Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Система моніторингу апаратно-програмних компонентів комп'ютерної техніки*

Виконав: студент 4 курсу, групи СІ-41
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

Балаєв В. А.
(підпис) (прізвище та ініціали)

Керівник *Лещишин Ю. З.*
(підпис) (прізвище та ініціали)

Нормоконтроль *Луцик Н. С.*
(підпис) (прізвище та ініціали)

Завідувач кафедри *Осухівська Г.М.*
(підпис) (прізвище та ініціали)

Рецензент *Дмитроца Л. П.*
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«25» квітня 2026 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студента *Балаєва Віталія Андрійовича*
(прізвище, ім'я, по батькові)

1. Тема роботи *Система моніторингу апаратно-програмних компонентів комп'ютерної техніки*

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

Затверджені наказом ректора від *« 24 » квітня 2026 року № 4/9-188*
2. Термін подання студентом завершеної роботи *15.06.2026р.*
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання на розробку системи моніторингу компонентів
2. Проєктування системи моніторингу апаратно-технічних компонентів комп'ютера
3. Програмування та тестування системи моніторингу компонентів комп'ютера
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Структурна схема системи
2. Схема електрична принципова
3. Блок-схема алгоритму роботи програмного забезпечення
4. Блок-схема алгоритму роботи пристрою

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*доц. каф. МТ Сенчишин В. С.*		

7. Дата видачі завдання *25.04.2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*26.01 – 02.02*	
2.	*Аналіз технічного завдання на розробку системи моніторингу компонентів*	*03.02 – 15.02*	
3.	*Проєктування системи моніторингу апаратно-технічних компонентів комп'ютера*	*20.04 – 10.05*	
4.	*Програмування та тестування системи моніторингу апаратно-технічних компонентів комп'ютера*	*11.05 – 24.05*	
5.	*Безпека життєдіяльності, основи охорони праці*	*25.05 – 31.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*1.06 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*	*22.06*	

Студент _____ *Балаєв Віталій Андрійович*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Лещишин Юрій Зіновійович*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Балаєв В. А. Система моніторингу апаратно-програмних компонентів комп’ютерної техніки: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 123 — комп’ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2026.

Ключові слова: система моніторингу, компоненти ПК, STM32F769I-DISCO, Bluetooth Low Energy, HM-10, WPF, LibreHardwareMonitor, LVGL, AES-128, LCD-дисплей.

Кваліфікаційна робота присвячена розробці системи моніторингу апаратно-технічних компонентів комп’ютерної техніки на базі плати STM32F769I-DISCO з використанням Bluetooth-модуля HM-10, графічного LCD-дисплея та настільного застосунку WPF.

У першому розділі проведено аналіз технічного завдання: визначено вимоги до системи моніторингу, розглянуто існуючі рішення та обґрунтовано вибір напряму реалізації.

У другому розділі описано проєктну частину: розроблено структурну схему системи, наведено опис електричної принципової схеми, а також побудовано блок-схеми алгоритмів роботи програми та пристрою.

У третьому розділі розкрито практичну реалізацію системи: описано створення застосунку, реалізацію захищеної передачі даних, розробку програми для STM32, роботу кнопок, а також результати тестування системи.

У четвертому розділі висвітлено питання безпеки життєдіяльності та охорони праці: розглянуто вимоги ергономіки до організації робочого місця і заходи захисту від ураження електричним струмом під час роботи з комп’ютером.

ANNOTATION

Balaiev V. A. Monitoring System for Hardware and Software Components of Computer Equipment: Bachelor's Graduation Thesis: speciality 123 — computer engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2026.

Keywords: monitoring system, PC components, STM32F769I-DISCO, Bluetooth Low Energy, HM-10, WPF, LibreHardwareMonitor, LVGL, AES-128, LCD display.

The qualification work is devoted to the development of a monitoring system for hardware components of a personal computer based on the STM32F769I-DISCO board using the HM-10 Bluetooth module, a graphical LCD display, and a WPF desktop application.

The first chapter analyzes the technical task: the requirements for the monitoring system are defined, existing solutions are reviewed, and the chosen implementation approach is justified.

The second chapter describes the design part: the structural diagram of the system is developed, the electrical schematic is described, and flowcharts of the application and device operation algorithms are presented.

The third chapter covers the practical implementation of the system: the development of the application, implementation of protected data transmission, development of the STM32 firmware, operation of the buttons, and the results of system testing are described.

The fourth chapter addresses life safety and occupational safety issues: ergonomic requirements for organizing the workplace and measures for protection against electric shock during work with a computer are considered.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

ARM – Advanced RISC Machine

BLE – Bluetooth Low Energy

CPU – Central Processing Unit

CTR – Counter Mode

GATT – Generic Attribute Profile

GPU – Graphics Processing Unit

LVGL – Light and Versatile Graphics Library

RAM – Random Access Memory

UART – Universal Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

WPF – Windows Presentation Foundation

ПК – Персональний комп'ютер

ВСТУП

Засоби моніторингу стану апаратно-технічних компонентів обчислювальних систем набувають особливого значення в умовах стрімкого розвитку інформаційних технологій та зростання обчислювальної потужності персональних комп’ютерів. Ефективне функціонування процесора, графічного адаптера, оперативної пам’яті, накопичувачів даних та інших вузлів безпосередньо впливає на продуктивність, стабільність і надійність роботи комп’ютерної системи. Підвищення навантаження на апаратні ресурси, робота в умовах недостатнього охолодження або виникнення несправностей можуть призводити до деградації обладнання, зниження швидкодії та виникнення аварійних ситуацій. У зв’язку з цим виникає потреба у створенні спеціалізованих систем, здатних здійснювати безперервний контроль технічного стану комп’ютера та забезпечувати відображення ключових параметрів його функціонування [1].

Для реалізації таких функцій доцільним є використання спеціалізованих вбудованих систем, побудованих на базі сучасних мікроконтролерних платформ. У даній роботі як апаратну основу обрано відлагоджувальну плату STM32F769I-DISCO, що містить високопродуктивний мікроконтролер сімейства STM32F7 з ядром ARM Cortex-M7, апаратними засобами прискорення графічних операцій, кольоровим дисплеєм та достатнім обсягом оперативної і постійної пам’яті для реалізації складних алгоритмів обробки та відображення інформації.

У роботі представлено розробку системи моніторингу апаратно-технічних компонентів персонального комп'ютера, призначеної для збору, передачі, обробки та візуалізації інформації про стан основних ресурсів комп’ютера у режимі реального часу.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ НА РОЗРОБКУ СИСТЕМИ МОНІТОРИНГУ КОМПОНЕНТІВ

1.1 Основні технічні вимоги до системи моніторингу апаратно-технічних компонентів персонального комп’ютера

Система моніторингу апаратно-технічних компонентів повинна забезпечувати безперервний збір, обробку, передачу та відображення інформації про поточний стан основних апаратних ресурсів обчислювальної системи в режимі реального часу [2]. Основним призначенням системи є оперативний контроль параметрів функціонування центрального процесора, графічного адаптера, оперативної пам’яті, накопичувачів даних та мережевих інтерфейсів з метою своєчасного виявлення критичних режимів роботи, перевантажень та потенційних апаратних несправностей.

До переліку контрольованих параметрів повинні входити завантаження центрального процесора та графічного адаптера, температурні показники апаратних компонентів, робочі частоти процесорних ядер і графічного процесора, обсяг використаної оперативної пам’яті, завантаження відеопам’яті, швидкість передавання даних через мережеві інтерфейси, а також інформація про стан та заповнення накопичувачів даних. Частота оновлення інформації повинна забезпечувати актуальність відображуваних даних та бути достатньою для оперативного реагування на зміни технічного стану комп’ютерної системи. Затримка між отриманням інформації на персональному комп’ютері та її відображенням на зовнішньому пристрої не повинна перевищувати декількох секунд.

Архітектура апаратної частини повинна базуватися на сучасній

мікроконтролерній платформі, здатній виконувати одночасно обробку вхідних даних, керування користувацьким інтерфейсом та взаємодію із засобами бездротового зв’язку. Обчислювальні ресурси мікроконтролера повинні забезпечувати стабільне виконання програмного забезпечення, обробку потоку телеметричних даних та формування графічного інтерфейсу користувача без помітних затримок і втрати інформації.

Для відображення інформації система повинна використовувати інтегрований кольоровий дисплей відлагоджувальної плати, який забезпечує достатню роздільну здатність для одночасного виведення числових значень, графічних індикаторів, діаграм та службових повідомлень. Інтерфейс користувача має бути інтуїтивно зрозумілим і забезпечувати швидкий доступ до різних груп параметрів системи. Навігація між інформаційними сторінками повинна здійснюватися за допомогою чотирьох апаратних кнопок, призначених для переходу між екранами, вибору режимів відображення, виклику додаткової інформації та виконання службових функцій керування.

Передавання інформації між персональним комп’ютером та пристроєм моніторингу повинно здійснюватися через бездротовий канал зв’язку Bluetooth. Для реалізації даної функції використовується модуль, який забезпечуватиме енергоефективний обмін даними та дозволяє розміщувати пристрій моніторингу на значній відстані від комп’ютера без необхідності використання додаткових кабельних з’єднань [3]. Канал зв’язку повинен забезпечувати стабільне передавання телеметричної інформації, контроль цілісності отриманих даних та можливість автоматичного відновлення з’єднання після тимчасового розриву зв’язку.

Програмне забезпечення системи повинно складатися з двох взаємопов’язаних компонентів: програмного модуля персонального комп’ютера та програмного забезпечення вбудованого пристрою. Комп’ютерний модуль має забезпечувати отримання інформації про стан апаратних компонентів за допомогою спеціалізованих засобів моніторингу та

системних інтерфейсів операційної системи, виконувати попередню обробку отриманих даних і передавати їх на мікроконтролерний пристрій через Bluetooth-канал [4]. Програмне забезпечення мікроконтролера повинно виконувати приймання, декодування, обробку та відображення отриманої інформації на екрані користувача.

Особливу увагу необхідно приділити надійності функціонування системи. Програмне забезпечення повинно передбачати механізми контролю коректності отриманих даних, обробку помилок передавання, індикацію втрати зв’язку та захист від аварійного завершення роботи. Система повинна характеризуватися низьким енергоспоживанням, компактними габаритними розмірами та можливістю автономного або стаціонарного живлення [5].

Таким чином, сукупність наведених вимог дозволяє сформувати технічну основу для створення сучасної системи моніторингу апаратно-технічних компонентів персонального комп’ютера, яка забезпечуватиме оперативне отримання та відображення інформації про стан комп’ютерної системи, підвищуючи ефективність контролю її роботи та своєчасність виявлення потенційних несправностей.

1.2 Аналіз існуючих рішень

Альтернативні програмні засоби моніторингу, зокрема утиліти та оверлеї, дають змогу отримувати значну кількість показників про роботу процесора, графічного адаптера, оперативної пам'яті, накопичувачів і мережевих інтерфейсів. Їх недоліком є залежність від основного графічного інтерфейсу операційної системи. Користувач повинен перемикатися між вікнами або постійно тримати оверлей поверх інших програм, що не завжди зручно. Крім того, у разі зависання графічного середовища або роботи у повноекранному режимі доступність таких засобів знижується. Тому винесення основних показників на окремий фізичний дисплей підвищує

зручність контролю та покращує сприйняття інформації [6].

Готові апаратні панелі моніторингу також можуть виконувати подібні функції, однак вони часто мають закритий формат обміну даними, обмежений набір показників або прив'язку до програмного забезпечення виробника. У межах даної роботи доцільніше використати відкриту структуру, у якій формат кадрів, алгоритм приймання, сторінки інтерфейсу та логіка керування повністю визначаються під час розробки. Такий підхід спрощує налагодження, дозволяє пояснити всі етапи роботи системи в пояснювальній записці та залишає можливість подальшого розширення переліку показників або зміни способу їх відображення [7].

Поширені системні утиліти на зразок MSI Afterburner, HWiNFO або AIDA64 добре підходять для діагностики, але не завжди зручні як постійний засіб спостереження. Їх інтерфейси орієнтовані переважно на відображення великої кількості параметрів у таблицях, списках або окремих вікнах. Для швидкого контролю стану ПК такий підхід є надлишковим, оскільки користувачу потрібно самостійно шукати потрібні рядки серед багатьох сенсорів.

Іншим недоліком програмних оверлеїв є їх вплив на основний робочий простір користувача. Під час гри, роботи з графічними програмами, перегляду відео або виконання повноекранних задач оверлей займає частину екрана та може перекривати важливі елементи інтерфейсу. Якщо його вимкнути, контроль температур, завантаження процесора чи відеокарти знову потребує перемикання між вікнами. Окремий зовнішній дисплей вирішує цю проблему, оскільки вся службова інформація виводиться за межі основного екрана і не заважає виконанню основної роботи.

Типові апаратні рішення часто розраховані на конкретну екосистему виробника або певний набір програмного забезпечення. Через це користувач не завжди може змінити структуру сторінок, додати власні показники, змінити формат даних або реалізувати додаткові режими роботи. У розроблюваному

пристрої логіка роботи повністю відкрита: застосунок сам формує кадри параметрів, прошивка STM32 самостійно їх приймає та обробляє, а інтерфейс дисплея створюється відповідно до потреб проєкту. Це робить систему придатною не лише для використання, а й для подальшого навчального та інженерного вдосконалення.

Окрему проблему становить захист і контроль каналу передавання даних. У багатьох простих DIY-рішеннях дані передаються у вигляді звичайного тексту через послідовний порт або Bluetooth без перевірки цілісності. Такий підхід спрощує початкову реалізацію, але ускладнює стабільну роботу при втраті байтів, перешкодах або випадковому прийманні неповних повідомлень. У запропонованій системі використовується власний формат кадру з сигнатурою, заголовком, корисним навантаженням і CRC, а також шифрування AES-128. Це підвищує надійність обміну та дає змогу відкидати пошкоджені або некоректні пакети.

1.3 Обгрунтування вибраного напряму реалізації системи моніторингу компонентів персонального комп’ютера

Використання вбудованих систем для моніторингу апаратно-технічних компонентів персонального комп’ютера є одним із перспективних напрямків розвитку засобів технічної діагностики та контролю стану обчислювальних систем. Постійне зростання продуктивності сучасних процесорів, графічних адаптерів та інших компонентів супроводжується збільшенням енергоспоживання, тепловиділення та складності архітектури комп’ютерного обладнання. За таких умов своєчасне отримання достовірної інформації про стан апаратних ресурсів набуває важливого значення як для підтримання стабільності роботи системи, так і для запобігання виникненню критичних режимів експлуатації.

Область застосування систем моніторингу апаратних компонентів

охоплює широкий спектр обчислювальних платформ, починаючи від домашніх персональних комп’ютерів і закінчуючи професійними робочими станціями, серверними комплексами та спеціалізованими обчислювальними системами. Незалежно від сфери використання, основною вимогою до таких систем є можливість безперервного отримання актуальної інформації про технічний стан обладнання в режимі реального часу [8]. Контроль параметрів дозволяє оперативно виявляти перевищення допустимих температурних режимів, надмірне навантаження на окремі вузли, деградацію накопичувачів даних, нестабільність живлення та інші фактори, що можуть негативно впливати на функціонування комп’ютерної системи.

На початковому етапі проєктування необхідно визначити перелік параметрів, які підлягають моніторингу. До найбільш важливих характеристик належать завантаження центрального процесора, температура процесорних ядер, робочі частоти та енергоспоживання процесора, завантаження графічного адаптера, температура графічного процесора, використання відеопам’яті, обсяг зайнятої оперативної пам’яті, швидкість роботи накопичувачів даних та мережевих інтерфейсів. Додатково можуть контролюватися параметри материнської плати, швидкість обертання вентиляторів системи охолодження, рівень напруги окремих компонентів та інші службові характеристики. Враховуючи значну кількість потенційних джерел інформації, система повинна забезпечувати ефективну обробку та структурування отриманих даних для їх подальшого відображення користувачу.

Однією з важливих особливостей області застосування є необхідність мінімізації впливу процесу моніторингу на продуктивність основної комп’ютерної системи. Традиційні програмні засоби моніторингу працюють безпосередньо в операційній системі та використовують частину обчислювальних ресурсів комп’ютера. При значному навантаженні або виконанні ресурсомістких завдань це може призводити до зменшення

продуктивності або створення додаткового інформаційного навантаження на користувача. Саме тому актуальним є використання зовнішніх вбудованих пристроїв, які отримують інформацію через окремий канал зв’язку та виконують її відображення незалежно від програмного середовища комп’ютера.

Особливу увагу необхідно приділити архітектурі обміну даними між персональним комп’ютером та вбудованим пристроєм. У сучасних рішеннях широко використовуються дротові та бездротові канали зв’язку, серед яких найбільш поширеними є USB, Wi-Fi та Bluetooth. Для задач моніторингу перевага часто надається бездротовим технологіям, оскільки вони забезпечують гнучкість розміщення пристрою та спрощують його інтеграцію в існуючу інфраструктуру. Водночас система повинна забезпечувати достатню швидкість передавання даних, низький рівень затримок та стійкість до тимчасових втрат з’єднання.

Важливим фактором є також організація інтерфейсу користувача. Оскільки кількість контрольованих параметрів може бути досить великою, система повинна забезпечувати логічне групування інформації за категоріями. Зазвичай окремо відображаються показники центрального процесора, графічного адаптера, оперативної пам’яті, накопичувачів даних та мережевої активності. Для покращення сприйняття інформації доцільним є використання графічних індикаторів, діаграм, гістограм та кольорового маркування критичних станів. Інтерфейс повинен залишатися зрозумілим навіть при тривалому використанні та забезпечувати швидкий доступ до найбільш важливих параметрів системи.

З точки зору експлуатації, система повинна підтримувати тривалу безперервну роботу без необхідності регулярного втручання користувача. Для цього необхідно передбачити механізми автоматичного відновлення зв’язку після втрати з’єднання, обробку помилок передавання даних, контроль коректності отриманих пакетів та індикацію аварійних режимів. У випадку

використання автономного живлення додатково виникають вимоги щодо мінімізації енергоспоживання та підтримки режимів енергозбереження.

Не менш важливою є можливість масштабування функціональних можливостей системи. Архітектура програмного забезпечення повинна дозволяти додавання нових типів параметрів, модифікацію алгоритмів відображення інформації та інтеграцію з іншими програмними засобами моніторингу. Використання відкритих програмних інтерфейсів та модульного підходу до розробки забезпечує спрощення подальшої модернізації та адаптації системи до нових вимог.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ АПАРАТНО-ТЕХНІЧНИХ КОМПОНЕНТІВ КОМП’ЮТЕРА

2.1 Розробка структурної схеми системи моніторингу апаратно-технічних компонентів ПК

Проєктування системи моніторингу апаратно-технічних компонентів персонального комп'ютера розпочинається з визначення її загальної структури та взаємодії між окремими функціональними вузлами. Система повинна забезпечувати отримання параметрів комп'ютера, передавання їх бездротовим каналом, приймання на мікроконтролерному пристрої, перевірку коректності даних і відображення результатів на екрані. Така послідовність роботи визначає необхідність поділу системи на програмну частину персонального комп'ютера, канал передавання даних і апаратний пристрій відображення.

