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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Система блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32

Виконав: студент *IV* курсу, групи *CIc-42*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

(підпис) Косар Д.Б. (прізвище та ініціали)

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

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

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

Рецензент (підпис) Петрик М.Р. (прізвище та ініціали)

Тернопіль
2024

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

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
« ____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

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

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

студенту Косару Дмитру Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Система блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32*

Керівник роботи *Лещишин Юрій Зіновійович, к.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*26*» *04* *2024* року № *4/7-468*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Блок схема алгоритму роботи системи
2. Електрична принципова схема
3. Електрична структурна схема обміну телекомунікаційними даними
4. Електрична структурна схема

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Пилипець М.І., д.т.н., проф. каф. МТ		

7. Дата видачі завдання 26.04.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Розробка і затвердження технічного завдання*	*01.02 – 09.02*	
2	*Аналіз технічного завдання*	*05.02 – 11.02*	
3	*Аналіз вимог та принципів організації системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*26.04 – 03.05*	
4	*Проєктування системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*04.05 – 13.05*	
5	*Розробка схем і програмного забезпечення системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*14.05 – 25.05*	
6	*Розробка інструкцій з використання системи*	*26.05 – 09.06*	
7	*Безпека життєдіяльності, основи охорони праці*	*10.06 – 15.06*	
8	*Оформлення кваліфікаційної роботи*	*16.06 – 20.06*	
9	*Попередній захист кваліфікаційної роботи*	*14.06*	
10	*Захист кваліфікаційної роботи*	*24.06 – 28.06*	

Студент _____ Косар Дмитро Богданович

(підпис) (прізвище, ім’я, по батькові)

Керівник роботи _____ Лещишин Юрій Зіновійович

(підпис) (прізвище, ім’я, по батькові)

АНОТАЦІЯ

Система блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32 // Кваліфікаційна робота бакалавра // Косар Дмитро Богданович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних систем та мереж, група СІс-42 // Тернопіль, 2024 // с. – 65, рис. – 24, табл. – 0, додат. – 3, бібліогр. – 25.

Ключові слова: система блокування передачі телекомунікаційних даних, мікроконтролер, stm32, програмне забезпечення.

Кваліфікаційну роботу бакалавра присвячено розробці системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32. На основі результатів огляду та аналізу аналогів розроблено структурну схему системи блокування передачі телекомунікаційних даних. Здійснено обґрунтування вибору протоколів та інтерфейсів для роботи системи та описано процес розробки. Розроблено алгоритм роботи апаратної частини системи та здійснено опис програмних функцій мікроконтролера. Розроблено програмне забезпечення для тестування системи та подальшої роботи з нею. Розглянуто основні питання безпеки життєдіяльності та основ охорони праці, стосовно проєктованої системи та її використання.

ANNOTATION

Telecommunication data transmission blocking system for the electric power infrastructure manager based on STM 32 // Bachelor's thesis // Kosar Dmytro Bohdanovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Computer Systems and Networks, group CIs-42 // Ternopil, 2024 // p. – 65, fig. – 24, table – 0, bibliography – 25.

Key words: telecommunication data transmission blocking system, microcontroller, stm32, software.

The bachelor's thesis is devoted to the development of a telecommunication data transmission blocking system for the power infrastructure dispatcher based on STM 32. Based on the results of the review and analysis of analogues, a block diagram of the telecommunication data transmission blocking system was developed. The choice of protocols and interfaces for the system is justified and the development process is described. The algorithm of the system hardware operation is developed and the software functions of the microcontroller are described. Software for testing the system and further work with it is developed. The main issues of life safety and the basics of occupational safety and health related to the designed system and its use are considered.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

EEPROM – Electrically Erasable Programmable Read-Only Memory

GPIO – General-Purpose Input/Output

I2C – Inter-Integrated Circuit

I2S – Integrated Inter-chip Sound

IDE – Integrated Development Environment

MVC – Model-View-Controller

ROM – Read-Only Memory

SPI – Serial Peripheral Interface

SRAM – Static Random Access Memory

UART – Universal Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

АЦП – Аналого-Цифровий Перетворювач

СБПТД – система блокування передачі телекомунікаційних даних

ЦАП – Цифро-Аналоговий Перетворювач

ШІМ – Широтно-Імпульсна Модуляція

ВСТУП

У сучасному світі електроенергетика відіграє вирішальну роль у забезпеченні життєдіяльності та розвитку всіх галузей економіки. Безперервне постачання електроенергії є основою стабільного функціонування промислових підприємств, транспортної інфраструктури, медичних установ, а також забезпечення комфортних умов проживання для населення. З огляду на це, важливою задачею є забезпечення надійної роботи електроенергетичних систем.

Одним з ключових аспектів ефективного управління електроенергетичною інфраструктурою є моніторинг та контроль роботи всіх її елементів у режимі реального часу. Це завдання потребує високого рівня автоматизації процесів та використання сучасних технологій телекомунікацій для передачі даних. Надійна та безперебійна передача даних між різними компонентами системи є запорукою стабільної роботи енергосистеми та оперативного реагування на можливі аварійні ситуації.

Сучасні диспетчерські системи повинні не лише збирати та аналізувати дані з численних датчиків і пристроїв, але й забезпечувати безпечну передачу цієї інформації. Особливої уваги потребує фільтрація даних, що надходять, для виключення некоректних або потенційно небезпечних команд, які можуть призвести до збоїв у роботі енергосистеми. Таким чином, питання забезпечення безпеки передачі даних є одним з найважливіших у сфері електроенергетики.

В останні роки все більше уваги приділяється використанню мікроконтролерів у системах автоматизації та управління. Мікроконтролери забезпечують високу продуктивність, гнучкість у налаштуваннях і можуть виконувати складні обчислювальні задачі в реальному часі. Серед них особливо виділяються мікроконтролери STM32, які поєднують у собі потужний функціонал і широкий спектр застосувань, включаючи можливість реалізації різних інтерфейсів для передачі даних.

У контексті управління електроенергетичною інфраструктурою важливою задачею є розробка систем, що забезпечують надійну передачу даних та

можливість їх фільтрації. Використання мікроконтролера STM32 дозволяє створити гнучку та ефективну систему, яка відповідає сучасним вимогам до безпеки та надійності. Особливо важливим є забезпечення можливості диспетчера оперативно контролювати передачу даних, відсіювати непотрібні або небезпечні пакети інформації, що надходять, і реагувати на зміну ситуації в реальному часі.

Таким чином, розробка системи блокування та передачі телекомунікаційних даних на основі мікроконтролера STM32 є актуальною задачею, яка сприяє підвищенню ефективності та безпеки управління електроенергетичною інфраструктурою. Використання сучасних технологій і підходів у цій галузі дозволяє створювати більш надійні та функціональні системи, що відповідають вимогам сучасного етапу розвитку електроенергетики.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз вимог до комп'ютерної системи блокування передачі телекомунікаційних даних

1.1.1 Загальні вимоги до СБПТД

Для забезпечення надійної роботи системи необхідно ретельно проаналізувати вимоги до комп'ютерної системи. Це включає загальні вимоги, вимоги до продуктивності та надійності, безпеки, інтерфейсів і програмного забезпечення.

Загальні вимоги до комп'ютерної системи включають в себе набір основних характеристик, яким повинна відповідати система для забезпечення її ефективної роботи, а саме:

- надійність;
- стабільність;
- масштабованість;
- енергоефективність;
- сумісність.

Система повинна працювати безперебійно, з мінімальним часом простою. Це критично важливо для диспетчерських систем в енергетиці, де збої можуть призвести до серйозних наслідків. ЇЇ робота повинна бути стабільною навіть при високих навантаженнях, включаючи пік завантаження під час аварійних ситуацій. Вона повинна легко масштабуватися для підтримки збільшення кількості підключених пристроїв та обробки більших обсягів даних.

В умовах постійної роботи важливо, щоб система була енергоефективною, що дозволить знизити витрати на її експлуатацію.

Система повинна бути сумісною з існуючою інфраструктурою та підтримувати інтеграцію з різними типами обладнання та програмного забезпечення.

1.1.2 Вимоги до продуктивності СБПТД

Продуктивність системи є одним з ключових факторів її успішної роботи. Основні аспекти продуктивності включають:

- швидкодія;
- мультизадачність;
- надійність роботи.

Система повинна мати здатність обробляти великі обсяги даних в реальному часі без значних затримок. Це включає обробку запитів від різних пристроїв та передачу даних між ними. Вона повинна підтримувати одночасну роботу з багатьма завданнями та потоками даних, що дозволить диспетчеру ефективно управляти різними процесами.

Система повинна мати високу надійність та відмовостійкість, що дозволить їй продовжувати роботу навіть у випадку збоїв окремих компонентів. Це включає використання резервних копій, дублювання критичних компонентів та автоматичне відновлення після збоїв.

1.1.3 Вимоги до безпеки в СБПТД

Безпека є одним з найважливіших аспектів будь-якої системи, особливо в контексті електроенергетичної інфраструктури. Основні вимоги до безпеки включають:

- захист від несанкціонованого доступу;
- шифрування даних;
- безпека програмного забезпечення.

Для забезпечення конфіденційності переданої інформації повинні використовуватися сучасні алгоритми шифрування. Це стосується як даних, що передаються між пристроями, так і даних, що зберігаються в системі

Програмне забезпечення повинно бути розроблене з урахуванням кращих практик безпеки, включаючи регулярне оновлення та виправлення вразливостей.

1.1.4 Вимоги до інтерфейсів в СБПТД

Для забезпечення ефективної роботи системи необхідно забезпечити підтримку різних інтерфейсів. Основні вимоги включають:

- підтримка протоколів передачі даних;
- гнучкість налаштувань;
- інтеграція з існуючими системами.

Система повинна підтримувати кілька протоколів передачі даних, таких як Modbus, USB, що дозволить їй взаємодіяти з різними типами обладнання.

Інтерфейси повинні бути гнучкими, що дозволить налаштовувати параметри передачі даних відповідно до вимог конкретних застосувань та обладнання.

Система повинна легко інтегруватися з існуючими диспетчерськими системами та іншими компонентами електроенергетичної інфраструктури. Це включає підтримку стандартних протоколів та інтерфейсів, а також можливість налаштування під конкретні вимоги.

1.1.5 Вимоги до програмного забезпечення для СБПТД

Програмне забезпечення є ключовим компонентом системи, яке забезпечує її функціональність та ефективність. Основні вимоги до програмного забезпечення включають:

- надійність та ефективність;
- масштабованість;
- гнучкість та адаптивність;
- моніторинг та діагностика;
- оновлюваність.

Програмне забезпечення повинно бути розроблене таким чином, щоб забезпечувати високу надійність та ефективність роботи системи. Це включає мінімізацію затримок, оптимізацію використання ресурсів та забезпечення безперебійної роботи.

ПЗ повинно підтримувати масштабованість, що дозволить додавати нові функції та підтримувати збільшення кількості підключених пристроїв без суттєвих змін у коді. Воно

Система повинна бути гнучкою, що дозволить легко адаптувати її до змінних вимог та умов експлуатації. Це включає можливість налаштування параметрів та збереження їх у енергонезалежній пам’яті, додавання нових функцій та інтеграції з іншими системами.

Програмне забезпечення повинно мати інструменти для моніторингу та діагностики роботи системи, що дозволить швидко виявляти та усувати можливі проблеми. Це включає збір та аналіз даних про роботу системи, виявлення аномалій та генерацію звітів.

ПЗ повинно мати можливість регулярного оновлення для виправлення помилок, покращення функціональності та забезпечення захисту від нових загроз. Це включає можливість віддаленого оновлення та забезпечення зворотної сумісності.

Надалі ці вимоги будуть враховані при виборі апаратного та програмного забезпечення, що дозволить створити систему, яка відповідає всім необхідним характеристикам та вимогам.

1.2 Аналіз можливих рішень поставленого завдання для системи блокування передачі телекомунікаційних даних

1.2.1 Вибір апаратної платформи СБПТД

Вибір апаратної платформи є одним із критично важливих етапів при розробці системи. Апаратна платформа повинна відповідати вимогам щодо

продуктивності, надійності, сумісності з іншими компонентами системи та забезпечувати можливість масштабування в майбутньому.

Основні критерії вибору апаратної платформи включають:

- процесорну потужність;
- наявність інтерфейсів;
- енергоспоживання;
- надійність та стійкість до збоїв;
- підтримка розширень.

Вибрана платформа повинна мати достатню обчислювальну потужність для обробки великих обсягів даних у реальному часі.

Платформа повинна підтримувати різноманітні інтерфейси, такі як USB, UART, SPI, I2C, Ethernet тощо, для забезпечення гнучкості у підключенні різних пристроїв.

Враховуючи постійний режим роботи, апаратна платформа повинна бути енергоефективною. Апаратна платформа повинна забезпечувати високу надійність роботи, мати засоби захисту від електромагнітних завад, перегріву та інших впливів.

Платформа повинна мати можливість підключення додаткових модулів або пристроїв для забезпечення гнучкості та розширюваності системи.

1.2.2 Огляд можливих мікроконтролерів для СБПТД

Існує багато варіантів апаратних платформ, які можуть бути використані для реалізації системи блокування та передачі даних [1]. Огляд можливих рішень включає:

- ESP32 – мікроконтролер ESP32 від Espressif Systems відзначається високою продуктивністю, вбудованим Wi-Fi і Bluetooth модулями, які використовуються для бездротової передачі даних. ESP32 також підтримує численні інтерфейси, що дозволяє його використовувати в різних застосуваннях;
- Atmel SAM – мікроконтролери Atmel SAM базуються на архітектурі ARM Cortex-M і пропонують широкий спектр можливостей для побудови

складних систем управління. Вони мають високу продуктивність, багатофункціональні периферійні модулі;

– TI MSP430 – мікроконтролери MSP430 від Texas Instruments є енергоефективними рішеннями для вбудованих систем з низьким енергоспоживанням. Вони забезпечують достатню продуктивність для багатьох задач і мають розвинуту підтримку периферійних пристроїв;

– STM32 – мікроконтролери STM32 від STMicroelectronics відзначаються широким спектром моделей з різними характеристиками, високою продуктивністю та енергоефективністю. Вони підтримують численні периферійні інтерфейси, що дозволяє використовувати їх у різноманітних застосуваннях від простих датчиків до складних промислових систем.

Після проведеного аналізу можливих рішень для реалізації системи, можна зробити висновок, що мікроконтролери серії STM32 від компанії STMicroelectronics є оптимальним вибором.

1.2.3 Огляд мікроконтролерів STM32

Мікроконтролери серії STM32 від компанії STMicroelectronics є одним з найпопулярніших виборів для реалізації складних систем управління та передачі даних. Вони відзначаються високою продуктивністю, багатофункціональністю і наявністю численних периферійних пристроїв.

Основні характеристики мікроконтролерів STM32:

- архітектура ARM Cortex;
- широкий діапазон продуктивності;
- розвинуті периферійні модулі;
- інтеграція засобів захисту;
- широка підтримка розвитку.

Мікроконтролери STM32 базуються на архітектурі ARM Cortex (M0, M3, M4, M7), що забезпечує високу продуктивність та енергоефективність. Моделі STM32 варіюються від низькопотужних контролерів для простих завдань до високопродуктивних контролерів для складних обчислювальних задач. Вони

мають вбудовані модулі для роботи з різноманітними інтерфейсами (UART, SPI, I2C, USB, CAN, Ethernet) та аналоговими сигналами (ADC, DAC). STM32 забезпечують високий рівень безпеки за рахунок вбудованих засобів захисту даних, криптографічних модулів та механізмів захисту пам'яті.

STM32 підтримується розвинутим екосистемою засобів розробки, включаючи STM32CubeMX, що дозволяє швидко налаштувати та згенерувати початковий код проекту.

1.2.4 Вибір протоколів передачі даних для СБПТД

Для забезпечення ефективної роботи системи необхідно вибрати відповідні протоколи передачі даних, які забезпечать надійність, швидкість і безпеку комунікацій між різними компонентами системи. Основні протоколи, що можуть бути використані:

– Modbus є одним з найбільш поширених протоколів для передачі даних в промислових системах [8]. Він забезпечує простоту і надійність обміну даними між пристроями. Протокол підтримує різні варіанти реалізації, включаючи Modbus RTU, Modbus ASCII та Modbus TCP;

– CAN (Контролерна мережа) є високонадійним протоколом для обміну даними в реальному часі, який часто використовується в автомобільній промисловості та автоматизації. CAN забезпечує швидкий і надійний обмін даними між багатьма пристроями;

– Ethernet є стандартом для локальних мереж і забезпечує високу швидкість передачі даних та можливість підключення великої кількості пристроїв. Ethernet підтримує численні вищерівневі протоколи, такі як TCP/IP, що дозволяє інтегрувати систему в загальну мережу підприємства;

– UART (Universal Asynchronous Receiver/Transmitter) є простим та надійним способом для обміну даними між мікроконтролерами та периферійними пристроями [16]. UART часто використовується для комунікацій на короткі відстані;

– SPI та I2C – інтерфейси, які використовуються для високошвидкісного обміну даними між мікроконтролером та периферійними пристроями на короткі відстані. SPI забезпечує високу швидкість передачі даних, тоді як I2C дозволяє підключати багато пристроїв на одну шину з мінімальною кількістю проводів.

Після проведеного аналізу можливих протоколів передачі даних для реалізації системи можна зробити висновок, що протокол Modbus у поєданні з UART є оптимальним вибором.

