Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «**АЛГОРИТМІЧНА МОДЕЛЬ УПРАВЛІННЯ ЗОВНІШНІМИ ОБ'ЄКТАМИ ЦИФРОВОГО СИГНАЛЬНОГО ПРОЦЕСОРА ЗА ДОПОМОГОЮ USB-ІНТЕРФЕЙСУ**»

Спеціальність 123 – Комп'ютерна інженерія.
Галузь знань: 12 – Інформаційні технології.
Освітня програма: «Комп'ютерна інженерія».

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КІ–41
БОРОДІН Єгор Владиславович

Керівник: к.т.н., доцент кафедри
комп'ютерних систем та робототехніки
РЕВА Сергій Миколайович

Рецензент: к.т.н., ст. викл. кафедри
інтелектуальних програмних систем і
технологій
ЗІНОВ'ЄВ Дмитро Володимирович

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 92 сторінки, із яких 50 сторінок основної частини з 21 рисунком, 4 таблицями, 39 найменуваннями списку використаних джерел та чотирма додатками.

Метою роботи є розробка і практична реалізація алгоритмічної моделі управління зовнішніми об’єктами, підключеними до цифрового сигнального процесора (реалізованого на базі мікроконтролера ATmega328P), за допомогою USB-інтерфейсу, що передбачає створення спеціалізованого протоколу обміну та відповідного програмного забезпечення.

Досягнення мети здійснювалося шляхом аналізу архітектури цифрового сигнального процесора та USB-інтерфейсу, огляду існуючих рішень, проєктування багаторівневої алгоритмічної моделі управління та спеціалізованого протоколу обміну даними. Було реалізовано програмні компоненти: управляючу програму для мікроконтролера ATmega328P на платформі Arduino Uno (з використанням USB-UART перетворювача CH340C) та керуючу програму з консольним інтерфейсом для ПК. Експериментальна перевірка працездатності системи проводилася на тестовому стенді з використанням модуля світлодіодів HW-479, датчика температури та вологості KY-015, а аналіз USB-трафіку здійснювався за допомогою Wireshark та USBPcap.

Результатом роботи є діюча алгоритмічна модель та система управління, що включає спеціалізований, простий та розширюваний протокол пакетного обміну даними між ПК та мікроконтролером.

Ключові слова: ЦИФРОВИЙ СИГНАЛЬНИЙ ПРОЦЕСОР, USB-ІНТЕРФЕЙС, ПРОТОКОЛ ОБМІНУ ДАНИМИ, USB-UART, МІКРОКОНТРОЛЕР, UART-ПРОТОКОЛ.

ABSTRACT

The explanatory note to the bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references and four appendices. The total volume of the work is 92 pages, including 50 pages of the main part with 21 figures, 4 tables, 39 references and four appendices.

The aim of the work is the development and practical implementation of an algorithmic model for controlling external devices connected to a digital signal processor (implemented on the ATmega328P microcontroller) via a USB interface, which involves the creation of a specialized communication protocol and corresponding software.

The achievement of this aim was carried out by analyzing the architectures of the digital signal processor and the USB interface, reviewing existing solutions, designing a multi-level algorithmic control model, and a specialized data exchange protocol. Software components were implemented: a control program for the ATmega328P microcontroller on the Arduino Uno platform (using a CH340C USB-UART converter) and a control program with a console interface for a PC. Experimental verification of the system's operability was conducted on a test bench using an HW-479 LED module and a KY-015 temperature and humidity sensor, and USB traffic analysis was performed using Wireshark and USBPcap.

The result of the work is a functioning algorithmic model and control system that includes a specialized, simple, and extensible packet data exchange protocol between a PC and a microcontroller.

Keywords: DIGITAL SIGNAL PROCESSOR, USB INTERFACE, DATA EXCHANGE PROTOCOL, USB-UART, MICROCONTROLLER, UART PROTOCOL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

АЦП – Аналого-цифровий перетворювач;

ПК – Персональний комп’ютер;

ЦАП – Цифро-аналоговий перетворювач;

ЦСП – Цифровий сигнальний процесор;

ALU – Арифметико-логічний пристрій (Arithmetic Logic Unit);

AVR – Родина восьмибітових мікроконтролерів фірми Atmel.;

CRC – Циклічний надлишковий код (Cyclic Redundancy Check);

EEPROM – Тип енергонезалежної пам’яті для зберігання відносно невеликих обсягів даних, яка дозволяє стирати та перепрограмовувати окремі байти (Electrically Erasable Programmable Read-Only Memory);

MAC – Множення з накопиченням (Multiply-Accumulate);

MIPS – величина, що показує кількість мільйонів інструкцій, які виконує процесор за одну секунду (Million Instructions Per Second);

NRZI – Кодування без повернення до нуля з інверсією (Non Return to Zero Invertive);

PID (в USB) – Ідентифікатора Пакету(Packet Identifier);

RISC-архітектура – Архітектурний підхід до проєктування процесорів, в якій швидкодія збільшується за рахунок такого кодування інструкцій, щоб їхнє декодування було простішим, а час виконання – меншим (Reduced Instruction Set Computer);

SIMD – Одна інструкція, багато даних (Single Instruction, Multiple Data);

UART (УАПП) – Універсальний асинхронний прийомо-передавач (Universal asynchronous receiver/transmitter);

URB (в USB) – Блок запиту USB (USB Request Block);

USB – Універсальна послідовна шина (Universal serial bus);

USB-IF – некомерційна організація, створена для просування та підтримки USB (USB Implementers Forum).

ВСТУП

У сучасному світі цифрові сигнальні процесори (ЦСП) та мікроконтролери, що виконують їх функції, відіграють ключову роль у широкому спектрі систем керування, обробки даних та взаємодії з фізичним середовищем. Їх здатність ефективно обробляти сигнали та керувати зовнішніми периферійними пристроями, такими як датчики та виконавчі механізми, робить їх незамінними компонентами в промисловій автоматизації, робототехніці, медичному обладнанні, побутовій електроніці та багатьох інших галузях. Важливою задачею при розробці таких систем є забезпечення надійного та гнучкого каналу зв'язку між ЦСП/мікроконтролером та хост-комп'ютером для конфігурації, моніторингу та безпосереднього управління. Інтерфейс USB (Universal Serial Bus) зарекомендував себе як один із найпоширеніших та найзручніших стандартів для такого зв'язку завдяки своїй універсальності, достатній пропускній здатності для багатьох задач керування та підтримці живлення пристроїв.

Аналіз вітчизняної та зарубіжної науково-технічної літератури свідчить про наявність різноманітних підходів до реалізації взаємодії між ЦСП/мікроконтролерами та ПК через USB. Існуючі рішення включають використання мікроконтролерів із вбудованим апаратним USB-контролером, програмну реалізацію USB-протоколу на мікроконтролерах без апаратної підтримки, а також широке застосування зовнішніх мікросхем-мостів, зокрема USB-UART перетворювачів. Останній підхід є особливо популярним для мікроконтролерів, які не мають вбудованого USB, але оснащені стандартними послідовними інтерфейсами, такими як UART, завдяки простоті інтеграції та доступності компонентів. Такі рішення дозволяють операційній системі ПК розпізнавати підключений пристрій як віртуальний COM-порт, спрощуючи розробку програмного забезпечення на стороні хоста.

Незважаючи на наявність апаратних та базових програмних засобів для встановлення зв'язку, часто проблемою є відсутність уніфікованої

алгоритмічної моделі та спеціалізованого, але водночас гнучкого протоколу обміну даними, орієнтованого на ефективне управління набором різнорідних зовнішніх об'єктів. Розробники часто змушені щоразу створювати власні протоколи та логіку обробки команд і даних, що збільшує час розробки, знижує універсальність рішень та ускладнює їх повторне використання чи модифікацію.

У даній роботі пропонується розробка комплексної алгоритмічної моделі управління зовнішніми об'єктами, підключеними до мікроконтролера (що виконує функції, аналогічні ЦСП), за допомогою USB-інтерфейсу, реалізованого через міст USB-UART. Основна ідея полягає у створенні чітко структурованої системи взаємодії, що включає спеціалізований протокол обміну даними, програмні компоненти для мікроконтролера (управляюча програма) та для хост-комп'ютера (керуюча програма). Сенс роботи полягає в тому, щоб надати розробникам систематизований підхід та готовий до адаптації інструментарій для швидкої та ефективної організації управління зовнішніми пристроями, мінімізуючи необхідність розробки низькорівневих аспектів комунікації.

Актуальність даної роботи зумовлена зростаючою потребою в інтелектуальних системах керування, де мікроконтролери взаємодіють з персональними комп'ютерами для виконання складних завдань моніторингу та контролю. В умовах, коли швидкість розробки та адаптивність рішень відіграють ключову роль, наявність добре продуманої алгоритмічної моделі та стандартизованого (в рамках моделі) протоколу є критично важливою. Необхідність вивчення даної теми та проведення дослідницької роботи полягає у заповненні прогалини між базовими можливостями USB-UART зв'язку та потребами прикладних задач управління, пропонуючи більш високорівневе та системне рішення. Наукова новизна роботи полягає у розробці специфічної алгоритмічної моделі та протоколу обміну даними, оптимізованих для управління зовнішніми об'єктами мікроконтролера (на прикладі ATmega328P з мостом CH340C) через USB-інтерфейс.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі дослідження

Об’єкт дослідження

Об’єктом дослідження є процеси та взаємодії, пов’язані з управлінням зовнішніми периферійними об’єктами (такими як датчики та виконавчі механізми), підключеними до системи на базі мікроконтролера (ATmega328p, що виконує функції, подібні до ЦСП), з хост-комп’ютера.

Предмет дослідження

Предметом дослідження є алгоритмічна модель, спеціальний протокол зв’язку та пов’язані з ними програмні та апаратні компоненти, що забезпечують керування та обмін даними через USB-інтерфейс, зокрема з використанням моста USB-UART (CH340C).

Мета дослідження

Метою дослідження є розробка і практична реалізація алгоритмічної моделі управління зовнішніми об’єктами, підключеними до цифрового сигнального процесора, за допомогою USB-інтерфейсу.

Завдання дослідження

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести комплексний аналіз архітектури та принципів роботи цифрових сигнальних процесорів, включаючи оцінку придатності ATmega328p для виконання завдань керування, пов’язаних з ЦСП;
2. Виконати ретельне дослідження USB-інтерфейсу, охоплюючи його специфікації, протоколи передачі даних та відповідні стандарти, включаючи роль перетворювачів USB-UART;
3. Здійснити огляд та порівняння існуючих рішень та методологій для реалізації обміну даними між ЦСП та зовнішніми хост-системами через USB-інтерфейс;

4. Спроєктувати нову алгоритмічну модель для управління зовнішніми периферійними пристроями, керованими ATmega328p, через комунікаційний канал, опосередкований USB. Це включає визначення власного протоколу зв’язку;
5. Обрати та обґрунтувати вибір апаратних та програмних драйверних компонентів, необхідних для взаємодії між ATmega328p (через UART) та USB-інтерфейсом (через CH340C) як на стороні пристрою, так і на стороні хост-ПК;
6. Розробити та детально описати організацію механізмів обміну даними в рамках запропонованої моделі, охоплюючи передачу команд від ПК до ATmega328p та отримання даних/статусів від ATmega328p до ПК;
7. Реалізувати програмні компоненти алгоритмічної моделі, включаючи прошивку для ATmega328p (обробка UART, реалізація протоколу, керування пристроями) та прикладну програму для хост-ПК (USB-серійний зв’язок, реалізація протоколу, користувацький інтерфейс);
8. Провести систематичне тестування та налагодження реалізованої системи, з подальшою оптимізацією продуктивності та аналізом, оцінюючи її ефективність та надійність;
9. Розробити вичерпну документацію, включаючи інструкції користувача для реалізованої системи та технічні деталі розробленого драйвера та протоколу.

1.2 Аналіз предметної області та огляд існуючих рішень

1.2.1 Аналіз архітектури та принципів роботи ЦСП

Цифрові сигнальні процесори (ЦСП) є спеціалізованими мікропроцесорами, архітектура яких оптимізована для швидкого та ефективного виконання математичних операцій, характерних для обробки цифрових сигналів. Їх основне призначення – обробка сигналів у реальному часі, що включає фільтрацію, аналіз спектру (наприклад, швидке перетворення Фур’є), згортку, кореляцію, а також стиснення та декомпресію даних. ЦСП

знаходять широке застосування в аудіо та відео обробці, телекомунікаціях, системах керування, медичній візуалізації та багатьох інших галузях, де потрібна високопродуктивна обробка оцифрованих аналогових сигналів. На відміну від мікроконтролерів загального призначення, де обчислювальне ядро доповнюється периферійними пристроями, ЦСП часто будуються навколо ядра, оптимізованого для математичних операцій, до якого додаються швидкі АЦП та ЦАП [1; 2; 3].

Ефективність ЦСП зумовлена низкою архітектурних особливостей:

- Архітектура пам’яті: Більшість ЦСП використовують Гарвардську архітектуру або її модифіковані варіанти, які передбачають наявність окремих просторів пам’яті та шин для команд і даних. Це дозволяє одночасно вибирати інструкцію та операнди, значно підвищуючи пропускну здатність. Часто ЦСП мають багатопортову пам’ять на кристалі, що дозволяє одночасно отримувати доступ до кількох операндів [1; 2; 4];
- Спеціалізовані обчислювальні блоки:
    - MAC пристрій: Центральним елементом багатьох ЦСП є апаратний блок множення з накопиченням. Цей блок виконує операцію множення двох чисел та додавання результату до акумулятора за один тактовий цикл, що значно прискорює виконання алгоритмів ЦОС [1; 2];
    - ALU: Арифметико-логічний пристрій в ЦСП часто має розширену розрядність для запобігання переповненню [1; 2];
- Конвеєризація (Pipelining): ЦСП широко використовують глибоку конвеєризацію для збільшення пропускної здатності інструкцій. Розбиття виконання інструкції на кілька стадій (вибірка, декодування, вибірка операндів, виконання, запис результату) дозволяє одночасно обробляти кілька інструкцій на різних стадіях [2];
- Спеціалізований набір інструкцій: Набір команд ЦСП оптимізований для типових операцій цифрової обробки сигналів, включаючи

інструкції MAC, SIMD, інструкції для роботи з циклічними буферами та біт-реверсною адресацією, а також інструкції з нульовим або мінімальним часом виконання циклів [2].

У випадках коли не потрібна надзвичайно висока продуктивність спеціалізованих ЦСП, або де вартість та енергоспоживання не є критичними, можна використовувати сучасні мікроконтролери загального призначення, які вже стали достатньо швидкими для виконання функцій ЦСП.

1.2.2 Дослідження USB-інтерфейсу

Специфікації та стандарти USB

Універсальна послідовна шина (USB) є промисловим стандартом, який визначає кабелі, роз'єми та протоколи зв'язку, що використовуються для з'єднання, обміну даними та живлення між комп'ютерами та електронними пристроями [5; 6].

Специфікації та стандарти USB, які визначає некомерційна організація USB Implementers Forum (USB-IF):

- USB 1.x: Запровадив швидкості Low Speed (1.5 Мбіт/с) та Full Speed (12 Мбіт/с). Цього цілком достатньо для багатьох пристроїв керування та збору даних, таких як клавіатури або миші;
- USB 2.0: Додав High Speed (480 Мбіт/с), зберігаючи зворотну сумісність з USB 1.x;
- USB 3.x (Gen 1, Gen 2, Gen 2x2): Запровадив SuperSpeed (5 Гбіт/с), SuperSpeed+ (10 Гбіт/с, 20 Гбіт/с), значно збільшивши пропускну здатність;
- USB 4: Остання версія, що базується на специфікації Thunderbolt, пропонує швидкості до 40 Гбіт/с, 80 Гбіт/с і асиметричні 120 Гбіт/с, підтримуючи тунелювання DisplayPort та PCIe [5; 6].

Архітектура та топологія USB

USB використовує ієрархічну зіркоподібну топологію з одним хостом (зазвичай ПК) у центрі та кількома підключеними пристроями.

- Хост (Host): Керує всією шиною, ініціює всі передачі даних;
- Пристрій (Device): Периферійний пристрій, що підключається до хоста. Пристрої не можуть ініціювати зв'язок між собою напряму, тільки через хост;
- Хаби (Hub): Дозволяють розширити кількість портів USB [6].

Коли пристрій підключається до хоста, відбувається процес енумерації. Під час енумерації хост запитує у пристрою інформацію (дескриптори) для його ідентифікації, визначення його можливостей та завантаження відповідного драйвера [5; 6].

Протоколи передачі даних USB

Протокол передачі даних USB визначає, як дані структурують, передають та підтверджують між хостом та пристроєм. Важливо пам'ятати, що вся комунікація по шині USB ініціюється хостом. Пристрій не може самостійно надсилати дані; він завжди чекає запиту від хоста (за винятком спеціальних випадків, таких як "Remote Wakeup") [6].

Рівні протоколу:

- Фізичний рівень: Електричні сигнали на лініях D+ та D-, кодування даних – Non Return to Zero Invertive (NRZI) з вставкою бітів;
- Протокольний рівень: Визначає структуру пакетів, типи транзакцій та передач;
- Програмний рівень: Драйвери на хості та прошивка на пристрої, що реалізують логіку обміну [5; 6].

При кодуванні даних NRZI логічний "0", представляється інверсією рівня сигналу на лініях даних (D+ та D-), а логічна "1" представляється відсутністю зміни рівня сигналу. При кодуванні з вставкою бітів у масиві даних після шести однакових "1" додається один логічний "0". Це призведе до

примусової зміни рівня сигналу при NRZI-кодуванні, забезпечуючи перехід, необхідний приймачу для синхронізації [5; 6].

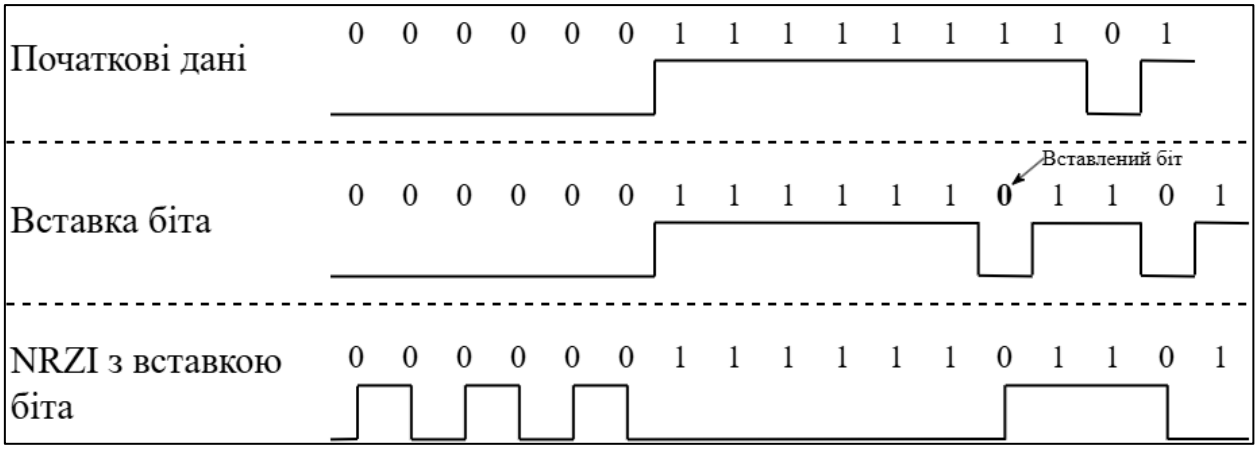

Рисунок 1.1 – Принцип кодування даних з використанням NRZI з вставкою бітів.

Дані по USB передаються у вигляді пакетів (Packets). Пакети об’єднуються у транзакції (Transactions), декілька транзакцій формують передачі (Transfer), а передачі з Start-of-Frame пакетом формують фрейми (Frames) [5; 6; 7].

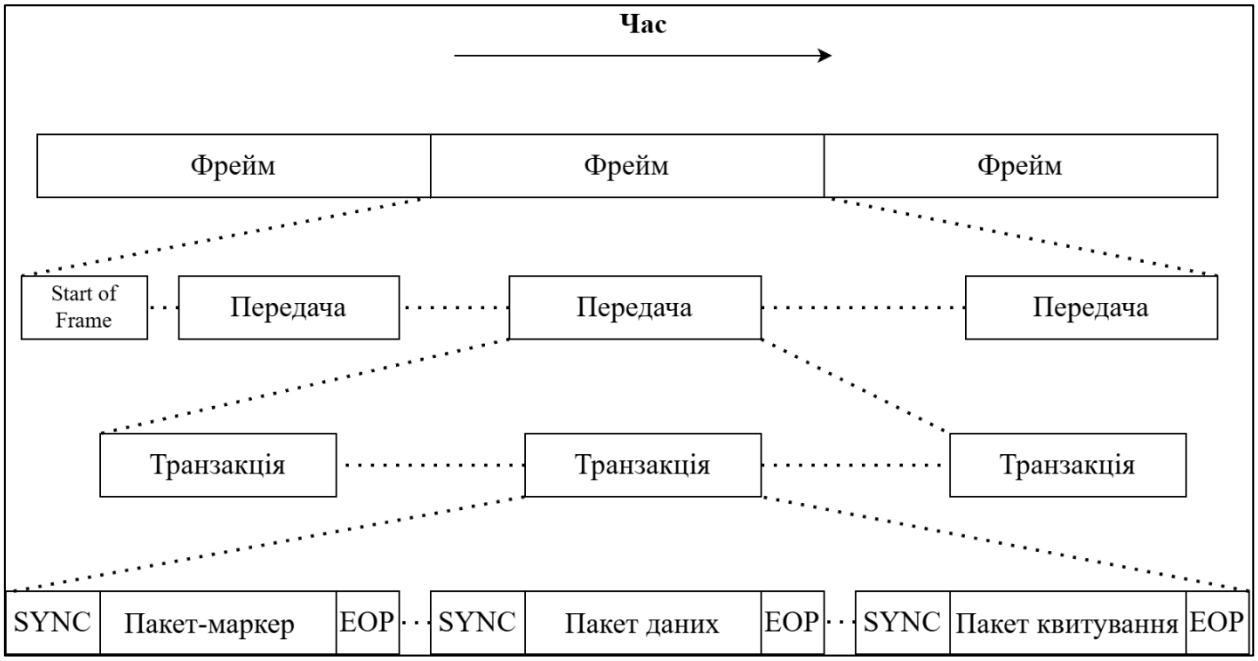

Рисунок 1.2 – USB комунікація з точки зору часу.