Структурна схема системи моніторингу наведена на рисунку 2.1. У її верхній частині зображено персональний комп'ютер, який виступає джерелом телеметричних даних. На ньому працює WPF-застосунок, що отримує інформацію про стан апаратних компонентів, формує пакет даних і передає його через Bluetooth Low Energy. Нижня частина схеми відповідає зовнішньому пристрою моніторингу, до складу якого входять Bluetooth-модуль, плата STM32F769I-DISCO з вбудованим дисплеєм та блок фізичних кнопок керування.

Запропонована структурна організація забезпечує логічне розмежування функцій між окремими компонентами системи, що спрощує її розробку, налагодження та подальшу модернізацію. Такий підхід забезпечує незалежність програмної та апаратної складових системи, підвищує її масштабованість і дозволяє без суттєвих змін архітектури розширювати

перелік контрольованих параметрів або інтегрувати додаткові функціональні модулі в процесі подальшого розвитку системи.

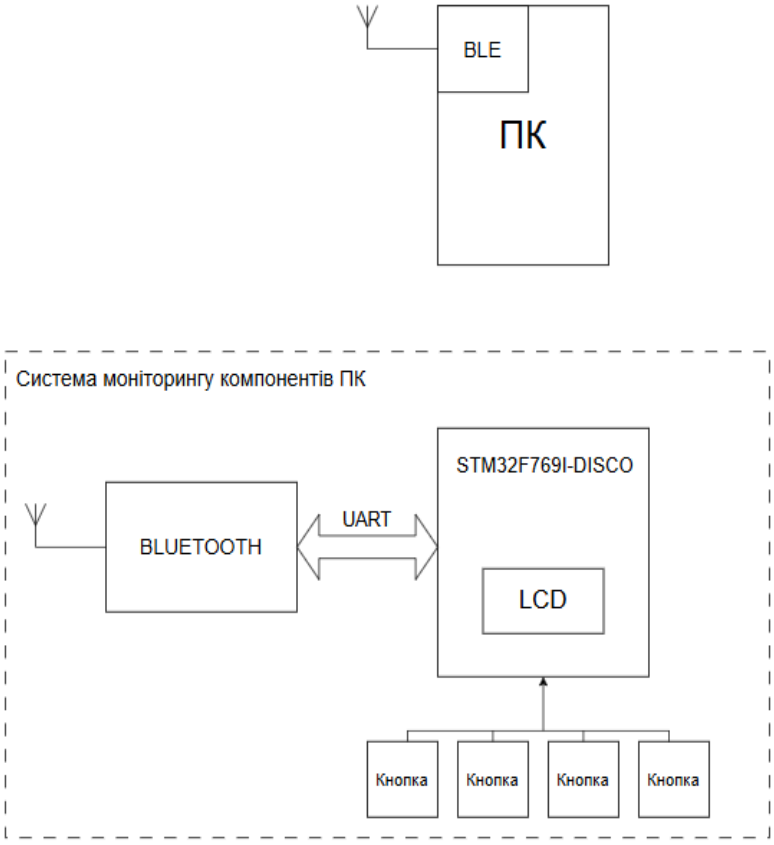

Рисунок 2.1 – Структурна схема системи моніторингу апаратно-технічних компонентів ПК

Основним джерелом інформації у системі є персональний комп'ютер. Саме на стороні операційної системи доступні засоби отримання показників процесора, графічного адаптера, оперативної пам'яті, накопичувачів і мережевих інтерфейсів. Для цього використовується настільний застосунок, який об'єднує дані, отримані з бібліотеки LibreHardwareMonitor та системних інтерфейсів Windows. Програмний модуль виконує попередню обробку показників, приводить їх до єдиного формату, формує структуру телеметрії та передає її на пристрій у вигляді компактного двійкового кадру.

Канал передавання даних реалізований за допомогою Bluetooth-модуля HM-10. З боку персонального комп'ютера модуль сприймається як BLE-пристрій, до якого застосунок підключається після сканування доступних пристроїв. З боку мікроконтролера HM-10 працює як послідовний UART-модуль, що передає отримані байти на вхід STM32. Така структура дозволяє розділити складність роботи з Bluetooth і логіку приймання даних: комп'ютер виконує BLE-з'єднання, а STM32 обробляє звичайний послідовний потік.

Центральним вузлом пристрою є плата STM32F769I-DISCO. Вона виконує приймання байтів через UART, відновлення кадрів протоколу, перевірку контрольної суми, розшифрування корисного навантаження та оновлення графічного інтерфейсу. До складу плати входить кольоровий дисплей, який використовується для виведення семи інформаційних сторінок. Вибір цієї плати є доцільним, оскільки вона має достатню продуктивність для роботи з графічною бібліотекою LVGL та апаратні периферійні блоки для роботи з дисплеєм.

Блок кнопок використовується для локального керування пристроєм без необхідності взаємодії з застосунком. Перша кнопка відповідає за перехід на попередню сторінку, друга – за перехід на наступну сторінку, третя – за швидке повернення на головну сторінку, четверта – за вмикання або вимикання режиму сну дисплея. Така схема керування зручна для пристрою, який розташований поряд із робочим місцем користувача і повинен дозволяти швидке перемикання між групами показників.

Структурна схема також відображає логіку розподілу задач між вузлами. Настільний застосунок не займається графічним відображенням на STM32-дисплеї, а лише формує дані у погодженому форматі. Мікроконтролер не виконує складного опитування апаратних сенсорів ПК, оскільки не має прямого доступу до них. Натомість він зосереджується на прийманні, контролі, декодуванні та візуалізації. Такий розподіл зменшує складність кожного окремого компонента і забезпечує можливість незалежного

налагодження програмної та апаратної частин.

Важливою особливістю структури є наявність власного протоколу обміну. Він потрібний для того, щоб потік байтів від Bluetooth-модуля можна було однозначно поділити на окремі повідомлення. Кожен кадр містить службову сигнатуру, версію протоколу, тип пакета, номер послідовності, довжину корисного навантаження, зашифровані дані та CRC-16. Завдяки цьому мікроконтролер може перевірити цілісність повідомлення, відкинути пошкоджений кадр і продовжити роботу без перезапуску системи.

У структурі системи передбачено два основні напрями передавання інформації. Перший напрям пов'язаний з регулярною передачею параметрів, коли WPF застосунок періодично надсилає поточний стан компонентів ПК. Другий напрям використовується для службових команд і налаштувань, зокрема передавання значення яскравості дисплея, таймера переходу в режим сну, ping-кадру для підтвердження з'єднання та команд керування живленням екрана. Наявність різних типів пакетів дозволяє не змішувати дані моніторингу з командами керування.

Побудована структурна схема є основою для подальшого проєктування електричних з'єднань, алгоритмів роботи програмного забезпечення та способу організації графічного інтерфейсу. Вона показує, що система є апаратно-програмним комплексом, у якому функції збору, передавання, обробки і відображення даних виконуються різними, але взаємопов'язаними компонентами.

2.2 Побудова та опис схеми електричної принципової

Після визначення загальної структури системи необхідно описати електричні з’єднання між основними компонентами. У розробленій системі апаратна частина побудована на базі відлагоджувальної плати STM32F769I-DISCO, Bluetooth-модуля HM-10, вбудованого LCD-дисплея та чотирьох

фізичних кнопок керування.

Основним керуючим елементом системи є мікроконтролер STM32F769NIH6 (рис. 2.2), встановлений на платі STM32F769I-DISCO. Він виконує приймання даних від Bluetooth-модуля, перевірку пакетів, розшифрування телеметрії, обробку натискань кнопок та формування графічного інтерфейсу для LCD-дисплея [9]. Використання процесорного ядра ARM Cortex-M7 забезпечує високу обчислювальну продуктивність, підтримку операцій із плаваючою комою, апаратне прискорення обробки графіки та достатній обсяг периферійних ресурсів для одночасної роботи дисплея, Bluetooth-модуля і цифрових входів кнопок керування [10].

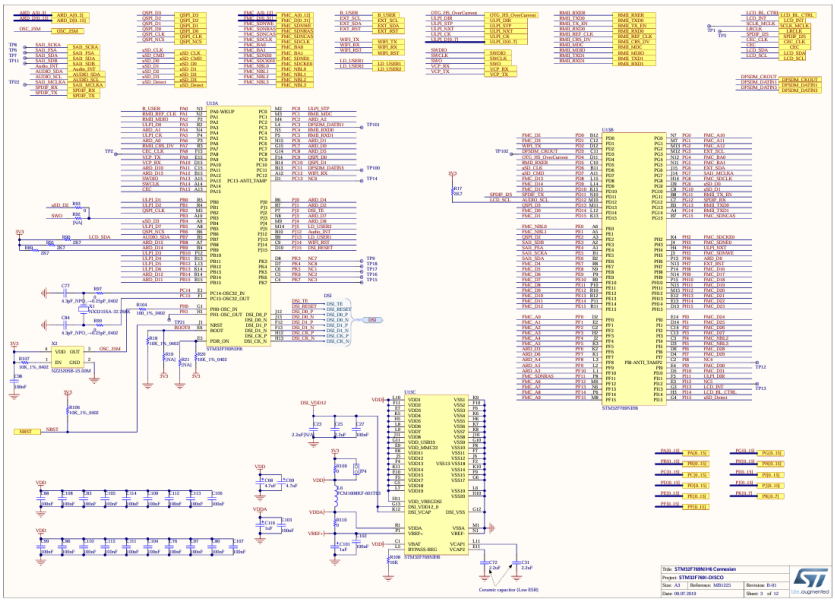

Рисунок 2.2 – Схема електрична принципова мікроконтролера STM32F769NIH6

Мікроконтролер працює з живленням 3,3 В, тому зовнішні сигнальні лінії системи також узгоджуються з цим рівнем. Це важливо для підключення Bluetooth-модуля HM-10, оскільки його UART-інтерфейс також використовує логічні рівні 3,3 В. Використання спільної лінії GND для всіх вузлів забезпечує однакову точку відліку логічних сигналів і коректну роботу обміну даними.

У розробленій системі задіяно GPIO-виводи мікроконтролера для

підключення фізичних кнопок керування. Кнопка на PA4 використовується для переходу на попередню сторінку, кнопка на PC2 – для переходу на наступну сторінку, кнопка на PF10 – для повернення на головну сторінку, а кнопка на PF8 – для керування режимом сну дисплея. Кожна кнопка працює за схемою з підтягуванням входу до логічної одиниці. У нормальному стані на вході присутній високий рівень, а при натисканні кнопка замикає відповідну лінію на землю.

LCD-дисплей (рис. 2.3) є вбудованою частиною відлагоджувальної плати STM32F769I-DISCO. У системі він використовується для відображення поточних параметрів комп'ютера: завантаження процесора та відеокарти, температур, стану оперативної пам'яті, накопичувачів і мережевого трафіку. Дисплей має роздільну здатність 800 x 472 пікселі, що дозволяє формувати повноколірний графічний інтерфейс із кільцевими індикаторами, прогрес-барами та графіками.

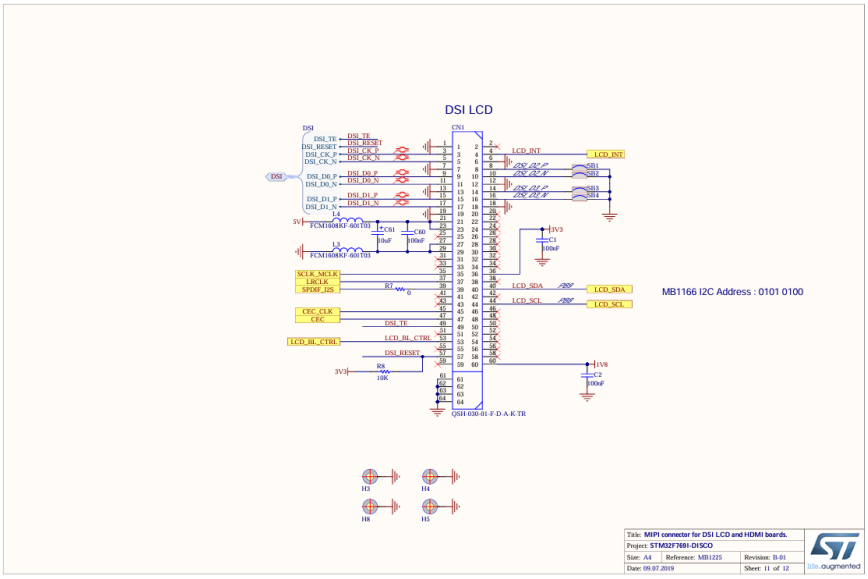

Рисунок 2.3 - Схема електрична принципова дисплейного вузла

Для роботи з дисплеєм у мікроконтролері використовуються апаратні блоки LTDC, DSI та DMA2D. LTDC відповідає за формування зображення, DSI забезпечує передавання даних до дисплейного модуля, а DMA2D

прискорює графічні операції. Зовнішня SDRAM використовується як пам’ять для графічного буфера. Така апаратна конфігурація дає можливість реалізувати плавний інтерфейс без надмірного навантаження на центральне ядро мікроконтролера.

Bluetooth-модуль HM-10 (рис. 2.4) використовується як зовнішній бездротовий інтерфейс системи [11]. Він забезпечує приймання даних від застосунку, що працює на ПК, і передає їх до мікроконтролера через UART. Після встановлення BLE-з’єднання модуль працює як прозорий послідовний канал, тому всі отримані байти передаються на STM32 без додаткової складної обробки на стороні модуля.

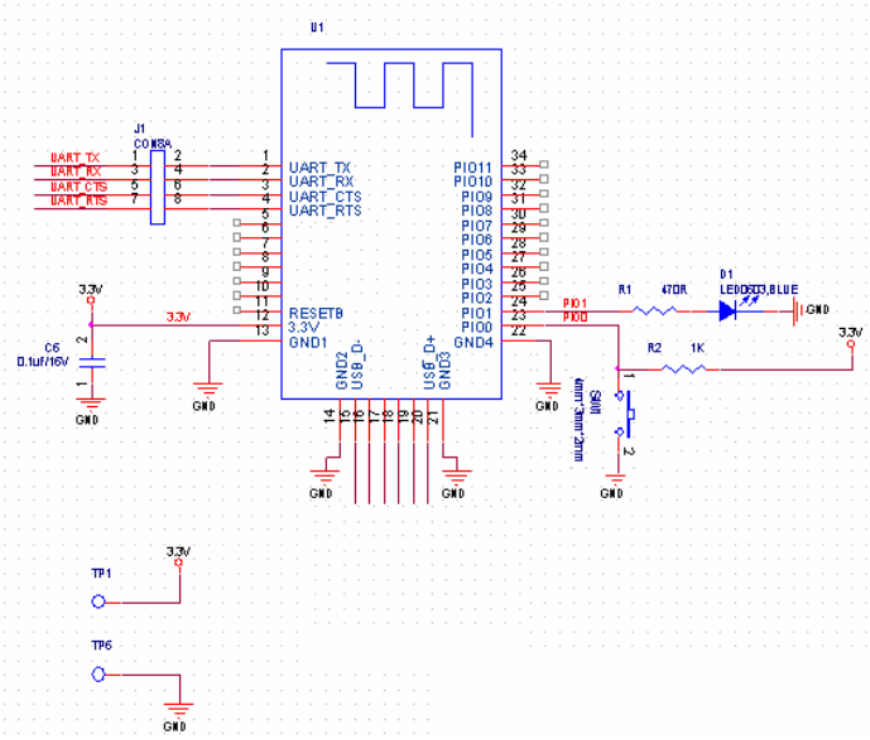

Рисунок 2.4 – Схема електрична принципова Bluetooth-модуля HM-10

Для коректної роботи HM-10 підключається до лінії живлення 3,3 В та спільного проводу GND. Використання саме 3,3 В є важливим, оскільки підключення модуля до 5 В без узгодження рівнів може призвести до його пошкодження або нестабільної роботи. Спільний GND між HM-10 і STM32 є

обов’язковим для правильного визначення логічних рівнів UART.

Лінії UART підключаються перехресно. Вихід передавання TXD модуля HM-10 з’єднується з входом приймання UART мікроконтролера, а вхід RXD модуля – з виходом передавання UART мікроконтролера. У розробленому пристрої для цього використано контакти D0 та D1 плати STM32F769I-DISCO, які в програмній частині налаштовані для обміну з Bluetooth-модулем.

Використання UART для зв’язку з HM-10 є доцільним, оскільки цей інтерфейс апаратно підтримується мікроконтролером STM32, легко налагоджується та добре підходить для передавання потоку телеметричних даних [12].

2.3 Розробка блок-схеми алгоритму програми

Програмна частина на персональному комп'ютері забезпечує збір даних про стан апаратних компонентів, підключення до Bluetooth-модуля, формування пакетів і передавання їх на пристрій. Алгоритм роботи WPF застосунку побудований таким чином, щоб користувач міг виконати основні дії послідовно: запустити програму, знайти HM-10, встановити з'єднання, запустити потокове передавання та за потреби змінити налаштування дисплея пристрою.

На блок-схемі відображено повну послідовність виконання основних функцій настільного застосунку – від моменту його запуску до початку безперервного передавання телеметричних даних.

Окремими етапами показано ініціалізацію програмних модулів, пошук доступних Bluetooth-пристроїв, встановлення з'єднання з модулем HM-10, формування телеметричного пакета, його шифрування та передавання на мікроконтролерний пристрій. Крім того, алгоритм передбачає можливість зміни параметрів відображення інформації на дисплеї, а також контроль стану Bluetooth-з'єднання, що забезпечує стабільну роботу системи та своєчасне

реагування на можливі помилки передавання даних. Блок-схема алгоритму програми наведена на рис. 2.5.

