Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації
(повна назва)
Кафедра Радіотехнологій інформаційно-комунікаційних систем
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти **перший (бакалаврський)**

## РОЗРОБКА ДОДАТКУ ДЛЯ ЗЧИТУВАННЯ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ З МК ЧЕРЕЗ COM ПОРТ
(тема)

Виконав:
студент **IV** курсу, групи **ІТІР-20-1**
**Гончар Б. В.**
(прізвище, ініціали)
Спеціальність **126 Інформаційні системи та технології**
(код і повна назва спеціальності)
Тип програми **освітньо-професійна**
Освітня програма **Інформаційні технології інтернету речей**
( повна назва освітньої програми)
Керівник **Старший викладач Ганшин Д.Г.**
(посада, прізвище, ініціали)

Допускається до захисту
В.о.зав. кафедри РТІКС _____ **Зарудний О.А.**
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій і технічного захисту інформації

Кафедра Радіотехнологій інформаційно-комунікаційних систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 Інформаційні системи та технології
(код і повна назва)

Тип програми Освітньо-професійна

Освітня програма Інформаційні технології інтернету речей
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____2024 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові **ГОНЧАРУ БОГДАНУ ВІТАЛІЙОВИЧУ**
(прізвище, ім'я, по батькові)

1. Тема роботи **РОЗРОБКА ДОДАТКУ ДЛЯ ЗЧИТУВАННЯ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ З МК ЧЕРЕЗ COM ПОРТ**

затверджена наказом по університету від **27.05 2024 р. № 500 Ст**

2. Термін подання студентом роботи до екзаменаційної комісії **10 червня 2024 р.**

3. Вихідні дані до роботи

3.1 Провести огляд та аналіз аналогічних систем

3.2 Розробити структурні схеми пристроїв

3.3 Обрати елементну базу для реалізації пристрою

3.4 Розробити програмне забезпечення

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, 1. Огляд com портів, 2. Порівняльний аналіз фреймворків для розробки графічного інтерфейсу, 3. Опис програмно-апаратного забезпечення, 4. Опис розробленого додатку, Висновки, Перелік джерел посилання, Додатки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) (п.5 включається до завдання за рішенням випускової кафедри)_____
Слайди комп'ютерної презентації, мікроконтролер ESP32, Arduino Uno, Qt Creator, UML діаграма класів, рисунки схеми, скріншоти додатку

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| Основна частина | Ст. викл. Ганшин Д. Г. | | |
| | | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Вступ | 13.05-16.05.2024 | виконано |
| 2 | Аналіз предметної області | 17.05-21.05.2024 | виконано |
| 3 | Розробка структурної схеми пристроїв | 22.05-25.05.2024 | виконано |
| 4 | Вибір та обґрунтування елементної бази | 26.05-03.06.2024 | виконано |
| 5 | Розробка програмного забезпечення | 04.06-09.06.2024 | виконано |
| 6 | Висновки | 10.06.2024 | виконано |
| 7 | Оформлення пояснювальної записки | 11.06.2024 | виконано |
| 8 | Оформлення ілюстрацій | 12.06.2024 | виконано |
| 9 | Представлення роботи на кафедру | 13.06.2024 | виконано |

Дата видачі завдання **06 травня 2024 р.**

Студент _____ Б. В. Гончар
(підпис)

Керівник роботи _____ ст. вик. Д. Г. Ганшин
(підпис)

# РЕФЕРАТ

Кваліфікаційна робота складається з пояснювальної записки, що містить 115 сторінку тексту, 12 рисунків, 4 таблиці, 10 джерел посилань та 3 додатків.

ARDUINO. IDE. UNO. ESP32. ПРОГРАМА. МІКРОКОНТРОЛЕР. ДАТЧИК. ЗЧИТУВАННЯ. ВІДОБРАЖЕННЯ. COM-ПОРТ.

Об’єктом розробки є програмне забезпечення для зчитування та візуалізації даних з мікроконтролерів через COM-порт.

Метод дослідження – описово-аналітичний.

Метою даної кваліфікаційної роботи бакалавра є розробка програмного забезпечення для зчитування та візуалізації даних з мікроконтролерів через COM-порт.

В ході роботи проведено описово-аналітичний аналіз предметної області, визначено вимоги до програмного забезпечення та розроблено програму, що відповідає індивідуальному завданню.

# ABSTRACT

The graduate work consists of an explanatory note, which contains 115 pages of text, 12 pictures, 4 tables, 10 references to sources and 3 additions.

ARDUINO. IDE. UNO. ESP32. PROGRAM. SENSOR. MICROCONTROLLER. READING. VISUALIZING. COM-PORT.

The object of development is software for reading and visualizing data from microcontrollers through the COM port.

The research method is descriptive and analytical.

The purpose of this bachelor’s qualification work is to develop software for reading and visualizing data from microcontrollers through the COM port.

In the course of the work, a descriptive and analytical analysis of the subject area was carried out, the requirements for the software were determined, and a program was developed that corresponds to the individual task.

ЗМІСТ

# ВСТУП

В сучасному світі зростає значення збору, аналізу та використання даних в різних галузях життя. Особливо актуальною стає задача моніторингу та управління різноманітними процесами, що вимагають зчитування даних з мікроконтролерів. Мікроконтролери є основою багатьох пристроїв та систем, які використовуються в промисловості, автомобільній техніці, побутовій електроніці та інших сферах. Для ефективного використання цих пристроїв та забезпечення надійності їх роботи необхідне програмне забезпечення, що забезпечить зчитування та візуалізацію даних з мікроконтролерів.

Метою даної дипломної роботи є розробка програмного забезпечення для зчитування та візуалізації даних з мікроконтролерів через COM-порт. Дана робота має на меті створення інструменту, який буде спрощувати процес зчитування та аналізу даних з мікроконтролерів, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс для користувача.

Для досягнення цієї мети, в рамках дипломної роботи будуть проведені огляд існуючих систем та програмних засобів для зчитування та візуалізації даних з мікроконтролерів через COM-порт, визначені вимоги до програмного забезпечення, розроблено програму, яка відповідає поставленим вимогам, та проведено тестування розробленого програмного забезпечення для підтвердження його працездатності та ефективності.

# 1 ОГЛЯД COM ПОРТІВ

## 1.1 Що таке COM порт, його переваги та недоліки

Послідовний порт – це тип комп’ютерного інтерфейсу, який передає дані по одному біту за раз, використовуючи асинхронний протокол. Цей термін часто асоціюється з RS-232 портами на персональних комп’ютерах, а також з різними послідовними портами у вбудованих системах. Однією з головних особливостей послідовних портів є їх двонаправленість, що дозволяє їм водночас надсилати та отримувати дані. Хоча передача даних по бітам може здатися менш ефективною порівняно з іншими методами, вона має важливі переваги, такі як можливість використання дешевих кабелів та компактних роз’ємів.

У сучасних ПК програми зазвичай доступаються до послідовних портів через інтерфейс COM-портів. Для розробників, які використовують бібліотеку класів Microsoft .NET Framework, доступ до COM-портів можна отримати за допомогою класу SerialPort. Це дозволяє програмам легко комунікувати з послідовними портами, як з фізичними, так і з віртуальними.

Сучасні USB пристрої можуть емулювати послідовні порти, функціонуючи як віртуальні COM-порти, що робить їх доступними для програм настільки ж легко, як і фізичні порти. Також деякі пристрої, що працюють через Ethernet або Wi-Fi, можуть виступати в ролі послідовних серверів, дозволяючи програмам отримувати доступ до послідовних портів через мережу. Це значно розширює можливості використання послідовних портів, особливо в промислових та вбудованих системах.

Мікроконтролери, які є ключовими елементами вбудованих систем, також активно використовують послідовні порти для зв’язку як з іншими пристроями, так і з персональними комп’ютерами. Компілятори для мікроконтролерів зазвичай включають бібліотеки, що полегшують роботу з послідовними портами, надаючи готові функції для їхнього програмування.

Це спрощує розробку програмного забезпечення для взаємодії з внутрішніми та зовнішніми системами через послідовні інтерфейси.

Послідовні порти є ідеальним рішенням для багатьох випадків комунікації між вбудованими системами або між вбудованими системами та ПК. Вони можуть бути особливо корисні, коли потрібно встановити зв’язок на довгі відстані за допомогою довгих кабелів або забезпечити базову мережу між ПК та вбудованими системами.

Наприклад, послідовні порти можуть бути використані для зв’язку між великою кількістю датчиків або пристроїв та центральним контролером у вбудованих системах. Вони також можуть стати частиною простої мережі між ПК та різними електронними пристроями.

Деякі вбудовані системи можуть мати послідовний порт, який прихований від звичайних користувачів, але доступний для технічних спеціалістів для налагодження та діагностики системи. Це може бути корисно для вирішення проблем або відлагодження програмного забезпечення без необхідності втручання в основні функції системи.

Ось декілька ключових переваг асинхронних послідовних портів та програмування COM-портів, які підкреслюють їхню універсальність та важливість:

– універсальність використання: Послідовні порти можуть обмінюватися майже будь-яким видом даних. Вони ідеально підходять для різноманітних застосувань, від зчитування даних з датчиків до керування двигунами чи дисплеями. Ця гнучкість робить їх незамінними в багатьох технічних сферах;

– доступність обладнання: Обладнання для послідовних портів, як правило, є дешевим і доступним. Навіть ПК без вбудованих послідовних портів можуть легко інтегрувати їх через USB/послідовні адаптери. Також більшість сучасних мікроконтролерів має вбудовані послідовні порти, що спрощує їх інтеграцію в різні проекти;

– простота в обробці даних: Послідовні порти не ставлять особливих вимог до формату даних, що передаються. Вони лише вимагають наявності початкових, стопових і додаткових бітів парності. Це відрізняється від складніших протоколів, як USB чи Ethernet, що вимагають специфічної реалізації передачі даних;

– низька вартість кабелів: Багато послідовних з'єднань можуть використовувати недорогі неекрановані кабелі, які містять лише 3–9 проводів. Це робить послідовні порти економічно ефективними для широкого спектру застосувань;

– підтримка в операційних системах: Операційні системи, як Windows, мають вбудовані драйвери для доступу до COM-портів, а мови програмування часто пропонують готові класи та бібліотеки для легкого зв'язку через COM-порти;

– бездротові можливості: Сучасні бездротові технології дозволяють передавати послідовні дані без потреби в фізичних кабелях, що значно розширює горизонти використання послідовних портів;

– гнучкість інтерфейсів: Пристрій з USB, доступ до якого здійснюється як до COM-порту, не обов'язково використовує асинхронний послідовний інтерфейс, а може мати паралельний чи інший тип інтерфейсу, залежно від вимог програми.

Хоча асинхронні послідовні інтерфейси мають багато переваг, вони не підходять для всіх ситуацій, і є певні обмеження, які слід враховувати:

– конвертація даних: Комп'ютери, що знаходяться на обох кінцях з'єднання, мають перетворювати дані з послідовного формату в паралельний, який використовується процесором. На щастя, цей процес зазвичай автоматизовано здійснюється за допомогою апаратного забезпечення;

– відсутність гарантій реального часу: Операційна система Windows не може гарантувати виконання операцій з послідовними портами в режимі реального часу. Це означає, що іноді передача або отримання даних може

відкладатися через виконання операційною системою інших задач. Хоча такі затримки зазвичай невеликі та властиві багатьом інтерфейсам у Windows, вбудовані системи часто краще справляються з контролем часу виконання послідовного зв’язку.

## 1.2 Компоненти системи

Для ефективного зв’язку через послідовні порти необхідні три основні компоненти: комп’ютери з послідовними портами, кабель або бездротовий засіб для забезпечення фізичного з’єднання між портами та програмне забезпечення для управління цим зв’язком. Використання послідовних портів зручне для широкого спектру комп’ютерних систем, від простих мікроконтролерів до повноцінних персональних комп’ютерів, навіть якщо у них відсутні вбудовані послідовні порти.

Пристрої з асинхронними послідовними портами зазвичай включають спеціалізований апаратний компонент, відомий як універсальний асинхронний приймач-передавач (UART). Цей компонент відіграє ключову роль у конвертації між паралельними та послідовними формами даних, а також управлінні іншими аспектами послідовного зв’язку.

Ось кілька прикладів застосування UART у різних пристроях:

- мікроконтролери: Багато мікроконтролерів мають один або декілька вбудованих UART для послідовного зв’язку. У разі відсутності апаратного UART, функції UART можуть бути емульовані програмним забезпеченням за допомогою внутрішнього таймера мікроконтролера;
- зовнішні мікросхеми UART: Ці мікросхеми можуть бути підключені до мікроконтролерів або інших процесорів для додавання послідовного зв’язку;
- RS-232 порти: Раніше ці порти були стандартом для ПК та інших пристроїв до поширення USB. Комп’ютери з вільними слотами розширення можуть легко додати RS-232 порти через плати розширення;

– PC Card (PCMCIA) порти: Ноутбуки та інші пристрої з вільними слотами PC Card можуть використовувати ці карти для додавання послідовних RS-232 портів;

– USB конвертери: Вони дозволяють підключати послідовні порти до ПК через USB, пропонуючи гнучкість в драйверах та підтримці різних типів послідовних інтерфейсів;

– порти Ethernet та Wi-Fi: Сервери послідовних портів на цих платформах дозволяють доступ до послідовних портів через мережу, розширюючи можливості інтеграції та віддаленого доступу.

Комунікація через послідовні порти не обмежується пристроями одного типу чи класу. Наприклад, маленькі мікроконтролери можуть ефективно обмінюватися даними з сучасними настільними ПК, доки в обох пристроях використовуються сумісні інтерфейси та протоколи зв’язку. В контексті цієї книги, приклади стосуються великої когорти комп’ютерів, що розвивались з IBM PC, включно з настільними комп’ютерами та лептопами. Також існують інші типи комп’ютерів, включно з тими, що мають вбудовані послідовні порти або підключаються через конвертери та карти розширення.

Вбудовані системи – це спеціалізовані комп’ютерні системи, створені для виконання конкретних функцій чи задач. Ці системи часто інтегровані безпосередньо в пристрої, якими вони управляють. Наприклад, модеми – це тип вбудованих систем, які займаються передачею даних через телефонні мережі. Вбудовані системи можуть бути унікальними або масово вироблятися для широкого застосування, зокрема для моніторингу та управління процесами.