1.2.5 Планування етапів розробки структури СБПТД

Розробка структури системи включає визначення основних компонентів, їх взаємодії та розподілу функцій між ними. Це дозволить створити ефективну та надійну систему, яка буде відповідати всім вимогам та забезпечить високу продуктивність і безпеку роботи. Основні етапи розробки структури системи включають:

- визначення основних компонентів;
- взаємодія між компонентами;
- моделювання та тестування;
- інтеграція з існуючою інфраструктурою.

Необхідно визначити функції кожного з компонентів і розподілити їх таким чином, щоб забезпечити оптимальну продуктивність і надійність роботи системи. Система повинна складатися з декількох основних компонентів, таких як модулі збору даних, модулі обробки даних, модулі зберігання даних і модулі передачі даних. Визначення способів і протоколів взаємодії між компонентами системи, що забезпечить надійну передачу даних і координацію роботи всіх модулів.

Створення моделі системи та проведення тестування для виявлення можливих проблем і оптимізації структури. Це включає моделювання потоків даних, аналіз продуктивності та перевірку надійності системи. Забезпечення сумісності системи з існуючою інфраструктурою підприємства, включаючи

можливість інтеграції з існуючими диспетчерськими системами та іншими компонентами електроенергетичної інфраструктури.

1.2.6 Програмне забезпечення та інструменти розробки СБПТД

Розробка програмного забезпечення для системи блокування та передачі даних включає використання спеціалізованих інструментів та платформ, що забезпечать швидку та ефективну розробку, тестування та розгортання системи. Основні інструменти та платформи включають:

– STM32CubeMX – інструмент для генерації початкового коду проекту на основі мікроконтролерів STM32. STM32CubeMX дозволяє налаштовувати периферійні модулі, обирати необхідні бібліотеки та генерувати початковий код для проекту;

– IDE (Integrated Development Environment) – використання інтегрованого середовища розробки (Keil, IAR Embedded Workbench, STM32CubeIDE) для написання, компіляції та налагодження програмного забезпечення. IDE забезпечує зручний інтерфейс та інструменти для розробки коду, налагодження та тестування;

– RTOS (Real-Time Operating System) – використання операційної системи реального часу (FreeRTOS, Azure RTOS) для керування задачами, забезпечення мультизадачності та оптимізації використання ресурсів мікроконтролера;

– інструменти для моделювання та тестування – використання інструментів для моделювання та тестування системи, таких як симулятори, відладчики, інструменти для аналізу продуктивності та перевірки безпеки.

Використання STM32CubeIDE у поєднанні з FreeRTOS є оптимальним вибором для розробки програмного забезпечення. Ці інструменти забезпечують потужний і зручний середовище розробки, підтримку мультизадачності, надійність та гнучкість, що дозволяє створити ефективну і надійну систему управління даними.

РОЗДІЛ 2 ПРОЕКТНА ЧАСТИНА

2.1 Вибір апаратної платформи системи блокування передачі телекомунікаційних даних

2.1.1 Обґрунтування вибору мікроконтролера STM32 для СБПТД

При виборі мікроконтролера для реалізації системи блокування та передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури враховується низка ключових факторів, таких як продуктивність, енергоспоживання, кількість і типи периферійних модулів, підтримка різних інтерфейсів передачі даних, надійність та доступність. Одним з найоптимальніших варіантів є мікроконтролери серії STM32, які поєднують у собі високу продуктивність, енергоефективність та багатий набір периферійних модулів.

STM32 має наступні серії мікроконтролерів:

– STM32F0 – серія мікроконтролерів базується на ARM Cortex-M0 і забезпечує базову функціональність з низьким енергоспоживанням [2]. Вона підходить для простих застосувань, але має недостатньо продуктивності для складних задач у телекомунікаційних системах;

– STM32F1 – мікроконтролери на базі ARM Cortex-M3 забезпечують більшу продуктивність порівняно з STM32F0. Вони добре підходять для широкого спектру застосувань, але можуть не мати достатньо ресурсів для систем з високими вимогами до обробки даних;

– STM32F3 – серія високопродуктивних 32-бітних мікроконтролерів, які поєднують ядро ARM Cortex-M4 з потужними аналоговими функціями. Вони створені для застосувань, що потребують високої продуктивності та точності обробки даних;

– STM32F4 – серія базується на ARM Cortex-M4 і пропонує високу продуктивність з вбудованим DSP-ядром, що дозволяє обробляти складні алгоритми. Вона забезпечує високий рівень функціональності та підходить для більшості промислових застосувань;

– STM32H7 – серія мікроконтролерів на базі ARM Cortex-M7, що забезпечує найвищу продуктивність серед всіх серій STM32. Вона підходить для критично важливих застосувань, що потребують максимальної обчислювальної потужності.

Для реалізації системи блокування та передачі телекомунікаційних даних було обрано мікроконтролер STM32F303RBT6 з серії STM32F3 який зображений на рис. 2.1.

Рисунок 2.1 – Мікроконтролер STM32F303RBT6

Опис основних характеристик пристрою:

– ARM Cortex-M4 з FPU (Floating Point Unit) для роботи з плаваючою точкою;

– DSP-розширення для швидкої обробки сигналів;

– частота до 72 МГц;

– 128 КБ Flash пам'яті для зберігання програмного коду;

– 32 КБ для зберігання даних;

Усі виводи STM32F303RBT6 показано на рис. 2.2.

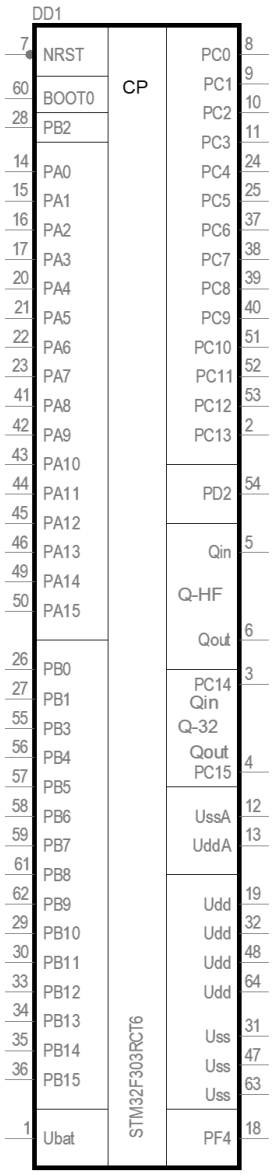

Рисунок 2.2 – Виводи STM32F303RBT6

Опис інтерфейсів та периферії пристрою:

– підтримка Full-Speed USB, що дозволяє реалізувати різні USB-пристрої, включаючи USB-to-COM порти;

– декілька UART-портів для серійної передачі даних;

– SPI та I2C – підтримка високошвидкісних комунікацій з периферійними пристроями;

– CAN для зв'язку з іншими мікроконтролерами та промисловими системами;

– вбудована підтримка Ethernet для мережевих комунікацій.

– декілька універсальних таймерів для задач реального часу;

– спеціалізовані таймери для генерації ШІМ сигналів та інших застосувань;

– вбудовані 12-бітні АЦП та ЦАП для аналогових вимірювань і сигналів;

– CRC обчислювач для перевірки цілісності даних;

– підтримка низькоенергетичних режимів для зниження енергоспоживання.

Мікроконтролер STM32F303RBT6 є оптимальним вибором для реалізації системи блокування та передачі телекомунікаційних даних завдяки своїй високій продуктивності, широкому набору периферійних модулів, достатньому обсягу пам'яті та надійності. Цей мікроконтролер забезпечує всі необхідні ресурси для ефективної реалізації вимог системи та гарантує стабільну та безперебійну роботу в умовах промислового середовища.

2.1.2 Вибір периферійних модулів та інтерфейсів для СБПТД

Мікроконтролер STM32F303RBT6 підтримує USB 2.0 інтерфейс з можливістю роботи в режимі Full-Speed (12 Мбіт/с) і High-Speed (480 Мбіт/с) за допомогою зовнішнього PHY [17]. Він має вбудований контролер USB OTG (On-The-Go), який дозволяє працювати як у ролі хоста, так і у ролі пристрою. У цьому проекті USB буде використовуватися шляхом створення віртуальних COM портів.

USB (Universal Serial Bus) є одним з найбільш поширених інтерфейсів для підключення периферійних пристроїв до комп'ютерів і вбудованих систем. Його популярність обумовлена простотою використання, високою швидкістю передачі даних, можливістю підключення та відключення пристроїв без перезавантаження системи (hot-swapping), а також підтримкою широкого спектру пристроїв.

Стандарти USB:

– USB 1.1 – швидкість передачі даних до 12 Мбіт/с (Full-Speed);

– USB 2.0 – швидкість передачі даних до 480 Мбіт/с (High-Speed);

– USB 3.0 та вище – швидкість передачі даних до 5 Гбіт/с і вище (SuperSpeed).

Типи передачі даних:

– контрольні (Control Transfers) – використовуються для налаштування та управління пристроєм;

– інтервальні (Interrupt Transfers) – використовуються для передачі невеликих обсягів даних з низькою затримкою;

– ізохронні (Isochronous Transfers) – використовуються для передачі даних у реальному часі;

– пакетні (Bulk Transfers) – використовуються для передачі великих обсягів даних.

Конектори та кабелі:

– USB-A – стандартний роз'єм для підключення до хостів;

– USB-B – використовується для підключення периферійних пристроїв, таких як принтери показано на рис. 2.3;

– Micro-USB і Mini-USB – компактні варіанти для мобільних пристроїв;

– USB-C – новий стандарт, який підтримує реверсивне підключення та швидкісну передачу даних.

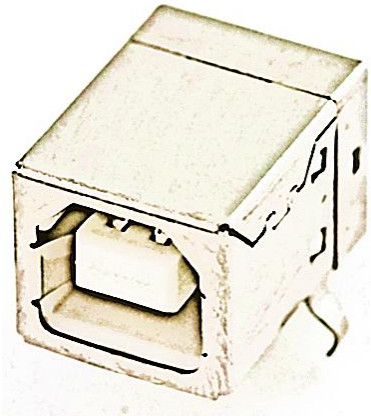

Рисунок 2.3 – USB-B

Переваги використання USB інтерфейсу:

– USB 2.0 забезпечує достатню швидкість для обробки та передачі великих обсягів даних в режимі реального часу;

– можливість підключення до будь-якого комп'ютера або іншого пристрою з підтримкою USB без потреби в спеціальних драйверах (завдяки CDC класу);

– простота підключення та налаштування, а також можливість hot-swapping роблять USB ідеальним вибором для інтеграції в існуючі системи;

– підтримка декількох віртуальних COM портів дозволяє реалізувати різні функції в рамках одного інтерфейсу, забезпечуючи зручну і ефективну комунікацію;

– завдяки вбудованим механізмам контролю цілісності даних та обробки помилок, USB забезпечує надійну передачу даних навіть в умовах промислового середовища.

USB інтерфейс є оптимальним вибором для реалізації системи блокування та передачі телекомунікаційних даних [18]. Він забезпечує високу швидкість, гнучкість та надійність, необхідні для ефективної роботи системи. Використання USB CDC для створення віртуальних COM портів дозволяє легко інтегрувати систему з існуючими засобами управління та моніторингу, забезпечуючи зручний інтерфейс для обробки та передачі даних.

Позначення USB-B на електрично принциповій схемі зображено на рис. 2.4.

Рисунок 2.4 – Позначення USB-B на електрично принциповій схемі

Для передачі даних використовуватиметься RS-485 показано на рис. 2.5, який є популярним стандартом для послідовного зв’язку, що широко використовується в промислових додатках через його надійність, здатність до передачі даних на великі відстані та стійкість до електромагнітних перешкод.

Рисунок 2.5 – RS-485

Основні характеристики RS-485:

– диференціальна передача – використовує два дроти (A і B) для передачі даних у диференціальному режимі, що підвищує стійкість до шумів;

– максимальна довжина – підтримка передачі даних на відстані до 1200 метрів;

– швидкість передачі – залежить від довжини кабелю, зазвичай до 10 Мбіт/с на коротких відстанях;

– лінійна топологія – підтримка до 32 пристроїв на одній лінії без повторювачів;

– можливість розширення – використання повторювачів для збільшення кількості пристроїв та загальної довжини мережі;

– протокол Modbus один з найпоширеніших протоколів для передачі даних у промислових мережах, що забезпечує надійну та стандартизовану комунікацію між пристроями.

STM32F303RBT6 підтримує послідовні інтерфейси, такі як UART, які можуть бути використані з зовнішніми драйверами RS-485 для реалізації даного інтерфейсу. Це дозволяє забезпечити надійну передачу даних між мікроконтролером та іншими пристроями у промислових мережах.

Позначення RS-485 на електрично принциповій схемі зображено на рис. 2.6.

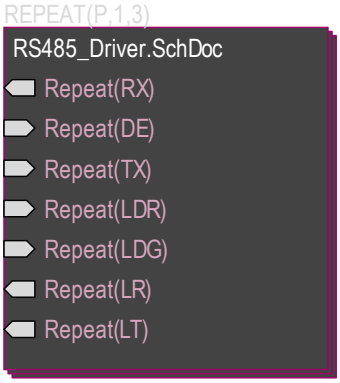

Рисунок 2.6 – Позначення RS-485 на електрично принциповій

Переваги використання RS-485 інтерфейсу:

– RS-485 дозволяє передавати дані на великі відстані (до 1200 метрів) без втрати якості сигналу, що ідеально підходить для промислових середовищ;

– диференціальна передача даних забезпечує високу стійкість до електромагнітних перешкод, що є критично важливим у промислових умовах;

– можливість підключення до 32 пристроїв на одній лінії без повторювачів та збільшення цієї кількості за допомогою повторювачів;

– використання стандартних протоколів, таких як Modbus, забезпечує надійну комунікацію між пристроями та легкість інтеграції в існуючі системи;

– RS-485 інтерфейс є відносно дешевим у впровадженні та обслуговуванні.

RS-485 інтерфейс є оптимальним вибором для реалізації передачі даних в системі у промислових умовах. Він забезпечує високу дальність передачі, стійкість до перешкод, гнучкість у підключенні великої кількості пристроїв та надійну комунікацію. Використання RS-485 у поєднанні з протоколом Modbus дозволяє створити ефективну та стабільну систему передачі даних, що відповідає вимогам промислових додатків.

2.1.3 Схематичне рішення апаратної частини СБПТД

Можна виділити основні функціональні блоки системи, що складається з мікроконтролера STM32, який забезпечує обробку та передачу даних через інтерфейси USB та RS-485. Система складається з трьох основних модулів: Modbus, фільтр та передавач (трансмітер), що взаємодіють між собою та з зовнішніми інтерфейсами. Електрично принципова схема системи зображена на рис. 2.7.

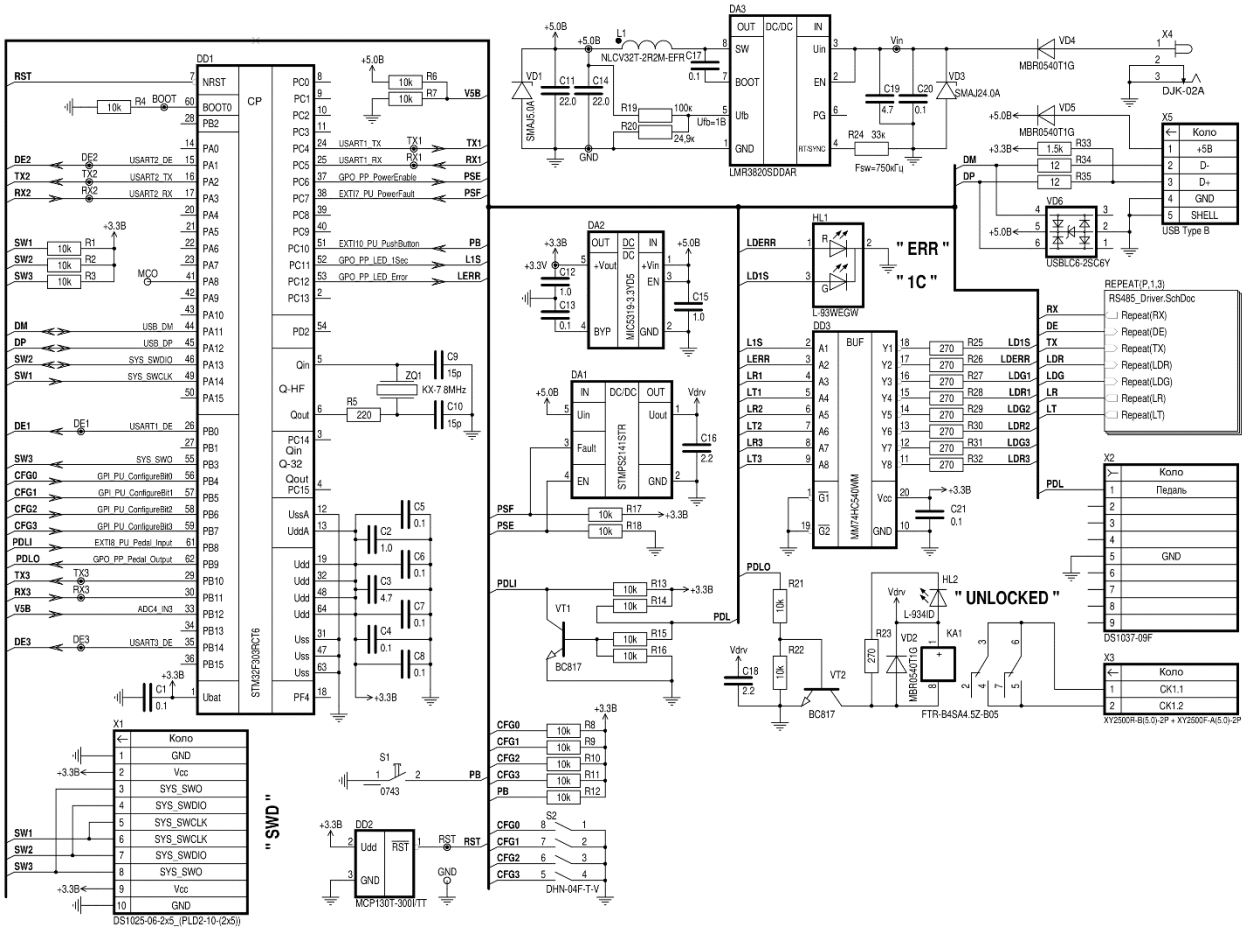

Рисунок 2.7 – Електрично принципова схема системи

Функціональна схема зображена на рис. 2.8.