Кожен **пакет** значно відрізняється в залежності від його типу, але всі пакети мають загальну структуру:

- Ідентифікатора Пакету (PID) – (8 біт) вказує на тип пакета та напрямок;
- Опціональна адреса пристрою (ADDR) – (7 біт);
- Опціональна адреса кінцевої точки (EP) – (4 біти);
- Опціональні дані (Payload) – (від 0 до 1023 байтів);
- Опціональний циклічний надлишковий код (CRC) – (5/16 біт) [5; 6].

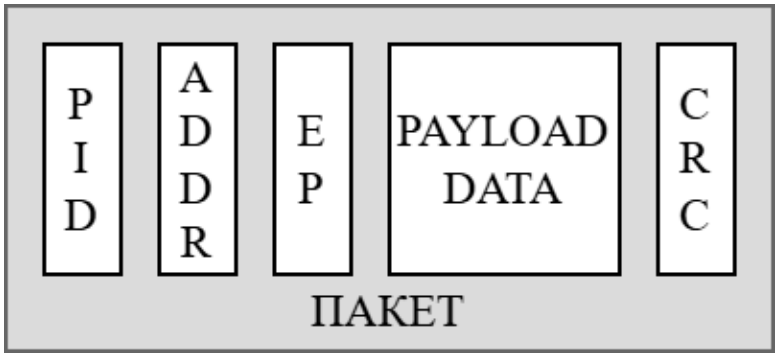

Рисунок 1.3 – Шаблон пакету USB [6].

Основні типи PID та пакетів:

- Token Packets (Пакети-маркери): Ініціюються хостом, ініціюють транзакції і ідентифікують пристрій, що бере участь у транзакції;
- Data Packets (Пакети даних): Містять дані (payload) від хосту або пристрою;
- Handshake Packets (Пакети квитування): Повідомляють про результат транзакції, надходять від одержувача даних до відправника даних;
- Special Packets (Спеціальні пакети): Використовується для виявлення різниці швидкості пристроїв, зазвичай між хостом і USB-хабом [6].

Транзакція – це послідовність пакетів, що виконує елементарну операцію обміну даними. Існує три основних типи транзакцій:

- IN Transaction (Транзакція Введення): Хост отримує дані від пристрою;
- OUT Transaction (Транзакція Виведення): Хост надсилає дані пристрою;
- SETUP Transaction (Транзакція Налаштування): Спеціальний тип OUT-транзакції, використовується для керуючих передач [6].

Транзакції об’єднуються в **передачі**:

- Control Transfers (Керуючі передачі): Використовуються для конфігурації пристрою під час енумерації, надсилання команд та отримання статусів і гарантують доставку даних;
- Interrupt Transfers (Переривчасті передачі): Призначені для невеликих обсягів даних, що передаються з гарантованою періодичністю (наприклад, для клавіатури чи миші). Важливо для пристроїв, яким потрібна швидка відповідь з низькою затримкою;
- Bulk Transfers (Блокові передачі): Призначені для передачі великих обсягів даних без гарантії швидкості чи затримки, але з гарантією доставки (наприклад, принтери, сканери, накопичувачі);
- Isochronous Transfers (Ізохронні передачі): Для потокової передачі даних у реальному часі (наприклад: аудіо і відео) з гарантованою пропускною здатністю, але без гарантії доставки при помилках [6].

Класи USB-пристроїв

Щоб спростити розробку драйверів, USB-IF визначила стандартні класи пристроїв. Якщо пристрій відповідає певному класу, він може використовувати стандартний драйвер операційної системи. Це дозволяє користуватися пристроєм і без завантаження драйверу відповідного конкретного пристрою [8].

Таблиця 1.1

Таблиця визначених кодів класів [6; 8]

Клас	Використання	Опис і приклад
00h	Пристрій	Не визначено. Клас пристрою не визначено, для визначення потрібних драйверів використовуються дескриптори інтерфейсів
01h	Інтерфейс	Аудіо. Динамік, мікрофон, звукова карта, MIDI
02h	Обидва	Комунікації та керування CDC. Модем, адаптер Ethernet, адаптер Wi-Fi
03h	Інтерфейс	Пристрої людського інтерфейсу (HID). Клавіатура, миша, джойстик
05h	Інтерфейс	Пристрій фізичного інтерфейсу (PID). Джойстик із силовим зворотним зв'язком
06h	Інтерфейс	Зображення. Камера, сканер
07h	Інтерфейс	Принтер Принтери, верстати з ЧПУ
08h	Інтерфейс	Масовий накопичувач. Зовнішні жорсткі диски, флеш-накопичувачі, карти пам'яті
09h	Пристрій	USB-концентратор
0Ah	Інтерфейс	CDC-Data. Використовується в поєднанні з класом 02h
0Bh	Інтерфейс	Smart Card. USB-зчитувач смарт-карт
0Dh	Інтерфейс	Content Security. Зчитування відбитків пальців
0Eh	Інтерфейс	Відео. Веб-камера
0Fh	Інтерфейс	Особиста охорона здоров'я. Пульсометр, глюкометр
10h	Інтерфейс	Аудіо/відео пристрої
11h	Пристрій	Клас пристрою Billboard
12h	Інтерфейс	Клас мосту USB Type-C
13h	Інтерфейс	Клас пристрою USB Bulk Display Protocol
14h	Інтерфейс	Клас кінцевого пристрою MCTP через USB
3Ch	Інтерфейс	Клас пристрою I3C
DCh	Обидва	Діагностичний пристрій. Пристрій для тестування відповідності USB
E0h	Інтерфейс	Бездротовий контролер. Адаптер Bluetooth
EFh	Обидва	Різне. Пристрій ActiveSync
FEh	Інтерфейс	Специфічний для програми. Міст IrDA, тестовий і вимірювальний клас (USBTMC), USB DFU (пряме оновлення прошивки)
FFh	Обидва	Специфічний для виробника. Вказує на те, що пристрій потребує драйверів певного виробника

1.2.3 Огляд існуючих підходів та рішень для реалізації обміну даними між ЦСП та ПК через USB-інтерфейс

Рішення на базі ЦСП/мікроконтролерів з вбудованим USB-контролером

Сучасні мікроконтролери та деякі ЦСП часто оснащуються вбудованими апаратними USB-контролерами. Це дозволяє реалізувати USB-зв’язок безпосередньо на кристалі, що може забезпечити вищу швидкість передачі даних та менше навантаження на центральний процесор мікроконтролера порівняно з програмними реалізаціями [9].

Переваги:

- Відсутність потреби у зовнішніх компонентах для реалізації USB-інтерфейсу;
- Апаратна реалізація USB-протоколу зазвичай забезпечує вищу швидкість, менші затримки і можливість оптимізації для низького енергоспоживання.

Недоліки та особливості:

- Мікроконтролери з апаратним USB можуть бути дорожчими за аналоги без нього;
- Налаштування та програмування вбудованого USB-контролера може вимагати глибокого розуміння специфікацій USB;
- Деякі USB-контролери завантажують свою прошивку з хост-комп’ютера в оперативну пам’ять і не можуть працювати автономно без підключення до хоста [10].

Пристрої з вбудованим USB, можуть використовувати стандартні драйвери операційної системи або драйвери від виробника. Хост-комп’ютер ідентифікує пристрій та його можливості за допомогою дескрипторів, які пристрій надає під час процесу енумерації [9].

Рішення з використанням зовнішніх USB-UART мостів

Це найбільш поширений підхід, особливо для мікроконтролерів, які не мають вбудованого USB-інтерфейсу, але мають апаратні UART, SPI або I2C.

Міст USB-UART (або USB-Serial) – це спеціалізована мікросхема, яка перетворює сигнали USB на сигнали послідовного інтерфейсу (і навпаки), значно спрощуючи розробку USB-пристроїв [9; 11; 12].

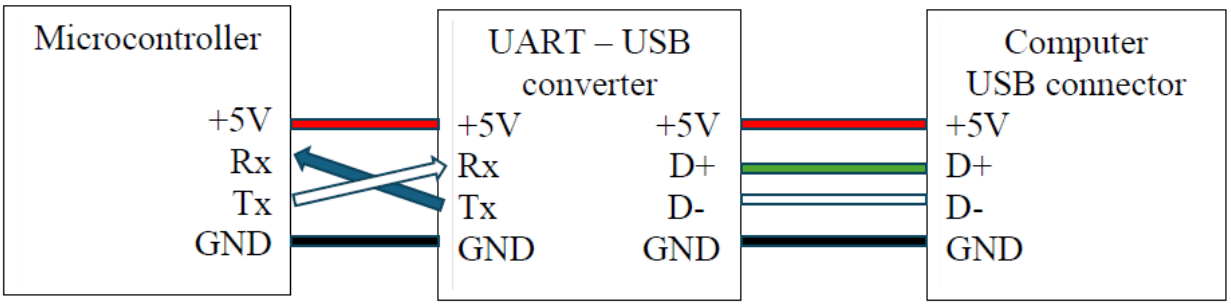

Рисунок 1.4 – Схема з'єднання комп'ютера з мікроконтролером через USB-UART міст [13].

Принцип роботи та переваги:

- Мікросхема-міст і драйвер Windows беруть на себе всю складність реалізації USB-протоколу, включаючи енумерацію, обробку пакетів, керування живленням тощо. Програмісту мікроконтролера потрібно лише реалізувати обмін даними через UART (або інший послідовний інтерфейс);
- Для операційної системи ПК пристрій виглядає як звичайний COM-порт. Програмне забезпечення на ПК отримує можливість використовувати прикладний програмний інтерфейс для роботи з COM-портами для обміну даними з мікроконтролером;
- Пристрої, що раніше використовували RS-232, можуть бути легко адаптовані для роботи через USB шляхом додавання мікросхеми-моста [9; 11; 12].

Програмна реалізація USB-протоколу

Цей підхід використовується, коли мікроконтролер не має апаратного USB-контролера і немає можливості використовувати зовнішню мікросхему-міст. USB-протокол (зазвичай обмежена версія) реалізується програмно, використовуючи стандартні лінії вводу/виводу мікроконтролера та його таймери і переривання.

Як приклад можна навести бібліотеку V-USB – яка надає можливість створити програмну реалізацію низькошвидкісного USB-пристрою для мікроконтролерів Atmel AVR без додаткових мікросхем.

Основні характеристики V-USB:

- Повністю сумісний з USB 1.1 (Low-Speed), за винятком обробки помилок зв’язку та електричних специфікацій;
- Підтримує декілька кінцевих точок: одну контрольну, дві interrupt/bulk-in та до семи interrupt/bulk-out (хоча специфікація USB забороняє bulk-endpoints для Low-Speed пристроїв, V-USB їх частково підтримує);
- Розміри передачі до 254 байт за замовчуванням, більше – як опція конфігурації;
- Працює на будь-якому AVR з щонайменше 2 кБ Flash, 128 байт RAM та тактовою частотою від 12 МГц;
- Не вимагає UART, таймера, блоку захоплення вхідного сигналу або іншого спеціального обладнання [10].

Переваги V-USB над мікроконтролерами з апаратним USB:

- Мікроконтролери серії AVR можуть бути швидшими та дешевшими;
- Забезпечують автономну роботу не потребуючи завантаження прошивку з хоста) [10].

Переваги V-USB над окремими USB-мостом мікросхемами:

- Відсутність додаткових витрат та апаратної складності;
- Можуть мати менше споживання енергії, коли USB відключено;
- Більша гнучкість у виборі USB-дескрипторів;
- Використовує лише 2-3 лінії вводу/виводу [10].

Недоліки програмної реалізації:

- Зазвичай підтримується лише Low-Speed USB (1.5 Мбіт/с);
- Значна частина ресурсів мікроконтролера витрачається на обробку USB-протоколу, що може обмежувати виконання основних завдань;

- Складність реалізації та відлагодження, точне дотримання часових характеристик USB-сигналів є критичним;
- Може бути складно забезпечити повну відповідність електричним специфікаціям USB без спеціалізованих апаратних компонентів [10].

Висновок за розділом 1

Проведений у першому розділі аналіз предметної області висвітлив ключову роль ЦСП та, зокрема, мікроконтролерів з функціями ЦСП, у сучасних системах керування та обробки даних. Важливим аспектом їхньої ефективної інтеграції є організація взаємодії з хост-пристроями, де USB-інтерфейс виступає поширеним та зручним рішенням.

В ході дослідження були розглянуті архітектурні особливості ЦСП, що дозволяють їм ефективно виконувати математичні операції, а також було зазначено, що для багатьох завдань керування та збору даних, де не потрібна екстремальна продуктивність, цілком придатними є мікроконтролери загального призначення. Детальний аналіз USB-інтерфейсу, його специфікацій, протоколів передачі даних та стандартних класів пристроїв показав його гнучкість та широкі можливості для підключення різноманітних периферійних пристроїв.

Огляд існуючих рішень для реалізації обміну даними між ЦСП (або мікроконтролерами, що виконують їхні функції) та персональним комп’ютером через USB виявив три основні підходи:

- Використання ЦСП/мікроконтролерів з вбудованим апаратним USB-контролером: Забезпечує високу швидкість та менше навантаження на процесор, однак може збільшувати вартість рішення та складність програмування;
- Програмна реалізація USB-протоколу: Дозволяє використовувати дешевші мікроконтролери без апаратної підтримки USB, але суттєво обмежує швидкість (зазвичай Low-Speed USB) та значно завантажує ресурси мікроконтролера, ускладнюючи реалізацію та налагодження;

- Використання зовнішніх USB-UART (або інших послідовних інтерфейсів) мостів: Цей підхід, поєднує простоту реалізації на стороні мікроконтролера (робота зі стандартним UART), доступність та низьку вартість компонентів, а також забезпечує достатню пропускну здатність для задач управління зовнішніми об’єктами. Операційна система ПК розпізнає такий пристрій як стандартний COM-порт, що спрощує розробку керуючого програмного забезпечення, спрощуючи роботу з USB-інтерфейсом.

Аналіз переваг та недоліків показав, що хоча існують різноманітні способи підключення, а стандартні класи USB-пристроїв та драйвери для USB-UART мостів забезпечують базову комунікацію, часто бракує уніфікованої алгоритмічної моделі та спеціалізованого, але гнучкого протоколу обміну для ефективного управління набором різнорідних зовнішніх об’єктів, підключених до ЦСП/мікроконтролера. Застосування простого послідовного каналу вимагає від розробника щоразу створювати власний протокол та логіку обробки команд і даних, що збільшує час розробки та зменшує універсальність рішень.

Таким чином, виникає обґрунтована необхідність у дослідженні та розробці комплексної алгоритмічної моделі управління зовнішніми об’єктами ЦСП через USB-інтерфейс, яка б враховувала особливості використання мікроконтролерів у ролі ЦСП та USB-UART перетворювачів. Розробка такої моделі, що включає чітко визначений протокол обміну, програмні драйвери для мікроконтролера та керуючу програму на ПК, є доцільною та актуальною задачею. Це дозволить систематизувати процес взаємодії, підвищити ефективність розробки та забезпечити гнучке й надійне керування зовнішніми пристроями, що і є основною метою кваліфікаційної роботи.

РОЗДІЛ 2.
РОЗРОБКА АЛГОРИТМІЧНОЇ МОДЕЛІ УПРАВЛІННЯ ЗОВНІШНІМИ ОБ’ЄКТАМИ ЦСП

2.1 Аналіз задачі управління та вибір платформи

Постановка задачі управління зовнішніми об'єктами через USB

Для реалізації зв’язку між хост-комп’ютером та ЦСП був обраний підхід з використанням зовнішнього USB-UART перетворювача. Такий вибір було зроблено внаслідок значної поширеності USB-UART перетворювачів, їх простоти використання та відсутності необхідності тонкого налаштування USB-інтерфейсу.

Основна проблема такого підходу полягає у необхідності створення власного механізму, який дозволить хост-комп’ютеру надсилати команди управління (наприклад, змінити стан виконавчого пристрою, запитати дані з сенсора) та отримувати відповіді (статуси, виміряні значення) від системи на базі мікроконтролера. Це вимагає створення власного протоколу обміну, який би враховував особливості передачі даних через USB та обмеження мікроконтролерної платформи і розробити керуючу програму для ПК та управляючу програму для мікроконтролера які будуть обмінюватися даними за цим протоколом.

Обґрунтування вибору апаратної платформи

В якості апаратної платформи для реалізації системи управління зовнішніми об’єктами була обрана плата, що є функціональним аналогом Arduino Uno. Ця платформа поєднує в собі мікроконтролер ATmega328P та інтегрований USB-UART перетворювач CH340C. Вибір у користь Arduino Uno був зроблений внаслідок того, що вона є надзвичайно популярною платформою для розробки та прототипування, має велику кількість навчальних матеріалів, прикладів коду та готових бібліотек, які спрощують підключення зовнішніх компонентів та розробку програмного забезпечення.

ATmega328P – це 8-бітний мікроконтролер сімейства AVR від Atmel. Він базується на вдосконаленій RISC-архітектурі з більшістю інструкцій, що виконуються за один тактовий цикл. Хоча ATmega328P не є спеціалізованим ЦСП, для поставлених завдань управління зовнішніми об’єктами та обробки команд від ПК його продуктивності цілком достатньо і використання ATmega328P все ще дозволить реалізувати загальний принцип комунікації з ЦСП за допомогою USB-інтерфейсу [13; 14; 15].

Мікроконтролер оснащений необхідними для роботи периферійними модулями:

- Апаратний модуль UART: використовується для зв’язку з USB-UART мостом CH340C;
- Цифрові порти введення/виведення;
- Вбудована енергонезалежна пам’ять EEPROM (1 КБ);
- Таймери/лічильники;
- 10-бітний аналого-цифровий перетворювач [14].

USB-UART міст CH340C є мікросхемою-конвертером USB шини, що перетворює USB в послідовний порт. У режимі UART CH340C дозволяє розширити послідовний порт для комп’ютерів або доєднати звичайний послідовний пристрій до USB. На копії плати Arduino Uno CH340C забезпечує зв’язок між UART-інтерфейсом мікроконтролера ATmega328P та USB-портом ПК, дозволяючи завантажувати прошивку та обмінюватися даними [11].

Характеристики CH340C:

- Повношвидкісний USB-інтерфейс, сумісний з USB 2.0;
- Має апаратний повнодуплексний UART інтерфейс з вбудованим буфером прийому-передачі;
- Підтримує швидкості передачі даних від 50 біт/с до 2 Мбіт/с;
- CH340C має вбудований тактовий генератор і не потребує зовнішнього кварцового резонатора;
- Підтримує живлення 5В та 3.3В. У даній конфігурації плати використовується живлення 5В від USB [11].

Обґрунтування вибору програмних засобів

В якості середовища розробки для мікроконтролера було обрано Arduino IDE, яка має величезну кількість готових бібліотек для роботи з різноманітними датчиками, виконавчими пристроями і інтерфейсами зв’язку, що значно прискорює процес розробки. Arduino IDE нативно підтримує плати сімейства AVR, в тому числі Arduino Uno.

Для розробки керуючої програми на персональному комп’ютері, яка буде взаємодіяти з мікроконтролерною системою через USB-UART, було обрано мову програмування Python з бібліотекою PySerial.

Бібліотека PySerial надає зручний доступ до роботи з послідовними портами і дозволяє легко налаштовувати параметри порту, та відправляти і отримувати дані. Оскільки USB-UART міст CH340C визначається в операційній системі як віртуальний COM-порт, PySerial ідеально підходить для організації обміну даними з мікроконтролером, надаючи можливість створити інтерфейс користувача [16].

Визначення зовнішніх об’єктів управління

В якості зовнішніх пристроїв мікроконтролеру, які забезпечать перевірку роботи створеного протоколу було обрано два модулі:

- Модуль HW-479 – трикольоровий діод з трьома резисторами, робота з яким продемонструє можливість управління зовнішніми об’єктами;
- Модуль KY-015 – датчик температури та вологості на базі чіпу DHT11. Діапазон вимірювання температури становить від 0 до 50°C, а вологості повітря – 20-90% Відносної вологості. Зв’язок здійснюється за допомогою спеціалізованого однопровідного протоколу. Робота з модулем продемонструє можливість передачі масиву даних до хосту [17; 18].

Обрана комбінація апаратних та програмних засобів дозволяє створити відносно просту в реалізації систему для управління зовнішніми об’єктами через USB-інтерфейс.

2.2 Проєктування алгоритмічної моделі управління

Декомпозиція задачі управління

Для системного підходу до розробки, задачу управління зовнішніми об'єктами було декомпозовано на наступні ключові функціональні рівні (блоки), кожен з яких виконує свої специфічні функції:

- Рівень ПК (Хост-система):
    - Відповідає за взаємодію з користувачем через інтерфейс користувача;
    - Формує команди управління на основі дій користувача.
    - Здійснює надсилання команд та прийом даних/статусів через віртуальний COM-порт;
    - Розбирає отримані відповіді, відображає інформацію користувачеві та, за потреби, зберігає дані.
- Рівень каналу зв'язку:
    - Включає фізичний USB-інтерфейс ПК, кабель USB;
    - Міст USB-UART (CH340C) на Arduino Uno, що перетворює USB-сигнали в UART-сигнали і навпаки;
    - UART-інтерфейс мікроконтролера ATmega328P, що забезпечує фізичний обмін даними з мостом CH340C.
- Рівень мікроконтролера (ATmega328P):
    - Приймає дані від ПК через апаратний модуль UART;
    - Обробляє вхідний потік байтів, збирає пакети даних відповідно до розробленого протоколу;
    - Перевіряє цілісність пакетів (наприклад, контрольну суму);
    - Розбирає команди та вилучає параметри;
    - Виконує команди, взаємодіючи з зовнішніми об'єктами;
    - Формує пакети-відповіді зі статусною інформацією або даними;
    - Надсилає відповіді на ПК через UART.