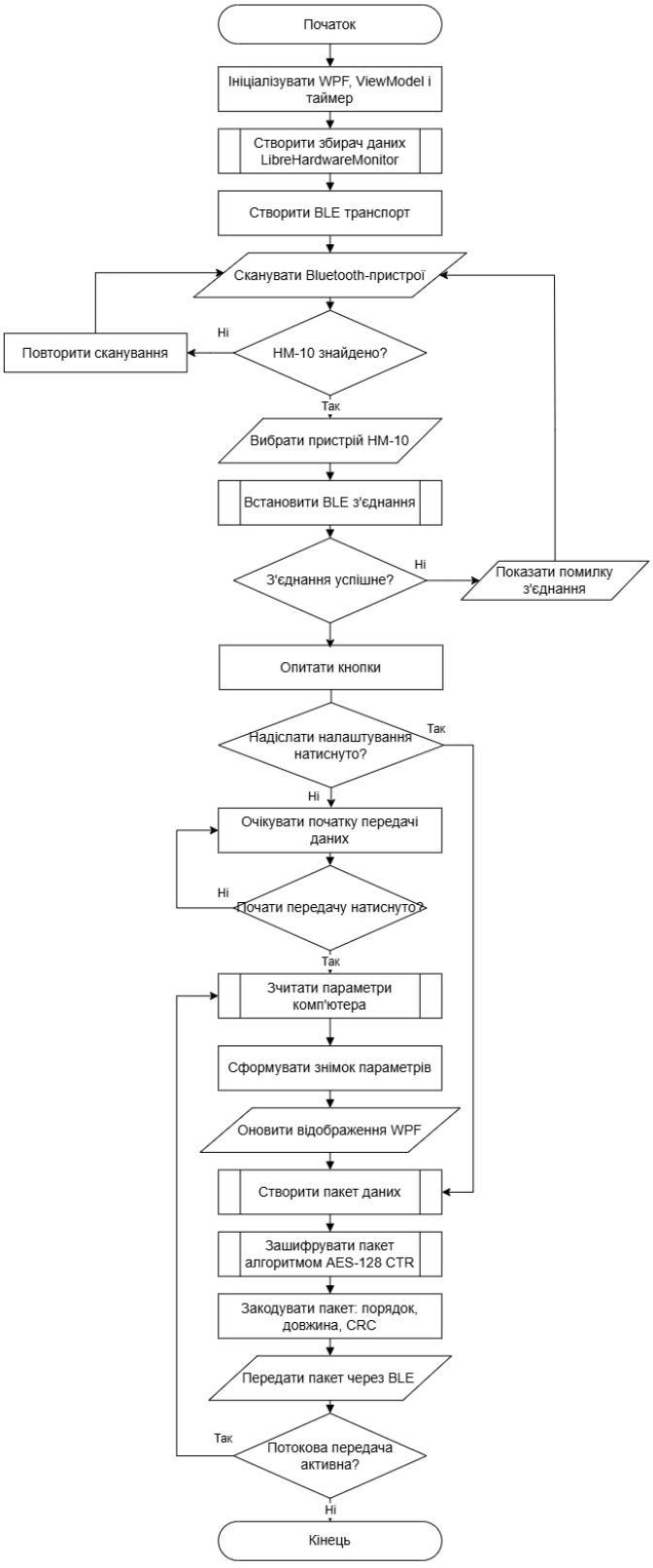

Рисунок 2.5 – Блок-схема алгоритму роботи WPF застосунку

Після запуску застосунку виконується ініціалізація графічного інтерфейсу WPF, основної моделі представлення, команд керування та таймера оновлення. На цьому етапі створюються об'єкти, які відповідають за збір даних, формування кадрів протоколу та передавання пакетів. Структура програми побудована за принципом розділення відповідальностей: модуль збору метрик не залежить від модуля Bluetooth-передавання, а модуль формування протоколу може використовуватись як для реального пристрою, так і для симулятора.

Наступним кроком створюється збирач даних LibreHardwareMonitor. Він виконує опитування сенсорів центрального процесора, графічного адаптера, накопичувачів та інших компонентів. Для показників, які недоступні через апаратні сенсори, використовуються додаткові системні механізми Windows. Наприклад, обсяг фізичної пам'яті визначається через системний API, а активність накопичувачів і мережевих інтерфейсів може отримуватись через лічильники продуктивності. Це дає змогу сформувати більш повний знімок стану ПК.

Після ініціалізації джерел даних створюється транспортний модуль Bluetooth. Користувач запускає сканування пристроїв, після чого програма формує список доступних BLE-пристроїв. Якщо HM-10 не знайдено, користувач може повторити сканування. Якщо пристрій знайдено, він вибирається у списку, після чого програма встановлює BLE GATT-з'єднання. У разі помилки з'єднання користувач отримує повідомлення про помилку і може повторити підключення.

Після успішного підключення програма надсилає службовий ping-кадр. Він не містить телеметрії, але дає змогу пристрою визначити, що Bluetooth-зв'язок уже встановлено. До початку потокового передавання STM32 може показувати на дисплеї статус підключення і підказку щодо запуску потокової передачі. Така проміжна індикація спрощує користування системою, оскільки користувач бачить різницю між повною відсутністю зв'язку і станом, коли

модуль підключений, але дані ще не надходять.

Основний цикл роботи програми починається після натискання кнопки Start streaming. У цьому режимі таймер періодично запускає зчитування параметрів комп'ютера. Отримані значення об'єднуються в єдину структуру. Вона містить показники CPU, GPU, RAM, мережі, накопичувачів і температурних сенсорів. На основі цього знімка одночасно оновлюється головна сторінка WPF додатку і формується пакет для передачі на пристрій.

Передавання даних на STM32 виконується не у вигляді текстового повідомлення, а у вигляді двійкового кадру. Спочатку формується пакет параметрів із фіксованою версією формату. Показники квантуються у цілі числа: наприклад, проценти та температури передаються з множником 10, а обсяги пам'яті – у зручних одиницях, придатних для компактного кодування. Такий підхід зменшує обсяг переданих даних і спрощує декодування на мікроконтролері.

Після формування пакету виконується його шифрування алгоритмом AES-128 у режимі CTR. Далі програма формує кадр протоколу: додає версію, тип пакета, номер послідовності, довжину даних та CRC-16. Контрольна сума обчислюється по службових полях і зашифрованому пакету, що дозволяє STM32 перевірити цілісність повідомлення до розшифрування. Готовий кадр передається через BLE до модуля HM-10.

Алгоритм передбачає повторення циклу доти, доки потокове передавання залишається активним. Якщо користувач зупиняє потокову передачу, програма припиняє періодичне надсилання параметрів, але може залишатися підключеною до пристрою. Окрема гілка алгоритму відповідає за передавання налаштувань. Коли користувач змінює яскравість або таймер сну, застосунок формує пакет налаштувань, шифрує його, пакує у кадр протоколу та передає на STM32 аналогічно до телеметричних пакетів.

Застосунок також підтримує симуляційний режим, який використовується для перевірки інтерфейсу і формування пакетів без

фізичного підключення до мікроконтролера. Це важливо на етапі проєктування, оскільки дозволяє окремо налагоджувати частину програмного забезпечення та структуру даних. Після перевірки симулятора той самий код формування кадрів використовується для реального BLE-транспорту, що зменшує ймовірність розбіжностей між тестовим і робочим режимами.

Такий алгоритм застосунку забезпечує поетапний і контрольований процес взаємодії з пристроєм. Користувач бачить стан підключення, може запускати та зупиняти передавання, а програмні модулі залишаються достатньо незалежними для подальшого розвитку.

2.4 Розробка блок-схеми алгоритму пристрою

Прошивка мікроконтролерного пристрою відповідає за запуск апаратної платформи, ініціалізацію периферійних модулів, приймання даних від Bluetooth-модуля HM-10, розбір кадрів протоколу, обробку команд, оновлення графічного інтерфейсу дисплея та реагування на натискання фізичних кнопок керування. Крім виконання основних функцій, програмне забезпечення здійснює контроль стану Bluetooth-з'єднання, перевірку коректності отриманих телеметричних пакетів і обробку можливих помилок передавання даних. Такий підхід забезпечує безперервне функціонування системи та підтримує актуальність інформації, що відображається користувачу. Алгоритм пристрою повинен бути стійким до втрати окремих байтів, помилок контрольної суми, відсутності з'єднання та нерівномірного надходження пакетів.

Блок-схема відображає послідовність виконання основних етапів роботи програмного забезпечення мікроконтролерного пристрою після його ввімкнення. На ній показано ініціалізацію периферійних модулів, запуск графічного інтерфейсу, встановлення зв'язку з Bluetooth-модулем, приймання та аналіз телеметричних пакетів, перевірку їх цілісності й коректності, а також

оновлення інформації на дисплеї. Окрему увагу приділено обробці натискань фізичних кнопок, що забезпечують навігацію між сторінками інтерфейсу та керування режимами роботи пристрою. Блок-схема алгоритму роботи пристрою наведена на рисунку 2.6.

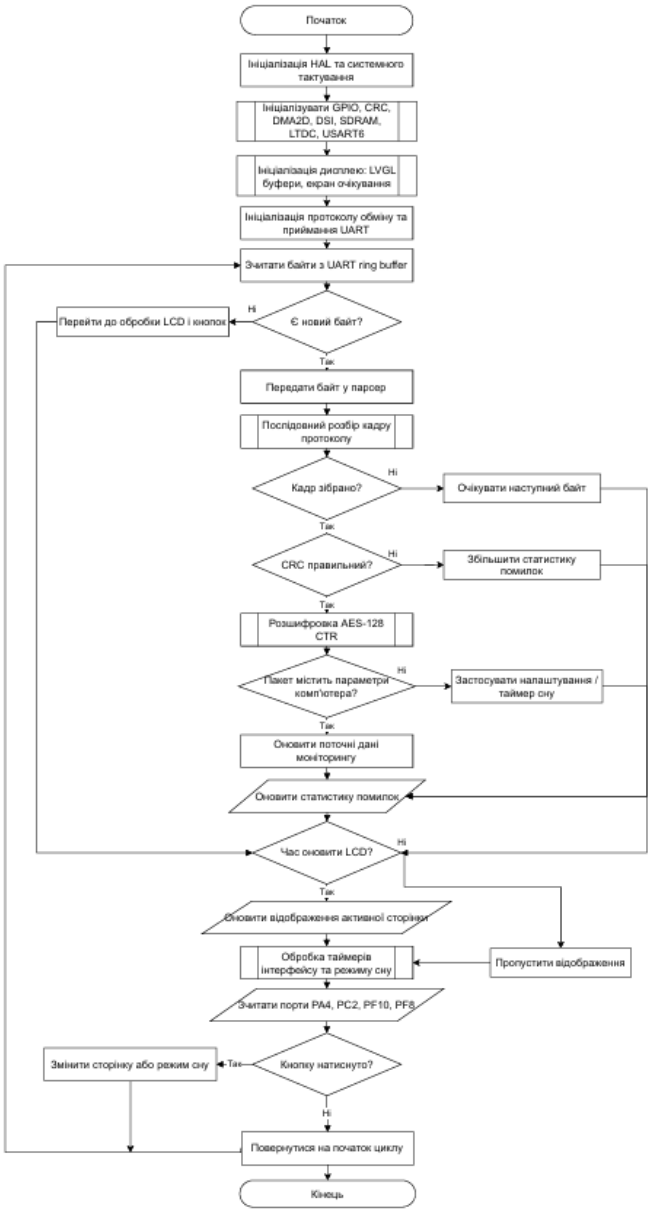

Рисунок 2.6 – Блок-схема алгоритму роботи мікроконтролерного пристрою

Після скидання мікроконтролера виконується початкова ініціалізація HAL та налаштування системного тактування. Далі ініціалізуються

периферійні блоки, необхідні для роботи пристрою: GPIO, CRC, DMA2D, DSI, SDRAM, LTDC і USART. GPIO використовується для кнопок та службових сигналів, USART – для обміну з HM-10, а LTDC, DSI, DMA2D і SDRAM забезпечують роботу графічного дисплея.

Після ініціалізації периферії виконується запуск графічної підсистеми. Модуль MainLcd налаштовує дисплей, графічні буфери LVGL, framebuffer і початковий екран очікування. До отримання першого валідного кадру пристрій виводить повідомлення про очікування Bluetooth-з'єднання. Якщо отримано службовий ping-кадр, але потік параметрів ще не запущено, на дисплеї відображається стан Bluetooth connected і підказка щодо запуску передавання з застосунку.

Для приймання даних від Bluetooth-модуля запускається UART-прийом у режимі переривань. Приймання не виконує складної обробки, а тільки записує отриманий байт у кільцевий буфер. Такий підхід зменшує час перебування у перериванні та дозволяє не втрачати байти навіть тоді, коли основний цикл зайнятий оновленням графічного інтерфейсу. Обробка накопичених байтів виконується вже у головному циклі.

У головному циклі прошивка послідовно перевіряє наявність байтів у кільцевому буфері. Якщо нових байтів немає, пристрій переходить до обробки дисплея і кнопок. Якщо байт доступний, він передається у парсер протоколу. Парсер працює як послідовний автомат станів: спочатку шукає сигнатуру початку кадру, далі читає заголовок, довжину пакету, сам пакет і контрольну суму CRC. Якщо повний кадр ще не зібрано, пристрій очікує наступний байт.

Після складання повного кадру виконується перевірка CRC-16. Якщо контрольна сума неправильна, кадр відкидається, збільшується статистика помилок, а парсер повертається до пошуку наступного повідомлення. Якщо CRC правильний, зашифроване корисне навантаження розшифровується алгоритмом AES-128 CTR [13]. Лише після цього прошивка аналізує тип пакета і передає пакет відповідному обробнику.

Для пакета параметрів виконується декодування структури поточних даних моніторингу. Отримані значення зберігаються в оперативній пам'яті як актуальний знімок. Він використовується всіма сторінками інтерфейсу для відображення числових значень, індикаторів і графіків історії. Для пакета налаштувань застосовуються параметри яскравості дисплея і таймера сну, а командні пакети можуть переводити дисплей у режим сну або пробуджувати його.

Оновлення дисплея виконується з обмеженою частотою, щоб уникнути мерехтіння і надмірного навантаження на мікроконтролер. Під час оновлення активна графічна сторінка отримує актуальні значення зі знімку параметрів і змінює відповідні елементи інтерфейсу. Для кільцевих індикаторів оновлюються значення та колір дуги, для графічних шкал – рівень заповнення, для історичних графіків – масив останніх значень. Якщо дисплей перебуває у режимі сну, відмальовування пропускається.

Окремо обробляються фізичні кнопки. Для підвищення чутливості їхній стан швидко сканується в періодичному системному таймері, а в головний цикл передаються вже зафіксовані події натискання. Це дозволяє не втрачати короткі натискання навіть тоді, коли основний цикл зайнятий прийманням пакетів або оновленням сторінки [14].

Перша кнопка перемикає активну сторінку назад, друга – вперед, третя повертає користувача на головну сторінку, четверта перемикає стан сну дисплея. Після обробки кнопок пристрій повертається на початок головного циклу. Такий алгоритм забезпечує одночасне виконання трьох основних задач: приймання Bluetooth-даних, підтримання графічного інтерфейсу та реагування на локальне керування.

Розроблений алгоритм пристрою є циклічним і не містить блокувальних очікувань, які могли б надовго зупиняти приймання даних або оновлення дисплея. Усі довготривалі процеси розбиті на короткі операції: байти накопичуються в буфері, кадри збираються поступово, дисплей оновлюється

за таймером, а кнопки фіксуються окремим швидким скануванням. Це підвищує стабільність роботи системи та дозволяє підтримувати актуальне відображення параметрів ПК у режимі реального часу.

Окрема частина алгоритму пов'язана з підтриманням актуального стану графічного інтерфейсу. Після приймання нового знімку параметрів пристрій не перемальовує весь екран безперервно, а оновлює лише активну сторінку з фіксованим інтервалом. Такий підхід зменшує мерехтіння дисплея, скорочує кількість операцій копіювання у framebuffer і залишає достатньо часу для обробки UART-даних. Значення, які використовуються для графіків історії, накопичуються у невеликих циклічних масивах, тому для їх зберігання не потрібні значні обсяги оперативної пам'яті.

Для підвищення зручності користування прошивка розділяє декілька станів взаємодії з комп'ютером. Якщо від застосунку ще не отримано жодного коректного кадру, на дисплеї відображається повідомлення про очікування Bluetooth-з'єднання. Після отримання службового ping-кадру пристрій переходить у стан підтвердженого підключення, але продовжує очікувати запуску потокового передавання. Після надходження першого пакета налаштувань система автоматично переходить до головної сторінки. Завдяки цьому користувач може легко визначити, чи проблема пов'язана з відсутністю Bluetooth-з'єднання, чи лише з тим, що передавання параметрів ще не запущено.

Алгоритм також передбачає обробку налаштувань, які надходять із застосунку. Пакет налаштувань містить значення яскравості дисплея та таймаут переходу в режим сну. Після приймання такого пакета прошивка застосовує нові параметри без перезапуску пристрою. Якщо таймер сну активний і користувач протягом заданого часу не взаємодіє з пристроєм, підсвітка дисплея вимикається для зменшення енергоспоживання. Будь-яка дія користувача або команда пробудження повертає екран у робочий стан.

Значну увагу приділено обробці помилок. Якщо кадр має неправильну

версію протоколу, завелику довжину пакету або некоректну CRC-16, він не передається у модуль декодування телеметрії. Прошивка збільшує відповідні лічильники помилок і продовжує очікувати наступний кадр. Такий підхід запобігає використанню пошкоджених даних для оновлення інтерфейсу. Для швидкої діагностики передбачено службову світлодіодну індикацію: один індикатор коротко активується після приймання валідного пакета, інший – після помилки протоколу.

Загальна логіка пристрою побудована так, щоб жоден етап не блокував виконання інших задач на тривалий час. UART-приймання винесене в переривання, розбір кадрів виконується поступово, відображення оновлюється з обмеженою частотою, а кнопки скануються окремо від важких графічних операцій. Така організація є важливою для мікроконтролерної системи, оскільки дозволяє одночасно підтримувати стабільний зв'язок, чутливе керування і плавне оновлення екранного інтерфейсу.

РОЗДІЛ 3 ПРОГРАМУВАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ МОНІТОРИНГУ АПАРАТНО-ТЕХНІЧНИХ КОМПОНЕНТІВ ПЕРСОНАЛЬНОГО КОМП’ЮТЕРА

3.1 Реалізація програмної частини системи моніторингу компонентів ПК

Практична реалізація системи моніторингу компонентів ПК розпочинається з розроблення програмної частини, яка виконується на персональному комп'ютері. Цей компонент відповідає за отримання поточних показників апаратних вузлів, підготовку даних до передавання, формування кадрів власного протоколу та взаємодію з Bluetooth-модулем HM-10. Для реалізації настільного застосунку використано платформу .NET 8 і технологію WPF, що дозволяє створити графічний інтерфейс для Windows та одночасно застосовувати сучасні засоби роботи з асинхронними операціями, Bluetooth-пристроями і системними даними [15].