Значна частина вбудованих систем використовує мікроконтролери, які поєднують в собі центральний процесор з апаратними засобами введення/виведення, включно з такими елементами як UART для забезпечення послідовного зв’язку. Мікроконтролери класифікуються за шириною шини даних: 8-бітні мікроконтролери використовують 8-бітну шину даних і є вкрай

популярними в управлінських додатках завдяки своїй економічності та достатньому функціоналу для багатьох завдань. Існують також мікроконтролери з 4-, 16-, 32- і навіть 64-бітними шинами, які пропонують різні комбінації можливостей, включаючи як асинхронні, так і синхронні послідовні порти, контролери USB, різноманітні типи і обсяги пам’яті для зберігання програм і даних, а також підтримку різних режимів енергозбереження.

Фізичне підключення між комп’ютерами включає в себе не тільки дроти чи інші середовища для передачі даних, а й конектори та інші компоненти, які сполучають ці середовища з комп’ютерами. Наприклад, з’єднання RS-232 може використовувати різноманітні кабелі, потребуючи один кабель для кожного сигналу плюс один для загального заземлення. У мережах RS-485 типово використовуються кабелі з витою парою, де одна пара призначена для кожного диференціального сигналу. Інші варіанти послідовного зв’язку включають волоконно-оптичні кабелі, які передають дані у вигляді світлових сигналів, і бездротові технології, що дозволяють передавати дані через повітря за допомогою радіо чи інфрачервоних сигналів. Для забезпечення надійності, проводові з’єднання часто вимагають спільного заземлення, яке зазвичай здійснюється через заземлюючий провід у кабелі.

Процес послідовного зв’язку вимагає від комп’ютерів виконання кількох ключових завдань:

- вони повинні виявляти і обробляти дані, що надходять;
- при необхідності, комп’ютери повинні ініціювати передачу даних;
- комп’ютери також відповідають за виконання інших задач, пов’язаних із їх основними функціями.

У разі здійснення зв’язку через послідовну мережу, кожен комп’ютер в мережі повинен ігнорувати сигнали, призначені для інших пристроїв, та дотримуватися мережевих протоколів для правильної адресації переданих даних відповідним комп’ютерам. Програмне забезпечення, що керує цими

процесами, часто підтримується апаратними компонентами для покращення ефективності та надійності.

Програмування для послідовних інтерфейсів може використовувати різноманітні мови програмування, і не обов'язково мова на одному кінці зв'язку має співпадати з мовою на іншому кінці. Основна вимога полягає в тому, що всі системи мають узгодити формати даних, які вони обмінюють. Програмне забезпечення мікроконтролерів може безпосередньо взаємодіяти з регістрами UART для налаштування параметрів зв'язку або використовувати розроблені бібліотечні функції для спрощення процесу. Додатки для ПК, у свою чергу, зазвичай користуються більш високорівневими функціями для доступу до портів.

Протокол – це сукупність правил, які визначають, як комп'ютери мають взаємодіяти під час зв'язку. Щодо послідовного зв'язку, протокол встановлює базові правила для передачі бітів, включно з тим, коли комп'ютери мають право передавати дані, якою швидкістю це має відбуватися, та в якому порядку біти повинні бути відправлені. Зазвичай, UART (універсальний асинхронний приймач-передавач) відповідає за обробку деталей передачі окремих бітів та зберігання прийнятих бітів на послідовному порту. Коли два комп'ютери мають намір взаємодіяти, вони мають домовитися, чи можуть вони відправляти дані одночасно або чи слід їм діяти по черзі. Більшість провідних з'єднань між двома комп'ютерами є повнодуплексними, дозволяючи обом машинам передавати дані одночасно. Натомість багато бездротових з'єднань є напівдуплексними, що вимагає чергування передачі даних. Симплексні зв'язки обмежені односторонньою передачею.

У мережах із трьома або більше комп'ютерами, які діляться одним шляхом передачі даних, необхідно використовувати протокол, який визначає, коли кожен комп'ютер може передавати дані. Протокол зв'язку також може включати використання ліній стану та керування, які допомагають передавачу знати, коли у нього є дані для передачі, а приймачу - коли він готовий

приймати нові дані. Цей процес взаємодії відомий як керування потоком. Апаратне керування потоком використовує спеціалізовані лінії для цих сигналів, тоді як програмне керування потоком передає аналогічну інформацію, використовуючи визначені коди.

Додаткові лінії стану та керування можуть надавати іншу важливу інформацію, наприклад, про наявність несучої частоти або телефонного дзвінка. В послідовних мережах, де активний тільки один передавач за раз, кожен комп’ютер має спеціальну лінію, яка вмикає або вимикає передавач за потреби.

Послідовний зв’язок часто включає обмін повідомленнями, які складаються з блоків даних у визначених форматах. Протокол повідомлень встановлює, які типи даних містить кожне повідомлення і як організована інформація всередині нього. Мережі, де є кілька комп’ютерів, зазвичай призначають кожному комп’ютеру унікальну адресу, яка включається у кожне повідомлення, щоб вказати призначеного одержувача. Наприклад, дуже просте повідомлення може містити два байти: один для ідентифікації приймача та один з актуальними даними. Щоб одержувач міг розрізнити початок і кінець повідомлення, повідомлення можуть містити спеціальні коди або заголовок з інформацією про довжину повідомлення. Для перевірки на помилки повідомлення можуть також включати додаткові байти, які допомагають одержувачу визначити точність переданої інформації.

## 1.3 Надсилання послідовних даних та бітів

Вивід послідовного порту, який виконує роль передавача, передає дані біт за бітом до входу іншого послідовного порту, що служить приймачем, зазвичай розташованого на іншому комп’ютері. Зазвичай для кожного напрямку передачі даних існує окремий кабель. Однак, деякі послідовні

інтерфейси використовують один спільний канал для передачі даних у обох напрямках, де передавачі чергуються між собою.

Послідовний зв’язок базується на асинхронному протоколі. Це означає, що такий інтерфейс не використовує спеціальну лінію для синхронізації. Натомість, кожен комп’ютер використовує свій власний годинник для таймінгу, і обидва кінці зв’язку повинні бути узгоджені з великою точністю. Стартовий біт, що передається першим, допомагає синхронізувати годинники передавача і приймача. Водночас, синхронний протокол включає лінію тактового сигналу, яка зазвичай контролюється одним із комп’ютерів, і всі передані біти синхронізуються з цим тактовим сигналом. Кожен біт є активним у визначений момент часу після зміни стану тактового сигналу, що залежить від конкретного протоколу. До прикладів синхронних послідовних інтерфейсів належать I2C, SPI та Microwire, які використовуються для взаємодії між різними пристроями в електроніці.

UART, який відомий як універсальний асинхронний передавач/приймач, працює, відправляючи дані частинами, відомими як «слова». Кожне таке слово містить початковий біт, кілька бітів даних, можливо біт парності для перевірки помилок, а також один чи декілька стоп-бітів, що вказують на завершення передачі даного слова. Найбільш поширеним форматом передачі є 8-N-1, де передається один стартовий біт, вісім бітів інформації та один стоп-біт без додаткового біта парності.

У випадках, коли потрібна перевірка помилок, використовується біт парності. Наприклад, у форматі 7-E-1 передаються сім бітів даних з одним бітом парності, що дозволяє виявити деякі помилки в переданих даних. Якщо система використовує парну парність, загальна кількість одиниць у переданих бітах даних та біті парності повинна бути парною, і навпаки для непарної парності.

Окремі біти, передані через UART, іноді називають символами, і вони можуть репрезентувати текстові символи чи інші дані. Іноді для забезпечення

додаткового часу для приймача, UART дозволяє використання подовжених стоп-бітів, що може тривати до 1,5 або 2 бітів. Ця особливість була особливо корисною в минулому для механічних телетайпів, яким потрібен був час для переходу в стан очікування між символами.

Таким чином, UART надає гнучкість і точність у передачі даних, але також включає складність, особливо коли використовується для забезпечення надійності та точності в середовищах, де можливі помилки в передачі даних.

Швидкість передачі даних означає кількість біт, які можуть бути передані чи отримані за одну секунду, і зазвичай це вимірюється у бітах на секунду (біт/с). Це основний показник, який вказує на те, наскільки швидко дані можуть переміщуватися між пристроями в системі зв'язку. В базових цифрових системах передачі, які ми розглядаємо у цій книзі, кожен тактовий імпульс може представляти один біт даних, тому швидкість передачі даних дорівнює кількості біт, що передаються за секунду.

З іншого боку, у високошвидкісних телефонних модемах використовуються більш складні методи, такі як фазовий зсув, які дозволяють кодувати кілька бітів даних в одному такті. Це зменшує загальну швидкість передачі даних порівняно з кількістю біт, що фактично передаються. Таким чином, хоча термін "швидкість передачі даних" часто використовується як синонім швидкості передачі, вони можуть не завжди бути ідентичними залежно від контексту і методології передачі.

Кількість символів, що передаються за секунду, визначається шляхом поділу загальної швидкості передачі даних на кількість біт, які входять до одного «слова» або кадру даних. Наприклад, у випадку формату 8-N-1, де кожне повідомлення містить один стартовий біт, вісім біт даних і один стоповий біт, загалом 10 біт на символ, швидкість передачі одного байту даних буде на 10% нижчою від загальної швидкості біт/с. Тож, якщо канал має швидкість 9600 біт/с, використання формату 8-N-1 дозволить передавати приблизно 960 байтів даних за секунду.

У персональних комп’ютерах та багатьох мікроконтролерах завдання обробки нюансів передачі та прийому послідовних даних зазвичай покладене на UART, який є спеціалізованим апаратним компонентом. Операційні системи комп’ютерів і мови програмування надають додаткову підтримку, забезпечуючи драйвери та бібліотеки, які спрощують доступ до послідовних портів для розробників. Це дозволяє програмістам працювати з послідовними портами, не занурюючись глибоко у технічні деталі UART.

Для взаємодії з портом програма встановлює необхідні параметри зв’язку, такі як швидкість передачі даних, і потім відкриває доступ до потрібного порту. Коли програма хоче надіслати дані, вона поміщає байт у вихідний буфер відповідного порту. UART автоматично обробляє передачу цих даних, відправляючи їх біт за бітом і додаючи стартові, стопові біти та біти парності, якщо це необхідно. Аналогічним чином, UART збирає вхідні дані в буфер прийому. Коли байт надходить повністю, UART може сповістити програму за допомогою переривання, або програма може регулярно перевіряти порт, щоб дізнатися, чи є нові дані для читання.

Система драйверів COM-порту на ПК забезпечує додаткове управління, використовуючи програмні буфери для вирівнювання потоку даних між апаратними буферами UART і програмою. Це забезпечує більш плавне та надійне з’єднання.

В деяких випадках, коли вбудованого UART не достатньо або коли мікроконтролеру потрібно більше послідовних портів, ніж передбачено його апаратними засобами, можливо використання зовнішнього UART або реалізація послідовних функцій в програмному забезпеченні. Наприклад, модуль Basic Stamp від Parallax Inc. включає в себе програмно реалізований UART. Інші мікроконтролери містять USART, який підтримує як синхронний, так і асинхронний режими зв’язку, надаючи гнучкість для обрання найкращого режиму в залежності від конкретної задачі.

В багатьох програмах для роботи з послідовними портами на ПК існує інтерфейс, де користувач може вибирати швидкість передачі даних через випадаюче меню. Найзручніше, коли комп’ютери автоматично синхронізують швидкість передачі даних, адаптуючись один до одного без додаткових налаштувань.

Щоб досягти цього автоматичного зіставлення швидкостей, комп’ютер-відправник починає процес, відправляючи певний байт даних кілька разів. Цей байт може мати будь-яке значення, яке дозволяє визначити кінцевий біт даних як нуль. Комп’ютер-одержувач починає зчитування на своїй максимальній можливій швидкості. Якщо він отримує декілька байтів замість одного, це означає, що швидкість занадто висока і не відповідає швидкості відправника.

Відправник тоді зменшує швидкість і повторює спробу. Процес продовжується доки одержувач не отримає лише один коректний байт, що вказує на збіг швидкостей. Для подальшої перевірки одержувач може переслати цей байт назад відправнику як підтвердження, що швидкість вірна, і тоді обидва комп’ютери можуть розпочати стандартне спілкування.

Існує ще один метод автоматичного визначення швидкості, де комп’ютер-одержувач вимірює тривалість отриманого біту. Якщо відправник передає конкретне значення, одержувач може виміряти час, протягом якого він отримує цей біт, і відповідно налаштувати свою швидкість передачі, щоб вона була точно синхронізована з відправником.

Ці методи спрощують процес налаштування та дозволяють обладнанню більш гнучко взаємодіяти без зайвих ручних налаштувань, знижуючи ризик помилок і покращуючи загальну надійність системи передачі даних.

Комп’ютери часто мають кілька COM-портів, тому вибір правильного порту для спілкування може бути важливим завданням. Зазвичай, програми надають користувачам зручний список, з якого вони можуть вибрати потрібний порт. Щоб спростити процес, програма може автоматично запам’ятати останній використаний порт та встановити його як стандартний

наступного разу, коли користувач запустить програму, за умови, що цей порт доступний. Крім того, програма може зберігати інші параметри налаштувань порту, такі як швидкість передачі даних, що є дуже зручним для повторного використання налаштувань без необхідності їх повторного введення.

Для додаткової зручності, програма може включати функцію, яка автоматично шукає правильний порт, відправляючи спеціальне повідомлення на кожен доступний COM-порт. Комп’ютер, який приймає це повідомлення, відповідає, підтверджуючи, що зв’язок встановлено через правильний порт. Це може виявитися надзвичайно корисним, коли є потреба автоматизувати процес встановлення зв’язку. Однак, такий метод слід застосовувати обережно, оскільки непередбачені дані, відправлені на інші пристрої, які використовують COM-порти, можуть спричинити небажані збої або помилки. Тому важливо заздалегідь переконатись, що відправлення даних не завадить іншим пристроям, підключеним до системи.

## 1.4 Формати даних

У передачі даних через послідовний порт різноманітна інформація може бути представлена у вигляді команд, даних з датчиків, інформації про стан пристроїв, кодів помилок, конфігураційних даних або навіть текстових файлів або виконуваного коду. Незалежно від типу інформації, її все одно можна розглядати як послідовність байтів або іншу кількість бітів. У широкому розумінні, байт є 8-бітовим значенням, і в цьому контексті будемо вважати, що він складається з 8 бітів, але важливо зауважити, що іноді термін "байт" використовується для будь-якої кількості бітів, що обробляються як одиниця інформації.