- Рівень зовнішніх об'єктів:
    - Безпосередньо керовані пристрої: світлодіодний модуль HW-479, датчик температури та вологості KY-015;
    - Внутрішні ресурси мікроконтролера, що використовуються як об'єкти управління/зберігання: регістри портів, пам'ять EEPROM.

Розробка загальної архітектури системи взаємодії

Загальна архітектура системи взаємодії побудована за моделлю “Master-Slave” (Ведучий-Ведений), де ПК виступає в ролі ведучого (Master), що ініціює обмін даними та надсилає команди управління. Мікроконтролерна система на базі ATmega328P функціонує як ведений пристрій (Slave), що обробляє отримані команди, взаємодіє з зовнішніми об’єктами та надсилає відповіді або дані на ПК. Канал зв’язку між ними реалізується через послідовне з’єднання USB-UART, де міст CH340C забезпечує фізичне та логічне перетворення сигналів.

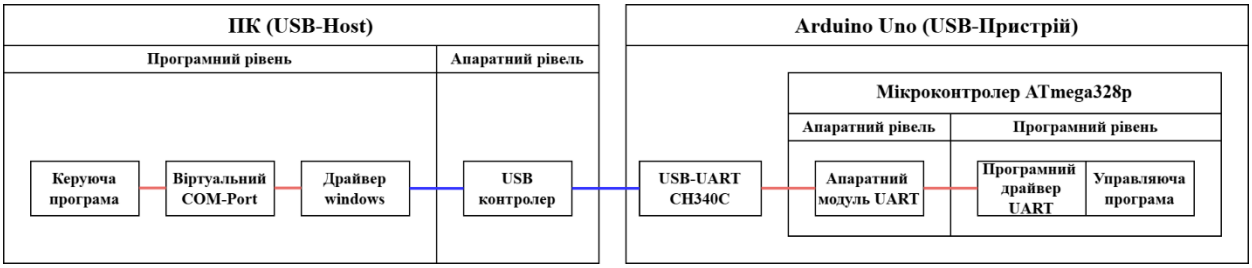

Рисунок 2.1 – Структурна схема каналу обміну інформації.

На рисунку 2.1 синім позначено передачу даних протоколу USB, а червоним позначено передачу даних власного протоколу.

2.3 Розробка протоколу обміну даними

Загальні вимоги до протоколу

Спочатку було визначено загальні вимоги, яким потрібен відповідати спроєктований протокол:

- Протокол повинен легко адаптуватися для передачі різних типів команд та даних, а також мати можливість розширення набору команд у майбутньому;
- Структура пакетів має бути компактною, щоб мінімізувати накладні витрати, особливо важливі для мікроконтролера з обмеженими ресурсами;
- Протокол повинен включати механізми виявлення помилок передачі для забезпечення цілісності даних;
- Формат пакету має бути таким, щоб його було легко розбирати як на стороні мікроконтролера, так і на стороні ПК.

З врахуванням цих вимог було розроблено спеціалізований протокол обміну даними який визначає формат пакетів, перелік команд та механізми їх обробки.

Опис структури пакету

Протокол базується на пакетній передачі даних. Кожен пакет має структуру, описану в таблиці 2.1.

Таблиця 2.1

Структура пакету

Тип	Назва	Розмір
Заголовок	Довжина поля даних	4 біти
	Команда/інструкція	4 біти
Дані	Параметри/Дані	0-14 байт
Кінець	Контрольна сума	0-1 байт

Елементи пакету:

- Довжина поля даних: Визначає довжину наступного поля “параметри/дані” в байтах (від 0 до 14 байт). Максимальна довжина даних 14 байт обрана як компроміс між можливістю передавати достатній обсяг інформації та обмеженнями буфера приймача мікроконтролера, також, USB-пакети які формує CH340C мають вміст до 32 байт, отже при максимальній довжині поля даних в 14 байт з заголовком і контрольним бітом в один USB-пакети буде вміщуватися рівно два пакети розробленого протоколу;
- Команда/Інструкція: Ідентифікатор команди, що передається від хоста до пристрою, або ідентифікатор відповіді/повідомлення від пристрою до хоста. Дозволяє кодувати до 16 різних команд/відповідей;
- Параметри/Дані: Містить параметри, необхідні для виконання команди (при передачі від ПК до мікроконтролера), або дані, що передаються у відповідь (при передачі від мікроконтролера до ПК). Розмір поля визначається значенням “Довжина поля даних”;
- Контрольна сума: Опціональне поле, що використовується для перевірки цілісності переданого пакету. Для даної реалізації обрано простий алгоритм XOR-суми всіх байтів пакету (заголовка та даних). Цей байт відсутній якщо довжина поля даних дорівнює нулю.

Формування заголовка:

Перший байт пакету містить як довжину поля даних, так і команду/інструкцію. Старші 4 біти цього байту представляють довжину поля даних, а молодші 4 біти – код команди/інструкції.

Специфікація команд

Далі була продумана система команд, яка визначає набір операцій, які можуть бути ініційовані хостом у таблиці 2.2, та типи відповідей, які може надсилати пристрій у таблиці 2.3.

Таблиця 2.2

Специфікація команд для управляючої програми на мікроконтролері

Номер (код)	Дія з боку пристрою	Очікувані параметри	Формат відповіді (команда для ПК, таблиця 2.3)
0	Відправка поточного статусу пристрою. Мікроконтролер зчитує стан регістра PORTB та поточні значення вказівників EEPROM.	-	Команда 0: “Відправлено статус”.
1	Зміна стану виводів порту B (до яких підключені світлодіоди). Мікроконтролер записує отримане значення в регістр PORTB. Також треба відзначити, що в Arduino Uno порти PB6 і PB7 використовуються як входи для тактового генератора, і їх значення ця команда не змінює.	Нове значення для регістра PORTB (1 байт)	Якщо успішно: Команда 3: “Успішно виконана команда (1) зміни стану регістру PORTB”. Якщо недостатньо даних: Команда 4: “Помилка виконання команди (1)...: недостатньо даних”.
2	Запит на отримання поточних даних з датчика температури та вологості. Мікроконтролер зчитує дані з підключеного датчика.	-	Команда 5: “Відправка поточних даних з датчику”.
3	Запит на отримання даних з датчика температури та вологості, збережених в EEPROM, які зберігаються 1 раз на хвилину за останні 8 годин роботи мікроконтролера.	-	Якщо дані є: Команда 6: “Відправка даних з EEPROM”. Якщо даних немає: Команда 7: "У EEPROM відсутні дані".
4	Запит на очищення збережених даних датчика з пам’яті EEPROM. Мікроконтролер виконує процедуру очищення відповідної області EEPROM.	-	Команда 8: “Звіт про успішне очищення даних з EEPROM”.

Таблиця 2.3

Специфікація команд (повідомлень/статусів) для керуючої програми на ПК

Номер (код)	Опис повідомлення/статусу	Формат даних
0	Відправлено статус пристрою. Містить поточний стан регістра PORTB (світлодіоди) та інформацію про використання пам’яті EEPROM.	1 байт: поточний вміст регістра PORTB. 2 байти: поточне значення температури і вологості з датчику. 2 байти: вказівник на наступну вільну комірку в EEPROM.
1	Помилка: отриманий від ПК пакет мав неправильну контрольну суму.	1 байт: розрахована мікроконтролером контрольна сума або отримана неправильна контрольна сума.
2	Помилка: мікроконтролер отримав невідомий (непідтримуваний) код команди від ПК.	1 байт: код невідомої команди, що був отриманий.
3	Повідомлення: команду (1) на зміну стану регістра PORTB було успішно виконано.	1 байт: новий, актуальний вміст регістра PORTB після виконання команди.
4	Помилка: при виконанні команди (1) на зміну стану регістра PORTB виявлено, що в пакеті від ПК було недостатньо даних (очікувався 1 байт, отримано 0).	-
5	Відправка поточних даних з датчика температури та вологості.	1 байт: значення температури. 1 байт: значення вологості.
6	Відправка даних історії показань датчика з EEPROM. Може знадобитися передача кількома пакетами, якщо обсяг даних перевищує 14 байт.	Послідовність записів, кожен запис складається з 2 байт: значення температури і значення вологості. Максимальна кількість записів в одному пакеті обмежена 7 (14 байт / 2 байти на запис). Загальна кількість збережень до 960 байт.
7	Повідомлення: у пам’яті EEPROM відсутні запити дані (наприклад, при запиті історії, якщо вона порожня, або після очищення).	-
8	Повідомлення: дані з EEPROM були успішно очищені.	-

Принципи розширюваності набору команд

Чотирьохбітове поле команди дозволяє визначити до 16 унікальних команд для кожного напрямку передачі. Якщо цього виявиться недостатньо, протокол може бути розширений кількома способами:

1. Використання одного з кодів команди як префікса для розширеного набору команд (наприклад, команда 1111 може означати, що наступний байт даних є розширеним кодом команди);
2. Збільшення розміру поля команди за рахунок зменшення поля довжини даних, або збільшення загальної довжини заголовка (що потребуватиме модифікації структури пакету). Для даної роботи 16 команд є достатнім.

Обмеження розробленої моделі та протоколу

- Кількість команд: Поточна реалізація протоколу з 4-бітним полем команди обмежує кількість унікальних команд до 16 для кожного напрямку передачі. Хоча передбачені механізми розширення, вони не реалізовані на даному етапі;
- Пропускна здатність: Загальна пропускна здатність каналу залежить від обраної швидкості UART-з'єднання (CH340C підтримує до 2 Мбіт/с, UART ATmega328P також підтримує до 2 Мбіт/с на частоті 16МГц) та накладних витрат протоколу (заголовки, контрольна сума). Для задач управління та періодичного збору даних з датчиків цього зазвичай достатньо [11; 14];
- Тип взаємодії: Модель реалізує синхронний обмін даними за типом “запит-відповідь” в рамках одного ведучого та одного веденого. Протокол не розрахований на багатомайстерні конфігурації або асинхронні повідомлення від пристрою без попереднього;
- Обробка помилок: Реалізовано базовий рівень обробки помилок (контрольна сума, невідома команда, недостатньо даних). Більш складні механізми, такі як нумерація пакетів для виявлення втрат або повторного впорядкування, не передбачені.

2.4 Реалізація програмних компонентів

Розробка програмного забезпечення для мікроконтролера (управляючої програми)

Управляюча програма (прошивка) для мікроконтролера ATmega328P була розроблена в середовищі Arduino IDE з використанням мови C/C++ та стандартних бібліотек, вбудованих в Arduino IDE: AVR Libc і Arduino Core. Програма відповідає за ініціалізацію периферії, прийом та обробку команд від ПК, взаємодію з зовнішніми об'єктами: світлодіодним модулем HW-479 та датчиком температури/вологості та формування відповідей.

Далі розглянемо частину програми, яка працює з створеним протоколом.

Ініціалізація:

- Ініціалізація UART на 9600 бод і переривання по завершенню прийому байта. На тактовій частоті 16 МГц похибка швидкості буде близько ~0.16% [13];
- Вмикаються глобальні переривання.

Обробка пакетів та реалізація протоколу:

- Прийом пакетів в перериванні по прийому UART:
  - Переривання спрацьовує при отриманні кожного байта по UART. Байти послідовно збираються у глобальний буфер *rx_buffer* розміром 16 байт за індексом *rx_index*;
  - Після отримання першого байту (заголовка), з нього вилучається довжина поля даних і програма розраховує очікувану повну довжину пакету;
  - Здійснюється перевірка, чи заявлена довжина не перевищує максимальний розмір буфера. Якщо перевищує, прийом скидається для уникнення переповнення;
  - Коли кількість отриманих байтів збігається з очікуваною повною довжиною пакету, встановлюється прапорець готовності обробки пакету, і *rx_index* скидається для прийому наступного пакету;

- Початкова реалізація не мала достатньо надійного захисту від переповнення *rx_buffer* у випадку отримання пошкодженого заголовка з валідною, але великою довжиною даних за якою слідувала несподівана кількість байт. Це було вирішено додаванням перевірки переповненості перед записом у буфер та скиданням індексу при спробі виходу за його межі;

- Обробка готового пакету в головному циклі програми:
  - Основний цикл програми перевіряє прапорець готовності обробки пакету;
  - Якщо пакет готовий, його вміст копіюється в локальний буфер всередині критичної секції (без дозволу переривань), а прапорець готовності обробки скидається. Це мінімізує час, протягом якого глобальний *rx_buffer* не може бути змінений перериванням, та запобігає обробці неповних або змінених даних;
  - З локальної копії вилучаються заголовок, довжина даних та код команди;
  - Якщо довжина даних дорівнює 0, команда виконується без перевірки контрольної суми. Якщо довжина даних більше 0, обчислюється контрольна сума для отриманих заголовка та даних і порівнюється з отриманою контрольною сумою з пакету. Якщо вони збігаються, викликається обробка отриманої команди, а якщо не збігаються, формується пакет-відповідь з кодом 1 (“Неправильна контрольна сума”) та отриманою невірною контрольною. сумою, який надсилається на ПК.

- Відправка пакетів-відповідей після обробки пакету:
  - Формує пакет згідно з протоколом (Розділ 2.3);
  - Побайтово надсилає пакет через UART за допомогою простого UART драйверу.

- Обчислення контрольної суми, реалізує простий алгоритм XOR для байтів заголовка та даних.

Частина програми, яка присвячена роботі з зовнішніми пристроями описана не буде, оскільки кваліфікаційна робота присвячена саме налаштуванню комунікації між управляючою і керуючою програмою. Повний код програми надано в лістингу Г.1.

Далі на рисунку 2.2 зображено спрощену блок-схему алгоритму управляючої програми.

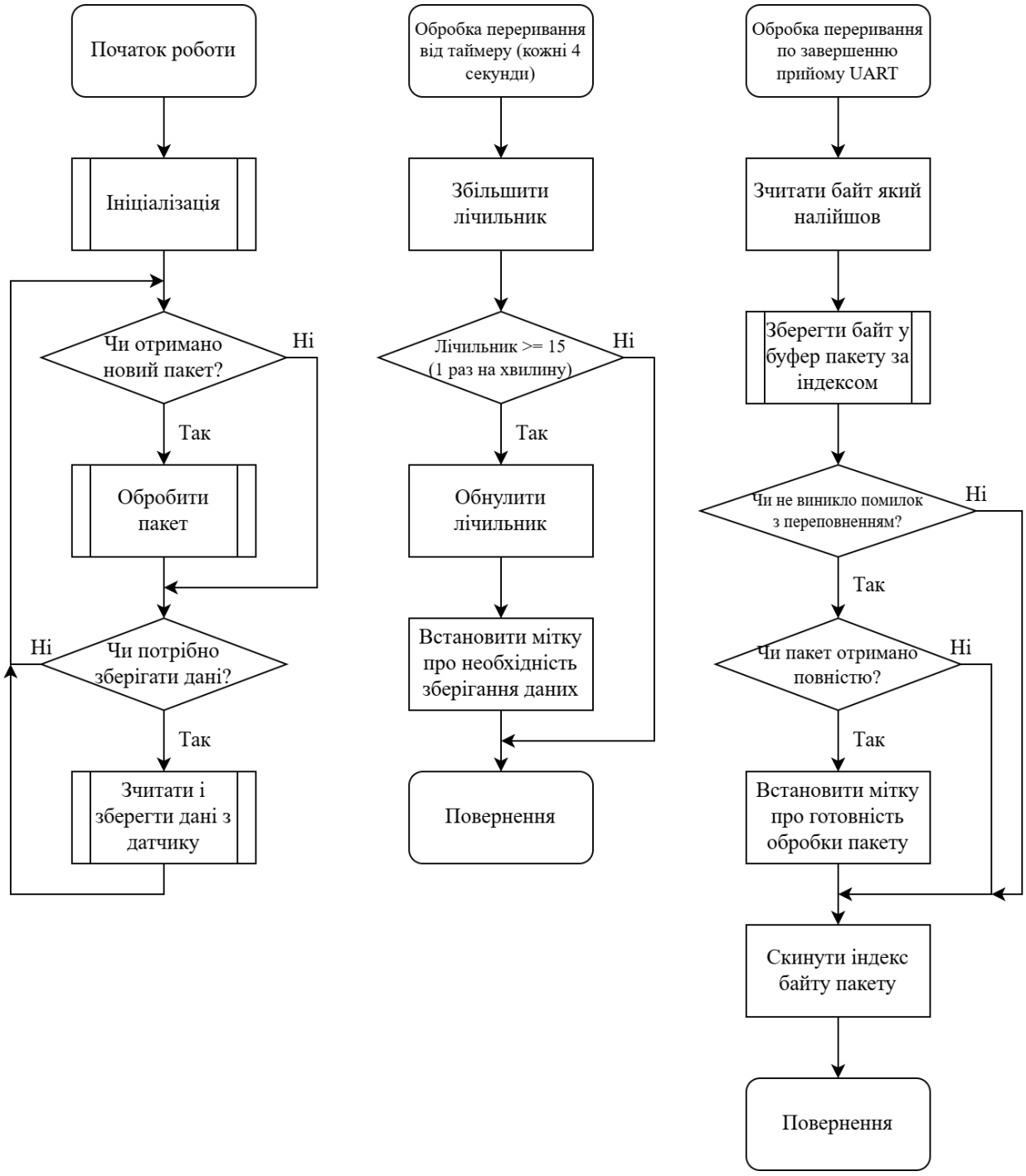

Рисунок 2.2 – Блок-схема роботи головного циклу і переривань управляючої програми.

Розробка програмного забезпечення для ПК (керуючої програми)

Керуюча програма на ПК розроблена мовою Python з використанням бібліотеки PySerial для взаємодії з послідовним портом. Програма надає користувачеві можливість вибору між прямим введенням команд (для діагностики роботи системи) та простим консольним інтерфейсом.

Далі аналогічно управляючій команді описано як програма працює саме з послідовним портом та пакетами:

- Відкриває послідовний порт з таймаутом читання і робить невелику паузу для стабілізації з'єднання;
- Відповідно протоколу, формує заголовок та додає контрольну суму для пакетів з даними;
- Розбирає отримані байти, перевіряє довжину та контрольну суму (для пакетів з даними). Показує користувачеві отриману команду і дані та їх значення або показує і описує помилку;

Послідовність функції прийому пакету:

- Після отримання першого байту (заголовка), визначає очікувану повну довжину пакету;
- Продовжує читати дані, поки не буде отримано достатньо байтів для повного пакету або не вийде загальний таймаут;
- Якщо заявлена довжина пакету з заголовка є надмірною, буфер очищується для запобігання проблемам;
- Розбирає і аналізує зібраний пакет;
- Повертає розібраний пакет або *None* у випадку таймауту/помилки.

Повний код програми наведено в лістингу Г.2.

Висновок за розділом 2

У другому розділі було проведено детальну розробку алгоритмічної моделі управління зовнішніми об'єктами цифрового сигнального процесора (в ролі якого виступає мікроконтролер ATmega328P) за допомогою USB-інтерфейсу.

На першому етапі проаналізовано задачу та обґрунтовано вибір апаратно-програмної платформи, включаючи підхід з USB-UART перетворювачем (CH340C на Arduino Uno) для спрощення USB-комунікації. Обґрунтовано вибір мікроконтролера ATmega328P, його відповідність задачам управління, а також програмних засобів: середа розробки Arduino IDE і мова C/C++ для мікроконтролера та мова Python (PySerial) для ПК. Визначено зовнішні об’єкти (модуль HW-479, датчик KY-015) для тестування моделі.

На другому етапі було спроєктовано алгоритмічну модель управління. Задача була декомпозивана на чотири ключові функціональні рівні: рівень ПК, рівень каналу зв'язку, рівень мікроконтролера та рівень зовнішніх об'єктів, що дозволило чітко структурувати систему. Розроблено загальну архітектуру системи взаємодії на основі моделі “Master-Slave” та візуалізовано її у вигляді структурної схеми каналу обміну інформації на рисунку 2.1.

Ключовим третім етапом стала розробка спеціалізованого протоколу обміну даними, обґрунтована вимогами до ефективності, мінімальної надлишковості та гнучкості. Детально описано структуру пакету та специфіковано набори команд для мікроконтролеру (Таблиця 2.2) і ПК (Таблиця 2.3). Охарактеризовано послідовності взаємодії, механізми обробки помилок, принципи розширюваності та обмеження моделі й протоколу.

На завершальному етапі реалізовано програмні компоненти: управляючу програму для ATmega328P та керуючу для ПК. Описано методику розробки, ключові алгоритми (ініціалізація, прийом/відправка пакетів, обробник переривань UART) та реалізацію функцій протоколу на ПК, включаючи обробку потокових даних та консольний інтерфейс.

Таким чином, у другому розділі було послідовно пройдено шлях від аналізу задачі до детального опису спроєктованої та реалізованої алгоритмічної моделі та протоколу, що створює основу для подальшої повної програмної реалізації та тестування системи управління зовнішніми об’єктами ЦСП через USB-інтерфейс.

РОЗДІЛ 3.
ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ АЛГОРИТМІЧНОЇ МОДЕЛІ

3.1 Тестування розробленої системи управління зовнішніми об’єктами

Тестовий стенд та програмне забезпечення

Для проведення експериментальних досліджень було зібрано тестовий стенд, що включав апаратні та програмні компоненти, описані в розділах 2.1 і 2.4.

Апаратна частина:

- Персональний комп’ютер (ПК) з операційною системою Windows та встановленим Python;
- Плата, функціональний аналог Arduino Uno, на базі мікроконтролера ATmega328P та USB-UART перетворювача CH340C;
- Модуль світлодіодів HW-479, підключений до цифрових виводів порту B мікроконтролера;
- Датчик температури та вологості KY-015 (на базі DHT11), підключений до цифрового виводу мікроконтролера;
- Кабель USB для з'єднання плати Arduino Uno з ПК.