Структура програмної частини побудована за принципом розділення відповідальностей. Модуль збору даних відповідає лише за отримання показників ПК, модуль протоколу – за формування кадрів, модуль транспорту – за передавання через BLE, а модель представлення керує станом інтерфейсу користувача. Такий підхід спрощує налагодження і дозволяє перевіряти окремі частини програми незалежно одна від одної. Наприклад, симуляційний транспорт може використовуватись без підключення реального пристрою, а код формування кадрів залишається однаковим для симулятора і для HM-10.

Для отримання апаратних показників використовується бібліотека LibreHardwareMonitor. Вона надає доступ до сенсорів процесора, графічного адаптера, накопичувачів та інших компонентів [16]. На практиці було

встановлено, що доступність окремих сенсорів залежить від моделі комп'ютера і прав запуску застосунку. Тому реалізація передбачає не лише пряме зчитування основних сенсорів, а й використання альтернативних джерел даних, коли конкретний показник відсутній у LibreHardwareMonitor.

Дані про оперативну пам'ять отримуються через системні механізми Windows, оскільки апаратні бібліотеки можуть відображати віртуальну пам'ять або ліміти замість фактичного обсягу фізичної RAM. Для мережевої активності та активності накопичувачів використовуються лічильники продуктивності Windows. Такий підхід дозволяє отримувати не лише статичні характеристики, а й динамічні показники, наприклад швидкість читання і запису дисків, навантаження на мережевий адаптер та поточну активність накопичувача.

На рисунку 3.1 зображено графічний інтерфейс розробленого застосунку.

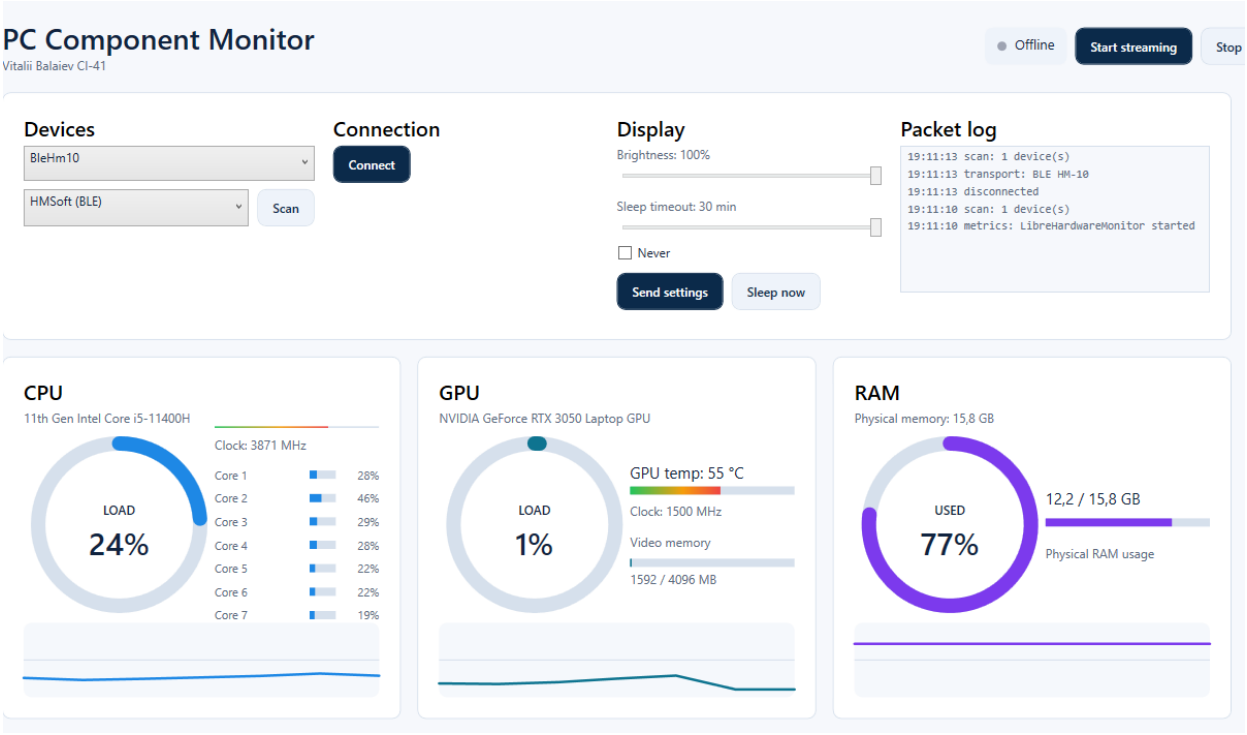

Рисунок 3.1 – Графічний нтерфейс настільного застосунку системи моніторингу компонентів

Після збору показників формується об'єкт поточного стану системи. Він містить групи параметрів CPU, GPU, RAM, дисків, мережі і температур. Цей об'єкт використовується у двох напрямах: для оновлення WPF-інтерфейсу і для формування пакету, який буде переданий на STM32. Таким чином, графічний інтерфейс застосунку і зовнішній пристрій отримують узгоджений набір даних, що зменшує ризик розбіжностей між тим, що бачить користувач у Windows, і тим, що відображається на дисплеї мікроконтролерного пристрою.

Особливе значення має модуль формування протоколу. Передавання даних у вигляді текстового JSON-повідомлення було б простим для читання людиною, але збільшило б обсяг BLE-пакетів і ускладнило б розбір на STM32. Тому реалізовано компактний двійковий формат. Кожен кадр містить сигнатуру PM, версію протоколу, тип пакета, номер послідовності, довжину payload, зашифроване корисне навантаження та CRC-16. Фрагмент методу формування кадру наведено на рисунку 3.1.

```
private byte[] EncodeFrame(PacketType type, byte[] payload)
{
    var frame = new byte[8 + payload.Length + 2];
    frame[0] = Magic0;
    frame[1] = Magic1;
    frame[2] = Version;
    frame[3] = (byte)type;
    var sequence = _sequence++;
    BinaryPrimitives.WriteUInt16LittleEndian(frame.AsSpan(4, 2), sequence);
    BinaryPrimitives.WriteUInt16LittleEndian(frame.AsSpan(6, 2),
(ushort)payload.Length);
    payload.CopyTo(frame.AsSpan(8));
    ProtocolCrypto.Apply((byte)type, sequence, frame.AsSpan(8,
payload.Length));
    var crc = Crc16Ccitt.Compute(frame.AsSpan(2, 6 + payload.Length));
    BinaryPrimitives.WriteUInt16LittleEndian(frame.AsSpan(frame.Length - 2,
2), crc);
    return frame;
}
```

Рисунок 3.1 – Лістинг формування кадру протоколу у WPF-застосунку

У наведеному фрагменті видно, що параметри копіюються у кадр перед обчисленням контрольної суми. Після цього до нього застосовується шифрування AES-128 CTR, а CRC-16 розраховується вже для зашифрованого

вмісту. Така послідовність дозволяє мікроконтролеру спочатку перевірити цілісність отриманих байтів, а лише після цього виконувати розшифрування і декодування. Поле sequence використовується для нумерації пакетів і може бути застосоване під час подальшого аналізу втрати або перестановки кадрів.

Формування пакету виконується з урахуванням обмежень вбудованого пристрою. Значення температур, завантаження і швидкостей передаються не у форматі рядків, а як цілі числа з попередньо визначеним масштабуванням. Наприклад, температура 53,4 °C може бути передана як число 534, а завантаження 27,5 % – як 275. Це зменшує розмір пакета і спрощує обчислення на STM32, де бажано уникати надмірної роботи з рядками та форматами з плаваючою комою.

Bluetooth-взаємодія у застосунку реалізована через транспортний модуль. Після натискання кнопки Scan програма отримує список доступних BLE-пристроїв. Користувач вибирає HM-10 і натискає Connect. Після успішного з'єднання програма надсилає службовий ping-кадр, який дозволяє пристрою відобразити стан Bluetooth connected ще до початку передавання телеметрії. Після натискання кнопки «Start streaming» застосунок переходить до періодичного зчитування метрик і надсилання пакетів параметрів.

Для спрощення запуску застосунку передбачено формування готового виконуваного файлу. Проєкт публікується як одиночний .exe для Windows x64. Завдяки цьому користувач може запускати програму без ручної збірки в середовищі розробки. Такий підхід є важливим для демонстрації системи, оскільки desktop-застосунок виступає не лише як допоміжний інструмент розробника, а як повноцінна частина апаратно-програмного комплексу.

3.2 Реалізація апаратної частини системи моніторингу компонентів ПК

Апаратна частина системи реалізована на базі плати STM32F769I-DISCO та Bluetooth-модуля HM-10. Мікроконтролерна плата виконує функції

приймача даних, обробника протоколу і графічного індикатора. HM-10 забезпечує бездротовий зв'язок із застосунком і передає прийняті BLE-дані на STM32 через UART. Додатково до плати підключено чотири кнопки, які використовуються для локального керування сторінками та режимом сну дисплея.

Початкова ініціалізація прошивки включає налаштування системного тактування, GPIO, UART, CRC, LTDC, DSI, DMA2D і зовнішньої SDRAM. Після запуску периферії ініціалізується модуль дисплея MainLcd, бібліотека LVGL і протокол обміну з ПК [17]. Особливістю STM32F769I-DISCO є необхідність повної ініціалізації дисплейного тракту, оскільки простий запис у framebuffer без BSP-драйвера не забезпечує коректного запуску вбудованого LCD.

Приймання даних від HM-10 організовано у режимі переривань. UART не виконує розбір протоколу і не форматує дані, а лише записує отриманий байт у кільцевий буфер. Така реалізація зменшує тривалість обробки переривання і знижує ризик втрати наступних байтів. Основний цикл поступово дістає байти з буфера та передає їх у парсер протоколу. Фрагмент такого циклу наведено на рисунку 3.2.

```
static void HM10_ProcessReceivedBytes(void)
{
  uint8_t byte;
  PcMonitorParseResult result;

  while (HM10_PopRxByte(&byte) != 0U)
  {
    result = PcMonitorProtocol_ProcessByte(&hm10_parser, byte);
    HM10_HandleParseResult(result);
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == USART6)
  {
    HM10_PushRxByte(hm10_rx_byte);
    HM10_StartReceive();
  }
}
```

Лістинг 3.2 – Обробка прийнятих UART-байтів у firmware STM32

У наведеному фрагменті приймання UART виконує лише дві дії: записує байт у буфер і знову запускає приймання наступного байта. Безпосередній розбір кадру винесений у функцію HM10_ProcessReceivedBytes, яка викликається з головного циклу. Така організація відповідає типовій практиці розробки вбудованих систем: у перериванні виконується мінімальний обсяг роботи, а складніші операції переносяться у контекст основної програми.

Парсер протоколу реалізовано як автомат станів. Він послідовно шукає сигнатуру, читає заголовок, визначає довжину пакету, накопичує корисне навантаження і приймає CRC-16. Якщо версія протоколу неправильна або довжина пакету перевищує допустиме значення, кадр відкидається. Якщо кадр зібрано повністю, обчислюється контрольна сума, після чого дані передаються на розшифрування [18]. Фрагмент початкових станів парсера наведено на рисунку 3.3.

```
switch (parser->state)
{
  case PM_STATE_MAGIC0:
    if (byte == PM_PROTOCOL_MAGIC0)
    {
      parser->state = PM_STATE_MAGIC1;
    }
    return PM_PARSE_NONE;

  case PM_STATE_MAGIC1:
    if (byte == PM_PROTOCOL_MAGIC1)
    {
      parser->header[0] = PM_PROTOCOL_MAGIC0;
      parser->header[1] = PM_PROTOCOL_MAGIC1;
      parser->header_index = 2U;
      parser->state = PM_STATE_HEADER;
    }
    return PM_PARSE_NONE;
}
```

Рисунок 3.3 – Лістинг пошуку сигнатури та читання заголовка кадру на STM32

Такий автомат станів дає змогу відновлювати синхронізацію навіть після помилкових або зайвих байтів у потоці. Якщо послідовність сигнатури не збігається, парсер повертається до пошуку початку кадру. Завдяки цьому пошкоджений пакет не призводить до зупинки пристрою, а лише збільшує

статистику помилок. Ця статистика використовувалась під час налагодження для перевірки відповідності, структури кадру і контрольної суми.

Після приймання валідного пакета параметрів, прошивка оновлює поточний набір даних. Структура містить значення для всіх інформаційних сторінок: завантаження CPU і GPU, температури, частоти, оперативну пам'ять, накопичувачі, мережу і додаткові сенсори. Для пакета налаштувань застосовуються налаштування яскравості та таймера сну, а для командних пакетів виконуються дії сон або пробудження.

Графічний інтерфейс пристрою реалізовано за допомогою LVGL. Для кожної сторінки створено окремий екран і набір об'єктів: текстові мітки, кільцеві індикатори, графічні шкали та компактні графіки історії. Якщо екран ще не створений, відповідна функція створює його об'єкти; якщо екран уже існує, прошивка лише оновлює значення. Фрагмент вибору активної сторінки наведено на рисунку 3.4.

```
if (active_page == PAGE_GPU)
{
  if (gpu_page_ready == 0U)
  {
    CreateGpuScreen();
  }
  UpdateGpuPage(snapshot, telemetry_packets);
  lv_scr_load(gpu_screen);
}
else if (active_page == PAGE_RAM)
{
  if (ram_page_ready == 0U)
  {
    CreateRamScreen();
  }
  UpdateRamPage(snapshot, telemetry_packets);
  lv_scr_load(ram_screen);
}
else
{
  MainLcd_RenderOverview(snapshot, telemetry_packets);
  lv_scr_load(gpu_screen);
}
```

Рисунок 3.4 – Лістинг вибору активної сторінки інтерфейсу LVGL

Повний інтерфейс складається із семи сторінок. Overview використовується як головна сторінка з узагальненими показниками CPU, GPU, RAM і температур. CPU, GPU, Memory, Temperatures, Storage та Network призначені для детальнішого перегляду окремих груп параметрів. Графічний інтерфейс пристрою зображено на рисунку 3.2.

Рисунок 3.2 – Графічний інтерфейс пристрою на дисплеї STM32F769I-DISCO

Обробка кнопок реалізована окремо від важких графічних операцій. Для усунення проблеми, коли натиск на кнопку може не призводити до очікуваної дії - кнопки скануються у системному таймері, а в головний цикл передаються вже зафіксовані події. Фрагмент сканування кнопок наведено на рисунку 3.5.

```
states = 0U;
states |= HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4) == GPIO_PIN_RESET ? 0U : 0x01U;
states |= HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_2) == GPIO_PIN_RESET ? 0U : 0x02U;
states |= HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_10) == GPIO_PIN_RESET ? 0U :
0x04U;
states |= HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_8) == GPIO_PIN_RESET ? 0U : 0x08U;

falling_edges = button_sample_previous_states & (uint8_t)(~states) & 0x0FU);
button_sample_previous_states = states;
if ((falling_edges & 0x01U) != 0U)
{
  button_event_flags |= BUTTON_EVENT_PREVIOUS_PAGE;
}
```

Рисунок 3.5 – Лістинг фіксації коротких натискань фізичних кнопок

У цій реалізації натискання визначається за спадним фронтом сигналу, оскільки кнопки підключені з використанням внутрішніх підтягувальних резисторів до логічної одиниці, а під час натискання на відповідному вході формується логічний нуль. Такий спосіб обробки сигналів є типовим для мікроконтролерних систем, оскільки забезпечує високу завадостійкість та зменшує ймовірність хибних спрацювань. Для реєстрації натискань використовується періодичне опитування стану входів у системному таймері, що дозволяє своєчасно виявляти зміну логічного рівня незалежно від виконання інших програмних модулів. У результаті короткочасні натискання не втрачаються навіть у випадках, коли основний цикл програми виконує оновлення графічного інтерфейсу, обробку телеметричних пакетів або інші ресурсоємні операції. Реалізований механізм забезпечує стабільне та передбачуване керування інтерфейсом користувача за будь-яких режимів роботи системи.

У прошивці також реалізовано послідовність стартових станів, що відображають поточний етап готовності системи до роботи. Після ввімкнення живлення та завершення ініціалізації апаратних ресурсів пристрій переходить у режим очікування встановлення Bluetooth-з'єднання. До моменту приймання першого коректного службового кадру на дисплеї відображається повідомлення «Waiting for Bluetooth connection», яке інформує користувача про необхідність встановлення бездротового зв'язку. Після успішного приймання ping-кадру виконується перевірка працездатності каналу передавання, а інтерфейс автоматично змінюється на повідомлення «Bluetooth connected» із додатковою підказкою щодо запуску потокового передавання телеметричних даних із настільного застосунку. Лише після приймання першого пакета з параметрами персонального комп'ютера система автоматично переходить до відображення головної сторінки інтерфейсу, що виключає появу порожніх або неактуальних значень на дисплеї.

Запропонована послідовність переходів між початковими станами дозволяє користувачу однозначно визначити етап роботи системи та швидко

локалізувати можливі проблеми під час підключення. Відображення інформаційних повідомлень на кожному етапі запуску спрощує перевірку працездатності Bluetooth-модуля, правильності встановлення з'єднання та початку передавання телеметричних даних. Такий підхід підвищує зручність експлуатації пристрою, скорочує час налагодження та забезпечує більш інтуїтивне сприйняття процесу взаємодії між персональним комп'ютером і мікроконтролерною системою моніторингу.

3.3 Налаштування та тестування системи

Після реалізації програмної та апаратної частин було виконано налаштування й поетапне тестування системи. Перевірка проводилась окремо для настільного застосунку, Bluetooth-каналу, UART-приймання, протоколу, дисплея та фізичних кнопок. Такий порядок дозволив локалізувати помилки на конкретному рівні системи і не змішувати проблеми програмного інтерфейсу з помилками апаратного підключення або прошивки [19].

Першим етапом перевірявся WPF-застосунок. Було виправлено помилку пов’язану з WPF binding, що призводила до аварійного завершення готового виконуваного файлу. Після цього застосунок успішно запускався, відображав головну сторінку і оновлював дані ПК.

Окремо перевірялась коректність отримання показників. Для оперативної пам’яті було встановлено, що початковий варіант міг брати не фізичну пам'ять, а значення віртуальної пам'яті. Тому отримання RAM було переведено на системний API Windows. Для температур процесора виявлено, що LibreHardwareMonitor може не показувати сенсори без прав адміністратора. Після запуску застосунку від імені адміністратора температури CPU стали доступними, і ці дані були включені у пакет параметрів.

Для накопичувачів перевірялась наявність активності, вільного місця, швидкості читання і запису та температур. У WPF-застосунку реалізовано

виведення кількох дисків окремо, що дозволило передавати на STM32 не лише один загальний показник, а інформацію про SSD, HDD та інші логічні диски. Для мережі перевірялись її навантаження, а також назва активного мережевого адаптера.

