КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра фізичної електроніки

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри фізичної електроніки
від 16 травня 2025 року, протокол № 22
Завідувач кафедри д.ф.-м.н., професор
_____Анатолій ВЕКЛИЧ

ДИПЛОМНА РОБОТА МАГІСТРА
на тему:
«СИСТЕМА КЕРУВАННЯ ПРОЦЕСОМ РЕЄСТРАЦІЇ ТА ОБРОБКИ СПЕКТРІВ ВИПРОМІНЮВАННЯ ПЛАЗМИ»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 105 Прикладна фізика та наноматеріали
ОНП «Прикладна фізика та наноматеріали»
Стащенко Денис Олександрович _____

Науковий керівник:
д.ф.-м.н., професор
Веклич Анатолій Миколайович _____

Рецензент:
д.ф.-м.н., старший науковий співробітник
Кукла Олександр Леонідович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Денис СТАЩЕНКО

Київ – 2025

РЕФЕРАТ

Робота містить 79 сторінку, включаючи 15 рисунків, 4 додатка.

Ключові слова**:** МІКРОКОНТРОЛЕР, STM32F103C6T6, RS-485, ПЛАЗМОВА УСТАНОВКА, МОНІТОРИНГ ПАРАМЕТРІВ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, ДАТЧИКИ, ПЕРЕДАЧА ДАНИХ, АВТОМАТИЗАЦІЯ.

У магістерській роботі розглянуто розробку пристрою на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485. Пристрій призначений для передачі даних між персональним комп'ютером та сторонніми пристроями, які збирають інформацію про параметри плазми в плазмовій установці. Розроблений пристрій виконує функцію керування сторонніми пристроями та збір даних від них та передачу даних до персонального комп’ютера. У роботі детально описано апаратну частину пристрою, включаючи вибір компонентів та схему підключення, а також розробку програмного забезпечення з інтуїтивно зрозумілим інтерфейсом користувача. Проведено тестування та налагодження системи, підтверджено її надійність та ефективність. Результати роботи демонструють значне значення розробки для наукових досліджень та промислових застосувань, відкриваючи перспективи для подальших вдосконалень та інтеграції з іншими системами.

ЗМІСТ

1. Вступ

1.1. Мета роботи

Метою даної магістерської роботи є розробка та впровадження пристрою на базі мікроконтролера STM32F103C6T6 для керування та збору даних від сторонніх пристроїв через шину RS-485. Основна увага приділяється створенню надійного та ефективного рішення, яке забезпечує доставку даних між персональним комп'ютером (ПК) та сторонніми пристроями, які надсилатимуть дані, такі як сила струму, напруга, частота.

Розроблений пристрій повинен забезпечити високу швидкість та надійність передачі даних, а також стійкість до електромагнітних перешкод, характерних для плазмових установок. Крім того, метою роботи є створення інтуїтивно зрозумілого інтерфейсу користувача для ПК, який дозволить легко керувати процесом збору даних та аналізувати отримані результати.

Досягнення цієї мети передбачає виконання наступних завдань:

- Вивчення теоретичних основ фізики плазми та принципів роботи плазмових установок.
- Аналіз вимог до пристроїв для збору даних в умовах плазмових установок.
- Огляд існуючих рішень та технологій для збору даних.
- Розробка апаратної частини пристрою на базі мікроконтролера STM32F103C6T6.
- Розробка програмного забезпечення для мікроконтролера та інтерфейсу користувача для ПК.
- Інтеграція пристрою зі сторонніми пристроями та проведення експериментальних випробувань.
- Аналіз результатів випробувань та оцінка ефективності розробленого рішення.

Успішне виконання цих завдань дозволить створити ефективний та надійний пристрій для збору даних, який значно спростить процес моніторингу параметрів плазми та підвищить точність і швидкість отримання необхідної інформації.

1.2. Актуальність теми

Сучасні наукові та промислові дослідження в галузі фізики плазми вимагають високоточних та надійних систем для збору та обробки даних. Плазмові установки використовуються в різноманітних областях, включаючи енергетику, матеріалознавство, медицину та нанотехнології. Ефективне керування та моніторинг параметрів плазми, таких як сила струму, напруга та розхід газу, є критично важливим для забезпечення стабільної роботи установок та отримання точних наукових результатів.

Однією з ключових проблем, з якою стикаються дослідники, є необхідність забезпечення надійного зв'язку між різноманітними датчиками та центральним

обчислювальним вузлом. Традиційні системи збору даних (DAQ) часто є надто складними та дорогими для масового застосування, а також можуть не задовольняти специфічні вимоги плазмових установок, такі як стійкість до електромагнітних перешкод.

Використання мікроконтролерів, таких як STM32F103C6T6, для створення спеціалізованих пристроїв збору даних дозволяє значно знизити вартість та складність розробки, забезпечуючи при цьому високу надійність та швидкість передачі даних. Протокол RS-485, завдяки своїй стійкості до перешкод та можливості передачі даних на великі відстані, є оптимальним вибором для застосування в умовах плазмових установок.

Розробка та впровадження спеціалізованого пристрою для збору даних на базі STM32 та RS-485 дозволить підвищити ефективність та точність моніторингу параметрів плазми, що, в свою чергу, сприятиме прогресу в наукових дослідженнях та промислових застосуваннях. Крім того, таке рішення може бути адаптоване для використання в інших галузях, де потрібна надійна та ефективна передача даних між різноманітними пристроями.

Таким чином, актуальність теми обумовлена необхідністю створення доступних та надійних систем для збору даних в умовах плазмових установок, що дозволить підвищити точність наукових досліджень та ефективність промислових процесів.

1.3. Цілі та завдання

Основна ціль роботи полягає у розробці та впровадженні надійного та ефективного пристрою на базі мікроконтролера STM32F103C6T6 для збору та передачі даних від сторонніх пристроїв через шину RS-485 в умовах роботи плазмової установки. Цей пристрій повинен забезпечити високу швидкість та надійність передачі даних, а також стійкість до електромагнітних перешкод.

1.4. Обґрунтування вибору теми

Вибір теми "Використання мікроконтролера для керування пристроями на шині RS-485" для магістерської роботи обумовлений кількома ключовими факторами, які підкреслюють її важливість та актуальність у сучасному науковому та технологічному контексті.

- Потреба в надійних системах збору даних: сучасні наукові дослідження та промислові процеси вимагають високоточних та надійних систем для збору та обробки даних. Плазмові установки, які використовуються в різноманітних галузях, включаючи енергетику, матеріалознавство та медицину, потребують постійного моніторингу параметрів плазми для забезпечення стабільної роботи та отримання точних результатів. Розробка спеціалізованого пристрою для збору даних дозволить задовольнити ці потреби, забезпечивши високу надійність та швидкість передачі інформації.

- Використання сучасних технологій: мікроконтролери, такі як STM32F103C6T6, надають широкі можливості для створення компактних, енергоефективних та високопродуктивних пристроїв. Їх використання дозволяє значно знизити вартість та складність розробки порівняно з традиційними системами збору даних (DAQ). Протокол RS-485, завдяки своїй стійкості до електромагнітних перешкод та можливості передачі даних на великі відстані, є оптимальним вибором для застосування в умовах плазмових установок.
- Інноваційний підхід: розробка пристрою, який виконує функцію доставщика даних між ПК та сторонніми пристроями, є інноваційним рішенням, яке відрізняється від традиційних систем DAQ. Такий підхід дозволяє зосередитися на основній задачі – надійній та швидкій передачі даних, залишаючи заміри та зберігання інформації на стороні сторонніх пристроїв. Це спрощує архітектуру системи та підвищує її надійність.
- Практичне застосування: розроблений пристрій має широке практичне застосування не тільки в плазмових установках, але і в інших галузях, де потрібна надійна та ефективна передача даних між різноманітними пристроями. Це може включати промислову автоматизацію, системи моніторингу, медичну діагностику та інші області, де важлива точність та швидкість обробки інформації.
- Особистий інтерес та професійний розвиток: особистий інтерес до розробки електронних пристроїв та систем збору даних, а також бажання внести вклад у розвиток сучасних технологій, є важливими мотиваційними факторами для вибору цієї теми. Виконання даної роботи дозволить мені поглибити знання в галузі мікроконтролерів та протоколів зв'язку, а також набути досвіду в розробці та впровадженні інноваційних рішень.
- Внесок у науковий та технологічний прогрес: успішне виконання цієї роботи дозволить внести вагомий внесок у розвиток технологій для збору даних в умовах плазмових установок. Результати досліджень можуть бути використані для подальших наукових розробок та промислових застосувань, сприяючи прогресу в цій галузі.

Таким чином, вибір теми "Використання мікроконтролера для керування пристроями на шині RS-485" є обґрунтованим та актуальним, оскільки він відповідає сучасним викликам науки та технологій, має широке практичне застосування та сприяє професійному розвитку.

2. Теоретичні основи

2.1. Фізика плазми

Плазма є четвертим станом речовини, який відрізняється від твердого, рідкого та газоподібного станів своїми унікальними властивостями. Вона складається з іонізованих атомів та молекул, тобто атомів, які втратили один або кілька електронів, утворюючи позитивно заряджені іони, та вільних електронів. Плазма характеризується високою електричною провідністю та взаємодією з

електромагнітними полями, що робить її важливим об'єктом вивчення в різних галузях науки та техніки.

Основні властивості плазми:

- Квазінейтральність: плазма залишається електрично нейтральною в макроскопічному масштабі, оскільки кількість позитивно заряджених іонів приблизно дорівнює кількості негативно заряджених електронів. Однак на мікроскопічному рівні можуть виникати локальні зміни заряду, які впливають на поведінку плазми.
- Колективна поведінка: частинки в плазмі взаємодіють одна з одною через електромагнітні сили, що призводить до колективної поведінки. Це означає, що рух однієї частинки може впливати на рух інших частинок, створюючи хвильові та коливальні процеси.
- Висока температура: плазма зазвичай має високу температуру, оскільки для іонізації атомів потрібна значна енергія. Температура плазми може досягати мільйонів градусів Кельвіна, що робить її важливим об'єктом вивчення в астрофізиці та термоядерній енергетиці.
- Електрична провідність: висока концентрація вільних електронів робить плазму добрим провідником електричного струму. Ця властивість використовується в різних технологічних процесах, таких як зварювання, різання металів та плазмове напилення.

Плазму можна класифікувати за різними ознаками, такими як температура, густина частинок та ступінь іонізації. Основні типи плазми включають:

- Термічна плазма: виникає при високих температурах, коли теплова енергія достатня для іонізації атомів. Приклади включають зоряну плазму та плазму в термоядерних реакторах.
- Нетермічна плазма: виникає при нижчих температурах, коли іонізація відбувається за рахунок зовнішніх джерел енергії, таких як електричне поле або лазерне випромінювання. Приклади включають газорозрядні лампи та плазмові телевізори.
- Низькотемпературна плазма: має температури нижче 10^4 Кельвіна. Використовується в промислових процесах, таких як плазмове напилення та очищення поверхонь.

Плазма знаходить широке застосування в різних галузях науки та техніки:

- Енергетика: вивчення термоядерного синтезу, де плазма використовується для отримання енергії шляхом злиття ядер водню.
- Матеріалознавство: плазмове напилення, очищення поверхонь та модифікація матеріалів за допомогою плазмових технологій.
- Медицина: плазмова стерилізація та обробка біологічних тканин.
- Астрофізика: вивчення процесів, що відбуваються в зорях та міжзоряному середовищі.

- Промисловість: плазмове різання та зварювання металів, а також плазмове травлення.

2.2. Вимоги до пристроїв для плазмових установок

Пристрої, призначені для роботи в умовах плазмових установок, повинні задовольняти ряд специфічних вимог, які забезпечують їх надійну та ефективну роботу. Ці вимоги визначаються особливостями плазмових процесів, такими як високі температури, електромагнітні перешкоди та агресивне середовище. Розглянемо основні вимоги до таких пристроїв.

- Стійкість до електромагнітних перешкод: плазма є джерелом потужних електромагнітних полів, які можуть впливати на роботу електронних пристроїв. Тому пристрої повинні бути захищені від електромагнітних перешкод за допомогою екранування, фільтрів та інших засобів. Використання протоколів зв'язку, стійких до перешкод, таких як RS-485, також є важливим аспектом.
- Надійність та довговічність: пристрої повинні бути надійними та довговічними, щоб забезпечити безперебійну роботу плазмової установки. Це включає використання якісних компонентів, регулярне технічне обслуговування та можливість легкого ремонту або заміни пристроїв у разі пошкодження.
- Безпека експлуатації: безпека є одним з ключових аспектів при роботі з плазмовими установками. Пристрої повинні бути спроектовані таким чином, щоб мінімізувати ризики для операторів та обладнання. Це включає захист від коротких замикань, перегріву, витоку газу та інших потенційно небезпечних ситуацій.
- Можливість дистанційного керування та моніторингу: для забезпечення безпеки та зручності експлуатації пристрої повинні підтримувати можливість дистанційного керування та моніторингу. Це дозволяє операторам контролювати параметри плазми та керувати процесами з віддалених робочих місць, зменшуючи ризики та підвищуючи ефективність роботи.
- Енергоефективність: пристрої повинні бути енергоефективними, щоб зменшити витрати на експлуатацію плазмової установки. Це включає оптимізацію споживання енергії, використання енергозберігаючих технологій та мінімізацію втрат енергії.
- Модульність та масштабованість: пристрої повинні бути модульними та масштабованими, щоб їх можна було легко адаптувати до різних завдань та умов експлуатації. Це дозволяє змінювати конфігурацію системи без значних витрат та зусиль, а також розширювати її функціональні можливості за потреби.

2.3. Основи мікроконтролерів

Мікроконтролери є важливим компонентом сучасних електронних систем, які використовуються для автоматизації, керування та обробки даних. Вони

представляють собою інтегральні схеми, які об'єднують в собі центральний процесор, пам'ять, периферійні пристрої та інтерфейси вводу-виводу на одному кристалі. Це робить їх компактними, енергоефективними та зручними для використання в різноманітних застосуваннях. Розглянемо основні принципи роботи мікроконтролерів та їх класифікацію.

Мікроконтролери мають наступні основні компоненти:

- Центральний процесор (CPU): виконує обробку команд та керує роботою всієї системи. Процесор може бути 8-розрядним, 16-розрядним, 32-розрядним або 64-розрядним, що визначає його продуктивність та можливості.
- Пам'ять:
    - Оперативна пам'ять (RAM): використовується для тимчасового зберігання даних під час виконання програм.
    - Постійна пам'ять (ROM): зберігає програмне забезпечення та константи, які не змінюються під час роботи.
    - Флеш-пам'ять: дозволяє зберігати та змінювати програми та дані, забезпечуючи можливість оновлення прошивки.
- Периферійні пристрої: включають таймери, лічильники, аналого-цифрові перетворювачі (ADC), цифро-аналогові перетворювачі (DAC), генератори тактових імпульсів та інші модулі, які розширюють функціональні можливості мікроконтролера.
- Інтерфейси вводу-виводу (I/O): забезпечують зв'язок мікроконтролера з зовнішніми пристроями. Інтерфейси можуть бути цифровими (GPIO) або аналоговими, а також підтримувати різні протоколи зв'язку, такі як UART, SPI, I2C, CAN тощо.

Мікроконтролери можна класифікувати за різними ознаками, такими як архітектура, розрядність, продуктивність та призначення. Основні типи мікроконтролерів включають:

- 8-розрядні мікроконтролери: відрізняються простотою та низькою вартістю. Використовуються в простих застосуваннях, таких як керування світлодіодами, реле та іншими елементами. Приклади: AVR (наприклад, ATmega328), PIC.
- 16-розрядні мікроконтролери: забезпечують більшу продуктивність та функціональність порівняно з 8-розрядними. Використовуються в застосуваннях, де потрібна вища швидкість обробки даних та більший обсяг пам'яті. Приклади: MSP430.
- 32-розрядні мікроконтролери: надають високу продуктивність та широкі можливості для обробки даних. Використовуються в складних системах, таких як промислова автоматизація, медичні пристрої та комунікаційні системи. Приклади: ARM Cortex-M (наприклад, STM32), ESP32.
- 64-розрядні мікроконтролери: забезпечують максимальну продуктивність та можливості для обробки великих обсягів даних. Використовуються в

застосуваннях, де потрібна висока швидкість обробки та великий обсяг пам'яті. Приклади: деякі моделі на базі ARM Cortex-A.

Робота мікроконтролера починається з завантаження програми з постійної пам'яті або флеш-пам'яті в оперативну пам'ять. Потім процесор починає виконувати команди програми послідовно, керуючи периферійними пристроями та обробляючи дані з інтерфейсів вводу-виводу. Основні етапи роботи мікроконтролера включають:

- Ініціалізація: налаштування периферійних пристроїв, таймерів та інтерфейсів вводу-виводу відповідно до вимог програми.
- Основний цикл: виконання основних завдань програми, таких як збір даних, обробка сигналів та керування зовнішніми пристроями.
- Обробка переривань: мікроконтролери підтримують механізм переривань, який дозволяє швидко реагувати на зовнішні події або внутрішні сигнали. Переривання дозволяють тимчасово припинити виконання основного циклу та виконати пріоритетні завдання.

Мікроконтролери знаходять широке застосування в різних галузях:

- Промислова автоматизація: керування роботами, конвеєрами, системами моніторингу та іншими промисловими процесами.
- Споживча електроніка: використання в побутовій техніці, такій як пральні машини, мікрохвильові печі, кондиціонери тощо.
- Автомобільна електроніка: керування системами керування двигуном, антиблокувальними гальмами (ABS), подушками безпеки та іншими автомобільними системами.
- Медичні пристрої: використання в портативних медичних пристроях, таких як глюкометри, пульсоксиметри, кардіостимулятори тощо.
- Комунікаційні системи: керування модемами, маршрутизаторами, бездротовими мережами та іншими комунікаційними пристроями.

2.4. Основи протоколів зв’язку

Протоколи зв'язку є набором правил та стандартів, які визначають, як дані передаються між електронними пристроями. Вони забезпечують надійну та ефективну передачу інформації, гарантуючи, що дані будуть правильно інтерпретовані отримувачем. Розглянемо основні принципи роботи протоколів зв'язку, їх класифікацію та застосування.

Основні принципи роботи протоколів зв'язку:

- Синхронізація: для успішної передачі даних необхідно, щоб відправник та отримувач були синхронізовані. Це може бути досягнуто за допомогою тактових сигналів або спеціальних синхронізаційних послідовностей.
- Адресація: кожен пристрій у мережі повинен мати унікальну адресу, щоб дані могли бути надіслані до правильного отримувача. Адресація дозволяє розрізняти пристрої та керувати потоком даних.