У випадку двійкових даних кожен байт може приймати значення від 00h до FFh, і в цьому випадку програмне забезпечення не має жодних припущень щодо значення переданої інформації. Такий підхід надає максимальну

гнучкість у використанні переданих даних, а програмне забезпечення більш високого рівня може інтерпретувати ці дані у відповідний спосіб.

Кожен біт у байті можна проіндексувати від 0 до 7, причому кожен біт представляє значення (0 або 1), помножене на 2 в ступені, що відповідає його позиції. Наприклад, байт 11111111b можна представити як FFh або 255, а байт 00010001b - як 11h або 17. У асинхронних зв'язках найменш значущий біт (LSb) приходить першим, що може призводити до певних особливостей під час аналізу даних на осцилографі чи логічному аналізаторі. Пам'ятайте про це при інтерпретації даних у відповідному порядку.

І, нарешті, важливо врахувати, що послідовний порт може передавати дані, які мають різну кількість бітів, наприклад, 16 або 32, розділяючи їх на байти та передаючи їх послідовно. Для успішного обміну багатобайтовими значеннями обидва комп'ютери повинні мати однаковий порядок передачі байтів, щоб правильно інтерпретувати дані.

Деякі програми розглядають дані через призму тексту, де кожний символ представлений своїм унікальним кодом. Це робить роботу з даними значно простішою для програмістів. У більшості мов програмування текст може зберігатися у двох формах: як рядки, які складаються з одного або кількох символів, і як масиви, де кожен елемент є окремим символом. Наприклад, код може передбачати відправку рядка «hello», і тоді компілятор або інтерпретатор перетворить цей рядок на послідовність окремих символів для передачі. З іншого боку, код може зажадати відправлення масиву, де кожен елемент містить код символу.

На стороні прийому, програмне забезпечення може вибирати зберігати отримані символи у вигляді рядків або масивів символів. Якщо програми обмінюються лише простим текстом на англійській мові, кодування зазвичай не викликає проблем. Але коли мова йде про спеціальні символи або символи з інших мовних систем, комп'ютери повинні домовитися про спосіб їхнього кодування.

Сучасні програмні системи, такі як .NET Framework, використовують універсальні методи кодування, розроблені на основі Unicode Standard, який публікується Unicode Consortium. Unicode підтримує більше мільйона символів у широкому діапазоні алфавітів та інших систем письма, а також включає знаки пунктуації, математичні символи, геометричні фігури та інше. В Unicode кожен символ ідентифікується за допомогою кодової точки, унікального числового значення. Кодові точки в Unicode мають позначення у форматі U+code_point, де code_point – це шістнадцяткове число. Наприклад, символ "A" має кодову точку U+0041, і кодові точки варіюються від U+0000 до U+10FFFF. Кожен символ має строго одну кодову точку.

Програмне забезпечення може інтерпретувати надіслані через послідовний порт дані як текст, і кожний символ у такому тексті представлений унікальним кодом. Робота з текстовими даними є досить зручною, оскільки більшість мов програмування дозволяють маніпулювати текстом як рядками, що можуть містити декілька символів, або як масивами окремих символів. Наприклад, програма може вказувати на передачу рядка «hello», і в цьому випадку компілятор або інтерпретатор перетворить цей рядок у послідовність окремих символів для передачі. Також можливе запитання на передачу масиву, де кожен елемент масиву представляє окремий символ.

На стороні прийому, програмне забезпечення може зберігати отримані символи у вигляді рядків або масивів символів. Це особливо просто, коли обмін відбувається у формі звичайного англійського тексту, де кодування зазвичай не становить проблему. Однак, коли потрібно передати спеціальні символи або символи з інших мов, комп'ютери повинні домовитись про спосіб їх кодування.

На сучасних програмних платформах, таких як .NET Framework, використовуються методи кодування, які базуються на стандарті Unicode, що підтримує більше мільйона різних символів у безлічі мовних систем. Unicode

дозволяє кодувати текст за допомогою щось, що називається «кодовими точками». Кожен символ у Unicode має унікальну кодову точку, яка ідентифікує цей символ. Наприклад, латинська літера «A» має кодову точку U+0041.

Кодові точки використовуються для представлення символів в трьох основних форматах кодування Unicode: UTF-8, UTF-16 та UTF-32, кожен з яких може кодувати будь-який символ в Unicode. UTF-8 є найбільш поширеним, оскільки він дозволяє кодувати символи в один до чотирьох байтів, роблячи його зворотно сумісним з ASCII для англійських символів. Це означає, що текст на англійській мові, кодований в UTF-8, можна читати як ASCII без необхідності декодування.

Кожен символ у UTF-8 може мати від одного до чотирьох байтів, залежно від символу. Наприклад, символ «©» має кодову точку U+00A9 і кодується у UTF-8 як два байти: C2 A9. Кожен з цих байтів є частиною системи, що дозволяє визначити, як символи мають бути правильно прочитані та відображені на екрані або оброблені іншими програмами.

Кодування ANSI, яке часто використовувалось в минулому, розглядало символи за допомогою однобайтових значень на основі стандарту, який Microsoft впровадила як кодову сторінку 1252. На відміну від UTF-8, яке використовує мінімум один байт для кодування символів і може розширитись до чотирьох байтів, ANSI обмежує символи до одного байта, що призводить до обмеження на символи, які можуть бути представлені.

UTF-16 та UTF-32 – це більш новітні методи кодування, які використовують відповідно 16-бітні та 32-бітні блоки для представлення символів. У UTF-16 більшість символів кодуються однією одиницею коду, але деякі символи вимагають використання так званих сурогатних пар, двох блоків коду, для їхнього представлення. Наприклад, латинська літера «A» має код 0041h у UTF-16, а символ «©» представлено як 00A9h. UTF-32, з іншого боку, використовує одну 32-бітну одиницю коду для кожного символу, що

робить його більш простим у використанні, але водночас збільшує обсяг використовуваної пам'яті.

Обидва ці методи кодування можуть бути налаштовані для використання як big-endian (BE), так і little-endian (LE) порядків байтів. Big-endian (великий край) зберігає старший байт першим, тоді як little-endian (малий край) зберігає молодший байт першим. Це важливо для того, щоб програмне забезпечення на різних комп’ютерах правильно інтерпретувало дані.

Для того, щоб забезпечити правильну обробку порядку байтів при передачі даних між системами, використовується позначка порядку байтів (BOM). Якщо перед даними стоїть позначка FEFFh, то порядок байтів не міняється; якщо ж позначка FFFEh, то порядок байтів має бути змінено. Ці позначки не відображаються як символи в тексті і єдине їхнє призначення – це вказівка на порядок байтів у даних.

### 1.5 Запобігання втраті даних

У світі комп’ютерів завжди багато завдань, які вимагають уваги, не лише очікування на дані з послідовного порту. Наприклад, системи збору даних збирають інформацію, яка потім може бути періодично відправлена на інший комп’ютер для аналізу чи зберігання. Інші пристрої можуть слідкувати за обладнанням і періодично відправляти оновлення або отримувати команди, щоб забезпечити правильну роботу системи.

Іноді комп’ютер, який надсилає дані, може спробувати зробити це в момент, коли приймальний комп’ютер зайнятий іншими завданнями. Тому важливо мати надійне апаратне та програмне забезпечення, яке допомагає забезпечити, що всі дані будуть сприйняті коректно та без помилок. Декілька способів, якими це можна досягнути, включають керування потоком даних, буферизацію даних для згладжування піків навантаження, техніки опитування

або переривань для визначення часу прийому даних, а також механізми перевірки наявності помилок та підтвердження отримання повідомлень.

Керування потоком є критичним для забезпечення плавного обміну даними між двома комп’ютерами через послідовний порт, особливо коли буфери приймальника можуть не встигати обробити весь обсяг вхідних даних. У таких випадках, для уникнення втрати інформації, використовується механізм, який називають рукостисканням. Це дозволяє одному пристрою сигналізувати іншому про свою готовність отримувати або перестати отримувати дані.

При реалізації апаратного керування потоком, приймальний комп’ютер використовує спеціальну лінію для сигналізації про свою готовність до прийому даних. Якщо ця лінія встановлена в певний стан (наприклад, позитивна напруга на лінії RS-232 означає готовність), то передавальний комп’ютер може почати відсилати дані. Якщо лінія показує, що приймач не готовий, то передавач почекає.

Таке керування потоком необхідне в обох напрямках, тому кожен напрямок зв’язку має свою лінію: RTS (Request to Send) для вказівки наявності даних до відправки та CTS (Clear to Send) для підтвердження готовності до прийому даних. Ці лінії зазвичай з’єднуються між двома ПК так, що вихід RTS одного з’єднаний з входом CTS іншого.

Мікроконтролери можуть не мати виділених ліній для керування потоком, але можуть використовувати програмне керування потоком, де команди, такі як Xon і Xoff, використовуються для контролю передачі. Команда Xon (зазвичай символ Control+Q) вказує, що приймач готовий отримувати дані, тоді як Xoff (Control+S) говорить передавачу зупинити відправку.

Апаратні та програмні буфери відіграють ключову роль у забезпеченні безперебійного обміну даними між комп’ютерами, особливо в ситуаціях, де виникають затримки в обробці даних. Буфери можуть тимчасово зберігати

дані, що надходять, або дані, що очікують на відправку, що допомагає уникнути втрати інформації і забезпечує плавну передачу даних, навіть коли один з комп’ютерів зайнятий іншими завданнями.

У системах без керування потоком апаратний буфер прийому може стати своєрідним рятівником, зберігаючи вхідні дані, поки програма не буде готова їх опрацювати. Якщо ж система обладнана керуванням потоком і буфером прийому, то передавальний комп’ютер зможе відправляти великі пакети даних, знаючи, що приймальний буфер зможе їх зберегти до того, як буде здійснена обробка.

З іншого боку, буфери передачі дають можливість програмному забезпеченню зберегти дані для відправки і продовжувати роботу над іншими завданнями, поки дані відправляються. Ці буфери можуть бути реалізовані як на апаратному, так і на програмному рівні.

На ПК послідовні порти зазвичай оснащені 16-байтовими апаратними буферами, вбудованими в UART. У режимі прийому такий UART може зберігати до 16 байт даних, перш ніж вони будуть прочитані програмою. В режимі передачі апаратний буфер також може вмістити до 16 байт, що відправляються згідно із заданим протоколом. Окремі моделі UART можуть мати і більші апаратні буфери, особливо у випадках, коли мова йде про віртуальні COM-порти.

Драйвери COM-портів на ПК підтримують програмні буфери, розмір яких може бути налаштований і може бути досить великим, обмежуючись лише системною пам’яттю. Ці буфери забезпечують ефективний обмін даними між програмним і апаратним рівнями.

Мікроконтролери часто обмежені в плані розміру буферів, і деякі з них можуть навіть не мати вбудованих апаратних буферів, що вимагає від системи застосування додаткових стратегій для уникнення втрати даних. Наприклад, може використовуватися програмне керування потоком або спеціальні мікропрограми для керування прийомом та передачею даних.

Програмне забезпечення для роботи з послідовними портами може обробляти різні типи подій, такі як прийом та відправка даних, зміни в контрольних сигналах, помилки в даних, а також ситуації, коли виникає перевищення відведеного часу на передачу чи прийом. Існує два основних способи, якими програмне забезпечення може виявити та реагувати на ці події.

Перший спосіб – це подієве програмування, де програма миттєво реагує на події. Використовуючи цей підхід, програма може переходити безпосередньо до виконання певної дії, коли відбувається подія, наприклад, коли дані надходять на порт. Це ефективно тому, що не потребує зайвих перевірок на наявність діяльності, коли нічого не відбувається. Такий метод використовується у класі SerialPort в .NET Framework, де доступні події такі як DataReceived, PinChanged, та ErrorReceived, які активуються відповідно до конкретних змін у порту.

Другий спосіб виявлення подій – опитування порту. Тут програма регулярно перевіряє стан порту, читаючи його властивості та стани сигналів, щоб дізнатися, чи сталася подія. Цей підхід не використовує апаратні переривання і вимагає від коду частішої перевірки стану, щоб вчасно зареєструвати всі події і запобігти втраті даних. Частота перевірок залежить від обсягу даних, розміру буфера та важливості швидкої реакції на події. Наприклад, система з малим буфером, яка опитує порт раз на секунду, може не встигнути зібрати всі дані, що приходять, що може призвести до їх втрати.

Обидва методи мають свої переваги і недоліки, і вибір між ними залежить від конкретної ситуації та технічних можливостей обладнання. Наприклад, опитування може бути ідеальним для систем, які періодично передають невеликі пакети даних, тоді як подієве програмування підходить для сценаріїв з високим об’ємом або критичними до часу завданнями.

У деяких випадках, коли комп’ютер намагається відправити дані, важливо отримати підтвердження, що дані успішно отримано. Це особливо корисно у мережах, де кілька комп’ютерів використовують спільний канал

зв’язку, і важливо уникнути блокування інших передач даних через невідповідний час активації драйвера. Підтвердження може приймати форму певного сигналу або ж повідомлення від комп’ютера-одержувача, яке підтверджує успішне отримання даних. Якщо комп’ютер-передавач не отримує очікуваної відповіді, він може припустити, що сталася помилка і повторити спробу відправки або виконати інші дії.

Під час передачі до комп’ютера з обмеженим або відсутнім вхідним буфером, комп’ютер-передавач може використовувати метод повного рукостискання. Це означає, що він спочатку надсилає сигнал, щоб попередити про намір передачі даних, а потім, отримавши відповідь, продовжує передачу. Це дозволяє впевнитися, що комп’ютер-одержувач готовий прийняти дані. Побачивши підтвердження від отримувача, передавач може безпечно надсилати решту даних.

Отримувачі даних часто використовують спеціальні методи перевірки для забезпечення точності і цілісності отриманих інформаційних пакетів. Такі методи включають в себе використання бітів парності, контрольних сум, а також метод дублювання даних, про що йшлося раніше в цій главі.