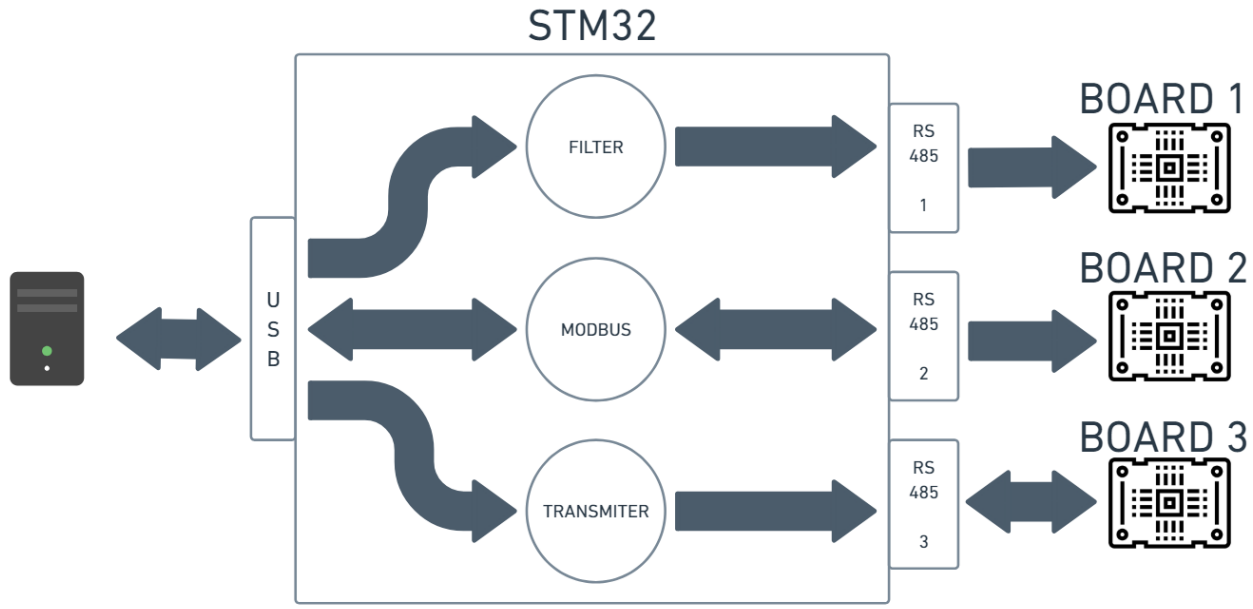

Рисунок 2.8 – Функціональна схема

Мікроконтролер STM32 виступає центральним елементом системи, виконуючи роль обробника та контролера для всіх функціональних блоків.

Інтерфейс USB використовується для підключення до комп'ютера. Він дозволяє здійснювати комунікацію з системою через віртуальні COM порти, реалізовані за допомогою USB CDC класу. Основні функції USB інтерфейсу в системі:

– Modbus сервер – забезпечує обмін даними між системою та зовнішніми пристроями через Modbus протокол;

– фільтрація даних – отримані дані проходять через фільтр, де обробляються згідно з заданими критеріями;

– передача даних – фільтровані дані передаються через інтерфейс RS-485 до інших пристроїв системи.

RS-485 використовується для передачі даних на великі відстані з високою стійкістю до електромагнітних перешкод. Він дозволяє об'єднувати до 32 пристроїв на одній лінії. Основні функції RS-485 інтерфейсу:

– отримання даних – дані від зовнішніх пристроїв надходять через RS-485 і передаються до фільтра для обробки.

– передача даних – фільтровані дані передаються до зовнішніх пристроїв через RS-485.

Схема друкованої плати зображена на рис. 2.9.

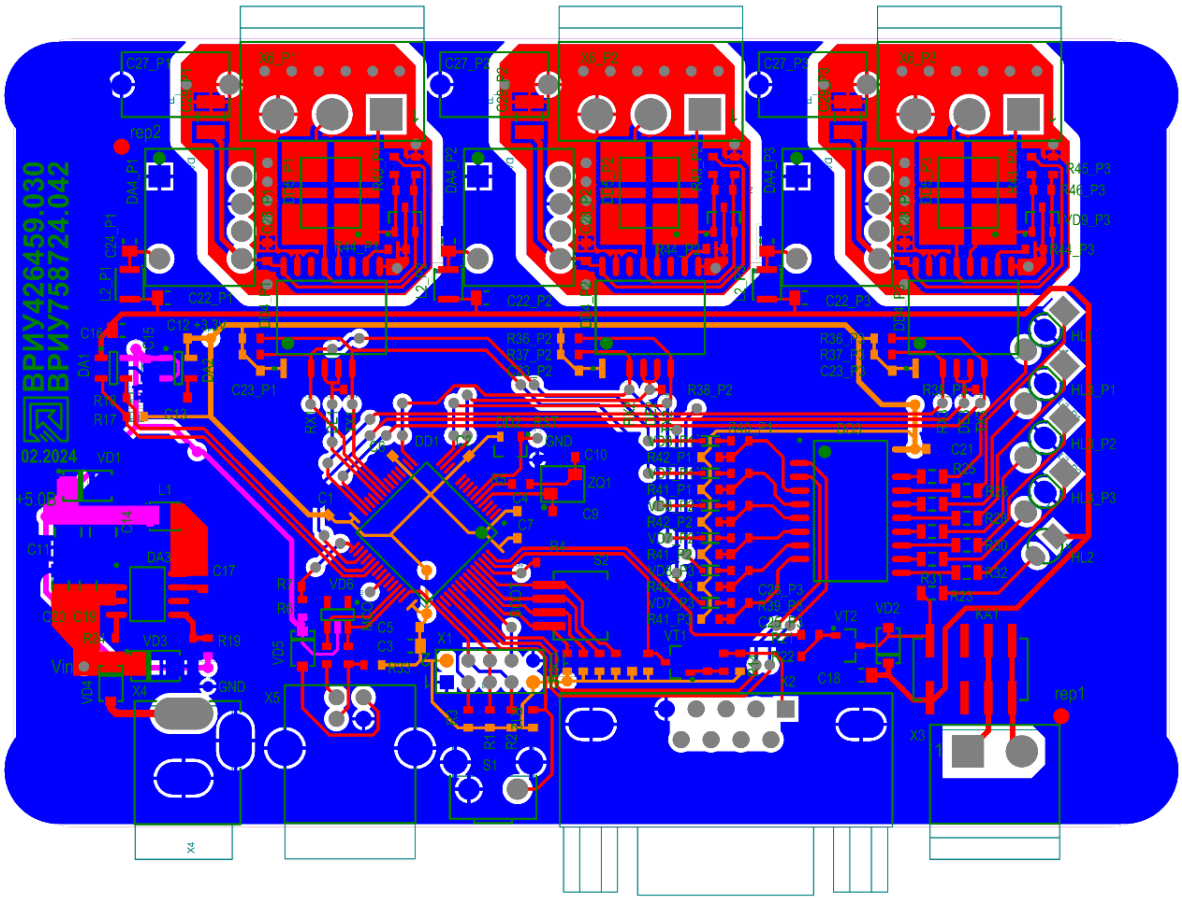

Рисунок 2.9 – Схема друкованої плати

Схематичне рішення апаратної частини забезпечує ефективну взаємодію між функціональними модулями системи та зовнішніми інтерфейсами. Використання мікроконтролера STM32F303RBT6 дозволяє реалізувати всі необхідні функції системи, включаючи Modbus сервер, фільтрацію даних та передачу даних через USB і RS-485.

Завдяки продуманій схемі підключення та оптимальному вибору компонентів, система забезпечує високу надійність і ефективність передачі даних.

2.2 Проектування системи блокування передачі телекомунікаційних даних

2.2.1 Алгоритм роботи СБПТД

Архітектура програмного забезпечення для системи блокування та передачі телекомунікаційних даних на основі мікроконтролера STM32 повинна забезпечити ефективну та надійну роботу всіх компонентів системи. Вона включає в себе кілька основних модулів, які взаємодіють між собою для забезпечення основних функцій системи. Кожен з цих модулів має своє призначення і відповідає за певний аспект функціонування системи.

Основні компоненти програмної архітектури:

- система управління (Kernel);
- Modbus сервер;
- модуль фільтрації даних;
- передавач (трансмітер);
- керування кнопкою диспетчера.

Система управління (Kernel) – ядро системи управління відповідає за координацію роботи всіх інших модулів, забезпечуючи належне виконання задач і управління ресурсами мікроконтролера. Здійснює управління задачами, пріоритетами та перериваннями.

Modbus сервер – забезпечує зв'язок з зовнішніми пристроями через протокол Modbus, дозволяє змінювати налаштування системи та отримувати дані [11]. Використовує USB інтерфейс для обміну даними. Обробляє запити на читання і запис даних, забезпечуючи необхідні функції управління.

Модуль фільтрації даних – аналізує отримані дані відповідно до заданих критеріїв, вирішує, чи пропускати пакет даних або ігнорувати його. Отримує дані від Modbus сервера та передавача, передає фільтровані дані до передавача. Включає алгоритми аналізу та перевірки даних.

Передавач (трансмітер) – забезпечує передачу фільтрованих даних до зовнішніх пристроїв через інтерфейс RS-485. Отримує оброблені дані від модуля фільтрації та забезпечує їх передачу через RS-485.

Керування кнопкою диспетчера – перевіряє стан кнопки диспетчера для прийняття рішення про пропуск або ігнорування пакету даних. Інтегрується з модулем фільтрації даних для прийняття рішень на основі вводу від диспетчера.

Взаємодія між компонентами здійснюється через стандартизовані інтерфейси та протоколи, що дозволяє забезпечити узгоджене та безперебійне виконання всіх функцій системи.

Блок-схема алгоритму роботи системи зображена на рисунку 2.10.

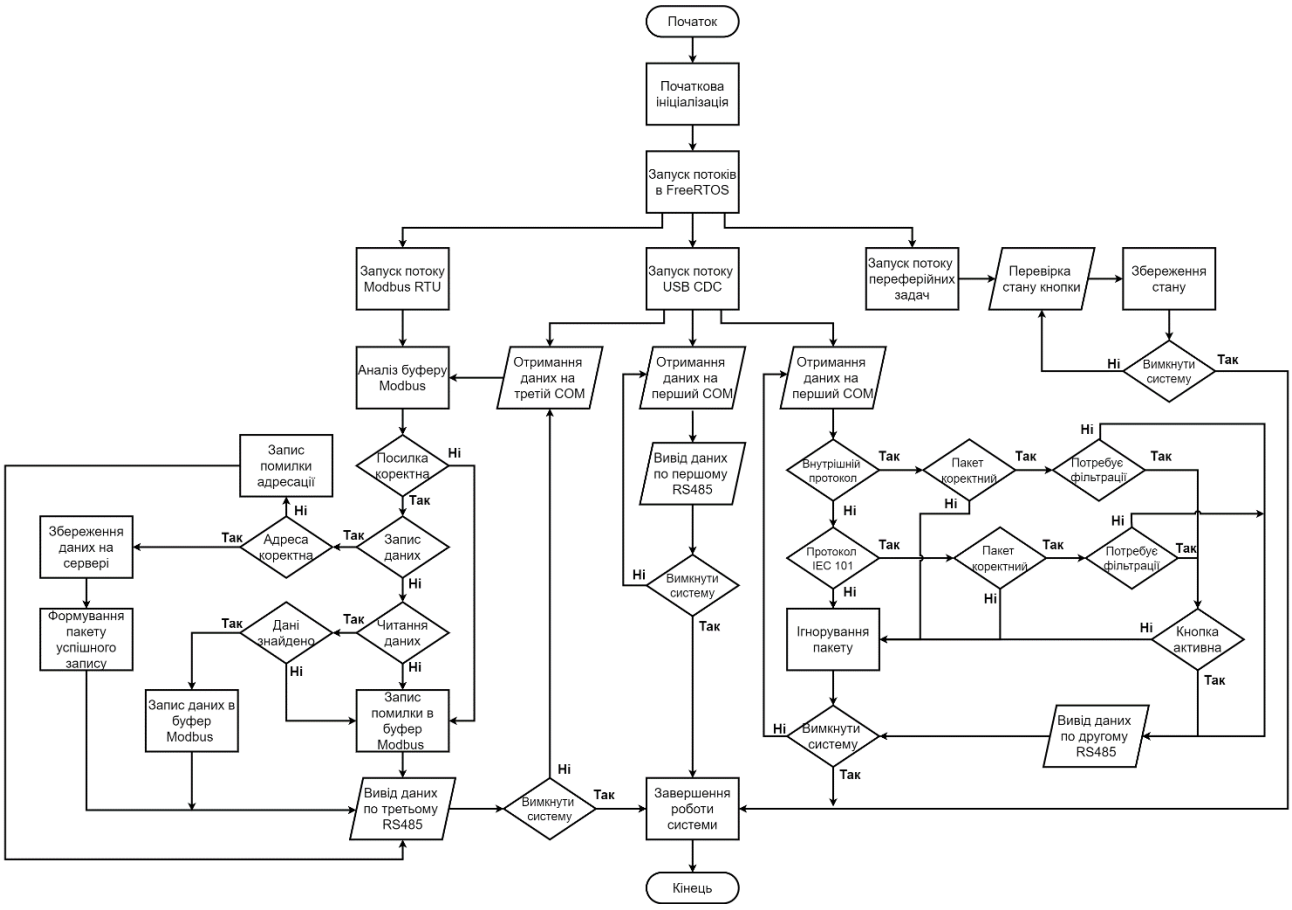

Рисунок 2.10 – Блок-схема алгоритму роботи системи

Основний алгоритм програми ініціалізує систему та запускає три завдання в FreeRTOS:

– перше завдання забезпечує роботу Modbus сервера;

– друге завдання обробляє отримані дані по usb та обробляє їх для передавача та фільтра;

– третє завдання індикує стан роботи системи.

Для забезпечення ефективного виконання всіх функцій системи використовується планування задач на основі їх пріоритетів:

– задача обробки Modbus запитів має високий пріоритет, оскільки забезпечує комунікацію з зовнішніми пристроями та управління системою;

– задача фільтрації даних має середній пріоритет, оскільки необхідно забезпечити своєчасну обробку отриманих даних;

– задача передачі даних має середній пріоритет, оскільки передача даних може бути здійснена з певною затримкою (яку дозволяють протоколи передачі даних) без втрати функціональності;

– задача управління кнопкою диспетчера має динамічний пріоритет, який залежить від стану кнопки (високий при активному стані).

Для забезпечення ефективної розробки та підтримки програмного забезпечення використовуються наступні бібліотеки:

– HAL (Hardware Abstraction Layer);

– FreeRTOS;

– USB_CDC.

Архітектура програмного забезпечення для системи блокування та передачі телекомунікаційних даних на основі мікроконтролера STM32 забезпечує надійну, ефективну та стабільну роботу системи. Використання чітко структурованих модулів, ефективних інтерфейсів взаємодії, а також сучасних бібліотек та шаблонів програмування дозволяє забезпечити високу продуктивність та надійність системи в умовах реального часу, що є критично важливим для енергетичної інфраструктури.

2.2.2 Використання STM32CubeIDE для розробки СБПТД

STM32CubeIDE є інтегрованим середовищем розробки (IDE), що надає всі необхідні інструменти для програмування, налагодження та тестування програмного забезпечення для мікроконтролерів STM32. Воно забезпечує комплексне рішення, що працюють з платформою STM32, поєднуючи

функціональні можливості для проектування програмного забезпечення, генерації коду та налагодження.

Основні компоненти STM32CubeIDE:

- редактор коду;
- інструменти генерації коду;
- налагодження та аналіз.

STM32CubeIDE підтримує мови програмування C і C++ та забезпечує підсвічування синтаксису, автоматичне завершення коду та рефакторинг. Середовище включає інтегровані інструменти для контролю версій (Git, TortoiseSVN).

Інтегроване середовище розробки містить в собі STM32CubeMX – графічний інструмент для налаштування апаратних ресурсів мікроконтролера та генерації початкового коду на основі HAL (Hardware Abstraction Layer). Середовище розробки надає підтримку різних бібліотек і стеків протоколів, таких як USB, TCP/IP, FATFS, FreeRTOS.

Графічний інтерфейс налагодження забезпечує можливості для покрокового виконання коду, встановлення точок зупину, моніторингу змінних та регістрів. Середовище дозволяє аналізувати виконання коду в режимі реального часу, що особливо важливо для систем реального часу.

Вбудований компонент RTOS Viewer – інструмент для моніторингу задач та їхнього стану під час виконання в середовищі FreeRTOS.