- Форматування даних: дані повинні бути сформовані у визначеному форматі, щоб отримувач міг правильно їх інтерпретувати. Форматування включає визначення структури пакету даних, включаючи заголовки, тіло та контрольні суми.
- Контроль помилок: протоколи зв'язку передбачають механізми виявлення та корекції помилок, які можуть виникнути під час передачі даних. Це може включати використання контрольних сум, CRC (циклічний надлишковий код) та інших методів.
- Керування потоком: для запобігання переповнення буферів та втрати даних протоколи зв'язку передбачають механізми керування потоком. Це може включати використання сигналів "готовий до прийому" та "зайнятий".

Протоколи зв'язку можна класифікувати за різними ознаками, такими як тип зв'язку, швидкість передачі даних, відстань передачі та призначення. Основні типи протоколів зв'язку включають:

- Послідовні протоколи: дані передаються послідовно, біт за бітом, через один канал зв'язку. Приклади: UART, RS-232, RS-485.
- Паралельні протоколи: дані передаються паралельно, декілька бітів одночасно, через кілька каналів зв'язку. Приклади: паралельний інтерфейс принтерів, деякі внутрішньосистемні шини.
- Синхронні протоколи: дані передаються з фіксованою швидкістю, з використанням тактового сигналу для синхронізації. Приклади: SPI, I2C.
- Асинхронні протоколи: дані передаються без фіксованої швидкості, з використанням стартових та стопових бітів для синхронізації. Приклади: UART, RS-232.
- Протоколи зв'язку для локальних мереж: використовуються для передачі даних між пристроями в локальних мережах. Приклади: Ethernet, Wi-Fi.
- Протоколи зв'язку для промислових мереж: спеціалізовані протоколи, призначені для роботи в умовах промислових установок. Приклади: CAN, Modbus, Profibus.

Розглянемо деякі з найпоширеніших протоколів зв'язку, які використовуються в сучасних електронних системах:

- UART (Universal Asynchronous Receiver/Transmitter): послідовний асинхронний протокол, який використовується для передачі даних між мікроконтролерами та периферійними пристроями. UART передає дані біт за бітом, з використанням стартових та стопових бітів для синхронізації.
- RS-232: стандартний послідовний інтерфейс для передачі даних між комп'ютерами та периферійними пристроями. RS-232 використовує асинхронну передачу даних та підтримує відстані до 15 метрів.
- RS-485: послідовний інтерфейс, який дозволяє передавати дані на великі відстані (до 1200 метрів) та підтримує багатоточковий зв'язок. RS-485 стійкий до електромагнітних перешкод, що робить його ідеальним для промислових застосувань.

- SPI (Serial Peripheral Interface): синхронний послідовний інтерфейс, який використовується для зв'язку між мікроконтролерами та периферійними пристроями. SPI передає дані за допомогою чотирьох ліній: SCLK (тактовий сигнал), MOSI (вихід даних), MISO (вхід даних) та SS (вибір слейву).
- I2C (Inter-Integrated Circuit): синхронний послідовний інтерфейс, який використовується для зв'язку між мікроконтролерами та периферійними пристроями. I2C передає дані за допомогою двох ліній: SDA (дані) та SCL (тактовий сигнал).
- CAN (Controller Area Network): протокол зв'язку, призначений для передачі даних між пристроями в автомобільних та промислових мережах. CAN забезпечує високу надійність та стійкість до перешкод, а також підтримує багатоточковий зв'язок.

Вибір протоколу зв'язку залежить від конкретних вимог застосування, таких як швидкість передачі даних, відстань передачі, стійкість до перешкод та кількість пристроїв у мережі. Основні критерії вибору включають:

- Швидкість передачі даних: визначає, наскільки швидко дані можуть бути передані між пристроями.
- Відстань передачі: визначає максимальну відстань, на яку дані можуть бути передані без втрат.
- Стійкість до перешкод: визначає, наскільки протокол стійкий до електромагнітних перешкод та інших збурень.
- Кількість пристроїв: визначає, скільки пристроїв можуть бути підключені до мережі одночасно.
- Складність реалізації: визначає, наскільки складно реалізувати протокол зв'язку в конкретному застосуванні.

3. Огляд літератури

3.1. Історія розвитку технологій для збору даних в плазмових установках

Збір даних в плазмових установках є критично важливим для розуміння та контролю плазмових процесів. Історія розвитку цих технологій відображає еволюцію від простих аналогових приладів до складних цифрових систем, які забезпечують високу точність та швидкість вимірювань. Розглянемо основні етапи розвитку технологій для збору даних в плазмових установках.

Ранні етапи (1950-1970-ті роки): аналогові прилади та осцилографи

На початкових етапах дослідження плазми використовувалися переважно аналогові прилади. Осцилографи, які відображали сигнали на екрані, були основним інструментом для вимірювання електричних параметрів плазми, таких як напруга та струм. Ці прилади дозволяли візуалізувати зміни сигналів у реальному часі, але мали обмежену точність та можливості зберігання даних.

1970-1980-ті роки: поява цифрових технологій

З розвитком мікроелектроніки та цифрових технологій з'явилися перші цифрові осцилографи та аналого-цифрові перетворювачі (ADC). Ці прилади дозволяли перетворювати аналогові сигнали в цифрові дані, які можна було зберігати та обробляти за допомогою комп'ютерів. Це значно підвищило точність вимірювань та можливості аналізу даних.

1980-1990-ті роки: розвиток систем автоматизованого збору даних (DAQ)

Системи автоматизованого збору даних (DAQ) стали наступним кроком у розвитку технологій для збору даних в плазмових установках. Ці системи включали модулі для збору даних, які могли підключатися до різних датчиків та передавати інформацію на комп'ютер для обробки та зберігання. DAQ-системи дозволили автоматизувати процес збору даних, зменшити помилки та підвищити ефективність досліджень.

1990-2000-ті роки: інтеграція з персональними комп'ютерами

З появою потужних персональних комп'ютерів та розвитком програмного забезпечення DAQ-системи стали ще більш функціональними. Програми для обробки даних, такі як LabVIEW та MATLAB, дозволили дослідникам створювати складні алгоритми аналізу та візуалізації даних. Це дозволило проводити більш детальні дослідження та отримувати нові знання про поведінку плазми.

2000-2010-ті роки: розвиток спеціалізованих систем

У цей період з'явилися спеціалізовані системи для збору даних, призначені для конкретних застосувань в плазмових установках. Ці системи включали високошвидкісні ADC, цифрові фільтри та модулі для обробки сигналів в реальному часі. Вони дозволили досягти ще вищої точності та швидкості вимірювань, а також забезпечили можливість роботи в умовах високих електромагнітних перешкод.

2010-2020-ті роки: інтернет речей (IoT) та хмарні технології

З розвитком Інтернету речей (IoT) та хмарних технологій з'явилися нові можливості для збору та обробки даних в плазмових установках. Датчики та пристрої збору даних можуть бути підключені до інтернету та передавати інформацію на віддалені сервери для зберігання та аналізу. Це дозволяє проводити дослідження в реальному часі та обмінюватися даними з іншими науковими установами.

Сучасні тенденції (2020-ті роки): штучний інтелект та машинне навчання

Сучасні технології збору даних в плазмових установках активно використовують методи штучного інтелекту та машинного навчання. Ці методи дозволяють аналізувати великі обсяги даних, виявляти закономірності та прогнозувати

поведінку плазми. Використання нейронних мереж та алгоритмів машинного навчання дозволяє підвищити точність та ефективність досліджень, а також автоматизувати процес прийняття рішень.

3.2. Основні досягнення в галузі

Галузь технологій для збору даних в плазмових установках пройшла значний шлях розвитку, і за цей час було досягнуто багато важливих результатів. Ці досягнення включають як технологічні інновації, так і наукові відкриття, які значно розширили наше розуміння плазмових процесів та їх застосування. Розглянемо основні досягнення, які сформували сучасний стан галузі.

Розвиток високошвидкісних аналого-цифрових перетворювачів (ADC)

Одним з ключових досягнень є розробка високошвидкісних ADC, які дозволяють перетворювати аналогові сигнали в цифрові дані з високою точністю та швидкістю. Це дозволило значно підвищити якість вимірювань та зменшити час обробки даних. Сучасні ADC можуть працювати з частотами дискретизації до декількох гігагерц, що робить їх незамінними для досліджень плазми, де швидкість зміни параметрів може бути дуже високою.

Інтеграція з цифровими системами обробки сигналів (DSP)

Інтеграція цифрових систем обробки сигналів (DSP) з пристроями збору даних дозволила проводити складну обробку даних в реальному часі. DSP-процесори можуть виконувати такі завдання, як фільтрація, аналіз спектру та виявлення аномалій, що значно підвищує ефективність досліджень. Використання DSP дозволяє автоматизувати процес аналізу даних та прийняття рішень, зменшуючи потребу в ручному втручанні.

Розробка спеціалізованих датчиків

Створення спеціалізованих датчиків для вимірювання параметрів плазми, таких як температура, густина, склад та електричні характеристики, є важливим досягненням. Ці датчики можуть працювати в екстремальних умовах, характерних для плазмових установок, та забезпечувати високу точність вимірювань. Розробка таких датчиків дозволила отримувати більш детальну інформацію про поведінку плазми та її взаємодію з навколишнім середовищем.

Використання оптичних методів діагностики

Оптичні методи діагностики, такі як спектроскопія, інтерферометрія та лазерна діагностика, стали важливими інструментами для вивчення плазми. Ці методи дозволяють вимірювати параметри плазми без контакту з нею, що зменшує ризик пошкодження обладнання та забезпечує високу точність вимірювань. Оптичні методи дозволили вивчати такі явища, як розподіл температури, густини та складу плазми, а також виявляти нестабільності та турбулентні процеси.

Розвиток систем автоматизованого збору даних (DAQ)

Системи автоматизованого збору даних (DAQ) стали невід'ємною частиною сучасних плазмових установок. Вони дозволяють автоматизувати процес збору, обробки та зберігання даних, зменшуючи помилки та підвищуючи ефективність досліджень. Сучасні DAQ-системи можуть працювати з великою кількістю датчиків та пристроїв, забезпечуючи синхронізацію та обробку даних в реальному часі.

Використання штучного інтелекту та машинного навчання

Одним з найновіших досягнень є застосування методів штучного інтелекту та машинного навчання для аналізу даних, отриманих з плазмових установок. Ці методи дозволяють виявляти закономірності, прогнозувати поведінку плазми та оптимізувати параметри експериментів. Використання нейронних мереж та алгоритмів машинного навчання дозволяє обробляти великі обсяги даних та отримувати нові знання про плазмові процеси.

Розробка стійких до перешкод протоколів зв'язку

Розробка протоколів зв'язку, стійких до електромагнітних перешкод, є важливим досягненням для забезпечення надійної передачі даних в умовах плазмових установок. Протоколи, такі як RS-485, дозволяють передавати дані на великі відстані та забезпечувати стійкість до збурень, що є критично важливим для точності та надійності вимірювань.

Інтеграція з хмарними технологіями та Інтернетом речей (IoT)

Сучасні технології збору даних в плазмових установках активно використовують хмарні обчислення та Інтернет речей (IoT) для зберігання та обробки даних. Це дозволяє проводити дослідження в реальному часі, обмінюватися даними з іншими науковими установами та забезпечувати доступ до інформації з будь-якої точки світу. Використання хмарних технологій та IoT відкриває нові можливості для співпраці та інновацій в галузі.

3.3. Огляд статті "Peculiarities of interaction of Cu-W composite materials with thermal arc discharge plasma"

У статті "Peculiarities of interaction of Cu-W composite materials with thermal arc discharge plasma" досліджується взаємодія композитних матеріалів на основі міді та вольфраму (Cu-W) з термічною плазмою дугового розряду. Автори зосереджуються на аналізі спектрів випромінювання плазми, що утворюється між електродами з Cu-W композиту, виготовленого за технологією ударного пресування при температурі 750°C. Дослідження включає визначення радіального розподілу температури плазми в різних перерізах плазмового каналу.

Ця робота тісно пов'язана з моєю магістерською роботою, яка стосується системи керування процесом реєстрації та обробки спектрів випромінювання плазми. У моїй роботі розроблено пристрій на базі мікроконтролера STM32F103C6T6, який

може взаємодіяти з ПК через USB та з іншими пристроями через інтерфейс RJ-45. Це дозволяє передавати дані від пристроїв до ПК та відправляти команди з ПК до конкретних пристроїв.

Обидва дослідження зосереджені на оптимізації параметрів плазми для різних застосувань. У статті наголошується на важливості контролю параметрів плазми, таких як температура та спектральні характеристики, що є критично важливим для промислових застосувань. У моїй роботі акцент робиться на розробці системи керування, яка може ефективно збирати та обробляти дані спектрів випромінювання плазми, що також має велике значення для аналізу та оптимізації плазмових процесів.

Таким чином, обидва дослідження доповнюють одне одного, надаючи як теоретичну базу для розуміння поведінки плазми, так і практичні рішення для керування та обробки даних плазмових процесів.

3.4. Огляд статті "Influence of pulse modulation frequency on helium RF atmospheric pressure plasma jet characteristics"

У статті "Influence of pulse modulation frequency on helium RF atmospheric pressure plasma jet characteristics" досліджується вплив частоти імпульсної модуляції на характеристики гелієвої плазми атмосферного тиску, що генерується радіочастотним (RF) джерелом. Автори вивчають, як зміна частоти модуляції впливає на форму плазмового струменя, його інтенсивність, температуру газу та генерацію реактивних видів. Дослідження показує, що низькі частоти модуляції (близько 50 Гц) забезпечують вищу щільність електронів, більшу кількість реактивних видів та збільшені розміри плазми порівняно з вищими частотами модуляції.

Обидва дослідження зосереджені на оптимізації параметрів плазми для різних застосувань. У статті наголошується на важливості контролю параметрів плазми, таких як температура газу та генерація реактивних видів, що є критично важливим для промислових та біомедичних застосувань. У моїй роботі акцент робиться на розробці системи керування, яка може ефективно збирати та обробляти дані спектрів випромінювання плазми, що також має велике значення для аналізу та оптимізації плазмових процесів.

Таким чином, обидва дослідження доповнюють одне одного, надаючи як теоретичну базу для розуміння поведінки плазми, так і практичні рішення для керування та обробки даних плазмових процесів.

4. Архітектура STM32

4.1. Опис архітектури

STM32 — це сімейство мікроконтролерів, розроблених компанією STMicroelectronics, які базуються на архітектурі ARM Cortex-M. Ці мікроконтролери широко використовуються в різних застосуваннях завдяки

своїй високій продуктивності, енергоефективності та гнучкості. Розглянемо основні компоненти та особливості архітектури STM32, зокрема моделі STM32F103C6T6, яка використовується в даній магістерській роботі.

STM32F103C6T6 базується на ядрі ARM Cortex-M3, яке є 32-розрядним RISC-процесором. Основні характеристики цього ядра включають:

- Тактова частота: до 72 МГц.
- Архітектура: 32-розрядна, що забезпечує високу продуктивність та ефективність обробки даних.
- Набір інструкцій: ARMv7-M, який включає інструкції для роботи з цілими числами, числами з рухомою комою та DSP-операції.
- Підтримка переривань: швидке реагування на зовнішні події завдяки механізму переривань.

STM32F103C6T6 має наступні типи пам'яті:

- Flash-пам'ять: 64 КБ для зберігання програмного забезпечення та констант.
- Оперативна пам'ять (SRAM): 20 КБ для тимчасового зберігання даних під час виконання програм.
- EEPROM: емульована флеш-пам'ять для зберігання конфігураційних даних.

STM32F103C6T6 оснащений широким набором периферійних пристроїв, які розширюють його функціональні можливості:

- Таймери: до 16 таймерів, включаючи загального призначення, PWM та таймери захоплення/порівняння.
- Аналого-цифрові перетворювачі (ADC): до 12 каналів з роздільною здатністю до 12 біт.
- Цифро-аналогові перетворювачі (DAC): до 2 каналів з роздільною здатністю до 12 біт.
- Інтерфейси зв'язку: підтримка UART, SPI, I2C, CAN та USB.
- GPIO: до 51 виводів загального призначення, які можуть бути налаштовані як входи або виходи.
- DMA: прямий доступ до пам'яті для швидкої передачі даних між периферійними пристроями та пам'яттю.

STM32F103C6T6 підтримує широкий діапазон напруг живлення (2.0-3.6 В) та має вбудовані регулятори напруги для забезпечення стабільної роботи. Мікроконтролер також має режими низького енергоспоживання, такі як режим сну та режим очікування, що дозволяє зменшити споживання енергії в періоди простою.

STM32F103C6T6 має вбудовані механізми захисту, такі як:

- Захист від перезапису: обмеження доступу до флеш-пам'яті для запобігання несанкціонованим змінам.
- Захист від читання: обмеження доступу до даних, збережених у пам'яті.

- Захист від переповнення стеку: запобігання переповненню стеку під час виконання програм.

Для розробки програмного забезпечення для STM32F103C6T6 використовуються наступні інструменти:

- STM32CubeMX: графічний інструмент для конфігурування мікроконтролера та генерації стартового коду.
- STM32CubeIDE: інтегроване середовище розробки, яке включає компілятор, дебагер та інші інструменти для розробки та налагодження програм.
- STM32CubeF1: бібліотека драйверів та прикладів для роботи з периферійними пристроями мікроконтролера.

STM32F103C6T6 широко використовується в різних застосуваннях, включаючи:

- Промислова автоматизація: керування роботами, конвеєрами та іншими промисловими процесами.
- Споживча електроніка: побутова техніка, така як пральні машини, мікрохвильові печі, кондиціонери.
- Автомобільна електроніка: керування системами двигуна, антиблокувальними гальмами (ABS), подушками безпеки.
- Медичні пристрої: портативні медичні пристрої, такі як глюкометри, пульсоксиметри, кардіостимулятори.
- Комунікаційні системи: керування модемами, маршрутизаторами, бездротовими мережами.

4.2. Переваги та недоліки

Мікроконтролер STM32F103C6T6 є популярним вибором для різних застосувань завдяки своїм високим характеристикам та гнучкості. Однак, як і будь-який інший пристрій, він має свої переваги та недоліки. Розглянемо їх детальніше.