Біт парності дозволяє приймальному пристрою виявляти помилки, які могли статися під час передачі. Наприклад, використання класу SerialPort у .NET або аналогічних класів у інших бібліотеках дозволяє розробникам легко вибрати необхідний тип парності. Програмне забезпечення автоматично обчислює і вставляє правильний біт парності, вказуючи на помилки, коли дані приймаються з неправильною парністю.

Для більш глибокої перевірки, програми можуть використовувати контрольні суми – це спеціальні числові значення, обчислені на основі вмісту блоку даних. Наприклад, проста контрольна сума може бути обчислена шляхом додавання всіх байтів блоку, при цьому використовуються лише молодші байти отриманого результату. Цей метод є стандартом у форматах

даних, як Intel Hex або Motorola S-Record, де контрольні суми забезпечують додаткову перевірку на помилки.

Інший вдосконалений метод – циклічна перевірка надмірності (CRC), яка використовує складніші алгоритми для генерації контрольної суми, яка є більш ефективною у виявленні помилок. Протоколи передачі даних, такі як Kermit або XModem, активно використовують CRC для забезпечення надійності передачі.

Хеш-функції для виявлення повідомлень також надають високий рівень безпеки, використовуючи спільний ключ між відправником і одержувачем для створення і перевірки хеш-значення. Якщо контрольна сума або хеш не відповідає очікуваному значенню, це вказує на помилку, і система може зажадати повторної передачі даних або виконати інші коригувальні дії.

## 1.6 Приклади додатків

В системах, де відбувається передача даних між комп’ютерами, один може виконувати роль збору даних чи відправлення команд, тоді як інші комп’ютери можуть обробляти різні завдання, такі як моніторинг або управління. У деяких випадках, всі комп’ютери в системі можуть бути налаштовані на однакове відправлення та отримання даних. В інших сценаріях, основний обсяг даних може концентруватись на центральному комп’ютері, який збирає інформацію з розташованих на відстані машин.

Прикладом такої системи є мережа метеорологічного спостереження. Десктоп-комп’ютер може служити центральною точкою, керуючи одним чи кількома додатковими комп’ютерами чи вбудованими системами. Цей основний комп’ютер може відправляти інструкції підпорядкованим комп’ютерам про те, як часто збирати дані, які саме дані збирати, і коли їх надсилати назад. Зібрані дані можуть включати параметри такі як температура, атмосферний тиск, вологість, і так далі. Періодично, кожен підпорядкований комп’ютер відправляє зібрані дані на головний комп’ютер,

де вони зберігаються та доступні для подальшого аналізу. Цю конфігурацію можна адаптувати до багатьох інших додатків, які займаються збором даних.

Інші системи фокусуються більше на управлінні зовнішніми пристроями, а не на зборі інформації від них. Наприклад, у вітрині магазину може бути ряд маленьких роботизованих пристроїв, кожен з яких включає в себе мотори, освітлення та інші механізми. Кожен робот функціонує як вбудована система, і головний комп'ютер керує цими пристроями, надсилаючи команди. Хоча роботи можуть також відправляти інформацію про свій стан назад до центрального комп'ютера, головна функція системи — це керування, а не збір даних.

Домашнє управління є ще одним прикладом системи, яка інтегрує як моніторинг, так і управління. Така система стежить за температурою, вологістю, рухом і станами перемикачів, у той же час керуючи опаленням, кондиціонуванням, освітленням та безпекою будинку. У випадку аномалій, система може активувати тривогу [1].

Розроблений додаток має ряд конкурентних переваг, які вирізняють його на тлі існуючих аналогів у сфері цифрових систем управління та моніторингу. Наступні аспекти детально описують унікальні характеристики та переваги додатку:

– інтегрованість: Додаток відрізняється можливістю інтеграції широкого спектру функцій контролю та управління в єдиному інтерфейсі. Це забезпечує користувачам зручний доступ до всіх необхідних інструментів, ефективно знижуючи часові витрати на перемикання між різними програмами;

– користувацький досвід: Програмне забезпечення має інтуїтивно зрозумілий, легко адаптований інтерфейс, що робить процес взаємодії з системою приємним і легким для кінцевого користувача;

– масштабованість: Додаток спроектовано з урахуванням потреби в масштабуванні, дозволяючи легко розширювати функціональність та обслуговувати збільшену кількість пристроїв без втрати продуктивності;

– адаптивність: Завдяки модульній архітектурі додаток може бути легко налаштований під специфічні потреби користувача або вимоги певної галузі.

Ці переваги свідчать про високий потенціал додатку у поліпшенні ефективності та продуктивності систем управління та моніторингу, забезпечуючи водночас легкість у використанні та налаштуванні під конкретні вимоги та задачі.

# 2 ПОРІВНЯЛЬНИЙ АНАЛІЗ ФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ ГРАФІЧНОГО ІНТЕРФЕЙСУ

## 2.1 Опис Qt

Багато бізнес-власників зараз розглядають Qt для розробки інтерфейсів користувача та використовують його як ключову технологію у своєму програмному забезпеченні. Qt – це фреймворк, написаний на мові програмування C++, який пропонує принцип WOCA (Write Once, Compile Anywhere), що означає, що програми, розроблені з використанням Qt, можуть компілюватися для різних операційних систем. В основному, він використовується для створення програм та графічних інтерфейсів користувача, які працюють на різних платформах.

Початкове вікно середовища розробки Qt Creator зображено на рисунку 2.1.

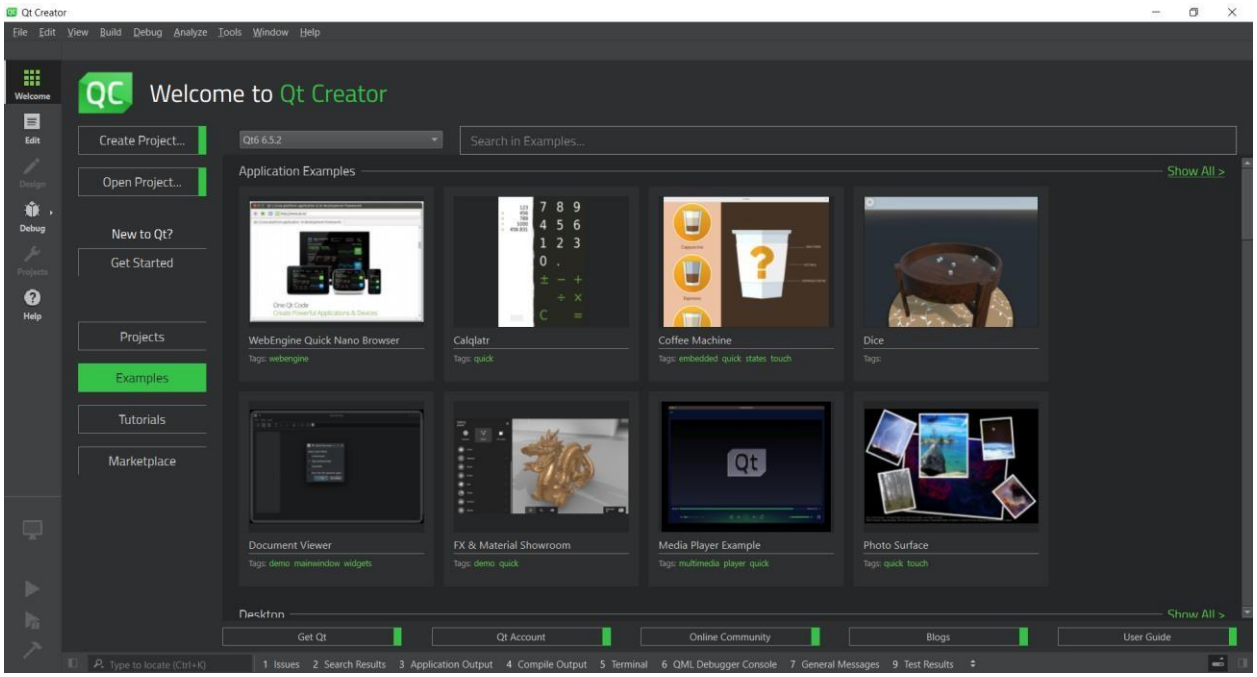


Рисунок 2.1 – Qt Creator

Розробка програм з нативним графічним інтерфейсом стає значно простішою завдяки фреймворку Qt. Він включає у себе низку потужних інструментів, таких як Qt Creator, Qt Quick і Qt Design Studio, які допомагають

ефективно співпрацювати і прискорюють процес створення проектів. Завдяки Qt, ви можете швидко реалізовувати ідеї з меншою кількістю ітерацій і більшою продуктивністю.

Qt також пропонує Qt Modeling Language (QML) – спеціальну мову для розробки програм, орієнтованих на інтерфейс користувача. Ця мова сумісна з багатьма популярними мовами програмування, такими як Java, Python, Go, PHP і Ruby, що робить її вельми універсальною.

Компоненти Qt полегшують процес створення програм, надаючи широкі можливості для управління інтерфейсом користувача, доступу до мережі, роботи з веб-контентом і багато іншого. Ви можете використовувати типи QML для створення визначених інтерфейсів у вашому документі QML, що дозволяє легко повторно використовувати компоненти в інших частинах вашого проекту.

Qt фреймворк надає багатий набір універсальних доповнень, які значно спрощують розробку комплексних мультиплатформних рішень. Серед таких доповнень:

- XML: Цей модуль дозволяє легко створювати та маніпулювати документами XML;
- network: Забезпечує інструменти для з'єднання з серверами та реалізації мережевих протоколів, розширюючи мережеві можливості ваших застосунків;
- SQL: Сприяє інтеграції вашого коду Qt з SQL базами даних, забезпечуючи потужні засоби для роботи з даними;
- multimedia: Підтримує відтвореннята запис аудіо та відео, розширюючи медіа можливості додатків;
- widgets: Дозволяє створювати різноманітні елементи інтерфейсу користувача, що вдосконалює взаємодію з користувачем;
- quick layouts: Надає зручні інструменти для організації елементів інтерфейсу, спрощуючи розробку складних компонувань.

Ці доповнення роблять Qt чудовим інструментом для створення рішень, які задовольняють широкий спектр бізнес- та технічних потреб. Завдяки своїй платформно-незалежній природі, Qt використовується для розробки графічних інтерфейсів користувача та крос-платформних додатків, які ефективно функціонують на всіх основних операційних системах – від настільних та мобільних до вбудованих систем. Така універсальність робить Qt ідеальним вибором для компаній, що прагнуть до економії на розробці нативних рішень для кожної платформи. Додатки, створені на Qt, не тільки виглядають нативно на кожній платформі, а й легко інтегруються з іншими технологіями.

Qt є неймовірно гнучким фреймворком, що дозволяє не тільки розробляти інтерфейси користувача, але й реалізовувати широкий спектр інших функцій. Ось детальніше про ключові можливості Qt:

– мультимедіа: Qt має розгалужену бібліотеку для розробки мультимедійних додатків. За допомогою модуля Qt Multimedia розробники можуть легко імплементувати функції відтворення, запису та потокової передачі аудіо та відео. Цей модуль надає як QML типи, так і C++ класи, які спрощують роботу з медійним контентом;

– робота з базами даних: Qt підтримує широкий діапазон SQL та NoSQL баз даних через свої SQL плагіни. Розробники можуть використовувати такі драйвери, як SQLite, MySQL, Oracle та інші, для ефективної взаємодії з базами даних через Qt SQL;

– мережеві можливості: Qt забезпечує комплексні інструменти для створення мережевих додатків. Модуль Qt Network дозволяє легко працювати з TCP/IP, UDP, HTTP, та WebSockets, надаючи необхідні класи для реалізації мережевого зв'язку та обробки мережевих подій;

– розробка веб-контенту: Завдяки використанню WebKit, Qt дозволяє створювати розширені веб-додатки, використовуючи HTML5, CSS, і

JavaScript. Розробники можуть інтегрувати веб-технології з C++, QML та JavaScript для створення багатих веб-інтерфейсів;

– 2D і 3D графіка: Qt підтримує розширену графіку через API такі як OpenGL, OpenVG та SVG, що дозволяє створювати динамічні 2D та 3D візуалізації. Це робить Qt ідеальним вибором для проектів, що вимагають складної візуалізації і високої продуктивності графіки.

Є кілька переконливих причин обирати Qt для розробки GUI. Серед них варто відзначити підтримку різних мов програмування, таких як C++, а також власну мову інтерфейсу користувача (QML).

Нижче перераховано деякі з найпоширеніших переваг використання Qt:

– кросплатформність: Qt дозволяє створювати програмне забезпечення, що працює на різних настільних та мобільних платформах, таких як Linux, Windows, iOS та Android. Це зменшує зусилля та витрати, які витрачаються на розробку окремих версій для кожної платформи, і робить ваш продукт більш доступним для різних аудиторій;

– відкритий код та комерційні ліцензії: Qt пропонує як комерційні, так і ліцензії з відкритим вихідним кодом, що дозволяє вам вибрати найбільш підходящий варіант для вашого проекту. Відкритий код сприяє спільному розвитку та співпраці з іншими розробниками;

– мова програмування C++: Qt базується на мові програмування C++, що дозволяє створювати різноманітні програмні рішення, від GUI до складних вбудованих систем. Також існують обгортки для інших мов, які дозволяють використовувати Qt з Python, Java, Ruby, PHP, Go і багатьма іншими;

– довгий термін використання та перевіреність: Qt має майже 30-річну історію розвитку і вдосконалення, завдяки чому він став надійним інструментом для багатьох розробників. Завдяки широкому спектру можливостей, Qt готовий вирішити практично будь-які завдання;

– Різноманітні інструменти: Qt надає багато інструментів для створення інтуїтивно зрозумілих і адаптивних дизайнів користувацького інтерфейсу.

Наприклад, QML, QtGui, Qt Widgets і QtUiTools дозволяють розробникам створювати дизайни, які працюють ефективно на різних операційних системах.