Переваги використання STM32CubeIDE:

- інтеграція з STM32CubeMX;
- зручність розробки та налагодження;
- підтримка RTOS;
- спільнота та підтримка.

Автоматична генерація коду компонентом STM32CubeMX дозволяє швидко налаштовувати апаратні ресурси мікроконтролера через графічний інтерфейс і автоматично генерувати необхідний код для їх використання.

Легка конфігурація периферійних модулів надає можливість швидко налаштовувати такі модулі, як UART, USB, GPIO, ADC, завдяки зручному графічному інтерфейсу.

Інтегрована налагоджувальна середовище надає можливість швидко виявляти та виправляти помилки в коді завдяки потужним інструментам налагодження.

Перегляд в реальному часі дозволяє моніторити та аналізувати виконання коду безпосередньо під час роботи мікроконтролера, що є критично важливим для систем реального часу.

Інтеграція з FreeRTOS дозволяє легко розробляти багатозадачні програми, забезпечуючи ефективне управління задачами та ресурсами.

Інструменти аналізу RTOS дозволяють моніторити задачі, пріоритети, споживання ресурсів та інші параметри роботи системи в режимі реального часу.

STM32CubeIDE має велику кількість документації, прикладів коду та навчальних матеріалів, що дозволяють швидко освоїти роботу з середовищем та ефективно використовувати його можливості. Велика спільнота STM32 забезпечує доступ до форумів, блогів та інших ресурсів, де можна отримати допомогу та обмінятися досвідом.

Використання STM32CubeIDE для розробки програмного забезпечення надає потужні інструменти для швидкої та ефективної реалізації проекту. Інтеграція з STM32CubeMX, підтримка FreeRTOS, зручні інструменти налагодження та велика кількість допоміжних ресурсів роблять STM32CubeIDE ідеальним вибором для створення надійних та ефективних систем.

2.2.3 Використання FreeRTOS для СБПТД

FreeRTOS — це популярна операційна система реального часу для вбудованих систем, що була реалізована на 35 мікроконтролерах. Доступна під ліцензіями MIT, та комерційною [4]. Система розроблялась як проста і легка система. Основною мовою реалізації є C. Кількість коду, з використанням асемблера приблизно 1%. Вона забезпечує методи для роботи з декількома

нитками або задачами, м'ютексами, семафорами і таймерами. А режим з таймером без переривань є доступний для малопотужних застосувань. Є підтримка пріоритетів завдань.

FreeRTOS (Free Real-Time Operating System) є популярною операційною системою реального часу для вбудованих систем, яка надає можливість створення багатозадачних програм. FreeRTOS забезпечує ефективне управління задачами, пріоритетами та ресурсами, що особливо важливо для систем, які потребують надійності та передбачуваності виконання [5].

Основною метою використання FreeRTOS є забезпечення стабільної та ефективної роботи всіх компонентів системи в умовах реального часу. Це дозволяє організувати програму у вигляді окремих задач, які можуть виконуватися незалежно одна від одної, що значно спрощує розробку та тестування складних систем.

FreeRTOS надає ряд важливих функцій та механізмів, які забезпечують багатозадачність. Одним з основних компонентів FreeRTOS є планувальник задач (scheduler), який відповідає за управління задачами та розподіл процесорного часу між ними. Планувальник може використовувати кілька алгоритмів планування, таких як планування з пріоритетами, що дозволяє задавати різні рівні важливості для різних задач.

Однією з ключових можливостей FreeRTOS є підтримка пріоритетів задач. Це дозволяє визначити, які задачі мають виконуватися першочергово, що особливо важливо для систем, де певні задачі мають вищий пріоритет. У системі блокування та передачі даних задачі обробки Modbus запитів можуть мати вищий пріоритет, ніж задачі передачі даних, оскільки від швидкої обробки запитів залежить правильність роботи всієї системи.

FreeRTOS також підтримує різні механізми синхронізації та взаємодії між задачами, такі як семафори, м'ютекси, черги повідомлень і таймери. Це дозволяє ефективно координувати роботу різних задач та забезпечувати їх взаємодію. Семафори можуть використовуватися для синхронізації доступу до спільних

ресурсів, м'ютекси — для запобігання одночасного доступу до критичних секцій коду, а черги повідомлень — для передачі даних між задачами.

Для реалізації багатозадачності в системі блокування та передачі даних використання FreeRTOS дозволяє розробити чітко структуровану архітектуру програмного забезпечення, де кожна задача має свій чітко визначений функціонал. В системі виділяються наступні задачі:

- обробка Modbus запитів;
- отримання даних по USB;
- індикація стану пристрою;
- перевірки стану кнопки диспетчера.

Крім того, FreeRTOS надає механізми управління пам'яттю, що дозволяє ефективно використовувати обмежені ресурси мікроконтролера. Вона підтримує динамічне виділення та звільнення пам'яті, що дозволяє гнучко керувати ресурсами в процесі виконання програми. Також дана RTOS надає можливість налаштування параметрів управління пам'яттю, що дозволяє оптимізувати використання пам'яті для конкретних потреб проекту.

FreeRTOS має багату документацію та підтримку, що значно полегшує процес розробки та інтеграції. Вона підтримує широкий спектр мікроконтролерів, включаючи STM32, і має велику кількість прикладів коду, які можуть бути використані як основа для власних розробок. Це дозволяє швидко освоїти роботу з FreeRTOS та використовувати її можливості для розробки ефективних та надійних систем.

Важливою перевагою FreeRTOS є її модульна структура, що дозволяє вибирати лише ті компоненти, які необхідні для конкретного проекту. Це дозволяє зменшити обсяг коду та оптимізувати його виконання. Крім того, FreeRTOS має високу портативність, що дозволяє легко переносити розроблене програмне забезпечення на інші платформи.

У системі блокування та передачі даних на основі STM32 використання FreeRTOS забезпечить високу надійність та передбачуваність виконання всіх задач, що є критично важливим для систем енергетичної інфраструктури. Вона

дозволить розробити чітко структуровану архітектуру програмного забезпечення, де кожна задача має свій чітко визначений функціонал, та забезпечити ефективну взаємодію між ними.

Використання FreeRTOS дозволяє зосередитися на реалізації функціоналу системи, не турбуючись про деталі управління задачами та ресурсами. Це значно спрощує процес розробки та забезпечує високу якість кінцевого продукту. Завдяки своїм можливостям та перевагам FreeRTOS є ідеальним вибором для реалізації багатозадачності в системах реального часу на основі мікроконтролерів STM32.

2.3 Реалізація інтерфейсів передачі даних системи блокування передачі телекомунікаційних даних

2.3.1 Програмна реалізація USB-to-COM портів СБПТД

Для зв'язку між мікроконтролерами та іншими пристроями через COM інтерфейс використовується клас USB Communication Device Class (CDC).

USB CDC є стандартним класом пристроїв USB, що визначає, як пристрій реалізує функціонал віртуального COM порту. Це дозволяє створити зв'язок між комп'ютером і мікроконтролером, де комп'ютер бачить мікроконтролер як стандартний COM порт. Однією з основних переваг цього підходу є те, що він забезпечує сумісність з великою кількістю операційних систем та програмного забезпечення, що працює з COM портами [14].

STM32CubeMX, інструмент для генерації коду, підтримує USB CDC клас і автоматично генерує початковий код для реалізації одного COM порту. Цей процес включає налаштування USB підсистеми мікроконтролера, створення необхідних дескрипторів USB та реалізацію обробки запитів від хоста. Однак, для реалізації задачі, яка включає створення кількох COM портів, необхідне додаткове масштабування коду.

Для початку, генерується базовий проект за допомогою STM32CubeMX, де налаштовується USB периферія як пристрій CDC. Після генерації коду,

створюється структура, яка забезпечує обробку одного серійного порту. Ця структура включає ініціалізацію USB, налаштування кінцевих точок (endpoints) та обробку переривань, що пов'язані з передачею та прийомом даних.

Наступним кроком є масштабування цього коду для підтримки кількох віртуальних COM портів. Це вимагає модифікації USB дескрипторів для додавання нових кінцевих точок, які будуть обробляти додаткові порти. Кожен новий COM порт потребує власної пари кінцевих точок для передачі і прийому даних. Це включає додавання нових інтерфейсів у дескриптор конфігурації USB та налаштування відповідних обробників переривань.

Для забезпечення багатопортової підтримки, створюються окремі буфери для кожного порту та реалізовуються функції, які будуть керувати цими буферами. Ці функції забезпечують правильне маршрутизацію даних між USB кінцевими точками та відповідними буферами серійних портів. Також забезпечується обробка запитів від хоста для кожного порту, що включає налаштування швидкості передачі даних, контрольних сигналів та інших параметрів COM порту.

Окрім технічних аспектів, слід також враховується ефективність і надійність програмної реалізації. Використання FreeRTOS у цьому контексті дозволяє створити окремі задачі для обробки кожного COM порту. Це забезпечує паралельну обробку даних, що надходять з різних портів, та підвищує загальну продуктивність системи. Кожна задача виконує прийом та передачу даних через відповідний порт.

Для розробки також враховується можливі проблеми з синхронізацією та блокуванням ресурсів. Використання м'ютексів та семафорів FreeRTOS допомагає уникнути конфліктів при доступі до спільних ресурсів, таких як буфери даних чи апаратні ресурси USB.

Розробка багатопортового USB-to-COM потребує ретельного тестування. Кожен порт буде протестований на предмет коректності передачі та прийому даних, а також сумісності з різними операційними системами та програмним

забезпеченням. Важливо забезпечити надійне відновлення роботи системи після можливих збоїв, пов'язаних з USB підключенням чи відключенням.

В результаті, програмна реалізація USB-to-COM портів на базі мікроконтролера STM32 та використання USB CDC класу дозволяє створити гнучку та ефективну систему для передачі даних. Завдяки можливостям STM32CubeMX та FreeRTOS, можна швидко та ефективно реалізувати багатопортові рішення, що відповідають вимогам сучасних вбудованих систем. Це забезпечує високу продуктивність та надійність, що є критично важливими для систем у сфері енергетичної інфраструктури.

2.3.2 Інтеграція протоколу Modbus в СБПТД

Протокол Modbus є одним з найпоширеніших протоколів для обміну даними в промислових мережах, особливо в галузі автоматизації та енергетики. Його простота, ефективність та надійність роблять його ідеальним для застосувань, де необхідно забезпечити обмін даними між різними пристроями та системами. Розглянемо інтеграцію протоколу Modbus в систему на основі мікроконтролера STM32, враховуючи, що всі елементи роботи з протоколом були розроблені самостійно, без використання сторонніх бібліотек, відповідно до специфікації протоколу.

Першим етапом розробки власної реалізації Modbus є детальне вивчення специфікації протоколу. Modbus підтримує кілька різних варіантів реалізації, включаючи Modbus RTU (Remote Terminal Unit), Modbus ASCII та Modbus TCP/IP. Для реалізації системи використовується Modbus RTU, оскільки він найбільш поширений у промислових додатках та забезпечує ефективне кодування даних з мінімальним накладним витратами.

Основними компонентами реалізації Modbus RTU є формування та парсинг повідомлень, обробка CRC-контролю, управління таймаутами та обробка помилок. Повідомлення Modbus RTU складаються з адреси пристрою, функціонального коду, даних та контрольної суми. Кожне повідомлення починається з паузи, яка сигналізує про початок нового повідомлення, і

завершується двома байтами CRC (Cyclic Redundancy Check), що забезпечують перевірку цілісності даних.

Другим етапом Реалізація Modbus є налаштування UART для обміну даними. На STM32 це включає налаштування відповідних апаратних ресурсів, таких як USART або UART модуль, вибір швидкості передачі даних, налаштування парності та стоп-бітів. Ці параметри визначаються специфікацією протоколу та вимогами конкретного застосування. Після налаштування апаратної частини реалізовуватиметься програмна обробка даних, що надходять через серійний порт.

Наступним етапом є реалізація обробки повідомлень Modbus включає розпізнавання початку та кінця повідомлення, перевірку контрольної суми та обробку функціональних кодів. Функціональний код 03 відповідає за читання значень регістрів, а код 06 - за запис одного регістру. Кожен функціональний код обробляється окремою функцією, яка виконує відповідні дії, такі як читання або запис даних в пам'ять мікроконтролера.

Важливою частиною реалізації є обробка CRC. Алгоритм CRC дозволяє виявляти помилки, що можуть виникнути при передачі даних. Для реалізації CRC використовується поліном, визначений специфікацією Modbus. Реалізація алгоритму CRC включає обробку кожного байта даних для обчислення контрольної суми, яка додається до кінця кожного повідомлення перед його передачею. На стороні приймача обчислена CRC порівнюється з отриманою для перевірки цілісності повідомлення.

Також важливим аспект – управління таймаутами реалізації Modbus. Оскільки Modbus RTU працює в реальному часі, необхідно враховувати затримки між повідомленнями. Це включає визначення максимальної тривалості паузи між байтами повідомлення та таймаутів для очікування відповіді від пристрою. Реалізація таймаутів забезпечує надійну роботу протоколу навіть у випадку втрати або пошкодження даних.

Обробка помилок включає розпізнавання та обробку різних типів помилок, таких як помилки таймауту, помилки CRC та інші. При виявленні помилки

протокол Modbus визначає, як саме має бути повідомлено про помилку та які дії необхідно виконати. Це включає повторну передачу повідомлення або повідомлення хоста про помилку для подальшої обробки.

Однією з основних переваг розробки власної реалізації Modbus є гнучкість та можливість оптимізації під конкретні вимоги проекту. Це дозволяє досягти максимальної продуктивності та надійності системи. Можна оптимізувати обробку повідомлень для мінімізації затримок або додати додаткові функції, які специфічні для даного застосування.

Для забезпечення надійної роботи протоколу Modbus також проводиться ретельне тестування. Це включає тестування на реальному обладнанні з використанням різних сценаріїв обміну даними, симуляцію помилок та перевірку правильності обробки помилок. Важливо також перевірити сумісність з різними пристроями, що підтримують Modbus, для забезпечення інтероперабельності.

Інтеграція протоколу Modbus у систему на базі STM32 забезпечує ефективний та надійний обмін даними між різними компонентами енергетичної інфраструктури. Завдяки детальному вивченню специфікації протоколу та розробці власної реалізації, вдалося створити систему, яка відповідає всім вимогам до надійності та продуктивності, забезпечуючи при цьому високу гнучкість та можливість подальшого розвитку.

2.3.3 Реалізація фільтрації даних в СБПТД

Фільтрація даних є одним з основних компонентів системи. Основна задача фільтра полягає в блокуванні певних повідомлень, що передаються через протокол IEC 60870-5-101, який широко використовується для телекомунікацій у галузі енергетики [7]. Цей протокол забезпечує обмін інформацією між контролерами та центральними системами управління, включаючи команди, сигнали та параметри обладнання.

Реалізація фільтрації даних розпочинається з визначення критеріїв для блокування повідомлень. У системі блокування виконується для повідомлень з

певним функціональним кодом. Функціональні коди в протоколі IEC 60870-5-101 визначають типи команд та повідомлень, що передаються між пристроями. Це можуть бути команди включення або вимкнення обладнання, запити на отримання даних або повідомлення про події.

Основна логіка фільтрації включає моніторинг всіх вхідних повідомлень і перевірку їх на відповідність заданим критеріям. При отриманні повідомлення система аналізує його структуру, витягує функціональний код та порівнює його з переліком кодів, що підлягають блокуванню. Якщо повідомлення містить функціональний код, який необхідно заблокувати, система переходить до наступного етапу - перевірки стану кнопки диспетчера.

Кнопка диспетчера виконує роль механізму ручного контролю, який дозволяє оператору втручатися в процес фільтрації повідомлень. Коли кнопка натиснута, блокування повідомлень ігнорується, і всі повідомлення, незалежно від їх функціонального коду, передаються далі по системі. Це дозволяє диспетчеру оперативно реагувати на надзвичайні ситуації або потреби в зміні параметрів роботи системи в ручному режимі.

Для реалізації цієї функціональності розробляється програмний модуль, що забезпечує обробку повідомлень протоколу IEC 60870-5-101. Цей модуль включає кілька основних компонентів: парсер повідомлень, логіку фільтрації та обробку стану кнопки диспетчера. Парсер повідомлень відповідає за аналіз вхідних даних, їх декодування та витягування необхідної інформації, включаючи функціональний код. Логіка фільтрації порівнює отримані коди з переліком заблокованих кодів і приймає рішення щодо подальшої обробки повідомлення.

Обробка стану кнопки диспетчера реалізується через переривання. У випадку використовується переривання, натискання кнопки викликає відповідне переривання, яке змінює стан прапорця. Цей прапорець враховується під час прийняття рішення про блокування повідомлень.

Для забезпечення надійної роботу фільтра реалізовуються механізми обробки помилок та відновлення після збоїв. Інтеграція фільтрації даних у загальну архітектуру системи потребує тісної взаємодії з іншими компонентами,

такими як модулі обробки даних, комунікаційні інтерфейси та інтерфейси користувача. Це включає передачу фільтрованих даних до інших компонентів системи, а також забезпечення інформування оператора про стан фільтрації та можливі помилки. Крім того, важливо забезпечити сумісність фільтра з іншими протоколами та стандартами.