Наступним етапом було тестування Bluetooth-з'єднання. Спочатку HM-10 не відображався під очікуваною назвою під час сканування, однак після перевірки налаштувань Windows Bluetooth було встановлено, що модуль визначається як HMSoft. Після підключення застосунок почав передавати кадри на пристрій.

На стороні STM32 спочатку перевірялась фізична передача байтів через UART. Для цього використовувались лічильники прийнятих байтів і кількість повних кадрів. На одному з етапів було виявлено, що байти надходять, але кадри не збираються. Причиною виявилась невідповідність швидкості UART: типовий baud rate HM-10 становить 9600, тоді як початкове налаштування STM32 відрізнялось. Після встановлення 9600 бод кількість повних кадрів і пакетів даних почала зростати.

Далі перевірялась робота дисплея. Початковий прямий запис у framebuffer не дав очікуваного зображення, тому було встановлено необхідність повної ініціалізації LCD через BSP для STM32F769I-DISCO. Після підключення BSP-драйверів дисплей засвітився, було підтверджено роботу framebuffer, підсвітки і графічного тракту. На наступному етапі додано LVGL, що дозволило перейти від текстового debug-екрана до повноцінного інтерфейсу з картками, індикаторами і графіками.

Під час передачі даних було помічено мерехтіння дисплея. Причина полягала в занадто частому повному оновленні екрана. Для усунення проблеми частоту оновлення LCD обмежено інтервалом 250 мс, а сторінки LVGL оновлюються тільки через відповідні функції зміни значень. Це зменшило навантаження на графічну підсистему і зробило відображення стабільнішим.

Фізичні кнопки перевірялись після завершення реалізації всіх сторінок

графічного інтерфейсу та інтеграції їх із програмною логікою пристрою. Основною метою тестування було підтвердження коректності навігації між сторінками, швидкості реагування на натискання та стабільності роботи системи при одночасному оновленні графічного інтерфейсу. На початковому етапі було виявлено, що короткі натискання могли не реєструватися, якщо основний цикл програми був зайнятий прийманням даних або оновленням великої кількості графічних елементів дисплея. Для усунення цієї проблеми реалізовано швидке періодичне опитування стану кнопок у системному таймері з подальшою передачею сформованих подій до головного циклу обробки. Після внесення відповідних змін натискання почали стабільно реєструватися незалежно від навантаження на систему, що забезпечило комфортне керування пристроєм без необхідності тривалого утримання кнопок.

Під час тестування також перевірялась передача пакетів налаштувань від настільного застосунку до мікроконтролерного пристрою. Застосунок формував окремі пакети, що містили параметри яскравості дисплея та часу переходу пристрою в режим сну. Після приймання цих пакетів прошивка аналізувала отримані значення та застосовувала нові параметри без необхідності перезапуску пристрою або повторного встановлення Bluetooth-з'єднання. На початкових етапах розробки було виявлено, що передавання будь-якого значення яскравості могло призводити до повного вимкнення підсвічування дисплея через некоректне трактування діапазону допустимих значень. Після внесення змін до алгоритму обробки параметрів мінімальний рівень яскравості було обмежено безпечним значенням, що унеможливлює повне згасання дисплея, а початковими параметрами системи встановлено максимальну яскравість та автоматичний перехід у режим сну через 30 хвилин бездіяльності.

Фінальна перевірка підтвердила коректність функціонування всіх програмних та апаратних компонентів системи в комплексі. Настільний застосунок успішно отримує телеметричні дані про стан апаратно-технічних

компонентів персонального комп'ютера, формує телеметричний пакет, виконує його шифрування та передає через інтерфейс Bluetooth Low Energy на модуль HM-10. Мікроконтролер STM32F769I-DISCO приймає послідовність байтів через інтерфейс UART, формує завершений пакет, перевіряє контрольну суму CRC, виконує розшифрування отриманих даних, оновлює внутрішні структури збереження параметрів і відображає актуальну інформацію на дисплеї. Проведене тестування підтвердило відсутність втрати пакетів за нормальних умов експлуатації, а також правильність роботи механізмів обробки можливих помилок передавання даних.

Під час перевірки графічного інтерфейсу було протестовано коректність відображення всіх сторінок системи, включаючи «Overview», «CPU», «GPU», «RAM», «Temperatures», «Storage» та «Network». Для кожної сторінки перевірялась відповідність відображуваних телеметричних параметрів фактичному стану персонального комп'ютера, правильність оновлення числових значень, графічних шкал та інших елементів інтерфейсу. Додатково оцінювалась плавність перемикання між сторінками, швидкість оновлення інформації та відсутність візуальних артефактів під час тривалої безперервної роботи пристрою.

Поетапне тестування дозволило своєчасно виявити та усунути проблеми, пов'язані зі збиранням програмного забезпечення на платформі .NET, отриманням телеметричних даних від апаратних сенсорів персонального комп'ютера, налаштуванням швидкості обміну Bluetooth-модуля HM-10, ініціалізацією дисплея, мерехтінням графічного інтерфейсу та недостатньою чутливістю фізичних кнопок керування. Комплексна перевірка підтвердила працездатність запропонованої архітектури, стабільність функціонування всіх модулів та можливість практичного використання розробленої системи для безперервного моніторингу стану апаратно-технічних компонентів персонального комп'ютера в режимі реального часу.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Вимоги ергономіки до організації робочого місця оператора ПК

Під час розроблення системи моніторингу компонентів ПК основна частина робіт виконується за персональним комп'ютером у положенні сидячи. Оператором у цьому випадку виступає розробник або користувач застосунку, тому робоче місце організовується з додатковою зоною для низьковольтного електронного макета. Ергономічні параметри робочого місця приймаються з урахуванням вимог ДСТУ 7951:2015 до крісла оператора та фактичного характеру роботи: тривале сидіння, робота з екраном, клавіатурою, мишею та періодичне підключення плати [20].

Робоче місце поділяється на основну та допоміжну зони. В основній зоні розміщуються ноутбук або зовнішній монітор, клавіатура та миша. Пристрій розміщується праворуч або ліворуч від клавіатури на рівній непровідній поверхні. Кабелі не перетинають зону руху миші та не проходять перед клавіатурою. Таке компонування дає змогу працювати з кодом і застосунком без постійного перенесення рук над відкритою платою, а також зменшує ймовірність випадкового зміщення HM-10 або провідників TXD/RXD.

Положення основного екрана вибирається так, щоб відстань від очей оператора до площини екрана становила 600–700 мм. Верхня межа екрана розміщується на рівні очей або на 50–100 мм нижче, тому напрямок погляду під час роботи залишається природно нахиленим униз. Дисплей плати використовується як допоміжний екран, тому він встановлюється нижче основного монітора, але в межах прямої видимості.

Стіл використовується з достатньою глибиною для одночасного розміщення монітора, клавіатури, миші, плати та кабелів. Перед клавіатурою

залишається вільна ділянка 100–150 мм для опори кистей. Висота робочої поверхні узгоджується з висотою сидіння так, щоб передпліччя під час роботи з клавіатурою утворювали приблизно горизонтальну лінію, а кут у ліктьовому суглобі був близький до 90°.

Крісло оператора вибирається регульованим. Висота поверхні сидіння встановлюється відповідно до висоти підколінної ямки оператора при згинанні колінного та гомілково-стопного суглобів на 90°. У ДСТУ 7951:2015 також враховано висоту підошви 30 мм, тому під час налаштування сидіння стопи розміщуються на підлозі без зависання. Регулювання висоти сидіння та інших лінійних параметрів виконується плавно або ступінчасто з кроком 15–25 мм. Це дає змогу точно підібрати положення для тривалої роботи з кодом, графічним інтерфейсом та апаратним макетом.

Поверхня сидіння приймається плоскою з нахилом 0–7° або профільованою. Для профільованого сидіння передній кут нахилу становить 4–5°, а задній кут — 10–15°. Така форма зменшує тиск на стегна та підтримує стабільну посадку під час програмування і тестування пристрою. Ширина сидіння вибирається за найбільшим діаметром стегон для 95-го перцентиля, а глибина сидіння — як 2/3 відстані від найбільш виступаючої назад точки сідниць до передньої поверхні коліна. Передній край сидіння не тисне на підколінну ділянку, оскільки це порушує кровообіг під час тривалого сидіння.

Спинка крісла забезпечує опору поперекової та грудної ділянок хребта. Для поперекової спинки висота опорної поверхні приймається в межах 150–280 мм із горизонтальною віссю симетрії на рівні лінії талії. Профільована спинка має радіус кривизни поперекової опори 460 мм і радіус вигину для грудного відділу 620 мм. Найбільш виступаюча точка поперекової опори розміщується на висоті 140 мм від нижньої кромки опорної поверхні, точка переходу кривизни — на висоті 255 мм, а верхня точка вигину грудного відділу — на висоті 380 мм. Кут нахилу спинки для роботи встановлюється в межах 95–110°, а для короткого відпочинку допускається збільшення кута до 115–135°.

Підлокітники використовуються для зменшення статичного навантаження на плечовий пояс під час роботи з клавіатурою, мишею та тестуванням застосунку. Висота підлокітника налаштовується за висотою ліктя над сидінням при куті згинання ліктьового суглоба 90°. Ширина підлокітника приймається 50–80 мм. Фіксований кут нахилу підлокітників становить 0–5°, а для регульованих підлокітників допускається діапазон 0–20°. Якщо висота столу або сидіння не дозволяє стабільно поставити стопи на підлогу, використовується підставка для ніг з кутом нахилу 10–40°. При куті понад 20° передбачається упор для п’яти висотою 25 мм.

Освітлення робочого місця організовується так, щоб на основному екрані та LCD-дисплеї плати не виникали прямі відблиски. Освітленість робочої поверхні в зоні клавіатури, миші та апаратного макета приймається не менше 300 лк, а для перевірки дрібних підключень доцільно забезпечити локальне освітлення на рівні 500 лк. Основне освітлення рівномірно освітлює стіл, а локальне світло спрямовується на клавіатуру та зону підключення проводів, не потрапляючи безпосередньо в очі. Монітор розміщується так, щоб вікно або лампа не відбивались у темних ділянках інтерфейсу.

Візуальна ергономіка врахована на рівні програмного інтерфейсу. Основні параметри згруповані в окремі картки, а для швидкого сприйняття використовуються кільцеві індикатори, прогрес-бари, графіки та короткі підписи. На сторінках не використовується надлишковий текст, а числові значення розміщені поруч із відповідними індикаторами.

Інтерфейс STM32 також виконує ергономічну функцію. Дисплей має сім сторінок із цифровим індикатором номера сторінки, що дає змогу швидко визначити поточний екран. Головна сторінка містить лише ключові показники: завантаження процесора і відеокарти, використання пам’яті та температури. Детальні сторінки поділяють інформацію за компонентами, тому на дисплеї не виникає перевантаження дрібним текстом. Стан Bluetooth показується окремою піктограмою та кольоровим індикатором. Екран очікування містить коротке повідомлення про стан підключення, а після

встановлення зв’язку відображається готовність до запуску передачі з desktop-застосунку.

Додатково контролюється порядок розміщення кабелів. USB-кабель ST-LINK, кабель живлення ноутбука та провідники Bluetooth-модуля не проходять через зону руху миші. Проводи UART укладаються коротким маршрутом без натягу, щоб під час переміщення плати не виникало зусилля на контактах HM-10. На робочому столі не залишаються металеві предмети біля відкритої електроніки. Така організація робочої зони одночасно підвищує ергономіку та зменшує ризик пошкодження макета [21]. Робоче місце забезпечує зручне виконання програмування, тестування, демонстрації та обслуговування системи моніторингу компонентів ПК

4.2 Заходи щодо захисту від ураження електричним струмом

Розроблена система належить до малопотужних електронних пристроїв, однак під час розроблення вона взаємодіє з обладнанням, що живиться від електричної мережі. До складу робочого комплексу входять ноутбук, штатний блок живлення ноутбука, плата STM32F769I-DISCO, Bluetooth-модуль HM-10, USB-кабелі та допоміжні провідники. Відповідно до ДСТУ 61140:2019 небезпечні струмовідні частини не мають бути доступними за нормальних умов, а доступні провідні частини не мають ставати небезпечними за умови одиничного пошкодження. Тому основні рішення з електробезпеки приймаються на рівні вибору низьковольтного живлення, ізоляції, порядку підключення та відокремлення відкритої плати від мережевої напруги [22].

У ДСТУ 61140:2019 наднизька напруга визначається як напруга менше 50 В змінного струму або менше 120 В постійного струму. У розробленій системі робочі напруги апаратного макета значно нижчі за ці межі: плата STM32F769I-DISCO живиться через USB напругою 5 В постійного струму, а логічні лінії HM-10 та UART працюють на рівні 3,3 В. Це зменшує ризик ураження струмом під час роботи з макетом. Відкрита частина пристрою не

підключається безпосередньо до мережі 220 В, а живлення від мережі використовується лише через штатний адаптер ноутбука.

Блок живлення ноутбука розглядається як основний елемент, що відокремлює користувача від мережевої напруги 220 В частотою 50 Гц. Для роботи використовується лише справний адаптер без тріщин корпуса, пошкоджень вилки, перегинів кабелю та слідів перегрівання. Перед початком роботи перевіряється цілісність ізоляції на всій довжині кабелю. Розетка використовується на номінальну напругу 220–250 В і струм не менше 10 А, без люфту контактів і нагрівання під час роботи. Якщо застосовується подовжувач, його номінальний струм приймається не менше 10 А, а для запасу за навантаженням доцільно використовувати подовжувач на 16 А. Кабель подовжувача повністю розмотується при тривалій роботі, не затискається ніжками стола або стільця і не прокладається в місці проходу. Це усуває механічне пошкодження ізоляції та зменшує ризик перегрівання під час тривалого тестування.

Захист низьковольтної частини забезпечується правильним вибором джерела та рівнів сигналів. HM-10 підключається до лінії 3,3 В, а не до 5 В, оскільки його UART-лінії працюють з логічними рівнями 3,3 В. Лінії TXD і RXD підключаються перехресно, а GND модуля з’єднується зі спільною землею плати. Перед подаванням живлення перевіряється правильність підключення чотирьох основних ліній. Підключення до випадкових контактів плати не застосовується, оскільки воно може пошкодити порт мікроконтролера або модуль Bluetooth.

Низьковольтні з’єднання виконуються при вимкненому живленні або до підключення USB-кабелю до ПК. Після зміни монтажу проводиться візуальний контроль, і лише після цього плата підключається до комп’ютера. Такий порядок відповідає принципу основного захисту: оператор не контактує з провідниками під час подавання живлення. Якщо потрібно змінити підключення HM-10 або кнопок, USB-кабель спочатку від’єднується, а повторне підключення виконується після перевірки відсутності короткого

замикання між 3,3 В і GND.

У місцях частого підключення використовуються готові кабелі з надійними роз'ємами. Така організація монтажу зменшує ризик короткого замикання між лінією живлення 3,3 В, землею та сигнальними лініями UART.

Для основного захисту від ураження електричним струмом використовуються ізоляція та недоступність небезпечних частин. У цій системі небезпечна напруга мережі залишається всередині штатного адаптера ноутбука. Доступна частина пристрою працює від 5 В і 3,3 В постійного струму. Для додаткового захисту робоче місце підключається через справну розетку, а в електромережі доцільно використовувати пристрій захисного вимикання з номінальним диференційним струмом не більше 30 мА.

У ДСТУ 61140:2019 наведено пороги напруги дотику для реакції людини: реакція переляку можлива від 2 В змінного струму або 8 В постійного струму, а м'язова реакція — від 20 В змінного струму або 40 В постійного струму за сухих умов і площі контакту 35 см². У макеті використовуються 3,3 В і 5 В постійного струму, тому робочі напруги нижчі за поріг м'язової реакції. Незважаючи на це, дотик до контактів під час роботи не допускається, оскільки коротке замикання може пошкодити плату, Bluetooth-модуль або USB-порт комп'ютера.

Під час роботи забороняється виконувати підключення мокрими руками, розміщувати напої біля ноутбука та плати, торкатися оголених контактів під напругою і використовувати пошкоджені кабелі. У разі появи запаху перегрівання, миготіння живлення, сильного нагрівання компонентів, нестабільної роботи дисплея або самовільного перезапуску плати живлення відключається одразу. Після відключення проводиться огляд USB-кабелю, модуля HM-10, роз'ємів, ліній 3,3 В, GND, TXD і RXD. Повторний запуск виконується лише після усунення причини несправності. Такий порядок мінімізує ризик ураження електричним струмом і зменшує ймовірність пошкодження компонентів системи.

ВИСНОВКИ

У результаті виконаної роботи було розроблено та реалізовано апаратно-програмну систему моніторингу апаратно-технічних компонентів персонального комп'ютера, призначену для безперервного збору, передавання, обробки та візуалізації телеметричних даних у режимі реального часу. Запропонована система поєднує програмне забезпечення персонального комп'ютера, бездротовий канал зв'язку Bluetooth Low Energy та мікроконтролерний пристрій на базі плати STM32F769I-DISCO, що забезпечує відображення інформації про стан основних апаратних ресурсів на вбудованому дисплеї.

У процесі розробки було обґрунтовано вибір архітектури системи та її основних апаратних і програмних компонентів. Програмна частина персонального комп'ютера реалізована із застосуванням платформи WPF та забезпечує отримання параметрів про завантаження центрального та графічного процесорів, використання оперативної пам'яті, температурні показники й інші параметри функціонування персонального комп'ютера. Використання бездротового інтерфейсу Bluetooth Low Energy для передавання інформації дозволило відмовитися від дротового з'єднання між персональним комп'ютером та пристроєм відображення.

Отримані результати підтверджують ефективність запропонованої архітектури та доцільність використання розробленої системи для оперативного моніторингу стану апаратно-технічних компонентів персонального комп'ютера. Створений програмно-апаратний комплекс характеризується модульною побудовою, можливістю подальшого функціонального розширення та адаптації до інших типів обчислювальних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 39 с.

2. Буров Є.В., Митник М.М. Комп'ютерні мережі. Підручник. Том другий. Львів: «Магнолія 2006», 2024. 204 с.

3. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп'ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.

4. Антонюк В.І., Луцик Н.С., Паламар А.М. Комп'ютеризована IoT-система для аналізу споживання електроенергії у житлових приміщеннях. Актуальні задачі сучасних технологій : збірник тез доповідей XIV міжнародної науково-практичної конференції молодих учених та студентів (Тернопіль, 11-12 грудня 2025 року), Тернопіль: ФОП Паляниця В. А., 2025. С. 225.

5. Луцик Н., Антонюк В., Паламар А. Структура IoT-системи для моніторингу параметрів електричних мереж житлових приміщень. Матеріали XIII науково-технічної конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 17-18 грудня 2025 року), Тернопіль: ТНТУ, 2025. С. 205.