Окрім численних переваг Qt, існують певні моменти, які варто розглянути перед використанням цього фреймворку для вашого проекту:

– складність навчання: Освоєння Qt, який базується на C++, може бути часомістким. Часто більш ефективно наймати розробників, які вже мають досвід роботи з цим фреймворком, аніж навчати існуючих інженерів. C++ має складну структуру, що може ускладнити швидке впровадження технології в команду, що не має попереднього досвіду з Qt;

– залежність від старого коду: Відсутність глибоких знань з C++ у широких колах розробників може ускладнити підтримку та оновлення кодової бази Qt. Це може призвести до ситуації, коли бізнес вирішує не оновлювати свою кодову базу, що в кінцевому підсумку призводить до накопичення застарілого коду та проблем з підтримкою;

– потенційне зниження продуктивності: Складність вибору відповідних шаблонів може призвести до зниження продуктивності програми. Крім того, через свою кросплатформність, Qt може працювати повільніше, ніж рідні рішення для певних платформ;

– виклики з управлінням пам'яттю: У C++, який використовується у Qt, потрібно ручне управління пам'яттю, що може призвести до помилок та збоїв у системі. Це може негативно вплинути на продуктивність програми. Однак розробникам доступний ряд інструментів для ефективного управління пам’яттю та оптимізації ресурсів [2].

## 2.2 Опис JavaFx

JavaFX, як комплексна програмна платформа, є важливим інструментом у арсеналі будь-якого Java-розробника, оскільки вона надає великий набір

графічних та медіа інструментів для створення різноманітних десктопних та веб-додатків. Зі зменшенням популярності інтернет-додатків (RIA) через зростання використання HTML5 та відходу від застарілих технологій, як Flash і Silverlight, JavaFX зайняла місце технології, яка забезпечує гарантію стабільності та портативності через принцип «напиши один раз, запусти скрізь». Це означає, що програми, розроблені за допомогою JavaFX, забезпечують однакову функціональність на всіх основних операційних системах без необхідності змінювати код.

З часом JavaFX еволюціонувала і починаючи з Java 8 стала інтегрованою частиною стандартного Java Runtime Environment (JRE) та Java Development Kit (JDK), що робить її більш доступною для розробників, які більше не потребують окремого завантаження JavaFX як зовнішньої бібліотеки. Це інтеграція значно спростила процес розробки, дозволяючи розробникам зосередитися на створенні більш багатофункціональних та візуально привабливих програм без додаткових зусиль з управління зовнішніми бібліотеками.

Ось деякі важливі функції, які роблять JavaFX відмінним вибором для розробників:

– Java API: Оскільки JavaFX написана на Java, вона легко інтегрується з іншими мовами, що виконуються на JVM, забезпечуючи велику гнучкість та доступність;

– FXML та Scene Builder: FXML – це мова розмітки, заснована на XML, яка використовується для побудови GUI в JavaFX. Розробники можуть вручну писати FXML або використовувати Scene Builder, зручний інструмент з інтерфейсом "перетягування", що спрощує дизайн інтерфейсу;

– сумісність зі Swing: JavaFX дозволяє інтегрувати компоненти Swing безпосередньо у свої додатки за допомогою класу Swing Node, що відкриває можливість плавного оновлення існуючих додатків Swing із новими функціями JavaFX;

– вбудовані елементи керування та CSS: JavaFX оснащений всіма необхідними елементами керування для створення багатофункціональних додатків. Компоненти можна легко стилізувати за допомогою CSS, що дає розробникам можливість легко кастомізувати вигляд своїх додатків;

– багатий набір API: JavaFX включає великий набір API, який не тільки покращує розробку GUI, але й використовує можливості платформи Java, такі як генерики, анотації, багатопоточність та лямбда-вирази;

– графічний конвеєр: Рендеринг у JavaFX підтримується апаратним прискоренням за допомогою Prism, який оптимізує графіку для використання з GPU. У випадку відсутності GPU, Prism автоматично переходить на Java 2D для рендерингу.

JavaFX має переваги, які спрощують розробку багатофункціональних додатків на Java:

– властивості та прив’язка даних: JavaFX забезпечує можливість спостереження за змінами у властивостях і дозволяє реагувати на ці зміни автоматично. Це означає, що ви можете встановлювати залежності між властивостями таким чином, що коли одна властивість змінюється, інша пов’язана властивість автоматично оновлюється;

– CSS для стилізації: Розробники, знайомі з веб-розробкою, оцінять можливості стилізації JavaFX за допомогою CSS. Це дозволяє легко і гнучко керувати візуальним аспектом програми, змінюючи теми або адаптуючи інтерфейс без глибоких змін у коді;

– спецефекти та анімація: JavaFX робить легким додавання візуальних ефектів, таких як тіні, відблиски, розмиття, а також анімації. Різні класи переходів і анімацій дозволяють з легкістю додавати динаміку до інтерфейсу. У порівнянні з Java Swing, де анімації потребують додаткового налаштування, JavaFX пропонує вбудовані можливості для створення плавних і вражаючих анімацій;

– підтримка сенсорних пристроїв: Управління сенсорними жестами в JavaFX так само просте, як обробка кліків миші. Це робить платформу ідеальною для розробки додатків для сенсорних пристроїв, забезпечуючи гладке взаємодія з користувачем через дотик, прокручування, масштабування тощо;

– мультимедіа та веб-технології: JavaFX підтримує відтворення аудіо та відео, інтегрує можливості HTML5, що дозволяє розробникам створювати багатомедійні і багатофункціональні інтернет-додатки з сучасними веб-стандартами.

Хоча JavaFX пропонує багато сучасних функцій для розробки інтерфейсів користувача, є деякі аспекти, які потрібно врахувати перед його використанням. Одним із недоліків є те, що JavaFX має відносно меншу підтримку спільноти порівняно зі Swing, який існує набагато довше і має більш велику базу користувачів та розробників. Це може ускладнити пошук рішень та відповідей на специфічні проблеми або потреби при розробці програм.

Іншою проблемою є тривалий час запуску програм, розроблених на JavaFX, порівняно з тими, що створені за допомогою Swing. Програми на базі JavaFX можуть вимагати більше часу для ініціалізації, особливо в середовищах з обмеженими ресурсами або коли програма використовує складні графічні функції, що може вплинути на загальний досвід користувача. Такі особливості слід урахувати при виборі технології для наступного проекту, особливо якщо проект зажадає швидкого часу відгуку та високої продуктивності в реальному часі. Важливо зважити на потенційні технічні виклики і вибрати технологію, яка найкраще відповідає вимогам і ресурсам вашої команди [3].

## 2.3 Опис .NET Framework (WPF)

WPF, яке стоїть за Windows Presentation Foundation, є розробкою компанії Microsoft. Ця технологія була створена спеціально для розробки

динамічних і візуально привабливих настільних додатків для Windows. Раніше відомий під назвою Avalon під час своєї розробки, WPF став частиною .NET Framework 3.0 і продовжує бути ключовим елементом в екосистемі .NET.

WPF використовує мову розмітки на основі XML, відому як XAML, для структурування і дизайну інтерфейсів користувача. Ця можливість забезпечує гнучкість у розробці різних відображувальних компонентів, дозволяючи створювати як окремі десктопні програми, так і рішення, що інтегруються з веб-сайтами.

Завдяки своїм багатим можливостям, WPF часто вибирають як альтернативу стандартним Windows Forms для створення сучасних інтерфейсів користувача, які потребують більшої інтерактивності та кращих візуальних ефектів. WPF пропонує велику колекцію контрольних елементів, потужні засоби для створення графіки та анімації, а також підтримує двостороннє зв’язування даних, що спрощує синхронізацію елементів інтерфейсу з даними моделі.

Ось декілька ключових переваг, які роблять WPF відмінним вибором для розробників:

– неймовірний інтерфейс користувача: WPF вирізняється завдяки своєму здатному вражати візуальному та інтерактивному інтерфейсу. Це не тільки підвищує зацікавленість розробників, але й забезпечує багатий набір можливостей для створення захоплюючих візуальних ефектів через використання мультимедіа, анімації та векторної графіки;

– мультимедійна інтеграція: WPF дозволяє інтегрувати мультимедійні компоненти безпосередньо в програми, що дає змогу легко комбінувати різні медіа-технології;

– використання XAML: XAML, або Extensible Application Markup Language, є мовою розмітки, яка використовується для визначення інтерфейсу користувача в WPF. Це спрощує розробку, дозволяючи розробникам та

дизайнерам більш тісно співпрацювати, ефективно створюючи і редагуючи UI компоненти;

– сприяння кращим відносинам між розробником і дизайнером: WPF дозволяє дизайнерам легко змінювати візуальні аспекти додатку без необхідності переробки коду розробниками. Це сприяє більш ефективному і гармонійному процесу розробки;

– легке розмежування бізнес-логіки та інтерфейсу користувача: WPF дозволяє розробникам використовувати шаблон проектування MVVM, який ізолює бізнес-логіку від інтерфейсу користувача, спрощуючи тестування та обслуговування;

– анімація для бізнес-додатків: Використання анімацій у бізнес-додатках може значно підвищити їх функціональність та зручність для користувача. WPF робить процес додавання анімації неймовірно простим;

– чудова підтримка спільноти: WPF підтримується активною спільнотою розробників та різними сторонніми ресурсами, що включають інструменти і бібліотеки для розширення можливостей платформи;

– гнучке налаштування і тематизація: WPF відомий своїми обширними можливостями налаштування і тематизації, дозволяючи розробникам адаптувати додатки до конкретних потреб і вимог користувачів.

Незважаючи на багато переваг WPF, існують певні недоліки, які важливо розуміти перед його використанням:

– складність навчання: Оволодіти WPF може бути складно, особливо для новачків у сфері програмування. Розуміння основних концепцій, а також специфіки XAML і архітектури MVVM може зайняти час;

– вимоги до ресурсів: Хоча апаратне прискорення робить графіку плавною і швидкою, воно також може використовувати значну кількість системних ресурсів, таких як процесорна потужність та пам'ять. Це може стати проблемою для старіших систем або тих, що мають обмежені характеристики;

– відсутність кросплатформеності: WPF розроблено виключно для Windows. Якщо вам потрібно створити додатки, які працюють на різних операційних системах, доведеться шукати альтернативи, такі як Electron або Flutter;

– застаріла документація: Оновлення документації WPF не відбувається регулярно. Це може ускладнити пошук інформації про нові функції або вирішення специфічних проблем, а доведеться звертатися до спільноти розробників за допомогою;

– великі розміри додатків: Додатки, створені за допомогою WPF, зазвичай мають великий розмір, що може вплинути на час завантаження та продуктивність на менш потужних машинах. Більше того, вони залежать від .NET Framework, який повинен бути встановлений на комп'ютері користувача [4].

Для додатку, який виконує зчитування даних через COM порт з мікроконтролерів, їх візуалізацію та збереження, вибір Qt як основного фреймворку для розробки був обумовлений декількома ключовими чинниками:

– міжплатформна сумісність: Qt підтримує різні операційні системи, що дозволяє вашому додатку працювати на Windows, Linux, MacOS і навіть на мобільних платформах без додаткових зусиль з адаптації. Це важливо, оскільки можливість запускати програму на різних пристроях розширює її потенційне використання і популярність;

– підтримка роботи з послідовними портами: Qt має вбудовані засоби для роботи з послідовними портами через модуль Qt Serial Port. Це робить розробку зв’язку з мікроконтролерами простішою і ефективнішою, що є критично важливим для вашого додатку;

– візуалізація даних: Qt включає в себе розширені інструменти для візуалізації даних, зокрема бібліотеки для створення графіків і діаграм. Це забезпечує ефективний спосіб представлення отриманих даних користувачам у зрозумілій і візуально привабливій формі;

– багатий набір ресурсів та підтримка спільноти: Qt має велику активну спільноту розробників і широкий спектр навчальних матеріалів, форумів і документації. Це забезпечує доступ до готових рішень та швидке вирішення можливих проблем під час розробки;

– гнучкість і масштабованість: Фреймворк Qt дозволяє легко масштабувати ваш додаток, додавати нові функції або адаптувати його до змінюваних технічних вимог, що є важливим для тривалого життєвого циклу програмного продукту.

# 3 ОПИС ПРОГРАМНО-АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Опис апаратної частини

На рисунку 3.1 та 3.2 приведені структурні схеми розроблених пристроїв для моніторингу навколишнього середовища. Ці пристрої складаються з набору датчиків, з’єднаних з мікроконтролерами, що дозволяє збирати дані датчиків.

Зібрані дані передаються через COM порт на комп’ютер, де додаток і візуалізує інформацію для користувача. Це забезпечує не тільки безпечне зберігання даних, але й можливість їх подальшого аналізу і використання для прийняття рішень на основі зібраної інформації.

Для розробки першої схеми потрібні такі компоненти: мікроконтролер Arduino Uno, світлодіод, фоторезистор та датчик температури та вологості DHT11.

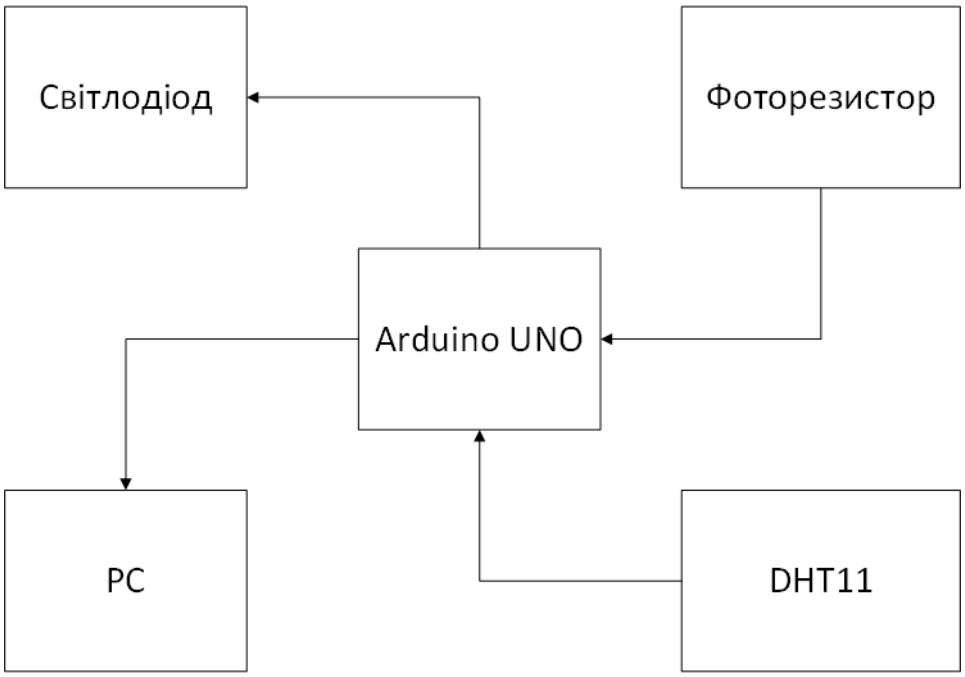


Рисунок 3.1 – Структурна схема для Arduino Uno

Для розробки другої схеми потрібні такі компоненти: мікроконтролер ESP32 та датчик відстані HC-SR04.