Щоб забезпечити надійну роботу фільтра, будуть реалізовані механізми обробки помилок та відновлення після збоїв. Система виявлятиме некоректні або пошкоджені повідомлення та вживати відповідних заходів, таких як повторна передача або повідомлення оператора про помилку. Крім того, забезпечуватиметься збереження коректного стану системи після перезавантаження або відключення живлення, що реалізовуватиметься за допомогою збереження критичних налаштувань у енергонезалежній пам'яті.

Інтеграція фільтрації даних у загальну архітектуру системи потребує тісної взаємодії з іншими компонентами, такими як модулі обробки даних, комунікаційні інтерфейси та інтерфейси користувача. Це включає передачу фільтрованих даних до інших компонентів системи, а також забезпечення інформування оператора про стан фільтрації та можливі помилки. Крім того, важливо забезпечити сумісність фільтра з іншими протоколами та стандартами, що використовуються в енергетичній інфраструктурі.

Загалом, реалізація фільтрації даних у системі на основі мікроконтролера STM32 забезпечує високий рівень контролю над передачею інформації, що є критично важливим для роботи енергетичної інфраструктури. Завдяки детальному підходу до розробки та тестування, система забезпечує надійну та ефективну фільтрацію повідомлень, що дозволяє підвищити загальну безпеку та надійність роботи.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Реалізація програмного коду для системи блокування передачі телекомунікаційних даних

3.1.1 Розробка загальної структури системи СБПТД

Основні функції системи реалізуються через запуск завдань у середовищі реального часу FreeRTOS: обробка протоколу Modbus, обробка USB інтерфейсу, натискання кнопки диспетчера та індикація стану системи. Використання FreeRTOS дозволяє ефективно керувати багатозадачністю, забезпечуючи одночасне виконання цих завдань і раціональне використання ресурсів мікроконтролера STM32.

Завдання, відповідальне за протокол Modbus, полягає в обробці запитів від клієнтів Modbus та передачі відповідей. Це включає прийом даних через USB або RS485, їх обробку та формування відповідей. Протокол Modbus є широко розповсюдженим у промислових додатках для обміну інформацією між пристроями.

Основна функція, що виконує ці операції, називається modbusRTU. Вона перевіряє стан USB, обробляє Modbus запити та перевіряє можливі зміни в конфігурації. Лістинг функції modbusRTU зображений на рис. 3.1.

```
void modbusRTU(void) {
    checkCDCState();
    eMBPoll();
    chekChangeConf();
}
```

Рисунок 3.1 – Лістинг функції modbusRTU

Функція checkCDCState перевіряє стан USB CDC інтерфейсу та виконує перемикання доступу до Modbus сервера між портами USB та RS485. eMBPoll обробляє Modbus запити, а chekChangeConf перевіряє зміни в конфігурації системи та у випадку виконання команди зміни налаштувань виконує збереження у флеш пам’ять.

Завдання, що відповідає за обробку USB інтерфейсу, займається прийом та передачу даних через USB порти. Воно забезпечує роботу трьох віртуальних COM портів: Modbus, трансмітера та фільтра. Лістинг функції dataCDC зображений на рис. 3.2.

```
void dataCDC(void) {
     for (uint8_t i = 0; i < NUM_CDC; i++) {
          if (lenCDCRec[i] != 0) {
               if (i == TRANSMITER) {
                    CDC_Transmit(TRANSMITER,
     (uint8_t*)buffCDC[TRANSMITER], lenCDCRec[TRANSMITER]);
               } else if (i == FILTER) {
                    if (buffCDC[FILTER][POS_FILTER_TYPE] ==
          FILTER_TYPE_101||buffCDC[FILTER][POS_FILTER_TYPE] ==
          FILTER_TYPE_101_FIXED)
                         checkP101();
                    else
                         checkSEP();
               } else if (i == MODBUS) {
                    for (uint8_t j = 0; j < lenCDCRec[i]; j++){
                         buffCDCMB[j + lenCDCRecMB] =
                         buffCDC[i][j];}
                    lenCDCRecMB += lenCDCRec[i];
                    prvvCDCIRQHandler();
               }
               memset(buffCDC[i], 0, SIZE_CDC_BUFF);
               lenCDCRec[i] = 0;}}}
```

Рисунок 3.2 – Лістинг функції dataCDC

У цій функції дані, прийняті через USB, розподіляються між трьома різними портами. Якщо дані відносяться до трансмітера, вони передаються далі. Для фільтра дані перевіряються та обробляються відповідними функціями checkP101 або checkSEP. Дані для Modbus додаються до буфера і передаються для подальшої обробки.

Завдання, що стежить за натисканням кнопки диспетчера, відповідальне за перевірку стану кнопки і зміну відповідного стану системи. Кнопка дозволяє диспетчеру здійснювати певні дії, наприклад, дозволяти чи блокувати передачу даних.

Ця функція перевіряє стан GPIO піну, до якого підключена кнопка. Якщо кнопка натиснута, змінюється відповідний стан системи, що дозволяє або блокує передачу певних повідомлень.

Завдання індикації стану системи відповідає за відображення поточного стану системи за допомогою світлодіодів. Індикація стану системи реалізована шляхом зміни стану GPIO пінів, до яких підключені світлодіоди. Світлодіоди відображають різні стани, такі як активність передачі даних, помилки та зміну конфігурації.

Загальна архітектура програмного забезпечення включає ефективну взаємодію між цими завданнями, що забезпечує стабільну і надійну роботу системи. FreeRTOS дозволяє здійснювати пріоритетне виконання завдань, що особливо важливо для реального часу обробки даних у системах енергетичної інфраструктури.

3.1.2 Розробка USB CDC в СБПТД

Розробка USB CDC (Communication Device Class) для мікроконтролера STM32 є важливим етапом створення системи, яка дозволяє перетворювати USB інтерфейс в кілька віртуальних COM портів [20]. Це забезпечує гнучкий та надійний обмін даними між мікроконтролером і ПК або іншими пристроями через стандартний USB порт.

Першим етапом є налаштування апаратної частини мікроконтролера STM32 для роботи з USB інтерфейсом. Крім того, потрібно налаштувати зовнішній кварц або резонатор для забезпечення стабільної роботи USB модуля. Налаштування годинника мікроконтролера в CubeMX зображене на рис. 3.3.

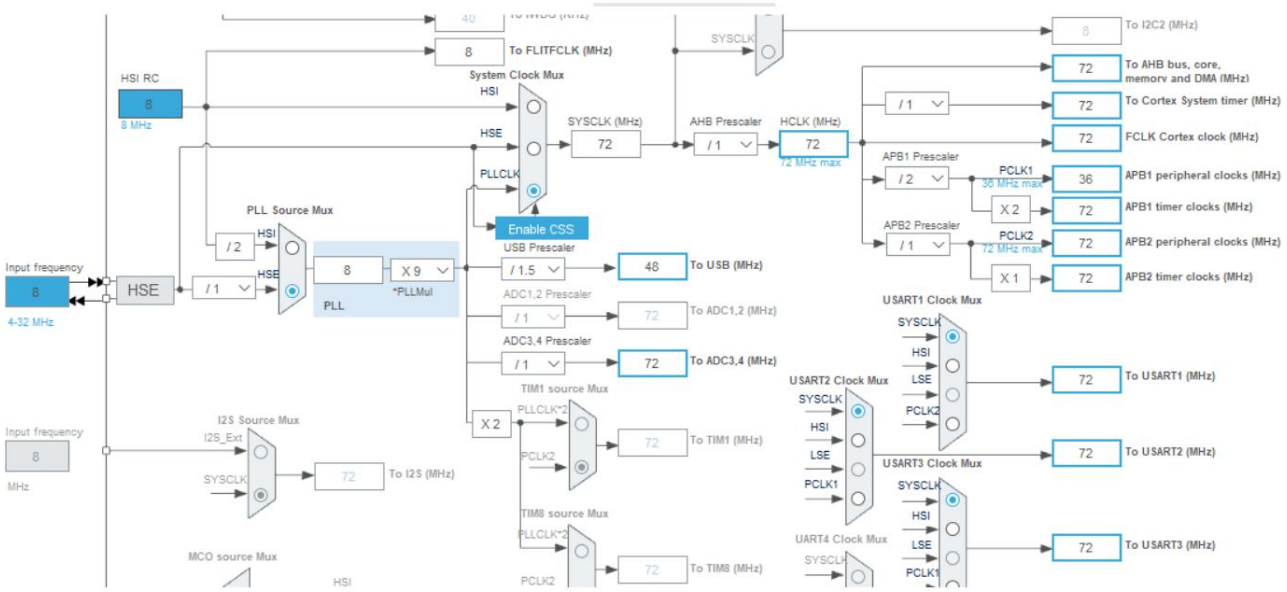

Рисунок 3.3 – Налаштування годинника мікроконтролера в CubeMX

Після налаштування апаратної частини, виконується конфігурація проекту у середовищі розробки STM32CubeIDE. Вибір STM32CubeIDE обґрунтований його зручністю для налаштування проектів на основі STM32, а також можливістю автоматичного генерування початкового коду для різних периферійних модулів, включаючи USB.

На першому етапі створення проекту в середовищі проектування вибирається конкретний мікроконтролер з серії STM32 – STM32F303RBT6. Після цього у вкладці “Configuration”, у розділі “Connectivity” елемент USB периферійний модуль, якому задаються налаштування для роботи в режимі Device (пристрій), що дає змогу автоматично згенерувати необхідний код для USB CDC. Налаштування STM32 в CubeMX зображене на рис. 3.4.

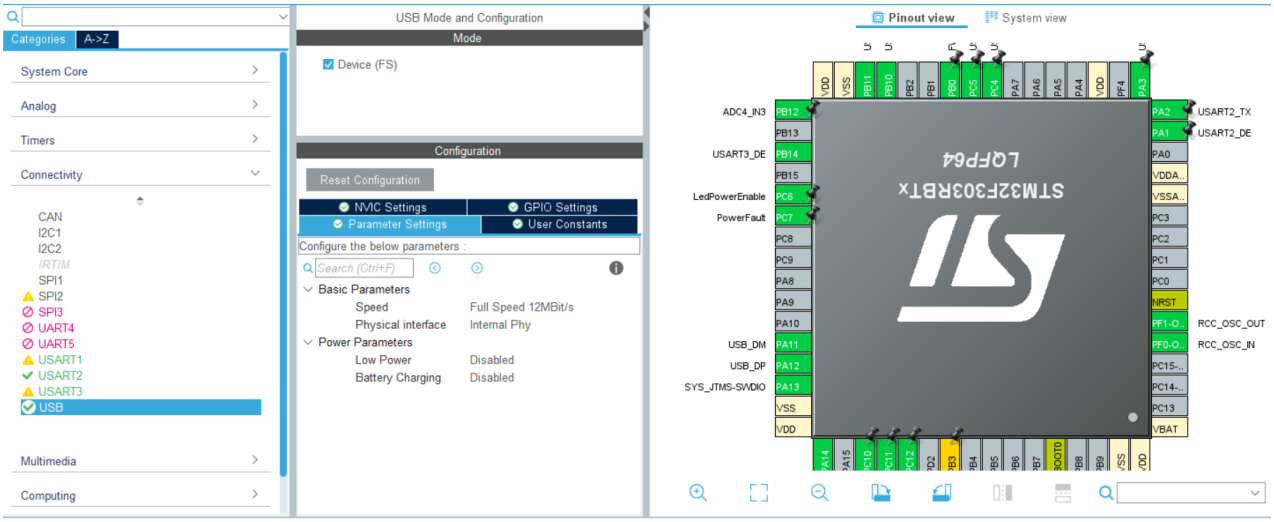

Рисунок 3.4 – Налаштування STM32 в CubeMX

Основна ідея масштабування USB CDC на кілька COM портів полягає у створенні декількох інстанцій USB CDC інтерфейсу. Це включає створення додаткових структур даних для кожного COM порту, а також модифікацію функцій передачі та прийому даних для підтримки багатопортового режиму. Для реалізації системи, для прийому та відправлення даних створений двовимірний масив та у функціях прийому, передачі та допоміжних функціях додано параметр, який вказує, який саме порт використовувати.

У файлі usbd_cdc_acm_if.h оголошені масиви для зберігання даних та допоміжні змінні для оперування з портами. Лістинг попередньо наведених змінних зображений на рис. 3.5.

```
#define APP_RX_DATA_SIZE 128
#define APP_TX_DATA_SIZE 128
/** RX buffer for USB */
uint8_t RX_Buffer[NUMBER_OF_CDC][APP_RX_DATA_SIZE];
/** TX buffer for USB, RX buffer for UART */
uint8_t TX_Buffer[NUMBER_OF_CDC][APP_TX_DATA_SIZE];
USBD_CDC_ACM_LineCodingTypeDef Line_Coding[NUMBER_OF_CDC];
uint32_t Write_Index[NUMBER_OF_CDC];
uint32_t Read_Index[NUMBER_OF_CDC];
```

Рисунок 3.5 – Лістинг основних змінних для роботи з портами

Для підтримки кількох інстанцій CDC змінюється файл дескриптора usbd_desc.c. Додаються нові інтерфейси для кожного додаткового COM порту, включаючи їх кінцеві точки. У файлі usbd_conf.c додається обробка нових кінцевих точок у функціях, які відповідають за ініціалізацію та обробку запитів USB. Лістинг частини одного доданого набору дескрипторів зображений на рис. 3.6.

```
/********************  CDC1 block ********************/
     /******** IAD to associate the two CDC interfaces */
      0x08,              /* bLength */
      0x0B,              /* bDescriptorType */
      _CDC_CMD_ITF_NBR, /* bFirstInterface */
      0x02,              /* bInterfaceCount */
      0x02,              /* bFunctionClass */
      0x02,              /* bFunctionSubClass */
      0x01,              /* bFunctionProtocol */
      0x00,              /* iFunction */
     /* Endpoint OUT Descriptor */
      0x07,                       /* bLength */
      USB_DESC_TYPE_ENDPOINT,     /* bDescriptorType */
      _CDC_OUT_EP,                /* bEndpointAddress */
      0x02,                       /* bmAttributes: Bulk */
      LOBYTE(CDC_DATA_HS_MAX_PACKET_SIZE),/* wMaxPacketSz */
      HIBYTE(CDC_DATA_HS_MAX_PACKET_SIZE),
      0x00, /* bInterval: ignore for Bulk transfer */
     /* Endpoint IN Descriptor */
      0x07,                       /* bLength: */
      USB_DESC_TYPE_ENDPOINT,     /* bDescriptorType*/
      _CDC_IN_EP,                 /* bEndpointAddress */
      0x02,                       /* bmAttributes: Bulk */
      LOBYTE(CDC_DATA_HS_MAX_PACKET_SIZE), /* wMaxPacketSz*/
      HIBYTE(CDC_DATA_HS_MAX_PACKET_SIZE),
      0x00, /* bInterval: ignore for Bulk transfer */
```

Рисунок 3.6 – Лістинг частини одного доданого набору дескрипторів

У файлі usbd_cdc_if.c додається параметр порту до функцій передачі та прийому. Лістинг модифікованої функції прийому даних зображений на рис. 3.7.

```
static int8_t CDC_Receive(uint8_t cdc_ch, uint8_t *Buf,
uint32_t *Len) {
        setLenCDC(cdc_ch, * Len);
        setBuffCDC(cdc_ch, Buf);
        USBD_CDC_SetRxBuffer(cdc_ch, &hUsbDevice, &Buf[0]);
        USBD_CDC_ReceivePacket(cdc_ch, &hUsbDevice);
        return (USBD_OK);
    }
```

Рисунок 3.7 – Лістинг модифікованої функції прийому даних

На завершення, тестування реалізованої функціональності USB CDC з кількома COM портами є важливим етапом. Після написання основного коду виконується перевірка коректності передачі та прийому даних для кожного порту, а також взаємодію між портами. Це включає перевірку роботи Modbus сервера, фільтрації даних і реперу в різних умовах експлуатації.

Розробка USB CDC для мікроконтролера STM32 з підтримкою кількох COM портів є складним, але надзвичайно важливим завданням для забезпечення гнучкості та надійності системи. Завдяки детальному підходу до налаштування апаратної частини, модифікації згенерованого коду та тестуванню, можна забезпечити високу якість та стабільність роботи системи у різних умовах.

3.1.3 Розробка Mobus сервера в СБПТД

Розробка Modbus сервера є ключовою частиною системи. Сервер Modbus відповідає за прийом запитів від клієнтів, обробку цих запитів та відправку відповідей. У реалізації всі елементи роботи з протоколом були розроблені без використання сторонніх бібліотек, з дотриманням специфікації протоколу [13].

Основні функції сервера Modbus включають обробку різних типів регістрів, а також передачу даних через USB інтерфейс. Зокрема, для реалізації

цих функцій було створено кілька колбеків, що відповідають за доступ до регістрів:

- module_vcp_access_holding_reg();
- module_vcp_access_coil();
- module_vcp_read_discrete_input();
- module_vcp_read_input_reg().

Лістинг колбеку доступу до регістрів утримання зображений на рис. 3.8.

```
    modbus_dev_status_t   _access_holding_reg(struct   modbus_dev
*device, int address, uint16_t *value, bool is_write) {
        if (is_write) {
            switch (address) {
                case STRILA_MB_MODULE_MBADDR_REGADDR: {
                    sys_cfg_set(SYS_CONFIG_MODULE_MODBUS_ADDRESS,
                        mbtohs(*value));
                    return MODBUS_DEV_OK;} // …
                default:
    { break; }}
        } else {
            switch (address) {
                case STRILA_MB_MODULE_SWVER_HIGH_REGADDR:
                  *value= htombs(sys_cfg_get
                    (SYS_CONFIG_SYSTEM_SWVER) >> 16);
                    return MODBUS_DEV_OK;
                default: break;
         }
       }
    }
```

Рисунок 3.8 – Лістинг колбеку доступу до регістрів утримання

Кожен з цих колбеків реалізує доступ до відповідного типу даних Modbus. Розглянемо детальніше реалізацію функції _access_holding_reg, яка обробляє запити на читання та запис регістрів утримання (holding registers).