Переваги

- Висока продуктивність
    - 32-розрядне ядро ARM Cortex-M3: забезпечує високу швидкість обробки даних та ефективність виконання програм.
    - Тактова частота до 72 МГц: дозволяє виконувати складні обчислення в реальному часі.
- Широкий набір периферійних пристроїв
    - Таймери, ADC, DAC, GPIO: дозволяють реалізовувати різноманітні функції, такі як керування моторами, вимірювання аналогових сигналів та обробка цифрових даних.
    - Інтерфейси зв'язку (UART, SPI, I2C, CAN, USB): забезпечують гнучкість та сумісність з різними пристроями та мережами.
- Висока енергоефективність

    - Режими низького енергоспоживання: дозволяють зменшити споживання енергії в періоди простою, що є важливим для портативних та автономних пристроїв.
    - Широкий діапазон напруг живлення (2.0-3.6 В): забезпечує стабільну роботу в різних умовах.
- Висока надійність та стійкість
    - Захист від перезапису та читання: забезпечує безпеку даних та програмного забезпечення.
    - Захист від переповнення стеку: запобігає помилкам під час виконання програм.
- Простота розробки та налагодження
    - STM32CubeMX та STM32CubeIDE: інструменти розробки спрощують процес конфігурування мікроконтролера, генерації стартового коду та налагодження програм.
    - Багата бібліотека драйверів (STM32CubeF1): дозволяє швидко інтегрувати периферійні пристрої та створювати функціональні програми.
- Доступність та підтримка
    - Широка доступність: мікроконтролери STM32F103C6T6 легко доступні на ринку та мають конкурентоспроможну ціну.
    - Активна спільнота розробників: велика кількість ресурсів, прикладів та форумів, де можна знайти допомогу та поради.

Недоліки

- 64 КБ Flash-пам'ять та 20 КБ SRAM: може бути недостатньо для дуже складних застосувань, які вимагають великого обсягу програмного забезпечення та даних.
- Порівняно з деякими конкурентами: STM32F103C6T6 може бути дорожчим за деякі інші мікроконтролери з подібними характеристиками.
- Багатофункціональність: широкий набір периферійних пристроїв та можливостей може ускладнити процес налаштування та інтеграції для новачків.
- Документація: хоча документація є досить повною, вона може бути складною для розуміння без достатнього досвіду.
- Відсутність вбудованої підтримки деяких сучасних протоколів зв'язку: може вимагати додаткових модулів або зовнішніх пристроїв для реалізації деяких функцій.
- Високе споживання енергії при максимальній продуктивності: може бути проблемою для застосувань, де важлива автономність та тривалий час роботи від батареї.
- Підтримка лише деяких операційних систем: може вимагати додаткових зусиль для інтеграції з певними платформами або системами.

4.3. Приклади застосування

Мікроконтролер STM32F103C6T6, завдяки своїй високій продуктивності, енергоефективності та широкому набору периферійних пристроїв, знаходить застосування в різноманітних галузях. Розглянемо деякі конкретні приклади застосувань, які демонструють його універсальність та ефективність.

Керування роботами:

- Опис: використання STM32F103C6T6 для керування рухом роботів, включаючи управління двигунами, сенсорами та виконавчими механізмами.
- Переваги: висока точність та швидкість обробки даних, можливість інтеграції з різними датчиками та пристроями зв'язку.

Конвеєрні системи:

- Опис: керування конвеєрними лініями, включаючи моніторинг стану, управління швидкістю та синхронізацію роботи різних вузлів.
- Переваги: надійність та стійкість до електромагнітних перешкод, можливість роботи в режимах низького енергоспоживання.

Системи моніторингу та діагностики:

- Опис: збір та аналіз даних з різних датчиків для моніторингу стану обладнання та виявлення аномалій.
- Переваги: висока точність вимірювань, можливість обробки великих обсягів даних в реальному часі.

Побутова техніка:

- Опис: використання STM32F103C6T6 в пральних машинах, мікрохвильових печах, кондиціонерах та інших побутових пристроях для керування роботою та інтерфейсом користувача.
- Переваги: енергоефективність, компактність, можливість інтеграції з сенсорними екранами та бездротовими модулями.

Розумний дім:

- Опис: керування системами розумного дому, такими як освітлення, опалення, безпека та автоматизація.
- Переваги: можливість інтеграції з різними протоколами зв'язку (Wi-Fi, Zigbee, Bluetooth), висока надійність та стійкість до перешкод.

Системи керування двигуном:

- Опис: використання STM32F103C6T6 для керування роботою двигуна, включаючи регулювання паливної суміші, контроль викидів та оптимізацію споживання палива.

- Переваги: висока продуктивність, можливість роботи в екстремальних умовах, надійність та стійкість до перешкод.

Антиблокувальні гальмівні системи (ABS):

- Опис: керування роботою антиблокувальних гальмівних систем для запобігання блокуванню коліс та покращення керованості автомобіля.
- Переваги: швидке реагування на зовнішні події, висока точність обробки даних.

Подушки безпеки:

- Опис: керування роботою подушок безпеки, включаючи виявлення зіткнень та активацію подушок.
- Переваги: надійність, швидке реагування, можливість інтеграції з іншими системами безпеки.

Портативні медичні пристрої:

- Опис: використання STM32F103C6T6 в глюкометрах, пульсоксиметрах, кардіостимуляторах та інших портативних медичних пристроях.
- Переваги: енергоефективність, компактність, висока точність вимірювань, можливість роботи в режимах низького енергоспоживання.

Системи моніторингу пацієнтів:

- Опис: збір та аналіз даних з різних медичних сенсорів для моніторингу стану пацієнтів в реальному часі.
- Переваги: висока надійність, можливість інтеграції з бездротовими модулями, обробка великих обсягів даних.

Модеми та маршрутизатори:

- Опис: використання STM32F103C6T6 для керування роботою модемів та маршрутизаторів, включаючи обробку мережевих пакетів та управління з'єднаннями.
- Переваги: висока продуктивність, можливість роботи з різними протоколами зв'язку, надійність та стійкість до перешкод.

Бездротові мережі:

- Опис: керування роботою бездротових мереж, включаючи Wi-Fi, Bluetooth, Zigbee та інші протоколи.
- Переваги: можливість інтеграції з різними бездротовими модулями, висока енергоефективність, надійність.

Системи збору даних в плазмових установках:

- Опис: використання STM32F103C6T6 для збору та обробки даних з різних датчиків в плазмових установках.
- Переваги: висока точність вимірювань, можливість роботи в умовах високих електромагнітних перешкод, надійність та стійкість.

Експериментальні стенди:

- Опис: керування роботою експериментальних стендів для проведення наукових досліджень та тестування нових технологій.
- Переваги: гнучкість, можливість інтеграції з різними датчиками та пристроями зв'язку, висока продуктивність.

5. Протокол RS-485

5.1. Опис протоколу

RS-485 — це стандарт послідовного зв'язку, який широко використовується в промислових та комерційних застосуваннях для передачі даних на великі відстані та в умовах електромагнітних перешкод. Протокол RS-485 був розроблений для забезпечення надійної та ефективної передачі даних між різними пристроями. Розглянемо детальний опис протоколу RS-485, його переваги, недоліки та застосування.

Топологія зв'язку:

- Багатоточковий зв'язок: RS-485 підтримує багатоточковий зв'язок, що дозволяє підключати до 32 пристроїв до однієї лінії зв'язку.
- Типи топологій: лінійна, кільцева та зіркова топології.

Швидкість передачі даних:

- Максимальна швидкість: до 10 Мбіт/с.
- Залежність від довжини лінії: швидкість передачі даних залежить від довжини лінії зв'язку. На великих відстанях швидкість може бути обмежена.

Довжина лінії зв'язку:

- Максимальна довжина: до 1200 метрів при швидкості 100 кбіт/с.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії зменшується.

Типи сигналів:

- Диференціальний сигнал: використання диференціального сигналу забезпечує високу стійкість до електромагнітних перешкод.
- Лінії A та B: для передачі даних використовуються дві лінії — A (негативний сигнал) та B (позитивний сигнал).

5.2. Переваги та недоліки

Протокол RS-485 є одним з найпоширеніших стандартів послідовного зв'язку, який широко використовується в промислових та комерційних застосуваннях. Він забезпечує надійну та ефективну передачу даних на великі відстані та в умовах електромагнітних перешкод. Однак, як і будь-який інший протокол, RS-485 має свої переваги та недоліки. Розглянемо їх детальніше.

Переваги

- Диференціальний сигнал: використання диференціального сигналу дозволяє зменшити вплив електромагнітних перешкод, що робить RS-485 ідеальним вибором для промислових застосувань.
- Екранування: можливість використання екранованих кабелів для додаткового захисту від перешкод.
- Підключення до 32 пристроїв: можливість підключення великої кількості пристроїв до однієї лінії зв'язку спрощує архітектуру мережі та зменшує витрати на обладнання.
- Гнучкість: підтримка різних топологій зв'язку, таких як лінійна, кільцева та зіркова.
- Довжина лінії до 1200 метрів: RS-485 дозволяє передавати дані на великі відстані, що є важливим для промислових та комерційних застосувань.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії зменшується, але все одно залишається значною.
- До 10 Мбіт/с: висока швидкість передачі даних забезпечує ефективну роботу мережі та швидке обмін інформацією між пристроями.
- Гнучкість: можливість налаштування швидкості передачі даних в залежності від потреб застосування.
- Сумісність з різними пристроями: RS-485 підтримує інтеграцію з широким спектром пристроїв, включаючи датчики, контролери та інтерфейси користувача.
- Стандартизовані інтерфейси: використання стандартизованих інтерфейсів спрощує процес розробки та впровадження рішень.
- Висока надійність: RS-485 забезпечує надійну передачу даних навіть в умовах високих електромагнітних перешкод.
- Стійкість до збоїв: використання диференціального сигналу та екранування забезпечує стійкість до збоїв та помилок передачі.

Недоліки

- Залежність від довжини лінії: при збільшенні довжини лінії зв'язку швидкість передачі даних зменшується, що може бути обмеженням для деяких застосувань.
- Компроміс: необхідність вибору між швидкістю передачі даних та довжиною лінії зв'язку.

- Конфігурація мережі: налаштування багатоточкового зв'язку може вимагати додаткових зусиль та знань, особливо при підключенні великої кількості пристроїв.
- Синхронізація: необхідність синхронізації роботи пристроїв для забезпечення надійної передачі даних.
- Якість кабелів: для забезпечення надійної передачі даних необхідно використовувати якісні кабелі, що може збільшити витрати на обладнання.
- Екранування: вимога до використання екранованих кабелів для захисту від електромагнітних перешкод.
- Максимум 32 пристрої: хоча RS-485 підтримує багатоточковий зв'язок, кількість підключених пристроїв обмежена 32, що може бути недостатнім для деяких великих мереж.
- Складність масштабування: збільшення кількості пристроїв може вимагати додаткових зусиль для забезпечення надійної передачі даних.
- Необхідність додаткових механізмів: RS-485 не має вбудованих механізмів корекції помилок, що може вимагати додаткових зусиль для забезпечення надійності передачі даних.
- Програмне забезпечення: необхідність розробки програмного забезпечення для виявлення та корекції помилок.
- Високе споживання енергії: при максимальній продуктивності RS-485 може споживати значну кількість енергії, що може бути проблемою для автономних та портативних пристроїв.
- Оптимізація: необхідність оптимізації енергоспоживання для забезпечення тривалої роботи від батареї.

6. Порівняння STM32 з іншими мікроконтролерами

6.1. Порівняння з Arduino

Arduino є однією з найпопулярніших платформ для розробки електронних проектів, особливо серед початківців та хобістів. Порівняння мікроконтролера STM32F103C6T6 з Arduino дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Тактова частота:

- STM32F103C6T6: має тактову частоту до 72 МГц, що забезпечує високу продуктивність та швидкість обробки даних.
- Arduino (наприклад, Arduino Uno): має тактову частоту 16 МГц, що є значно нижче порівняно з STM32.

Споживання енергії:

- STM32F103C6T6: має режими низького енергоспоживання, які дозволяють зменшити споживання енергії в періоди простою.

- Arduino: має обмежені можливості для управління енергоспоживанням, що може бути проблемою для автономних та портативних пристроїв.

Сумісність з іншими пристроями:

- STM32F103C6T6: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, CAN та USB, що забезпечує високу сумісність з різними пристроями.
- Arduino: обмежений можливостями UART та SPI, що може ускладнити інтеграцію з деякими пристроями.

Ціна компонентів:

- STM32F103C6T6: має конкурентоспроможну ціну, але може бути дорожчим порівняно з Arduino.
- Arduino: є однією з найдоступніших платформ для розробки електронних проектів, що робить її ідеальним вибором для початківців та хобістів.

Доступність на ринку:

- STM32F103C6T6: широко доступний на ринку та має активну підтримку від виробника.
- Arduino: широко доступний на ринку та має велику спільноту розробників, яка надає підтримку та ресурси.

Доступність інструментів розробки:

- STM32F103C6T6: має потужні інструменти розробки, такі як STM32CubeMX та STM32CubeIDE, які надають широкі можливості для розробки та налагодження програмного забезпечення.
- Arduino: має простий та інтуїтивно зрозумілий інтерфейс розробки (Arduino IDE), який ідеально підходить для початківців.

6.2. Порівняння з ESP32

ESP32 є популярним мікроконтролером, який широко використовується в різних застосуваннях, особливо в IoT (Internet of Things) та бездротових проектах. Порівняння мікроконтролера STM32F103C6T6 та ESP32 дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Тактова частота:

- STM32F103C6T6: має тактову частоту до 72 МГц, що забезпечує високу продуктивність та швидкість обробки даних.
- ESP32: має тактову частоту до 240 МГц, що робить його значно продуктивнішим порівняно з STM32F103C6T6.

Споживання енергії:

- STM32F103C6T6: має режими низького енергоспоживання, які дозволяють зменшити споживання енергії в періоди простою.
- ESP32: має режими глибокого сну та низького енергоспоживання, які дозволяють значно зменшити споживання енергії, що є важливим для автономних та портативних пристроїв.

Сумісність з іншими пристроями:

- STM32F103C6T6: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, CAN та USB, що забезпечує високу сумісність з різними пристроями.
- ESP32: підтримує Wi-Fi, Bluetooth, UART, SPI, I2C, що забезпечує високу сумісність з бездротовими пристроями та іншими компонентами.

Ціна компонентів:

- STM32F103C6T6: має конкурентоспроможну ціну, але може бути дорожчим порівняно з ESP32.
- ESP32: є одним з найдоступніших мікроконтролерів для IoT та бездротових застосувань, що робить його ідеальним вибором для бюджетних проектів.

Доступність на ринку:

- STM32F103C6T6: широко доступний на ринку та має активну підтримку від виробника.
- ESP32: широко доступний на ринку та має велику спільноту розробників, яка надає підтримку та ресурси.

Доступність інструментів розробки:

- STM32F103C6T6: має потужні інструменти розробки, такі як STM32CubeMX та STM32CubeIDE, які надають широкі можливості для розробки та налагодження програмного забезпечення.
- ESP32: має інтуїтивно зрозумілий інтерфейс розробки (ESP-IDF), який надає широкі можливості для розробки бездротових застосувань.

6.3. Порівняння з PIC

Мікроконтролери сімейства PIC (Peripheral Interface Controller) від компанії Microchip Technology є популярним вибором для різних електронних проектів, особливо в промислових та наукових застосуваннях. Порівняння мікроконтролера STM32F103C6T6 з мікроконтролерами PIC дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Тактова частота:

- STM32F103C6T6: має тактову частоту до 72 МГц, що забезпечує високу продуктивність та швидкість обробки даних.

- PIC: мікроконтролери PIC мають різні тактові частоти, але зазвичай вони нижчі порівняно з STM32. Наприклад, PIC18F4550 має тактову частоту до 48 МГц.

Споживання енергії:

- STM32F103C6T6: має режими низького енергоспоживання, які дозволяють зменшити споживання енергії в періоди простою.
- PIC: мікроконтролери PIC також мають режими низького енергоспоживання, але їх ефективність може бути нижчою порівняно з STM32.

Сумісність з іншими пристроями:

- STM32F103C6T6: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, CAN та USB, що забезпечує високу сумісність з різними пристроями.
- PIC: підтримує UART, SPI, I2C, але може бути обмежений можливостями інтеграції з деякими пристроями.

Ціна компонентів:

- STM32F103C6T6: має конкурентоспроможну ціну, але може бути дорожчим порівняно з деякими моделями PIC.
- PIC: є одним з найдоступніших мікроконтролерів для промислових та наукових застосувань, що робить його ідеальним вибором для бюджетних проектів.

Доступність на ринку:

- STM32F103C6T6: широко доступний на ринку та має активну підтримку від виробника.
- PIC: широко доступний на ринку та має велику спільноту розробників, яка надає підтримку та ресурси.

Доступність інструментів розробки:

- STM32F103C6T6: має потужні інструменти розробки, такі як STM32CubeMX та STM32CubeIDE, які надають широкі можливості для розробки та налагодження програмного забезпечення.
- PIC: має інтуїтивно зрозумілий інтерфейс розробки (MPLAB X IDE), який надає широкі можливості для розробки та налагодження програмного забезпечення.

7. Порівняння RS-485 з іншими протоколами зв'язку

7.1. Порівняння з CAN

Протокол CAN (Controller Area Network) є одним з найпоширеніших стандартів для зв'язку в автомобільній та промисловій автоматизації. Порівняння протоколу

RS-485 з CAN дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Швидкість передачі даних

RS-485:

- Максимальна швидкість: до 10 Мбіт/с.
- Залежність від довжини лінії: швидкість передачі даних зменшується з збільшенням довжини лінії зв'язку.

CAN:

- Максимальна швидкість: до 1 Мбіт/с.
- Залежність від довжини лінії: швидкість передачі даних також зменшується з збільшенням довжини лінії зв'язку, але CAN має більш стабільну роботу на великих відстанях.

Довжина лінії зв'язку

RS-485:

- Максимальна довжина: до 1200 метрів при швидкості 100 кбіт/с.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії зменшується.

CAN:

- Максимальна довжина: до 1000 метрів при швидкості 50 кбіт/с.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії також зменшується, але CAN забезпечує більш стабільну роботу на великих відстанях.

Кількість підключених пристроїв

RS-485:

- Максимальна кількість: до 32 пристроїв.
- Топологія: підтримує лінійну, кільцеву та зіркову топології.

CAN:

- Максимальна кількість: до 112 пристроїв.
- Топологія: підтримує лінійну топологію.

Стійкість до електромагнітних перешкод

RS-485:

- Диференціальний сигнал: використовує диференціальний сигнал, що забезпечує високу стійкість до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

CAN:

- Диференціальний сигнал: використовує диференціальний сигнал, що також забезпечує високу стійкість до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

Надійність та захист від помилок

RS-485:

- Захист від помилок: вимагає додаткових механізмів захисту від помилок на рівні програмного забезпечення.
- Повторне відправлення: не має вбудованих механізмів автоматичного повторного відправлення.

CAN:

- Захист від помилок: має вбудовані механізми захисту від помилок, такі як CRC (Cyclic Redundancy Check) та автоматичне повторне відправлення.
- Приоритетизація: підтримує приоритетизацію повідомлень, що дозволяє забезпечити надійну передачу даних у критичних ситуаціях.

Простота інтеграції

RS-485:

- Сумісність: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, що забезпечує високу сумісність з різними пристроями.
- Топологія: підтримує різні топології мережі, що спрощує інтеграцію.

CAN:

- Сумісність: широко використовується в автомобільній та промисловій автоматизації, що забезпечує високу сумісність з відповідними пристроями.
- Топологія: підтримує лінійну топологію, що може бути обмеженням для деяких застосувань.