Структурна схема і зображення фізичного підключення зовнішніх об’єктів апаратної частини зображено на рисунках 3.1 і 3.2.

Програмна частина:

- Управляюча програма (прошивка) для мікроконтролера ATmega328P, розроблена в середовищі Arduino IDE;
- Керуюча програма на мові Python з використанням бібліотеки PySerial, що реалізує взаємодію з мікроконтролером з боку ПК.

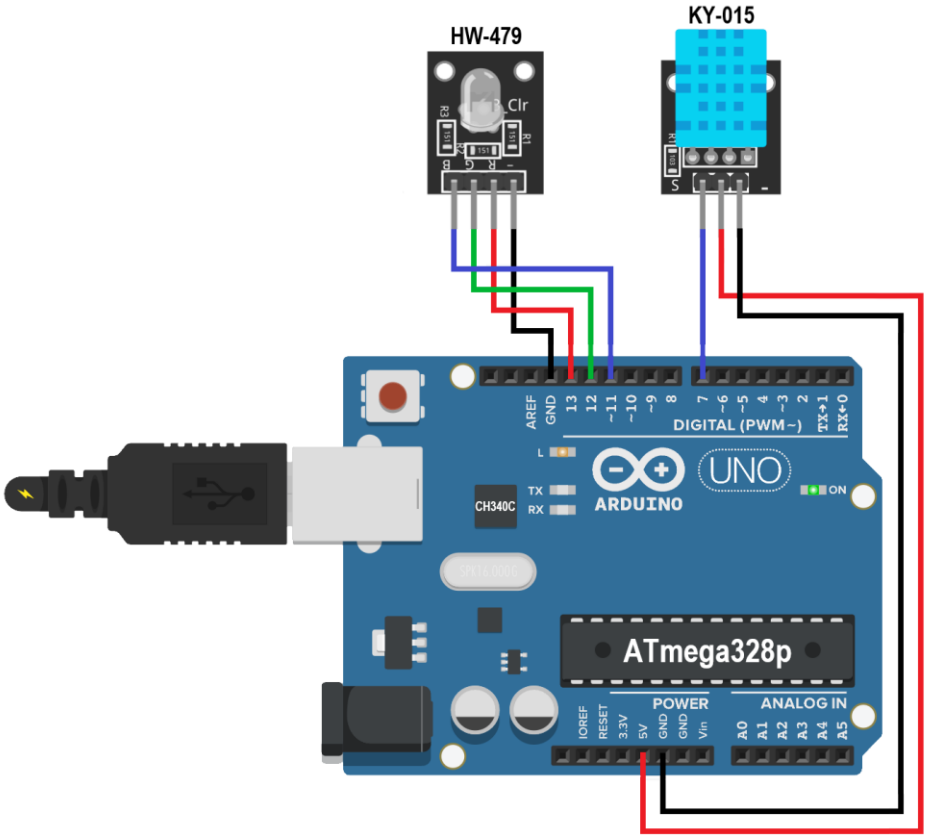

Рисунок 3.1 – Структурна схема підключення зовнішніх об’єктів.

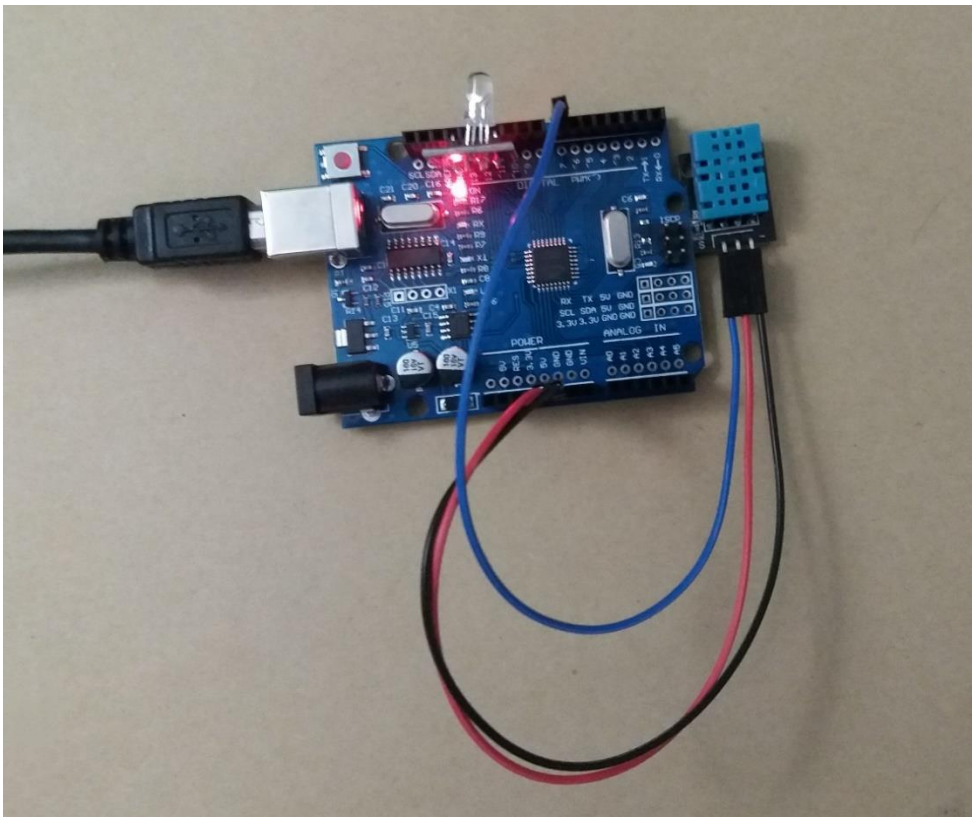

Рисунок 3.2 – Зображення підключення зовнішніх об’єктів.

Тестування функціональності протоколу обміну даними

Перевірка функціональності протоколу обміну даними здійснювалася шляхом відправки команд від керуючої програми на ПК до управляючої програми на мікроконтролері та аналізу отриманих відповідей. Для діагностики було використано програму-аналізатор трафіку Wireshark з підключеним драйвером захоплення USB-пакетів USBPcap.

Після підключення Arduino Uno до ПК і встановлення відповідного драйверу для чіпу CH340C з’явився віртуальний COM-порт.

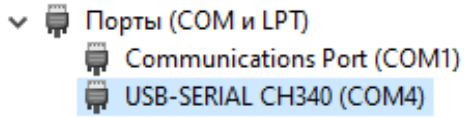

Рисунок 3.3 – Підключений пристрій у диспетчері пристроїв Windows.

Далі було запущено керуючу програму і відправлено команду запиту статусу “0”.

```
Спроба підключення до COM4 на швидкості 9600 бод.
З'єднано з COM4 на швидкості 9600 бод.
Використати CLI-інтерфейс? (y/n, за замовчуванням y): y
CLI-інтерфейс активовано.

Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 0
Відправка: 00
Отримано та розібрано пакет: 50 00 18 3D 08 01 7C
Отримано відповідь: Код=0x0, Дані='00 18 3D 08 01'
Статус: PORTB(0-5)=0b000000, Температура=24°C, Вологість=61%, EEPROM_ptr=264
```

Рисунок 3.4 – Термінал керуючої програми.

Як можна побачити на рисунку 3.4, керуюча програма сформувала і відправила відповідний пакет “0x00”, а управляюча програма надала відповідь на отриману команду: пакет “0x5000183D08017C”, після чого керуюча програма проаналізувала отримані дані і надала їх у зручній для користувача

формі. Пакети і їх опис було сформовано відповідно до створеного протоколу, описаного в розділі 2.3.

Тепер проаналізуємо ці пакети зі сторони USB за допомогою програми-аналізатор трафіку Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
11317	8783.424200	host	2.3.2	USB	28	URB_BULK out
11318	8783.424245	2.3.2	host	USB	27	URB_BULK out
11319	8783.468026	2.3.2	host	USB	34	URB_BULK in
11320	8783.468042	host	2.3.2	USB	27	URB_BULK in

Рисунок 3.5 – Захоплені USB-фрейми.

Як можна побачити на рисунку 3.5, пакети створеного протоколу на рівні USB мають вигляд 4 фреймів:

1. Передача команди від ПК до мікроконтролера. Тип передачі: *URB_BULK out* (Передача блокових даних назовні);
2. Підтвердження передачі команди;
3. Передача відповіді від мікроконтролера до ПК. Тип передачі: *URB_BULK in* (Передача блокових даних всередину);
4. Підтвердження прийому відповіді ПК.

Тепер проаналізуємо перший фрейм.

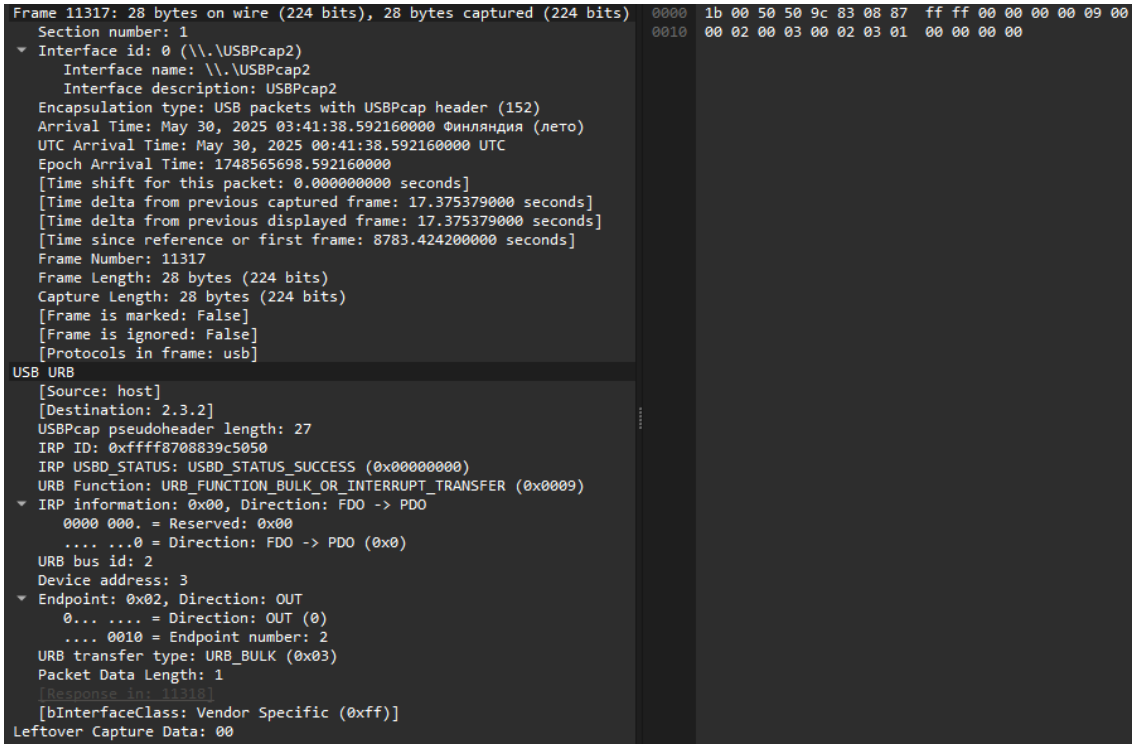

Рисунок 3.6 – Вміст і опис першого фрейму.

Проаналізуємо зміст і опис першого фрейму з рисунку В.6. Цей фрейм демонструє, як операційна система ПК надсилає USB-транзакції типу передачі блокових даних назовні (*BULK OUT*) на адресу 0x0300, яка відповідає віртуальному COM-порту. Довжину даних пакету зазначено в 1 байт з вмістом 0x00, що відповідає відправленому з керуючої програми пакету.

Тепер проаналізуємо третій фрейм з відповіддю від мікроконтролера.

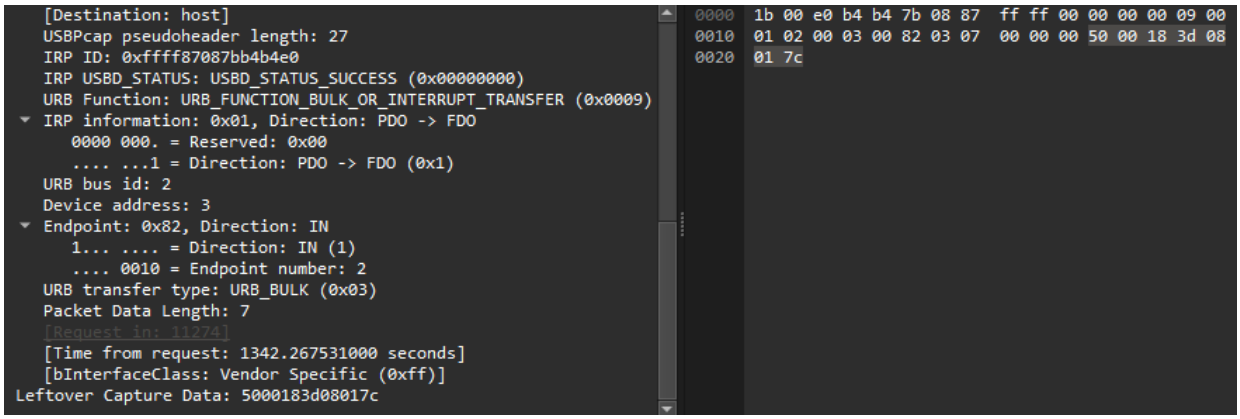

Рисунок 3.7 – Вміст і опис третього фрейму.

Як можна побачити на рисунку 3.7, в пакеті зазначено зворотній напрям блокових даних з 7 байтами даних: 0x5000183D08017C, що відповідають отриманому пакету даних в інтерфейсі керуючої програми на рисунку 3.4.

Отже, аналіз підтверджує, що розроблений протокол обміну даними коректно інкапсулюються у стандартні USB-передачі типу BULK, що забезпечує взаємодію між ПК та мікроконтролерною системою.

Тепер перейдемо до аналізу роботи всіх команд розробленого протоколу.

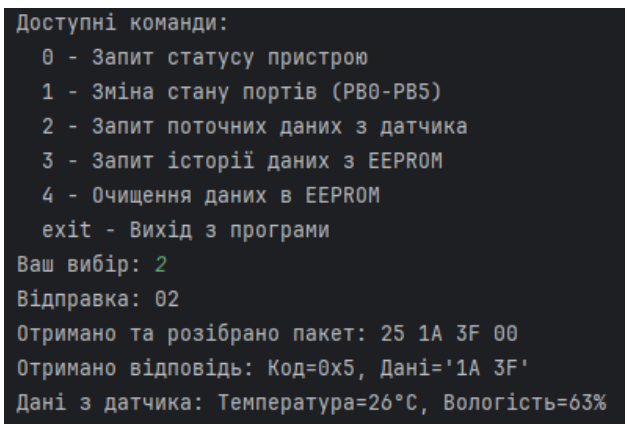

Рисунок 3.8 – Успішний запит даних з датчика температури та вологості.

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 1
Введіть значення для PORTB (PB0-PB5) у шістнадцятковому форматі (напр. '0F'): 10
Відправка: 11 10 01
Отримано та розібрано пакет: 13 10 03
Отримано відповідь: Код=0x3, Дані='10'
Команду 1 (зміна PORTB) виконано. Новий стан PORTB(0-5): 0b010000
```

Рисунок 3.9 – Тестування зміни вмісту регістру PORTB який відповідає за стан виходів порту B.

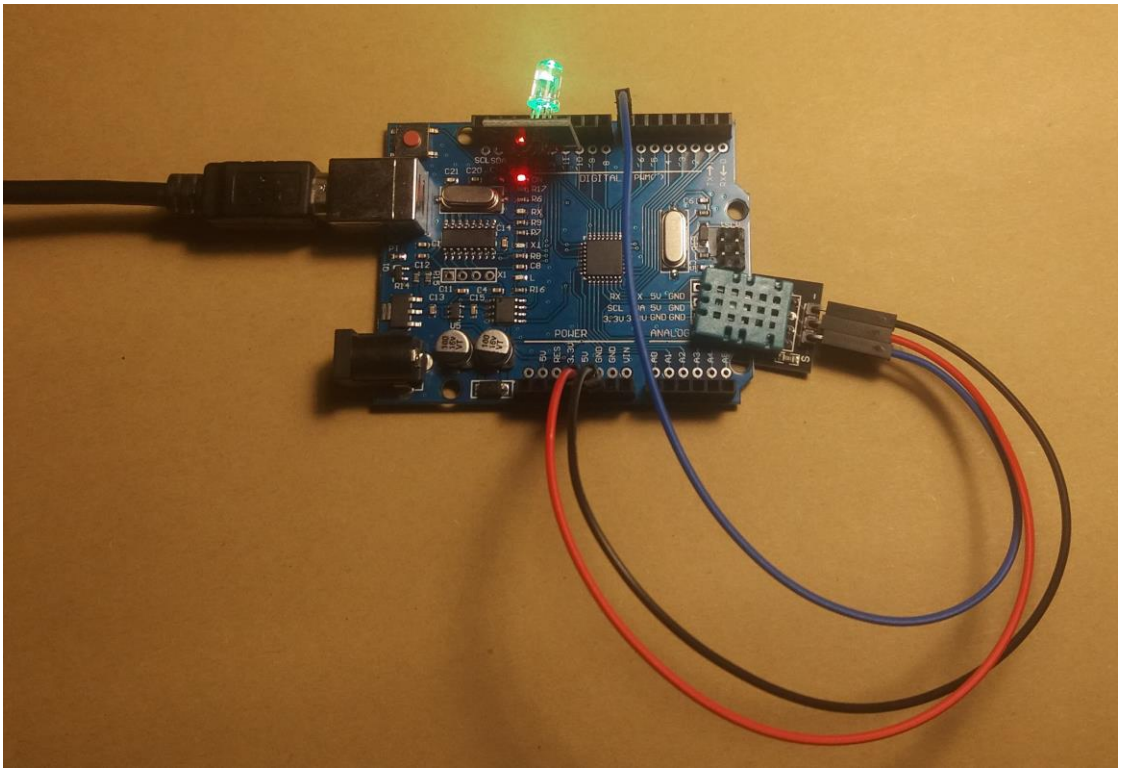

Рисунок 3.10 – Результат зміни стану портів B.

Як можна побачити на рисунках 3.9 і 3.10, успішно відбулася зміна стану регістру PORTB на 0x010000, змінивши стан порту 12 (PB4), до якого доєднано зелений діод модуля HW-479.

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 3
Відправка: 03
Очікування даних EEPROM...
Отримано та розібрано пакет: E6 1A 39 1A 39 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A C6
Отримано та розібрано пакет: E6 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A C6
Отримано та розібрано пакет: E6 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A 1A 3A C6
Отримано та розібрано пакет: E6 18 3C 18 3C 18 3C 18 3C 18 3C 18 3C 18 3C C2
Отримано та розібрано пакет: E6 18 3C 18 3C 18 3C 18 3C 18 3C 18 3C 18 3C C2
Отримано та розібрано пакет: E6 18 3B 18 3B 18 3B 18 3B 18 3C 18 3B 18 3B C2
Отримано та розібрано пакет: 86 18 3B 18 3A 18 39 19 39 86
Увага: Таймаут очікування відповіді (2.0с). Отримано:
Дані з EEPROM (235 записів) збережено у файл 'eeprom_data.dat'
```

Рисунок 3.11 – Запит на відправку 960 байт даних з EEPROM.

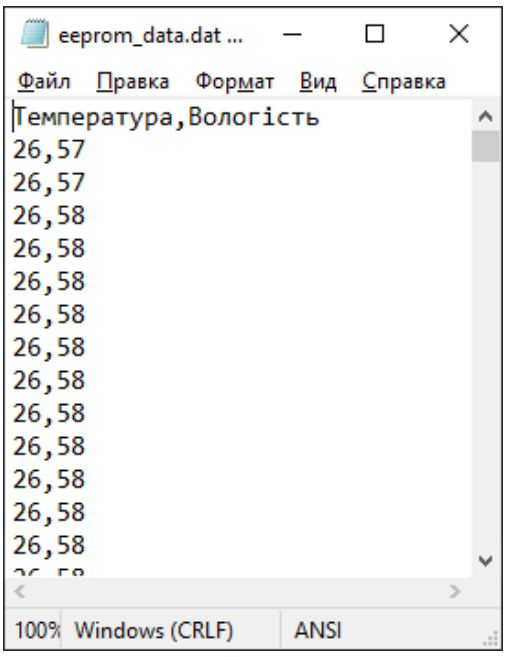

Рисунок 3.12 – Збережені дані.

Під час перевірки роботи команди запиту даних з EEPROM, результат роботи якої зображено на малюнках 3.11 і 3.12 було виявлено недолік керуючої програми. Цей недолік полягав у тому, що при спробі передати

великий обсяг даних, такий як 960 байт з EEPROM, керуюча програма не встигала обробити всі пакети, що надходили від мікроконтролера. Мікроконтролер надсилав дані пакетами по 16 байт, які міст USB-UART CH340C агрегував у 32-байтні USB-фрейми. Початкова версія Python-скрипту зчитувала весь USB-фрейм, але обробляла лише перший 16-байтний пакет, втрачаючи другий через використання локального буфера, що не зберігався між ітераціями. Проблему було вирішено шляхом введення стійкого буфера, який накопичує всі байти, отримані з послідовного порту. Спеціальна функція тепер ітеративно намагається виділити та розібрати один повний пакет з початку цього стійкого буфера.

```
Пакет 68: Дані='1A 38 1A 38 1A 38 1A 38 1A 38 1A 38 1A 38'
Пакет 69: Дані='1A 38 1A 38 1A 38 1A 38'
Дані з EEPROM (480 записів, 69 пакетів) збережено у файл 'eeprom_data.dat'
```

Рисунок 3.13 – Результат виправлення роботи керуючої програми.