3.2 Тестування системи блокування передачі телекомунікаційних даних

Тестування системи є важливим етапом для забезпечення її надійної роботи в реальних умовах експлуатації. Для цього була використана тестова плата з наявністю USB, що дозволяє перевірити роботу системи у різних режимах. Через обмеження можливості передачі даних на RS485 по UART було вирішено виконувати зворотню передачу даних на USB. Це створює додаткове навантаження на систему, що дозволяє краще оцінити її стресостійкість.

Для тестування системи було розроблено програмне забезпечення у Visual Studio з використанням Windows Forms .NET Framework. Це ПЗ дозволяє підключатися до COM портів, надсилати повідомлення з підрахунком контрольної суми, а також автоматично надсилати повідомлення з вказаним інтервалом для перевірки стресостійкості системи.

Для тестування передавача (Transmitter) використовується наступний тестовий сценарій:

а) підключити тестову плату до ПК через USB. Відображення підключеної плати в диспетчері пристроїв зображене на рис. 3.9;

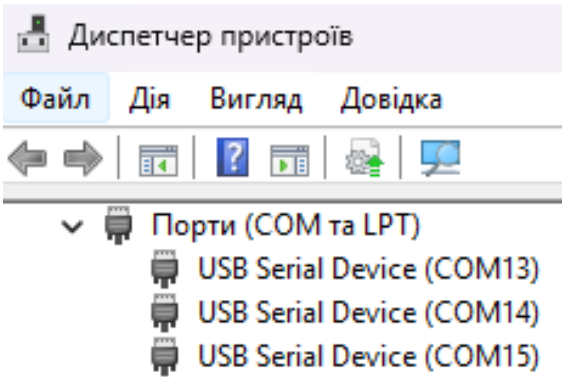

Рисунок 3.9 – Відображення підключеної плати в диспетчері пристроїв

б) запустити розроблене ПЗ;

в) вибрати відповідний COM порт для передачі даних;

г) надіслати тестові дані через USB на RS485 (зворотня передача на USB1). Обмін повідомленнями з передавачем зображений на рис. 3.10.

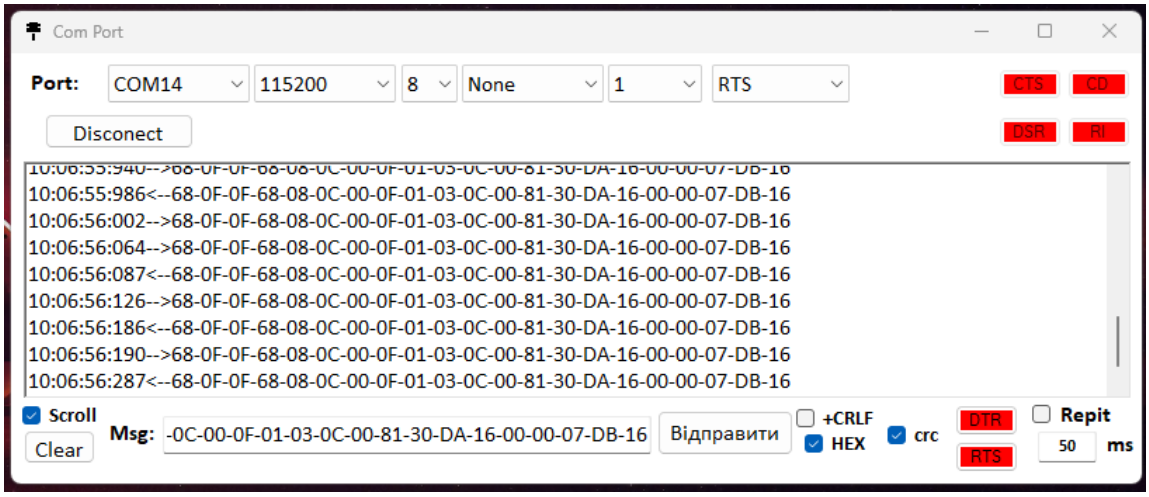

Рисунок 3.10 – Обмін повідомленнями з передавачем

Під час тестування передавача було встановлено, що система здатна ефективно обробляти великі обсяги даних на великій швидкості без втрати інформації та без збоїв у роботі.

Для тестування фільтра (Filter) використовується наступний тестовий сценарій:

а) підключити тестову плату до ПК через USB;

б) запустити розроблене ПЗ;

в) вибрати відповідний COM порт для передачі даних;

г) надіслати тестові пакети. Обмін повідомленнями з фільтром без натискання копки зображений на рис. 3.11;

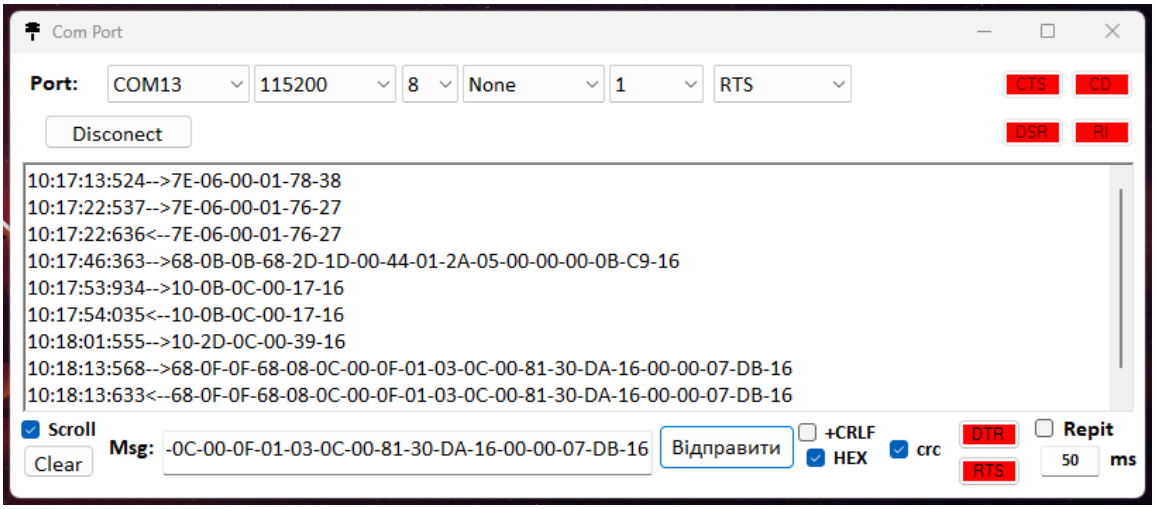

Рисунок 3.11 – Обмін повідомленнями з фільтром без натискання копки

ґ) натиснути кнопку та перевірити, що пакети передаються. Обмін повідомленнями з фільтром без натискання копки зображений на рис. 3.12.

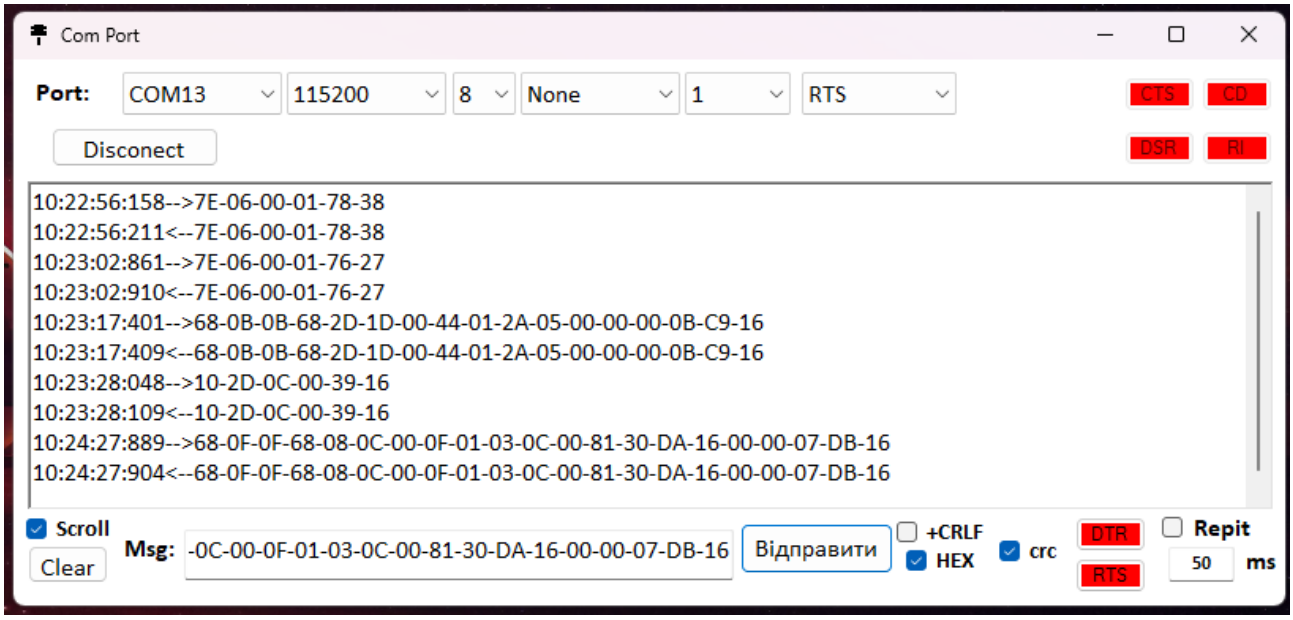

Рисунок 3.12 – Обмін повідомленнями з фільтром з натиснутою кнопкою

Під час тестування було встановлено, що фільтр коректно розпізнає та блокує відповідні пакети, забезпечуючи тим самим захист системи від несанкціонованого виконання команд. Також перевірено, що коли кнопка диспетчера натиснута, фільтр пропускає ці пакети, дозволяючи їх передачу.

Для тестування Modbus сервера використовується наступний тестовий сценарій:

а) Підключити тестову плату до ПК через USB.

б) Запустити ПЗ для роботи з Modbus сервером.

в) Провести операції з читання та запису регістрів і котушок через USB.

г) Перевірити коректність виконання всіх колбеків. Отримані дані від Modbus сервера зображені на рис. 3.13.

ґ) Відключити USB і перевірити, чи коректно переключається Modbus сервер.

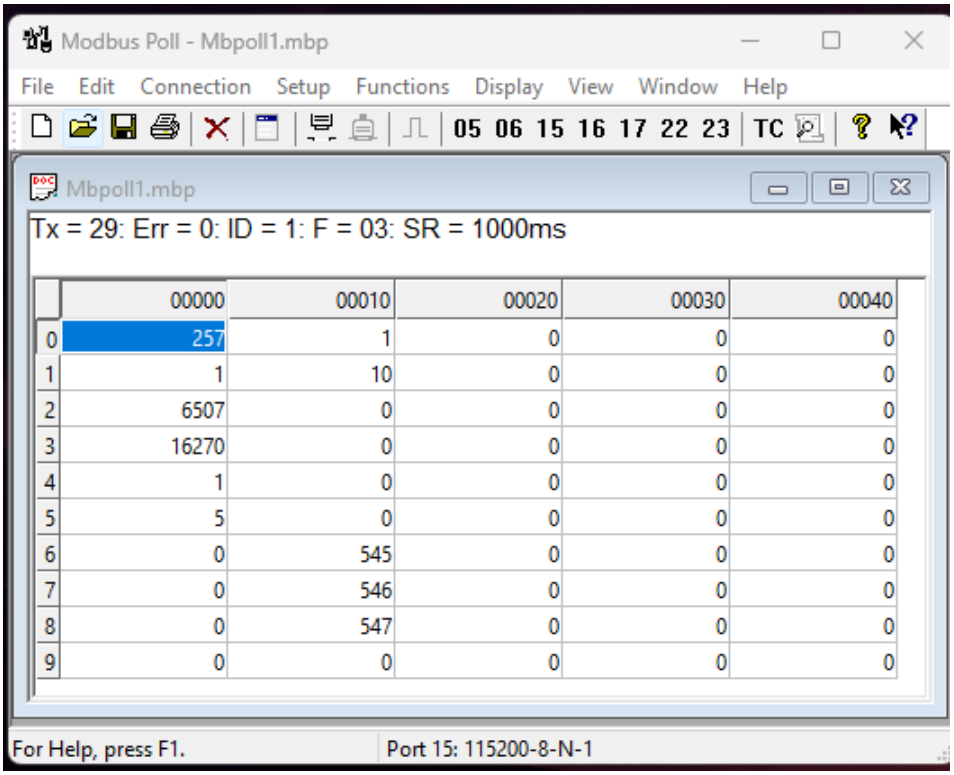

Рисунок 3.13 – Отримані дані від Modbus сервера

Для перевірки роботи Modbus сервера були протестовані всі реалізовані колбеки через USB. Під час тестування проводилися операції з читання та запису регістрів, і всі функції виконувалися коректно відповідно до специфікації протоколу Modbus.

Також була перевірена здатність системи до переключення Modbus сервера при відключенні USB. Під час відключення USB Modbus сервер коректно переключався, забезпечуючи безперервність роботи системи. Це важлива характеристика, оскільки в реальних умовах експлуатації можливі ситуації, коли USB з'єднання може бути втрачено або тимчасово недоступне. Система показала високу надійність та стійкість до таких змін [14].

Ці тестові сценарії забезпечують комплексну перевірку роботи системи в умовах наближених до реальних. Вони дозволяють оцінити її продуктивність, стійкість до навантажень та коректність виконання основних функцій. Використання розробленого ПЗ значно полегшує процес тестування та дозволяє швидко і точно визначати будь-які можливі проблеми в роботі системи.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Особливості заходів електробезпеки на підприємствах

При роботі з системою блокування передачі телекомунікаційних даних виділяють три системи засобів і заходів забезпечення електробезпеки:

- система технічних засобів і заходів;
- система електрозахисних засобів;
- система організаційно-технічних заходів і засобів.

Блок-схема системи електробезпеки зображена на рис. 4.1.

Рисунок 4.1 – Блок-схема системи електробезпеки

Технічні засоби і заходи з електробезпеки реалізуються в конструкції електроустановок при їх розробці, виготовленні і монтажі відповідно до чинних нормативів [21].

За своїми функціями технічні засоби і заходи забезпечення електробезпеки поділяються на дві групи:

– технічні заходи і засоби забезпечення електробезпеки при нормальному режимі роботи електроустановок;

– технічні заходи і засоби забезпечення електробезпеки при аварійних режимах роботи електроустановок.

Основні технічні засоби і заходи забезпечення електробезпеки при нормальному режимі роботи електроустановок включають:

– ізоляцію струмовідних частин;

– недоступність струмовідних частин;

– блоківки безпеки;

– засоби орієнтації в електроустановках;

– виконання електроустановок, ізольованих від землі;

– захисне розділення електричних мереж;

– компенсацію ємнісних струмів замикання на землю;

– вирівнювання потенціалів.

Із метою підвищення рівня безпеки, залежно від призначення, умов експлуатації і конструкції, в електроустановках застосовується одночасно більшість з перерахованих технічних засобів і заходів.

Ізоляція струмовідних частин. Забезпечує технічну працездатність електроустановок, зменшує вірогідність потраплянь людини під напругу, замикань на землю і на корпус електроустановок, зменшує струм через людину при доторканні до неізольованих струмовідних частин в електроустановках, що живляться від ізольованої від землі мережі за умови відсутності фаз із пошкодженою ізоляцією. Розрізняється ізоляція:

– робоча — забезпечує нормальну роботу електроустановок і захист від ураження електричним струмом;

– додаткова — забезпечує захист від ураження електричним струмом на випадок пошкодження робочої ізоляції;

– подвійна — складається з робочої і додаткової;

– підсилена — поліпшена робоча ізоляція, яка забезпечує такий рівень захисту як і подвійна.

При експлуатації електроустановок опір ізоляції приймається в межах 1 кОм/В, якщо технічними умовами не передбачені більш жорсткі вимоги відповідно до чинних актів. З метою забезпечення працездатності електроустановок і безпечної їх експлуатації проводиться контроль стану ізоляції, який характеризується електричною міцністю ізоляції, її електричним опором і діелектричними втратами.

Більшість електротравм пов'язані з дотиком до струмовідних частин електробустановок. Небезпека електротравм пов'язана, переважно, з дотиком до неізольованих струмовідних елементів електроустановок. Основними заходами забезпечення недоступності струмовідних частин є застосування захисних огороджень, закритих комутаційних апаратів (пакетних вимикачів, комплектних пускових пристроїв, дистанційних електромагнітних приладів управління споживачами електроенергії), розміщення неізольованих струмовідних частин на недосяжній для ненавмисного доторкання до них інструментом висоті, різного роду пристосуваннями тощо, обмеження доступу сторонніх осіб в електротехнічні приміщення.

Застосування блоківок безпеки. Блоківки безпеки застосовуються в електроустановках, експлуатація яких пов'язана з періодичним доступом до огороджених струмовідних частин (випробувальні і дослідні стенди, установки для випробування ізоляції підвищеною напругою), в комутаційних апаратах, помилки в оперативних переключеннях яких можуть призвести до аварії і нещасних випадків, в рубильниках, пусковій апаратурі, автоматичних вимикачах, які працюють в умовах підвищеної небезпеки.