Вартість

RS-485:

- Ціна компонентів: має конкурентоспроможну ціну, але може вимагати додаткових витрат на кабелі та трансивери.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

CAN:

- Ціна компонентів: має середню ціну, але може вимагати додаткових витрат на контролери та трансивери.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

Програмне забезпечення та інструменти розробки

RS-485:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки для різних платформ.
- Налагодження: вимагає додаткових зусиль для налагодження та реалізації механізмів захисту від помилок.

CAN:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки, особливо для автомобільних та промислових застосувань.
- Налагодження: має вбудовані механізми захисту від помилок та автоматичне повторне відправлення, що спрощує налагодження.

7.2. Порівняння з I2C

Протокол I2C (Inter-Integrated Circuit) є одним з найпоширеніших стандартів для послідовного зв'язку, який широко використовується в електронних пристроях для зв'язку між мікроконтролерами та периферійними компонентами. Порівняння протоколу RS-485 з I2C дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Швидкість передачі даних

RS-485:

- Максимальна швидкість: до 10 Мбіт/с.
- Залежність від довжини лінії: швидкість передачі даних зменшується з збільшенням довжини лінії зв'язку.

I2C:

- Максимальна швидкість: до 400 кбіт/с (стандартний режим), до 1 Мбіт/с (швидкий режим), до 3.4 Мбіт/с (високошвидкісний режим).
- Залежність від довжини лінії: швидкість передачі даних також зменшується з збільшенням довжини лінії зв'язку, але I2C зазвичай використовується для коротких відстаней.

Довжина лінії зв'язку

RS-485:

- Максимальна довжина: до 1200 метрів при швидкості 100 кбіт/с.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії зменшується.

I2C:

- Максимальна довжина: зазвичай до 1 метра, але може бути збільшена до 10 метрів за допомогою буферизації.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії також зменшується.

Кількість підключених пристроїв

RS-485:

- Максимальна кількість: до 32 пристроїв.
- Топологія: підтримує лінійну, кільцеву та зіркову топології.

I2C:

- Максимальна кількість: до 127 пристроїв.
- Топологія: підтримує лінійну топологію, але може бути використана зіркова топологія з додатковими схемами.

Стійкість до електромагнітних перешкод

RS-485:

- Диференціальний сигнал: використовує диференціальний сигнал, що забезпечує високу стійкість до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

I2C:

- Однонаправлений сигнал: використовує однонаправлений сигнал, який є менш стійким до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

Надійність та захист від помилок

RS-485:

- Захист від помилок: вимагає додаткових механізмів захисту від помилок на рівні програмного забезпечення.
- Повторне відправлення: не має вбудованих механізмів автоматичного повторного відправлення.

I2C:

- Захист від помилок: має вбудовані механізми захисту від помилок, такі як виявлення колізій та контроль цілісності даних.
- Повторне відправлення: не має вбудованих механізмів автоматичного повторного відправлення, але може бути реалізовано на рівні програмного забезпечення.

Простота інтеграції

RS-485:

- Сумісність: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, що забезпечує високу сумісність з різними пристроями.
- Топологія: підтримує різні топології мережі, що спрощує інтеграцію.

I2C:

- Сумісність: широко використовується в електронних пристроях, що забезпечує високу сумісність з різними компонентами.
- Топологія: підтримує лінійну топологію, що може бути обмеженням для деяких застосувань.

Вартість

RS-485:

- Ціна компонентів: має конкурентоспроможну ціну, але може вимагати додаткових витрат на кабелі та трансивери.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

I2C:

- Ціна компонентів: має низьку ціну, що робить його ідеальним вибором для бюджетних проектів.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

Програмне забезпечення та інструменти розробки

RS-485:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки для різних платформ.
- Налагодження: вимагає додаткових зусиль для налагодження та реалізації механізмів захисту від помилок.

I2C:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки, особливо для вбудованих систем.
- Налагодження: має вбудовані механізми захисту від помилок, що спрощує налагодження.

7.3. Порівняння з SPI

Протокол SPI (Serial Peripheral Interface) є одним з найпоширеніших стандартів для синхронного послідовного зв'язку, який широко використовується для зв'язку між мікроконтролерами та периферійними пристроями. Порівняння протоколу RS-485 з SPI дозволяє виявити їхні переваги та недоліки, а також визначити, в яких випадках кожен з них є оптимальним вибором. Розглянемо основні аспекти порівняння.

Швидкість передачі даних:

RS-485:

- Максимальна швидкість: до 10 Мбіт/с.
- Залежність від довжини лінії: швидкість передачі даних зменшується з збільшенням довжини лінії зв'язку.

SPI:

- Максимальна швидкість: до 50 Мбіт/с (залежно від тактової частоти мікроконтролера).
- Залежність від довжини лінії: швидкість передачі даних не залежить від довжини лінії, але обмежена фізичними характеристиками кабелів та мікроконтролерів.

Довжина лінії зв'язку:

RS-485:

- Максимальна довжина: до 1200 метрів при швидкості 100 кбіт/с.
- Залежність від швидкості: при збільшенні швидкості передачі даних максимальна довжина лінії зменшується.

SPI:

- Максимальна довжина: зазвичай обмежена декількома метрами через фізичні обмеження кабелів та мікроконтролерів.

- Залежність від швидкості: швидкість передачі даних не залежить від довжини лінії, але обмежена фізичними характеристиками.

Кількість підключених пристроїв:

RS-485:

- Максимальна кількість: до 32 пристроїв.
- Топологія: підтримує лінійну, кільцеву та зіркову топології.

SPI:

- Максимальна кількість: зазвичай один ведучий (master) та один ведений (slave) пристрій, але можлива реалізація багатоточкового зв'язку за допомогою мультиплексорів.
- Топологія: підтримує зіркову топологію, але вимагає додаткових схем для багатоточкового зв'язку.

Стійкість до електромагнітних перешкод

RS-485:

- Диференціальний сигнал: використовує диференціальний сигнал, що забезпечує високу стійкість до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

SPI:

- Однонаправлений сигнал: використовує однонаправлений сигнал, який є менш стійким до електромагнітних перешкод.
- Екранування: можливість використання екранованих кабелів для додаткового захисту.

Надійність та захист від помилок:

RS-485:

- Захист від помилок: вимагає додаткових механізмів захисту від помилок на рівні програмного забезпечення.
- Повторне відправлення: не має вбудованих механізмів автоматичного повторного відправлення.

SPI:

- Захист від помилок: не має вбудованих механізмів захисту від помилок, але може бути реалізовано на рівні програмного забезпечення.
- Повторне відправлення: не має вбудованих механізмів автоматичного повторного відправлення, але може бути реалізовано на рівні програмного забезпечення.

Простота інтеграції:

RS-485:

- Сумісність: підтримує різні інтерфейси зв'язку, такі як UART, SPI, I2C, що забезпечує високу сумісність з різними пристроями.
- Топологія: підтримує різні топології мережі, що спрощує інтеграцію.

SPI:

- Сумісність: широко використовується в електронних пристроях, що забезпечує високу сумісність з різними компонентами.
- Топологія: підтримує зіркову топологію, але вимагає додаткових схем для багатоточкового зв'язку.

Вартість:

RS-485:

- Ціна компонентів: має конкурентоспроможну ціну, але може вимагати додаткових витрат на кабелі та трансивери.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

SPI:

- Ціна компонентів: має низьку ціну, що робить його ідеальним вибором для бюджетних проектів.
- Доступність: широко доступний на ринку та має активну підтримку від виробників.

Програмне забезпечення та інструменти розробки:

RS-485:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки для різних платформ.
- Налагодження: вимагає додаткових зусиль для налагодження та реалізації механізмів захисту від помилок.

SPI:

- Доступність інструментів: має широкий вибір програмного забезпечення та інструментів розробки, особливо для вбудованих систем.
- Налагодження: вимагає додаткових зусиль для реалізації механізмів захисту від помилок на рівні програмного забезпечення.

8. Розробка пристрою

8.1. Постановка задачі

У даному розділі розглядається процес розробки пристрою на базі мікроконтролера STM32F103C6T6, призначеного для передачі даних між персональним комп'ютером (ПК) та сторонніми пристроями через інтерфейс RS-485. Основна мета розробки полягає в створенні надійного та ефективного засобу збору та передачі даних, необхідних для моніторингу параметрів плазми в плазмовій установці.

Основні завдання, які необхідно вирішити в ході розробки:

- Вибір мікроконтролера та периферійних компонентів:
  - Обрати мікроконтролер, який забезпечує необхідну продуктивність та функціональність для обробки даних в реальному часі.
  - Визначити та підібрати необхідні периферійні компоненти, такі як трансивери RS-485, USB-конвертери та інші елементи схеми.
- Розробка апаратної частини:
  - Спроектувати схему підключення мікроконтролера до ПК через інтерфейс USB та до сторонніх пристроїв через RS-485.
  - Забезпечити надійне живлення та захист схеми від можливих перешкод і помилок.
- Розробка програмного забезпечення:
  - Реалізувати алгоритми обробки та передачі даних, включаючи протоколи зв'язку та управління потоками даних.
  - Розробити програмне забезпечення для взаємодії з ПК, яке дозволить користувачеві керувати пристроєм та відображати отримані дані.
- Тестування та налагодження:
  - Провести комплексне тестування апаратної та програмної частин пристрою для перевірки їх надійності та ефективності.
  - Виявити та усунути можливі помилки та недоліки, оптимізувати роботу системи.
- Документування та опис роботи пристрою:
  - Скласти технічну документацію, яка описує принципи роботи пристрою, схеми підключення та алгоритми програмного забезпечення.
  - Підготувати інструкції для користувачів та рекомендації з експлуатації пристрою

8.2. Вибір компонентів

У процесі розробки пристрою для передачі даних між персональним комп'ютером (ПК) та сторонніми пристроями було здійснено ретельний вибір компонентів, які забезпечують надійну та ефективну роботу системи. Розглянемо основні компоненти, обрані для реалізації даного проекту.

Мікроконтролер:

Для виконання центральних функцій пристрою був обраний мікроконтролер STM32F103C6T6. Цей мікроконтролер має наступні переваги:

- Висока продуктивність та енергоефективність.
- Наявність необхідних периферійних модулів, таких як UART для зв'язку з іншими пристроями.
- Можливість роботи в реальному часі, що є критично важливим для задач моніторингу параметрів плазми.

Інтерфейс зв'язку з ПК:

Для обміну даними з ПК через USB був обраний перетворювач CH340. Цей компонент забезпечує:

- Простий та надійний інтерфейс USB-UART.
- Сумісність з різними операційними системами без необхідності встановлення додаткових драйверів.
- Високу швидкість передачі даних, що дозволяє оперативно обмінюватися інформацією між ПК та мікроконтролером.

Інтерфейс RS-485:

Для реалізації зв'язку по протоколу RS-485 був вибраний трансивер MAX485. Цей компонент має такі характеристики:

- Висока стійкість до перешкод та електромагнітних впливів.
- Можливість передачі даних на великі відстані, що є важливим для підключення сторонніх пристроїв.
- Надійна робота в умовах промислового застосування.

Гальванічна розв'язка:

Для забезпечення гальванічної розв'язки між ПК та мікроконтролером було використано оптопару. Оптопара забезпечує:

- Електричну ізоляцію між мікроконтролером та ПК, що запобігає пошкодженню компонентів через різницю потенціалів.
- Зменшення впливу електромагнітних перешкод на передачу даних.
- Підвищення надійності та стійкості системи в цілому.

Фізичне з'єднання:

Для фізичного підключення до сторонніх пристроїв використовується інтерфейс RJ-45. Цей стандартний інтерфейс дозволяє:

- Легко підключати та відключати пристрої без необхідності використання спеціальних інструментів.
- Забезпечувати надійне механічне з'єднання, стійке до вібрацій та механічних впливів.

- Використовувати стандартні мережеві кабелі, що спрощує монтаж та обслуговування системи.

8.3. Інтерфейс користувача для ПК

Для забезпечення зручної взаємодії з розробленим пристроєм було створено програмне забезпечення з графічним інтерфейсом користувача (GUI) на мові програмування Python. Інтерфейс дозволяє користувачеві керувати процесом обміну даними між ПК та сторонніми пристроями, а також відображати отримані дані в реальному часі.

Основні функції інтерфейсу:

- Вибір та підключення COM-порту:
  - Користувач може вибрати доступний COM-порт з випадаючого списку та підключитися до пристрою.
  - Після успішного підключення інтерфейс відображає повідомлення про статус з'єднання.
- Налаштування адрес:
  - Інтерфейс дозволяє користувачеві конфігурувати адреси сторонніх пристроїв, присвоюючи їм зрозумілі імена для зручності ідентифікації.
  - Для цього передбачено діалогове вікно, де можна додавати, редагувати та видаляти адреси.
- Відправка та прийом повідомлень:
  - Користувач може вводити адресу та повідомлення для відправки на сторонній пристрій.
  - Відправлені та отримані повідомлення відображаються в таблиці з вказаними часовими мітками, напрямком (відправка/прийом), адресою, ім'ям пристрою та змістом повідомлення.
- Збереження даних:
  - Інтерфейс дозволяє зберігати історію обміну повідомленнями у форматі CSV для подальшого аналізу.
  - Користувач може вибрати місце для збереження файлу та ініціювати процес збереження.
- Системні повідомлення:
  - У нижній частині інтерфейсу передбачено текстове поле для відображення системних повідомлень, таких як стан з'єднання та помилки.

Технічна реалізація:

Інтерфейс користувача реалізовано за допомогою бібліотеки PyQt5, яка забезпечує можливість створення багатофункціональних графічних застосунків. Основні компоненти інтерфейсу включають:

- QComboBox для вибору COM-порту.

- QPushButton для керування діями (підключення, відправка повідомлень тощо).
- QTableWidget для відображення таблиці повідомлень.
- QTextEdit для відображення системних повідомлень.
- QDialog для налаштування адрес.

Принцип роботи:

- Ініціалізація:
  - При запуску програми інтерфейс сканує доступні COM-порти та заповнює випадаючий список.
  - Завантажуються конфігураційні дані з файлу config.json, якщо такий існує.
- Обмін даними:
  - Після підключення до COM-порту запускається окремий потік для читання даних з пристрою.
  - Отримані повідомлення обробляються та відображаються в інтерфейсі.
- Збереження конфігурації:
  - При закритті програми поточна конфігурація (вибраний порт, адреси тощо) зберігається у файл для подальшого використання.

Інтерфейс користувача забезпечує інтуїтивно зрозумілий та зручний спосіб взаємодії з розробленим пристроєм, дозволяючи ефективно керувати процесом обміну даними та аналізувати отримані результати.

9. Практичне значення

9.1. Значення розробки для науки

Розробка пристрою на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485 має значне значення для наукових досліджень, особливо в галузі плазмових технологій. Цей пристрій виконує роль центрального вузла збору та передачі даних, що дозволяє інтегрувати різноманітні сторонні пристрої в єдину систему моніторингу та управління.

Внесок у наукові дослідження:

- Покращення якості даних:

  Завдяки можливості збирати дані з різних джерел в реальному часі, пристрій дозволяє отримувати більш повну та точну картину стану плазмової установки. Це сприяє підвищенню якості проведених експериментів та достовірності наукових висновків.

- Інтеграція з різними пристроями:

  Пристрій може працювати в парі з іншими пристроями, які передають запрограмовані дані, такими як контролери потоку газу, сили струму та

напруги. Це дозволяє дослідникам гнучко налаштовувати систему моніторингу під конкретні завдання експерименту.

- Спрощення процесу збору даних:

  Автоматизація процесу збору даних зменшує потребу в ручній обробці інформації, що економить час дослідників та знижує ймовірність людських помилок. Це дозволяє зосередитися на аналізі результатів та розробці нових гіпотез.

- Можливості для масштабування:

  Архітектура пристрою дозволяє легко додавати нові пристрої до системи, розширюючи її функціональність без значних змін. Це відкриває можливості для масштабування досліджень та впровадження нових методик контролю плазми.

- Підвищення надійності експериментів:

  Використання надійних протоколів зв'язку, таких як RS-485, забезпечує стійкість до перешкод та мінімізує втрати даних. Це особливо важливо для довготривалих експериментів, де безперервність моніторингу є критично важливою.

- Сприяння міждисциплінарним дослідженням:

  Пристрій може бути використаний не лише в плазмових технологіях, а й у інших галузях науки та промисловості, де необхідна надійна передача даних між різними системами. Це сприяє обміну досвідом та розвитку міждисциплінарних проектів.

9.2. Економічний аспект

Розробка та впровадження пристрою на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485 має не лише наукове, а й значне економічне значення. Враховуючи потенціал пристрою для оптимізації процесів збору та обробки даних, можна виділити кілька ключових економічних переваг.

Зниження витрат на обладнання:

- Універсальність:

  Пристрій може бути використаний у різних галузях, де необхідна надійна передача даних між системами. Це дозволяє знизити витрати на придбання спеціалізованого обладнання, оскільки один пристрій може виконувати функції декількох.

- Скорочення часу розробки:

Використання готового рішення на базі STM32 дозволяє скоротити час та витрати на розробку нових систем моніторингу та управління. Це особливо важливо для малих та середніх підприємств, які не мають значних ресурсів на дослідження та розробки.

Підвищення ефективності роботи:

- Автоматизація процесів:

  Автоматизація збору та обробки даних дозволяє зменшити потребу в ручній праці, що призводить до зниження операційних витрат. Це також зменшує ймовірність помилок, пов'язаних з людським фактором, та підвищує загальну продуктивність.

- Швидке впровадження:

  Пристрій може бути швидко інтегрований в існуючі системи, що дозволяє оперативно реагувати на зміни у виробничих процесах або наукових дослідженнях. Це сприяє підвищенню гнучкості та адаптивності бізнесу.

Зниження витрат на обслуговування:

- Надійність та довговічність:

  Використання надійних компонентів, таких як мікроконтролер STM32 та трансивер MAX485, забезпечує тривалий термін служби пристрою. Це зменшує витрати на ремонт та технічне обслуговування.

- Можливість оновлення:

  Програмне забезпечення пристрою може бути легко оновлено, що дозволяє адаптувати його до нових вимог без необхідності заміни апаратної частини. Це додатково знижує витрати на модернізацію системи.

Потенціал для комерціалізації:

- Ринок застосування:

  Пристрій має широкий спектр застосування, включаючи наукові дослідження, промисловість, медицину та інші галузі. Це відкриває можливості для комерціалізації та масового виробництва, що може принести значні прибутки.

- Конкурентні переваги:

  Висока функціональність та надійність пристрою дозволяють виділитися на фоні конкурентів, запропонувавши ринку інноваційне рішення для задач моніторингу та управління.

Соціально-економічний вплив:

- Сприяння інноваціям:

  Впровадження такого пристрою сприяє розвитку інноваційних технологій, що може стимулювати економічне зростання та створення нових робочих місць у галузях, пов'язаних з науковими дослідженнями та високими технологіями.