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 4
Відправка: 04
Отримано та розібрано пакет: 08
Отримано відповідь: Код=0x8, Дані=''
Відповідь мікроконтролера: Дані з EEPROM успішно очищено.

Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 3
Відправка: 03
Очікування даних EEPROM...
Отримано та розібрано пакет: 07
Мікроконтролер повідомив, що в EEPROM відсутні дані.
```

Рисунок 3.14 – Перевірка роботи при видаленні даних з EEPROM.

```
Спроба підключення до COM4 на швидкості 9600 бод.
З'єднано з COM4 на швидкості 9600 бод.
Використати CLI-інтерфейс? (y/n, за замовчуванням y): n
Режим прямого введення пакетів. Введіть 'exit' для виходу.
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 0A
Відправка пакету: 0A
Отримано та розібрано пакет: 12 0A 18
Отримано відповідь: Код=0x2, Дані='0A'
Помилка від мікроконтролера: Невідома команда. Код отриманої команди: 0x0A
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 01
Відправка пакету: 01
Отримано та розібрано пакет: 04
Отримано відповідь: Код=0x4, Дані=''
Помилка від мікроконтролера: Для команди 1 (зміна PORTB) недостатньо даних.
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 11 10 0A
Відправка пакету: 11 10 0A
Отримано та розібрано пакет: 11 0A 1B
Отримано відповідь: Код=0x1, Дані='0A'
Помилка від мікроконтролера: Неправильна контрольна сума пакету від ПК. Отримана МК контрольна сума (якщо передано): 0x0A
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 10
Відправка пакету: 10
Увага: Таймаут очікування відповіді (3с). Отримано:
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 00
Відправка пакету: 00
Увага: Таймаут очікування відповіді (3с). Отримано:
Введіть пакет для відправки (шістнадцятковий формат, напр. '01 0F' або '02'): 00
Відправка пакету: 00
Отримано та розібрано пакет: 11 00 11
Отримано відповідь: Код=0x1, Дані='00'
Помилка від мікроконтролера: Неправильна контрольна сума пакету від ПК. Отримана МК контрольна сума (якщо передано): 0x00
```

Рисунок 3.15 – Перевірка роботи при відправці некоректних пакетів.

Під час перевірки обробки некоректних пакетів, описаної на рисунку 3.13, була виявлена проблема. У випадку коли ПК надсилає заголовок, що вказує на наявність даних (наприклад, 10, де 1 - довжина даних), але потім не надсилає самі дані та контрольну суму, мікроконтролер залишається в стані очікування цих байтів. Коли ПК надсилає наступний пакет (наприклад, 00), мікроконтролер сприймає його як продовження попереднього, некоректно зібраного, пакету. Це призводить до таймаутів на стороні ПК, а зрештою до помилки контрольної суми на мікроконтролері, коли він нарешті “збирає” пакет із частин різних передач.

3.2 Аналіз отриманих результатів

Відповідність результатів теоретичним очікуванням

Загалом, експериментальні результати підтвердили працездатність розробленої алгоритмічної моделі та системи управління зовнішніми об’єктами.

- Було підтверджено, що обраний підхід із використанням USB-UART перетворювача (CH340C) забезпечує стабільний канал зв'язку між персональним комп'ютером та мікроконтролером ATmega328P. Керуюча програма на ПК успішно ініціювала зв'язок через віртуальний COM-порт, а управляюча програма на мікроконтролері коректно приймала та відправляла дані через UART-інтерфейс;
- Аналіз USB-трафіку за допомогою Wireshark показав, що пакети розробленого прикладного протоколу коректно інкапсулюються у стандартні USB-транзакції типу BULK. Це відповідає теоретичним очікуванням щодо роботи USB-UART мостів, які транслюють дані послідовного порту через USB без зміни їх змісту на даному рівні;
- Команди, визначені у протоколі в таблиці 2.2, виконувалися мікроконтролером коректно, що підтверджувалося як відповідями, отриманими керуючою програмою на ПК, так і спостереженням за станом зовнішніх об'єктів (зміна стану світлодіодів, зчитування даних з датчика);
- У ході тестування було виявлено певні недоліки у програмній реалізації, зокрема, проблему з обробкою великих обсягів даних з EEPROM керуючою програмою, яку було відразу виправлено та потенційну проблему з “зависанням” мікроконтролера при отриманні неповних пакетів.

Обмеження розробленої моделі та протоколу:

- Пропускна здатність каналу зв'язку: Канал зв'язку реалізовано через UART на швидкості 9600 біт/с. Враховуючи формат кадру UART 8N1 (10 біт на байт), теоретична максимальна пропускна здатність становить 960 байт/с. Накладні витрати розробленого протоколу складають 2 байти для пакетів з даними. Отже, для пакета з максимальним корисним навантаженням 14 байт даних (загальний розмір пакету 16 байт), ефективна пропускна здатність для корисних даних становить 960 байт/с × (14/16) = 840 байт/с. Така пропускна

здатність є достатньою для передачі команд управління, зчитування невеликих обсягів даних з датчиків та оновлення стану. Однак для передачі великих масивів даних (наприклад, оцифрованих сигналів або великих файлів конфігурації) швидкість 9600 біт/с буде суттєвим обмеженням, що призводитиме до значних затримок. Хоча мікроконтролер ATmega328P та міст CH340C підтримують вищі швидкості UART (до 2Мбіт/с), у поточній реалізації в них немає потреби;

- Мікроконтролер ATmega328P не є спеціалізованим ЦСП. Для задач, що вимагають інтенсивної цифрової обробки сигналів, необхідні більш потужні процесори. Однак розроблена модель взаємодії та протокол можуть бути адаптовані і для них;
- Реалізована модель “запит-відповідь” є синхронною. Протокол у поточній версії не підтримує асинхронні повідомлення від пристрою до ПК без попереднього запиту.

Відповідність світовим тенденціям вирішення поставленої задачі:

- Використання USB-інтерфейсу, зокрема через доступні мости USB-UART (як CH340C), для підключення мікроконтролерних пристроїв до ПК є широко розповсюдженою та стандартною практикою у промисловості, аматорській електроніці та освітніх цілях;
- Розробка власних прикладних протоколів поверх стандартних транспортних механізмів (як UART) є типовим підходом, коли функціонал стандартних протоколів є надлишковим;
- Застосування високорівневих мов програмування, таких як Python, для створення керуючих програм та інтерфейсів користувача на ПК є сучасною тенденцією, що значно прискорює процес розробки та забезпечує кросплатформеність.
- Поточна реалізація обробника пакетів на мікроконтролері не включає механізм таймауту очікування повного пакету після отримання його заголовка. Як було продемонстровано під час тестування, якщо хост-

комп'ютер надсилає коректний заголовок, що декларує певну довжину даних, але не передає ці дані та контрольну суму повністю (наприклад, через обрив зв'язку або програмний збій на хості), мікроконтролер залишається в стані очікування решти байтів. Дане обмеження є наслідком свідомого спрощення логіки прийому пакетів в обробнику переривань UART, припускаючи низьку ймовірність непередбаченої поведінки внаслідок помилки.

3.3 Оцінка значущості роботи

Передбачувані області використання результатів роботи:

- Освітня сфера: Створення лабораторних стендів для вивчення мікроконтролерів, ЦСП, інтерфейсів, розробки систем збору даних та управління;
- Науково-дослідна діяльність: Прототипування систем збору даних з датчиків (з передачею на ПК для аналізу) та автоматизація експериментів (управління пристроями, реєстрація параметрів);
- Системи “розумного будинку” та домашня автоматизація: Розробка доступних рішень для управління освітленням, кліматом, механізмами та моніторингу приміщень з ПК;
- Прототипування промислових контролерів: Швидка розробка й тестування логіки управління для промислової автоматики, з взаємодією з ПК для конфігурації, моніторингу чи як частина розподілених систем;
- Аматорська робототехніка та власні проєкти: Інструмент для зв'язку мікроконтролерних пристроїв з ПК для управління та візуалізації даних;
- Інструменти для відлагодження вбудованих систем: ПЗ для ПК, що через розроблений протокол взаємодіє з мікроконтролером для діагностики та відлагодження.

Оцінка господарської, наукової та соціальної значущості роботи:

- Господарська значущість:
  - Зменшення часу та витрат на розробку систем взаємодії ПК-мікроконтролер через USB завдяки готовій алгоритмічній моделі та протоколу;
  - Зниження собівартості розробки завдяки доступній елементній базі (аналоги Arduino Uno) та вільному ПЗ (Arduino IDE, Python);
- Наукова значущість:
  - Робота полягає у розробці алгоритмічної моделі управління зовнішніми об’єктами ЦСП через USB-інтерфейс, що систематизує підхід до проєктування таких систем, особливо при використанні стандартних USB-UART перетворювачів;
  - Запропоновано спеціалізований, але гнучкий та розширюваний протокол обміну даними, який оптимізовано для передачі команд управління та телеметричної інформації з урахуванням обмежених ресурсів мікроконтролера, при цьому забезпечуючи надійність передачі через контрольну суму;
  - Досліджено та практично реалізовано механізми взаємодії на різних рівнях (апаратний, програмний драйвер мікроконтролера, прикладне програмне забезпечення), що становить методичну основу для побудови аналогічних систем;
- Соціальна значущість:
  - Розширення можливостей для технічної творчості та інновацій серед аматорів, студентів та невеликих компаній завдяки спрощенню розробки систем управління;
  - Потенційне використання розроблених принципів у створенні допоміжних технологій або пристроїв для покращення якості життя, наприклад, у системах домашньої автоматизації для людей з обмеженими можливостями.

3.4 Оцінка реалізації поставлених завдань та досягнення мети роботи

Метою кваліфікаційної роботи була розробка алгоритмічної моделі управління зовнішніми об’єктами, підключеними до цифрового сигнального процесора (в ролі якого виступає мікроконтролер ATmega328P на платформі Arduino Uno), за допомогою USB-інтерфейсу.

Для досягнення поставленої мети в ході виконання кваліфікаційної роботи було вирішено наступні завдання:

1. Проведено комплексний аналіз архітектури та принципів роботи цифрових сигнальних процесорів і мікроконтролерів, що виконують схожі функції (зокрема ATmega328P), та оцінено їх придатність для реалізації поставлених завдань управління (Розділи 1.2.1, 2.1);
2. Виконано дослідження USB-інтерфейсу, його специфікацій, протоколів передачі даних, стандартів, а також ролі та принципів роботи перетворювачів USB-UART (на прикладі CH340C) як засобу реалізації зв'язку (Розділи 1.2.2, 2.1);
3. Здійснено огляд та порівняння існуючих рішень і методологій для реалізації обміну даними між ЦСП/мікроконтролерами та зовнішніми хост-системами через USB, що дозволило обґрунтувати вибір підходу з використанням зовнішнього USB-UART моста (Розділ 1.2.3);
4. Спроєктовано алгоритмічну модель управління зовнішніми периферійними пристроями, керованими мікроконтролером, через комунікаційний канал, опосередкований USB. В рамках моделі розроблено структуру взаємодії та спеціалізований протокол обміну даними, що включає формат пакетів, набір команд та механізми їх обробки (Розділи 2.2, 2.3);
5. Обрано та обґрунтовано вибір апаратних (плата на базі ATmega328P з USB-UART CH340C) та програмних (Arduino IDE для мікроконтролеру, Python з бібліотекою PySerial для ПК) засобів, необхідних для взаємодії (Розділ 2.1);

6. Розроблено та детально описано організацію механізмів обміну даними в рамках запропонованої моделі, охоплюючи передачу команд від ПК до мікроконтролеру та отримання даних/статусів від мікроконтролеру до ПК відповідно до розробленого протоколу (Розділи 2.3, 2.4);
7. Реалізовано програмні компоненти алгоритмічної моделі: управляючу програму (прошивку) для мікроконтролера ATmega328P, що обробляє команди та управляє зовнішніми об’єктами, та керуючу програму для ПК, що формує команди, надсилає їх та обробляє відповіді (Розділ 2.4);
8. Проведено систематичне тестування та налагодження реалізованої системи, проаналізовано її продуктивність, оцінено ефективність та надійність обміну даними, виявлено та проаналізовано обмеження (Розділи 3.1, 3.2);
9. Розроблено документацію у вигляді даної пояснювальної записки, що включає опис предметної області, постановку задачі, теоретичні основи, детальне проєктування алгоритмічної моделі та протоколу, опис програмної реалізації та результати експериментальної перевірки.

Таким чином, мета кваліфікаційної роботи, яка полягала в розробці і реалізації алгоритмічної моделі управління зовнішніми об’єктами цифрового сигнального процесора (реалізованого на базі мікроконтролера) за допомогою USB-інтерфейсу, вважається досягнутою.

Це підтверджується такими результатами:

- Розроблено комплексну алгоритмічну модель, що структурує процес управління зовнішніми об’єктами через USB-UART перетворювач, описуючи рівні взаємодії, архітектуру системи та логіку обробки інформації;
- Створено спеціалізований, гнучкий та розширюваний протокол обміну даними між ПК та мікроконтролером, який забезпечує

передачу команд управління, отримання даних від зовнішніх пристроїв та враховує обмежені ресурси мікроконтролера;

- Здійснено програмну реалізацію розробленої моделі та протоколу у вигляді управляючої програми для мікроконтролера ATmega328P та керуючої програми на мові Python для ПК;
- Експериментальна перевірка на реальному тестовому стенді підтвердила працездатність системи, коректність виконання команд та обміну даними. Система дозволяє ефективно управляти зовнішніми об’єктами (світлодіодним модулем, датчиком температури та вологості) та взаємодіяти з енергонезалежною пам'яттю мікроконтролера.

Висновок за розділом 3

У третьому розділі проведено експериментальну перевірку розробленої системи та аналіз отриманих результатів. Експерименти підтвердили досягнення мети кваліфікаційної роботи: розроблено та практично реалізовано алгоритмічну модель управління зовнішніми об’єктами мікроконтролера (в ролі ЦСП) через USB-UART перетворювач. Усі поставлені завдання було успішно вирішено.

Отримані результати загалом відповідають теоретичним очікуванням, демонструючи працездатність та ефективність розробленої алгоритмічної моделі й протоколу обміну даними для задач управління та моніторингу. Позитивним є стабільне функціонування системи, що забезпечує надійний обмін командами та даними між ПК та мікроконтролером. Розроблений протокол дозволяє гнучко керувати зовнішніми пристроями, отримувати телеметричну інформацію та взаємодіяти з енергонезалежною пам'яттю. Застосування USB-UART мосту CH340C значно спростило реалізацію USB-комунікації, дозволивши зосередитись на логіці прикладного рівня.

Водночас виявлено певні обмеження. По-перше, поточна реалізація управляючої програми на мікроконтролері не має механізму таймауту очікування повного пакету, що може призвести до “зависання” логіки прийому

у випадку отримання неповного пакету від хоста. Це є наслідком свідомого спрощення обробника UART, однак для більш надійних систем це обмеження слід усунути. По-друге, пропускна здатність каналу, обмежена швидкістю UART 9600 біт/с, є достатньою для поточних завдань, але може стати вузьким місцем для систем з великими обсягами даних у реальному часі, хоча апаратні компоненти підтримують вищі швидкості. По-третє, використання ATmega328P як ЦСП є умовним, оскільки для складних задач цифрової обробки сигналів перевагу слід віддавати спеціалізованим процесорам.

Обраний підхід із використанням USB-UART мостів, розробкою власних протоколів та застосуванням високорівневих мов для хост-додатків цілком відповідає сучасним світовим тенденціям у розробці вбудованих систем.

Результати роботи мають практичну цінність та можуть знайти застосування в освітній сфері, науково-дослідній діяльності, розробці систем “розумного будинку”, прототипуванні промислових контролерів, аматорській робототехніці та як інструменти для відлагодження вбудованих систем.

Господарська значущість роботи полягає у потенційному зниженні часу та витрат на розробку систем взаємодії ПК з мікроконтролерними пристроями завдяки запропонованій готовій моделі, протоколу та використанню доступної елементної бази. Наукова значущість визначається розробкою комплексної алгоритмічної моделі та спеціалізованого, гнучкого протоколу, що систематизує підхід до проєктування подібних систем і може слугувати основою для подальших досліджень. Соціальна значущість проявляється у розширенні можливостей для технічної творчості, підтримці освітнього процесу та потенційному використанні у створенні допоміжних технологій.

Таким чином, у третьому розділі продемонстровано успішну експериментальну перевірку розробленої алгоритмічної моделі, підтверджено її працездатність, проаналізовано результати та визначено значущість виконаної роботи. Виявлені обмеження вказують на можливі напрямки подальшого вдосконалення системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно розроблено та експериментально перевірено алгоритмічну модель управління зовнішніми об’єктами цифрового сигнального процесора за допомогою USB-інтерфейсу. В ролі ЦСП виступав мікроконтролер ATmega328P на платформі Arduino Uno, а зв'язок з персональним комп'ютером реалізовувався через USB-UART перетворювач CH340C.

Особливо цінні моменти розробки:

- Розробка рішення: Створено завершену систему, що охоплює всі рівні взаємодії: від формування команд на ПК до безпосереднього управління апаратними ресурсами мікроконтролера та збору даних з сенсорів. Це включає програмне забезпечення для ПК (керуюча програма) та прошивку для мікроконтролера (управляюча програма);
- Створення спеціалізованого протоколу обміну даними: Розроблено власний, простий і гнучкий протокол пакетної передачі даних. Його ключовими особливостями є:
  - Компактність: Однобайтовий заголовок, що містить довжину поля даних та код команди/відповіді, мінімізує накладні витрати;
  - Ефективність для мікроконтролера: Максимальний розмір поля даних у 14 байт дозволяє ефективно використовувати обмежені буфери мікроконтролера та формувати пакети, що добре узгоджуються з розмірами USB-фреймів USB-UART мосту;
  - Розширюваність: 4-бітне поле команди дозволяє до 16 команд/відповідей, з можливістю подальшого розширення;
  - Наявність контрольної суми: Забезпечує базову перевірку цілісності даних;
- Практична реалізація та обґрунтування підходу з USB-UART: Робота наочно демонструє ефективність використання поширених USB-UART мостів (як CH340C) для організації зв'язку між ПК та

мікроконтролерними системами. Це дозволяє абстрагуватися від складнощів прямої реалізації USB-стеку на мікроконтролері, зосередившись на прикладній логіці;

- Системний підхід до проєктування: Проведено декомпозицію задачі на функціональні рівні (ПК, канал зв'язку, мікроконтролер, зовнішні об'єкти), що сприяло чіткій структуризації розробки та розумінню взаємозв'язків у системі;
- Документування та аналіз обмежень: Проведено не лише розробку та тестування, але й аналіз отриманих результатів, включаючи обмеження моделі та протоколу (пропускна здатність, синхронність, обробка помилок), що є важливим для розуміння сфери застосування та потенційних напрямків розвитку.

Міркування щодо продовження та вдосконалення рішення

Якби виникла потреба у подальшому розвитку та вдосконаленні розробленого рішення, можна було б розглянути такі напрямки:

- Підвищення надійності протоколу:
  - Для запобігання “зависанню” логіки прийому у випадку отримання неповних пакетів від хоста додати механізм таймауту при очікуванні повного пакету після отримання заголовка;
  - Для критично важливих даних можна додати нумерацію пакетів для виявлення втрат та механізми підтвердження доставки на рівні прикладного протоколу;
  - Розглянути використання більш стійких алгоритмів контрольної суми (наприклад CRC) замість простої XOR-суми для кращого виявлення помилок, особливо при передачі більших обсягів даних або в умовах сильних завад;
- Збільшення пропускної здатності та ефективності:
  - Перехід на використання вищих швидкостей UART: Апаратні компоненти (ATmega328P та CH340C) підтримують значно вищі швидкості UART (до 2 Мбіт/с);

- Додавання можливості асинхронної відправки повідомлень від мікроконтролера до ПК (наприклад, при виникненні певних подій на пристрої) без попереднього запиту від хоста;

- Розширення функціональності:
  - Підтримка складніших типів даних: Розширити протокол для зручної передачі структурованих даних, рядків змінної довжини, масивів тощо;
  - Розробка графічного інтерфейсу користувача для ПК: Замість консольної програми розробити інтуїтивно зрозумілий графічний інтерфейс для зручного управління та візуалізації даних;
- Адаптація для спеціалізованих ЦСП:
  - Хоча робота фокусувалася на мікроконтролері в ролі ЦСП, розроблену алгоритмічну модель та принципи побудови протоколу можна адаптувати для взаємодії з більш потужними, спеціалізованими цифровими сигнальними процесорами, враховуючи їхні специфічні інтерфейси та можливості.

Ці вдосконалення дозволили б створити ще більш надійне, швидке та функціональне рішення для управління зовнішніми об’єктами через USB-інтерфейс, розширивши сфери його можливого застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yovits M. C. Advances in Computers. Vol. 37. Indianapolis : Academic Press, 1993. 453 p.
2. Smith S. W. The Scientist and Engineer's Guide to Digital Signal Processing. San Diego, California : California Technical Publishing, 1997. [Електронний ресурс]. – режим доступу: URL: https://www.dspguide.com/ (дата звернення: 11.11.2024).
3. A Beginner's Guide to Digital Signal Processing (DSP). Analog Devices. [Електронний ресурс]. – режим доступу: URL: https://www.analog.com/en/lp/001/beginners-guide-to-dsp (дата звернення: 14.11.2025).
4. Noergaard T. Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers. Burlington : Elsevier/Newnes, 2005. 496 p.
5. Universal Serial Bus 4 (USB4®) Specification Version 2.0 with Errata and ECN through September 25, 2024. USB Promoter Group, 2024. 833 p. [Електронний ресурс]. – режим доступу: URL: https://www.usb.org/document-library/usb4r-specification-v20 (дата звернення: 24.01.2025).
6. Murphy R. USB 101: An Introduction to Universal Serial Bus 2.0. Infineon Technologies AG, 2021. 57 p.
7. USB Protocol: Types of USB Packets and USB Transfers. Engineers Garage. [Електронний ресурс]. – режим доступу: URL: https://www.engineersgarage.com/usb-protocol-types-of-usb-packets-and-usb-transfers-part-2-6/ (дата звернення: 26.01.2025).
8. Defined Class Codes. USB Implementers Forum. [Електронний ресурс]. – режим доступу: URL: https://www.usb.org/defined-class-codes (дата звернення: 27.01.2025).

9. Axelson J. Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems. 2nd ed. Madison, WI : Lakeview Research LLC, 2007. 400 p.
10. V-USB - Опис бібліотеки V-USB. Objective Development. [Електронний ресурс]. – режим доступу: URL: https://www.obdev.at/products/vusb/index.html (дата звернення: 03.02.2025).
11. USB to Serial Port Chip CH340 Datasheet. Ver. 3B. WCH. 15 p.
12. FT232R USB UART IC Datasheet. Ver. 2.16. Future Technology Devices International Ltd, 2020. 40 p.
13. Kotvytskyi A. Intellectual capital as the basis of innovative development of robotic systems. Karlsruhe, 2019. 150 p.
14. ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash Datasheet. Atmel, 2014. 294 p.
15. Arduino UNO R3 Product Reference Manual. Arduino S.r.l., 2024. 14 p.
16. Документація pySerial. pythonhosted.org. [Електронний ресурс]. – режим доступу: URL: https://pythonhosted.org/pyserial/ (дата звернення: 15.04.2025).
17. KY-015 Temperature and humidity sensor. JOY-IT, 2020. 7 p.
18. DHT11 Humidity & Temperature Sensor. Mouser Electronics. 9 p.

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма: «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бородін Єгор Владиславович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **"Алгоритмічна модель управління зовнішніми об'єктами цифрового сигнального процесора за допомогою USB-інтерфейсу"**

керівник роботи **Рева Сергій Миколайович, кандидат технічних наук.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***16 квітня 2025 року № 4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити
1. Аналіз архітектури та принципів роботи цифрових сигнальних процесорів (ЦСП).
2. Дослідження USB-інтерфейсу: специфікації, протоколи передачі даних, стандарти USB.
3. Огляд існуючих рішень для реалізації обміну даними з ЦСП через USB-інтерфейс.
4. Проєктування алгоритмічної моделі управління зовнішніми пристроями через USB-інтерфейс.
5. Вибір апаратного драйвера для взаємодії ЦСП із USB.
6. Організація управління зовнішніми пристроями ЦСП через USB-інтерфейс.
7. Реалізація програмного драйвера для роботи з USB-інтерфейсом.
8. Тестування роботи драйвера, його відлагодження, оптимізація та аналіз продуктивності.
9. Написання документації та інструкцій з використання драйвера.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи	05.09.2024 - 02.10.2024
2	Вивчення літератури з основ ЦСП, USB-інтерфейсу, драйверів та їх реалізації.	02.10.2024 - 31.12.2024
3	Дослідження сучасних підходів взаємодії з ЦСП через USB-інтерфейс.	01.01.2025 - 01.02.2025
4	Вибір апаратної платформи та інструментів розробки.	02.02.2025 - 08.02.2025
5	Проєктування алгоритмічної моделі управління зовнішніми пристроями через USB.	09.02.2025 - 01.03.2025
6	Вибір апаратного драйвера для взаємодії ЦСП із USB-інтерфейсом.	02.03.2025 - 30.03.2025
7	Реалізація програмного драйвера для роботи з USB-інтерфейсом.	31.03.2025 - 13.04.2025
8	Тестування працездатності драйвера та усунення помилок.	14.04.2025 - 20.04.2025
9	Написання документації та інструкцій щодо використання драйвера.	21.04.2025 - 30.04.2025
10	Оформлення пояснювальної записки відповідно до вимог.	01.05.2025 - 30.05.2025
11	Представлення кваліфікаційної роботи керівнику та рецензенту.	30.05.2025-

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент **Бородін Є. В.** 
ініціали, прізвище підпис

Керівник роботи **Рева С. М.** 
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання
на розробку програмного виробу «Алгоритмічна модель управління зовнішніми об’єктами цифрового сигнального процесора за допомогою USB-інтерфейсу»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: «Алгоритмічна модель управління зовнішніми об’єктами цифрового сигнального процесора за допомогою USB-інтерфейсу». 1.2. Галузь застосування: Вбудовані системи, інформаційні технології, системи збору даних, автоматизація експериментів, навчальні стенди, розробка прототипів пристроїв з комп'ютерним управлінням.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп’ютерна інженерія. 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/962 від 16 квітня 2025 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	3.1. Мета розробки програмного виробу: Створення програмних засобів на основі розробленої алгоритмічної моделі для забезпечення управління зовнішніми об’єктами, підключеними до мікроконтролера (що виконує функції ЦСП), з персонального комп’ютера через USB-інтерфейс з використанням USB-UART перетворювача. 3.2. Призначення програмного виробу: Керування зовнішніми пристроями та ресурсами ЦСП через USB-інтерфейс. 3.3. Вихідні дані для розробки: Специфікації ЦСП, деталі зовнішніх об’єктів, стандарти USB-зв’язку, вимоги до протоколів управління та обміну даними.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: - Встановлення та підтримка USB-з’єднання через міст USB-UART. - Надсилання команд управління з хост-комп’ютера до ЦСП для керування зовнішніми об’єктами. - Отримання та обробка даних від ЦСП, включаючи показання датчиків.

	4.2. Вимоги до надійності: - Перевірка цілісності даних (XOR-сума). - Обробка помилок протоколу (невірна сума/команда, недостатньо даних). - Стабільний зв'язок через USB-UART (9600 бод). 4.3. Вимоги до умов експлуатації: Стандартні умови експлуатації для персонального комп'ютера та мікроконтролерної плати в лабораторних або офісних умовах. 4.4. Вимоги до складу і параметрів технічних засобів: Персональний комп'ютер з USB-портом та плата мікроконтролера (наприклад, Arduino Uno) з мостом USB-UART (CH340C), підключена до зовнішніх об'єктів, таких як модуль світлодіодів та датчик температури/вологості. 4.5. Вимоги до інформаційної та програмної сумісності: Програмне забезпечення повинно бути сумісним з операційними системами Windows та дотримуватися стандартних протоколів USB-зв'язку через міст USB-UART. Прошивка повинна бути сумісною з архітектурою мікроконтролера AVR. 4.6. Вимоги до маркування та упаковки: Відсутні. 4.7. Вимоги до транспортування і зберігання: Відсутні. 4.8. Спеціальні вимоги: Відсутні.
5. Вимоги до програмної документації	Програмною документацією до виробу «Алгоритмічна модель управління зовнішніми об'єктами цифрового сигнального процесора за допомогою USB-інтерфейсу» вважати: 1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи). 3) Опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи). 4) Текст програми виробу (представити у вигляді Додатку Г до пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	В даному розділі можуть бути представлені: 1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)

7. Стадії і етапи розробки	Дата	Назва етапу
	02.10.2024 - 31.12.2024	Вивчення літератури з основ ЦСП, USB-інтерфейсу, драйверів та їх реалізації.
	01.01.2025 - 01.02.2025	Дослідження сучасних підходів взаємодії з ЦСП через USB-інтерфейс.
	02.02.2025 - 08.02.2025	Вибір апаратної платформи та інструментів розробки.
	09.02.2025 - 01.03.2025	Проєктування алгоритмічної моделі управління зовнішніми пристроями через USB.
	02.03.2025 - 30.03.2025	Вибір апаратного драйвера для взаємодії ЦСП із USB-інтерфейсом.
	31.03.2025 - 13.04.2025	Реалізація програмного драйвера для роботи з USB-інтерфейсом.
	14.04.2025 - 20.04.2025	Тестування працездатності драйвера та усунення помилок.
	21.04.2025 - 30.04.2025	Написання документації та інструкцій щодо використання драйвера.
	01.05.2025 - 30.05.2025	Оформлення пояснювальної записки відповідно до вимог.
	30.05.2025-	Представлення кваліфікаційної роботи керівнику та рецензенту.
8. Порядок контролю і приймання	1) Перевірку ходу розробки програмного виробу керівнику робіт виконувати раз на 3 тижні. 2) Випробування програмного виробу провести відповідно до програми та методики випробувань на базі комп'ютерного класу. 3) Захист розробленої моделі провести на засіданні Атестаційної комісії. 4) Пояснювальну записку подати в електронному вигляді в одному примірнику.	

Виконавець
студент групи КІ- 41
БОРОДІН Є. В.
_____

Замовник
к. т. н., доцент.
РЕВА С. М.
_____

Додаток В

Програма і методика випробувань програмного виробу

«АЛГОРИТМІЧНА МОДЕЛЬ УПРАВЛІННЯ ЗОВНІШНІМИ ОБ'ЄКТАМИ ЦИФРОВОГО СИГНАЛЬНОГО ПРОЦЕСОРА ЗА ДОПОМОГОЮ USB-ІНТЕРФЕЙСУ»

1. Об'єкт випробувань

1.1. Назва програмного виробу: «Алгоритмічна модель управління зовнішніми об'єктами цифрового сигнального процесора за допомогою USB-інтерфейсу».

1.2. Галузь застосування: інформаційні технології, будовані системи, системи збору даних, автоматизація експериментів, навчальні стенди, розробка прототипів пристроїв з комп'ютерним управлінням.

1.3. Умовне позначення розробки: Не застосовується.

2. Мета випробувань

Підтвердження відповідності функціональних та інших характеристик розробленого програмного виробу (керуючої програми для ПК та управляючої програми для мікроконтролера) вимогам, які сформульовані в Технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань:

Підставою для проведення випробувань є завдання на кваліфікаційну роботу та необхідність перевірки працездатності розробленого програмного виробу перед захистом кваліфікаційної роботи в рамках атестації.

3.2. Місце і тривалість випробувань

Приймальні випробування проводяться на базі комп'ютерного класу кафедри або з використанням персонального комп'ютера розробника та зібраного тестового стенду в період підготовки до захисту кваліфікаційної роботи та під час роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі, відповідному до цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться розробником (студентом) під контролем керівника кваліфікаційної роботи та можуть бути продемонстровані атестаційній комісії.

4. Вимоги до програми або програмного виробу

Програмний виріб (керуюча програма для ПК та управляюча програма для мікроконтролера) повинен задовольняти наступним основним вимогам, що підлягають перевірці:

1. Встановлення та підтримка USB-з’єднання через міст USB-UART (CH340C) між ПК та мікроконтролером ATmega328P;
2. Надсилання команд управління з хост-комп’ютера (ПК) до мікроконтролера для керування зовнішніми об’єктами (світлодіодний модуль, датчик температури/вологості, пам'ять EEPROM) відповідно до розробленого протоколу;
3. Отримання та обробка даних від мікроконтролера на ПК, включаючи статуси, показання датчиків та дані з EEPROM, відповідно до розробленого протоколу;
4. Перевірка цілісності даних за допомогою XOR-суми, реалізованої в протоколі;
5. Обробка помилок протоколу, таких як невірна контрольна сума, невідома команда, недостатня кількість даних для команди;
6. Стабільний обмін даними через UART на швидкості 9600 бод (або іншій узгодженій);
7. Коректне виконання команд мікроконтролером.

5. Вимоги до програмної документації

Склад програмної документації, що подається на випробування, включає в себе:

1. Технічне завдання на розробку програмного виробу (представлено у Додатку Б до пояснювальної записки до кваліфікаційної роботи);
2. Ця Програма і методика випробувань розробленого програмного виробу (представлена у Додатку В і розділі 3 до пояснювальної записки до кваліфікаційної роботи);
3. Опис програмного виробу (представлено в розділі 3 пояснювальної записки до кваліфікаційної роботи);
4. Текст програм (представлений у Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Технічні засоби:

1. Персональний комп’ютер з USB-портом;
2. Плата мікроконтролера, функціональний аналог Arduino Uno (на базі ATmega328P та USB-UART перетворювача CH340C);
3. Кабель USB і з’єднувальні проводи для зовнішніх об’єктів;
4. Зовнішні об’єкти: світлодіодний модуль HW-479, датчик температури та вологості KY-015 (DHT11).

Програмні засоби:

1. Операційна система Windows на ПК;
2. Встановлений інтерпретатор Python (версії, що підтримує PySerial);
3. Встановлена бібліотека PySerial;
4. Встановлений драйвер для USB-UART перетворювача CH340C;
5. Середовище розробки Arduino IDE (для завантаження прошивки);
6. Розроблена управляюча програма (прошивка) для ATmega328P;
7. Розроблена керуюча програма на Python для ПК;
8. Програма-аналізатор трафіку Wireshark з USBPcap.

6.2. Порядок проведення випробувань

Випробування проводяться в два етапи: ознайомчий етап (1-й етап) та власне випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації;
2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації;
3. Перевірку комплектності складу технічних і програмних засобів;
4. Методику проведення перевірок на 1 етапі випробувань.

Власне випробування програмного виробу (2-й етап):

1. Перевірка відповідності технічних характеристик програми вимогам технічного завдання: Перевіряється стабільність USB-UART з'єднання, відсутність непередбачених відключень, відповідність швидкості обміну (9600 бод), коректність обробки сигналів керування зовнішніми об'єктами;
2. Перевірка ступеня виконання функціональних вимог до програми: Виконується шляхом послідовної перевірки роботи кожної команди протоколу та відповідних функцій програмного виробу;
3. Методика проведення перевірок, що входять до переліку за 2-м етапом випробувань:
   3.1. Програмний виріб (керуюча програма на ПК та управляюча програма на мікроконтролері) повинен працювати відповідно до умов експлуатації, зазначених у ТЗ, та на апаратному забезпеченні;
   3.2. Порядок проведення випробувань:
   - Зібрати тестовий стенд: підключити модуль світлодіодів HW-479 та датчик KY-015 до плати Arduino Uno;
   - Підключити плату Arduino Uno до USB-порту ПК;

- Переконатися, що операційна система ПК розпізнала пристрій і віртуальний COM-порт;
- Запустити розроблену керуючу програму на Python на ПК. У програмі вказати правильний COM-порт, якщо це необхідно;
- Провести серію тестів, описаних нижче. Фіксувати відправлені команди, параметри та отримані відповіді, а також поведінку зовнішніх об'єктів.

4. Для проведення випробувань пропонуються три тести:
   4.1. Перевірка формування пакетів та інкапсуляції протоколу в USB;
   4.2. Перевірка обробки функціональних команд мікроконтролером:
   - Запит поточного статусу пристрою;
   - Зміна стану виводів порту B (керування світлодіодами);
   - Запит поточних даних з датчика температури та вологості;
   - Запит даних історії показань датчика з EEPROM;
   - Запит на очищення збережених даних датчика з пам’яті EEPROM.

   4.3. Перевірка обробки помилок: неправильна контрольна сума, невідома команда, недостатньо даних для команди.
5. Програмний виріб вважається таким, що пройшов випробування в цілому, якщо всі тести успішно пройдені, що підтверджує коректну реалізацію всіх заявлених функцій та механізмів обробки помилок. Програмна документація відповідає вимогам. Технічні та програмні засоби забезпечують стабільну роботу системи відповідно до вимог ТЗ. Не виявлено критичних помилок, що перешкоджають основному функціонуванню.

Тест 1: Перевірка формування пакетів та інкапсуляції протоколу в USB

За допомогою інструментальних засобів (керуюча програма на ПК, Wireshark з USBPcap) переконатися, що керуюча програма на ПК формує пакети команд відповідно до розробленого протоколу, управляюча програма на мікроконтролері коректно формує пакети-відповіді, і ці пакети правильно інкапсулюються для передачі через USB.

Спочатку надсилається команда “0” (запиту статусу) з ПК.