Призначення блоківок безпеки: унеможливити доступ до неізольованих струмовідних частин без попереднього зняття з них напруги, попередити помилкові оперативні та керуючі дії персоналу при експлуатації електроустановок, не допустити порушення рівня електробезпеки та

вибухозахисту електрообладнання без попереднього відключення його від джерела живлення. Основними видами блоківок безпеки є механічні, електричні і електромагнітні.

4.2 Долікарська допомога при ураженні електричним струмом

При роботі з системою блокування передачі телекомунікаційних даних ураження електричним струмом є небезпечним та потенційно смертельним інцидентом, який може виникнути у будь-якому місці та часі.

Ураження електричним струмом відбувається, коли людина контактує з електричним струмом, що може призвести до травм, опіків або навіть смерті. Ознаки ураження електричним струмом включають:

- утруднене дихання або зупинка дихання;
- нерегулярний або відсутній пульс;
- опіки в місці контакту з джерелом електричного струму;
- втрата свідомості;
- судомні скорочення або ригідність м'язів;
- сплутаність свідомості або дезорієнтація;
- біль або дискомфорт у грудях.

При наданні першої допомоги особі, яка зазнала ураження електричним струмом, слід дотримуватися таких вимог:

а) забезпечте свою безпеку. Перш ніж наблизитися до потерпілого, переконайтеся, що джерело струму вимкнене або від'єднане від мережі, щоб запобігти подальшому травмуванню;

б) оцініть стан потерпілого. Перевірте наявність ознак дихання, пульсу та свідомості. Якщо потерпілий не дихає або у нього відсутній пульс, зателефонуйте в службу порятунку за номером 103 і негайно почати робити штучне дихання;

в) обробіть опіки. Якщо потерпілий отримав електричні опіки, накрийте їх чистою сухою тканиною або стерильною пов'язкою. Уникайте нанесення мазей або кремів на опіки;

г) спостерігайте за потерпілим. Слідкуйте за станом потерпілого і будьте готові зробити йому штучне дихання, якщо це необхідно. Якщо потерпілий притомний і стабільний, забезпечте йому тепло і комфорт до прибуття допомоги.

Звільняючи потерпілого від дії електричного струму, необхідно уникати наступних дій:

– доторкатися безпосередньо до потерпілого, поки він все ще перебуває в контакті з джерелом електричного струму;

– використання металевих предметів для відключення струму, оскільки вони можуть проводити електричний струм і збільшують ризик травмування;

– спроби пересунути обірвану лінію електропередач або інше джерело струму.

Щоб безпечно звільнити потерпілого від дії електричного струму, необхідно виконати наступні дії:

– відключити джерело живлення, якщо це можливо, або вимкнути автоматичний вимикач;

– якщо вимкнути живлення неможливо, використати непровідний матеріал (дерев'яну палицю або суху тканину), щоб відокремити потерпілого від джерела електричного струму.

Необхідно бути обережним та уникати прямого контакту з потерпілим, доки джерело струму не буде відключено або нейтралізовано.

Щоб уникнути ураження електричним струмом, слід дотримуватися правил безпеки:

– не вмикати пристрої з пошкодженим кабелем, вилкою або корпусом. Відключати пристрої від мережі, тримаючи за вилку, а не за провід;

– не коруватися електрикою біля води. Вода є провідником струму, тому потрібно уникати використання електричних пристроїв біля душу, раковини або інших джерел води;

– застосовувати правильну ізоляцію. Перевіряти, чи мають електричні кабелі та розетки належну ізоляцію, щоб уникнути прямого контакту з електричним струмом;

– не працювати з електрикою під час грози;

– отримати навички та знання з електробезпеки. Пройти курс з електробезпеки та дотримуватись рекомендацій фахівців, щоб зменшити ризик ураження електричним струмом;

– використовувати відповідні інструменти. При роботі з електричними системами, потрібно використовувати ізольовані інструменти та захисне обладнання;

– не перевантажувати електричні мережі. Уникати перевантаження розеток та електричних мереж, розподіляючи навантаження на декілька розеток. Не використовувати подовжувачі на постійній основі;

– відключати пристрої від мережі перед ремонтом. Перед тим, як почати ремонт або чистку електричного пристрою, обов'язково потрібно відключити його від мережі, щоб уникнути ураження струмом.

Дотримуючись правил електробезпеки, значно знизиться ризик ураження електричним струмом та підвищиться рівень безпеки.

Знання того, як надати першу допомогу при ураженні електричним струмом, може мати вирішальне значення в надзвичайних ситуаціях. Розуміючи ознаки ураження електричним струмом, забезпечуючи свою безпеку та надаючи відповідну першу допомогу, можна допомогти запобігти подальшим травмам і врятувати життя. Завжди необхідно звертатися за професійною медичною допомогою у разі ураження електричним струмом, оскільки внутрішні пошкодження можуть бути не одразу помітними.

ВИСНОВКИ

Розробка системи блокування та передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32 дозволила досягти всіх поставлених цілей та виконати заплановані завдання. В процесі роботи було проведено детальний аналіз вимог до комп'ютерної системи, розглянуто можливі рішення поставленого завдання та обрано оптимальні компоненти для реалізації проєкту.

Програмне забезпечення було розроблено з використанням STM32CubeIDE та FreeRTOS. Використання цих інструментів дозволило забезпечити високу продуктивність та багатозадачність системи. Були реалізовані основні функції системи, включаючи запуск завдань для Modbus, USB, натискання кнопки та індикації стану системи. Було створено та протестовано USB CDC для реалізації декількох COM портів, а також інтегровано протокол Modbus.

Однією з ключових частин системи стала реалізація фільтрації даних, яка дозволяє блокувати повідомлення з певними функціональними кодами в протоколах IEC 60870-5-101 та внутрішньому енергетичному протоколі. Було реалізовано механізм блокування повідомлень з можливістю їх пропускання за допомогою натискання кнопки диспетчером.

Тестування системи показало її високу надійність та стійкість до навантажень. Було використано спеціально розроблене програмне забезпечення для підключення до COM портів, надсилання повідомлень та перевірки роботи основних функцій системи.

Розроблена система успішно виконує свої функції та може бути ефективно використана в електроенергетичній інфраструктурі. Завдяки гнучкості та модульності, система може бути адаптована до різних умов експлуатації та потреб користувачів, що робить її цінним інструментом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ARM Cortex-M4 Development Cookbook. Mark Fisher. Packt Publishing, 2016. 300 с.
2. ARM Cortex-M4 Processor Technical Reference Manual. ARM, 2020. URL: https://developer.arm.com/documentation/ddi0439/b/ (дата звернення: 15.06.2024).
3. Developing USB CDC Class Devices. Cypress Semiconductor, 2015. URL: https://www.cypress.com/file/133656/download (дата звернення: 15.06.2024).
4. FreeRTOS Reference Manual. Real Time Engineers Ltd., 2021. URL: https://freertos.org/Documentation/RTOS_API.html (дата звернення: 15.06.2024).
5. FreeRTOS User Manual. FreeRTOS, 2021. URL: https://freertos.org/Documentation/RTOS_book.html (дата звернення: 15.06.2024).
6. IEC 60870-5-101: Telecontrol equipment and systems – Part 5-101: Transmission protocols – Companion standard for basic telecontrol tasks. International Electrotechnical Commission, 2003.
7. IEC 60870-5-104: Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles. International Electrotechnical Commission, 2006.
8. Modbus Application Protocol Specification V1.1b3. Modbus Organization, 2012. URL: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (дата звернення: 15.06.2024).
9. Modbus Messaging Implementation Guide. Modbus Organization, 2010. URL: https://modbus.org/docs/Modbus_Messaging_Implementation_V1_0b.pdf (дата звернення: 15.06.2024).
10. Modbus over Serial Line Specification and Implementation Guide V1.02. Modbus Organization, 2006. URL: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (дата звернення: 15.06.2024).

11. Modbus Protocol Implementation Guide. Modbus Organization, 2013. URL: https://modbus.org/docs/Modbus_Implementation_Guide_V1_0b.pdf (дата звернення: 15.06.2024).

12. Practical Industrial Data Communications: Best Practice Techniques. Deon Reynders, Steve Mackay, Edwin Wright. Elsevier, 2004. 432 с.

13. Practical Modbus Troubleshooting. Control Engineering, 2018. URL: https://www.controleng.com/articles/practical-modbus-troubleshooting/ (дата звернення: 15.06.2024).

14. Serial Communication Protocols for Embedded Systems. Elektor, 2019. URL: https://www.elektormagazine.com/magazine/elektor-201901/40112 (дата звернення: 15.06.2024).

15. STM32F3 Reference Manual. STMicroelectronics, 2020. URL: https://www.st.com/resource/en/reference_manual/dm00031020-stm32f40541507-and-stm32f303-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата звернення: 15.06.2024).

16. UART Communication Protocols. Texas Instruments, 2018. URL: https://www.ti.com/lit/an/slyt514/slyt514.pdf (дата звернення: 15.06.2024).

17. USB 2.0 Specification. USB Implementers Forum, 2000. URL: https://usb.org/sites/default/files/usb_20.pdf (дата звернення: 15.06.2024).

18. USB Class Definitions for Communication Devices. USB Implementers Forum, 2010. URL: https://www.usb.org/sites/default/files/CDC1.2_WMC1.1.zip (дата звернення: 15.06.2024).

19. USB Complete: Everything You Need to Develop Custom USB Peripherals. Jan Axelson. Lakeview Research LLC, 2019. 524 с.

20. USB Made Simple. Steve Ciarcia. Circuit Cellar, 1999. 300 с.

21. Гурик О. Я., Король О. І., Сенчишин В. С. Методичні вказівки до лабораторної роботи №2 з дисципліни :”Основи охорони праці” ”Дослідження метеорологічних умов у виробничих приміщеннях” . Тернопіль, 2016. 35 с.

22. Лещишин Ю. З. Розробка системи зв’язку як інтегрованого елементу роботизованих систем. Проблеми створення, розвитку та застосування

високотехнологічних систем спеціального призначення з урахуванням досвіду антитерористичної операції. Збірник тез доповідей XXI Всеукраїнської науковопрактичної конференції. Житомир, 2016. С. 102.

23. Лещишин Ю. З. Створення вбудованих систем на базі структурнопараметричних моделей цифрових каналів зв'язку : Лещишин Ю.З., Назаревич Т.О., Міська І.В. VIII Науково-технічна конференція «Інформаційні моделі, системи та технології». Тернопіль, 2020. С. 127.

24. Марків В.А., Осухівська Г.М., Лещишин Ю.З., Луцків А.М. Комп'ютерна система аутентифікації осіб : Матеріали XX наукової конференції ТНТУ ім. І. Пулюя. Тернопіль, 2017. С. 90–91.

25. Осухівська Г.М., Тиш Є.В., Луцик Н.С., Паламар А.М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2022. 28 с.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС _____ Осухівська Г.М.

“___” _____ 2024 р.

СИСТЕМА БЛОКУВАННЯ ПЕРЕДАЧІ ТЕЛЕКОМУНІКАЦІЙНИХ ДАНИХ ДЛЯ ДИСПЕТЧЕРА ЕЛЕКТРОЕНЕРГЕТИЧНОЇ ІНФРАСТРУКТУРИ НА ОСНОВІ STM 32

ТЕХНІЧНЕ ЗАВДАННЯ на __ листках

На здобуття освітньо-кваліфікаційного рівня *бакалавр*

Напрям *123 Комп’ютерна інженерія*

Спеціальність *123 Комп’ютерна інженерія*

«УЗГОДЖЕНО»

Керівник кваліфікаційної роботи

_____ к.т.н., доцент

Лещишин Ю.З.

“____” _____ 2024 р.

«ВИКОНАВЕЦЬ»

Студент групи СІс-42

_____ Косар Д.Б.

“____” _____ 2024 р.

Тернопіль 2024

1. Назва та підстава для виконання роботи.

1.1. Система блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32.

1.2. Підставою для виконання кваліфікаційної роботи бакалавра (КРБ) є Наказ по Університету (№ 4/7-468 від 26.04.2024 р.).

2. Виконавець.

2.1. Студент групи СІс-42 кафедри КС Тернопільського національного технічного університету ім. І. Пулюя, Косар Дмитро Богданович.

3. Мета роботи.

3.1. Метою роботи є розробити структуру та апаратне забезпечення система блокування передачі телекомунікаційних даних.

4. Склад виробу.

4.1. До складу виробу повинні входити:

- мікроконтролер;
- монтажні отвори;
- три роз’єми RS485;
- роз’єм USB.

5. Технічні вимоги.

5.1. Вимоги по призначенню.

5.1.1. Вбудована система повинна забезпечувати наступні функції:

- підтримка протоколів Modbus та IEC 60870-5-101;

– підключення до трьох COM портів через інтерфейс USB;
– роз’єм підключення програматора;
– фільтрація даних з можливістю пропуску або блокування;
– роз’єм DS1037;
– стабільна робота при одночасному обміні даними.

5.1.2. Система повинна живитись напругою постійного струму 5 В через USB порт та/або 12 В.

5.2. Вимоги до умов експлуатації.

5.2.1. По умовам експлуатації виріб повинен відповідати насиупним вимогам:

– робочий температурний діапазон: від -20°C до +85°C;
– відносна вологість: до 95% без конденсації;
– витримка впливу електромагнітних завад.

5.3. Конструктивні вимоги.

5.3.1. Корпус приладу в КРБ:

– розміри плати: не більше 2000 мм х 1000 мм;
– надійне кріплення всіх компонентів;
– зручний доступ до роз'ємів USB та кнопки управління.

5.3.2. Для побудови системи має бути використана сучасна компоненти.

5.3.3. При побудові системи необхідно передбачити наявність роз’ємів живлення і обміну даними.

5.4. Вимоги до надійності.

5.4.1. Система повинна відповідати вимогам ДСТУ 2862-94.

5.4.2. Наробка на відмову, не менше 30000 год.

5.5. Вимоги метрології.

5.5.1. Вимірювання параметрів системи при моделюванні повинно виконуватись на універсальних вимірювальних приладах.

6. Економічні показники.

6.1. Собівартість системи повинна бути не більше 3000 грн.

7. Вимоги до документації.

7.1. Конструкторська документація повинна відповідати вимогам ЄСКД та ДСТУ.

7.2. До складу документації повинно входити:

- ПЗ;
- термінал для тестування прботи системи;
- функціональна схема ЕЗ;
- блок схема алгоритму роботи КС.

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

8. Стадії та етапи розробки КРБ

8.1 Стадії та етапи виконання КРБ наведенні в таблиці 1.

Таблиця 1

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	*Розробка і затвердження технічного завдання*	*01.02 – 09.02*
2	*Аналіз технічного завдання*	*05.02 – 11.02*
3	*Аналіз вимог та принципів організації системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*26.04 – 03.05*
4	*Проєктування системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*04.05 – 13.05*
5	*Розробка схем і програмного забезпечення системи блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури*	*14.05 – 25.05*
6	*Розробка інструкцій з використання системи*	*26.05 – 09.06*
7	*Безпека життєдіяльності, основи охорони праці*	*10.06 – 15.06*
8	*Оформлення кваліфікаційної роботи*	*16.06 – 20.06*
9	*Попередній захист кваліфікаційної роботи*	*14.06*
10	*Захист кваліфікаційної роботи*	*24.06 – 28.06*

У дане ТЗ можуть вноситись зміни по узгодженню сторін.