Рисунок 3.2 – Структурна схема для ESP32

Розглянемо детальніше блоки схем розроблених пристроїв:

– мікроконтролери Arduino Uno та ESP32 – ядро системи, яке управляє всіма підключеними датчиками та відправляє дані;

– світлодіод – використовується для індикації станів або сигналізації про певні події;

– фоторезистор – датчик, який реагує на зміни освітлення, дозволяє моніторити рівень світла у середовищі;

– датчик температури та вологості DHT11 – вимірює температуру та вологість, забезпечуючи важливі дані про умови навколишнього середовища;

– датчик відстані HC-SR04 – використовується для точного вимірювання відстаней за допомогою ультразвуку, що дозволяє визначати присутність та відстань до об’єктів у середовищі.

Arduino Uno базується на мікроконтролері ATmega328 і містить усе необхідне для зручної роботи з цим пристроєм. Це включає в себе 14 цифрових входів та виходів, шість з яких можуть використовуватися для ШИМ-виходів, шість аналогових входів, 16-мегагерцевий кварцовий резонатор, порт USB для підключення до комп'ютера, порт живлення для зовнішніх джерел та роз’єм

ICSP для програмування мікроконтролера. Є також кнопка для скидання налаштувань. Щоб почати роботу з Arduino Uno, достатньо підключити його до джерела живлення через AC/DC адаптер або батарею, або з'єднати з комп'ютером за допомогою USB кабелю.

Мікроконтролер Arduino UNO зображено на рисунку 3.3.


Рисунок 3.3 – Arduino UNO

Коротка характеристика мікроконтролера зображена у таблиці 3.1 [5].

Таблиця 3.1 – Коротка характеристика Arduino UNO

| | |
|---|---|
| Мікроконтролер | ATmega328 |
| Робоча напруга | 5 V |
| Вхідна напруга (рекомендована) | 7-12 V |
| Вхідна напруга (граничне) | 6-20 V |
| Цифрові входи/виходи | 14 (6 з яких можуть працювати як виходи ШІМ) |
| Аналогові входи | 6 |
| Постійний струм через вхід/вихід | 40 mA |
| Постійний струм для виводу 3.3 В | 50 mA |
| Флеш-пам’ять | 32 KB (з яких 0.5 KB використовуються для завантажувача) |

Продовження таблиці 3.1

| | |
|---|---|
| SRAM | 2 KB |
| EEPROM | 1 KB |
| Тактова частота | 16 MHz |

ESP32 – це високофункціональний мікроконтролер, який став популярним завдяки своїй енергоефективності та вбудованим можливостям Wi-Fi та Bluetooth. Ця серія мікроконтролерів виробляється китайською компанією Espressif Systems і доступна як у двоядерних, так і в одноядерних конфігураціях з використанням мікропроцесорів Tensilica Xtensa LX6 або Xtensa LX7, а також одноядерного мікропроцесора RISC-V.

ESP32 включає в себе набір вбудованих компонентів, таких як антенні перемикачі, підсилювачі потужності, малошумні приймальні підсилювачі, фільтри, а також модулі керування живленням, що робить його ідеальним для застосування в мобільних пристроях і розумних гаджетах. Цей мікроконтролер є наступником попередньої моделі ESP8266 і вже зарекомендував себе як надзвичайно корисний інструмент для широкого спектру проектів в області IoT [6]. ESP32 зображено на рисунку 3.4.

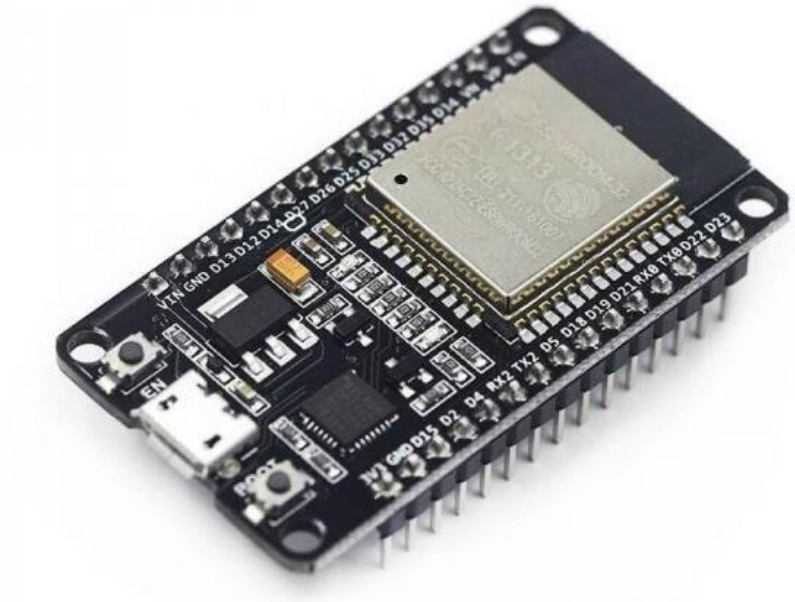

Рисунок 3.4 – ESP32

Короткий виклад технічних характеристик ESP32 зображено у таблиці 3.2 [7].

Таблиця 3.2 – Коротка характеристика ESP32

| | |
|---|---|
| Мікроконтролер | ESP32 |
| Ядра | 2 |
| Архітектура | 32-біт |
| Clock | Tensilica Xtensa LX106 160-240MHz |
| Wi-Fi | IEEE802.11 b/g/n |
| Bluetooth | Yes - classic & BLE |
| RAM | 520KB |
| Flash | Extern QSPI - 16MB |
| GPIO | 34 |
| ADC | 18 |
| Інтерфейси | SPI-I2C-UART-I2S-CAN |

Датчик температури та вологості DHT11 для Arduino призначений для точного вимірювання температури та вологості повітря в навколишньому середовищі. Цей модуль відзначається широким діапазоном вимірювань температури (від 0 до 50 °C), низьким енергоспоживанням і довготривалою стабільністю роботи. DHT11 зображено на рисунку 3.5.

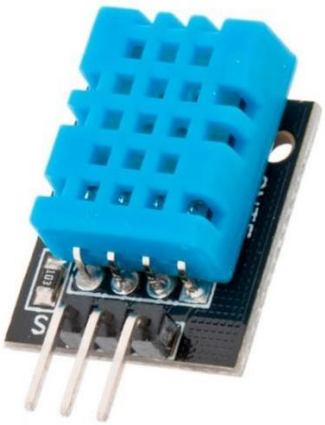

Рисунок 3.5 – DHT11

У складі датчика DHT11 використовується високопродуктивний 8-розрядний мікроконтролер, що забезпечує точність вимірювань та стабільність роботи.

Основний інтерфейс виведення даних з датчика – це один дротовий цифровий вихідний інтерфейс. Це робить його сумісним з різними платами Arduino та іншими мікроконтролерами, і забезпечує зручність у використанні [8].

Характеристики датчика температури та вологості DHT11 зображена у таблиці 3.3.

Таблиця 3.3 – Характеристики датчика температури та вологості DHT11

| | |
|---|---|
| Напруга живлення | 3 V - 5,5 V |
| Струм живлення | 0,5 mA - 2,5 mA |
| Діапазон вимірювання вологості | 20-90 % |
| Діапазон вимірювання температури | 0-50 °C |
| Точність вимірювання температури | ± 2 °C |

Фоторезистор – це чутливий до світла електронний компонент, який широко застосовується для визначення рівня освітленості у різних умовах. Цей компонент складається з двох електродів, розташованих навколо фоточутливого матеріалу, що має здатність змінювати свій електричний опір в залежності від кількості світла, яке на нього потрапляє. Фоторезистор зображено на рисунку 3.6.

Рисунок 3.6 – Фоторезистор

Фоторезистор вирізняється високою чутливістю до світлових змін і зазвичай працює в діапазоні напруг від 3 до 5 вольт. Його можна використовувати в різноманітних застосуваннях, таких як моніторинг рівня освітленості у приміщеннях, регулювання яскравості екранів, автоматизація вуличного освітлення та багато іншого. Завдяки своїй простоті у підключенні фоторезистор легко інтегрувати з мікроконтролерами, як-от Arduino чи Raspberry Pi, що дозволяє розробникам використовувати його у складних проектах, де він може комбінуватись з іншими датчиками та алгоритмами обробки даних для забезпечення точніших та ефективніших вимірювань [9].

Датчик відстані HC-SR04 представляє собою компактну плату з вбудованим випромінювачем та приймачем ультразвукових сигналів, а також електронною схемою для керування ним. Цей датчик має простий інтерфейс, складається з двох виводів живлення, одного входу та одного виходу, що робить його досить зручним у використанні. Він може знайти застосування як датчик присутності у розумних будинках або системах безпеки, а також в робототехніці та автоматизованих пристроях. Крім того, його можна використовувати для створення систем паркування для автомобілів, хоча відмітно, що в умовах зовнішнього середовища швидко може забруднитися [10]. Датчик відстані HC-SR04 зображено на рисунку 3.7.


Рисунок 3.7 – Ультразвуковий датчик відстані HC-SR04

Характеристики ультразвукового датчика відстані HC-SR04 зображена у таблиці 3.4.

Таблиця 3.4 – Характеристики ультразвукового датчика відстані HC-SR04

| | |
|---|---|
| Напруга живлення | 5 V |
| Споживання в режимі тиші | 2 mA |
| Споживання під час роботи | 15 mA |
| Діапазон відстаней | 2-400 cm |
| Ефективний кут спостереження | 15° |
| Робочий кут спостереження | 30° |

## 3.2 Опис програмної частини

Програмна частина для мікроконтролерів розроблювалась в середовище розробки Arduino IDE. Повний код програм приведено в додатку А.

Програмна частина складається з двох основних виконуючих файлів, в яких виконується основна програма та бібліотеки, яка дозволяє відправляти дані з мікроконтролера через COM порт.

Розглянемо детальніше основні функції в кожному файлі.

У програмі для Arduino UNO підключаємо наступні бібліотеки:

- DHT.h: Бібліотека для роботи з датчиком температури та вологості DHT11;
- Logger.h: Власна бібліотека для логування даних датчиків.

Після підключення бібліотек оголошуємо глобальні зміні:

- DHT_PIN: Номер піну, до якого підключено датчик DHT11;
- DHT_TYPE: Тип датчика, який використовується DHT11;
- LED_PIN: Номер піну, до якого підключено світлодіод;
- PHOTORESISTOR_PIN: Аналоговий вхід, до якого підключено фоторезистор;
- steps: Лічильник, який використовується для зміни стану світлодіода.

Функція setup(): Функція виконується один раз при запуску програми. Вона ініціалізує серійний порт для комунікації з ПК, запускає датчик DHT і налаштовує пін світлодіода як вихідний.

Функція loop(): Основна функція програми, яка виконується циклічно. Вона зчитує значення температури та вологості з датчика DHT11, а також значення від фоторезистора. Вона також змінює стан світлодіода (вмикає або вимикає через кожний цикл) та записує отримані дані через бібліотеку Logger.

У програмі для ESP32 підключаємо наступні бібліотеки:

– Ultrasonic.h: Ця бібліотека використовується для взаємодії з ультразвуковим датчиком відстані HC-SR04. Вона надає методи для ініціалізації датчика та зчитування відстані.

Після підключення бібліотек оголошуємо глобальні зміні:

– TRIG_PIN і ECHO_PIN: Цілочисельні змінні, які вказують номери пінів GPIO, до яких підключені тригерний (TRIG) та exo (ECHO) контакти ультразвукового датчика. Вони використовуються для відправлення ультразвукових сигналів та прийому відлуння відповідно;

– sonic: Об’єкт класу Ultrasonic, створений з використанням пінів TRIG і ECHO для взаємодії з ультразвуковим датчиком.

Функція setup(): Виконується один раз при старті програми. Встановлює швидкість передачі даних для серійного порту (115200 бод), яка використовується для відправки даних до комп'ютера чи іншого пристрою.

Функція loop(): Основна функція програми, яка виконується циклічно. Вона вимірює відстань за допомогою методу read() об’єкта sonic, записує ці дані за допомогою методу write() бібліотеки Logger, і повторюється кожну секунду (затримка 1000 мс).

Бібліотека Logger розроблена для спрощення процесу логування даних із різних типів сенсорів у проектах на базі Arduino.

Заголовний файл #pragma once: Цей заголовок забезпечує, що файл бібліотеки включається (або компілюється) лише один раз в проекті. Це

запобігає проблемам з багаторазовим включенням та визначенням одних і тих же класів чи функцій.

Підключення #include "Arduino.h": Цей рядок забезпечує доступ до стандартних функцій та типів даних Arduino, що необхідно для взаємодії з апаратною частиною мікроконтролерів Arduino.

Перелічення SensorType: Цей enum class визначає типи датчиків, з якими може працювати система логування:

- DHT – для датчиків температури і вологості;
- LED – для індикації стану світлодіодів;
- PHOTORESISTOR – для фоторезисторів, які вимірюють інтенсивність світла;
- HC_SR04 – для ультразвукових датчиків відстані.

Клас Logger: Цей клас реалізований як синглтон (одиничний екземпляр), що означає, що він не може бути створений за допомогою конструктора (Logger() = default; в приватній секції класу запобігає створенню екземплярів).

Метод write() цього класу є статичним, що дозволяє йому викликатися без створення об’єкта Logger. Цей метод приймає різну кількість параметрів (...) та може бути адаптований для запису даних різних типів сенсорів. write() може використовуватися для логування даних у зовнішні системи, файлові системи або навіть через серійний порт.

Приклад використання: Метод Logger::write() можна використовувати для запису температури, вологості, стану світлодіодів або виміряних відстаней від датчиків у форматі, який легко читати та аналізувати. Це забезпечує централізоване управління логами і знижує залежність від декількох інструментів для обробки даних із різних джерел.