```
Спроба підключення до COM4 на швидкості 9600 бод.
З'єднано з COM4 на швидкості 9600 бод.
Використати CLI-інтерфейс? (y/n, за замовчуванням y): y
CLI-інтерфейс активовано.

Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 0
Відправка: 00
Отримано та розібрано пакет: 50 00 18 3D 08 01 7C
Отримано відповідь: Код=0x0, Дані='00 18 3D 08 01'
Статус: PORTB(0-5)=0b000000, Температура=24°C, Вологість=61%, EEPROM_ptr=264
```

Рисунок В.1 – Відправка і отримання пакету.

Як можна побачити на рисунку В.1, керуюча програма сформувала і відправила відповідний пакет “0x00”, а управляюча програма надала відповідь на отриману команду: пакет “0x5000183D08017C”, після чого керуюча програма проаналізувала отримані дані і надала їх у зручній для користувача формі. Пакети і їх опис було сформовано відповідно до створеного протоколу, описаного в розділі 2.3.

Тепер проаналізуємо ці пакети зі сторони USB за допомогою програми-аналізатора трафіку Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
11317	8783.424200	host	2.3.2	USB	28	URB_BULK out
11318	8783.424245	2.3.2	host	USB	27	URB_BULK out
11319	8783.468026	2.3.2	host	USB	34	URB_BULK in
11320	8783.468042	host	2.3.2	USB	27	URB_BULK in

Рисунок В.2 – Захоплені USB-фрейми.

Як можна побачити на рисунку В.2, пакети створеного протоколу на рівні USB мають вигляд 4 фреймів:

1. Передача команди від ПК до мікроконтролера. Тип передачі: *URB_BULK out* (Передача блокових даних назовні);
2. Підтвердження передачі команди;
3. Передача відповіді від мікроконтролера до ПК. Тип передачі: *URB_BULK in* (Передача блокових даних всередину);
4. Підтвердження прийому відповіді ПК.

Тепер проаналізуємо перший фрейм.

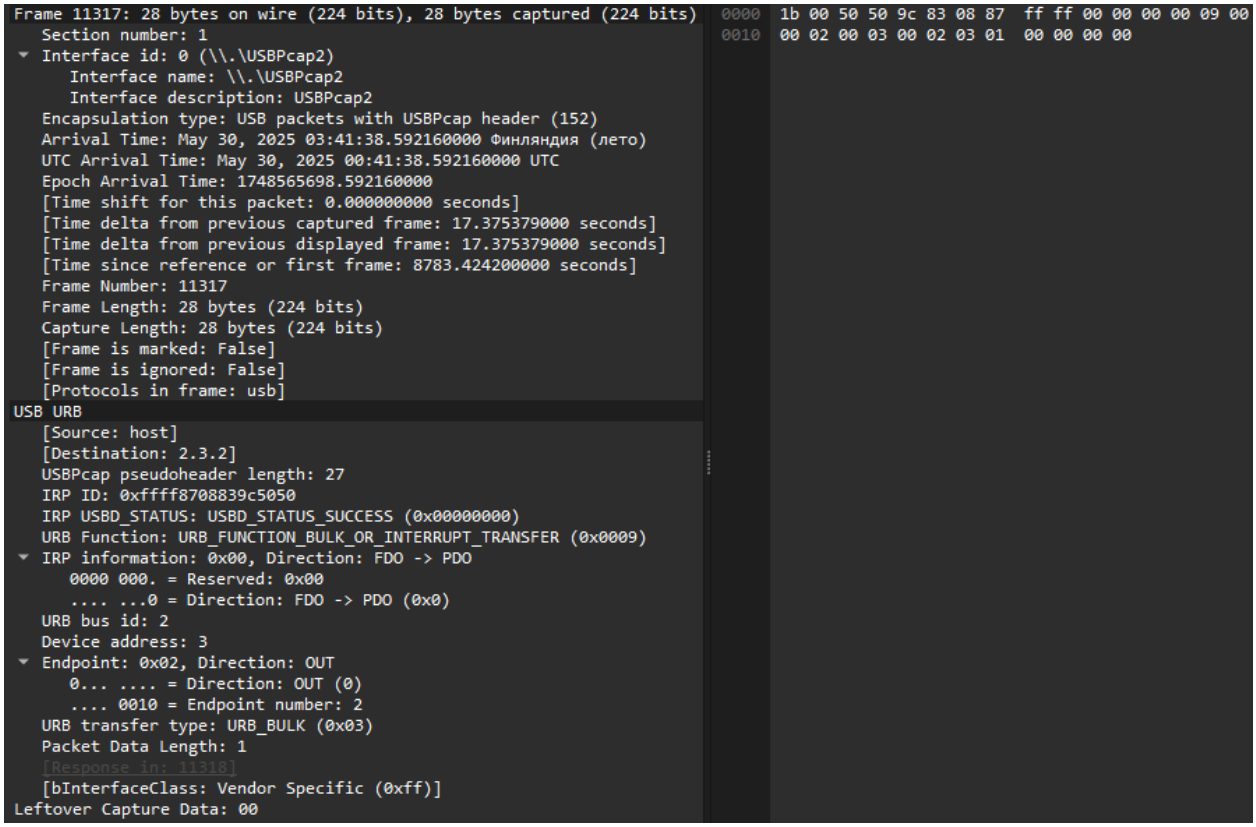

Рисунок В.3 – Вміст і опис першого фрейму.

Проаналізуємо зміст і опис першого фрейму з рисунку В.3. Цей фрейм демонструє, як операційна система ПК надсилає USB-транзакції типу передачі блокових даних назовні (*BULK OUT*) на адресу 0x0300, яка відповідає віртуальному COM-порту. Довжину даних пакету зазначено в 1 байт з вмістом 0x00, що відповідає відправленому з керуючої програми пакету.

Тепер проаналізуємо третій фрейм з відповіддю від мікроконтролера.

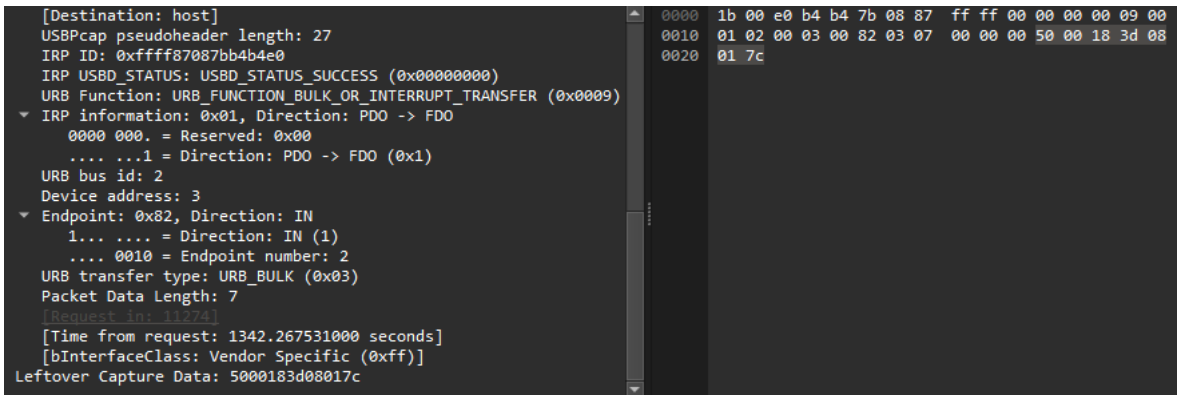

Рисунок В.4 – Вміст і опис третього фрейму.

Як можна побачити на рисунку В.4, в пакеті зазначено зворотній напрям блокових даних з 7 байтами даних: 0x5000183D08017C, що відповідають отриманому пакету даних в інтерфейсі керуючої програми на рисунку В.1.

Отже, аналіз підтверджує, що програми коректно формують пакети розробленого протоколу обміну даними і ці пакети коректно інкапсулюються у стандартні USB-передачі типу *BULK*.

Тест 2: Перевірка обробки команд мікроконтролером

Перевірити коректність виконання мікроконтролером основних функціональних команд, передбачених протоколом та перевірити коректність обробки отриманих пакетів-відповідей керуючою програмою на ПК.

Оскільки коректність роботи команди “0” (запиту статусу) було перевірено у першому тесті перейдемо до команди “1” (Зміна стану виходів порту B).

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 1
Введіть значення для PORTB (PB0-PB5) у шістнадцятковому форматі (напр. '0F'): 10
Відправка: 11 10 01
Отримано та розібрано пакет: 13 10 03
Отримано відповідь: Код=0x3, Дані='10'
Команду 1 (зміна PORTB) виконано. Новий стан PORTB(0-5): 0b010000
```

Рисунок В.5 – Зміна вмісту регістру PORTB.