6. Бородій, Г. Осухівська. Порівняльний аналіз архітектур обробки IoT-даних для екологічного моніторингу на Raspberry Pi 5. Вісник Хмельницького національного університету. Технічні науки. 2026. Т. 365, №3. *365*(3), С. 474-479. https://doi.org/10.31891/2307-5732-2026-365-67

7. Оконський М.В., Лупенко С.А., Паламар А.М. Комп'ютерна система для моніторингу метеорологічних параметрів на основі IoT. Актуальні задачі сучасних технологій : збірник тез доповідей X міжнародної науково-

практичної конференції молодих учених та студентів (Тернопіль, 24–25 листопада 2021 року), Тернопіль: ТНТУ, 2021. С. 112.

8. Паламар А., Величко Д. Система моніторингу якості повітря в приміщеннях. Матеріали V Міжнародної студентської науково-технічної конференції "Природничі та гуманітарні науки. Актуальні питання" (Тернопіль, 28-29 квітня 2022 року), Тернопіль: ТНТУ. 2022. С. 138.

9. STM32F769I-DISCO, STMicroelectronics Data sheet. URL: https://www.st.com/en/evaluation-tools/32f769idiscovery.html (дата звернення: 08.05.2026)

10. ARM Cortex-M7 Technical Reference Manual. URL: https://developer.arm.com/documentation/ddi0489/f/ (дата звернення: 27.04.2026)

11. Bluetooth module HM-10 Datasheet. URL: https://people.ece.cornell.edu/land/courses/ece4760/PIC32/uart/HM10/DSD%20TECH%20HM-10%20datasheet.pdf (дата звернення: 21.04.2026)

12. UART протокол. URL: https://resources.altium.com/p/serial-communications-protocols-part-two-uart (дата звернення: 22.04.2026)

13. Симетричний алгоритм блочного шифрування AES. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (дата звернення: 19.05.2026)

14. Романов Д.В., Осухівська Г.М., Паламар А.М. Функціональна схема системи керування зовнішнім освітленням на основі технології LoRa. Матеріали IX науково-технічної конференції "Інформаційні моделі, системи та технології" Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 8–9 грудня 2021 року), Тернопіль: ТНТУ, 2021. С. 124.

15. Документація бібліотеки WPF. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 18.05.2026)

16. Бібліотека для збору параметрів ПК LibreHardwareMonitor. URL: https://librehardwaremonitor.com/category/guide/ (дата звернення: 23.05.2026)

17. Документація бібліотеки графічного нтерфейсу для вбудованих систем LVGL. URL: https://lvgl.io/docs (дата звернення: 15.05.2026)

18. Ясінський Р.В., Осухівська Г.М., Паламар А.М., Величко Д.В. Комп'ютерна система для контролю параметрів мікроклімату теплиць на основі інтернету речей. Актуальні задачі сучасних технологій : збірник тез доповідей XI міжнародної науково-практичної конференції молодих учених та студентів (Тернопіль, 7-8 грудня 2022 року), Тернопіль: ФОП Паляниця В. А., 2022. С. 177.

19. Слюз І., Жаровський Р. Принципи та основні етапи комплексного тестування комп'ютерної інформаційної системи. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 93.

20. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці. – К.: Каравела, 2007. 408 с.

21. ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора.

22. НПАОП 0,00-7.15-18 Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, від 14.02.2018 року №207.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС

_____ Осухівська Г.М.

“ 2 ” лютого 2026 р.

Система моніторингу апаратно-програмних компонентів комп’ютерної техніки

ТЕХНІЧНЕ ЗАВДАННЯ

на _9_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп’ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-41
_____ к.т.н., доц. Лещишин Ю. З.	_____ Балаєв В. А.
“ 2 ” лютого 2026 р.	“ 2 ” лютого 2026 р.

Тернопіль 2026

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Система моніторингу апаратно-програмних компонентів комп’ютерної техніки».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.141.00.00

1.2 Виконавець

Студент групи СІ-41, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Балаєв В. А.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/9-188 від 24.04.2026 р)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 26.01.2026 р.

Плановий термін завершення виконання кваліфікаційної роботи – 21.06.2026 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Розроблювана система призначена для моніторингу апаратно-технічних компонентів персонального комп’ютера з виведенням основних показників на окремий графічний пристрій. Система забезпечує зчитування параметрів процесора, відеокарти, оперативної пам’яті, накопичувачів, мережевого інтерфейсу та температурних датчиків за допомогою застосунку на ПК, передавання даних через Bluetooth-модуль HM-10 та відображення отриманої інформації на LCD-дисплеї плати STM32F769I-DISCO.

2.2 Мета створення системи

Метою створення системи є розробка апаратно-програмного комплексу для зручного локального моніторингу компонентів персонального комп’ютера з використанням графічного пристрою відображення. Система має забезпечити автоматизоване зчитування параметрів ПК, захищене бездротове передавання даних на мікроконтролерну плату та відображення інформації у вигляді сучасного графічного інтерфейсу.

2.3 Характеристика об'єкту

Об’єктом моніторингу є персональний комп’ютер або ноутбук, апаратні компоненти якого змінюють свої параметри залежно від навантаження. До таких параметрів належать завантаження процесора, частота процесора, температура ядер, завантаження та температура відеокарти, використання відеопам’яті, використання оперативної пам’яті, активність накопичувачів, температура дисків, а також швидкість передавання і приймання даних через мережевий інтерфейс.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Система має складатися з двох основних частин: програмної частини на персональному комп’ютері та апаратної частини на базі STM32F769I-DISCO. Настільний застосунок виконує зчитування параметрів ПК, формує пакет параметрів та передає його через Bluetooth. Апаратна частина приймає дані, обробляє отримані пакети, перевіряє їх цілісність, розшифровує корисне навантаження та відображає інформацію на LCD-дисплеї.

Структура системи має передбачати окремі інформаційні сторінки для різних груп параметрів. Перемикання сторінок здійснюється фізичними кнопками, підключеними до GPIO-виводів мікроконтролера.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Зв’язок між ПК і пристроєм відображення має виконуватись за допомогою Bluetooth Low Energy через модуль HM-10. На стороні ПК передавання даних виконується застосунком, а на стороні STM32 модуль HM-10 передає прийняті байти до мікроконтролера через UART.

3.1.3 Вимоги до режимів функціонування системи

У нормальному режимі застосунок регулярно зчитує показники компонентів ПК, оновлює локальний інтерфейс і передає актуальні дані на пристрій. STM32 приймає пакети, оновлює внутрішній знімок телеметрії та перемальовує активну сторінку інтерфейсу.

У режимі очікування Bluetooth-з’єднання пристрій відображає повідомлення про очікування підключення. Після встановлення з’єднання, але до запуску потокового передавання, на дисплеї відображається повідомлення про готовність до приймання даних. У режимі сну дисплей вимикається для зменшення енергоспоживання, при цьому пристрій залишається готовим до подальшої роботи.

3.1.4 Вимоги по діагностуванню системи

Система має підтримувати засоби діагностики для перевірки коректності передавання і приймання даних. На стороні застосунку має відображатись стан Bluetooth-підключення, активність потокового передавання та доступність апаратних сенсорів. На стороні STM32 має виконуватись підрахунок прийнятих байтів, успішно оброблених кадрів, помилок CRC, помилок розшифрування та кількості прийнятих пакетів телеметрії.

Для налагодження апаратної частини допускається використання світлодіодної індикації, яка сигналізує про приймання валідних пакетів або помилки протоколу. Такий підхід спрощує перевірку системи на етапах, коли дисплей або Bluetooth-з’єднання ще не повністю налаштовані.

3.1.5 Перспективи розвитку, проектування системи

Подальший розвиток системи може передбачати додавання нових сторінок інтерфейсу, підтримку інших типів сенсорів, розширення списку апаратних параметрів, збереження історії показників на ПК, автоматичне

виявлення доступних Bluetooth-пристроїв і створення профілів для різних комп’ютерів.

Також можливе вдосконалення апаратної частини шляхом додавання сучаснішого BLE-модуля з вищою швидкістю передавання даних.

3.2 Показники призначення

Система розрахована на роботу з одним персональним комп’ютером і одним пристроєм відображення. Оновлення телеметрії має виконуватись із частотою, достатньою для зручного спостереження за зміною параметрів у реальному часі. Затримка між зчитуванням показників на ПК та їх відображенням на дисплеї має залишатись візуально непомітною для користувача.

Пристрій має відображати не менше семи інформаційних сторінок, що охоплюють загальний стан системи, процесор, відеокарту, оперативну пам’ять, температури, накопичувачі та мережу. Desktop-застосунок має забезпечувати керування потоковим передаванням, налаштування яскравості дисплея та встановлення таймауту сну.

3.2.1 Вимоги до надійності

Система має стабільно працювати під час тривалого передавання телеметрії. Втрата окремого пакета не повинна призводити до зависання пристрою або некоректного стану інтерфейсу. При відновленні передавання STM32 має продовжити оновлення показників за наступними валідними пакетами.

Програмне забезпечення пристрою має обробляти помилки протоколу, некоректні CRC, неповні кадри та пошкоджені дані. Desktop-застосунок має коректно реагувати на відсутність Bluetooth-з’єднання, недоступність окремих апаратних сенсорів або запуск без прав адміністратора, якщо частина показників не може бути зчитана.

3.3 Вимоги до безпеки

Апаратна частина системи має працювати від низьковольтного живлення. Плата STM32F769I-DISCO підключається через USB, а Bluetooth-модуль HM-10 живиться від лінії 3,3 В. Підключення HM-10 до 5 В без узгодження рівнів не допускається, оскільки це може пошкодити модуль.

Усі зовнішні провідники мають бути надійно підключені та не створювати ризику короткого замикання. Під час налагодження пристрій розміщується на непровідній поверхні. У готовому виконанні доцільно передбачити корпус або захисну основу, що закриває контакти живлення та сигнальні лінії від випадкового дотику.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Система експлуатується в умовах звичайного робочого приміщення. Під час роботи необхідно уникати потрапляння вологи на плату STM32F769I-DISCO, Bluetooth-модуль і USB-з’єднання. Підключення та відключення провідників UART виконується при вимкненому живленні пристрою.

Технічне обслуговування полягає у перевірці цілісності USB-кабелю, надійності підключення HM-10, справності фізичних кнопок і коректності роботи дисплея. У разі нестабільної роботи необхідно перевірити живлення, спільну землю, правильність підключення TXD/RXD та наявність Bluetooth-з’єднання з ПК.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система має забезпечувати захист даних, що передаються між desktop-застосунком і пристроєм. Для цього корисне навантаження пакетів телеметрії шифрується алгоритмом AES-128. Додатково кожен кадр містить службову

сигнатуру та контрольну суму CRC, що дозволяє відкидати пошкоджені або некоректні пакети.

3.4.1 Вимоги по збереженню інформації при аваріях

Система призначена для оперативного моніторингу поточного стану ПК, тому основні дані не потребують обов’язкового довготривалого збереження на пристрої. У разі втрати Bluetooth-з’єднання або зупинки передавання на дисплеї можуть залишатись останні успішно отримані значення або відображатись стан очікування нових даних.

3.4.2 Вимоги по стандартизації і уніфікації

У системі використовуються стандартні інтерфейси та поширені програмні засоби: UART для обміну між STM32 і HM-10, Bluetooth Low Energy для зв’язку з ПК, USB для живлення та прошивання плати, WPF для desktop-застосунку, LibreHardwareMonitor для отримання показників ПК і LVGL для побудови графічного інтерфейсу на дисплеї.

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– зчитування параметрів комп’ютерної техніки;
– пошук і вибір Bluetooth-пристрою HM-10;
– формування пакетів параметрів;
– шифрування корисного навантаження пакетів AES-128;
– розшифрування прийнятих даних;
– відображення показників на LCD-дисплеї;
– перемикання сторінок фізичними кнопками;
– відображення стану Bluetooth-з’єднання.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

а) Структурна схема.

б) Блок схема роботи комп’ютеризованої системи.

в) Схема електрична принципова.

г) Схема електрична монтажна (з’єднань).

*Примітка: У комплект документації можуть вноситися зміни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи бакалавра	Термін виконання
1	*Розробка технічного завдання*	*26.01 – 02.02*
2	*Аналіз технічного завдання на розробку системи моніторингу компонентів*	*03.02 – 15.02*
3	*Проєктування системи моніторингу апаратно-технічних компонентів комп’ютера*	*20.04 – 10.05*
4	*Програмування та тестування системи моніторингу апаратно-технічних компонентів персонального комп’ютера*	*11.05 – 24.05*
5	*Безпека життєдіяльності, основи охорони праці*	*25.05 – 31.05*
6	*Оформлення пояснювальної записки і графічного матеріалу*	*1.06 – 7.06*
7	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*
8	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*
9	*Захист кваліфікаційної роботи бакалавра*	*22.06*