# 4 ОПИС РОЗРОБЛЕНОГО ДОДАТКУ

## 4.1 Опис завдання до виконання

Розробити програмне забезпечення для зчитування даних через COM порт з мікроконтролерів, їх візуалізації та збереження. Основні функції програми включають:

– зчитування даних через COM порт: Програма повинна мати можливість підключатися до мікроконтролера через COM порт і отримувати потокові дані;

– візуалізація даних в різних вкладках: Отримані дані повинні відображатися у різних вкладках інтерфейсу програми, що дозволяє користувачам легко відстежувати та аналізувати їх;

– збереження даних в Excel: Користувач повинен мати можливість зберегти отримані дані у файлі Excel для подальшого аналізу або обробки;

– гнучкість та адаптивність до кількості датчиків: Програма повинна бути гнучкою і здатною адаптуватися до будь-якої кількості датчиків, що дозволяє користувачам легко розширювати її функціонал у майбутньому;

– можливість додавання нових датчиків: Користувач повинен мати можливість додавати нові датчики безпосередньо в коді програми. Це дозволить йому налагоджувати та вдосконалювати систему, додаючи нові сенсори та пристрої без необхідності зміни загальної структури додатка;

– автоматизація збору та аналізу даних: Програма повинна автоматично зчитувати та відображати дані, а також забезпечувати можливість їх збереження у зручному для подальшого використання форматі;

– зручний інтерфейс користувача: Програма повинна мати інтуїтивно зрозумілий і зручний інтерфейс для користувачів, що дозволяє легко керувати процесом збору та аналізу даних.

## 4.2 Розробка додатку

Програмний код складається з 7 класів: MainWindow, ConnectionChecker, Settings, LEDTab, PhotoresistorTab, TemperatureTab та UltrasonicTab. Вони в свою чергу мають свої поля та методи. UML діаграма класів зображена на рисунку 4.1.

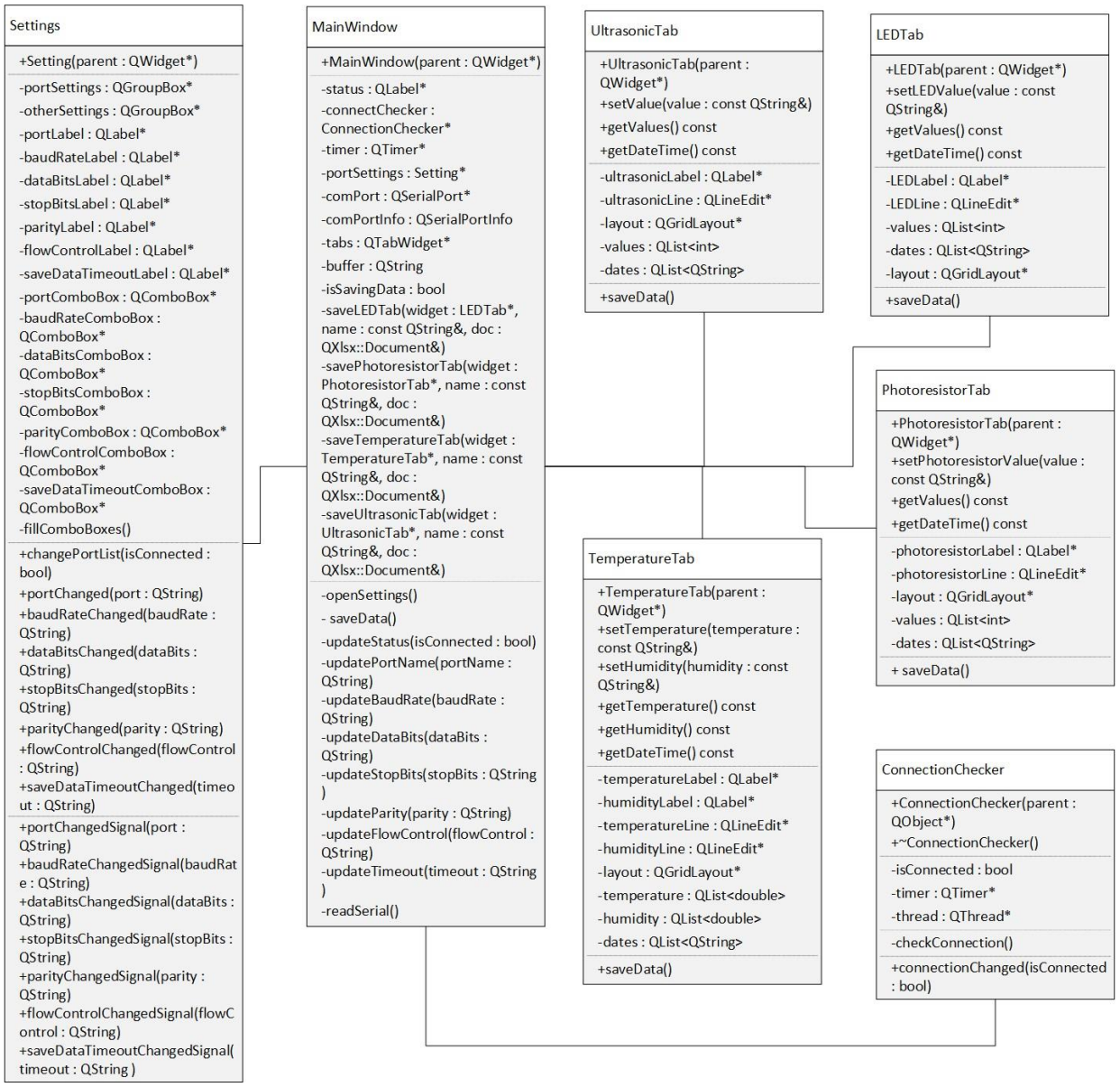


Рисунок 4.1 – UML діаграма класів

Клас MainWindow є головним вікном програми для моніторингу даних, що надходять через COM-порт з підключених пристроїв. Основними

функціями класу є зчитування даних з COM-порту, відображення їх на вікні у вигляді вкладок (для кожного типу датчика окрема вкладка), налаштування параметрів COM-порту, збереження даних у форматі Excel та керування іншими діями користувача через графічний інтерфейс.

Кожне поле та метод класу MainWindow відповідає певному функціоналу:

Поля класу:

– status: Поле для відображення статусу підключення до COM-порту;

– connectChecker: Вказівник на об’єкт класу ConnectionChecker, який перевіряє стан підключення до COM-порту;

– timer: Вказівник на об’єкт класу QTimer, який використовується для періодичного оновлення даних з COM-порту;

– portSettings: Вказівник на об’єкт класу Setting, який відповідає за відображення та налаштування параметрів COM-порту;

– comPort: Вказівник на об’єкт класу QSerialPort, який забезпечує зчитування та запис даних через COM-порт;

– comPortInfo: Об’єкт класу QSerialPortInfo, який містить інформацію про COM-порти на системі;

– tabs: Вказівник на об’єкт класу QTabWidget, який відображає дані з різних датчиків на окремих вкладках;

– buffer: Рядок, який використовується для зберігання даних, що надходять з COM-порту;

– isSavingData: Прапорець, який вказує, чи йде процес збереження даних.

Методи класу:

– MainWindow(QWidget *parent = nullptr): Конструктор класу, який ініціалізує всі поля та встановлює розміри та назву вікна;

– saveLEDTab(LEDTab *widget, const QString& name, QXlsx::Document& doc): Метод для збереження даних з вкладки з LED;

– savePhotoresistorTab(PhotoresistorTab *widget, const QString& name, QXlsx::Document& doc): Метод для збереження даних з вкладки з фоторезистором;

– saveTemperatureTab(TemperatureTab *widget, const QString& name, QXlsx::Document& doc): Метод для збереження даних з вкладки з температурою та вологістю;

– saveUltrasonicTab(UltrasonicTab *widget, const QString &name, QXlsx::Document &doc): Метод для збереження даних з вкладки з ультразвуковим сенсором;

– openSettings(): Слот, який відкриває вікно налаштувань порту;

– saveData(): Слот, який зберігає дані у файл Excel;

– updateStatus(bool isConnected): Слот, який оновлює статус з’єднання з портом;

– updatePortName(QString portName): Слот, який оновлює назву порту;

– updateBaudRate(QString baudRate): Слот, який оновлює швидкість передачі даних;

– updateDataBits(QString dataBits): Слот, який оновлює розмір даних;

– updateStopBits(QString stopBits): Слот, який оновлює розмір стопового біту;

– updateParity(QString parity): Слот, який оновлює тип парності;

– updateFlowControl(QString flowControl): Слот, який оновлює тип контролю потоку;

– updateTimeout(QString timeout): Слот, який оновлює таймаут збереження даних;

– readSerial(): Слот, який зчитує дані з COM-порту та оновлює відповідні вкладки з даними.

Клас Setting є віджетом налаштувань і відповідає за відображення та обробку різноманітних налаштувань для підключення до COM-порту, таких як порт, швидкість передачі даних (бод-швидкість), біти даних, стоп-біти,

парітет, контроль потоку та таймаут збереження даних. Цей клас дозволяє користувачеві вибирати потрібні параметри підключення та зберігати їх.

Поля класу:

– portSettings, otherSettings: Об’єкти класу QGroupBox, що представляють групи налаштувань для портових параметрів та інших налаштувань відповідно;

– portLabel, baudRateLabel, dataBitsLabel, stopBitsLabel, parityLabel, flowControlLabel, saveDataTimeoutLabel: Об’єкти класу QLabel, які представляють підписи для різних налаштувань;

– portComboBox, baudRateComboBox, dataBitsComboBox, stopBitsComboBox, parityComboBox, flowControlComboBox, saveDataTimeoutComboBox: Об’єкти класу QComboBox, які представляють випадаючі списки для вибору значень різних налаштувань.

Методи класу:

– Setting(QWidget *parent = nullptr): Конструктор класу, який ініціалізує всі поля та встановлює відповідні зв’язки між сигналами та слотами;

– fillComboBoxes(): Метод, який заповнює випадаючі списки варіантами для вибору параметрів підключення;

– changePortList(bool isConnected): Слот, який відповідає за зміну списку доступних портів COM в залежності від стану підключення;

– portChanged(QString port), baudRateChanged(QString baudRate), dataBitsChanged(QString dataBits), stopBitsChanged(QString stopBits), parityChanged(QString parity), flowControlChanged(QString flowControl), saveDataTimeoutChanged(QString timeout): Слоти, які реагують на зміну відповідних налаштувань та емітують відповідні сигнали для сповіщення про зміни;

– Сигнали: portChangedSignal(QString port), baudRateChangedSignal(QString baudRate), dataBitsChangedSignal(QString dataBits), stopBitsChangedSignal(QString stopBits), parityChangedSignal(QString

parity), flowControlChangedSignal(QString flowControl), saveDataTimeoutChangedSignal(QString timeout): Сигнали, які емітуються при зміні відповідних налаштувань для сповіщення про зміни.

Клас ConnectionChecker відповідає за перевірку наявності з'єднання з COM-портом. Нижче наведено детальний опис кожного поля та методу класу:

Поля класу:

– isConnected: Булеве значення, що вказує, чи встановлено з'єднання з COM-портом;

– timer: Вказівник на об'єкт класу QTimer, який використовується для періодичної перевірки стану з'єднання;

– thread: Вказівник на об'єкт класу QThread, який використовується для виконання перевірки з'єднання у окремому потоці.

Методи класу:

– ConnectionChecker(QObject *parent = nullptr): Конструктор класу, який ініціалізує поля та починає періодичну перевірку з'єднання;

– ~ConnectionChecker(): Деструктор класу, який завершує роботу потоку, коли об'єкт класу знищується;

– checkConnection(): Приватний слот, який виконує перевірку стану з'єднання з COM-портом;

– connectionChanged(bool isConnected): Сигнал, який відправляється, коли стан з'єднання змінюється.

Клас LEDTab є підкласом класу QWidget і представляє вкладку для відображення та збереження значень світлодіода (LED). Нижче наведено детальний опис кожного поля та методу класу:

Поля класу:

– LEDLabel: Вказівник на об'єкт класу QLabel, який використовується для відображення мітки «LED value»;

– LEDLine: Вказівник на об'єкт класу QLineEdit, який використовується для відображення та редагування значення світлодіода;

– layout: Вказівник на об’єкт класу QGridLayout, який використовується для організації розміщення віджетів на вкладці;

– values: Список цілих чисел, який зберігає значення світлодіода;

– dates: Список рядків типу QString, який зберігає дату та час збереження значення світлодіода.

Методи класу:

– LEDTab(QWidget *parent = nullptr): Конструктор класу, який ініціалізує віджети та налаштовує їх розміщення на вкладці;

– setLEDValue(const QString& value): Метод, який встановлює значення світлодіода для відображення на вкладці;

– getValues() const: Константний метод, який повертає список значень світлодіода;

– getDateTime() const: Константний метод, який повертає список дат та часу збереження значень світлодіода;

– saveData(): Слот, який додає поточне значення світлодіода та поточну дату та час до відповідних списків.

Клас PhotoresistorTab є підкласом класу QWidget і використовується для створення вкладки, призначеної для відображення та збереження значень фоторезистора. Нижче подано детальний опис кожного поля та методу цього класу:

Поля класу:

– photoresistorLabel: Вказівник на об’єкт класу QLabel, призначений для відображення мітки «Photoresistor value»;

– photoresistorLine: Вказівник на об’єкт класу QLineEdit, що використовується для відображення та редагування значення фотодатчика;

– layout: Вказівник на об’єкт класу QGridLayout, який використовується для організації розміщення віджетів на вкладці;

– values: Список цілих чисел, що зберігає значення фотодатчика;

– dates: Список рядків типу QString, який зберігає дату та час збереження значень фотодатчика.

Методи класу:

– PhotoresistorTab(QWidget *parent = nullptr): Конструктор класу, який ініціалізує віджети та налаштовує їх розміщення на вкладці;

– setPhotoresistorValue(const QString& value): Метод, що встановлює значення фотодатчика для відображення на вкладці;

– getValues() const: Константний метод, який повертає список значень фотодатчика;

– getDateTime() const: Константний метод, який повертає список дат та часу збереження значень фотодатчика;

– saveData(): Слот, який додає поточне значення фотодатчика та поточну дату та час до відповідних списків.