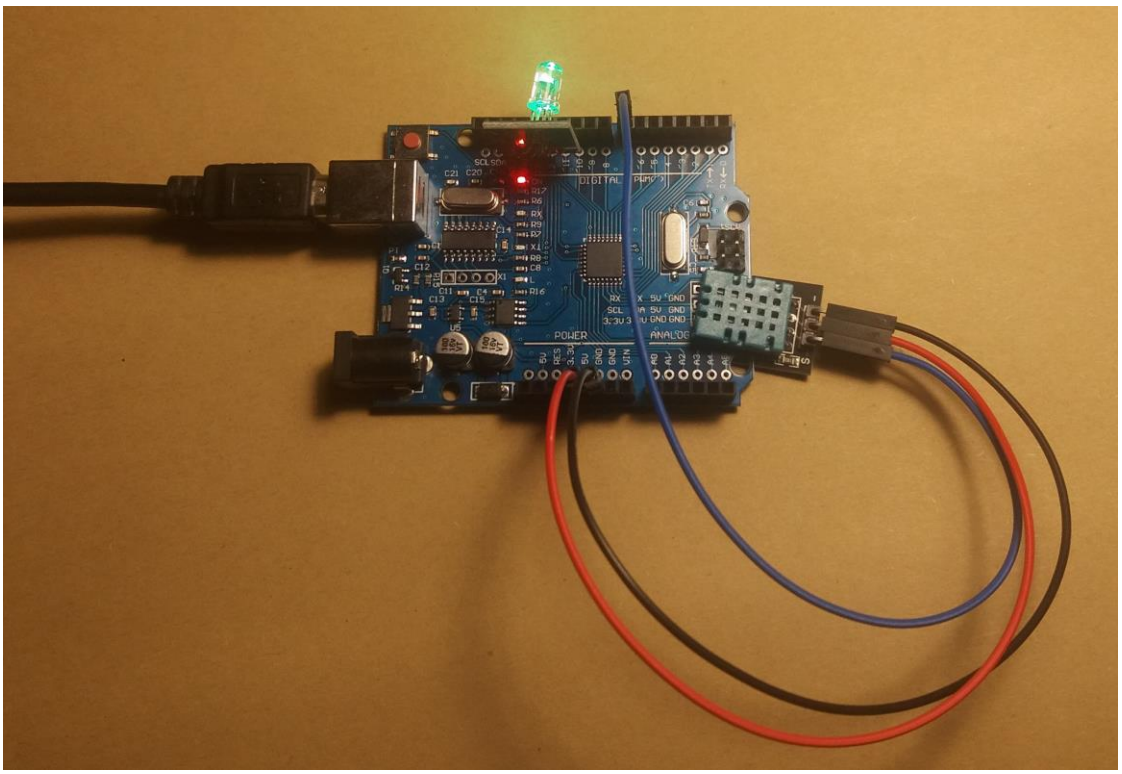

Рисунок В.6 – Результат зміни стану портів В.

Як можна побачити на рисунках В.5 і В.6, успішно відбулася зміна стану регістру PORTB на 0x010000, змінивши стан порту 12 (PB4), до якого доєднано зелений діод модуля HW-479.

Тепер перевіримо команду “2” (Запит поточних даних з датчика температури та вологості).

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 2
Відправка: 02
Отримано та розібрано пакет: 25 1A 3F 00
Отримано відповідь: Код=0x5, Дані='1A 3F'
Дані з датчика: Температура=26°C, Вологість=63%
```

Рисунок В.7 – Успішний запит даних з датчика температури та вологості.

Далі перевіримо команду “3” (Запит даних історії показань датчика з EEPROM).

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 3
Відправка: 03
Очікування даних EEPROM...
Пакет 1: Дані='18 3B 18 3A 18 3A 18 3A 18 3A 18 3A 18 3A'
Пакет 2: Дані='18 39 18 38 19 36 1B 36 1B 36 1B 36 1B 36'
Пакет 3: Дані='1B 36 1B 36 1B 36 1B 36 1B 36 1B 36 1B 36'
.....................................................................
Пакет 67: Дані='1A 38 1A 38 1A 38 1A 38 1A 38 1A 38 1A 38'
Пакет 68: Дані='1A 38 1A 38 1A 38 1A 38 1A 38 1A 38 1A 38'
Пакет 69: Дані='1A 38 1A 38 1A 38 1A 38'
Дані з EEPROM (480 записів, 69 пакетів) збережено у файл 'eeprom_data.dat'
```

Рисунок В.8 – Запит на відправку 960 байт даних з EEPROM.

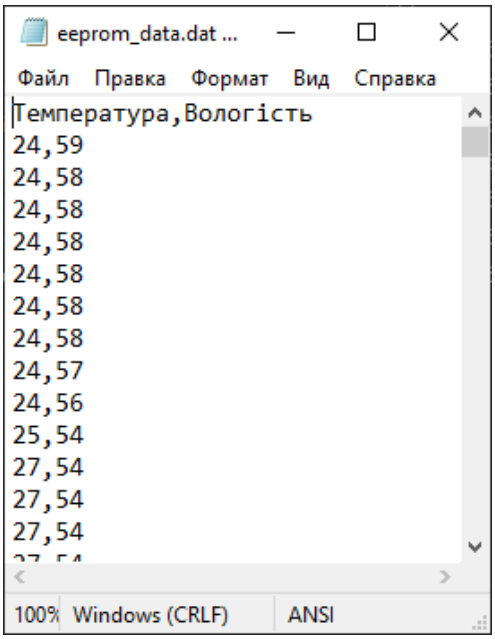

Рисунок В.9 – Успішно отримані і збережені дані.

Тепер перевіримо команду “4” (Запит на очищення збережених даних датчика з пам’яті EEPROM).

```
Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 4
Відправка: 04
Отримано та розібрано пакет: 08
Отримано відповідь: Код=0x8, Дані=''
Відповідь мікроконтролера: Дані з EEPROM успішно очищено.

Доступні команди:
  0 - Запит статусу пристрою
  1 - Зміна стану портів (PB0-PB5)
  2 - Запит поточних даних з датчика
  3 - Запит історії даних з EEPROM
  4 - Очищення даних в EEPROM
  exit - Вихід з програми
Ваш вибір: 3
Відправка: 03
Очікування даних EEPROM...
Отримано та розібрано пакет: 07
Мікроконтролер повідомив, що в EEPROM відсутні дані.
```

Рисунок В.10 – Перевірка роботи при видаленні даних з EEPROM.

Отже, аналіз підтверджує, що управляюча програма коректно виконує всі основні функціональні команди, а керуюча програма коректно обробляє пакети-відповіді на ці команди.

Тест 3: Перевірка обробки помилок

Перевірити здатність системи коректно обробляти помилкові ситуації, передбачені протоколом.

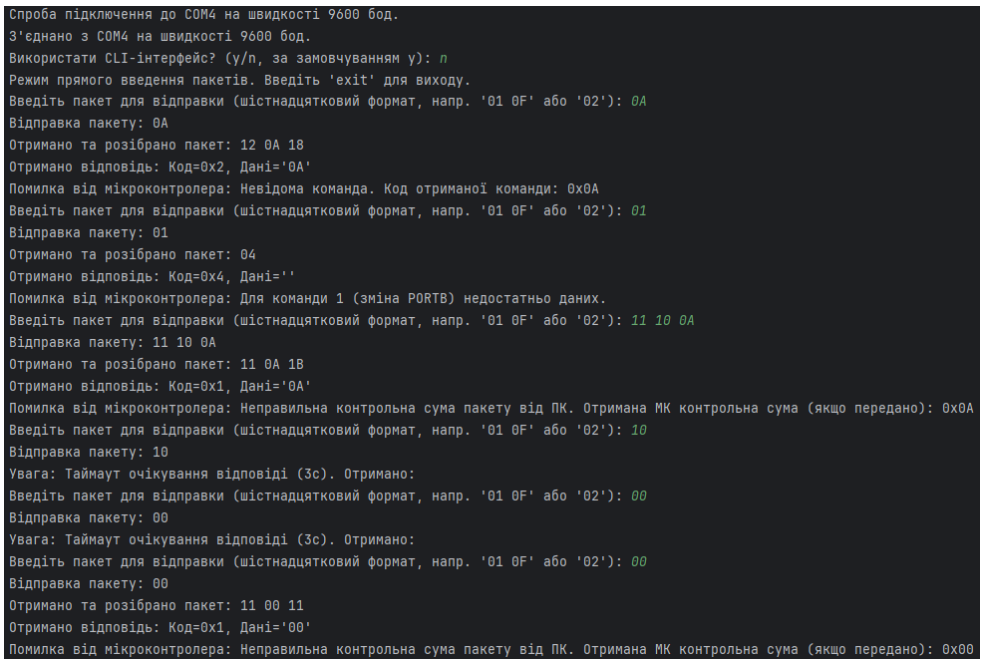

Рисунок В.11 – Перевірка роботи при відправці некоректних пакетів.

Як можна побачити на рисунку В.11 більшість некоректних пакетів система обробляє коректно. Однак в ситуації, коли ПК надсилає заголовок, що вказує на наявність даних, але потім не надсилає самі дані та контрольну суму (або надсилає їх не повністю), мікроконтролер залишається в стані очікування цих байтів без таймауту. Такий недолік було залишено оскільки це є наслідком свідомого спрощення логіки обробника переривань UART для даної кваліфікаційної роботи, припускаючи низьку ймовірність такої непередбаченої поведінки в рамках тестового використання. Для більш надійних систем, призначених для реальної експлуатації, цей аспект потребував би доопрацювання шляхом впровадження механізму таймауту очікування повного пакету. Незважаючи на вказаний недолік, основні механізми обробки визначених помилок протоколу функціонують правильно.

Загальний висновок за результатами випробувань

Проведені випробування показали, що в цілому, програмний виріб функціонує відповідно до основних вимог Технічного завдання, реалізує ключові аспекти розробленої алгоритмічної моделі і може вважатися таким, що пройшов випробування, з урахуванням зазначеного обмеження.

Виконавець: студент групи КІ-41, Бородін Є.В. _____

Додаток Г

Лістинг Г.1 – Управляюча програма

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <util/delay.h>
#include <EEPROM.h>

// Піни RGB світлодіода
#define RED_PIN PB5
#define GREEN_PIN PB4
#define BLUE_PIN PB3

#define MAX_PACKET_SIZE 16 // Макс. розмір пакету UART

// Піни та регістри для датчика DHT11
#define DHT11_PIN PD7
#define DHT11_DDR DDRD
#define DHT11_PORT PORTD
#define DHT11_PIN_IN PIND

// Коди стану для DHT11
#define DHT11_SUCCESS 0
#define DHT11_ERROR_TIMEOUT 1
#define DHT11_ERROR_CHECKSUM 2

// Глобальні змінні
volatile uint16_t seconds_counter = 0;          // Лічильник для
 інтервалів збереження в EEPROM
volatile uint8_t rx_buffer[MAX_PACKET_SIZE]; // Буфер прийому
 пакету з UART
volatile uint8_t rx_index = 0;                // Індекс для запису
 в rx_buffer
volatile bool packet_ready = 0;             // Прапорець готовності
 пакету
volatile bool time_to_save_data = false;       // Прапорець для
 збереження даних DHT11 в EEPROM

// Ініціалізація Таймера1 для періодичного збереження даних
void Timer1_init() {
    TCCR1B |= (1 << WGM12);                 // Режим CTC (OCR1A -
 TOP)
    TCCR1B |= (1 << CS12) | (1 << CS10); // Дільник 1024 (F_timer
 = 15625 Гц)
    OCR1A = 62499;                              // Порівняння для
 переривання кожні 4с ((62499+1)/15625)
```

```
    TIMSK1 |= (1 << OCIE1A);              // Дозвіл переривання по
 співпадінню OCR1A
}

// Ініціалізація UART (9600 бод, 8N1)
void UART_init() {
    UCSR0A |= (1 << U2X0);              // Подвійна швидкість UART
    UBRR0H = 0;
    UBRR0L = 207;                        // UBRR для 9600 бод при
 F_CPU=16MHz, U2X0=1
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0); // Дозвіл прийому та
 передачі
    UCSR0B |= (1 << RXCIE0);             // Дозвіл переривання по
 завершенню прийому
}

// Відправка одного байта через UART
void UART_send_byte(uint8_t byte) {
    while (!(UCSR0A & (1 << UDRE0))); // Очікування порожнього
 буфера передавача
    UDR0 = byte;                         // Запис байта в буфер для
 відправки
}

// Формування та відправка пакету-відповіді на ПК
void send_response_packet(uint8_t response_code, uint8_t*
 payload, uint8_t payload_len) {
    if (payload_len > 14) payload_len = 14; // Обмеження довжини
 даних

    uint8_t packet[MAX_PACKET_SIZE];
    uint8_t header = ((payload_len & 0x0F) << 4) | (response_code
 & 0x0F); // Формування заголовка
    packet[0] = header;

    if (payload_len == 0) {
        UART_send_byte(packet[0]); // Відправка тільки заголовка
    } else {
        for (uint8_t i = 0; i < payload_len; i++) { // Копіювання
 даних
            packet[1 + i] = payload[i];
        }
        uint8_t checksum = calculate_checksum(packet, 1 +
 payload_len); // Обчислення КС
        packet[1 + payload_len] = checksum;

        for (uint8_t i = 0; i < (2 + payload_len); i++) { //
 Відправка всього пакету
            UART_send_byte(packet[i]);
        }
    }
}
```

```
// Зчитування одного байта даних від датчика DHT11
uint8_t dht11_read_byte() {
    uint8_t result = 0;
    for (uint8_t i = 0; i < 8; i++) { // Читання 8 біт
        uint8_t timeout = 0;
        while (!(DHT11_PIN_IN & (1 << DHT11_PIN))) { // Очікування
 HIGH (початок біта)
            if (timeout++ > 200) return 0xFF; // Таймаут
            _delay_us(1);
        }
        _delay_us(30); // Затримка для розрізнення '0'/'1'
        if (DHT11_PIN_IN & (1 << DHT11_PIN)) { // Якщо HIGH -> '1'
            result |= (1 << (7 - i));
            timeout = 0;
            while (DHT11_PIN_IN & (1 << DHT11_PIN)) { //
 Очікування LOW (кінець біта '1')
                if (timeout++ > 200) return 0xFF; // Таймаут
                _delay_us(1);
            }
        }
    }
    return result;
}

// Зчитування повних даних (температура, вологість, КС) з DHT11
int dht11_read_data(int *temperature, int *humidity) {
    uint8_t data[5]; // Буфер для 5 байтів: RH_int, RH_dec, T_int,
 T_dec, Checksum

    // Старт-сигнал від МК до DHT11
    DHT11_DDR |= (1 << DHT11_PIN);    // Пін на вихід
    DHT11_PORT &= ~(1 << DHT11_PIN);  // LOW
    _delay_ms(30);                    // Тримати LOW min 18ms
    DHT11_PORT |= (1 << DHT11_PIN);   // HIGH
    _delay_us(40);                    // Тримати HIGH 20-40us
    DHT11_DDR &= ~(1 << DHT11_PIN);   // Пін на вхід

    // Очікування відповіді від DHT11
    uint8_t timeout = 0;
    while (DHT11_PIN_IN & (1 << DHT11_PIN)) { // Очікування LOW
 від DHT11
        if (timeout++ > 200) return DHT11_ERROR_TIMEOUT;
        _delay_us(1);
    }
    _delay_us(80); // Пропуск LOW періоду відповіді DHT11 (~80us)
    timeout = 0;
    while (!(DHT11_PIN_IN & (1 << DHT11_PIN))) { // Очікування
 HIGH від DHT11
        if (timeout++ > 200) return DHT11_ERROR_TIMEOUT;
        _delay_us(1);
    }
    _delay_us(80); // Пропуск HIGH періоду відповіді DHT11 (~80us)
```

```
    // Зчитування 5 байтів даних
    for (uint8_t i = 0; i < 5; i++) {
        data[i] = dht11_read_byte();
        if (data[i] == 0xFF) { // Помилка читання байта
            DHT11_DDR |= (1 << DHT11_PIN);
            DHT11_PORT |= (1 << DHT11_PIN); // Відновлення стану
 піна
            return DHT11_ERROR_TIMEOUT;
        }
    }

    // Перевірка контрольної суми
    uint8_t checksum = data[0] + data[1] + data[2] + data[3];
    DHT11_DDR |= (1 << DHT11_PIN);    // Завершення сеансу, пін на
 вихід
    DHT11_PORT |= (1 << DHT11_PIN);   // Встановити HIGH

    if (checksum == data[4]) {
        *humidity = data[0];    // Ціла частина вологості
        *temperature = data[2]; // Ціла частина температури
        return DHT11_SUCCESS;
    } else {
        return DHT11_ERROR_CHECKSUM;
    }
}

// Ініціалізація RGB-світлодіода
void RGB_init() {
    DDRB |= (1 << RED_PIN) | (1 << GREEN_PIN) | (1 << BLUE_PIN);
 // Піни на вихід
    PORTB &= ~((1 << RED_PIN) | (1 << GREEN_PIN) | (1 <<
 BLUE_PIN)); // Початково вимкнути
}

// Обробка отриманої команди від ПК
void execute_command(uint8_t cmd, uint8_t* data, uint8_t len) {
    uint8_t payload[14]; // Буфер для даних відповіді
    int temperature = 0, humidity = 0;

    switch (cmd) {
        case 0x00: // Команда: Запит статусу
            if   (dht11_read_data(&temperature,   &humidity)   !=
 DHT11_SUCCESS) {
                temperature = 255; humidity = 255; // Індикація
 помилки DHT11
            }
            payload[0] = PORTB & 0x3F;              // Стан PB0-
 PB5
            payload[1] = (uint8_t)temperature;
            payload[2] = (uint8_t)humidity;
            payload[3] = EEPROM.read(0);             // Молодший
 байт вказівника EEPROM
```

```
            payload[4] = EEPROM.read(1);          // Старший байт
вказівника EEPROM
            send_response_packet(0x0, payload, 5); // Відповідь:
статус
            break;

        case 0x01: // Команда: Керування портами PB0-PB5
            if (len >= 1) {
                PORTB = (PORTB & 0xC0) | (data[0] & 0x3F); // Зміна
PB0-PB5
                payload[0] = PORTB & 0x3F;
                send_response_packet(0x3,   payload,   1);   //
Відповідь: успішно
            } else {
                send_response_packet(0x4,  NULL,  0);         //
Відповідь: помилка, недостатньо даних
            }
            break;

        case 0x02: // Команда: Запит поточних даних з DHT11
            if   (dht11_read_data(&temperature,   &humidity)   !=
DHT11_SUCCESS) {
                temperature = 255; humidity = 255; // Індикація
помилки
            }
            payload[0] = (uint8_t)temperature;
            payload[1] = (uint8_t)humidity;
            send_response_packet(0x5, payload, 2); // Відповідь:
дані з датчика
            break;

        case 0x03: { // Команда: Зчитування історії даних з EEPROM
            uint16_t  current_write_index  =  EEPROM.read(0)  |
(EEPROM.read(1) << 8);
            if (current_write_index >= 480) current_write_index =
0; // Циклічний буфер (480 записів)

            int read_pos_circular = (current_write_index == 0) ?
479  :  (current_write_index  -  1);  //  Читання  з  останнього
записаного
            uint8_t records_in_payload_count = 0;

            for     (int     records_to_read_total     =     0;
records_to_read_total < 480; records_to_read_total++) {
                int eeprom_physical_addr = 2 + read_pos_circular
* 2; // Адреса даних (після індексу)
                uint8_t             temp_from_eeprom             =
EEPROM.read(eeprom_physical_addr);
                uint8_t         hum_from_eeprom                  =
EEPROM.read(eeprom_physical_addr + 1);

                if (temp_from_eeprom == 0xFF && hum_from_eeprom ==
0xFF) { // Кінець даних (порожня комірка)
```