- Підвищення конкурентоспроможності:

  Використання сучасних технологій дозволяє підприємствам підвищити свою конкурентоспроможність на міжнародній арені, що сприяє економічному розвитку країни в цілому.

10. Висновки

10.1. Основні досягнення

У ході розробки пристрою на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485 було досягнуто ряду значних результатів, які підтверджують ефективність та перспективність даного рішення. Основні досягнення включають:

- Успішна інтеграція компонентів:

  Вдалося успішно інтегрувати мікроконтролер STM32F103C6T6 з периферійними компонентами, такими як трансивер MAX485 для зв'язку по RS-485, перетворювач CH340 для USB-зв'язку та оптопари для гальванічної розв'язки. Це забезпечило надійну роботу системи в цілому.

- Розробка програмного забезпечення:

  Створено програмне забезпечення, яке забезпечує ефективний обмін даними між ПК та сторонніми пристроями. Програма включає функції для налаштування адрес, відправки та прийому повідомлень, а також збереження даних у форматі CSV.

  Реалізовано багатопотоковість для паралельної обробки даних, що підвищує продуктивність системи.

- Створення інтуїтивно зрозумілого інтерфейсу користувача:

  Розроблено графічний інтерфейс користувача (GUI) на базі PyQt5, який дозволяє легко керувати процесом обміну даними. Інтерфейс включає зручні елементи управління, такі як вибір COM-порту, налаштування адрес та відображення повідомлень у реальному часі.

- Забезпечення надійного зв'язку:

  Реалізовано надійний та стійкий до перешкод зв'язок по протоколу RS-485, що дозволяє передавати дані на великі відстані без втрати якості. Це

особливо важливо для застосувань у промислових умовах та наукових дослідженнях.

- Можливість масштабування системи:

  Архітектура пристрою дозволяє легко додавати нові сторонні пристрої до системи, розширюючи її функціональність. Це відкриває можливості для масштабування досліджень та впровадження нових методик контролю.

- Підвищення точності та надійності даних:

  Завдяки автоматизації процесу збору даних вдалося значно зменшити ймовірність людських помилок та підвищити точність отриманих результатів. Це сприяє підвищенню якості наукових досліджень та прийняттю обґрунтованих рішень.

- Успішне тестування та налагодження:

  Проведено комплексне тестування пристрою в різних режимах роботи, що підтвердило його надійність та ефективність. Виявлені та усунені помилки, оптимізовано роботу системи для досягнення найкращих результатів.

- Документування та стандартизація:

  Створено технічну документацію, яка описує принципи роботи пристрою, схеми підключення та алгоритми програмного забезпечення. Це сприяє легкому впровадженню та експлуатації системи в різних умовах.

10.2. Рекомендації для подальших досліджень

Розробка пристрою на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485 відкриває широкі перспективи для подальших досліджень та вдосконалень. На основі отриманих результатів та аналізу поточного стану системи можна виділити кілька напрямків, які можуть бути корисними для майбутніх досліджень:

- Розширення функціональності:
  - Підтримка нових протоколів зв'язку: дослідити можливість інтеграції додаткових протоколів зв'язку, таких як Modbus або CAN, для розширення сумісності пристрою з іншими системами.
  - Реалізація додаткових алгоритмів обробки даних: розробити алгоритми для аналізу даних в реальному часі, такі як фільтрація шумів, виявлення аномалій та прогнозування поведінки системи.
- Покращення інтерфейсу користувача:
  - Розробка мобільного додатку: створити мобільний додаток для дистанційного моніторингу та керування пристроєм, що дозволить користувачам отримувати доступ до даних з будь-якого місця.

- Інтеграція з хмарними сервісами: реалізувати можливість збереження та обробки даних у хмарі для підвищення доступності та безпеки інформації.

- Оптимізація апаратної частини:
  - Використання енергоефективних компонентів: дослідити можливість застосування більш енергоефективних мікроконтролерів та периферійних компонентів для зменшення енергоспоживання пристрою.
  - Мініатюризація пристрою: розробити компактнішу версію пристрою для застосування в обмежених просторових умовах.
- Підвищення надійності та безпеки:
  - Реалізація механізмів захисту даних: розробити алгоритми шифрування та аутентифікації для захисту передаваних даних від несанкціонованого доступу.
  - Тестування в екстремальних умовах: провести випробування пристрою в умовах підвищеної температури, вологості та вібрацій для оцінки його стійкості до зовнішніх впливів.
- Інтеграція з системами штучного інтелекту:
  - Використання машинного навчання: дослідити можливість застосування алгоритмів машинного навчання для аналізу великих обсягів даних та виявлення закономірностей, які можуть бути корисними для оптимізації роботи системи.
- Міждисциплінарне застосування:
  - Дослідження в інших галузях: вивчити можливості застосування пристрою в інших галузях, таких як медицина, сільське господарство та транспорт, для розширення сфери його застосування.
- Розробка стандартів та протоколів:
  - Стандартизація рішень: розробити стандарти та рекомендації для інтеграції пристрою в існуючі системи, що сприятиме його широкому впровадженню.

10.3. Підсумки

У ході виконання магістерської роботи було розроблено пристрій на базі мікроконтролера STM32F103C6T6 для керування пристроями на шині RS-485. Пристрій призначений для передачі даних між персональним комп'ютером та сторонніми пристроями, які збирають інформацію про різні параметри, такі як сила струму, напруга та розхід газу, необхідні для моніторингу стану плазми в плазмовій установці.

Основні результати роботи включають:

- Успішну інтеграцію апаратних компонентів, таких як трансивер MAX485 для RS-485, перетворювач CH340 для USB-зв'язку та оптопари для гальванічної розв'язки.
- Розробку програмного забезпечення, яке забезпечує надійний обмін даними, налаштування адрес та збереження інформації у форматі CSV.

- Створення інтуїтивно зрозумілого інтерфейсу користувача на базі PyQt5, який дозволяє легко керувати процесом обміну даними.
- Підвищення точності та надійності даних, що сприяє покращенню якості наукових досліджень.

Розроблений пристрій демонструє високу ефективність та надійність, що підтверджується проведеними тестуваннями. Він має значний потенціал для застосування не лише в плазмових технологіях, а й у інших галузях науки та промисловості, де необхідна надійна передача даних між різними системами.

Рекомендації для подальших досліджень включають розширення функціональності пристрою, покращення інтерфейсу користувача, оптимізацію апаратної частини та інтеграцію з системами штучного інтелекту. Ці напрямки можуть сприяти подальшому розвитку системи та її адаптації до нових вимог і завдань.

Таким чином, розробка пристрою на базі мікроконтролера STM32F103C6T6 є важливим кроком до покращення методів збору та обробки даних у наукових дослідженнях. Він відкриває нові можливості для експериментаторів та інженерів, дозволяючи їм працювати з більш точними та надійними даними, що, в свою чергу, сприяє прискоренню наукового прогресу.

11. Додатки

11.1. Додаток 1 - зображення плати та інтерфейсу

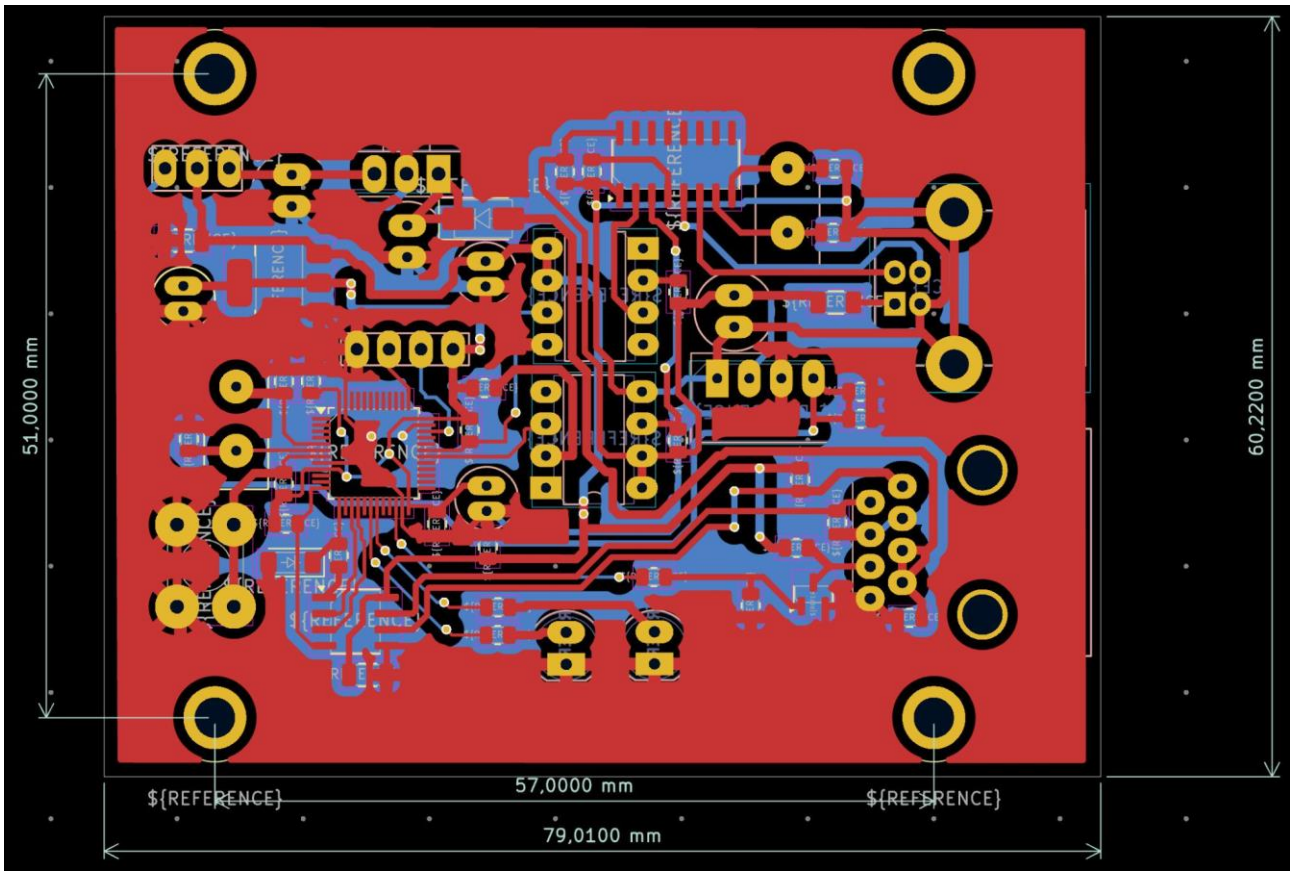

рис. 1. На зображенні показано розташування основних компонентів, таких як мікроконтролер та інтерфейси зв'язку

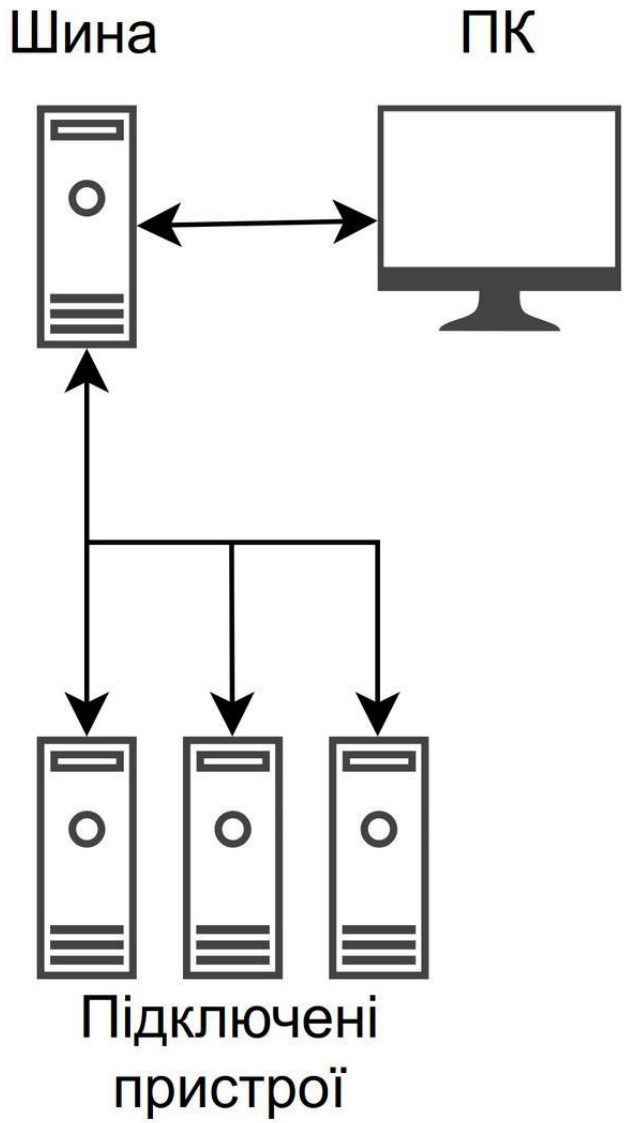

рис. 2. На зображенні показано, як пристрої підключені до мережі RS-485 та розробленого під час цієї роботи пристрою, який виконує роль шлюза

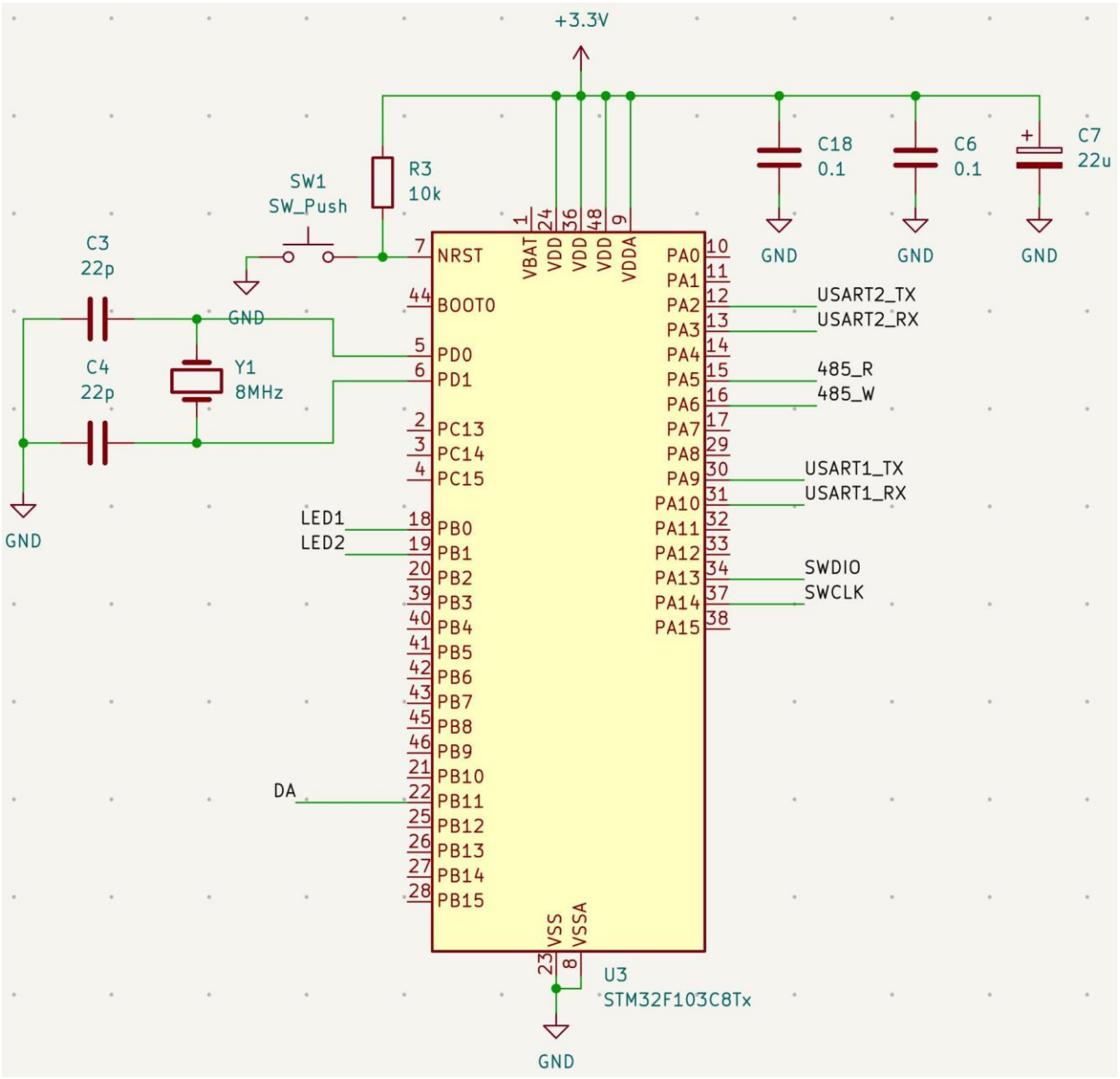

рис. 3. На зображенні показано розташування пінів та їх підключення

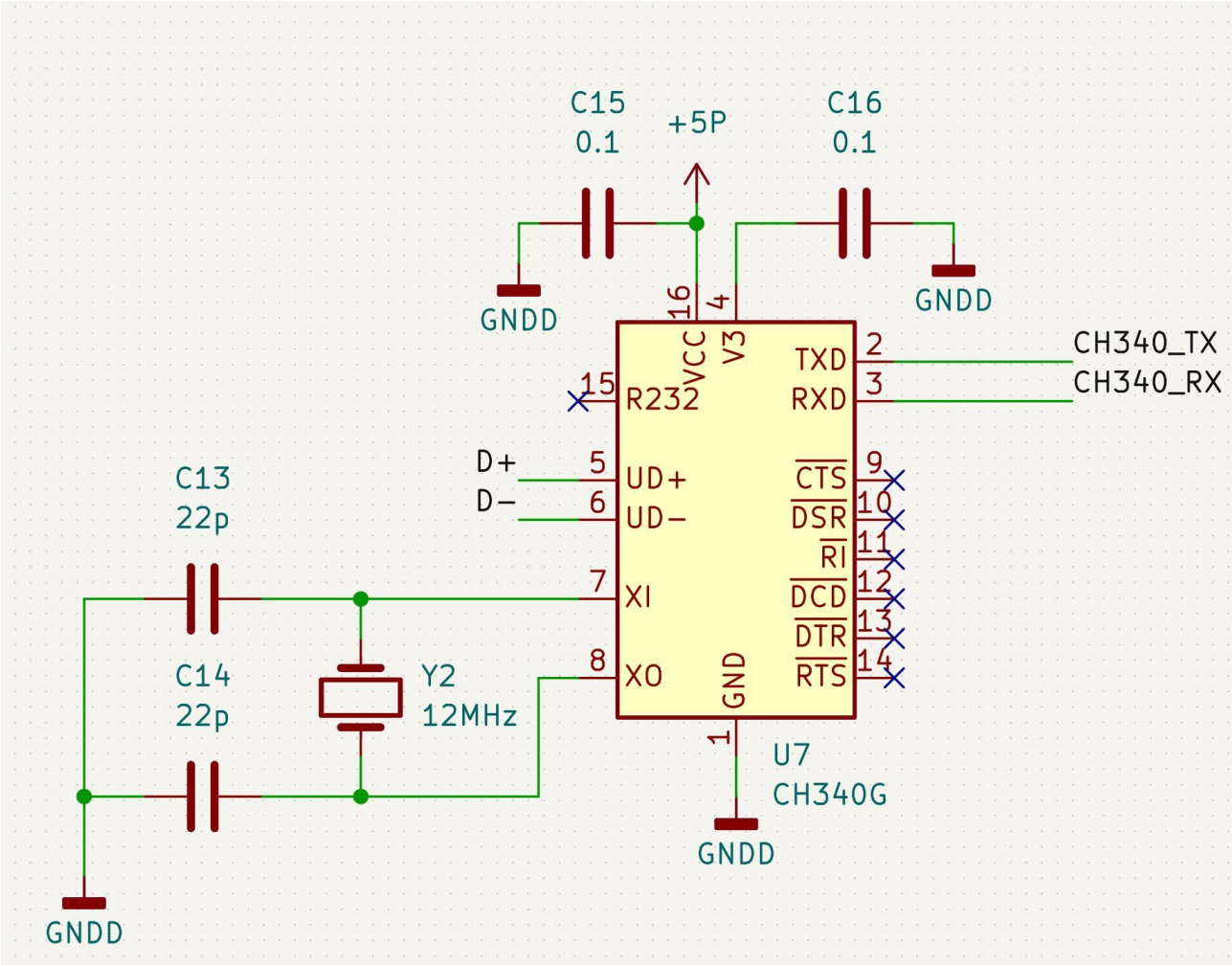

рис. 4. На зображенні показано, як пристрій підключається до мікроконтролера для забезпечення зв'язку через USB

