 0.0, "batt_percent": 0, "status": "disconnected"
}

def get_data():
    global last_good_data

```

```

# 1. Очищуємо порт
while uart.any():
    uart.read()

# 2. Відправляємо запит
uart.write(inverter_crc.get_full_command("QPIGS"))

# 3. Читаємо відповідь
start_time = time.time()
raw_bytes = b''

while time.time() - start_time < 1500:
    if uart.any():
        raw_bytes += uart.read()
        if b'\r' in raw_bytes:
            break
    time.sleep(0.05)

if raw_bytes:
    try:
        if raw_bytes.endswith(b'\r') and len(raw_bytes) > 5:
            clean_bytes = raw_bytes[:-3]
        else:
            clean_bytes = raw_bytes

        raw_text = clean_bytes.decode('ascii', 'ignore')

        # Шукаємо початок даних
        if '(' in raw_text:
            clean_text = raw_text[raw_text.find('('):]
            parsed = inverter_parser.parse_qpigs(clean_text)

            if parsed and parsed.get('batt_voltage', 0) > 0:
                last_good_data.update(parsed)
                last_good_data['status'] = "connected"
                return last_good_data

    except Exception as e:
        print(f'[ПОМИЛКА] Збій у get_data: {e}')

last_good_data['status'] = "disconnected"
return last_good_data

```

Файл inverter_parser.py

Файл inverter_parser.py містить програмний модуль лексичного аналізу, що виконує синтаксичний розбір відповідей інвертора. У ньому реалізовано:

- валідацію початкового символу пакета;
- розділення текстового рядка на масив окремих значень;
- безпечне вилучення числових показників напруги мережі та акумулятора;
- розрахунок активного струму (заряду або розряду).

Алгоритм формує структурований словник (JSON-об'єкт), який використовується для подальшої відправки на клієнтську частину.

```
def parse_qpigs(raw_response):
    try:
        if raw_response.startswith('('):
            clean_str = raw_response[1:].strip()
            data = clean_str.split(' ')

            def safe_float(idx):
                if idx < len(data):
                    try:
                        return float(data[idx])
                    except ValueError:
                        return 0.0
                return 0.0

            def safe_int(idx):
                if idx < len(data):
                    try:
                        return int(data[idx])
                    except ValueError:
                        return 0
                return 0

            # Струм зарядки та розрядки
            charge_c = safe_float(9)
            discharge_c = safe_float(15)
            active_current = charge_c if charge_c > 0 else discharge_c

            return {
                "grid_voltage": safe_float(0),
                "load_power": safe_int(5),      # Вт (Навантаження)
                "batt_voltage": safe_float(8),  # Напруга
                "charge_current": active_current, # Струм
                "batt_percent": safe_int(10)    # Відсотки
            }
        except Exception as e:
            print("[ПАРСЕР] Помилка обробки:", e)

    return None
```

Файл `inverter_crc.py`

Файл `inverter_crc.py` реалізує алгоритм перевірки цілісності інформаційних пакетів. У ньому виконується:

- побайтове зчитування текстової команди;
- розрахунок двобайтової контрольної суми за методом циклічного надлишкового коду (CRC);

- коригування специфічних байтів відповідно до вимог протоколу інвертора;
- формування фінального пакета з додаванням символу повернення каретки.

Цей модуль гарантує, що жодна команда не буде відхилена інвертором через помилки передачі даних.

```
def get_full_command(command_text):
    crc = 0
    for char in command_text:
        crc = crc ^ (ord(char) << 8)
        for _ in range(8):
            if crc & 0x8000:
                crc = (crc << 1) ^ 0x1021
            else:
                crc = crc << 1
        crc &= 0xFFFF

    crc_high = (crc >> 8) & 0xFF
    crc_low = crc & 0xFF

    # Специфічне правило протоколу: CRC не може бути певними керуючими символами
    if crc_high in (0x28, 0x0D, 0x0A):
        crc_high += 1
    if crc_low in (0x28, 0x0D, 0x0A):
        crc_low += 1

    # Формуємо фінальний пакет: команда + 2 байти CRC + символ повернення каретки
    (0x0D)
    return command_text.encode('ascii') + bytes([crc_high, crc_low, 0x0D])
```