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Лістинг коду настільної WPF програми

```
using LibreHardwareMonitor.Hardware;
using Microsoft.Win32;
using MonitorDesktopApp.Models;
using System.Diagnostics;
using System.IO;
using System.Runtime.InteropServices;
using System.Security.Principal;

namespace MonitorDesktopApp.Services.Metrics;

public sealed class LibreHardwareMetricCollector : IMetricCollector
{
    private readonly Computer _computer;
    private readonly List<PerformanceCounter> _performanceCounters =
new();
    private readonly Dictionary<string, PerformanceCounter>
_logicalDiskCounters = new(StringComparer.OrdinalIgnoreCase);
    private readonly double _baseCpuClockMhz;
    private readonly PerformanceCounter? _cpuPerformanceCounter;
    private bool _disposed;

    public LibreHardwareMetricCollector()
    {
        _baseCpuClockMhz = ReadCpuClockFromRegistry();
        _cpuPerformanceCounter = TryCreatePerformanceCounter("Processor
Information", "% Processor Performance", "_Total") ??
                                 TryCreatePerformanceCounter("Processor
Information", "% Processor Performance", "0,_Total");

        _computer = new Computer
        {
            IsCpuEnabled = true,
            IsGpuEnabled = true,
            IsMemoryEnabled = true,
            IsMotherboardEnabled = true,
            IsControllerEnabled = true,
            IsNetworkEnabled = true,
```

```
                IsStorageEnabled = true
            };
            _computer.Open();
        }

        public string SourceName => "LibreHardwareMonitor";

        public Task<PcMetricsSnapshot> ReadSnapshotAsync(CancellationToken
cancellationToken)
        {
            cancellationToken.ThrowIfCancellationRequested();
            UpdateHardwareTree();

            var allHardware = FlattenHardware(_computer.Hardware).ToArray();
            var cpuHardware = allHardware.FirstOrDefault(hardware =>
hardware.HardwareType == HardwareType.Cpu);
            var gpuHardware = allHardware.FirstOrDefault(hardware =>
                hardware.HardwareType is HardwareType.GpuNvidia or
HardwareType.GpuAmd or HardwareType.GpuIntel);
            var memoryHardware = allHardware.FirstOrDefault(hardware =>
hardware.HardwareType == HardwareType.Memory);
            var networkHardware = PickNetworkHardware(allHardware);
            var storageHardware = allHardware.Where(hardware =>
hardware.HardwareType == HardwareType.Storage).ToArray();
            var drives = ReadDrives(storageHardware);
            var cpu = ReadCpu(cpuHardware);
            var gpu = ReadGpu(gpuHardware);
            var temperatures = ReadTemperatures(allHardware).ToList();
            var diagnostics = ReadSensorDiagnostics(allHardware).ToList();
            diagnostics.InsertRange(0, ReadCollectorDiagnostics());

            cpu = ApplyCpuTemperatureFromLibreHardwareSensors(cpu,
temperatures);

            var snapshot = new PcMetricsSnapshot
            {
                Timestamp = DateTime.Now,
                Cpu = cpu,
                Gpu = gpu,
                Memory = ReadMemory(memoryHardware),
                Network = ReadNetwork(networkHardware),
                Storage = drives.OrderByDescending(drive =>
```

```
drive.ActivityPercent).FirstOrDefault() ?? new StorageMetrics(),
                Drives = drives,
                Temperatures = temperatures.Take(64).ToArray(),
                SensorDiagnostics = diagnostics.Take(180).ToArray()
            };

            return Task.FromResult(snapshot);
        }

        public void Dispose()
        {
            if (_disposed)
            {
                return;
            }

            _computer.Close();
            foreach (var counter in _performanceCounters)
            {
                counter.Dispose();
            }

            _logicalDiskCounters.Clear();
            _disposed = true;
        }

        private CpuMetrics ReadCpu(IHardware? hardware)
        {
            if (hardware is null)
            {
                return new CpuMetrics();
            }

            var sensors = FlattenHardware(new[] { hardware })
                .SelectMany(item => item.Sensors)
                .ToArray();
            var load = ReadValue(FindSensor(sensors, SensorType.Load, "CPU
Total", "Total")) ??
                       AverageSensors(sensors, SensorType.Load, "CPU Core",
"Core");
```

```
            var temp = ReadValue(FindSensor(sensors,
                           SensorType.Temperature,
                           "CPU Package",
                           "Package",
                           "Core Max",
                           "Core Average",
                           "Tctl",
                           "Tdie",
                           "CCD")) ?? 0;
            if (temp <= 0)
            {
                temp = AverageSensors(sensors, SensorType.Temperature, "CPU
Core", "Core #", "Core ");
            }

            if (temp <= 0)
            {
                temp = AverageSensors(sensors, SensorType.Temperature);
            }
            var power = ReadValue(FindSensor(sensors,
                            SensorType.Power,
                            "CPU Package",
                            "Package",
                            "PPT",
                            "Power")) ??
                        AverageSensors(sensors, SensorType.Power);
            var clock = ReadCurrentCpuClockMhz();
            if (clock <= 0)
            {
                clock = AverageSensors(sensors, SensorType.Clock, "CPU
Core", "Core", "Effective");
            }

            if (clock <= 0)
            {
                clock = AverageSensors(sensors, SensorType.Clock);
            }

            if (clock <= 0)
            {
```

```
                clock = ReadCpuClockFromRegistry();
            }
            var coreLoads = sensors
                .Where(sensor => sensor.SensorType == SensorType.Load)
                .Where(sensor => ContainsAny(sensor.Name, "CPU Core", "Core
#", "Core "))
                .Take(12)
                .Select((sensor, index) => new
NamedMetric(NormalizeCoreName(sensor.Name, index), ReadValue(sensor) ?? 0,
"%"))
                .ToArray();

            if (coreLoads.Length == 0 && load > 0)
            {
                coreLoads = new[] { new NamedMetric("Total", load, "%") };
            }

            return new CpuMetrics
            {
                Name = hardware.Name,
                LoadPercent = ClampPercent(load),
                TemperatureCelsius = Math.Max(0, temp),
                ClockMhz = Math.Max(0, clock),
                PowerWatts = Math.Max(0, power),
                CoreLoads = coreLoads
            };
        }

        private GpuMetrics ReadGpu(IHardware? hardware)
        {
            if (hardware is null)
            {
                return new GpuMetrics();
            }

            var sensors = hardware.Sensors;
            var load = ReadValue(FindSensor(sensors, SensorType.Load, "GPU
Core", "D3D 3D", "3D")) ??
                       AverageSensors(sensors, SensorType.Load, "GPU");
            var temp = ReadValue(FindSensor(sensors, SensorType.Temperature,
"GPU Core", "GPU Hot Spot", "GPU Hotspot", "Hot Spot")) ??
```

```
                        AverageSensors(sensors, SensorType.Temperature,
"GPU");

            var clock = ReadValue(FindSensor(sensors, SensorType.Clock, "GPU
Core", "Core")) ??

                        AverageSensors(sensors, SensorType.Clock);

            var power = ReadValue(FindSensor(sensors, SensorType.Power, "GPU
Power", "Total Board Power", "Board Power", "Power")) ??

                        AverageSensors(sensors, SensorType.Power);

            var memoryLoad = ReadValue(FindSensor(sensors, SensorType.Load,
"GPU Memory", "Memory")) ?? 0;

            var memoryUsed = ReadValue(FindSensor(sensors,
SensorType.SmallData, "GPU Memory Used", "Memory Used")) ??

                             ReadValue(FindSensor(sensors, SensorType.Data,
"GPU Memory Used", "Memory Used")) ?? 0;

            var memoryTotal = ReadValue(FindSensor(sensors,
SensorType.SmallData, "GPU Memory Total", "Memory Total")) ??

                              ReadValue(FindSensor(sensors, SensorType.Data,
"GPU Memory Total", "Memory Total")) ?? 0;


            return new GpuMetrics
            {
                Name = hardware.Name,
                LoadPercent = ClampPercent(load),
                TemperatureCelsius = Math.Max(0, temp),
                ClockMhz = Math.Max(0, clock),
                PowerWatts = Math.Max(0, power),
                MemoryLoadPercent = ClampPercent(memoryLoad),
                MemoryUsedMb = Math.Max(0, memoryUsed),
                MemoryTotalMb = Math.Max(0, memoryTotal)
            };
        }


        private MemoryMetrics ReadMemory(IHardware? hardware)
        {
            var windowsMemory = ReadPhysicalMemoryFromWindows();
            if (windowsMemory is not null)
            {
                return windowsMemory;
            }


            if (hardware is null)
            {
```

```
                return new MemoryMetrics();
            }

            var sensors = hardware.Sensors;
            var load = ReadValue(FindSensorExcluding(sensors,
SensorType.Load, new[] { "Virtual", "Commit" }, "Memory", "RAM")) ??
                       AverageSensorsExcluding(sensors, SensorType.Load,
new[] { "Virtual", "Commit" }, "Memory", "RAM");
            var used = ReadValue(FindSensorExcluding(sensors,
SensorType.Data, new[] { "Virtual", "Commit" }, "Memory Used", "Used Memory",
"Used")) ?? 0;
            var available = ReadValue(FindSensorExcluding(sensors,
SensorType.Data, new[] { "Virtual", "Commit" }, "Memory Available",
"Available")) ?? 0;
            var total = used + available;

            return new MemoryMetrics
            {
                LoadPercent = ClampPercent(load),
                UsedGb = Math.Max(0, used),
                TotalGb = Math.Max(0, total),
                VirtualLoadPercent = 0,
                VirtualUsedGb = 0,
                VirtualTotalGb = 0
            };
        }

        private NetworkMetrics ReadNetwork(IHardware? hardware)
        {
            if (hardware is null)
            {
                return new NetworkMetrics();
            }

            var sensors = hardware.Sensors;
            var upload = ReadValue(FindSensor(sensors,
SensorType.Throughput, "Upload", "Tx")) ?? 0;
            var download = ReadValue(FindSensor(sensors,
SensorType.Throughput, "Download", "Rx")) ?? 0;

            return new NetworkMetrics
            {
```

```
                AdapterName = hardware.Name,
                UploadBytesPerSecond = Math.Max(0, upload),
                DownloadBytesPerSecond = Math.Max(0, download)
            };
        }

        private IReadOnlyList<StorageMetrics>
ReadDrives(IEnumerable<IHardware> hardware)
        {
            var logicalDrives = ReadLogicalDrives();
            if (logicalDrives.Count > 0)
            {
                return logicalDrives
                    .OrderBy(drive => drive.Name)
                    .Take(4)
                    .ToArray();
            }

            var physicalDrives = hardware
                .Select(ReadStorage)
                .Where(drive => !string.IsNullOrWhiteSpace(drive.Name))
                .ToList();

            return physicalDrives
                .OrderByDescending(drive => drive.ActivityPercent)
                .ThenByDescending(drive => drive.ReadBytesPerSecond +
drive.WriteBytesPerSecond)
                .ThenBy(drive => drive.Name)
                .Take(4)
                .ToArray();
        }

        private StorageMetrics ReadStorage(IHardware hardware)
        {
            var sensors = hardware.Sensors;
            var used = ReadValue(FindSensor(sensors, SensorType.Load, "Used
Space", "Used")) ?? 0;
            var activity = ReadValue(FindSensor(sensors, SensorType.Load,
"Total Activity", "Activity", "Active")) ??
                           ReadValue(FindSensor(sensors, SensorType.Load,
"Read Activity", "Write Activity")) ??
```

```
                                0;

            var temperature = ReadValue(FindSensor(sensors,
SensorType.Temperature, "Temperature", "Composite")) ??

                                    AverageSensors(sensors,
SensorType.Temperature);

            var read = ReadValue(FindSensor(sensors, SensorType.Throughput,
"Read Rate", "Read", "Read Throughput")) ?? 0;

            var write = ReadValue(FindSensor(sensors, SensorType.Throughput,
"Write Rate", "Write", "Write Throughput")) ?? 0;


            return new StorageMetrics

            {

                Name = hardware.Name,

                Kind = GuessDriveKind(hardware.Name),

                ActivityPercent = ClampPercent(activity > 0 ? activity :
ScaleActivity(read + write)),

                UsedPercent = ClampPercent(used),

                TemperatureCelsius = Math.Max(0, temperature),

                ReadBytesPerSecond = Math.Max(0, read),

                WriteBytesPerSecond = Math.Max(0, write)

            };

        }


        private static IReadOnlyList<NamedMetric>
ReadTemperatures(IEnumerable<IHardware> hardware)

        {

            return hardware

                .SelectMany(item => item.Sensors.Select(sensor => new {
Hardware = item, Sensor = sensor }))

                .Where(item => item.Sensor.SensorType ==
SensorType.Temperature && item.Sensor.Value.HasValue)

                .Where(item => item.Sensor.Value!.Value > 0)

                .OrderByDescending(item =>
Priority($"{item.Hardware.HardwareType} {item.Hardware.Name}
{item.Sensor.Name}", "CPU", "Package", "Core", "GPU", "Hot Spot", "Hotspot",
"Storage", "SSD", "NVMe", "HFM", "HDD", "WDC"))

                .Take(64)

                .Select(item => new
NamedMetric(BuildTemperatureName(item.Hardware, item.Sensor),
item.Sensor.Value!.Value, "C"))

                .ToArray();

        }


        private static IReadOnlyList<string>
ReadSensorDiagnostics(IEnumerable<IHardware> hardware)
```

```
        {
            return hardware
                .SelectMany(item => item.Sensors.Select(sensor => new { Hardware = item, Sensor = sensor }))
                .Where(item => item.Sensor.SensorType is SensorType.Temperature or SensorType.Power)
                .OrderBy(item => item.Hardware.HardwareType.ToString())
                .ThenBy(item => item.Hardware.Name)
                .ThenBy(item => item.Sensor.SensorType.ToString())
                .ThenBy(item => item.Sensor.Name)
                .Take(120)
                .Select(item => FormatSensorDiagnostic(item.Hardware, item.Sensor))
                .ToArray();
        }

        private static CpuMetrics ApplyCpuTemperatureFromLibreHardwareSensors(
            CpuMetrics cpu,
            IReadOnlyList<NamedMetric> allTemperatures)
        {
            if (cpu.TemperatureCelsius > 0)
            {
                return cpu;
            }

            var cpuTemperature = PickCpuTemperature(allTemperatures);
            if (cpuTemperature <= 0)
            {
                return cpu;
            }

            return new CpuMetrics
            {
                Name = cpu.Name,
                LoadPercent = cpu.LoadPercent,
                TemperatureCelsius = cpuTemperature,
                ClockMhz = cpu.ClockMhz,
                PowerWatts = cpu.PowerWatts,
                CoreLoads = cpu.CoreLoads
            };
```

```
        }


        private static IReadOnlyList<string> ReadCollectorDiagnostics()
        {
            var assemblyVersion =
typeof(Computer).Assembly.GetName().Version?.ToString() ?? "unknown";
            return new[]
            {
                $"App | process architecture:
{RuntimeInformation.ProcessArchitecture}",
                $"App | running as administrator: {IsAdministrator()}",
                $"App | current directory: {Environment.CurrentDirectory}",
                $"App | base directory: {AppContext.BaseDirectory}",
                $"LHM | assembly version: {assemblyVersion}",
                $"LHM | runtime assembly:
{typeof(Computer).Assembly.FullName ?? "n/a"}"
            };
        }


        private static double PickCpuTemperature(IEnumerable<NamedMetric>
temperatures)
        {
            return temperatures
                .Where(item => item.Value is > 0 and < 125)
                .Where(item => !ContainsAny(item.Name, "GPU", "Graphics",
"SSD", "HDD", "NVMe", "Drive", "Storage"))
                .OrderByDescending(item => Priority(item.Name, "CPU
Package", "CPU", "Package", "Core Max", "Core Average", "Core", "ACPI",
"WMI"))
                .Select(item => item.Value)
                .FirstOrDefault();
        }


        private static string FormatSensorDiagnostic(IHardware hardware,
ISensor sensor)
        {
            var valueText = sensor.Value.HasValue ?
$"{sensor.Value.Value:0.##}" : "n/a";
            var unit = sensor.SensorType switch
            {
                SensorType.Temperature => "\u00B0C",
                SensorType.Power => "W",
                _ => string.Empty
```

```
            };

            return $"{hardware.HardwareType} | {hardware.Name} |
{sensor.SensorType} | {sensor.Name}: {valueText} {unit}";
        }

        private IHardware? PickNetworkHardware(IEnumerable<IHardware>
hardware)
        {
            return hardware
                .Where(item => item.HardwareType == HardwareType.Network)
                .OrderByDescending(item => item.Sensors
                    .Where(sensor => sensor.SensorType ==
SensorType.Throughput)
                    .Sum(sensor => sensor.Value ?? 0))
                .FirstOrDefault();
        }

        private void UpdateHardwareTree()
        {
            _computer.Accept(new UpdateVisitor());
        }

        private static IEnumerable<IHardware>
FlattenHardware(IEnumerable<IHardware> hardware)
        {
            foreach (var item in hardware)
            {
                yield return item;

                foreach (var child in FlattenHardware(item.SubHardware))
                {
                    yield return child;
                }
            }
        }

        private static ISensor? FindSensor(IEnumerable<ISensor> sensors,
SensorType type, params string[] nameParts)
        {
            return sensors
```

```
                    .Where(sensor => sensor.SensorType == type &&
sensor.Value.HasValue)
                    .OrderByDescending(sensor => Priority(sensor.Name,
nameParts))
                    .FirstOrDefault(sensor => nameParts.Length == 0 ||
ContainsAny(sensor.Name, nameParts));
            }


            private static ISensor? FindSensorExcluding(IEnumerable<ISensor>
sensors, SensorType type, IReadOnlyList<string> excludedParts, params
string[] nameParts)
            {
                return sensors
                    .Where(sensor => sensor.SensorType == type &&
sensor.Value.HasValue)
                    .Where(sensor => !ContainsAny(sensor.Name,
excludedParts.ToArray()))
                    .OrderByDescending(sensor => Priority(sensor.Name,
nameParts))
                    .FirstOrDefault(sensor => nameParts.Length == 0 ||
ContainsAny(sensor.Name, nameParts));
            }


            private static double AverageSensors(IEnumerable<ISensor> sensors,
SensorType type, params string[] nameParts)
            {
                var values = sensors
                    .Where(sensor => sensor.SensorType == type &&
sensor.Value.HasValue)
                    .Where(sensor => nameParts.Length == 0 ||
ContainsAny(sensor.Name, nameParts))
                    .Select(sensor => (double)sensor.Value!.Value)
                    .Where(value => value > 0)
                    .ToArray();

                return values.Length == 0 ? 0 : values.Average();
            }


            private static double AverageSensorsExcluding(IEnumerable<ISensor>
sensors, SensorType type, IReadOnlyList<string> excludedParts, params
string[] nameParts)
            {
                var values = sensors
                    .Where(sensor => sensor.SensorType == type &&
sensor.Value.HasValue)
```

```
                    .Where(sensor => !ContainsAny(sensor.Name,
excludedParts.ToArray()))
                    .Where(sensor => nameParts.Length == 0 ||
ContainsAny(sensor.Name, nameParts))
                    .Select(sensor => (double)sensor.Value!.Value)
                    .Where(value => value > 0)
                    .ToArray();

                return values.Length == 0 ? 0 : values.Average();
            }

            private static double? ReadValue(ISensor? sensor)
            {
                return sensor?.Value is null ? null : sensor.Value.Value;
            }

            private IReadOnlyList<StorageMetrics> ReadLogicalDrives()
            {
                try
                {
                    return DriveInfo.GetDrives()
                        .Where(drive => drive.DriveType == DriveType.Fixed &&
drive.IsReady && drive.TotalSize > 0)
                        .Select(drive =>
                        {
                            var used = (double)(drive.TotalSize -
drive.AvailableFreeSpace) / drive.TotalSize * 100;
                            var instance = drive.Name.TrimEnd('\\');
                            var read = ReadLogicalDiskCounter(instance, "Disk
Read Bytes/sec");
                            var write = ReadLogicalDiskCounter(instance, "Disk
Write Bytes/sec");
                            var activity = ReadLogicalDiskCounter(instance, "%
Disk Time");
                            return new StorageMetrics
                            {
                                Name = $"{instance} {drive.VolumeLabel}".Trim(),
                                Kind = "Volume",
                                UsedPercent = ClampPercent(used),
                                ActivityPercent = ClampPercent(activity > 0.5 ?
activity : ScaleActivity(read + write)),
                                TemperatureCelsius = 0,
```

```
                            ReadBytesPerSecond = Math.Max(0, read),
                            WriteBytesPerSecond = Math.Max(0, write)
                        };
                    })
                    .ToArray();
            }
            catch
            {
                return Array.Empty<StorageMetrics>();
            }
        }

        private double ReadCurrentCpuClockMhz()
        {
            try
            {
                if (_cpuPerformanceCounter is null)
                {
                    return 0;
                }

                var performancePercent = _cpuPerformanceCounter.NextValue();
                if (performancePercent <= 0 || _baseCpuClockMhz <= 0)
                {
                    return 0;
                }

                return Math.Clamp(_baseCpuClockMhz * performancePercent /
100d, 0, 10_000);
            }
            catch
            {
                return 0;
            }
        }

        private double ReadLogicalDiskCounter(string instance, string
counterName)
        {
            try
```

```
            {
                var key = $"{instance}|{counterName}";
                if (!_logicalDiskCounters.TryGetValue(key, out var counter))
                {
                    counter = TryCreatePerformanceCounter("LogicalDisk",
counterName, instance);
                    if (counter is null)
                    {
                        return 0;
                    }

                    _logicalDiskCounters[key] = counter;
                }

                return counter.NextValue();
            }
            catch
            {
                return 0;
            }
        }

        private PerformanceCounter? TryCreatePerformanceCounter(string
categoryName, string counterName, string instanceName)
        {
            try
            {
                var counter = new PerformanceCounter(categoryName,
counterName, instanceName, true);
                counter.NextValue();
                _performanceCounters.Add(counter);
                return counter;
            }
            catch
            {
                return null;
            }
        }

        private static MemoryMetrics? ReadPhysicalMemoryFromWindows()
```

```
        {
            try
            {
                var status = new MemoryStatusEx();
                if (!GlobalMemoryStatusEx(status))
                {
                    return null;
                }

                var totalGb = status.TotalPhys / 1024d / 1024d / 1024d;
                var availableGb = status.AvailPhys / 1024d / 1024d / 1024d;
                var usedGb = Math.Max(0, totalGb - availableGb);
                return new MemoryMetrics
                {
                    LoadPercent = ClampPercent(status.MemoryLoad),
                    UsedGb = usedGb,
                    TotalGb = totalGb,
                    VirtualLoadPercent = 0,
                    VirtualUsedGb = 0,
                    VirtualTotalGb = 0
                };
            }
            catch
            {
                return null;
            }
        }

        [DllImport("kernel32.dll", SetLastError = true)]
        private static extern bool GlobalMemoryStatusEx([In, Out]
MemoryStatusEx lpBuffer);

        [StructLayout(LayoutKind.Sequential)]
        private sealed class MemoryStatusEx
        {
            public uint Length = (uint)Marshal.SizeOf<MemoryStatusEx>();
            public uint MemoryLoad;
            public ulong TotalPhys;
            public ulong AvailPhys;
            public ulong TotalPageFile;
```

```
            public ulong AvailPageFile;
            public ulong TotalVirtual;
            public ulong AvailVirtual;
            public ulong AvailExtendedVirtual;
        }

        private static bool NamesLookRelated(string left, string right)
        {
            var normalizedLeft = NormalizeDriveName(left);
            var normalizedRight = NormalizeDriveName(right);
            return normalizedLeft.Length > 0 &&
                   normalizedRight.Length > 0 &&
                   (normalizedLeft.Contains(normalizedRight,
StringComparison.OrdinalIgnoreCase) ||
                    normalizedRight.Contains(normalizedLeft,
StringComparison.OrdinalIgnoreCase));
        }

        private static string NormalizeDriveName(string value)
        {
            return new string(value
                .Where(char.IsLetterOrDigit)
                .Select(char.ToUpperInvariant)
                .ToArray());
        }

        private static string GuessDriveKind(string name)
        {
            if (ContainsAny(name, "NVMe", "SSD", "KINGSTON", "SAMSUNG",
"WD_BLACK", "SN"))
            {
                return "SSD";
            }

            if (ContainsAny(name, "HDD", "WDC", "ST", "TOSHIBA"))
            {
                return "HDD";
            }

            return "Drive";
```

```
        }

        private static double ReadCpuClockFromRegistry()
        {
            try
            {
                using var key =
Registry.LocalMachine.OpenSubKey(@"HARDWARE\DESCRIPTION\System\CentralProcess
or\0");
                return Convert.ToDouble(key?.GetValue("~MHz") ?? 0);
            }
            catch
            {
                return 0;
            }
        }

        private static bool ContainsAny(string value, params string[] parts)
        {
            return parts.Any(part => value.Contains(part,
StringComparison.OrdinalIgnoreCase));
        }

        private static int Priority(string value, params string[] parts)
        {
            var score = 0;
            for (var i = 0; i < parts.Length; i++)
            {
                if (value.Contains(parts[i],
StringComparison.OrdinalIgnoreCase))
                {
                    score += parts.Length - i;
                }
            }

            return score;
        }

        private static string NormalizeCoreName(string name, int index)
        {
            if (name.Contains("total", StringComparison.OrdinalIgnoreCase))
```

```
            {
                return "Total";
            }

            return name.Length > 18 ? $"Core {index + 1}" : name;
        }

        private static string ShortenName(string name)
        {
            return name
                .Replace("Temperature", "Temp",
StringComparison.OrdinalIgnoreCase)
                .Replace("Package", "Pkg",
StringComparison.OrdinalIgnoreCase);
        }

        private static string BuildTemperatureName(IHardware hardware,
ISensor sensor)
        {
            if (hardware.HardwareType == HardwareType.Storage)
            {
                return $"{GuessDriveKind(hardware.Name)}
{ShortenStorageName(hardware.Name)}";
            }

            var prefix = hardware.HardwareType switch
            {
                HardwareType.Cpu => "CPU",
                HardwareType.GpuAmd or HardwareType.GpuIntel or
HardwareType.GpuNvidia => "GPU",
                HardwareType.Motherboard => "Board",
                _ => string.Empty
            };

            var sensorName = ShortenName(sensor.Name);
            return string.IsNullOrWhiteSpace(prefix) ||
sensorName.Contains(prefix, StringComparison.OrdinalIgnoreCase)
                ? sensorName
                : $"{prefix} {sensorName}";
        }

        private static string ShortenStorageName(string name)
```

```
          {
              var compact = name
                  .Replace("KINGSTON", "KNG",
StringComparison.OrdinalIgnoreCase)
                  .Replace("Samsung", "SAMS",
StringComparison.OrdinalIgnoreCase)
                  .Replace("Western Digital", "WD",
StringComparison.OrdinalIgnoreCase)
                  .Replace("WDC", "WD", StringComparison.OrdinalIgnoreCase)
                  .Replace("Technology", "",
StringComparison.OrdinalIgnoreCase)
                  .Trim();

              return compact.Length > 12 ? compact[..12] : compact;
          }

          private static double ClampPercent(double value) =>
Math.Clamp(value, 0, 100);

          private static double ScaleActivity(double bytesPerSecond)
          {
              return bytesPerSecond <= 0 ? 0 :
Math.Clamp(Math.Log10(bytesPerSecond + 1) / 8 * 100, 0, 100);
          }

          private static bool IsAdministrator()
          {
              try
              {
                  using var identity = WindowsIdentity.GetCurrent();
                  var principal = new WindowsPrincipal(identity);
                  return principal.IsInRole(WindowsBuiltInRole.Administrator);
              }
              catch
              {
                  return false;
              }
          }

          private sealed class UpdateVisitor : IVisitor
          {
              public void VisitComputer(IComputer computer)
```

```
        {
            computer.Traverse(this);
        }

        public void VisitHardware(IHardware hardware)
        {
            hardware.Update();
            foreach (var subHardware in hardware.SubHardware)
            {
                subHardware.Accept(this);
            }
        }

        public void VisitSensor(ISensor sensor)
        {
        }

        public void VisitParameter(IParameter parameter)
        {
        }
    }
}
```