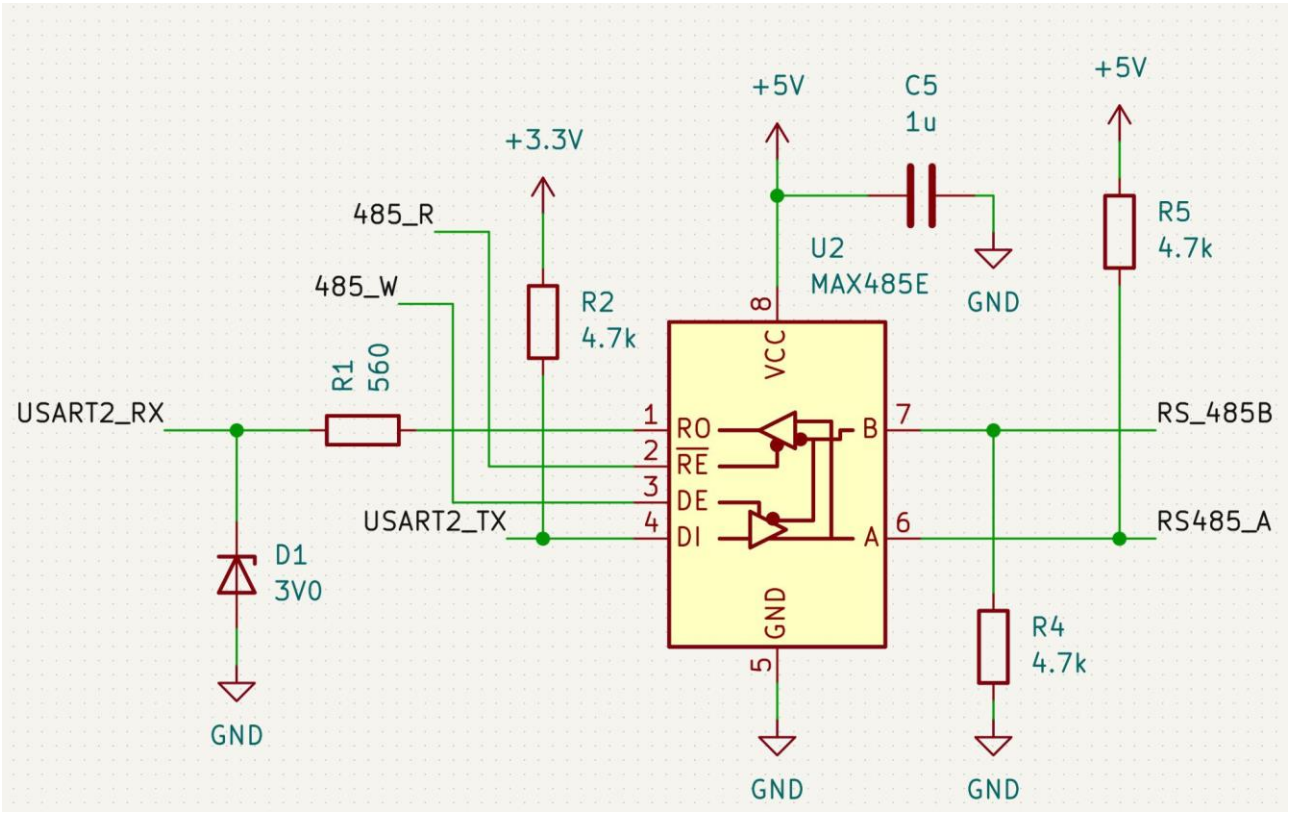

рис. 5. На зображенні показано, як пристрій підключається до мікроконтролера для забезпечення зв'язку через RS-485

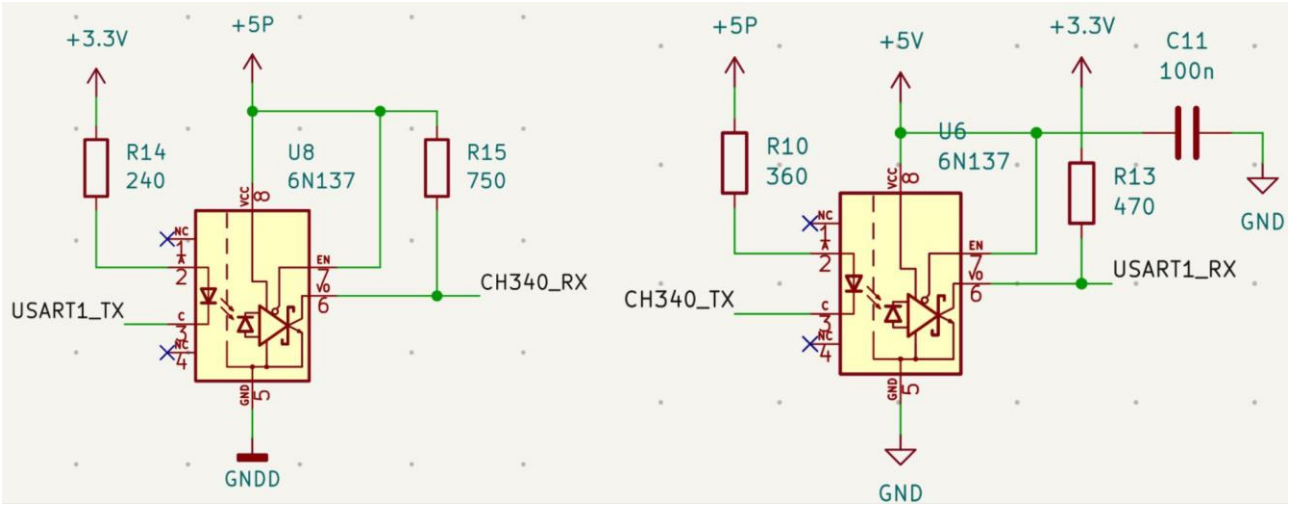

рис. 6. На зображенні показано, як пристрій використовується для гальванічної розв'язки між компонентами системи

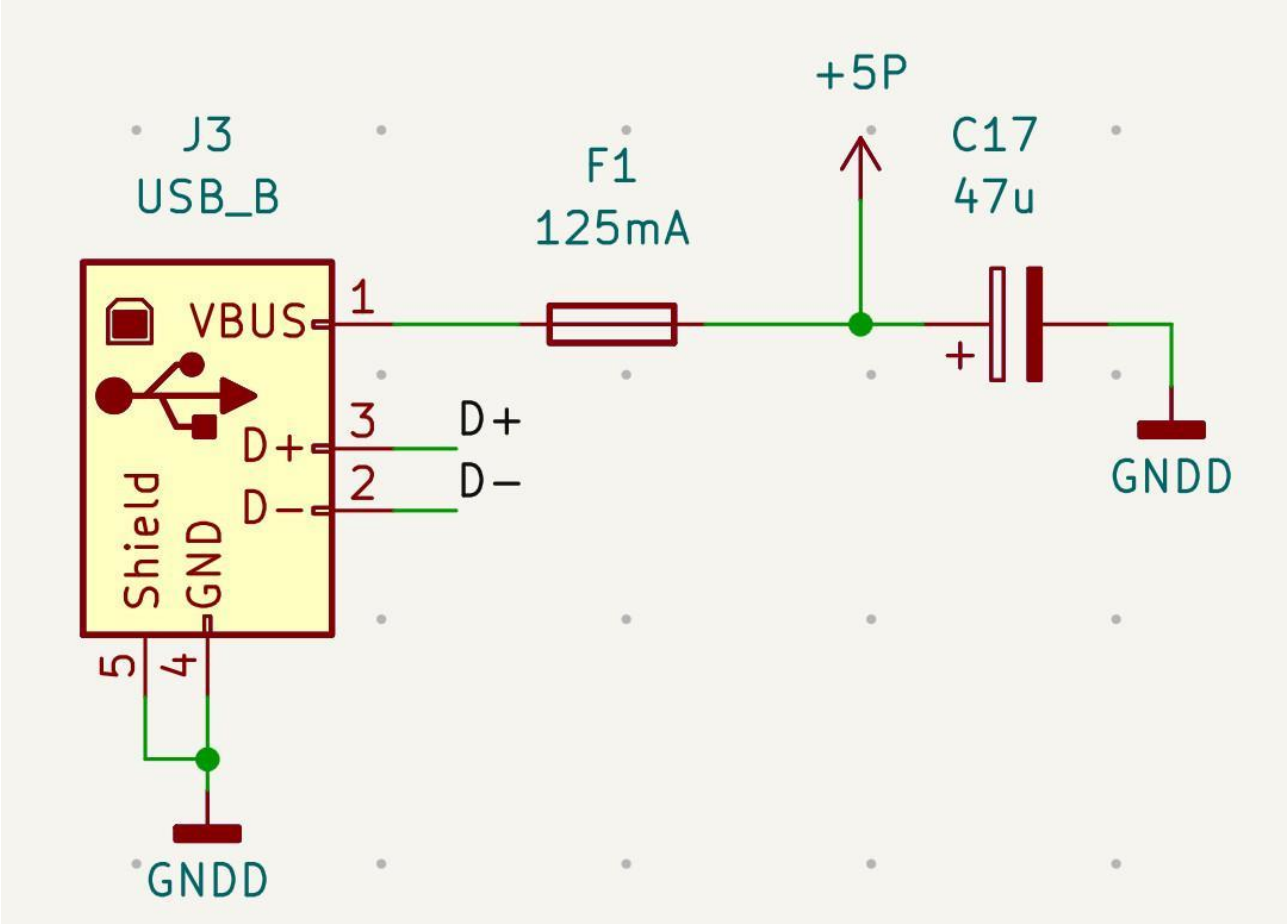

рис. 7. На зображенні показано, як роз'єм використовується для підключення пристрою до комп'ютера

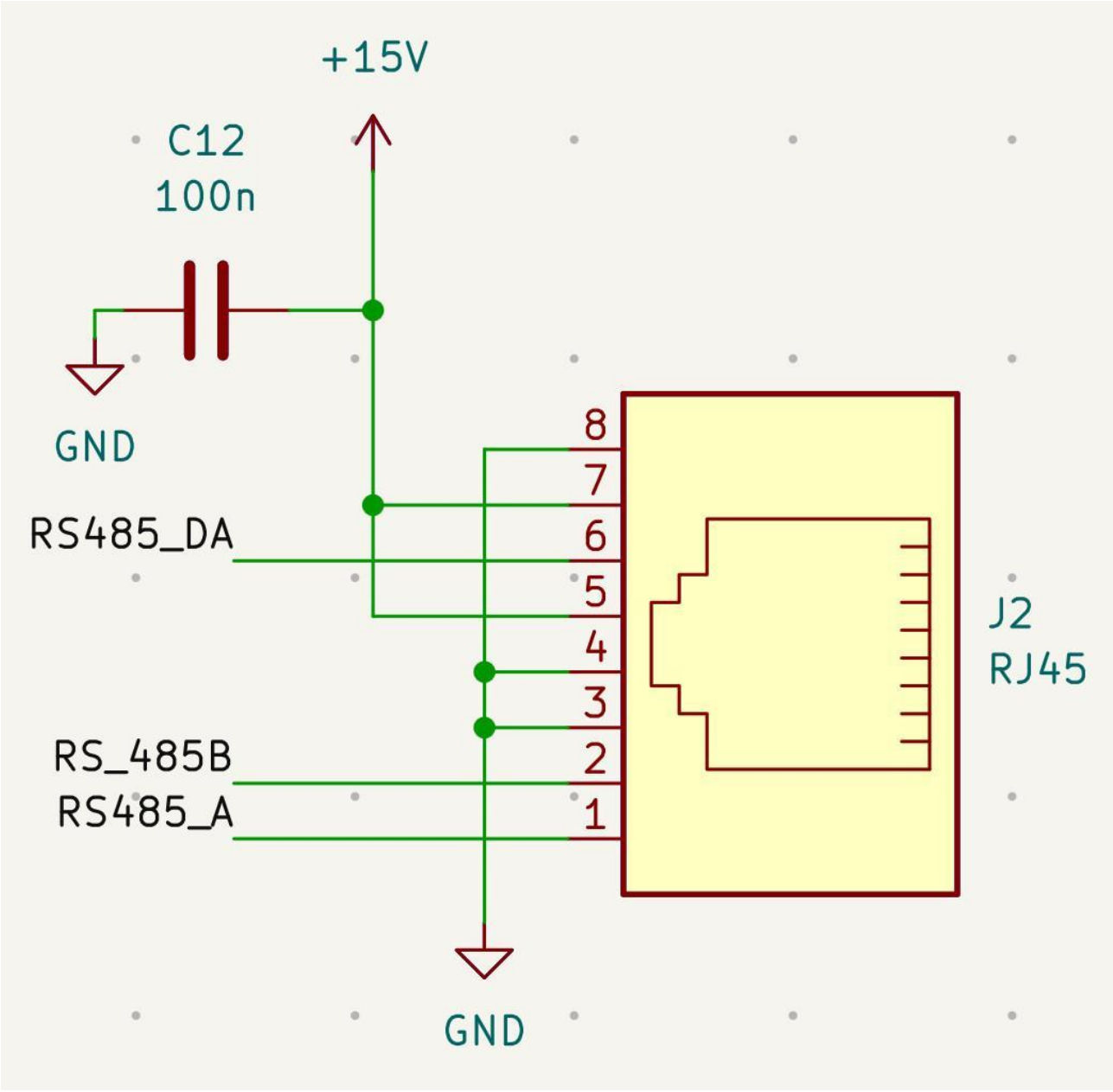

рис. 8. На зображенні показано, як роз'єм використовується для підключення пристрою до мережі Ethernet

11.2. Додаток 2 - опис інтерфейсу користувача програми для ПК

Інтерфейс користувача програми для ПК, розроблений для керування пристроєм на базі мікроконтролера STM32F103C6T6, призначений для забезпечення зручної та ефективної взаємодії користувача з системою передачі даних. Інтерфейс реалізований за допомогою бібліотеки PyQt5, яка дозволяє створювати багатофункціональні графічні застосунки. Ось основні компоненти та функції інтерфейсу:

Головне вікно програми

- Вибір COM-порту:
  - Елемент: випадаючий список (QComboBox).

- Функція: дозволяє користувачеві вибрати доступний COM-порт для підключення до пристрою. Список автоматично заповнюється доступними портами при запуску програми.

- Кнопки підключення та відключення:
  - Елемент: кнопка (QPushButton).
  - Функція: кнопка "Connect" встановлює з'єднання з вибраним COM-портом, а кнопка "Disconnect" розриває з'єднання. Стан з'єднання відображається в системних повідомленнях.
- Поле для вводу адреси:
  - Елемент: текстове поле (QLineEdit).
  - Функція: дозволяє користувачеві ввести адресу стороннього пристрою, до якого буде відправлено повідомлення.
- Поле для вводу повідомлення:
  - Елемент: текстове поле (QLineEdit).
  - Функція: користувач вводить текст повідомлення, яке буде відправлено на вказану адресу.
- Кнопка відправки повідомлення:
  - Елемент: кнопка (QPushButton).
  - Функція: ініціює відправку введеного повідомлення на адресу, вказану в полі для вводу адреси.
- Кнопка налаштування адрес:
  - Елемент: кнопка (QPushButton).
  - Функція: відкриває діалогове вікно для конфігурування адрес сторонніх пристроїв, де можна додавати, редагувати та видаляти адреси.
- Вивід повідомлень:
  - Елемент: таблиця (QTableWidget).
  - Функція: відображає історію обміну повідомленнями, включаючи часові мітки, напрямок (відправка/прийом), адресу, ім'я пристрою та зміст повідомлення. Нові повідомлення додаються вгору списку.
- Текстове поле для системних повідомлень:
  - Елемент: текстове поле (QTextEdit).
  - Функція: відображає системні повідомлення про стан з'єднання, помилки та інші важливі події.
- Кнопка для збереження у CSV:
  - Елемент: кнопка (QPushButton).
  - Функція: дозволяє користувачеві зберегти історію повідомлень у файл формату CSV для подальшого аналізу.

Діалогове вікно налаштування адрес

- Функція: дозволяє користувачеві керувати списком адрес сторонніх пристроїв.
- Елементи:
  - Таблиця адрес: відображає список адрес та їх імен.

- Кнопка додавання адреси: додає новий рядок для введення нової адреси та імені.
- Кнопка видалення адреси: видаляє вибрану адресу зі списку.
- Кнопка збереження: зберігає зміни, внесені в список адрес.

Робота з конфігурацією

- Завантаження та збереження конфігурації: при запуску програми завантажується конфігурація з файлу config.json, якщо такий існує. При закритті програми поточна конфігурація зберігається в цей файл, включаючи вибраний COM-порт, адреси та останнє введене повідомлення.