Додаток Б

Перелік елементів електрично-принципової схеми

Позн.	Найменування	К-сть	Примітка
	Мікросхеми		
DD1	STM32F303RCT6	1	
DA2	MIC5319-3.3YD5	1	
DA3	LMR3820SDDAR	1	
DA4_P1-3	AM1P-0505SZ	3	
DD2	MCP130T-300I/TT	1	
DD3	MM74HC540WM	1	
DD4_P1-3	ISO15DW	3	
DD5_P1-3	TBU-DF055-050-WH	3	
HL1	L-93WEGW	1	
HL3_P1-3	L-93WEGW	3	
HL2	L-934ID	1	
KA1	FTR-B4SA4.5Z-B05	1	
L1	NLCV32T-2R2M-EFR	1	
L2_P1-3	NLCV32T-100K-EFR	3	
DA1	STMPS2141STR	1	
	Перемикачі		
S1	SWITRONIC 0743	1	
S2	DHN-04F-T-V	1	
	Транзистори		
VD2	MBR0540T1G	1	
VD4	MBR0540T1G	1	
VD5	MBR0540T1G	1	
VD3	SMAJ24.0A	1	
VD6	USBLC6-2SC6Y	1	
VD7_P1-3	RB520S-30	3	
VD8_P1-3	RB520S-30	3	
VD1	SMAJ5.0A	1	
VT1, VT2	BC817	2	

Змн.	Арк.	№ докум.	Підпис	Дата	КС КРБ 123.317.00.00 ПЕ		
Розроб.		Косар Д.Б.			Система блокування передачі телекомунікаційних даних для диспетчера електроенергетичної інфраструктури на основі STM 32 *Перелік елементів*	Літ.	Арк. 1 / Аркушів 2
Перевір.		Лещишин					
Реценз.						ТНТУ, каф. КС, гр. СІс-42	
Н. контр.		Луцик Н.С.					
Зав. каф.		Осухівська					

Позн.	Найменування	К-сть	Примітка
	Конденсатори		
C1-13	0.1 CAP0603	13	
C1, C12	1.0 CAP0603	2	
C3	4.7 CAP0805	1	
C9, C10	15p CAP0603	2	
C11, C14	22.0 CAP1210	2	
C16-25	2.2 CAP0805	9	
C26	4.7 CAP1206	1	
C25P_1-3	0.33 CAP0603	3	
C27	2200 Disc cap D10x6_P10	1	
	Резистори		
R1-25	YAGEO 10k RES0603	25	
R26	YAGEO 100k RES0603	1	
R27	YAGEO 24,9 RES0603	1	
R33	YAGEO 1.5k RES0603	1	
R28-30	YAGEO 270 RES0603	2	
R39_P1-3	YAGEO 330 RES0603	3	
R5	YAGEO 220 RES0603		
	Периферія		
X3	XY2500R-B(5.0)-2P +	1	
X4	DJK-02A	1	
X5	USB Type B	1	
X6_P1-3	XY2500R-B(5.0)-3P +	3	
X1	DS1025-06-2x5_(PLD2-10-(2x5))	1	
X2	DS1037-09F	1	

Додаток В

Лістинги програмного забезпечення КС

Лістинг файла Main.cpp

```
#include "main.h"
#include "cmsis_os.h"
#include "adc.h"
#include "tim.h"
#include "usart.h"
#include "usb.h"
#include "gpio.h"
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#include "mb.h"
#include "modbus_dev.h"
#include "mbport.h"
#include "panic.h"

#include "system.h"

#include "usb_device.h"
#include "usbd_cdc_acm_if.h"

#define POS_FILTER_TYPE 0

#define NUM_FUNC_PACET 0x2D
#define FILTER_TYPE_101 0x68
#define FILTER_TYPE_101_FIXED 0x10
#define POS_FUNCT_PACET 4
#define POS_FUNCT_PACET_F 1
#define POS_SIZE 1

#define FILTER_TYPE_SEP 0x7E
```

```
#define NUM_FUNC_SEP 0x78
#define POS_FUNCT_PACET_SEP 4
void SystemClock_Config(void);
void MX_FREERTOS_Init(void);
void checkCDCState(void);
void checkSwitchModbusSerialPort(void);

bool checkP101(void);
bool checkSEP(void);

uint8_t crc8(const uint8_t *data, size_t len);
uint8_t UpdateCRC(uint8_t dataByte, uint8_t crcPrev);
uint16_t calculate_checksum(const uint8_t *data, size_t len,
uint8_t start);

extern void prvvCDCIRQHandler(void);

GPIO_PinState btnState;

uint8_t riState = 0;

USBD_ConnectStatusTypeDef cdcState = 0;
USBD_ConnectStatusTypeDef oldCdcState = 0;

uint8_t cdcPort = UART_PORT_3;
uint8_t oldCdcPort = UART_PORT_3;

uint8_t changeConf = 0;

uint32_t timeSec = 0;

uint8_t buffCDC[NUM_CDC][SIZE_CDC_BUFF];
uint32_t lenCDCRec[NUM_CDC];

uint32_t buffCDCMB[SIZE_CDC_MB_BUFF];
uint32_t lenCDCRecMB;
```

```
struct serial_port {
     uint8_t address;
    int baudrate;
    eMBParity parity;
    uint8_t port_num;
    uint8_t stop_bits;
};

struct serial_port serialConfig;

typedef enum
{
     MODBUS,
     TRANSMITER,
     FILTER
} VCP_FunctionalTypeDef;

/**
 * \brief Receiving all data for MODBUS, REPEATER, FILTER;
ansver for REPEATER, FILTER
 */
void dataCDC(void) {
     for (uint8_t i = 0; i < NUM_CDC; i++) {
          if (lenCDCRec[i] != 0) {
               if (i == TRANSMITER) {
                    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0,
GPIO_PIN_SET);
                    HAL_UART_Transmit_IT(&huart1, (uint8_t*)
buffCDC[TRANSMITER], lenCDCRec[TRANSMITER]);
                    //CDC_Transmit(REPEATER, (uint8_t*)
buffCDC[REPEATER], lenCDCRec[REPEATER]);
               } else if (i == FILTER) {
                    bool msgState;
                    if (buffCDC[FILTER][POS_FILTER_TYPE] ==
FILTER_TYPE_101
```

```
                                        ||
buffCDC[FILTER][POS_FILTER_TYPE] == FILTER_TYPE_101_FIXED)
                                msgState = checkP101();
                            else if (buffCDC[FILTER][POS_FILTER_TYPE]
== FILTER_TYPE_SEP)
                                msgState = checkSEP();
                            if (!msgState)
                                clearFilterBuff();

                        } else if (i == MODBUS) {
                            for (uint8_t j = 0; j < lenCDCRec[i]; j++)
{
                                buffCDCMB[j + lenCDCRecMB] =
buffCDC[i][j];
                            }
                            lenCDCRecMB += lenCDCRec[i];
                            prvvCDCIRQHandler();
                            memset(buffCDC[MODBUS], 0, SIZE_CDC_BUFF);
                            lenCDCRec[MODBUS] = 0;
                        }
                        //memset(buffCDC[TRANSMITER], 0,
SIZE_CDC_BUFF);
                        //lenCDCRec[TRANSMITER] = 0;
                    }
                }
            }


        /**
         * \brief Button for test, on push ring+
         */
        void btnSend(void) {
            btnState = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
            if (btnState)
                riState = 1;
            else
```

```
                riState = 0;

          if (timeSec % 2 == 0) {
                //off
                HAL_GPIO_WritePin(LedError_GPIO_Port,
LedError_Pin, GPIO_PIN_SET);
                HAL_GPIO_WritePin(LedOneSec_GPIO_Port,
LedOneSec_Pin, GPIO_PIN_SET);
          }
          else {
                //GR
                HAL_GPIO_WritePin(LedError_GPIO_Port,
LedError_Pin, GPIO_PIN_SET);
                HAL_GPIO_WritePin(LedOneSec_GPIO_Port,
LedOneSec_Pin, GPIO_PIN_RESET);
          }
     }

     /**
      * \brief Modbus main function
      */
     void modbusRTU(void) {
          checkCDCState();
          eMBPoll();
          chekChangeConf();
     }

     /**
      * \brief Change port ? VCP : UART
      */
     void checkCDCState(void) {
          if (oldCdcPort != cdcPort) {
               eMBDisable();
               if (cdcState == USB_LOGIC_CONNECT) {
                    eMBInit(MB_RTU, serialConfig.address, VCP_PORT,
serialConfig.baudrate, serialConfig.parity);
```

```
                } else {
                    eMBInit(MB_RTU, serialConfig.address,
UART_PORT_3, serialConfig.baudrate, serialConfig.parity);
                }
                eMBEnable();
                modbusBuffClear();
                oldCdcPort = cdcPort;
            }
        }

        /**
         * \brief Checking flag change configuration ? save
configuration : restart
         */
        void chekChangeConf(void){
            if (changeConf == SET_CHANGES){
                sys_cfg_save();
                changeConf = NO_CHANGES;
            }
            else if (changeConf == DEV_RESET){
                changeConf = NO_CHANGES;
                NVIC_SystemReset();
            }
        }

        /**
         * \brief Checking frame on norm checksum and standart form
for IEC 60870-5-101
         */
        bool checkP101(void) {
            if (buffCDC[FILTER][POS_FILTER_TYPE] == FILTER_TYPE_101)
{
                if (buffCDC[FILTER][POS_FUNCT_PACET] !=
NUM_FUNC_PACET
                        || (buffCDC[FILTER][POS_FUNCT_PACET] ==
NUM_FUNC_PACET && riState)) {
```

```
                    //uint16_t checksum_D =
calculate_checksum(buffCDC[FILTER], lenCDCRec[FILTER], 4); // check
cheksum for debug
                    if (calculate_checksum(buffCDC[FILTER],
lenCDCRec[FILTER], 4) == buffCDC[FILTER][lenCDCRec[FILTER] - 2]){
                         HAL_UART_Transmit_IT(&huart2, (uint8_t*)
buffCDC[FILTER], lenCDCRec[FILTER]);
                         return true;
                    }
               }
          } else {
               if (buffCDC[FILTER][POS_FUNCT_PACET_F] !=
NUM_FUNC_PACET
                         || (buffCDC[FILTER][POS_FUNCT_PACET_F] ==
NUM_FUNC_PACET && riState)) {
                    //uint8_t checksumF_D =
calculate_checksum(buffCDC[FILTER], lenCDCRec[FILTER], 1); // check
cheksum for debug
                    if (calculate_checksum(buffCDC[FILTER],
lenCDCRec[FILTER], 1) == buffCDC[FILTER][lenCDCRec[FILTER] - 2]){
                         HAL_UART_Transmit_IT(&huart2, (uint8_t*)
buffCDC[FILTER], lenCDCRec[FILTER]);
                         return true;
                    }
               }
          }
          return false;
     }

     /**
      * \brief Checking frame on norm crc8 and standart form for
SEP
      */
     bool checkSEP(void) {
          if (buffCDC[FILTER][POS_FUNCT_PACET_SEP] != NUM_FUNC_SEP
```

```
                    || (buffCDC[FILTER][POS_FUNCT_PACET_SEP] ==
NUM_FUNC_SEP && riState)) {
               //uint8_t crc8_D = crc8(buffCDC[FILTER],
lenCDCRec[FILTER] - 1); // check crc8 for debug
               if (crc8(buffCDC[FILTER], lenCDCRec[FILTER] - 1) ==
buffCDC[FILTER][lenCDCRec[FILTER] - 1]){
                    HAL_UART_Transmit_IT(&huart2, (uint8_t*)
buffCDC[FILTER], lenCDCRec[FILTER]);
                    return true;
               }
          }
          return false;
     }

     void clearFilterBuff(void) {
          memset(buffCDC[FILTER], 0, SIZE_CDC_BUFF);
          lenCDCRec[FILTER] = 0;
     }
     /* 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_USB_PCD_Init();
  //MX_USART3_UART_Init();
  MX_TIM4_Init();
  //MX_TIM7_Init();
  MX_ADC4_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  MX_USB_DEVICE_Init();
  HAL_TIM_Base_Start_IT(&htim4);
  /* Init protocol modbus */
  _init_protocol_modbus();
     eMBDisable();
     eMBEnable();
  /* USER CODE END 2 */

  /* Call init function for freertos objects (in freertos.c)
*/
  MX_FREERTOS_Init();

  /* Start scheduler */
  osKernelStart();
```

```
  while (1)
  {
    // main function
  }
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {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_MUL6;
  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_DIV2;
       RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

       if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1)
!= HAL_OK)
       {
         Error_Handler();
       }
       PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USB|RCC_PERIPHCLK_USART1

|RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_USART3
                                    |RCC_PERIPHCLK_ADC34;
       PeriphClkInit.Usart1ClockSelection =
RCC_USART1CLKSOURCE_SYSCLK;
       PeriphClkInit.Usart2ClockSelection =
RCC_USART2CLKSOURCE_SYSCLK;
       PeriphClkInit.Usart3ClockSelection =
RCC_USART3CLKSOURCE_SYSCLK;
       PeriphClkInit.Adc34ClockSelection = RCC_ADC34PLLCLK_DIV1;
       PeriphClkInit.USBClockSelection = RCC_USBCLKSOURCE_PLL;
       if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
       {
         Error_Handler();
       }
     }

     /* USER CODE BEGIN 4 */
     void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
          if (huart->Instance==USART1) {
               HAL_GPIO_WritePin(FW_USART1_DE_GPIO_Port,
FW_USART1_DE_Pin, GPIO_PIN_RESET);
```

```
            memset(buffCDC[TRANSMITER], 0, SIZE_CDC_BUFF);
            lenCDCRec[TRANSMITER] = 0;
        }
        else if (huart->Instance==USART2){
            clearFilterBuff();
        }
    }
    /**
     * \brief Set settings for modbus
     *
     * \param adres - modbus adres
     *     baudrate - modbus baudrate
     *     parity - modbus parity
     */
    void mbSettingsSet(uint8_t adres, int baudrate, uint8_t
parity){
        serialConfig.address = adres;
        serialConfig.baudrate = baudrate;
        serialConfig.parity = parity;
    }

    /**
     * \brief Set change configuratin flag
     *
     * \param changeConfig - flag for change configuration
     */
    void setChangeConf(uint8_t _changeConf){
        changeConf = _changeConf;
    }

    /**
     * \brief Set length frame on CDC VCP
     *
     * \param cdc_ch - channel of CDC
     *          Len    - length frame on channel
     */
```

```
void setLenCDC(uint8_t cdc_ch, uint8_t Len ){
     lenCDCRec[cdc_ch] = Len;
}

/**
 * \brief Get length frame on CDC VCP
 *
 * \param cdc_ch - channel of CDC
 * \return length frame on channel
 */
uint32_t getLenCDC(uint8_t cdc_ch ){
     return lenCDCRec[cdc_ch];
}

/**
 * \brief Get CDC VCP state
 *
 * \return cdcState - state of CDC
 */
int getCDCState(){
     return cdcState;
}

/**
 * \brief Set CDC VCP state
 *
 * \param CDCstate - state of CDC
 */
void setCDCState(uint8_t CDCstate) {
     if (CDCstate == USB_LOGIC_DISCONNECT && cdcState ==
USB_HARD_DISCONNECT)
     {
          cdcState =  USB_HARD_CONNECT;
          cdcPort = UART_PORT_3;
     }
```

```
          else if (CDCstate == USB_LOGIC_CONNECT && cdcState ==
USB_HARD_CONNECT)
          {
               cdcState =  USB_LOGIC_CONNECT;
               cdcPort = VCP_PORT;
          }
          else if (CDCstate == USB_LOGIC_DISCONNECT && cdcState ==
USB_LOGIC_CONNECT ){
               cdcState = USB_LOGIC_DISCONNECT;
               cdcPort = UART_PORT_3;
          }
          else if (CDCstate == USB_HARD_DISCONNECT){
               cdcState = USB_HARD_DISCONNECT;
               cdcPort = UART_PORT_3;
          }
     }

     /**
      * \brief Get ring state
      *
      * \return ring State
      */
     uint8_t getRiState(){
          return riState;
     }

     /**
      * \brief Set modbus buffer
      *
      * \param cdc_ch - channel of CDC
      *          Buf    - CDC buffer
      */
     void setBuffCDC(uint8_t cdc_ch, uint8_t * Buf){
          for (uint8_t i = 0; i < lenCDCRec[cdc_ch]; i++) {
               buffCDC[cdc_ch][i] = Buf[i];
          }
```

```
    }

    /**
     * \brief Get modbus buffer
     *
     * \param  iterator - channel of CDC
     * \return bufer data
     */
    uint8_t getBuffCDCMB(uint16_t iterator){
        return  buffCDCMB[iterator];
    }

    /**
     * \brief Clear modbus buffer
     */
    void modbusBuffClear (void) {
        lenCDCRecMB = 0;
        memset(buffCDCMB, 0, SIZE_CDC_MB_BUFF *
sizeof(*buffCDCMB));
    }

    /**
     * \brief Get length received frame on modbus
     *
     * \return length received frame on modbus
     */
    uint32_t getLenRecMB() {
        return lenCDCRecMB;
    }

    /**
     * \brief Volatile memset array
     */
    void volatile_memset(volatile void *ptr, int value, size_t
num) {
        volatile unsigned char *p = ptr;
```

```
        while (num--) {
            *p++ = value;
        }
    }

    /**
     * \brief CRC8
     *
     * \param  *data - buffer data
     *             len  - length bufer
     * \return  crc8
     */
    uint8_t crc8(const uint8_t *data, size_t len) {
        //uint8_t crc = 0;
        //for (size_t i = 0; i < len; ++i) {
            //crc ^= data[i];
            //for (int j = 0; j < 8; ++j) {
                //if (crc & 0x80)
                    // crc = (crc << 1) ^ 0x07; // Polynomial for
CRC-8
                //else
                    // crc <<= 1;
            //}
        //}
        //return crc;

         uint8_t crc = 0;

        for (int i = 0; i < len; i++)
        {
            crc = UpdateCRC(data[i], crc);
        }

        return crc;
    }
```

```
uint8_t UpdateCRC(uint8_t dataByte, uint8_t crcPrev)
{
     uint8_t crc = crcPrev;
     crc ^= dataByte;
     for (int bitInd = 0; bitInd < 8; bitInd++)
     {
          if ((crc & 0x01) != 0)
          {
               crc >>= 1;
               crc ^= 0x8C;
          }
          else
          {
               crc >>= 1;
          }
     }
     return crc;
}
/**
 * \brief Checksum IEC 60870-5-101
 *
 * \param  *data - buffer data
 *             len  - length bufer
 *             start - start bit for calculate
 * \return  checksum
 */
uint16_t calculate_checksum(const uint8_t *data, size_t len,
uint8_t start) {
     uint16_t sum = 0;
    for (size_t i = start; i < len - 2; ++i) {
        sum += data[i];
    }
    if (sum > 255)
     return ((sum & 0xFF) + ((sum >> 8) & 0xFF) - 1);
    else return sum;
}
```

```
/* Write log */
/*
int _write(int file, char *ptr, int len) {
     (void) file;
     int DataIdx;

     for (DataIdx = 0; DataIdx < len; DataIdx++) {
          ITM_SendChar(*ptr++);
     }
     return len;

}
*/
/* USER CODE END 4 */

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM1) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */
  if (htim == &htim4) {
```

```
            if (timeSec > 0xFFFFFFFF)
                 timeSec=0;
            else
                 timeSec++;
       }
       /* USER CODE END Callback 1 */
     }

     /**
       * @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();
       while (1)
       {
       }
       /* 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
       */
```