Додаток В

Лістинг коду програми для STM32

```
static void HM10_StartReceive(void)
{
  if (HAL_UART_Receive_IT(&huart6, &hm10_rx_byte, 1U) == HAL_OK)
  {
    return;
  }

  hm10_last_error_tick = HAL_GetTick();
}

static void HM10_PushRxByte(uint8_t byte)
{
  uint16_t next_head = (uint16_t)((hm10_rx_head + 1U) %
HM10_RX_BUFFER_SIZE);

  HM10_RecordDebugByte(byte);

  if (next_head == hm10_rx_tail)
  {
    hm10_rx_overflows++;
    hm10_last_error_tick = HAL_GetTick();
    return;
  }

  hm10_rx_buffer[hm10_rx_head] = byte;
  hm10_rx_head = next_head;
}

static uint8_t HM10_PopRxByte(uint8_t *byte)
{
  if (hm10_rx_tail == hm10_rx_head)
  {
    return 0U;
  }

  *byte = hm10_rx_buffer[hm10_rx_tail];
  hm10_rx_tail = (uint16_t)((hm10_rx_tail + 1U) % HM10_RX_BUFFER_SIZE);
```

```
  return 1U;
}

static void HM10_RecordDebugByte(uint8_t byte)
{
  hm10_debug_recent_bytes[hm10_debug_recent_index] = byte;
  hm10_debug_recent_index = (uint8_t)((hm10_debug_recent_index + 1U) %
HM10_DEBUG_RECENT_BYTES);
  if (hm10_debug_recent_count < HM10_DEBUG_RECENT_BYTES)
  {
    hm10_debug_recent_count++;
  }

  if (byte == PM_PROTOCOL_MAGIC0)
  {
    hm10_magic0_hits++;
  }
  if (hm10_debug_previous_byte == PM_PROTOCOL_MAGIC0 && byte ==
PM_PROTOCOL_MAGIC1)
  {
    hm10_magic_pair_hits++;
  }
  hm10_debug_previous_byte = byte;
}

static void HM10_CopyRecentBytes(uint8_t *destination, uint8_t *count)
{
  uint8_t i;
  uint8_t start;

  *count = hm10_debug_recent_count;
  if (*count == 0U)
  {
    return;
  }

  start = (uint8_t)((hm10_debug_recent_index + HM10_DEBUG_RECENT_BYTES -
*count) % HM10_DEBUG_RECENT_BYTES);
  for (i = 0U; i < *count; i++)
  {
```

```
          destination[i] = hm10_debug_recent_bytes[(uint8_t)((start + i) %
HM10_DEBUG_RECENT_BYTES)];
        }
      }

      static void HM10_ProcessReceivedBytes(void)
      {
        uint8_t byte;
        PcMonitorParseResult result;

        while (HM10_PopRxByte(&byte) != 0U)
        {
          result = PcMonitorProtocol_ProcessByte(&hm10_parser, byte);
          HM10_HandleParseResult(result);
        }
      }

      static void HM10_HandleParseResult(PcMonitorParseResult result)
      {
        const PcMonitorFrame *frame;

        switch (result)
        {
          case PM_PARSE_FRAME_READY:
            frame = PcMonitorProtocol_GetFrame(&hm10_parser);
            HM10_HandleFrame(frame);
            break;

          case PM_PARSE_BAD_VERSION:
          case PM_PARSE_PAYLOAD_TOO_LARGE:
          case PM_PARSE_CRC_ERROR:
            hm10_last_error_tick = HAL_GetTick();
            break;

          case PM_PARSE_NONE:
          default:
            break;
        }
      }
```

```
static void HM10_HandleFrame(const PcMonitorFrame *frame)
{
  if (frame->type == PM_PACKET_TELEMETRY)
  {
    hm10_last_frame = *frame;
    hm10_has_frame = 1U;
    if (PcMonitorProtocol_DecodeTelemetry(frame, &latest_telemetry))
    {
      hm10_telemetry_packets++;
      hm10_has_telemetry = 1U;
      hm10_last_packet_tick = HAL_GetTick();
    }
    else
    {
      hm10_decode_errors++;
      hm10_last_error_tick = HAL_GetTick();
    }
    return;
  }

  hm10_last_frame = *frame;
  hm10_has_frame = 1U;
  hm10_command_packets++;
  hm10_last_packet_tick = HAL_GetTick();

  if (frame->type == PM_PACKET_DEVICE_SETTINGS)
  {
    HM10_HandleDeviceSettings(frame);
  }
  else if (frame->type == PM_PACKET_SLEEP_NOW)
  {
    MainLcd_SleepNow();
  }
  else if (frame->type == PM_PACKET_WAKE)
  {
    MainLcd_Wake();
  }
}
```

```
static void HM10_HandleDeviceSettings(const PcMonitorFrame *frame)
{
  uint8_t settings_version;
  uint8_t brightness_percent;
  uint16_t sleep_timeout_seconds;

  if (frame == NULL || frame->payload_length < 4U)
  {
    hm10_decode_errors++;
    hm10_last_error_tick = HAL_GetTick();
    return;
  }

  settings_version = frame->payload[0];
  if (settings_version != 1U)
  {
    hm10_decode_errors++;
    hm10_last_error_tick = HAL_GetTick();
    return;
  }

  brightness_percent = frame->payload[1];
  sleep_timeout_seconds = (uint16_t)frame->payload[2] |
((uint16_t)frame->payload[3] << 8);
  MainLcd_ApplySettings(brightness_percent, sleep_timeout_seconds);
}

static void App_UpdateStatusLeds(void)
{
  uint32_t now = HAL_GetTick();
  uint8_t packet_active = (uint8_t)((now - hm10_last_packet_tick) <
HM10_PACKET_LED_MS);
  uint8_t error_active = (uint8_t)((now - hm10_last_error_tick) <
HM10_ERROR_LED_MS);

  HAL_GPIO_WritePin(GPIOJ, LD_USER1_Pin, packet_active ? GPIO_PIN_SET :
GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOJ, LD_USER2_Pin, error_active ? GPIO_PIN_SET :
GPIO_PIN_RESET);
}
```

```
static void App_UpdateMainLcd(void)
{
  const PcMonitorProtocolStats *stats;
  uint32_t now = HAL_GetTick();
  uint32_t protocol_errors;
  uint8_t recent_bytes[HM10_DEBUG_RECENT_BYTES];
  uint8_t recent_byte_count = 0U;

  if ((now - main_lcd_last_render_tick) < MAIN_LCD_RENDER_INTERVAL_MS)
  {
    return;
  }

  stats = PcMonitorProtocol_GetStats(&hm10_parser);
  HM10_CopyRecentBytes(recent_bytes, &recent_byte_count);
  protocol_errors = stats->crc_errors + stats->version_errors + stats->oversize_errors;
  (void)protocol_errors;
  if (hm10_has_telemetry != 0U || MainLcd_ShouldShowRxDebug() == 0U)
  {
    MainLcd_RenderTelemetry(&latest_telemetry,
                            hm10_telemetry_packets,
                            hm10_command_packets,
                            protocol_errors,
                            hm10_rx_overflows,
                            hm10_has_telemetry);
  }
  else
  {
    MainLcd_RenderRxDebug(stats,
                          &hm10_last_frame,
                          recent_bytes,
                          recent_byte_count,
                          hm10_telemetry_packets,
                          hm10_command_packets,
                          hm10_decode_errors,
                          hm10_rx_overflows,
                          hm10_magic0_hits,
                          hm10_magic_pair_hits,
                          hm10_has_frame,
```

```
                            hm10_has_telemetry);
  }
  main_lcd_last_render_tick = now;
}

static void App_UpdateButtons(void)
{
  uint8_t events;

  __disable_irq();
  events = button_event_flags;
  button_event_flags = 0U;
  __enable_irq();

  if ((events & BUTTON_EVENT_PREVIOUS_PAGE) != 0U)
  {
    MainLcd_PreviousPage();
  }
  if ((events & BUTTON_EVENT_NEXT_PAGE) != 0U)
  {
    MainLcd_NextPage();
  }
  if ((events & BUTTON_EVENT_OVERVIEW_PAGE) != 0U)
  {
    MainLcd_ShowOverview();
  }
  if ((events & BUTTON_EVENT_SLEEP_TOGGLE) != 0U)
  {
    MainLcd_ToggleSleep();
  }
}

static void App_ScanButtonsFast(void)
{
  uint8_t states;
  uint8_t falling_edges;
  uint32_t now;

  if (buttons_scan_ready == 0U)
```

```
        {
          return;
        }

        states = 0U;
        states |= HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4) == GPIO_PIN_RESET ? 0U :
0x01U;
        states |= HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_2) == GPIO_PIN_RESET ? 0U :
0x02U;
        states |= HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_10) == GPIO_PIN_RESET ? 0U
: 0x04U;
        states |= HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_8) == GPIO_PIN_RESET ? 0U :
0x08U;

        falling_edges = (uint8_t)(button_sample_previous_states &
(uint8_t)(~states) & 0x0FU);
        button_sample_previous_states = states;
        if (falling_edges == 0U)
        {
          return;
        }

        now = HAL_GetTick();
        if ((falling_edges & 0x01U) != 0U && (now -
button_last_press_ticks[0]) >= BUTTON_DEBOUNCE_MS)
        {
          button_event_flags |= BUTTON_EVENT_PREVIOUS_PAGE;
          button_last_press_ticks[0] = now;
        }
        if ((falling_edges & 0x02U) != 0U && (now -
button_last_press_ticks[1]) >= BUTTON_DEBOUNCE_MS)
        {
          button_event_flags |= BUTTON_EVENT_NEXT_PAGE;
          button_last_press_ticks[1] = now;
        }
        if ((falling_edges & 0x04U) != 0U && (now -
button_last_press_ticks[2]) >= BUTTON_DEBOUNCE_MS)
        {
          button_event_flags |= BUTTON_EVENT_OVERVIEW_PAGE;
          button_last_press_ticks[2] = now;
        }
        if ((falling_edges & 0x08U) != 0U && (now -
button_last_press_ticks[3]) >= BUTTON_DEBOUNCE_MS)
```

```
    {
      button_event_flags |= BUTTON_EVENT_SLEEP_TOGGLE;
      button_last_press_ticks[3] = now;
    }
  }

  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
  {
    if (huart->Instance == USART6)
    {
      HM10_PushRxByte(hm10_rx_byte);
      HM10_StartReceive();
    }
  }

  void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
  {
    if (huart->Instance == USART6)
    {
      hm10_last_error_tick = HAL_GetTick();
      HM10_StartReceive();
    }
  }

  /* USER CODE END 0 */

  /**
    * @brief  The application entry point.
    * @retval int
    */
  int main(void)
  {

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration--------------------------------------------------------
------*/
```

```
        /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
        HAL_Init();

        /* USER CODE BEGIN Init */

        /* USER CODE END Init */

        /* Configure the system clock */
        SystemClock_Config();

        /* Configure the peripherals common clocks */
        PeriphCommonClock_Config();

        /* USER CODE BEGIN SysInit */

        /* USER CODE END SysInit */

        /* Initialize all configured peripherals */
        MX_GPIO_Init();
        MX_ADC1_Init();
        MX_CRC_Init();
        MX_DMA2D_Init();
        MX_DSIHOST_DSI_Init();
        MX_FMC_Init();
        MX_HDMI_CEC_Init();
        MX_I2C1_Init();
        MX_I2C4_Init();
        MX_LTDC_Init();
        MX_RTC_Init();
        MX_SAI2_Init();
        MX_SPDIFRX_Init();
        MX_SPI2_Init();
        MX_TIM1_Init();
        MX_TIM3_Init();
        MX_TIM10_Init();
        MX_TIM11_Init();
        MX_TIM12_Init();
        MX_UART5_Init();
        MX_USART1_UART_Init();
```

```
  MX_USART6_UART_Init();
  MX_USB_OTG_HS_PCD_Init();
  /* USER CODE BEGIN 2 */
  MainLcd_Init(&hdsi);
  PcMonitorProtocol_Init(&hm10_parser);
  HM10_StartReceive();
  buttons_scan_ready = 1U;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HM10_ProcessReceivedBytes();
      App_UpdateButtons();
      MainLcd_Process();
      App_UpdateMainLcd();
      App_UpdateStatusLeds();

      HAL_Delay(10);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();

  /** Configure the main internal regulator output voltage
  */
```

```
        __HAL_RCC_PWR_CLK_ENABLE();
        __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

        /** Initializes the RCC Oscillators according to the specified
parameters
        * in the RCC_OscInitTypeDef structure.
        */
        RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI
                                    |RCC_OSCILLATORTYPE_HSE;
        RCC_OscInitStruct.HSEState = RCC_HSE_ON;
        RCC_OscInitStruct.HSIState = RCC_HSI_ON;
        RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
        RCC_OscInitStruct.LSIState = RCC_LSI_ON;
        RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
        RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
        RCC_OscInitStruct.PLL.PLLM = 25;
        RCC_OscInitStruct.PLL.PLLN = 432;
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
        RCC_OscInitStruct.PLL.PLLQ = 4;
        RCC_OscInitStruct.PLL.PLLR = 2;
        if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
        {
          Error_Handler();
        }

        /** Activate the Over-Drive mode
        */
        if (HAL_PWREx_EnableOverDrive() != HAL_OK)
        {
          Error_Handler();
        }

        /** Initializes the CPU, AHB and APB buses clocks
        */
        RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
        RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
        RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
        RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
```

```
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) !=
HAL_OK)
  {
    Error_Handler();
  }
  HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_HSI, RCC_MCODIV_1);
}

/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection =
RCC_PERIPHCLK_LTDC|RCC_PERIPHCLK_SAI2;
  PeriphClkInitStruct.PLLSAI.PLLSAIN = 192;
  PeriphClkInitStruct.PLLSAI.PLLSAIR = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIQ = 3;
  PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV4;
  PeriphClkInitStruct.PLLSAIDivQ = 1;
  PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
  PeriphClkInitStruct.Sai2ClockSelection = RCC_SAI2CLKSOURCE_PLLSAI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Додаток Г

Перелік елементів до схеми електричної принципової

Поз. позначення	*Найменування*	*Кіл.*	*Примітка*
	Мікросхеми		
U1	Плата розробника STM32F769I-DISCO	1	
	Модулі		
U2	Bluetooth-модуль HM-10 CC2541	1	
	Кнопки		
S1-S4	Omron B3F Series Tactile Switch	4	

					КС КРБ 123.141.00.01 ПЕ	
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		*Балаєв В. А.*			*Система моніторингу апаратно-програмних компонентів комп'ютерної техніки*	*Літ.* / *Арк.* / *Аркушів*
Перевір.		*Лещишин Ю. З.*				*1*
Реценз.		*Дмитроца Л. П.*				*ТНТУ, каф. КС, гр. СІ-41*
Н. Контр.		*Луцик Н. С.*				
Затверд.		*Осухівська Г.М.*			*Перелік елементів*	