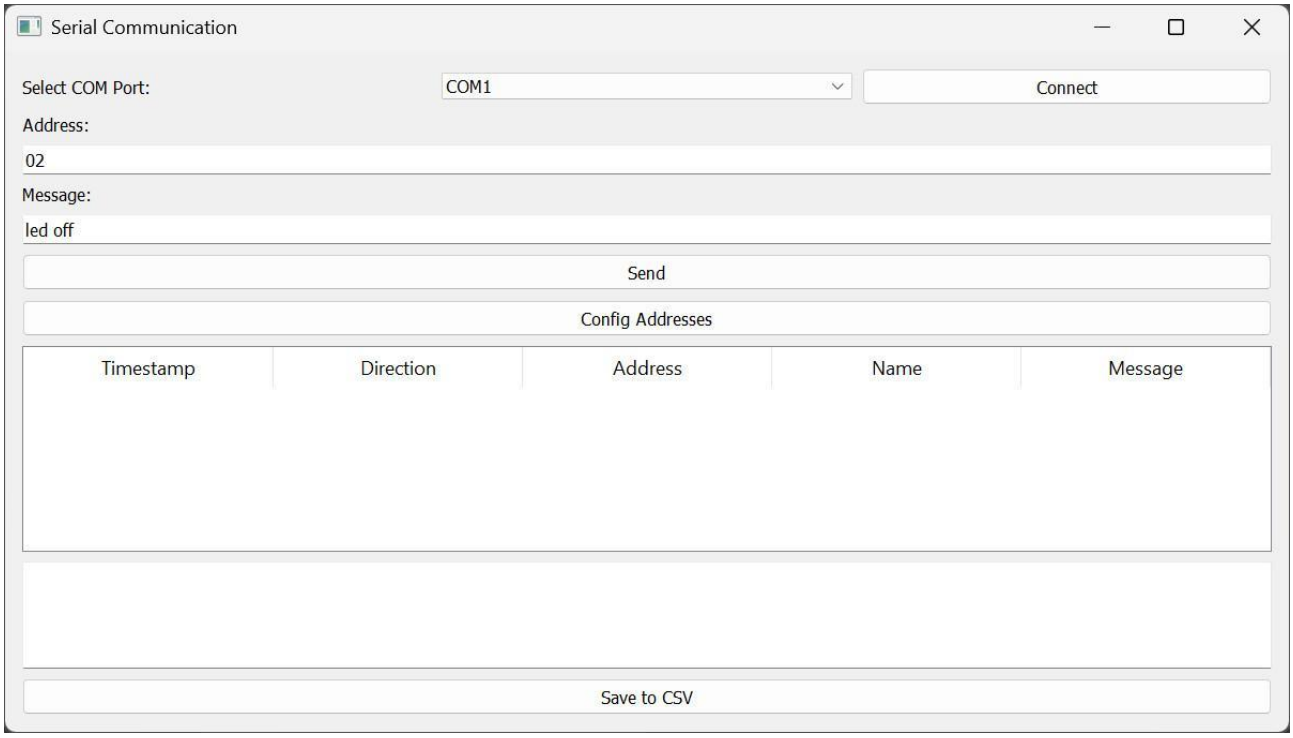

рис. 9. На зображенні показано основні елементи інтерфейсу, такі як головне вікно, вікно відображення даних та вікно налаштувань

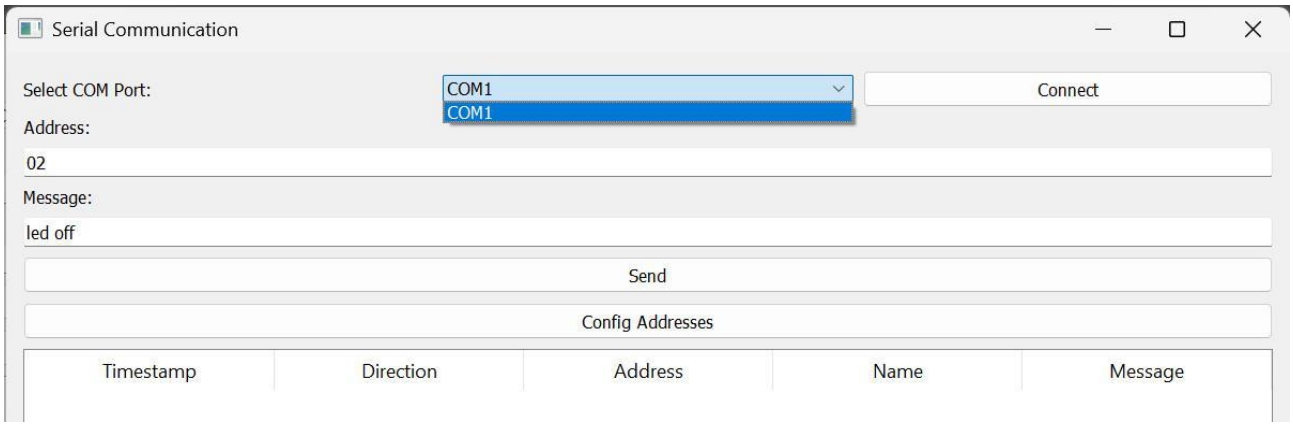

рис. 10. На зображенні показано, як користувач може вибрати COM порт для зв'язку з пристроєм

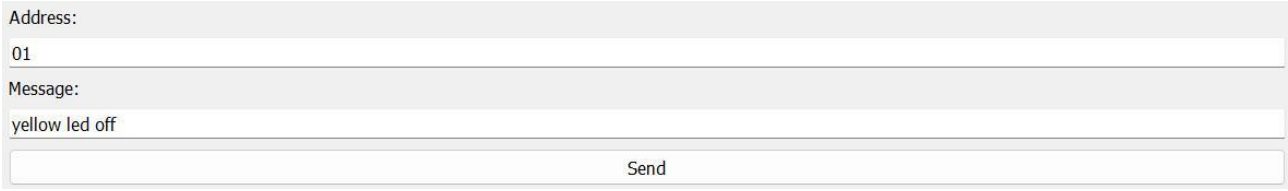

рис. 11. На зображенні показано, як користувач може вибрати отримувача, написати повідомлення та відправити його

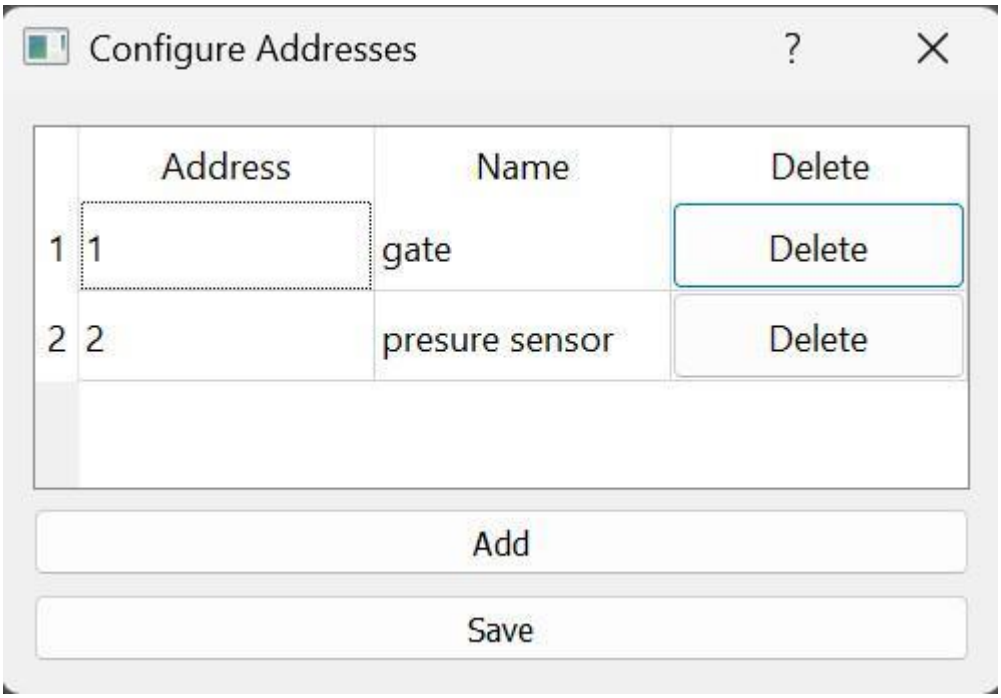

рис. 12. На зображенні показано, як користувач може призначити імена для різних пристроїв, підключених до системи

	A	B	C	D	E	F
1	Timestamp	Direction	Address	Name	Message	
2	2025-05-05 08:24:20	Sent	1	gate	yellow led off	
3	2025-05-05 08:24:17	Sent	1	gate	yellow led on	
4	2025-05-05 08:24:13	Sent	1	gate	blue led on	
5	2025-05-05 08:24:10	Sent	1	gate	blue led off	
6	2025-05-05 08:24:04	Sent	1	gate	led on	
7	2025-05-05 08:24:01	Sent	1	gate	led off	
8						
9						
10						
11						

рис. 13. На зображенні показано, експортовані дані у форматі CSV

Timestamp	Direction	Address	Name	Message
2025-05-05 08:24:20	Sent	01	gate	yellow led off
2025-05-05 08:24:17	Sent	01	gate	yellow led on
2025-05-05 08:24:13	Sent	01	gate	blue led on
2025-05-05 08:24:10	Sent	01	gate	blue led off
2025-05-05 08:24:04	Sent	01	gate	led on
2025-05-05 08:24:01	Sent	01	gate	led off

рис. 14. На зображенні показано, як користувач може отримувати інформацію про стан системи в реальному часі

рис. 15. На зображенні показано, як користувач може отримувати інформацію про системні події та помилки

11.3. Додаток 3 - опис прошивки мікроконтролера

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
```

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <ctype.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart1_rx;
DMA_HandleTypeDef hdma_usart1_tx;
DMA_HandleTypeDef hdma_usart2_rx;
DMA_HandleTypeDef hdma_usart2_tx;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
```

```
void processMessage(const char* message);  // Прототип функції обробки повідомлення
void sendConfirmation(uint8_t addr, const char* message);  // Прототип функції відправки підтвердження
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t msg[8] = "hello\r\n";  // Повідомлення для тесту

uint8_t address = 0x01;  // Адреса пристрою

uint8_t RxData1[30];  // Буфер для прийому даних через UART1
uint8_t TxData1[30];  // Буфер для передачі даних через UART1
int indx1 = 0;  // Індекс для відстеження кількості прийнятих байтів через UART1
uint8_t RxData2[30];  // Буфер для прийому даних через UART2
uint8_t TxData2[30];  // Буфер для передачі даних через UART2
int indx2 = 0;  // Індекс для відстеження кількості прийнятих байтів через UART2

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size) {
	// Обробник події прийому даних через UART
	if (huart->Instance == USART1) {
	indx1 = Size;  // Збереження кількості прийнятих байтів
	HAL_UARTEx_ReceiveToIdle_DMA(&huart1, RxData1, 30);  // Налаштування прийому даних через DMA
	}
	if (huart->Instance == USART2) {
	HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);  // Інвертування стану піна GPIOC13
	indx2 = Size;  // Збереження кількості прийнятих байтів
	HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxData2, 30);  // Налаштування прийому даних через DMA
	}
}

void processMessage(const char* message) {
	// Обробка отриманого повідомлення
	if (strcmp(message, "blue led on") == 0) {
	HAL_GPIO_WritePin(GPIOB, LED_BLUE_Pin, GPIO_PIN_SET);  // Увімкнення синього світлодіода
	sendConfirmation(address, "blue led on done");  // Відправка підтвердження
	} else if (strcmp(message, "blue led off") == 0) {
```

```
        HAL_GPIO_WritePin(GPIOB, LED_BLUE_Pin, GPIO_PIN_RESET);  // Вимкнення синього світлодіода
        sendConfirmation(address, "blue led off done");  // Відправка підтвердження
        } else if (strcmp(message, "yellow led on") == 0) {
        HAL_GPIO_WritePin(GPIOB, LED_YELLOW_Pin, GPIO_PIN_SET);  // Увімкнення жовтого світлодіода
        sendConfirmation(address, "yellow led on done");  // Відправка підтвердження
        } else if (strcmp(message, "yellow led off") == 0) {
        HAL_GPIO_WritePin(GPIOB, LED_YELLOW_Pin, GPIO_PIN_RESET);  // Вимкнення жовтого світлодіода
        sendConfirmation(address, "yellow led off done");  // Відправка підтвердження
        } else {
        sendConfirmation(address, "unknown command warning");  // Відправка попередження про невідому команду
        }
}

void sendConfirmation(uint8_t addr, const char* message) {
        // Відправка підтвердження через UART1
        HAL_UART_Transmit_IT(&huart1, RxData1, 30);  // Відправка даних через переривання
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
        // Обробник події завершення передачі даних через UART
        HAL_GPIO_WritePin(RS485_R_GPIO_Port, RS485_W_Pin, GPIO_PIN_RESET);  // Скидання стану піна RS485_W
        HAL_GPIO_WritePin(RS485_R_GPIO_Port, RS485_R_Pin, GPIO_PIN_RESET);  // Скидання стану піна RS485_R
}
/* USER CODE END 0 */

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

  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

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

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
```

```
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
      HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxData2, 30);  //
Налаштування прийому даних через DMA для UART2
      HAL_UARTEx_ReceiveToIdle_DMA(&huart1, RxData1, 30);  //
Налаштування прийому даних через DMA для UART1

      // Тестування світлодіодів
      HAL_GPIO_WritePin(GPIOB, LED_BLUE_Pin, GPIO_PIN_SET);
      HAL_Delay(200);
      HAL_GPIO_WritePin(GPIOB, LED_YELLOW_Pin, GPIO_PIN_SET);
      HAL_Delay(200);
      HAL_GPIO_WritePin(GPIOB, LED_YELLOW_Pin, GPIO_PIN_RESET);
      HAL_Delay(200);
      HAL_GPIO_WritePin(GPIOB, LED_BLUE_Pin, GPIO_PIN_RESET);
      HAL_Delay(200);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
      while (1) {
      HAL_Delay(500);  // Затримка на 500 мс
      if (indx1 == 30) {
      uint8_t receivedAddress = RxData1[0];  // Отримання адреси з прийнятих
даних
      char message[29];  // Буфер для повідомлення
      int messageLength = 0;  // Довжина повідомлення

      // Обробка прийнятих даних
      for (int i = 1; i < indx1; i++) {
            if (RxData1[i] != 0) {
```

```
                message[messageLength++] = RxData1[i];
                }
        }
        message[messageLength] = '\0';  // Завершення рядка

        // Видалення пробілів з кінця рядка
        while (messageLength > 0 && isspace((unsigned char)message[messageLength
- 1])) {
                message[messageLength - 1] = '\0';
                messageLength--;
        }

        if (receivedAddress == address) {
                processMessage(message);  // Обробка повідомлення
        } else {
                HAL_GPIO_WritePin(RS485_W_GPIO_Port, RS485_W_Pin,
GPIO_PIN_SET);  // Встановлення стану піна RS485_W
                HAL_GPIO_WritePin(RS485_R_GPIO_Port, RS485_R_Pin,
GPIO_PIN_SET);  // Встановлення стану піна RS485_R
                HAL_UART_Transmit_DMA(&huart2, RxData1, 30);  // Відправка
даних через DMA для UART2
                HAL_UART_Transmit_IT(&huart1, RxData1, 30);  // Відправка даних
через переривання для UART1
        }

        indx1 = 0;  // Скидання індексу
        }
        if (indx2 == 30) {
        HAL_UART_Transmit_DMA(&huart1, RxData2, 30);  // Відправка даних
через DMA для UART1
        indx2 = 0;  // Скидання індексу
        }
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        }
  /* USER CODE END 3 */
}

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

```
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
      Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
HAL_OK)
  {
      Error_Handler();
  }
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */
  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */
```

```
  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */
  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */
  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */
  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */
  /* USER CODE END USART2_Init 2 */
```

```
}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
  /* DMA1_Channel5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);
  /* DMA1_Channel6_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel6_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel6_IRQn);
  /* DMA1_Channel7_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel7_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel7_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* USER CODE BEGIN MX_GPIO_Init_1 */
  /* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, RS485_W_Pin|RS485_R_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : RS485_W_Pin RS485_R_Pin */
  GPIO_InitStruct.Pin = RS485_W_Pin|RS485_R_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */
  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
        /* User can add his own implementation to report the HAL error return state */
        __disable_irq();  // Вимкнення всіх переривань
        while (1) {
        // Нескінченний цикл при виникненні помилки
        }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *       where the assert_param error has occurred.
  * @param  file: pointer to the source file name
```

```
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
      /* User can add his own implementation to report the file name and line number,
      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