```
                    if (records_to_read_total == 0) { // Якщо
 EEPROM порожній
                        send_response_packet(0x7, NULL, 0); //
 Відповідь: немає даних
                    }
                    break;
                }
                payload[records_in_payload_count++]             =
 temp_from_eeprom;
                payload[records_in_payload_count++]             =
 hum_from_eeprom;

                if (records_in_payload_count == 14) { // Якщо
 буфер відповіді заповнений
                    send_response_packet(0x6, payload, 14); //
 Відповідь: дані з EEPROM
                    records_in_payload_count = 0;
                }
                if (read_pos_circular == 0) read_pos_circular =
 480;
                read_pos_circular--; // Перехід до попереднього
 запису
            }
            if (records_in_payload_count > 0) { // Відправка
 залишку
                send_response_packet(0x6,                payload,
 records_in_payload_count);
            }
            break;
        }

        case 0x04: { // Команда: Очищення EEPROM
            EEPROM.update(0, 0); EEPROM.update(1, 0); // Скидання
 вказівника
            for (int i = 0; i < 480; i++) { // Заповнення даними
 0xFF
                EEPROM.update(2 + i * 2, 0xFF);
                EEPROM.update(2 + i * 2 + 1, 0xFF);
            }
            send_response_packet(0x8, NULL, 0); // Відповідь:
 EEPROM очищено
            break;
        }

        default: // Невідома команда
            payload[0] = cmd;
            send_response_packet(0x2, payload, 1); // Відповідь:
 невідома команда
            break;
    }
}

// Обчислення контрольної суми пакету (XOR)
```

```
uint8_t calculate_checksum(uint8_t* packet, uint8_t len) {
    uint8_t checksum_val = 0;
    for (uint8_t i = 0; i < len; i++) { // XOR для заголовка та
 даних
        checksum_val ^= packet[i];
    }
    return checksum_val;
}

// Обробник переривання по завершенню прийому байта UART
ISR(USART_RX_vect) {
    uint8_t byte = UDR0; // Зчитування отриманого байта

    if (rx_index == 0) { // Очікування заголовка
        uint8_t data_len_from_header = (byte & 0xF0) >> 4;
        rx_buffer[0] = byte;
        rx_index = 1;

        uint8_t expected_total_len = (data_len_from_header == 0)
 ? 1 : (data_len_from_header + 2);
        if (expected_total_len > MAX_PACKET_SIZE) { // Перевірка
 розміру
            rx_index = 0; return; // Неправильний розмір, скидання
        }
        if (data_len_from_header == 0) { // Пакет без даних
            packet_ready = true;
            rx_index = 0;
        }
    } else { // Прийом даних або КС
        if (rx_index < MAX_PACKET_SIZE) { // Захист від
 переповнення
            rx_buffer[rx_index++] = byte;
        } else {
            rx_index = 0; packet_ready = false; return; //
 Переповнення, скидання
        }

        uint8_t data_len_from_header = (rx_buffer[0] & 0xF0) >> 4;
        uint8_t expected_total_len = (data_len_from_header == 0)
 ? 1 : (data_len_from_header + 2);

        if (rx_index == expected_total_len) { // Пакет повністю
 отримано
            packet_ready = true;
            rx_index = 0;
        }
    }
}

// Обробник переривання Таймера1 (для збереження в EEPROM)
ISR(TIMER1_COMPA_vect) {
    seconds_counter++; // Інкремент лічильника (кожне переривання
 = 4с)
```

```
    if (seconds_counter >= 15) { // Якщо пройшла 1 хвилина (15 *
 4с)
        seconds_counter = 0;
        time_to_save_data = true; // Прапорець для збереження
    }
}

// Основна програма
int main(void) {
    UART_init();
    RGB_init();
    Timer1_init();
    sei(); // Дозвіл глобальних переривань

    while (1) {
        if (packet_ready) { // Обробка отриманого пакету
            uint8_t local_rx_copy[MAX_PACKET_SIZE];
            uint8_t actual_packet_length_received;

            cli(); // Початок критичної секції
            uint8_t header_byte_copy = rx_buffer[0];
            uint8_t data_len_copy = (header_byte_copy & 0xF0) >>
 4;
            actual_packet_length_received = (data_len_copy == 0)
 ? 1 : (data_len_copy + 2);
            if (actual_packet_length_received > MAX_PACKET_SIZE)
 { // Захист
                actual_packet_length_received = MAX_PACKET_SIZE;
            }
            memcpy(local_rx_copy,     (const     void*)rx_buffer,
 actual_packet_length_received);
            packet_ready = false; // Скидання прапорця
            sei(); // Кінець критичної секції

            // Розбір та обробка пакету
            uint8_t header = local_rx_copy[0];
            uint8_t data_len = (header & 0xF0) >> 4;
            uint8_t cmd = (header & 0x0F);

            if (data_len == 0) { // Пакет без даних/КС
                execute_command(cmd, NULL, data_len);
            } else { // Пакет з даними та КС
                uint8_t* data_ptr = &local_rx_copy[1];
                uint8_t  checksum_received  =  local_rx_copy[1  +
 data_len];
                uint8_t           checksum_calculated           =
 calculate_checksum(local_rx_copy, 1 + data_len);

                if (checksum_received == checksum_calculated) { //
 КС вірна
                    execute_command(cmd, data_ptr, data_len);
                } else { // КС невірна
```

```
                    uint8_t      error_payload[1]      =      {
 checksum_received };
                    send_response_packet(0x1, error_payload, 1);
 // Відповідь: помилка КС
                }
            }
        }

        if (time_to_save_data) { // Періодичне збереження даних в
 EEPROM
            time_to_save_data = false;
            int temperature_val = 0, humidity_val = 0;
            if (dht11_read_data(&temperature_val, &humidity_val)
 == DHT11_SUCCESS) {
                uint16_t   eeprom_idx   =   EEPROM.read(0)   |
 (EEPROM.read(1) << 8); // Читання індексу
                if (eeprom_idx >= 480) eeprom_idx = 0; //
 Циклічний буфер

                int physical_address = 2 + eeprom_idx * 2; //
 Адреса для запису (+2 для індексу)
                EEPROM.update(physical_address,
 (uint8_t)temperature_val); // Запис температури
                EEPROM.update(physical_address       +       1,
 (uint8_t)humidity_val); // Запис вологості

                eeprom_idx++; // Інкремент індексу
                EEPROM.update(0, eeprom_idx & 0xFF);          //
 Збереження молодшого байта індексу
                EEPROM.update(1, (eeprom_idx >> 8) & 0xFF);   //
 Збереження старшого байта індексу
            }
        }
    }
}
```

Лістинг Г.2 – Керуюча програма

```
import serial
import time

MAX_DATA_LEN = 14  # Макс. довжина поля даних
DEFAULT_RESPONSE_TIMEOUT = 5    # Таймаут очікування відповіді
 (загальний)
SERIAL_OPERATION_TIMEOUT = 1     # Таймаут для операції
 читання/парсингу


# Обчислює контрольну суму (XOR)
def calculate_checksum(packet_bytes: bytes) -> int:
    checksum = 0
    for b_val in packet_bytes:
        checksum ^= b_val
    return checksum


# Формує пакет для відправки на МК
def build_packet(command: int, data: bytes = b'') -> bytes:
    if not (0 <= command <= 15):
        raise ValueError("Код команди 0-15.")
    if len(data) > MAX_DATA_LEN:
        raise ValueError(f"Дані > {MAX_DATA_LEN} байт.")

    header = ((len(data) & 0x0F) << 4) | (command & 0x0F) #
 Заголовок: [4 біти довжини | 4 біти команди]
    packet_without_checksum = bytes([header]) + data

    if len(data) > 0: # Якщо є дані, додаємо КС
        checksum = calculate_checksum(packet_without_checksum)
        return packet_without_checksum + bytes([checksum])
    else:
        return packet_without_checksum


# Розбирає отриманий від МК пакет
def parse_response_packet(packet_bytes: bytes) -> dict:
    if not packet_bytes:
        raise ValueError("Порожній пакет для парсингу.")

    header = packet_bytes[0]
    data_len = (header & 0xF0) >> 4
    response_code = header & 0x0F

    expected_packet_len = 1 + data_len + (1 if data_len > 0 else
 0) # Заголовок + Дані + [КС]

    if len(packet_bytes) != expected_packet_len:
```

```
        raise ValueError(f"Некоректна довжина пакету: очікувалось
 {expected_packet_len}, отримано {len(packet_bytes)}.")

    data = packet_bytes[1:1 + data_len]

    if data_len > 0: # Перевірка КС, якщо є дані
        checksum_received = packet_bytes[1 + data_len]
        checksum_calculated = calculate_checksum(packet_bytes[0:1
 + data_len])
        if checksum_calculated != checksum_received:
            raise    ValueError(f"Невірна    КС:    очікувалось
 {checksum_calculated:02X}, отримано {checksum_received:02X}.")

    return {"code": response_code, "data": data}


# Намагається вилучити та розібрати один пакет з
 persistent_buffer.
def _try_extract_and_parse_from_buffer(persistent_buffer:
 bytearray) -> dict | None:
    if not persistent_buffer:
        return None

    if len(persistent_buffer) >= 1:  # Чи є хоча б заголовок
        header = persistent_buffer[0]
        data_len_from_header = (header & 0xF0) >> 4

        if data_len_from_header > MAX_DATA_LEN: # Пошкоджений
 заголовок
            print(f"Помилка: Довжина даних
 ({data_len_from_header}) > MAX_DATA_LEN. Видалення байту:
 {header:02X}")
            del persistent_buffer[0]
            return None

        expected_len_of_this_packet = 1 + data_len_from_header +
 (1 if data_len_from_header > 0 else 0)

        if len(persistent_buffer) >= expected_len_of_this_packet:
 # Чи достатньо даних для повного пакету
            packet_to_parse =
 bytes(persistent_buffer[:expected_len_of_this_packet])
            try:
                parsed_packet =
 parse_response_packet(packet_to_parse)
                del
 persistent_buffer[:expected_len_of_this_packet]    #    Успіх,
 видаляємо з буфера
                return parsed_packet
            except ValueError as e:
                print(f"Помилка парсингу: {e}. Видалення першого
 байту: {persistent_buffer[0]:02X}")
```

```
                del persistent_buffer[0] # Помилка парсингу,
 спроба синхронізації
                return None
    return None # Недостатньо даних


# Читає дані з serial, додає до persistent_buffer, намагається
 розібрати ОДИН пакет.
def      receive_one_packet_with_buffer(ser:      serial.Serial,
 persistent_buffer: bytearray,
                                        timeout_seconds:   float)   ->
 dict | None:
    start_time = time.time()

    while time.time() - start_time < timeout_seconds:
        parsed                                                    =
 _try_extract_and_parse_from_buffer(persistent_buffer) # Спроба
 розібрати з наявного
        if parsed:
            return parsed

        bytes_in_waiting = ser.in_waiting # Читання нових даних з
 порту
        if bytes_in_waiting > 0:
            new_data = ser.read(bytes_in_waiting)
            persistent_buffer.extend(new_data)
            parsed                                                =
 _try_extract_and_parse_from_buffer(persistent_buffer)  # Знову
 спроба розібрати
            if parsed:
                return parsed
        time.sleep(0.005)

    parsed                                                        =
 _try_extract_and_parse_from_buffer(persistent_buffer)          #
 Останній шанс після таймауту
    return parsed


# Читає потік даних EEPROM та зберігає у файл.
def          read_eeprom_data_stream(ser:          serial.Serial,
 persistent_buffer: bytearray, file_path: str,
                             timeout_per_packet: float = 0.5,
                             overall_timeout:  float  =  10.0)  ->
 bool:
    all_eeprom_payload = bytearray()
    start_overall_time = time.time()
    last_packet_time = time.time()
    packets_received_count = 0
    print("Очікування даних EEPROM...")

    while time.time() - start_overall_time < overall_timeout:
```

```
      if time.time() - last_packet_time > timeout_per_packet *
2 and packets_received_count > 0:
          print(f"Таймаут очікування наступного пакету EEPROM.
Завершення.")
          break

      parsed_packet    =    receive_one_packet_with_buffer(ser,
persistent_buffer, timeout_per_packet)

      if parsed_packet:
          packets_received_count += 1
          last_packet_time = time.time()
          print(f"Пакет               {packets_received_count}:
Дані='{parsed_packet['data'].hex(' ').upper()}'")

          if parsed_packet['code'] == 0x6:   # Пакет з даними
EEPROM
              all_eeprom_payload.extend(parsed_packet['data'])
          elif  parsed_packet['code']  ==  0x7:    #  У  EEPROM
відсутні дані
              print("МК: в EEPROM відсутні дані.")
              return False if not all_eeprom_payload else True
          else: # Несподіваний пакет
              print(f"Отримано  несподіваний  пакет  з  кодом
0x{parsed_packet['code']:X}. Завершення.")
              break
      else: # Таймаут для одного пакету
          if  packets_received_count  >  0:  #  Ймовірно,  кінець
потоку
              break
          if  not  persistent_buffer  and  not  ser.in_waiting:  #
Якщо нічого немає і не було
              pass

  if all_eeprom_payload: # Запис у файл
      try:
          with open(file_path, 'w') as f:
              f.write("Температура,Вологість\n")
              for i in range(0, len(all_eeprom_payload), 2): #
Кожен запис - 2 байти
                  if i + 1 < len(all_eeprom_payload):
                      temp,   hum   =   all_eeprom_payload[i],
all_eeprom_payload[i + 1]
                      temp_str = "Помилка" if temp == 255 else
str(temp)
                      hum_str  =  "Помилка"  if  hum  ==  255  else
str(hum)
                      f.write(f"{temp_str},{hum_str}\n")
          print(f"Дані  EEPROM  ({len(all_eeprom_payload)  //  2}
записів) збережено у '{file_path}'")
          return True
      except IOError as e:
          print(f"Помилка запису EEPROM у файл: {e}")
```

```
            return False
    elif packets_received_count > 0 and not all_eeprom_payload:
        print("Отримано відповідь від МК, але даних EEPROM для
збереження немає.")
        return False
    else:
        print("Не отримано даних з EEPROM (можливо, таймаут).")
        return False


# Інтерпретує та виводить розібрану відповідь
def interpret_response(parsed_packet: dict | None):
    if parsed_packet is None:
        print("Відповідь не отримана/розібрана.")
        return

    code, data = parsed_packet['code'], parsed_packet['data']
    print(f"Отримано відповідь: Код=0x{code:X}, Дані='{data.hex('
').upper()}'")

    if code == 0x0:  # Статус
        if len(data) == 5:
            portb_state, temp, hum = data[0], data[1], data[2]
            eeprom_ptr = (data[4] << 8) | data[3]
            temp_str = "Помилка DHT" if temp == 255 else str(temp)
            hum_str = "Помилка DHT" if hum == 255 else str(hum)
            print(f"Статус:            PORTB=0b{portb_state:06b},
T={temp_str}°C, H={hum_str}%, EEPROM_ptr={eeprom_ptr}")
        else:   print(f"Статус:   Некоректна   довжина   даних
({len(data)}).")
    elif code == 0x1: print(f"Помилка МК: Неправильна КС від ПК.
Отримана КС: 0x{data.hex().upper() if data else 'N/A'}")
    elif code == 0x2: print(f"Помилка МК: Невідома команда. Код:
0x{data.hex().upper() if data else 'N/A'}")
    elif code == 0x3: print(f"Команду 1 (PORTB) виконано. Новий
стан: 0b{data[0]:06b}" if len(data) == 1 else "Відповідь на
команду 1: Некоректні дані.")
    elif code == 0x4: print("Помилка МК: Для команди 1 (PORTB)
недостатньо даних.")
    elif code == 0x5: # Дані з датчика
        if len(data) == 2:
            temp, hum = data[0], data[1]
            temp_str = "Помилка DHT" if temp == 255 else str(temp)
            hum_str = "Помилка DHT" if hum == 255 else str(hum)
            print(f"Датчик: T={temp_str}°C, H={hum_str}%")
        else: print("Дані з датчика: Некоректна довжина.")
    elif code == 0x6: print(f"МК: Пакет даних EEPROM (обробляється
окремо). Дані: {data.hex(' ').upper()}")
    elif code == 0x7: print("Відповідь МК: У EEPROM відсутні
дані.")
    elif code == 0x8: print("Відповідь МК: Дані з EEPROM успішно
очищено.")
```

```
    else: print(f"Увага: Невідомий код відповіді від МК:
 0x{code:X}")


# Режим прямого введення пакетів (для діагностики)
def direct_input_mode(ser: serial.Serial):
    print("Режим прямого введення пакетів. 'exit' для виходу.")
    shared_buffer = bytearray()
    while True:
        try:
            user_input = input("Введіть пакет (HEX, напр. '01 0F'
 або '02'): ").strip()
            if not user_input: continue
            if user_input.lower() == 'exit': break

            hex_input_no_spaces = user_input.replace(" ", "")
            if not hex_input_no_spaces or not all(c in
 "0123456789abcdefABCDEF" for c in hex_input_no_spaces) or
 len(hex_input_no_spaces) % 2 != 0:
                print("Помилка: Некоректний HEX формат.");
 continue

            packet_to_send = bytes.fromhex(hex_input_no_spaces)
            if not packet_to_send: print("Помилка: Порожній
 пакет."); continue
            print(f"Відправка: {packet_to_send.hex('
').upper()}")

            if packet_to_send != b'\x03': # Якщо не запит EEPROM,
 очищаємо буфери
                shared_buffer.clear()
                if ser.in_waiting > 0: ser.read(ser.in_waiting)

            ser.write(packet_to_send)

            if packet_to_send == b'\x03':  # Запит даних з EEPROM
 (код 0x03)
                read_eeprom_data_stream(ser,       shared_buffer,
 "eeprom_data_direct.dat")
            else: # Одиночна команда
                response_packet                                =
 receive_one_packet_with_buffer(ser,              shared_buffer,
 DEFAULT_RESPONSE_TIMEOUT)
                interpret_response(response_packet)

            if shared_buffer: print(f"УВАГА: Залишок у буфері:
 {shared_buffer.hex(' ').upper()}")
        except ValueError as ve: print(f"Помилка введення: {ve}")
        except Exception as e: print(f"Загальна помилка: {e}")


# Режим командного рядка (CLI)
def cli_mode(ser: serial.Serial):
```

```
   print("CLI-інтерфейс.")
   commands_desc = { # Опис команд для користувача
       "0": "0 - Запит статусу", "1": "1 - Зміна стану портів",
       "2": "2 - Запит даних з датчика", "3": "3 - Запит історії
з EEPROM",
       "4": "4 - Очищення EEPROM", "exit": "exit - Вихід"
   }
   shared_buffer_cli = bytearray()

   while True:
       print("\nДоступні команди:")
       for key_cmd in sorted(commands_desc.keys(), key=lambda x:
x if x != "exit" else "z"): print(f"  {commands_desc[key_cmd]}")
       choice = input("Ваш вибір: ").strip().lower()
       if choice == 'exit': break

       command_to_send, data_to_send, is_eeprom_request = -1,
b'', False

       try: # Формування команди на основі вибору користувача
           if choice == '0': command_to_send = 0x00
           elif choice == '1':
               command_to_send = 0x01
               portb_hex_input = input("Значення для PORTB (HEX,
напр. '0F'): ").strip()
               if not (1 <= len(portb_hex_input) <= 2 and all(c
in "0123456789abcdefABCDEF" for c in portb_hex_input)):
                   print("Некоректне значення PORTB."); continue
               data_to_send = bytes([int(portb_hex_input, 16)])
           elif choice == '2': command_to_send = 0x02
           elif choice == '3': command_to_send = 0x03;
is_eeprom_request = True
           elif choice == '4': command_to_send = 0x04
           else: print("Невідома команда."); continue

           packet = build_packet(command_to_send, data_to_send)
           print(f"Відправка: {packet.hex(' ').upper()}")

           if not is_eeprom_request: # Очищення буферів для
одиночних команд
               shared_buffer_cli.clear()
               if ser.in_waiting > 0: ser.read(ser.in_waiting)

           ser.write(packet)

           if is_eeprom_request: # Обробка потоку EEPROM
               read_eeprom_data_stream(ser, shared_buffer_cli,
"eeprom_data.dat")
           else: # Обробка відповіді на одиночну команду
               response_packet =
receive_one_packet_with_buffer(ser, shared_buffer_cli,
DEFAULT_RESPONSE_TIMEOUT)
               interpret_response(response_packet)
```

```
            if shared_buffer_cli and not is_eeprom_request: # Залишок у буфері після одиночної команди
                print(f"УВАГА: Залишок у CLI буфері: {shared_buffer_cli.hex(' ').upper()}")
                shared_buffer_cli.clear()
        except ValueError as ve: print(f"Помилка: {ve}")
        except Exception as e: print(f"Загальна помилка: {e}")


def main():
    port, baudrate = 'COM4', 9600 # Налаштування порту
    print(f"Підключення до {port} на {baudrate} бод.")
    ser_connection = None
    try:
        ser_connection = serial.Serial(port, baudrate, timeout=0.05) # Невеликий таймаут для ser.read()
        time.sleep(1) # Пауза для стабілізації
        print(f"З'єднано з {port}.")

        mode_choice = input("CLI-інтерфейс? (y/n, default y): ").strip().lower()
        if mode_choice == 'n': direct_input_mode(ser_connection) # Режим прямого введення
        else: cli_mode(ser_connection) # Режим CLI
    except serial.SerialException as e: print(f"Помилка: Неможливо відкрити порт {port}: {e}")
    except KeyboardInterrupt: print("\nРоботу завершено користувачем.")
    except Exception as e: print(f"Непередбачена помилка: {e}")
    finally:
        if ser_connection and ser_connection.is_open:
            ser_connection.close()
            print(f"Порт {port} закрито.")

if __name__ == '__main__':
    main()
```