Клас TemperatureTab є підкласом класу QWidget і використовується для створення вкладки, яка відображає температуру та вологість. Ось детальний опис кожного поля та методу цього класу:

Поля класу:

– temperatureLabel: Вказівник на об’єкт класу QLabel, який використовується для відображення мітки «Temperature, °C»;

– humidityLabel: Вказівник на об’єкт класу QLabel, який використовується для відображення мітки «Humidity, %»;

– temperatureLine: Вказівник на об’єкт класу QLineEdit, що використовується для відображення температури;

– humidityLine: Вказівник на об’єкт класу QLineEdit, що використовується для відображення вологості;

– layout: Вказівник на об’єкт класу QGridLayout, який використовується для організації розміщення віджетів на вкладці;

– temperature: Список дійсних чисел, що зберігає значення температури;

– humidity: Список дійсних чисел, що зберігає значення вологості;

– dates: Список рядків типу QString, який зберігає дату та час збереження значень температури та вологості.

Методи класу:

– TemperatureTab(QWidget *parent = nullptr): Конструктор класу, який ініціалізує віджети та налаштовує їх розміщення на вкладці;

– setTemperature(const QString& temperature): Метод, що встановлює значення температури для відображення на вкладці;

– setHumidity(const QString& humidity): Метод, що встановлює значення вологості для відображення на вкладці;

– getTemperature() const: Константний метод, який повертає список значень температури;

– getHumidity() const: Константний метод, який повертає список значень вологості;

– getDateTime() const: Константний метод, який повертає список дат та часу збереження значень температури та вологості;

– saveData(): Слот, який додає поточне значення температури та вологості та поточну дату та час до відповідних списків.

Клас UltrasonicTab є підкласом класу QWidget і використовується для створення вкладки, яка відображає відстань, виміряну ультразвуковим датчиком. Ось детальний опис кожного поля та методу цього класу:

Поля класу:

– ultrasonicLabel: Вказівник на об’єкт класу QLabel, який використовується для відображення мітки «Distance, cm»;

– ultrasonicLine: Вказівник на об’єкт класу QLineEdit, що використовується для відображення виміряної відстані;

– layout: Вказівник на об’єкт класу QGridLayout, який використовується для організації розміщення віджетів на вкладці;

– values: Список цілих чисел, що зберігає виміряні значення відстані;

– dates: Список рядків типу QString, який зберігає дату та час вимірювання відстані.

Методи класу:

– UltrasonicTab(QWidget *parent = nullptr): Конструктор класу, який ініціалізує віджети та налаштовує їх розміщення на вкладці;

– setValue(const QString& value): Метод, що встановлює значення виміряної відстані для відображення на вкладці;

– getValues() const: Константний метод, який повертає список збережених значень виміряної відстані;

– getDateTime() const: Константний метод, який повертає список дат та часу вимірювання відстані;

– saveData(): Слот, який додає поточне значення виміряної відстані та поточну дату та час до відповідних списків.

Код програми наведено у додатку А.

Головна сторінка додатку відображає набір вкладок, кожна з яких представляє певний тип даних або інформацію від певного датчика. Ця організація дозволяє користувачам з легкістю перемикатися між різними аспектами моніторингу та аналізу, забезпечуючи зручний та ефективний доступ до потрібної інформації.

Кожна вкладка має свою унікальну назву або ідентифікатор, що відображає зміст даних, які можна очікувати на цій вкладці. Кожна з цих вкладок містить дані, які надходять в реальному часі з відповідних датчиків або джерел.

Кожна вкладка дозволяє користувачам швидко зорієнтуватися в стані та динаміці відповідних параметрів та здійснювати аналіз для вирішення конкретних завдань або проблем.

Такий підхід забезпечує зручність інтерфейсу користувача та ефективне використання часу під час моніторингу та аналізу даних, оскільки він дозволяє легко орієнтуватися у великому обсязі інформації та швидко отримувати

потрібну інформацію для прийняття рішень. Головна сторінка зображена на рисунку 4.2.

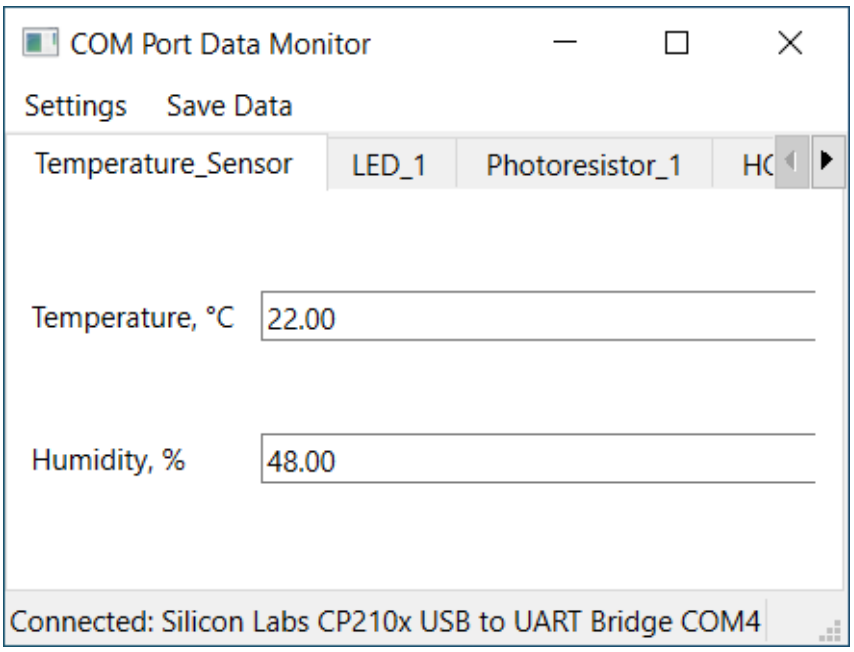


Рисунок 4.2 – Головна сторінка

На сторінці з налаштуваннями користувачі можуть налаштовувати параметри зчитування даних з мікроконтролера. Наприклад, вони можуть вибрати швидкість передачі даних, формат даних та інші параметри зв'язку. Це надає користувачам повний контроль над процесом зчитування та може бути корисним для оптимізації роботи системи залежно від конкретних умов або вимог. Сторінка з налаштуваннями зображена на рисунку 4.3.

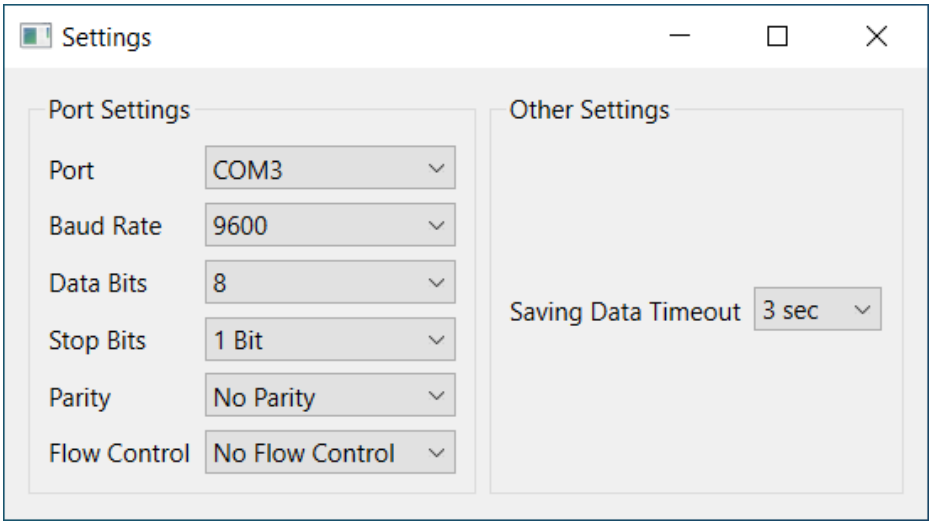


Рисунок 4.3 – Сторінка налаштувань

Однією з ключових функцій нашого додатку є можливість зберігати отримані дані у файл Excel, для подальшого аналізу та обробки даних. Завдяки цій функціональності користувачі можуть з легкістю і ефективністю виконувати глибокий аналіз отриманих даних, використовуючи різноманітні потужні інструменти, які надає програма Excel.

Після збереження даних у файл Excel, користувач може відкрити їх у програмі Excel і використовувати різноманітні функції для аналізу, візуалізації та обробки. Наприклад, вони можуть використовувати функції Excel для створення графіків, діаграм, таблиць та інших типів візуалізацій, що допомагають краще розуміти та аналізувати дані. Крім того, за допомогою вбудованих функцій Excel, можна застосовувати складні формули для отримання більш деталізованих результатів. Зображення однієї з сторінок Excel файлу зображено на рисунку 4.4.

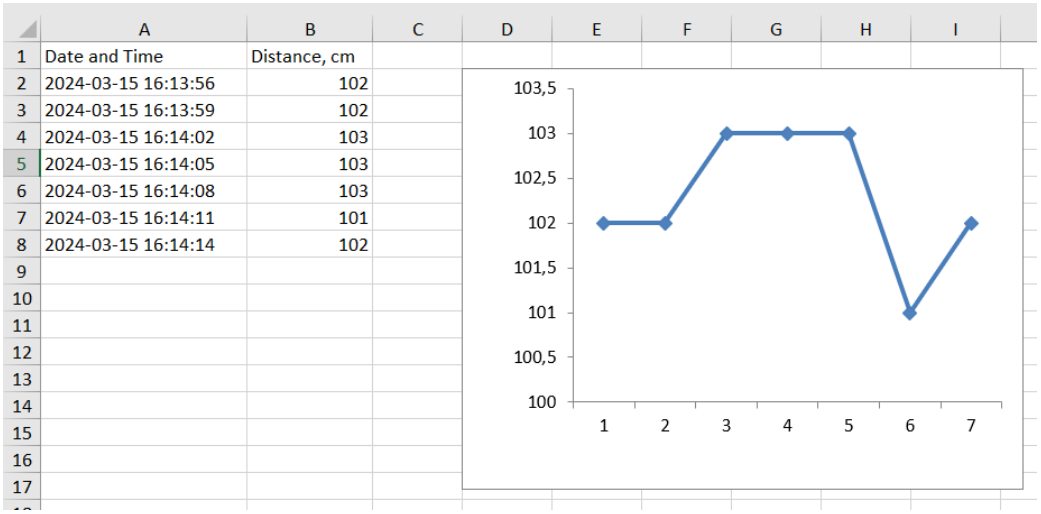


Рисунок 4.4 – Сторінка Excel файлу

Ця можливість забезпечує користувачам необхідні інструменти для глибокого вивчення та розуміння зібраних даних, допомагаючи їм приймати обґрунтовані рішення на основі аналізу отриманої інформації. В результаті, збереження даних у файл Excel стає важливим етапом у роботі з інформацією, який дозволяє максимально використовувати потенціал зібраних даних для досягнення поставлених цілей та завдань.

# ВИСНОВКИ

У рамках цієї кваліфікаційної роботи було успішно розроблено програмне забезпечення для зчитування, візуалізації та зберігання даних з мікроконтролерів через COM порти. Процес розробки охоплював аналіз вимог, вибір найбільш підходящих технологій і інструментів, а також ретельне тестування створеного програмного продукту.

Додаток, який було створено, забезпечує користувачам можливість легкого зчитування даних з мікроконтролерів, їхньої чіткої візуалізації та збереження у файл Excel для подальшого аналізу. Програма виділяється своєю високою інтегрованістю, легкістю управління, масштабованістю та адаптивністю, що робить її незамінною для систем моніторингу та управління.

В розділі про аналіз COM портів була висвітлена унікальність інтеграції широкого спектру функцій управління в одному інтерфейсі, що покращує взаємодію користувачів з системою. Порівняльний аналіз показав переваги використання Qt завдяки його міжплатформності та потужним візуальним можливостям.

Значний акцент у роботі було зроблено на детальну розробку програмно-апаратного забезпечення, зокрема на використання мікроконтролерів для збору даних. Автоматизація процесу дозволила точно і надійно збирати інформацію про умови навколишнього середовища.

Заключний аналіз підтвердив, що розроблена програма відповідає усім вимогам і має великий потенціал для подальшого розвитку та застосування у різних галузях індустрії. Завдяки її адаптивності та можливості масштабування, цей проект є перспективним інструментом для ефективного використання в практичних умовах.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Jan Axelson, Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems. Second Edition. Madison : Lakeview Research LLC, 2007. 380 p.

2. What Is Qt Framework, Why to Use It, and How [Електронний ресурс]. – Режим доступу: https://lembergsolutions.com/blog/why-use-qt-framework#:~:text=Qt%20is%20a%20C%2B%2B%20framework,run%20across%20different%20operating%20systems (дата звернення 30.04.2024).

3. JavaFX 8 – A fantastic tool to create beautiful GUI for your Java application easily [Електронний ресурс]. – Режим доступу: https://etrenddot.wordpress.com/2017/03/19/javafx-8-a-fantastic-tool-to-create-beautiful-gui-for-your-java-application-easily-part-1 (дата звернення 30.04.2024).

4. Advantages and Disadvantages of WPF [Електронний ресурс]. – Режим доступу: https://www.software-developer-india.com/advantages-and disadvantages-of wpf/#:~:text=While%20hardware%20acceleration%20is%20an,be%20detrimenta%20for%20older%20apps. (дата звернення 30.04.2024).

5. Arduino Uno [Електронний ресурс]. – Режим доступу: https://doc.arduino.ua/ru/hardware/Uno (дата звернення 02.05.2024).

6. ESP32 – Вікіпедія [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/ESP32 (дата звернення 02.05.2024).

7. ESP32 for IoT: A Complete Guide [Електронний ресурс]. – Режим доступу: https://www.nabto.com/guide-to-iot-esp-32/ (дата звернення 02.05.2024).

8. Датчик температури та вологості DHT11 для Arduino [Електронний ресурс]. – Режим доступу: https://arduinokit.com.ua/ua/p1127136968-datchik-temperatury-vlazhnosti.html (дата звернення 02.05.2024).

9. Фоторезистор вимірювання рівня освітленості Arduino [Електронний ресурс]. – Режим доступу: https://arduinokit.com.ua/ua/p1592103592-fotorezistor-izmereniya-urovnya.html (дата звернення 02.05.2024).

10. Ультразвуковий датчик відстані HC-SR04 [Електронний ресурс]. – Режим доступу: https://ardushop.in.ua/arduino/ultrasonic-distance-sensor-hc-sr04-distance-meter-sonar (дата звернення 02.05.2024).