11.4. Додаток 4 - опис коду програмного забезпечення

```
import sys
from typing import Dict

import serial
import time
import csv
import json
from datetime import datetime
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout,
QPushButton,
                             QComboBox, QLineEdit, QHBoxLayout, QLabel, QFileDialog,
                             QTableWidget, QTableWidgetItem, QHeaderView, QDialog,
QTextEdit)
from PyQt5.QtSerialPort import QSerialPortInfo
from PyQt5.QtCore import QThread, pyqtSignal

class SerialReaderThread(QThread):
    """Потік для читання даних з серійного порту."""

    # Сигнал, який випускається при отриманні повідомлення
    message_received = pyqtSignal(str, str, int, str)

    def __init__(self, serial_connection):
    """
    Ініціалізація потоку для читання даних з серійного порту.

    Аргументи:
    serial_connection -- об'єкт серійного з'єднання.
    """
    super().__init__()  # Виклик конструктора батьківського класу
    self.ser = serial_connection  # Збереження об'єкта серійного з'єднання
    self.running = True  # Прапорець для контролю виконання потоку
```

```
    def run(self):
    """
    Основний метод потоку, який читає дані з серійного порту.
    """
    while self.running:  # Поки прапорець running встановлений в True
    if self.ser and self.ser.is_open:  # Перевірка, чи серійний порт відкритий
            address_byte = self.ser.read(1)  # Читання одного байту (адреси)
            if address_byte:  # Якщо байт прочитано
            address = int.from_bytes(address_byte, byteorder='big')  # Конвертація
байту в ціле число
            message_bytes = bytearray()  # Створення масиву байтів для
повідомлення
            while True:  # Безкінечний цикл для читання повідомлення
            byte = self.ser.read(1)  # Читання одного байту
            if byte:  # Якщо байт прочитано
                message_bytes.extend(byte)  # Додавання байту до масиву
            else:  # Якщо байт не прочитано
                time.sleep(0.02)  # Затримка на 20 мс
                if not self.ser.in_waiting:  # Якщо немає даних для читання
                break  # Вихід з циклу
            message = message_bytes.decode('utf-8', errors='ignore')  #
Декодування байтів в рядок
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  #
Отримання поточної дати та часу
            self.message_received.emit(timestamp, "Received", address,
message.strip())  # Випуск сигналу з даними повідомлення

    def stop(self):
    """
    Зупинка потоку.
    """
    self.running = False  # Встановлення прапорця running в False
    self.wait()  # Очікування завершення потоку

class AddressConfigDialog(QDialog):
    """Діалогове вікно для налаштування адрес."""

    def __init__(self, addresses, parent=None):
    """
    Ініціалізація діалогового вікна для налаштування адрес.

    Аргументи:
    addresses -- словник адрес та їх імен.
    parent -- батьківський віджет (за замовчуванням None).
    """
```

```
        super().__init__(parent)  # Виклик конструктора батьківського класу
        self.setWindowTitle('Configure Addresses')  # Встановлення заголовку вікна
        self.setGeometry(100, 100, 500, 300)  # Встановлення розмірів та положення вікна

        self.addresses: Dict = addresses  # Збереження словника адрес та їх імен

        self.layout = QVBoxLayout()  # Створення вертикального компонувальника
        self.table = QTableWidget(len(addresses), 3)  # Створення таблиці з кількістю рядків, рівною кількості адрес
        self.table.setHorizontalHeaderLabels(["Address", "Name", "Delete"])  # Встановлення заголовків стовпців
        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  # Налаштування розтягування стовпців

        for row, (address, name) in enumerate(addresses.items()):  # Перебір адрес та їх імен
            self.table.setItem(row, 0, QTableWidgetItem(address))  # Встановлення адреси в перший стовпець
            self.table.setItem(row, 1, QTableWidgetItem(name))  # Встановлення імені в другий стовпець
            delete_button = QPushButton('Delete')  # Створення кнопки для видалення
            delete_button.clicked.connect(self.deleteRow)  # Підключення обробника події натискання
            self.table.setCellWidget(row, 2, delete_button)  # Встановлення кнопки в третій стовпець

        self.saveButton = QPushButton('Save')  # Створення кнопки для збереження
        self.saveButton.clicked.connect(self.saveAddresses)  # Підключення обробника події натискання

        self.addButton = QPushButton('Add')  # Створення кнопки для додавання
        self.addButton.clicked.connect(self.addAddress)  # Підключення обробника події натискання

        self.layout.addWidget(self.table)  # Додавання таблиці до компонувальника
        self.layout.addWidget(self.addButton)  # Додавання кнопки додавання до компонувальника
        self.layout.addWidget(self.saveButton)  # Додавання кнопки збереження до компонувальника
        self.setLayout(self.layout)  # Встановлення компонувальника для вікна

    def saveAddresses(self):
        """
        Збереження адрес та їх імен у словник.
        """
```

```
        for row in range(self.table.rowCount()):  # Перебір рядків таблиці
        address = self.table.item(row, 0).text()  # Отримання адреси з першого стовпця
        name = self.table.item(row, 1).text()  # Отримання імені з другого стовпця
        self.addresses[address] = name  # Збереження адреси та імені в словник
        self.accept()  # Закриття діалогу з поверненням коду Accepted

    def addAddress(self):
        """
        Додавання нового рядка для введення нової адреси та імені.
        """
        row_count = self.table.rowCount()  # Отримання кількості рядків в таблиці
        self.table.insertRow(row_count)  # Додавання нового рядка в кінець таблиці
        delete_button = QPushButton('Delete')  # Створення кнопки для видалення
        delete_button.clicked.connect(self.deleteRow)  # Підключення обробника події натискання
        self.table.setCellWidget(row_count, 2, delete_button)  # Встановлення кнопки в третій стовпець нового рядка

    def deleteRow(self):
        """
        Видалення рядка з таблиці та видалення адреси зі словника.
        """
        button = self.sender()  # Отримання об'єкта, який викликав подію (кнопка)
        if button:  # Якщо кнопка існує
            row = self.table.indexAt(button.pos()).row()  # Отримання індексу рядка кнопки
            address = self.table.item(row, 0).text()  # Отримання адреси з першого стовпця рядка
            self.addresses.pop(address)  # Видалення адреси зі словника
            self.table.removeRow(row)  # Видалення рядка з таблиці

class SerialApp(QWidget):
    """Головний віджет додатку для серійного зв'язку."""

    def __init__(self):
        """
        Ініціалізація головного віджету додатку.
        """
        super().__init__()  # Виклик конструктора батьківського класу

        self.initUI()  # Ініціалізація інтерфейсу користувача
        self.initSerial()  # Ініціалізація серійного з'єднання
        self.messages = []  # Список для збереження повідомлень
        self.addresses = {}  # Словник для збереження адрес та їх імен
        self.loadConfig()  # Завантаження конфігурації з файлу
```

```
    def initUI(self):
    """
    Налаштування інтерфейсу користувача.
    """
    self.setWindowTitle('Serial Communication')  # Встановлення заголовку вікна
    self.setMinimumWidth(1200)  # Встановлення мінімальної ширини вікна

    # Вибір COM-порту
    self.portLabel = QLabel('Select COM Port:', self)  # Створення мітки для вибору COM-порту
    self.portComboBox = QComboBox(self)  # Створення випадаючого списку для вибору COM-порту
    self.populateSerialPorts()  # Заповнення випадаючого списку доступними COM-портами

    # Кнопки для підключення та відключення
    self.connectButton = QPushButton('Connect', self)  # Створення кнопки для підключення
    self.connectButton.clicked.connect(self.connectSerial)  # Підключення обробника події натискання

    # Поле для вводу адреси
    self.addressLabel = QLabel('Address:', self)  # Створення мітки для вводу адреси
    self.addressInput = QLineEdit(self)  # Створення поля для вводу адреси

    # Поле для вводу повідомлення
    self.messageLabel = QLabel('Message:', self)  # Створення мітки для вводу повідомлення
    self.messageInput = QLineEdit(self)  # Створення поля для вводу повідомлення

    # Кнопка для відправки повідомлення
    self.sendButton = QPushButton('Send', self)  # Створення кнопки для відправки повідомлення
    self.sendButton.clicked.connect(self.sendMessage)  # Підключення обробника події натискання

    # Кнопка для налаштування адрес
    self.configButton = QPushButton('Config Addresses', self)  # Створення кнопки для налаштування адрес
    self.configButton.clicked.connect(self.configAddresses)  # Підключення обробника події натискання

    # Вивід повідомлень
```

```
        self.outputTable = QTableWidget(self)  # Створення таблиці для виводу повідомлень
        self.outputTable.verticalHeader().setVisible(False)  # Приховування вертикального заголовку
        self.outputTable.setColumnCount(5)  # Встановлення кількості стовпців
        self.outputTable.setHorizontalHeaderLabels(["Timestamp", "Direction", "Address", "Name", "Message"])  # Встановлення заголовків стовпців

        self.outputTable.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  # Налаштування розтягування стовпців

        # Текстове поле для системних повідомлень
        self.systemMessages = QTextEdit(self)  # Створення текстового поля для системних повідомлень
        self.systemMessages.setReadOnly(True)  # Встановлення режиму тільки для читання
        self.systemMessages.setFixedHeight(100)  # Встановлення фіксованої висоти

        # Кнопка для збереження у CSV
        self.saveButton = QPushButton('Save to CSV', self)  # Створення кнопки для збереження у CSV
        self.saveButton.clicked.connect(self.saveToCSV)  # Підключення обробника події натискання

        # Розташування віджетів
        hbox = QHBoxLayout()  # Створення горизонтального компонувальника
        hbox.addWidget(self.portLabel)  # Додавання мітки вибору COM-порту до компонувальника
        hbox.addWidget(self.portComboBox)  # Додавання випадаючого списку до компонувальника
        hbox.addWidget(self.connectButton)  # Додавання кнопки підключення до компонувальника

        vbox = QVBoxLayout()  # Створення вертикального компонувальника
        vbox.addLayout(hbox)  # Додавання горизонтального компонувальника до вертикального
        vbox.addWidget(self.addressLabel)  # Додавання мітки вводу адреси до компонувальника
        vbox.addWidget(self.addressInput)  # Додавання поля вводу адреси до компонувальника
        vbox.addWidget(self.messageLabel)  # Додавання мітки вводу повідомлення до компонувальника
        vbox.addWidget(self.messageInput)  # Додавання поля вводу повідомлення до компонувальника
        vbox.addWidget(self.sendButton)  # Додавання кнопки відправки до компонувальника
```

```
        vbox.addWidget(self.configButton)  # Додавання кнопки налаштування адрес до компонувальника
        vbox.addWidget(self.outputTable)  # Додавання таблиці виводу до компонувальника
        vbox.addWidget(self.systemMessages)  # Додавання текстового поля системних повідомлень до компонувальника
        vbox.addWidget(self.saveButton)  # Додавання кнопки збереження у CSV до компонувальника

        self.setLayout(vbox)  # Встановлення компонувальника для вікна

        def initSerial(self):
        """
        Ініціалізація з'єднання з COM-портом.
        """
        self.ser = None  # Ініціалізація об'єкта серійного з'єднання
        self.reader_thread = None  # Ініціалізація потоку для читання даних

        def populateSerialPorts(self):
        """
        Заповнення випадаючого списку доступними COM-портами.
        """
        self.portComboBox.clear()  # Очищення випадаючого списку
        for port in QSerialPortInfo.availablePorts():  # Перебір доступних COM-портів
        self.portComboBox.addItem(port.portName())  # Додавання назви порту до випадаючого списку

        def connectSerial(self):
        """
        Підключення до вибраного COM-порту.
        """
        if self.ser and self.ser.is_open:  # Якщо серійний порт вже відкритий
        self.ser.close()  # Закриття серійного порту

        port = self.portComboBox.currentText()  # Отримання вибраного COM-порту
        try:
        self.ser = serial.Serial(port, 9600, timeout=1)  # Відкриття серійного порту
        self.systemMessages.append(f"Connected to {port}")  # Вивід повідомлення про успішне підключення

        # Запускаємо потік для читання даних
        self.reader_thread = SerialReaderThread(self.ser)  # Створення потоку для читання даних
        self.reader_thread.message_received.connect(self.displayMessage)  # Підключення обробника сигналу
```

```
        self.reader_thread.start()  # Запуск потоку

        except serial.SerialException as e:  # Обробка винятків підключення
        self.systemMessages.append(f"Failed to connect to {port}: {e}")  # Вивід повідомлення про помилку

    def configAddresses(self):
        """
        Відкриття діалогу для налаштування адрес.
        """
        dialog = AddressConfigDialog(self.addresses, self)  # Створення діалогового вікна
        if dialog.exec_() == QDialog.Accepted:  # Якщо діалог закрито з кодом Accepted
        self.saveConfig()  # Збереження конфігурації
        self.loadConfig()  # Завантаження конфігурації

    def displayMessage(self, timestamp, direction, address, message):
        """
        Вивід повідомлення в інтерфейсі та збереження в списку.

        Аргументи:
        timestamp -- часова мітка повідомлення.
        direction -- напрямок повідомлення ("Received" або "Sent").
        address -- адреса повідомлення.
        message -- текст повідомлення.
        """
        name = self.addresses.get(str(address), "")  # Отримання імені за адресою
        self.outputTable.insertRow(0)  # Додавання нового рядка на початок таблиці
        self.outputTable.setItem(0, 0, QTableWidgetItem(timestamp))  # Встановлення часової мітки
        self.outputTable.setItem(0, 1, QTableWidgetItem(direction))  # Встановлення напрямку
        self.outputTable.setItem(0, 2, QTableWidgetItem(f"{address:02X}"))  # Встановлення адреси
        self.outputTable.setItem(0, 3, QTableWidgetItem(name))  # Встановлення імені
        self.outputTable.setItem(0, 4, QTableWidgetItem(message))  # Встановлення повідомлення
        self.messages.insert(0, (timestamp, direction, address, name, message))  # Додавання повідомлення до списку

    def sendMessage(self):
        """
        Відправка повідомлення через COM-порт.
        """
```

```
        if not self.ser or not self.ser.is_open:  # Якщо серійний порт не відкритий
        self.systemMessages.append("No serial connection.")  # Вивід повідомлення про помилку
        return

        address = int(self.addressInput.text(), 16)  # Конвертація адреси з рядка в число
        message = self.messageInput.text()  # Отримання тексту повідомлення

        # Перевірка довжини повідомлення
        if len(message) > 29:
        message = message[:29]  # Обрізання повідомлення до 29 символів
        elif len(message) < 29:
        message = message.ljust(29)  # Доповнення повідомлення пробілами до 29 символів

        self.ser.write(bytes([address]))  # Відправка адреси через серійний порт
        self.ser.write(message.encode('utf-8'))  # Відправка повідомлення через серійний порт

        # Збереження відправленого повідомлення
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  # Отримання поточної дати та часу
        name = self.addresses.get(str(address), "")  # Отримання імені за адресою
        self.displayMessage(timestamp, "Sent", address, message)  # Вивід повідомлення в інтерфейсі

        def saveToCSV(self):
        """
        Збереження виводу у CSV файл.
        """
        options = QFileDialog.Options()  # Опції діалогового вікна
        fileName, _ = QFileDialog.getSaveFileName(self, "Save to CSV", "", "CSV Files (*.csv);;All Files (*)", options=options)  # Відкриття діалогу збереження файлу
        if fileName:  # Якщо файл вибрано
        with open(fileName, 'w', newline='') as file:  # Відкриття файлу для запису
                writer = csv.writer(file)  # Створення об'єкта для запису в CSV
                writer.writerow(["Timestamp", "Direction", "Address", "Name", "Message"])  # Запис заголовків
                for message in self.messages:  # Перебір повідомлень
                writer.writerow(message)  # Запис повідомлення в файл

        def loadConfig(self):
        """
        Завантаження налаштувань з файлу config.json.
```

```
        """
        try:
        with open('config.json', 'r') as file:  # Відкриття файлу конфігурації для читання
                config = json.load(file)  # Завантаження конфігурації з файлу
                self.addresses = config.get('addresses', {})  # Отримання адрес з конфігурації
                last_port = config.get('last_port')  # Отримання останнього вибраного порту
                last_address = config.get('last_address')  # Отримання останньої вибраної адреси
                last_message = config.get('last_message')  # Отримання останнього введеного повідомлення

                if self.addresses:  # Якщо адреси завантажені
                for row in range(self.outputTable.rowCount()):  # Перебір рядків таблиці
                address_item = self.outputTable.item(row, 2)  # Отримання елемента адреси
                if address_item:  # Якщо елемент існує
                        address_text = address_item.text()  # Отримання тексту адреси
                        name = self.addresses.get(str(int(address_text)))  # Отримання імені за адресою

                        if name:  # Якщо ім'я існує
                        name_item = self.outputTable.item(row, 3)  # Отримання елемента імені
                        if name_item:  # Якщо елемент існує
                                name_item.setText(name)  # Встановлення імені
                        else:  # Якщо елемент не існує
                                self.outputTable.setItem(row, 3, QTableWidgetItem(name))  # Створення нового елемента імені

                if last_port:  # Якщо останній порт існує
                index = self.portComboBox.findText(last_port)  # Пошук індексу порту в випадаючому списку
                if index >= 0:  # Якщо порт знайдено
                self.portComboBox.setCurrentIndex(index)  # Встановлення вибраного порту

                if last_address:  # Якщо остання адреса існує
                self.addressInput.setText(f"{last_address:02X}")  # Встановлення вибраної адреси

                if last_message:  # Якщо останнє повідомлення існує
```

```
                self.messageInput.setText(last_message)  # Встановлення введеного повідомлення

        except FileNotFoundError:  # Обробка винятку, якщо файл не знайдено
        pass

        def saveConfig(self):
        """
        Збереження налаштувань у файл config.json.
        """
        config = {
        'addresses': self.addresses,  # Адреси та їх імена
        'last_port': self.portComboBox.currentText(),  # Останній вибраний порт
        'last_address': int(self.addressInput.text(), 16),  # Остання вибрана адреса
        'last_message': self.messageInput.text()  # Останнє введене повідомлення
        }
        with open('config.json', 'w') as file:  # Відкриття файлу конфігурації для запису
        json.dump(config, file, indent=4)  # Збереження конфігурації в файл

        def closeEvent(self, event):
        """
        Збереження налаштувань при закритті програми.

        Аргументи:
        event -- об'єкт події закриття вікна.
        """
        self.saveConfig()  # Збереження конфігурації
        if self.reader_thread:  # Якщо потік читання існує
        self.reader_thread.stop()  # Зупинка потоку
        self.reader_thread.wait()  # Очікування завершення потоку
        event.accept()  # Прийняття події закриття

def main():
        """
        Головна функція для запуску додатку.
        """
        app = QApplication(sys.argv)  # Створення об'єкта додатку
        ex = SerialApp()  # Створення головного віджету додатку
        ex.show()  # Відображення головного вікна
        sys.exit(app.exec_())  # Запуск головного циклу додатку

if __name__ == '__main__':
        main()  # Виклик головної функції при запуску скрипту
```

12. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) A. Murmantsev, A. Veklich, V. Boretskij, M. Kleshych, S. Fesenko, M. Bartlova (2022). Peculiarities of interaction of Cu-W composite materials with thermal arc discharge plasma. *Problems of Atomic Science and Technology*. https://doi.org/10.46813/2022-142-134
2) Mahreen, G. Veda Prakash, Satyananda Kar, Debaprasad Sahu, A. Ganguli (2022). Influence of pulse modulation frequency on helium RF atmospheric pressure plasma jet characteristics. *Contributions to Plasma Physics*. http://dx.doi.org/10.1002/ctpp.202200007
3) І.О.Анісімов (2018). "Фізика плазми"
4) Mark A Naivar, Mark E Wilder (2014). Development of small and inexpensive digital data acquisition systems using a microcontroller-based approach. *Cytomeri*. https://pmc.ncbi.nlm.nih.gov/articles/PMC3969846/
5) Peng Chen (2024). Data Acquisition System based on Serial Port. *Journal of Theory and Practice of Engineering Science*. https://doi.org/10.53469/jtpes.2024.04(03).09
6) Liang Zhao, Shaocheng Qu, Weigang Zhang (2020). Design of multi-channel data collector for highway tunnel lighting based on STM32 and Modbus protocol. *Optik - International Journal for Light and Electron Optics*. https://doi.org/10.1016/j.ijleo.2020.164388
7) STMicroelectronics. (n.d.). *STM32F103C4 datasheet*. Retrieved from https://www.st.com/resource/en/datasheet/stm32f103c4.pdf
8) STMicroelectronics. (n.d.). *RM0008: STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs reference manual*. Retrieved from https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
9) STMicroelectronics. (n.d.). *STM32 Cortex-M4 MCUs and MPUs programming manual*. Retrieved from https://www.st.com/resource/en/programming_manual/pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf
10) Texas Instruments. (n.d.). *The RS-485 Design Guide*. Retrieved from https://www.ti.com/lit/an/slla272d/slla272d.pdf
11) Texas Instruments. (n.d.). *Interface Circuits for TIA/EIA-485 (RS-485)*. Retrieved from https://www.ti.com/lit/an/slla036d/slla036d.pdf